Anil Kanduri

# Adaptive Knobs for Resource Efficient Computing

Turku Centre *for* Computer Science

TUCS Dissertations
No 236, December 2018

# Adaptive Knobs for Resource Efficient Computing

Anil Kanduri

*To be presented with the permission of the Faculty of Science and Engineering of the University of Turku for public criticism on December 13, 2018, at 12 noon in auditorium Agora XXI.*

University of Turku
Department of Future Technologies
20014 TURUN YLIOPISTO
FINLAND

2018

## Supervisors

Professor Pasi Liljeberg
Department of Future Technologies
University of Turku, Finland

Adjunct Professor Amir M. Rahmani
Department of Future Technologies
University of Turku, Finland
Marie Curie Global Fellow
University of California Irvine, USA and TU Wien, Austria

Professor Hannu Tenhunen
Department of Future Technologies
University of Turku, Finland

## Reviewers

Dr. Gianluca Palermo
Associate Professor
Department of Electronics, Information and Bio Engineering
Politecnico di Milano
Italy

Dr. Elaheh Bozorgzadeh
Associate Professor
Department of Computer Science
University of California, Irvine
USA

## Opponent

Professor Jari Nurmi
Department of Electronics and Communications Engineering
Tampere University of Technology
Finland

ISBN 978-952-12-3775-1
ISSN 1239-1883

# Abstract

Performance demands of emerging domains such as artificial intelligence, machine learning and vision, Internet-of-things etc., continue to grow. Meeting such requirements on modern multi/many core systems with higher power densities, fixed power and energy budgets, and thermal constraints exacerbates the run-time management challenge. This leaves an open problem on extracting the required performance within the power and energy limits, while also ensuring thermal safety. Existing architectural solutions including asymmetric and heterogeneous cores and custom acceleration improve performance-per-watt in specific design time and static scenarios. However, satisfying applications' performance requirements under dynamic and unknown workload scenarios subject to varying system dynamics of power, temperature and energy requires intelligent run-time management.

Adaptive strategies are necessary for maximizing resource efficiency, considering i) diverse requirements and characteristics of concurrent applications, ii) dynamic workload variation, iii) core-level heterogeneity and iv) power, thermal and energy constraints. This dissertation proposes such adaptive techniques for efficient run-time resource management to maximize performance within fixed budgets under unknown and dynamic workload scenarios. Resource management strategies proposed in this dissertation comprehensively consider application and workload characteristics and variable effect of power actuation on performance for pro-active and appropriate allocation decisions. Specific contributions include i) run-time mapping approach to improve power budgets for higher throughput, ii) thermal aware performance boosting for efficient utilization of power budget and higher performance, iii) approximation as a run-time knob exploiting accuracy-performance trade-offs for maximizing performance under power caps at minimal loss of accuracy and iv) co-ordinated approximation for heterogeneous systems through joint actuation of dynamic approximation and power knobs for performance guarantees with minimal power consumption.

The approaches presented in this dissertation focus on adapting existing mapping techniques, performance boosting strategies, software and dynamic approximations to meet the performance requirements, simultaneously considering system constraints. The proposed strategies are compared against relevant state-of-the-art run-time management frameworks to qualitatively evaluate their efficacy.

# Tiivistelmä

Suorituskykyvaatimukset kasvavat jatkuvasti uusilla tietotekniikan aloilla, kuten tekoäly, koneoppiminen, konenäkö ja esineiden internet. Näiden vaatimusten täyttäminen moderneilla moniydinjärjestelmillä, joissa on korkea tehotiheys, rajalliset teho- ja energiabudjetit ja lämpötilarajoitteita, tekee järjestelmän käytönaikaisen hallinnan haastavaksi. Ongelmaksi muodostuu, miten saada järjestelmistä tarvittava suorituskyky, huomioiden samalla teho- ja energiarajat sekä turvallisuus lämpötilan osalta. Olemassa olevat arkkitehtuuriratkaisut, kuten asymmetriset ja heterogeeniset ytimet ja räätälöity kiihdyttäminen, parantavat suorituskykyä wattia kohden staattisissa skenaarioissa. Sovellusten suorituskykyvaatimusten täyttäminen dynaamisissa ja ennalta tuntemattomissa kuormitustilanteissa, joissa tehon, lämpötilan ja energian järjestelmädynamiikka vaihtelee, vaatii kuitenkin älykästä ajoajan hallintaa.

Adaptiiviset strategiat ovat välttämättömiä resurssitehokkuuden maksimoimiseksi, huomioiden i) samanaikaisten sovellusten vaihtelevat vaatimukset ja ominaisuudet, ii) dynaaminen kuormituksen vaihtelevuus, iii) heterogeenisyys ytimien tasolla ja iv) tehon, lämpötilan ja energian rajoitteet. Tämä väitöskirja esittää nämä seikat huomioivia adaptiivisia tekniikoita järjestelmän käytönaikaiseen resurssihallintaan dynaamisissa kuormitustilanteissa. Väitöskirjassa esiteltävät resurssinhallintastrategiat huomioivat kattavasti sovellusten ja kuormitusten ominaispiirteet sekä tehoaktuaation vaihtelevat vaikutukset suorituskykyyn. Väitöskirjan spesifiset kontribuutiot käsittävät mm. i) ajoajan kartoitusmenettelyn tehobudjettien parantamiseksi korkeammalla suoritusteholla, ii) lämpötilan huomioivan suorituskyvyn tehostamisen korkeamman suorituskyvyn saavuttamiseksi, iii) approksimaatiolaskennan soveltaminen suorituskyvyn optimoimiseksi ja iv) heterogeenisten järjestelmien dynaaminen approksimaatio.

Väitöskirjan esittämät menetelmät keskittyvät olemassa olevien kartoitustekniikoiden, suorituskyvyn parannusstrategioiden, sovellusten ja dynaamisten approksimaatioiden adaptoimiseen niin, että suorituskykyvaatimukset täytetään pysyen samalla järjestelmärajoitteiden puitteissa. Ehdotettuja strategioita verrataan viimeisimpiin ja kehittyneimpiin käytönaikaisiin järjestelmänhallintamenetelmiin niiden tehokkuuden kvalitatiiviseksi arvioimiseksi.

# Acknowledgement

I thank my supervisor Prof. Pasi Liljeberg for shepherding this research by providing an organized environment, stability and freedom. His flexible approach eased my study period and the process of this dissertation. I also thank my co-supervisor and advisor Dr. Amir Rahmani for his guidance throughout the course of this research. His emphasis on identifying relevant research problems and encouragement in pursuing novel solutions shaped my learning process. He helped me in expressing and presenting different ideas with clarity, and forming an overall organized work structure.

Both the supervisors helped to foster collaborations with different research groups, which created valuable working experiences. I extend my regards to all the collaborators and co-authors - Prof. Nikil Dutt, Prof. Axel Jantsch, Prof. Cristiana Bolchini, Prof. Muhammad Shafique and Prof. Hannu Tenhunen - for exchange of ideas, providing their insights and comments in shaping my work.

I am indebted to Prof. Antonio Miele for his guidance, mentorship and the invaluable time he gave to evaluate my work. His hands-on approach and lively presence made the collaboration and the research visits thoroughly enjoyable. Dr. Hashem Haghbayan helped me to identify my strengths and weaknesses, and tried to make me understand many things in and out of research. I express my deepest gratitude to Hashem, for his belief and everything he has done for me.

This research was supported by Academy of Finland projects, University of Turku Graduate School fellowship, University of Turku Foundation, Nokia Foundation and TES Foundations' grants. I thank these organizations for their generosity.

I thank the dissertation reviewers Prof. Gianluca Palermo for the valuable comments and acceptance with honors, and Prof. Elaheh Bozorgzadeh for the feedback and acceptance.

I appreciate all the friends whose compassion kept my sanity. Finally, I thank my parents Pardha Saradhi and Andal, and my sister Swathi for their continued support, belief and encouragement. They have always been there to cheer me up, listen to me and gave the much needed advice.

# List of original publications

The work presented in this dissertation is based on the publications listed below:

| | |
|---|---|
| *Paper I* | **Anil Kanduri**, Hashem Haghbayan, Amir Rahmani, Axel Jantsch, Pasi Liljeberg, Hannu Tenhunen, "Dark Silicon Aware Run-time Mapping for Many-core Systems: A Patterning Approach", in IEEE International Conference on Computer Design (ICCD 2015), USA. |
| *Paper II* | **Anil Kanduri**, Hashem Haghbayan, Amir Rahmani, Muhammad Shafique, Axel Jantsch and Pasi Liljeberg, "adBoost: Thermal Aware Performance Boosting through Dark Silicon Patterning", in IEEE Transactions on Computer, (IEEE-TC 2018) |
| *Paper III* | **Anil Kanduri**, Hashem Haghbayan, Amir Rahmani,Pasi Liljeberg, Axel Jantsch, Nikil Dutt and Hannu Tenhunen, "Approximation Knob: Power Capping Meets Energy Efficiency", in IEEE/ACM International Conference on Computer Aided Design (ICCAD 2016), USA. |
| *Paper IV* | **Anil Kanduri**, Hashem Haghbayan, Amir Rahmani, Pasi Liljeberg, Axel Jantsch, Hannu Tenhunen and Nikil Dutt, "Accuracy-Aware Power Management for Many-Core Systems Running Error-Resilient Applications", in IEEE Transactions on Very Large Scale Integration (VLSI) Systems, (IEEE-TVLSI 2017). |
| *Paper V* | **Anil Kanduri**, Antonio Miele, Amir Rahmani, Pasi Liljeberg, Cristiana Bolchini and Nikil Dutt, "Approximation-Aware Coordinated Power/Performance Management of Heterogeneous Multi-cores", in ACM/IEEE Design Automation Conference (DAC 2018), USA. |

# List of Figures

# List of Tables

# Contents

# Part I

# Research Summary

# Chapter 1

# Overview

Advances in transistor scaling allows the integration of more components on a chip within the same area. With every generation of technology node scaling, compute capacity per area and thus performance increases progressively [Moore, 1965]. Such integration capabilities eases the design of multi- and many-core processors on a single chip. Chip multi-processors (CMP) exploit data and task level parallelism within several applications to offer increased performance gains [Hennessy and Patterson, 2012]. Reduced transistor sizes allows them to be operated at lower voltages. As a result, power consumption is reduced dramatically because of its quadratic dependence on voltage. An important consideration to be made amidst "transistor scaling for performance" is power density i.e., power consumption per unit area. Lower power density reflects in the ease of heat dissipation and lower on-chip temperatures. With transistor scaling, increase in number of components per area is coupled with reduction in power consumption per component. This theoretically ensures a constant power density over different technology nodes, proposed by [Dennard et al., 1974], acknowledged as the *Dennardian Scaling*. In summary, transistor scaling has enabled high performance, low power and energy efficient computer systems design - which in turn advanced several computing applications and domains.

## 1.1 Power Density Rise and Diminishing Gains

Aggressive technology node scaling is pushing transistor gate lengths to their physical limits. As such, operating voltages are also approaching their threshold levels. Such extremely lower operating voltages, closer to the threshold voltage, threatens unreliable operation and increase in leakage power. Consequently, voltage scaling has slowed down in comparison with transistor scaling [Semiconductor Industry Association, 2013]. With every new generation, power densities are no longer constant and are rather increasing [Semiconductor Industry Association, 2013]. Figure 1.1 (a) shows the rate at which voltage and transistor gate lengths have scaled, as

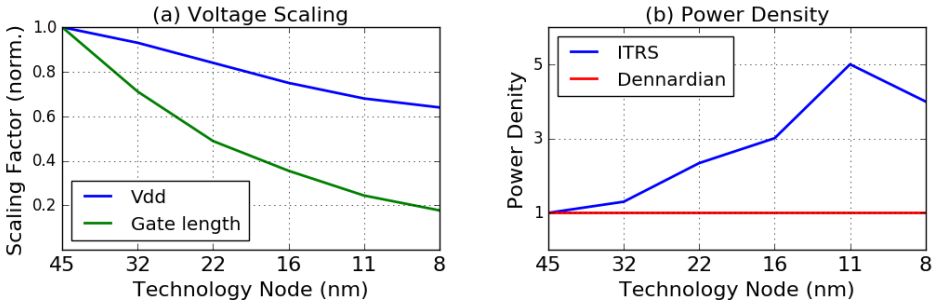



Figure 1.1: Consequences of technology scaling [Semiconductor Industry Association, 2013].
(a). Slack voltage scaling and (b). Rise in power density

per ITRS projections [Semiconductor Industry Association, 2013]. Starting from 45nm technology, the gap between voltage and transistor scaling became prominent. The obvious increase in power density as a result of slack voltage scaling is shown in Figure 1.1 (b). There is a sharp and significant rise in the power density, as opposed to Dennard's prediction of a constant power density. Increase in power densities accumulate on-chip temperatures faster and create hotspots frequently. This poses the threat of thermal violation [Intel Corporation, 2011], accelerates aging [Haghbayan et al., 2017c] and reduces the reliability of the chip [Haghbayan et al., 2017b]. Given the limited cooling solutions, trivial strategy to dissipate heat is to lower the power consumption - by scaling down resources to reduce activity within the chip. Consequently, only a fraction of on-chip logic can be simultaneously powered up, while the rest remains inactive. The inactive section of the chip is widely acknowledged as *dark silicon* [Esmaeilzadeh et al., 2012a]. The amount of active and dark silicon within a chip for different technology nodes is presented in Figure 2.3. This shows the extent of reduction in activity, and thus performance gains, with transistor scaling. As such, computer systems have hit the *utilization wall*, from which point the power, performance and energy gains are diminishing with further aggressive technology node scaling [Zhang et al., 2013].

## 1.2 Open Research Problem

With emerging application domains such as artificial intelligence (AI), machine learning (ML), computer graphics and vision (CG, CV), internet-of-things (IoT), cyber-physical systems (CPS), the demand for performance continues to grow [Arden et al., 2010]. Meeting such performance demands under the constraints of increased power densities, fixed power budgets and energy budgets becomes challenging. Further, mobile and embedded devices specifically deal with another facet of the same problem - since they are battery powered, they have stringent energy budgets. This leaves an open problem on *extracting the required performance*

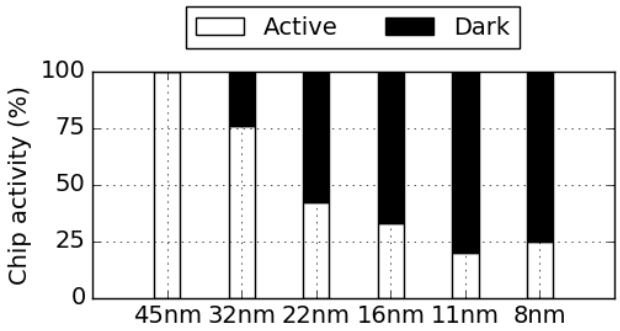



Figure 1.2: Reduced on-chip activity with technology node scaling and dark silicon [Semiconductor Industry Association, 2013].

*within the power and energy limits, while also ensuring thermal safety*.

Architectural solutions including asymmetric and heterogeneous cores [Venkatesh et al., 2010] [Goulding-Hotta et al., 2011] [Muthukaruppan et al., 2013b], and custom acceleration [Esmaeilzadeh et al., 2012c] alleviate the dark silicon challenge and improve performance-per-watt [Taylor, 2012]. Although, custom hardware devices compromise generality, reducing programmers' productivity and require extensive system software support to take advantage of underlying hardware capabilities. Existing performance provisioning techniques address this issue by considering variation among heterogeneous cores during application-to-core mapping, to improve performance and energy efficiency [Van Craeynest et al., 2012] [Sondag and Rajan, 2009] [Tomusk et al., 2016] [Lukefahr et al., 2014]. Apart from architectural solutions, several run-time management techniques optimize power consumption [Haghbayan et al., 2014] [Rahmani et al., 2015a] [Ma and Wang, 2012], on-chip temperatures [Pagani et al., 2017] [Khdr et al., 2017] and energy efficiency [Muthukaruppan et al., 2013b] [Yun et al., 2015]. Most of these strategies focus either on performance or power and thermal optimization, restricting their efficacy to a i) given workload scenario, ii) specific application characteristics and iii) fixed system dynamics.

Despite the efforts in hardware level and run-time systems, satisfying *applications' performance requirements* under dynamic and unknown *workload scenarios* subject to varying *system dynamics* of power, temperature and energy needs intelligent resource management systems. Resource management particularly becomes complex optimization problem with

- unknown and dynamic workload scenarios
- diverse requirements and characteristics of applications
- variable effect of power actuation on performance and core-level heterogeneity

#### Dynamic and Unknown Workloads

Resource allocation for a single application with variable phase behavior and/or concurrent execution of multiple (such) applications requires adaptive decision making. Power consumption and available power budget changes under dynamic workload scenarios i.e., with every new application entering/exiting the system and/or change of a compute phase within an application - effecting per-application performance and per-chip throughput. Also, there is an increased possibility of potential thermal hotspots, with neighboring active cores (on which concurrent applications are running) accumulating temperatures faster. Further, unknown sequence and nature of incoming applications forces reactive power/performance decisions, which could result in over/under compensating resource allocation decisions.

#### Application Characteristics and Requirements

Resource allocation decisions are often conflicting with the orthogonality between power and performance, and diversity among requirements and characteristics of concurrent applications. Specifically, provisioning for high performance would increase power consumption, whereas scaling down resources for low power degrades performance. Similarly, prioritizing an application and allocating system resources for it effects the other concurrent applications, while a completely fair allocation among all the applications might lead to inefficient and under utilization of resources.

#### Variable Power-Performance Characteristics

Heterogeneous architectures expose execution options with diverse power-performance characteristics, which can subjectively provide better performance-per-watt. However, advantage of heterogeneity comes only with appropriate choice of cores that are suitable for each application or threads within an application. This requires identification, expression and translation of applications' requirements, which in turn are to be matched with hardware capabilities to exploit the benefits of heterogeneous hardware. With both application and core diversity, finding the right application-to-core combination and appropriate scaling of resources upon the initial allocation becomes a complex optimization problem.

## 1.3 Research Objectives

A comprehensive resource allocation framework enables prompt servicing of incoming applications while honoring system's power, thermal and energy constraints. Concurrent execution of applications with diverse requirements can vary the overall workload scenarios significantly. This alters the power, performance,

energy and thermal constraints at run-time. Maximizing performance while minimizing power consumption and ensuring thermal safety under dynamic workloads requires robust strategies that adapt to such variations. Primary objective of this thesis is to design and implement techniques that efficiently i) handle varying workloads, ii) dynamically scale resources, iii) provide performance guarantees, iv) honor fixed power budgets and thermal constraints - by considering both applications' and system constraints simultaneously.

### 1.3.1 Principles

The exploration, solutions and insights provided throughout the dissertation are based on the principle of *adaptive resource management*. As the primary entity forming the interface between applications and hardware, resource management layer should adapt to both application and hardware characteristics and requirements. Specific themes of adaptive techniques presented in this thesis are described below.

#### Adapting to Dynamic Workloads

This dissertation considers resource management techniques to be able to handle dynamic workloads i.e., an unknown nature, sequence and time of arrival, by default. This causes significant variation in workload intensity, power, performance and thermal constraints at run-time. This thesis quantifies the effect of such variation in workload intensity and spatial alignment of active cores on power budget utilization and on-chip temperature accumulation. This insight is leveraged to maximize utilizable power budget and minimize the probability of potential hotspots through a pro-active application mapping strategy. The surplus power budget gained and lower on-chip temperatures are used to selectively boost the frequency levels of applications that require higher performance.

#### Adapting to Application Characteristics

This thesis considers the error resilient nature of applications from specific domains and leverages this behavior in the context of resource management. This provides a wider scope for exploring accuracy trade-offs for both performance and energy gains, and power management. This dissertation proposes to use approximation as a dynamic knob to minimize the performance loss incurred in power actuation. The central idea is to switch the mode of execution of a task/application from accurate to approximate, subject to power-performance dynamics of the system and the accuracy-performance dynamics of the application at run-time.

**Adapting to Underlying Hardware**

This thesis identifies the power-performance-accuracy Pareto-space of error resilient applications on heterogeneous hardware. Specifically, the varied power-performance characteristics of heterogeneous hardware platforms, combined with varied error resilience among different applications exposes a wider range of possible power-performance-accuracy states. Such behavior is embedded into analytical models for estimating the impact of using different resource actuation knobs on power-performance-accuracy. These models are used to establish coordination among power and performance management decisions for maximizing resource efficiency i.e., to provide performance guarantees within minimal power consumption and accuracy loss.

### 1.3.2 Contributions

The main research problem addressed in this dissertation is split into multiple sub-problems. Focusing on each such sub-problems, this dissertations makes the following contributions.

- This dissertation proposes a run-time mapping strategy, dark silicon patterning, for improving the amount of utilizable power budgets in many-core systems. It quantifies the effect of application mapping on power budgets and on-chip temperature accumulation of many-core systems. These insights form the basis for the mapping strategy, which interleaves cooler dark cores among hot active cores to reduce temperature accumulation for improved utilizable power budget.

- This dissertation presents thermal-aware performance boosting strategy to maximize performance within utilizable power and thermal headrooms. The proposed approach combines application mapping, power budget allocation and thermal feedback based controller in a pipelined manner to make performance provisioning decisions. This strategy exploits the outcomes of dark silicon patterning viz., lower on-chip temperatures and/or increased power budgets to boost the performance of applications, subject to their requirements and extent of power and thermal headroom available.

- This dissertation presents the idea of using approximation as a dynamic knob for addressing performance loss incurred in power actuation. The proposed approximation knob is triggered through mode switching - a strategy for disciplined invocation of approximation which is to be used in combination with traditional power knobs. Actuation of approximation at run-time considering application characteristics and system dynamics can cover up for the inevitable performance degradation incurred with using traditional power knobs alone. Further, this part of the thesis quantifies applications' sensitivity to error in order to minimize accuracy loss with approximation.

- This dissertation identifies the need for coordinating performance oriented run-time approximation with power actuation decisions. These insights are used to present coordinated approximation for power and performance management of heterogeneous systems, through joint actuation of approximation with traditional power knobs. The coordinated approach based on power/performance prediction models avoids over/under compensation of resource allocation decisions and ensures performance guarantees within minimal power consumption.

## 1.4 Organization

This dissertation is a compendium of articles that are originally published in international conference proceedings and journal series during the course of this research. The manuscript is organized into two major parts viz., Part I - Research Summary and Part II - Original Publications. Part I presents a summarized overview of the research, which is organized into Chapters I-VII. Part II consists of original publications which are included as attachments.

**Part I**

Chapter 2 provides preliminary background on resource management, different power and performance management knobs, existing resource management strategies and highlights the relevant research problems. Significant related work and state-of-the-art approaches are discussed throughout the manuscript wherever necessary and suitable. Chapter 3 - 6 presents specific research problems and proposed solutions, which form the core contributions of this dissertation. These chapters focus on providing the specifics and context of the research problem, necessary motivations, relevant details of the proposed solutions, approaches, strategies, significant results and concluding insights. These chapters are largely presented in an abstract manner, while detailed elaboration is covered through original publication attachments in Part II.

**Part II**

Part II consists of Papers I-V, which are originally published articles in various conference proceedings and journal series. Each paper attached covers aspects presented in Part I across different chapters. Specifically, Paper I corresponds to the contents presented in Chapter 3, Paper II corresponds to the contents presented in Chapter 4, Papers III and IV jointly cover the contents presented in Chapter 5 and Paper V covers the contents presented in Chapter 6.

# Chapter 2

# Run-time Resource Management

Run-time resource management techniques (RTM) are responsible for allocating system resources for applications, according to their requirements. Concurrent execution of applications with diverse requirements can alter workload conditions. Such scenarios further need dynamic scaling of allocated resources for adjusting to the workload variation. Further, RTM techniques have to consider system constraints on power consumption, energy budget, on-chip temperature and core availability etc., while making performance provisioning decisions. This chapter provides relevant background and overview of run-time resource management, existing strategies for power and performance actuation, their efficiency and limitations.

## 2.1 Resource Management Layer

Resource allocation decisions have to satisfy *system objectives* - to function within fixed power, thermal and energy budgets and *application objectives* - to guarantee a certain degree of performance and quality-of-service (QoS) simultaneously. A hierarchical overview of resource management layer is shown in Figure 2.1. As depicted here, typical resource management frameworks form an interfacing layer between applications and the underlying hardware, translating the notion of applications' performance requirements into expressive and measurable parameters. This enables appropriate resource allocation decisions by matching applications' requirements with hardware capabilities.

RTM techniques typically function in an observe-decide-act (ODA) loop in order to meet system and application objectives in terms of power and performance [Rahmani et al., 2015a]. Figure 2.2 shows the ODA loop structure of generic RTM strategy. The *observe* phase monitors instantaneous values of system's constraints (eg., power, temperature and energy budget) and applications' requirements (performance, QoS). The *decide* phase compares the measured values with predefined/expressed requirements to determine resource allocation decisions that satisfy applications' requirements within system constraints. The *act* phase enforces

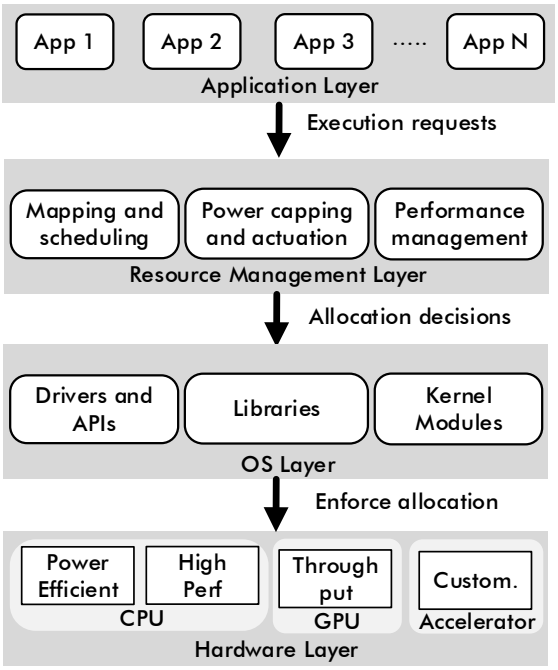



Figure 2.1: Resource management layer

the decisions made by driving appropriate tunable power/performance knobs. The resource manager senses both applications requirements and system constraints in the observe phase, makes resource allocation decisions in the decide phase and finally enforces those decisions in the act phase by tuning power knobs.

## 2.2 Resource Allocation Knobs

State-of-the-art resource allocation policies use techniques such as dynamic voltage and frequency scaling (DVFS), power gating (PG), clock gating (CG), CPU Utilization (Util)/time slice sharing/work load stealing, core folding/degree of parallelism (DoP), application-to-core mapping/thread-to-core binding and task migration (TM) etc. Most of these these techniques provide options for tunable knob settings that can eventually effect resource allocation decisions. Existing techniques using these knobs are summarized in Table 2.1. RTM strategies use one or more combination of these knobs at run-time to actuate power and/or performance. The efficacy of knob actuation on power, performance and energy depends on platform, objective and allocation strategy [Akram et al., 2017] [Majumdar et al., 2017] [Rao et al., 2017] [Akram et al., 2016]. The underlying discrepancy among these knobs is their respective effect on power-performance, which varies subject to dynamic workload characteristics and scenarios. A brief overview of major actuation knobs is presented in the following sub-sections.

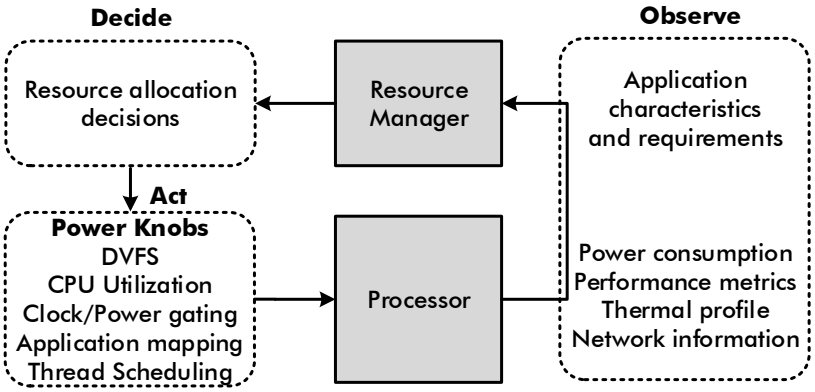



Figure 2.2: Run-time resource management in ODA loop

### CPU Utilization

CPU time slice sharing is a trivial and linearly measurable mechanism for fine grained compute resource allocation and sharing [Ding et al., 2014]. Several run-time management frameworks, middleware and operating system schedulers use the notion of *CPU Utilization* or *time slice sharing* to provide dedicated compute resources among different processes on a time-shared basis [Hindman et al., 2011]. This enables provisioning applications/threads with a an appropriate time slice within a given window such that their performance requirements are satisfied. In a similar vein, setting a fraction of the time slice to *idle* reduces the CPU Utilization below the maximum ($< 100\%$). Active power consumption and performance are reduced - proportional to the length of the *idle* phase in the specified window, while static power consumption remains at large [Sozzo et al., 2016].

### DVFS

Dynamic voltage and frequency scaling is by far the most widely used actuation knob for power/performance management [Kim et al., 2008]. Scaling voltage and frequency levels has a cubic effect on power consumption, following $P = \alpha C V^2 F$, where $P$ is power consumption, $V, F$ are voltage and frequency levels, $C$ is the charge capacitance and $\alpha$ is the activity factor within the chip. Further, reducing supply voltage also reduces static/leakage power in addition to the dynamic power. Although the impact of DVFS on power consumption is obvious, it should be noted that reducing frequency levels also degrades performance [Vega et al., 2013]. Several run-time management techniques have used DVFS to optimize power/performance actuation [Kim et al., 2008] [Cochran et al., 2011] [Haghbayan et al., 2014] [Vega et al., 2013] [Conoci et al., 2018]. DVFS can be actuated with both hardware support through on-chip regulators [Kim et al., 2008], system software support through operating system governors [Muthukaruppan et al., 2013b] and middleware support that guides the operating systems' decisions [Kan-

duri et al., 2018]. Ease of enabling DVFS decisions and subsequent overheads thus varies among different platforms used.

### Power gating

Power gating switches off the supply voltage to functional units using a sleep transistor, and thus reduces both active and leakage power [Hu et al., 2004]. Commercial processors have abstracted power gating to a core-level, enabling mechanism to shut down entire core [Rotem, 2012]. Power gating induces energy and performance overheads in setting core status to idle and then waking up when needed [Lungu et al., 2009]. Despite the overheads, power gating can complement chip-wide DVFS at a fine-grained granularity. Efficiency of power gating can be improved subject to workload characteristics and the window over which sleep-wake phases are invoked. Different strategies to maximize benefits with power gating and its combinatorial optimization along with DVFS have been proposed [Arora et al., 2015] [Madan et al., 2011] [Liu et al., 2014] [Kondo et al., 2014].

### Application Mapping

Servicing an incoming application starts with mapping it onto the chip [de Souza Carvalho et al., 2010]. For a many-core system baseline, mapping requires identifying a suitable location with enough number of free cores that the application can be mapped onto [Fattah et al., 2012a]. Mapping techniques break this into two steps viz., first node selection and core binding [Fattah et al., 2013]. The first node selection corresponds to finding a suitable free core around which the remaining tasks can be mapped. Smart first node selection avoids exhaustive search for free cores repetitively and has an effect on inter-application and intra-application congestion and interference [Haghbayan et al., 2015]. Mapping an application in a suitable region can minimize weighted inter-task communication, avoid interference from other concurrent applications and thus improve performance. Another aspect with efficient mapping strategies for concurrent applications is in avoiding dispersion and fragmentation of free cores, which otherwise could be left under utilized.

### Degree of Parallelism

Applications with inherent parallelism benefit from multi-threaded execution on a set of cores. The number of threads to be spawned for maximum performance gain depends on the applications' nature of parallelism, amount of serial computation, the extent of compute versus memory boundedness, and most importantly - the number of cores that are available to schedule parallel threads [Hennessy and Patterson, 2012]. Considering these aspects, number of threads to be spawned and the number of cores on which these threads will be scheduled largely determines

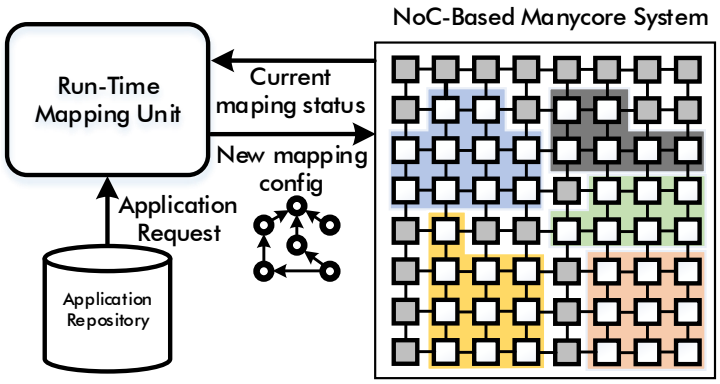



Figure 2.3: Application mapping overview

the performance gains. This also allows resource allocation decisions to provision more cores (for spawned threads) for applications that require higher performance while limiting such an allocation to applications that either do not require a higher performance or do not benefit from the additional compute resources [Cochran et al., 2011]. Further, an optimal power/performance combination can be found by increasing/decreasing the degree of parallelism and tweaking the voltage/frequency levels on top of that [Conoci et al., 2018].

### Core Selection and Task Migration

Asymmetric (same ISA) and heterogeneous (different ISA) chip multi-processors provide diverse compute options with different power-performance characteristics [Big, 2011]. Applications and/or threads within an application exhibit a certain degree of bias towards a specific type of core among the heterogeneous core options [Joao et al., 2013]. Selecting the appropriate core(s) types that are suitable for the applications and scheduling them on the selected cores can improve performance, power consumption and energy efficiency [Muthukaruppan et al., 2013b]. Core selection differs from application mapping in the sense that mapping focuses on availability, spatial alignment and maintenance of regular geometric structure of available cores, but not necessarily on their characteristics. However, mapping and core selection decisions can be combined for joint actuation [Khdr et al., 2017]. Task migration is a well established strategy to migrate the execution of a thread/task from specific type of core to other possible/available cores. The motivation behind migration is to handle variation and distinct phases of workload characteristics and performance requirements within a thread/task/application [Sondag and Rajan, 2009].

Table 2.1: Resource allocation knobs

| Knob | Technique |
|---|---|
| CPU Utilization | [Muthukaruppan et al., 2013b] [Gaspar et al., 2015] [Hindman et al., 2011] [Lo et al., 2015] [Petrucci et al., 2015] |
| DVFS | [Cochran et al., 2011] [Kim et al., 2008] [Vega et al., 2013] [Haghbayan et al., 2014] |
| Power gating | [Ma and Wang, 2012] [Arora et al., 2015] [Lungu et al., 2009] [Kondo et al., 2014] |
| Application Mapping | [Fattah et al., 2013] [Haghbayan et al., 2015] [Fattah et al., 2012b] [Fattah et al., 2014] |
| Degree of parallelism | [Conoci et al., 2018] [Kapadia and Pasricha, 2015] |
| Core Selection and Task migration | [Tomusk et al., 2016] [Yun et al., 2015] [Sondag and Rajan, 2009] [Van Craeynest et al., 2012] [Saez et al., 2012] [Lukefahr et al., 2014] |

## 2.3 Resource Allocation Objectives

Resource allocation decisions have to satisfy application objectives of performance and/or QoS within available system resources while also honoring system objectives of minimizing power consumption and ensuring thermal safety. Both these classes of objectives can be met by actuating power/performance knobs that are presented in the previous sections. A brief account of system and application objectives is presented in the following.

### 2.3.1 System Objectives

From a system's perspective, it is imperative to function within available power and energy budgets, and utilize the resources within thermal safe limits. Also, providing reliable operation [Haghbayan et al., 2017a], prolonging the chip's life time and slowing down ageing process [Haghbayan et al., 2017b] [Haghbayan et al., 2017c] are other crucial objectives. Power and temperature are the major first order system design parameters, since these metrics effect vast majority of aforementioned system objectives. On-chip temperatures accumulate proportional to the power consumption, and initial and ambient temperatures. Thus, actuation knobs and strategies for power and thermal management strategies overlap to an extent. RTM techniques focusing on power actuation have the objective of restricting the power consumption to a fixed upper limit, known as the *thermal design power* (TDP) [Intel Corporation, 2011]. It is analytically determined at design time such that power consumption beyond TDP would potentially translate into a thermal violation i.e., on-chip temperatures accumulate towards/reach the critical temperature.

For a conservative estimation, TDP is determined assuming worst case voltage and frequency levels and workloads. Analytically, capping the power consumption at the TDP ensures thermal safety.

### Power Capping and Regulation

Power capping techniques aim at restricting the power consumption to the fixed upper bound, TDP. Incoming applications in a multi-programmed scenario or phase behavior within a currently running applications causes workload variation - which in turn is reflected in power consumption. Power capping techniques monitor instantaneous power consumption over a time window to check whether power consumption is within the limits. Power violation i.e., power consumption exceeding TDP invokes a capping decision by actuating on available set of power knobs. As described in Section 2.2, these include DVFS, CPU Utilization, power gating, task migration and core folding etc. Precise settings of power knob actuation depends on the extent of power violation. Existing capping techniques have modulated the difference between power consumption and TDP to determine which power knob(s) are best suited for the current scenario and the settings for scaling down those resources. While some of the techniques used linear models [Ma and Wang, 2012] [Cochran et al., 2011] [Vega et al., 2013], others have control theoretic models [Haghbayan et al., 2014] [Muthukaruppan et al., 2013b]. Acting on power knobs would scale down system resources reducing the power consumption below the TDP. However, such an actuation is most likely to degrade the performance to some extent as a consequence.

Over time, system resources might become available with some of the concurrent applications leaving the system after finishing their execution or an application enters a less compute intensive phase. This prompts re-scaling of system resources to utilize the power budget that becomes available with lighter workload. Power capping and the consequent effects on performance of a dynamic workload scenario is shown in Figure 2.4. Two applications blackscholes and bodytrack from PARSEC [Bhadauria et al., 2009] benchmark suite are run concurrently on a 16-core processor, setting TDP to 15W. More details on the simulation platform are described in Section 3.3.1. Initially, `blackscholes` is run on 8 cores and `bodytrack` enters the system at around 5s. This increases the total power consumption to 16.4W, beyond the TDP prompting a power capping decision to be made. Frequency level is reduced from 3.2 GHz to 2.4 GHz to lower the power consumption, as shown in Figure 2.4 (a). As a consequence, scaled down voltage and frequency levels degrade the performance of both the applications as shown in Figure 2.4 (b). At around 11s, blackscholes finishes its execution, freeing up system resources which can be scaled up to improve the other application's performance. Frequency levels are once again set to 3.2 GHz and performance is restored. For optimal power knob settings, power regulation techniques measure instantaneous power consumption, compare it with TDP to determine the power headroom that

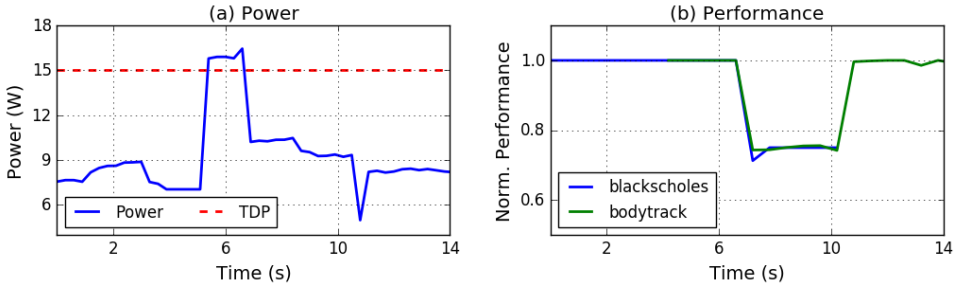



Figure 2.4: Power capping in a dynamic workload scenario. (a) Power consumption (b) Normalized performance. TDP is set to 15W and blackscholes and bodytrack applications are simulated concurrently.

is available. Similar to scaling down decisions of power capping, regulation techniques scale up the resources to i) utilize available power budget effectively and to ii) avoid any performance losses that might have incurred in power actuation [Rahmani et al., 2017] [Rahmani et al., 2015c]. Despite their efficiency in power capping and regulation, these strategies have certain restrictions. Among them, two major limitations are i) reactive decision making - resulting in frequent oscillation among system states and inefficient resource utilization and ii) inevitable performance degradation with power actuation. Given the aforementioned requirements and restrictions, managing system objectives becomes the priority during resource allocation.

### 2.3.2 Application Objectives

From an application's perspective, resource allocation decisions have to either maximize the performance or guarantee an acceptable level of performance, and satisfy the QoS requirements. Application objectives largely depend on the nature of computation, which determines performance and QoS requirements in terms of latency, throughput and quality of result (QoR). For example, streaming applications that iteratively process continuous input data are latency critical, whereas batch processing applications require throughput. Resource allocation decisions in case of latency critical applications target delivering a certain degree of performance in a given time window. With throughput sensitive applications, allocation strategies target maximizing the achievable performance. Execution of concurrent applications that have diverse requirements, for example - a combination of latency and throughput sensitive applications, further complicates resource allocation decisions. On the other hand, while most of the applications require hard guarantees on accuracy and quality of result, applications from emerging domains such as machine learning, computer graphics and vision, internet-of-things (IoT) exhibit tolerance to inaccurate computations. Performance-bound resource allocation decisions are influenced by such application characteristics and requirements. Partic-

ularly, concurrent execution of a heterogeneous combination of applications with diverse requirements can become resource efficient by understanding and leveraging workload characteristics.

## 2.4 Meeting Application and System Objectives

Run-time management has to balance between application and system objectives - to satisfy applications requirements within available system resources while honoring system constraints on power, energy, temperature. The challenging aspect of run-time management however is to improve resource efficiency while meeting both application and system objectives i.e., to maximize performance within minimal system resources and honoring power constraints. This leaves a wider design space for exploring power-performance and accuracy-performance trade-offs to find optimal resource allocation decisions.

### 2.4.1 Power-Performance Trade-offs

As discussed in previous sections, power consumption and performance are always pegged together. The trade-off between power and performance however depends on the architecture, application characteristics (for eg., memory vs compute intensity) and requirements (for eg., latency critical vs best effort), the choice of actuation knobs and optimality of the power/performance decisions. Power-performance trade-offs for widely used power/performance knobs viz., (a) CPU utilization, (b) DVFS, (c) degree of parallelism and (d) task migration are presented in Figure 2.5. This shows normalized performance versus power consumption for a k-means micro kernel, executed on 4 ARM A15 cores and 4 ARM A7 cores (only in case of (d)) . CPU utilization controls the idle and active period of execution, scaling the performance almost linearly. Power consumption on the other hand is sub linear, since static power is still drawn. With DVFS, performance scales proportional to the frequency levels, while power consumption is reduced super-linearly as scaled voltage and frequency levels have cubic impact on power. Performance and power scales non-linearly with scaling the degree of parallelism (assuming one thread per core), with the well known serial sections of code [Cochran et al., 2011] [Conoci et al., 2018]. Similarly, power and performance scale non-linearly with task migration since both A15 and A7 are architecturally optimized for high performance and low power respectively. The power-performance Pareto-space shown in Figure 2.5 is for a single application run in isolation. While the abstract behavior of different knobs might hold good, the extent of power/performance gain/loss varies for each application. This variation is further significant when running several concurrent applications. One of the challenges in making optimal resource allocation decisions is to leverage the insights from power/performance trade-offs - to decide i) which knobs to employ, ii) precise settings of the chosen knob combination, iii)

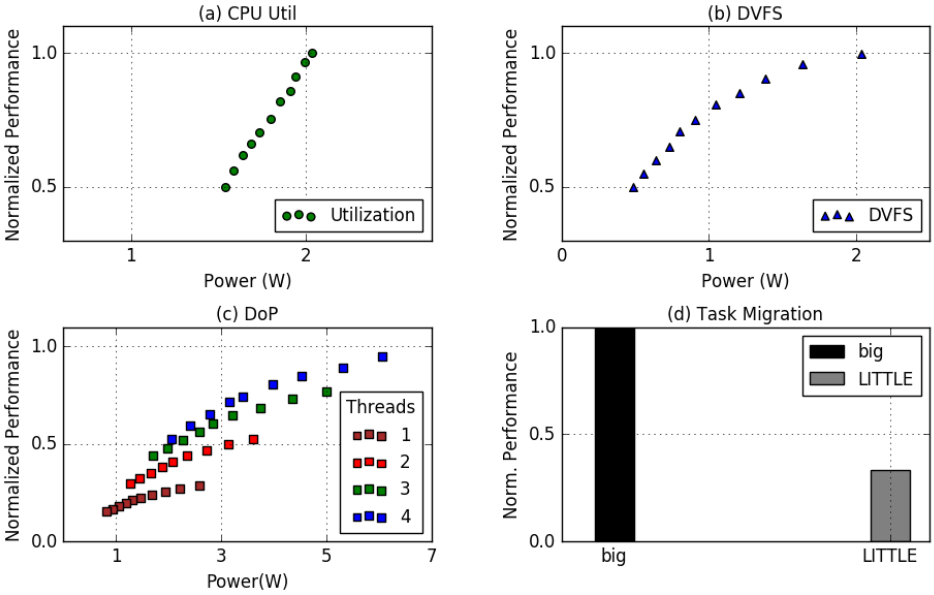



Figure 2.5: Power-performance Pareto-space with different knobs. (a) CPU Utilization, (b) DVFS, (c) Degree of parallelism, (d) Task migration. Experiments were run on quad-core ARM A15 for k-means micro kernel.

subsequent impact of provisioning one application on the other applications, and iv) effects on system constraints of power budget and temperature.

Most of the existing power and thermal management techniques ensure power capping and thermal safety, at the expense of performance degradation [Haghbayan et al., 2014] [Ma and Wang, 2012] [Vega et al., 2013] [Muthukaruppan et al., 2013b]. Some of the techniques do consider the potential impact on performance, and make allocation decisions to minimize such performance losses [Rahmani et al., 2017] [Khdr et al., 2017] [Akram et al., 2017] [Pricopi et al., 2013]. These techniques target fulfilling the objectives of power and performance, making deliberate compromises mutually either on power/thermal budgets or performance. Further, they are largely reactive in nature, resulting in oscillating between system states and phases of over provisioning and under utilization of resources.

### 2.4.2 Accuracy-Performance Trade-offs

Traditional computer architecture prioritizes performance, power consumption and robustness as design parameters, under the constraint that all computations are accurate. Power-performance Pareto space has been thoroughly explored to realize either high performance or ultra low power operation, or an optimal point in the design space. Tweaking the accuracy metric of a robust computer has significant impact on both power and performance, resulting in a new dimension in the Pareto-

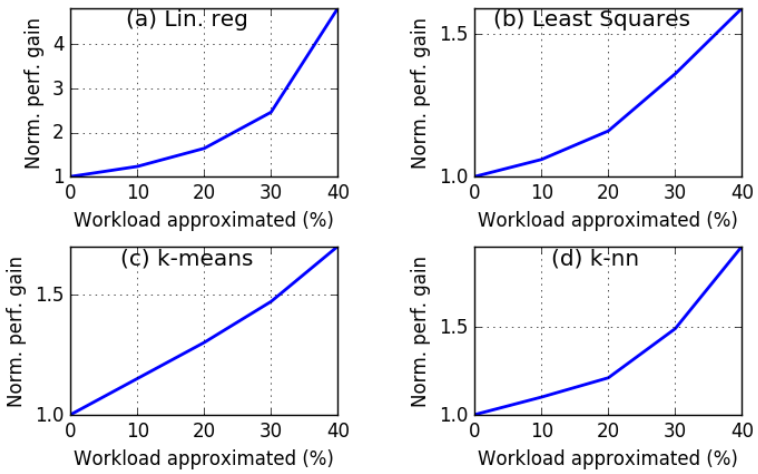



Figure 2.6: Performance gains with approximation. (a) Linear regression, (b) Least squares, (c) k-means, (d) k-nearest neighbors. Workload approximated corresponds to the percentage of loops skipped.

optimal space [Gupta et al., 2013a]. Similar to power-performance trade-offs, certain applications provide opportunities for accuracy-performance trade-offs, which can be exploited for performance and energy gains [Rinard et al., 2004]. *Approximate computing* has emerged as a paradigm for deliberately trading off accuracy for the required power/performance/energy efficiency [Esmaeilzadeh et al., 2012c]. The key idea is to exploit the inherent error resilience exhibited by a range of widely used applications from domains such as multi-media signal processing, machine learning, artificial intelligence, big data analytics etc [Nair, 2015]. Since not all computations have to be accurate, workloads get reduced paving the way for improved performance at lower energy budgets. Figure 2.6 shows the performance gains with reduced workloads for 4 machine learning micro-kernels. Experimental setup in this case is the same as the one mentioned in Figure 2.5. It can be noticed that increasing amount of workload relaxed trivially offers higher performance progressively. In general, the amount of performance and energy gains are subjective to the nature of computations and also depend on the degree of inaccuracy induced. Certain applications are inherently tolerant to inaccurate computation in the sense that an error prone result would still suffice and do not cause a significant difference to the eventual output. Particularly, applications with the following characteristics can tolerate inaccurate computations and error prone results, yet offer an acceptable output. These include, but not limited to applications that -

- deal with real world data which includes noise component
- process massively parallel data (eg. big data, server, data centers)
- are inherently stochastic computational kernels (eg. numerical methods,

Monte-Carlo etc.,)

- have human perception has final end result (eg. video, image, graphics, multimedia)
- involve NP-hard and NP-complete algorithmic problems

There might be a certain loss in quality and deterioration in functionality, however such a loss might not violate the eventual output. For example, embedded control systems' software usually requires high performance over continuous supply of real world sensory data. Since sensory data is analog originating from noisy and uncertain sources, it presents a chance for approximation [Chaudhuri et al., 2011]. From an implementation perspective, accurately describing the system specifications in software and then translating into hardware implementation would hypothetically guarantee an accurate result. However, by relaxing the accuracy of system implementation at both software and hardware layers of abstraction, we could gain an acceptable result, at a relatively low energy consumption with high performance. The notion of accuracy in conventional architectures is exclusively based on numerical correctness. This is a strict constraint since a correct architectural state is expected at every cycle of execution, although restoring the architectural state at the end of the execution is *good enough* [Li and Yeung, 2006]. This style of hard computing requires precision and numerical correctness as opposed to soft computing which exploits imprecision and uncertainty [Zadeh, 1994]. Relaxing the constraint of numerical correctness at an architectural level opens more possibilities for approximation through hardware. The scope of approximation lies in conjunction of applications and architecture [Misailovic et al., 2014]. Inherent error resilience of applications combined with controlled relaxation on numerical correctness of architectures results in high performance and energy efficient approximate computing systems that offer an acceptable quality. Existing techniques to realize approximation can be classified into programming languages [Rinard et al., 2004] [Sampson et al., 2011] [Baek and Chilimbi, 2010] [Bornholt et al., 2014], compilers [Ansel et al., 2009] [Samadi et al., 2013] [Samadi et al., 2014], micro-architectural [Esmaeilzadeh et al., 2012c] [San Miguel and Badr, 2014a] [Venkataramani et al., 2013], custom acceleration [Moreau et al., 2015] [Esmaeilzadeh et al., 2012b] [Du et al., 2015] and hardware [Tong et al., 2000] [Hegde and Shanbhag, 1999] [Gupta et al., 2013b] [Ye et al., 2013]. Most of these techniques perform a static analysis of accuracy-performance trade-offs and exploit these insights during the execution. However, not many techniques have used dynamic approximation i.e., invoke approximation only upon the system/application requirements. Some techniques have used approximation in resource constrained environment [Tan et al., 2015] [Palomino et al., 2016], but they are confined to a specific application and isolated workload scenario. In this dissertation, accuracy trade-offs are exploited opportunistically for meeting applications' performance requirements within available system resources.

## 2.5 Context

Resource management strategies that will be presented in the following chapters are targeted at many-core and multi-core systems with the capabilities of running concurrent applications. The system is assumed to have mechanisms to sense per-core/per-cluster power consumption, on-chip temperature, network characteristics and drive resource allocation decisions such as DVFS, power gating, controlling degree of parallelism, application mapping and power budgeting. Figure 2.7 shows a hierarchical overview of a generic multi-core/many-core system architecture that is targeted throughout this dissertation. Sensing and actuation of power and performance can be typically handled through interfaces or middleware that interacts with any general modern operating systems' utilities. Applications are assumed to be multi-threaded and/or task-graph based, with the ability of expressing performance requirements or a relative priority metric. For error resilient applications used in Chapter 5, one or more alternative thread/task representing the functionally approximate version are assumed to be provided. In Chapter 6, kernels are assumed to be configurable for various levels of accuracy using a parse-able control parameter input. The techniques that are presented in this dissertation can be implemented as such middleware in user-space and/or as operating system modules or a combination of both. The following chapters will present the proposed resource management strategies.

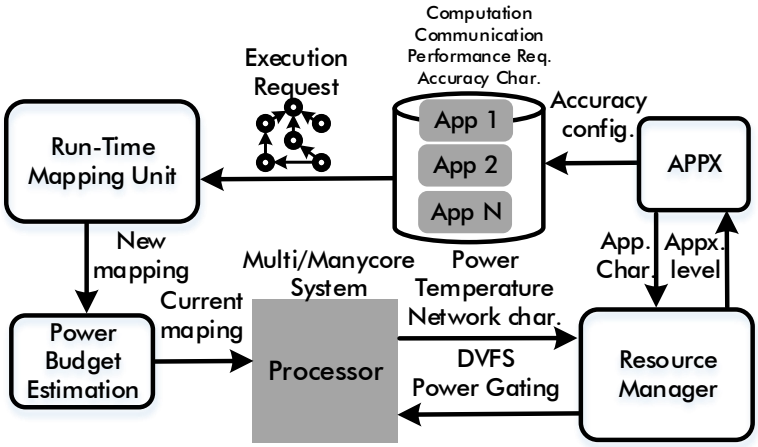



Figure 2.7: Overview of generic system architecture.

# Chapter 3

# Improving Utilizable Power Budget

As mentioned in Chapter 2, computer systems have to function within fixed power budgets to ensure thermal safety. Thermal design power (TDP) is widely used as the fixed upper bound on power consumption during power capping/actuation decisions [Intel Corporation, 2014]. TDP represents the maximum power that can be safely dissipated, hence recommendable to operate within this limit to ensure thermal safety. However, functioning within a fixed conservative power budget can often result in resource under-utilization, restricting the performance gains. Further, alignment of active cores has a significant impact on the amount of utilizable power budget [Shafique et al., 2014b]. Considering these aspects, a run-time estimate on power budget as a function of number of active cores and their spatial alignment (mapping configuration) can enhance the utilizable power budget and thus improve performance that can be extracted. In this Chapter, we provide insights into quantifying the effect of mapping on power budget and thermal accumulation, and present adaptive run-time mapping approach to maximize the utilizable power budget.

## 3.1 Efficient Power Budgeting

TDP is estimated at design time under the assumption that the chip is operating at worst case voltage and frequency levels, and intensive workloads [Intel Corporation, 2011]. Such conservative estimation can lead to resource under utilization, given the variation in workload intensities [Shafique et al., 2014b]. An efficient alternative to fixed TDP is having a variable upper bound on power, estimated at run-time with varying workloads. [Pagani et al., 2014] have proposed Thermal Safe Power (TSP) - a variable power bound calculated as a function of number of active cores and their spatial alignment. The key insights on power budgeting as proposed by [Pagani et al., 2014] and [Pagani et al., 2017] are that i) fixed power budget for variable workloads might cause resource under utilization, ii) upper bound

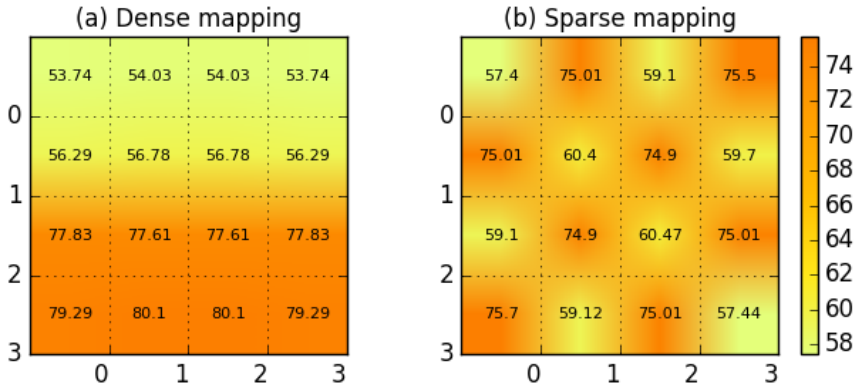



Figure 3.1: Effect of mapping on on-chip temperatures. (a). Dense mapping configuration, application is mapped contiguously on rows 2 and 3, (b). Sparse mapping configuration, application is mapped sparsely on every alternative core. Each tile shows temperature (in °C) of the core.

on power should be a run-time estimate, considering the workload intensities and number of active cores, iii) the upper bound on power should be a function of spatial alignment of active cores, since neighboring (active) cores mutually effect the temperature. These aspects put together bring relevance to application mapping in the context of power budgeting, in order to maximize utilizable power budgets and thus performance.

### 3.1.1 Effect of Mapping

Existing mapping strategies largely prioritize minimizing inter-core communication and maintaining a regular geometric structure to avoid dispersion and fragmentation [Fattah et al., 2014]. Such preferences lead to dense mapping configurations with each application and tasks within an application being tightly packed together. With dense mapping, each active core effects the temperature of its neighboring core, reaching critical temperatures faster. This effect is demonstrated through a simple example as shown in Figure 3.1. Here, we consider a synthetic application with 8 tasks that are mapped - (a) contiguously and (b) sparsely, onto a 16-core system. The power budgets were estimated using *TSP Calculator* [Pagani et al., 2014] - a light weight library for evaluating upper bound on power as function of mapping configuration. Figure 3.1 (a) shows the thermal profile of the chip with the dense mapping, where the active cores reach a critical temperature of 80°C after consuming 47.4 W (5.95 W/core) of overall power budget. Thermal profile of the same application, mapped sparsely is shown in Figure3.1 (b), where active cores are spread out, interleaving idle cores. This mapping configuration reduces the mutual heating effect of active cores and results in lower on-chip temperatures. This allows the active cores to safely utilize relatively higher power budget in com-

parison with the dense mapping. In this case, critical temperature is reached after utilizing 54.4W (6.8W/core) - providing a 14% higher power budget. The surplus budget achieved with sparse mapping can be used to: i) activate more cores, ii) run current tasks much faster, and iii) run more tasks without reaching critical temperatures.

### 3.1.2 Rationale

Although [Pagani et al., 2017] [Shafique et al., 2014a] present the effect of mapping on power budgets, there are no methodical approaches for sparse mapping strategies that achieve higher power budgets. With this motivation, this dissertation proposes a pro-active application mapping approach that prefers sparsity over conventional dense mappings to improve power budgets. The idea is to align hot active cores alongside cool dark cores to minimize the mutual thermal effect. This allows active cores to utilize higher power budget before reaching critical temperatures, and the surplus power budget is used to either i) activate more cores to service more applications or ii) accelerate current applications' execution. The power budget utilization decisions however depend on run-time workload scenarios. We refer to the sparse mapping approach which aligns cool inactive cores among hot active cores as Dark silicon patterning [Kanduri et al., 2015a]. The work flow of the proposed approach is presented in the following sections.

## 3.2 Dark Silicon Patterning

We propose dark silicon patterning in the context of NoC-based many-core systems, where application mapping becomes a relevant factor influencing power budgeting and allocation, temperature accumulation and performance. Our strategy combines i) run-time mapping technique - to increase utilizable power budgets by balancing heat distribution evenly across the chip, and ii) efficient power budget estimation and utilization - to allocate surplus power budget achieved with patterning by activating more cores for throughput, subject to the workload scenarios.

### 3.2.1 System Baseline

Throughout this chapter, a NoC-based many-core system that can concurrently run multi-programmed workloads is considered as the baseline. The workloads are modeled as task graphs where each task holds computation and inter-task communication volume information [TGG, 2017]. Applications are assumed to arrive at an unknown and random sequence, emulating dynamic workload scenarios. Application mapping, in this context, corresponds to binding each task of an incoming application to a core. Assuming a many-core system with $M \times N$ sized mesh, the heat dissipated by each core $C_i$ is a 3-tuple ($P_i$, $T_n$, $T_{amb}$), where $P_i$ is the core's

power consumption, $T_n$ is the weighted accumulated temperature of neighbouring cores and $T_{amb}$ is the ambient temperature. Active cores functioning at full throttle frequency consume relatively higher power and accumulate temperature that is proportional to the power consumption. The other aspect contributing to hot-spots is the neighboring cores' temperature. As more number of neighboring cores are simultaneously active, $T_n$ rises - increasing each active cores' eventual temperature. Dynamic power/thermal management techniques actuate knobs such as power gating and DVFS to reduce the activity, power consumption and thus on-chip temperatures - however these approaches have a performance penalty. Instead, adapting application mapping strategies towards sparse preferences can evenly distribute temperatures, leveraging the inevitable dark cores to provide the necessary cooling effect to active cores. Major distinctions of the proposed sparse mapping from the existing dense mapping strategies are i) spatial distribution of applications and ii) sparsity among tasks of each individual application. To achieve sparsity at both inter-application and intra-application levels, the mapping approach is split into two phases viz., i) selecting a region that is spatially farther from the current set of active cores (that are running concurrent applications) and ii) mapping the tasks of the application sparsely within the selected region such that active and inactive cores are aligned to minimize temperature accumulation.

### 3.2.2 Proposed Mapping Approach

The first phase of selecting a suitable region for mapping an application starts with finding the *first node*, around which an application can be mapped. [Fattah et al., 2013] have proposed first node selection based on the number of free cores that are available within a fixed radius of a given node, without interfering with other concurrent applications. This criteria thus finds regions that relatively have higher/enough number of free cores within which the application can be mapped and avoids intra- and inter- application interference.

#### Inter-Application Sparsity

The proposed approach has two objectives for first node selection viz., i) selecting a region with enough number of free cores to map a given application (retained from the one in [Fattah et al., 2013]) and ii) prioritize nodes (thus regions) that are spatially farther from active cores as *first nodes*. To realize the objective of availability of free cores, we use *Vicinity Counter (VC)* - a parameter that quantifies the number of free cores around a given node which do not interfere with other concurrent applications [Haghbayan et al., 2015]. For the second objective, we define a run-time parameter, distance factor (DF), to represent the extent of fartherness of a given node from the other other active cores. The effect of heat accumulation varies exponentially with distance i.e., being far from active cores indicates lower impact of temperature from neighbors. The *Distance Factor*, $DF_{ij}$, for a node

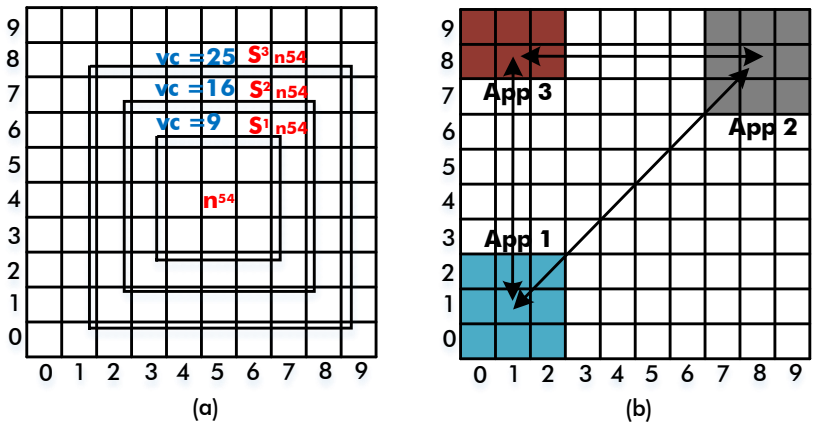



Figure 3.2: Region selection strategy with criteria of: (a) Number of free nodes around a given first node. Here, 3 different square regions (S) and VCs possible for the same first node (n) are shown. (b) Distance from other active cores. Here, applications App1, App2 and App3 being mapped progressively based on distance is shown.

located at (i,j) is modeled as the weighted sum of impact of distance from all the other occupied nodes located at (x,y) such that $(x, y) \in Mesh$. Distance factor is expressed as follows:

$$DF_{i,j} = \sum Wn_{i,j} \times (e^{-\alpha(d_{ij-xy})}) \tag{3.1}$$

where $Wn_{ij}$ is the weight of node $n_{ij}$ (= 1 when occupied and = 0 when free), $d_{ij-xy}$ is distance from nodes located at (i,j) and (x,y) and $\alpha$ is the mesh size. Since temperature accumulation is proportional to the proximity of active cores, a higher DF represents lower effect of temperature accumulation from cores running concurrent applications.

### Pro-active Selection Strategy

A combination of both DF and VC parameters identifies the suitable first node that is i) sparse, ii) has enough number of cores around it to map the applications and iii) has minimal congestion/interference from other concurrently running applications. Existing mapping strategies [Fattah et al., 2012a] [Fattah et al., 2014] have used a reactive search for first node finding, while [Fattah et al., 2013] have used an optimized search using hill climbing algorithm. To improve the turn-around time, our approach uses a pro-active strategy to identify suitable first nodes for incoming applications of unknown characteristics. With every applications' entry/exit, we keep track of a running count of DF and VC of every free node for different radii (within the mesh size). Figure 3.2 shows an example demonstrating the strategy for region selection based on availability of free cores and distance from active

cores. As shown in Figure 3.2 (a), assuming the node located at (5,4) on the mesh as the *first node*, there are an increasing number of free cores with increasing radii available around the *first node*. Precisely, the node (5,4) has a $VC_1 = 9$, $VC_2 = 16$, $VC_3 = 25$ for radii at distance 1, 2 and 3 respectively from the *first node*. Every node on the mesh has a set of VC values for different radii. This allows a pro-active and faster selection of suitable first node for an incoming application based on its size (number of tasks), without requiring an exhaustive across the mesh to find free cores. The pro-active first node selection used in our approach, *MapPro*, is detailed in [Haghbayan et al., 2015]. Figure 3.2 (b) shows 3 applications App1, 2 and 3 that are consecutively mapped, prioritizing distance from currently active cores running other applications. For instance, consider App1 being initially mapped and App2 arrives later. At this instance, App2 is mapped at the farther most region from the currently active set of cores based on the distance factor. A similar approach is taken also with the consecutive App3, which is then chosen to be farther from both App1 and App2. In combination, both VC and DF thus identify a suitable sparse region for mapping.

**Sparse Mapping**

After selecting the first node, all the free cores within the required radius (square root of number of tasks in the application) around the first node are listed into a set $P$. Within in this set, tasks have to mapped in a sparse way to avoid temperature effect from neighboring active cores. Sparsity of a node $n_{ij}$ should indicate the number of free cores that are neighboring it in all four cardinal directions (North, East, West, South). The Sparsity Factor ($SF_{ij}$) for a node $n_{ij}$ located at $(i, j)$ is expressed as:

$$SF_{i,j} = \sum_{i=1}^{4} \sum_{j=1}^{4} F(i + i', j + j') \tag{3.2}$$

where $i'$ = [0, 1, 1, -1], $j'$ = [-1, 0, 1, 1]. $F(i, j)$ denotes if a node located at $(i, j)$ is free or not, such that

$$F(i,j) = \begin{cases} 1 & \text{if } n_{i,j} \text{ is unoccupied} \\ 0 & \text{if } n_{i,j} \text{ is occupied} \end{cases}$$

Each node has a sparsity factor that gets updated upon a neighboring node becoming active/in-active. For an application to be mapped, tasks within the application are first sorted as per their communication volume. The task with the highest communication volume is chosen to be mapped onto the first node. This allows the most communicating task to be accessible within fewer hops to all the remaining tasks. The $SF$ for all the nodes within the set $P$ are updated, after mapping the task. Sorting the tasks and updating the $SF$ is repeated recursively until all the other tasks of the application are mapped. Since the most sparse nodes would be

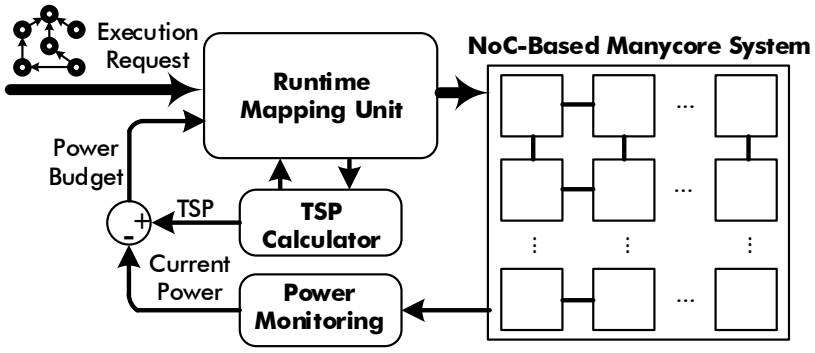



Figure 3.3: Hierarchical view of proposed approach

preferred for mapping, the least sparse nodes within $P$ would eventually be left unoccupied. These cores intuitively represent the patterned dark cores which provide the necessary cooling effect to the other active cores aligned around them.

### 3.2.3 System overview

An abstract view of the system using the proposed approach on mapping and runtime estimation of power budget is shown in Figure 3.3. The execution request corresponds to an incoming application which is modeled as a task graph. Applications are submitted at a random and unknown sequence and are released onto the system for execution. *Power monitoring* unit monitors instantaneous power consumption, while the *TSP Calculator* estimates a safe upper bound on power (TSP) using the current mapping configuration. Based on the measured power consumption and TSP, the amount of power budget available is determined. Each incoming application has a power profile i.e., an estimate of its power consumption when mapped onto the system, extracted through off-line profiling. When a new application arrives, the Run-time Mapping Unit (RMU) compares the power profile of the application and power budget available - to determine whether mapping the new application violates the upper bound on power. When enough power budget is available, RMU maps the application onto the chip using the proposed patterning strategy. In case of un-availability of power budget, the application's execution request is stalled until enough power budget becomes available potentially. This approach is similar to dark silicon aware power management proposed in [Haghbayan et al., 2014]. Once the application is mapped, *TSP Calculator* receives the current mapping configuration of the system as the input and calculates the new upper bound on power budget (TSP). The power monitoring unit uses the updated power budget (TSP) to estimate the available surplus budget. This will determine the number of cores that can be activated further, within the safe thermal limits. Allocation of surplus power budget may be subject to workload scenarios viz., i) activating more cores to map new applications and ii) activating more

cores to accelerate currently running applications. When a new application arrives or an execution request is already waiting, the RMU checks if the available surplus power budget is enough to fit the new/waiting application. In case of power budget being enough to fit the new application, surplus budget is utilized to map the new application. If not, currently running application(s) are accelerated, which would intuitively finish their execution faster and leave the system - creating power headroom for incoming applications. In the context of this work, we consider applications that are best effort in nature, such that the surplus power budget utilized to activate more cores translates into satisfying the performance requirements.

## 3.3 Evaluation

We evaluate the proposed patterning approach against state-of-the-art mapping techniques that prefer contiguity and proximity among concurrent applications and tasks of an applications. The proposed dark silicon patterning approach (referred to as *PAT*) is compared against the combination of *SHiC* [Fattah et al., 2013] and *CoNA* [Fattah et al., 2012b] (from here on referred to as *SC*), for first node selection and mapping respectively. These two are state-of-the-art strategies for mapping which prioritize regions with free nodes and contiguity among applications and also among cores. Since the proposed approach relaxes contiguity and prefers sparsity, this comparison would demonstrate the effect of our dark silicon patterning on power budgets and performance.

### 3.3.1 Experimental Setup

The system architecture presented in Figure 3.3 is implemented by extending experimental many-core simulator, Noxim [Fazzino et al., 2008]. Application and core models that are used for simulation are described in the following sections. The experimental setup detailed in this section is the baseline also for simulations and results which are presented in Chapters 4 and 5.

#### Application Model

For simulation purposes, applications are modeled as task graphs where each task holds a computation factor and communication volume. Application of different sizes ranging from 4 to 35 tasks are used, generated by `Task Graph Generator` [TGG, 2017]. Communication volumes among these tasks follow random Gaussian distribution.

#### Core Model

The traffic patterns of the applications generated as mentioned above are simulated using an in-house cycle-accurate many-core platform implemented in SystemC.

This is largely an extended version of Noxim NoC simulator [Fazzino et al., 2008]. The specifications of Niagara-2 like in-order cores obtained from McPAT [Li et al., 2009] are used as the baseline for processing elements. The communication network infrastructure among processing elements is a pruned version of Noxim. Throughout the simulations, a mesh topology and XY routing were used.

**Power Model**

Technology node scaling parameters are extracted from Lumos framework [Wang and Skadron, 2012] - an open source library which quantifies power-performance characteristics of many-core systems for different technology nodes. Voltage and frequency levels and power values for different technology nodes are provided by Lumos. We used TSP library [Pagani et al., 2014] to calculate the Thermal Safe Power. Thermal simulations (wherever necessary and used) are done using HotSpot [Huang et al., 2006] in its default configuration.

**System Model**

The control strategy including power management and mapping are implemented in C++. This process is pinned to the node $n_{(0,0)}$ of the mesh in the many-core system. Incoming applications arrive at a random sequence and are buffered into a FIFO like structure. Each application's execution request is serviced in a first-come-first-serve basis - subject to availability of enough power budget and free cores. With enough power budget, a suitable first node will be found and the application will be mapped around it.

### 3.3.2 Power Budget and Throughput Gains

The workloads were simulated over network sizes of $16 \times 16$ and $20 \times 20$. TDP is set to 177.7W and 277.7W respectively for $16 \times 16$ and $20 \times 20$ network sizes using 22nm technology [Wang and Skadron, 2012]. A varying amount of dark silicon ranging from 50% darkness through 90% is emulated by adjusting initial upper bound on power consumption (TDP). This is chosen considering ITRS projections which predict 50% dark silicon on contemporary processors and up to 90% dark silicon by year 2020 [Semiconductor Industry Association, 2013]. We limit the number of applications entering the system to be in accordance with dark areas.

Table 3.1: Surplus Power Budget of PAT over $TSP_{wc}$

| Network Size | 50% dark | | 75% dark | | 90% dark | |
|---|---|---|---|---|---|---|
| | Avg. | Best | Avg. | Best | Avg. | Best |
| 16×16 | 11.73 | 13.20 | 22.02 | 24.14 | 32.33 | 34.92 |
| 20×20 | 12.50 | 13.33 | 22.40 | 27.4 | 38.70 | 40.83 |

Run-time power budgets were calculated using the TSP library [Pagani et al., 2014] upon the entry/exit of an application, setting the ambient temperature to 45°C and critical temperature to 80°C. For a fixed number and sequence of incoming applications, power budgets with the proposed approach (PAT) against conservative power budgeting is shown in Table 3.1. Conservative power budget values were calculated using the TSP library [Pagani et al., 2014], which also provides the power budget with worst case workload scenario for given set of applications and active cores. Surplus power budget with PAT ranges between 11% to 40% for different mesh sizes and percentage of darkness. More dark cores allow PAT to exploit them for patterning alongside active cores, making the power budget gains significant. Thus, as the amount of dark silicon increases, power budget gains with the proposed patterning approach increase.

The average (arithmetic mean) and best case power budgets gains (in %) using the proposed *PAT* strategy over *SC* for different network sizes are presented in Table 3.2. *PAT* achieves a surplus power budget when compared to *SC* in all cases - as the active cores are suitably arranged, balancing heat distribution across the chip. In contrast, *SC* tries to map contiguously, leading to tightly packed active cores which get heated up already within a relatively lower power consumption, resulting in a lower power budget utilization. With PAT, the chip always operates under safe peak operating temperature (80°C), since the budgets are computed through TSP library which manages the upper bound on power avoiding hazardous hotspots. It can be observed that the surplus budget achieved in case of *PAT* increases with increase in amount of dark silicon on the chip. With 90% of the chip being dark, utilizing the remaining fewer number of cores that can originally be powered (active) becomes crucial. *PAT* performs better in such scenarios, given the wider choice of dark cores that can be patterned, while *SC* remains dark silicon agnostic. The gain also increases with increase in network size, once again due to increase in scope of the chip area that can be patterned.

Table 3.2: Surplus Power Budget (in %) of PAT over SC

| Network Size | 50% dark | | 75% dark | | 90% dark | |
|---|---|---|---|---|---|---|
| | Avg. | Best | Avg. | Best | Avg. | Best |
| 16×16 | 2.19 | 7.68 | 4.15 | 11.3 | 5.74 | 13.9 |
| 20×20 | 2.63 | 4.28 | 5.06 | 8.55 | 6.54 | 17.17 |

Gain in throughput for *PAT* compared to *SC* for different mesh sizes and dark regions is presented in Table 3.3. Throughput gain depends largely on surplus budget gained, which in turn can be used to activate more cores. Thus, throughput achieved using PAT strategy follows a similar trend to that of surplus power budget achieved. Since a surplus in power budget is gained through *PAT*, it can be utilized to power up more number of cores without violating the safe upper bound on power consumption, which reflects in the overall throughput. The proposed first node se-

Table 3.3: Throughput gain for PAT over SC

| Network Size | 50% dark | | 75% dark | | 90% dark | |
|---|---|---|---|---|---|---|
| | Avg. | Best | Avg. | Best | Avg. | Best |
| 16×16 | 2.42 | 8.58 | 4.59 | 13.92 | 7.27 | 15.64 |
| 20×20 | 2.89 | 4.54 | 5.88 | 10.21 | 8.5 | 20.99 |

lection method chooses different regions for different applications - which has no impact on application's latency. However, intra-application sparsity might incur a per-application latency penalty, due to the compromise on contiguity among tasks of a patterned application. Precisely, communicating tasks of an application that are sparsely mapped increases the number of hops and weighted Manhattan distance, which results in an increased latency. Despite the occasional latency, the overall throughput of the system using PAT still remains higher compared to that of *SC*, as the gain achieved in terms of power budget would (over) compensate for the latency.

**Summary**

In this Chapter, we presented a dark silicon aware run-time mapping strategy for achieving a higher utilizable power budget. The proposed approach is implemented in two phases viz., first node selection and patterning based mapping, where we evenly distribute tasks across the chip area to balance heat distribution. This allows active cores to utilize relatively more power before reaching critical temperatures. As a result, applications utilize surplus power budgets to gain performance, and improve resource utilization and throughput, while ensuring thermal safety of the chip. We evaluated the proposed mapping approach against relevant mappings that follow contiguity to observe the gain in terms of power budget and throughput with sparse mappings. The contents presented in this Chapter are based on the original publication which is attached as Paper I.

# Chapter 4

# Thermal Aware Performance Boosting

Chapter 3 has presented techniques to create surplus power budget through application mapping. Utilizing the created surplus budget has largely focused on activating more cores within the available limits. However, performance boosting techniques such as computational sprinting [Raghavan et al., 2012] and frequency over-boosting [Pagani et al., 2015] can provide more opportunities to improve the efficiency of utilizing the surplus power budget. Further, considering the fact that temperature accumulates with power not instantaneously, but over a delayed epoch - lower on-chip temperatures achieved with dark silicon patterning can be exploited for performance boosting. In this Chapter, we present thermal aware performance boosting which exploits surplus power budgets and lower on-chip temperatures created with dark silicon patterning. We design and implement controllers for run-time power and thermal monitoring, and strategy for efficient utilization of power and thermal headrooms available.

## 4.1 Performance Boosting

Boosting techniques are intended to improve responsiveness of an application by (over-) provisioning resources. Frequency scaling [Rotem, 2012], activating more cores or increasing degree of parallelism [Raghavan et al., 2012], selectively scaling up resources of an application while down scaling other concurrent applications [Pagani et al., 2015] are some of the examples for boosting. Boosting techniques deliberately scale up the resources beyond available power budget but within thermal safe limits, distinguishing them from conventional performance management techniques. Intel's TURBO boost [Rotem, 2012] is an example of boosting through scaling frequency level beyond the set base frequency. Computational sprinting is another example, where more cores are activated for higher performance within thermally safe limits, which otherwise would have been dark

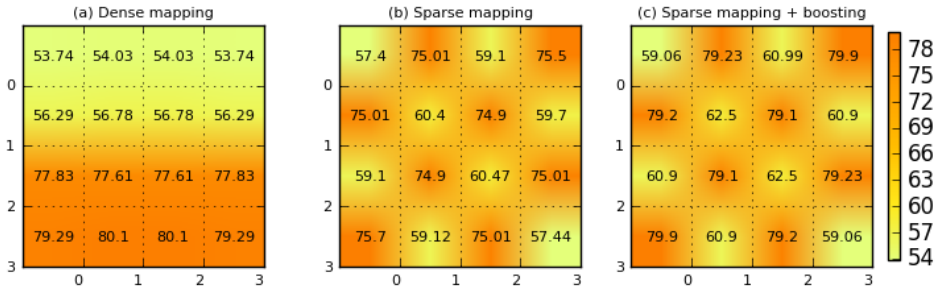



Figure 4.1: Thermal headroom for boosting with sparse mapping. (a) Dense mapping, (b) Sparse mapping without boosting, (c) Sparse mapping with thermal headroom utilized for boosting. Each tile shows the core's temperature (in °C)

to stay within power budget limits [Raghavan et al., 2012]. Scaling up the resources for performance would result in a sharp increase in power consumption. However, the subsequent increase in on-chip temperature happens only over a period of time. Boosting techniques leverage this behavior to throttle resources over shorter intervals that provide a burst of high performance. Nominal operation is resumed once thermal limits are reached, following the boosting decisions. Such opportunistic performance provisioning can address performance surges of specific applications.

### 4.1.1 Creating Thermal Headroom

Boosting techniques exploit the transit time elapsed between a power violation translating into a thermal violation. Performance benefits from boosting can be maximized when operating temperatures are lower and available thermal headroom is higher. As discussed in Section 3.1, sparse mappings operate at relatively lower temperatures and accumulate heat slower in comparison with dense mappings. This gives enough thermal headroom for boosting techniques to i) scale up resources under thermally safe limits and ii) sustain the burst of high performance for a relatively longer period. The effect of sparse mappings in creating thermal headroom is demonstrated through an example application with 8 threads run on a 16-core system. The experimental setup is the same as described in Section 3.3.1. Figure 4.1 shows thermal profile of the system with both dense and sparse mappings. With dense mapping, 8 threads are mapped contiguously on 8 cores and the remaining cores are power gated. After utilizing a power budget of 47.6 W (5.95W per core), this configuration reached the critical temperature of 80 °C. Thermal profile of the same application when mapped sparsely is shown in Figure 4.1 (b). For the same amount of power budget utilization, the sparse mapping configuration reached a peak temperature of 75.7 °C, well below the critical temperature. As described in Chapter 3, interleaving dark cores appropriately reduces the mutual thermal effect among active cores - minimizing on-chip temperatures when compared to dense mappings. The additional thermal headroom of about

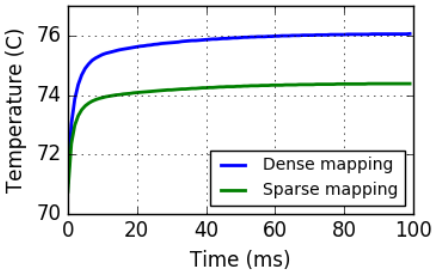



Figure 4.2: Temperature accumulation of dense and sparse mappings

4.3 °C available with sparse mappings can be exploited by boosting techniques to extract higher per-application performance or higher per-chip throughput. Figure 4.1 (c) shows thermal profile of sparse mapping when the thermal headroom created is utilized for higher performance. In this case, each core consumes 6.8 W of power before reaching the critical temperature - yielding a 14.2% improvement in performance.

### 4.1.2 Rationale

Dark silicon patterning from the previous Chapter provides the insight that utilizable power budget and performance of applications can be improved through sparse mapping. On the other hand, sparse mappings also offer relatively lower on-chip temperatures - providing thermal headroom for accelerating certain applications by *boosting* the frequency of active cores. This is demonstrated through an example in Figure 4.2, which shows the rate at which temperature accumulates with dense and sparse mappings for a synthetic application with 8 threads mapped into 8 cores. The experimental setup is the same as described in Section 3.3.1. Temperature profiles are extracted using HotSpot [Huang et al., 2006] in its default configuration. For the same initial conditions, the dense mapping reaches a temperature of 76°C whereas the sparse mapping reaches about 74°C. Noticeable fact is that in addition to lower operating temperatures, the rate at which temperature accumulates is slower with sparse mapping in comparison with dense mappings. This higher thermal headroom and lower rate of temperature accumulation provided with dark silicon patterning can be leveraged to boost the frequency over short bursts of time to improve performance while minimizing the likeliness of thermal violation. Throughout this chapter, this technique is referred to as boosting, distinguishing from DVFS. While conventional DVFS mechanisms are scaled subject to power budget available, boosting is largely dependent on thermal feedback. Figure 4.3 shows the thermal profile of the same example from Figure 4.2, when the applications are boosted from 3 GHz to 3.75 GHz. At t=100ms, application's performance requirements are addressed by scaling the frequency beyond the base frequency of 3 GHz to 3.75 GHz. With dense mapping already operating

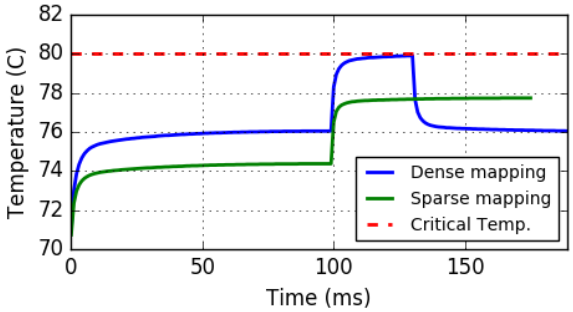



Figure 4.3: Thermal headroom for boosting using sparse mappings

at a relatively higher temperature when boosting is invoked, thermal accumulation towards critical temperature is faster. This lowers the period of boosting that could be sustained, effectively reducing the possible performance gains. The dense mapping reaches the critical temperature of 80 °C at 131 ms, after providing a boosting period of 31 ms. The frequency is scaled down to 3 GHz at this point, to prevent potential thermal violation. The application finished execution eventually after 189 ms. The sparse mapping on the other hand is operating at a relatively lower temperature when boosting is invoked. This allows sustaining a boosting period of 75ms - about 2x longer than that of the denser mapping. There is a significant performance gain during the period of boosting as a result. The application finishes execution by 175 ms, utilizing boosting to its full potential. The sparse mapping configuration could sustain 41% longer period of boosting, which yields about 8% of performance gain in this case, when compared to the dense mapping. Further, the sparsely mapped application remained well below the critical temperature throughout the execution time. This demonstrates the potential benefits with higher thermal headroom and ability to sustain longer periods and/or higher levels of boosting. In summary, mapping applications sparsely to create thermal headroom and boosting applications on top of it maximizes performance. This can also avoid frequent oscillation between frequency up and down scaling decisions that densely mapped applications suffer, with faster temperature accumulation.

## 4.2 Thermal Aware Boosting

Dark silicon patterning presented in Chapter 3 was used to improve the power budget and utilize the surplus power budget to achieve higher throughput. However, patterning can be exploited in two ways viz., i) utilize the surplus power budget to improve overall throughput or ii) utilize the thermal headroom to improve per-application latency. In this chapter, we present performance boosting as per applications' requirements through frequency scaling, which combines both the above strategies. Lower on-chip temperatures achieved from dark silicon patterning are

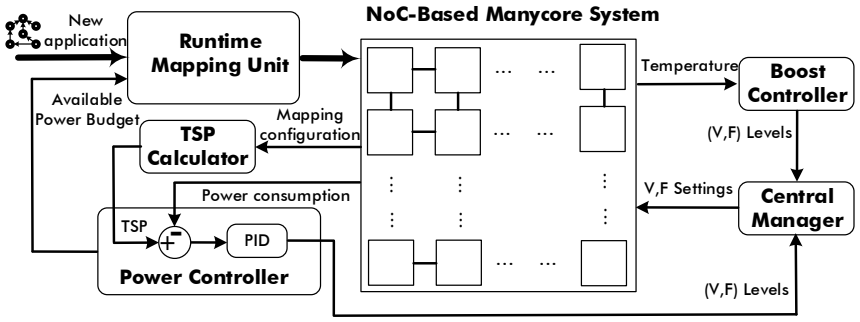



Figure 4.4: Hierarchical view of proposed approach

exploited to make performance boosting decisions, using a thermal feedback. The proposed approach is designed into three phases - i) run-time mapping that maximizes utilizable power budget, ii) power controller that allocates the surplus power budget - for efficient power budget utilization, and iii) boosting controller that increases frequency beyond the base frequency subject to thermal safety - for efficient thermal headroom utilization. The hierarchical view of the proposed system architecture is shown in Figure 4.4. We present the work flow of our approach along with details on individual modules shown in Figure 4.4 in the following sections.

### Run-time Mapping and TSP

The run-time mapping unit (RMU) is invoked upon arrival of a new application and is responsible for mapping incoming applications. The RMU chooses appropriate first node as per application requirements, followed by task-to-core binding. Whenever a new application is mapped or a currently running application leaves the system, overall mapping configuration of the system changes. The TSP Calculator receives the updated mapping configuration every time an application enters/exits the system - to estimate the new safe upper bound on power (thermal safe power, TSP), as a function of alignment of active cores. This remains the power budget until the mapping configuration changes subsequently.

### Power Controller

The power controller makes power actuation and allocation decisions by monitoring instantaneous power consumption of the chip. It receives the safe upper limit on power (TSP) from the TSP Calculator and compares the monitored power with TSP to estimate the available power headroom. A PID controller is used to modulate the available power budget to determine DVFS and power gating settings. The difference between TSP and power consumption is used as the input error metric for the PID controller's output. Gain constants of the PID controller are adjusted after offline MATLAB simulations. PID controller used in our approach is based

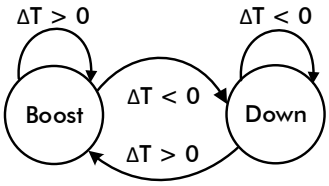



Figure 4.5: State transition of boost controller. $\Delta T$ represents the thermal headroom available.

on the controller presented in [Haghbayan et al., 2014] and [Rahmani et al., 2017]. Choice of DVFS and power gating depends on current workload scenario. For instance, arrival of a new application requires utilizing the available budget to activate more cores, while DVFS scaling decisions are preferred otherwise. In either cases, power knobs are scaled appropriately to fully utilize the available budget. Power controller is invoked over every parametric power epoch $E_{power}$.

**Boost Controller**

The boost controller makes performance boosting decisions as per applications' requirements. In the context of this chapter, we use scaling the frequency beyond the base frequency as the boosting mechanism. This is similar to Intel TURBO [Naveh et al., 2011] and selective boosting techniques [Pagani et al., 2015]. The boost controller monitors applications' performance requirements and per-core temperature over every parameterizable epoch $E_{boost}$. Upon identifying any applications' request for higher performance, the boost controller estimates thermal headroom available based on the on-chip temperature feedback. If enough thermal headroom is available, frequency is scaled by one $level$. Each $level$ represents the step size by which frequency will be scaled. We implemented $level$ as a control parameter that can be adjusted for platform specific constraints. In case of a thermal violation due to previous epoch's boosting decisions, frequency levels are scaled down to reduce on-chip temperatures. To avoid frequent upscaling and downscaling boosting decisions, we chose a conservative step size of 200 MHz and a boosting epoch of length 2.5ms. Within the considered platform and simulation framework, these values of frequency step size and boosting epoch are observed to be the minimum levels that could result in a noticebale increase in temperature. However, both frequency step size and boosting epoch length are subjective to specific platforms and architectures. Hence, they are used as generic parameters in the proposed design. Figure 4.5 shows a simple state chart on boost controller's decisions. $\Delta T$ represents the amount of thermal headroom available, which effects the transition between *Boost* - upscaling and *Down* - downscaling modes. The Central Manager shown in Figure 4.4 arbitrates between the power controller and boost controller decisions, to avoid any conflicts. At every boosting epoch, the application and corresponding frequency scaling information is communicated to the central manager. While the

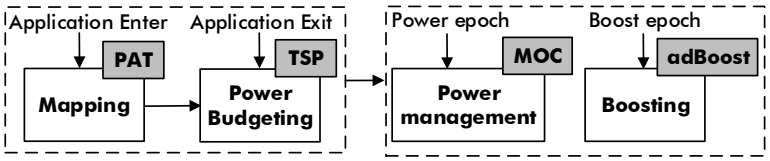



Figure 4.6: Workflow of proposed approach

power controller decisions are based on power budget available, boost controller's decisions are based on thermal headroom available. Boost controller's decisions rely on temperature feedback, irrespective of the power headroom. We set $E_{power}$ as 4 $\times$E$_{boost}$, such that boosting decisions are made conservatively to avoid thermal violation. This also minimizes the oscillation of voltage and frequency scaling decisions among power and boost controllers. Both power controller and boost controller act independently over their respective epochs and communicate their decisions to the central manager. The central manager enforces voltage and frequency levels and power gating settings as per the decisions made by power controller and the boost controller by arbitrating between them.

Figure 4.6 shows modular work flow of of the proposed approach and different control units, putting it all together. When an application enters, the application is mapped using dark silicon patterning [Kanduri et al., 2015a], which was detailed in Chapter 3. Power budget is estimated using TSP library [Pagani et al., 2017]. The power controller is based on multi-objective controller, presented in [Rahmani et al., 2015a]. The adaptive boosting control (adBoost), presented in this section is used as the boosting strategy.

## 4.3 Evaluation

The experimental setup used for evaluating the proposed approach including the application model, system model and power model are already described in Section 3.3.1 from Chapter 3. Our evaluation is two folds i.e., i) to demonstrate the effect of dense and sparse mappings on performance through boosting and ii) to compare the performance gains with conventional DVFS mechanism and adaptive boosting. For the dense and sparse mappings, we used contiguous mapping - CoNA [Fattah et al., 2012b] and dark silicon patterning - PAT [Kanduri et al., 2015a], respectively. As for controller strategy, we choose two baselines of runtime management frameworks viz., i) conventional DVFS and power gating and ii) adaptive boosting along with DVFS and power gating. We used multi-objective controller, MOC, presented in [Rahmani et al., 2015b] [Rahmani et al., 2017] as a conventional DVFS based power manager, to evaluate it against the proposed adBoost controller. We compare the combination of different mapping strategies with MOC (CoNA and PAT) against the proposed adaptive boost controller, adBoost (CoNA++ and PAT++). We simulated 100 synthetic applications to evalu-

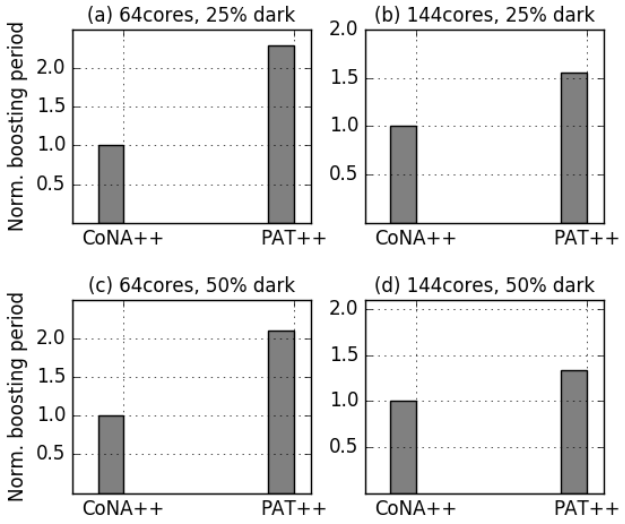



Figure 4.7: Normalized boosting periods with dense and sparse mappings for different configurations. CoNA++ and PAT++ represent dense and sparse mapping configurations with boosting.

ate the performance of these approaches over different configurations of network sizes - 64 core and 144 core, and amount of dark cores - 25% and 50%. Figure 4.7 shows the normalized period of boosting that could be sustained with each of the above core and percentage of dark silicon combinations. Normalized boosting period is calculated as the accumulated sum of the number of boosting epochs elapsed during the overall simulation time. This measure represents the efficacy of each approach in utilizing the available thermal budget for performance boosting. In Figure 4.7, CoNA++ and PAT++ represent dense versus sparse mappings when used in combination with the proposed adBoost controller. PAT++ achieves longer periods of sustained boosting in all configurations (a)-(d), when compared to CoNA++. This can be attributed to PAT++ strategy which prefers sparsity between active cores, slowing down temperature accumulation. Boosting active cores that are already operating at lower temperatures allows the frequency scaling decision to be sustained for longer intervals. Both higher utilizable power budgets and lower operating temperatures from dark silicon patterning favor PAT++ in sustaining longer boosting periods using the adBoost controller. Boosting in case of CoNA++ could be sustained only for lower periods, given the contiguous nature of mapping that accumulates temperature faster. Boosting the cores which are already at higher temperatures would further accelerate on-chip temperatures approaching towards the critical temperature, limiting the effective boosting period.

Figure 4.8 shows the normalized throughput gain with different mappings and controller strategies over different network sizes. CoNA and PAT represent throughputs of dense and sparse mappings using MOC controller (i.e., without

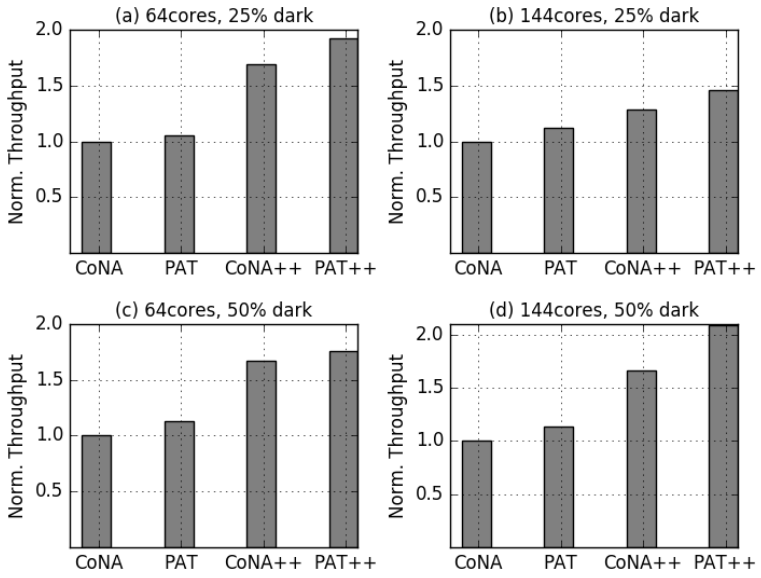



Figure 4.8: Normalized throughput for different mapping strategies without boosting (CoNA and PAT) and with boosting (CoNA++ and PAT++) for different configurations.

boosting) and CoNA++ and PAT++ represent the same with adBoost controller (i.e., with boosting). For every network size and percentage of dark cores, normalized throughput is increasingly higher in the order of CoNA, PAT, CoNA++ and PAT++. PAT has higher throughput than CoNA, since surplus budget gained from patterning is utilized to activate more cores. On the other hand, even the denser mapping, CoNA++, has higher throughput than both CoNA and PAT, benefiting from the performance boosting using the adBoost controller. The PAT++ - sparse mapping along with boosting, has the highest throughput in comparison with the all the other cases. As mentioned above, the longer boosting periods that PAT++ could sustain automatically reflects in higher throughput. Detailed evaluation of different aspects viz., run-time power and thermal profiles, turn around times, latency penalties with sparse mappings, percentile boosting period in overall execution time - for different core and percentage of dark silicon configurations are presented in the attached Paper II.

### Summary

This Chapter presents an adaptive thermal aware performance boosting technique to improve power budget utilization and meet performance surges of applications. We exploit dark silicon patterning based mapping to create enough thermal head-

room for performance boosting. We design a boost controller that uses thermal feedback to make frequency scaling decisions upon specific performance requirements of applications. We evaluated the proposed strategy of sparse mapping and boosting (adBoost) against dense mapping without boosting, to demonstrate the performance gains with adBoost. The contents presented in this Chapter are based on the attached Paper II.

# Chapter 5

# Approximation for Maximizing Performance

The previous have chapters presented techniques for increasing the power budget limits through mapping and efficiently utilizing the available power budget through boosting for improving performance. Intensive workload characteristics and/or increase in simultaneous execution requests (from concurrent applications) still creates performance demands that can result in higher power consumption. Under such workload scenarios, power capping i.e., restricting the peak power consumption to a fixed limit, becomes a significant aspect of system design to ensure thermal safety. While power consumption is reduced by lowering system resources, such an actuation results in performance degradation. Resource management techniques relying on traditional power knobs make inevitable compromises on performance loss [Muthukaruppan et al., 2013b] [Khdr et al., 2017]. Leveraging the fact that some of the workloads are error resilient, relaxing the accuracy can restore the performance. In this chapter, we present approximation as a dynamic knob for improving performance under a given power budget. We opportunistically trigger approximate mode of execution subject to performance requirements and power constraints.

## 5.1 Power Capping

Existing power capping techniques largely actuate on conventional knobs such as DVFS [Vega et al., 2013] [Gaspar et al., 2014] [pac, 2011], Near Threshold Computing (NTC) [Wang and Skadron, 2013] [Schlachter et al., 2015], (Per-core/cluster) Power Gating (PCPG/PG) [Ma and Wang, 2012], advanced scheduling [Kapadia and Pasricha, 2015], task mapping [Kanduri et al., 2015b] and task migration [Muthukaruppan et al., 2013b] [Gaspar et al., 2015] for reducing power consumption.

### 5.1.1 Limitations

With the cubic dependence of power consumption on voltage and frequency levels, DVFS is effective in power capping. However, appropriate resource scaling decisions with DVFS depend on availability of fine-grained frequency levels and overhead of actuation. At lower technology nodes where supply voltage is already relatively low and contribution of leakage power to total power consumption is high, DVFS alone would not suffice for power capping. Near Threshold Computing (NTC) is a variant of DVFS, where supply voltage is aggressively scaled beyond threshold voltage for ultra low power operation [Schlachter et al., 2015]. Such extreme down scaling of system resources results in dim silicon baseline, as opposed to dark silicon [Wang and Skadron, 2012]. This scenario features more number of active cores - however, operating at minimal voltage and frequency levels. NTC approach targets improving the overall throughput by compromising on per-core/per-application performance with severely throttled down yet active cores [Wang and Skadron, 2013]. Power gating (PG) provides significant reduction in power consumption by shutting down cores (granularity can vary) limiting both idle and active power. Although, this actuation incurs an overhead in switching between active and idle states and the overall performance degradation with only fewer cores being active. As proposed by [Vega et al., 2013] and [Ma and Wang, 2012], combination of power gating and DVFS can alleviate power capping challenge to a certain extent.

### 5.1.2 Performance Implications

Most of the power capping techniques are reactive in nature i.e., they are invoked upon a power violation. Further, the objective of lowering the instantaneous power consumption is confined to a short-term. With changing workloads, this can lead to frequent triggering of power management decisions which up/down scale resources perpetually. Such oscillation between high/low power states contributes to inefficient (over/under) resource utilization. Some strategies have addressed this challenge by classifying the system state into multiple levels of power consumption to represent the likeliness of potential power violation, and then manipulating power actuation knob settings by the extent of power headroom available [Wu et al., 2016] [Rahmani et al., 2017] [Intel Corporation, 2014]. Despite the effectiveness in reducing power consumption and pro-active decision making, most of the power capping strategies result in an inevitable performance degradation. Efficient power capping approaches have tried to maximize the performance under a given power cap by exploiting application level characteristics and dynamic workload variation scenarios [Cochran et al., 2011] [Hoffmann et al., 2012] [Zhan and Reda, 2013] [Tang et al., 2016] [Sha et al., 2018]. All these techniques though still rely on traditional power knobs of DVFS, power gating, core folding and degree of parallelism - which present an orthogonal trade-off between power and performance.

The extent of performance degradation though depends on power knobs used, capping/actuation strategy and concurrent workload pressure. In summary, satisfying the objective of power capping to ensure thermal safety, while maximizing the performance that can be extracted within the set cap is challenging with conventional power knobs alone. In the following sections, the central idea of exploiting applications' error resilience characteristics in the context of power capping will be presented.

## 5.2 Approximation for Performance

Applications from certain domains exhibit an algorithmic level tolerance to inaccurate computations [Li and Yeung, 2007]. This can be largely attributed to their redundant and noisy input data, iterative nature of computations and the eventual result being perceptive such as image, video, graphics etc [Esmaeilzadeh, 2015]. Several techniques across the computing stack have exploited such behavior to deliberately approximate resilient applications for performance and energy gains. The idea of exploiting accuracy trade-offs and its efficacy have already been introduced in Section 2.2.

### 5.2.1 Techniques

Existing approximation techniques include programming constructs [Sampson et al., 2011] [Bornholt et al., 2014] [Ansel et al., 2009] [Misailovic et al., 2014], compilation [Baek and Chilimbi, 2010] [Samadi et al., 2013] [Samadi et al., 2014] [Sidiroglou et al., 2011a], architectural innovation [Thwaites et al., 2014] [San Miguel and Badr, 2014b] [Esmaeilzadeh et al., 2012b] [Esmaeilzadeh et al., 2012c] [Venkataramani et al., 2013] and hardware optimization [Gupta et al., 2011] [Kahng and Kang, 2012] [Varatkar and Shanbhag, 2006] [Kedem et al., 2011] [Kim and Shanbhag, 2014] [Lingamneni et al., 2013]. An abstract classification on approximation techniques is to view them as *functional* approximation and *implementational* approximation. Although, there are numerous other classifications on approximation techniques based on models of computation [Mishra et al., 2014], application domain and the nature of approximation used [Mittal, 2016]. Functional approximation can be realized purely with algorithmic transformations and requires no effort from system software or hardware design. For example, loop perforation - skipping some iterations of nested loops [Sidiroglou et al., 2011a] and code perforation - skipping less significant tasks within an application [Rinard, 2006] provide approximate kernels as expressed by the programmer. On the other hand, implementational approximation requires efforts vertically including programming constructs, compiler passes, instruction set extensions, architectural blocks and circuitry. In the context of this thesis, functional approximations are considered as the primary source for exploiting accuracy trade-offs without tweaking the hardware.

### 5.2.2 Rationale

Algorithmic level transformations and software approximations such as loop perforation, task skipping and input data sampling reduce the amount of computation of an application. With relatively fewer instructions, such functionally approximate kernels will provide higher performance at the expense of possible inaccuracies. Most of the existing approximation techniques and frameworks are open-looped without specific objectives of run-time power/performance management. Some of the techniques have adapted the implementation for run-time quality monitoring or maximizing performance within minimal error in the context of a single application/domain. An efficient way to exploit accuracy trade-offs is to opportunistically trigger approximate execution subject to system dynamics such as power consumption and performance requirements. This dissertation proposes approximation as another knob for power/performance management. The idea is to switch the mode of execution from accurate to approximate upon performance requirements and/or power constraints. The approximation knob, when actuated cooperatively with other power knobs, can lend performance and energy benefits to the effectiveness of power actuation. Specifically, approximation can be used to cover up the performance loss in the context of power capping. Figure 5.1 shows the hierarchical view of the proposed approach on using approximation as a knob, in combination with other power knobs. Our approach integrates approximation in a typical observe-decide-act structure of resource management platforms, where run-time power and performance are observed, different knob settings are decided and these decisions are enforced in the actuation phase. While approximate hardware and architectural techniques require substantial system design effort and can provide larger gains, the proposed approach relies specifically on minimal programmer and architectural support for software approximations - since one of the objectives of this dissertation is to leverage approximation in the context of run-time resource management.

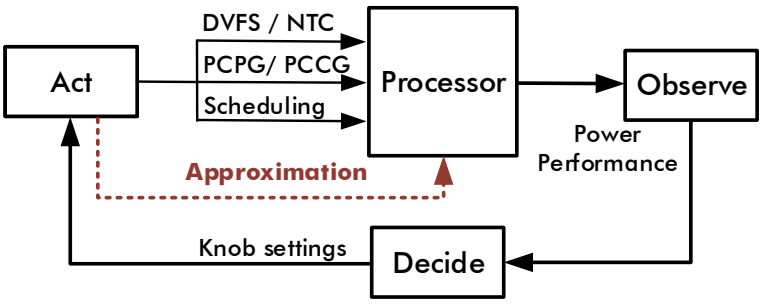



Figure 5.1: Approximation as a resource management knob

## 5.3 Approximation as a Dynamic Knob - APPEND

The proposed approach is used in the context of NoC based many-core systems that support multi-programmed workloads. Using approximation (APPX) along with other power knobs requires a framework that *monitors* critical system chip parameters viz., power consumption, per-application performance and per-chip throughput, and network intensity, *decides* on appropriate knob settings and *acts* upon the decisions made. Power consumption, overall workload intensity, resource utilization and network intensity of the chip varies subject to incoming applications' requests. The proposed framework monitors these metrics over fixed epochs, to make i) power capping/actuation decisions through power knobs of DVFS and power gating (PG) and ii) performance management decisions using approximation (APPX) - as per the requirements. From here on, the proposed approximation enabled power/performance management framework is referred to as APPEND.

### 5.3.1 System Architecture

Hierarchical view of the proposed framework is shown in Figure 5.2. Applications are assumed to arrive at an unknown and random sequence. Workflow of the proposed approach and components of the framework shown in Figure 5.2 will be described in the following sections. The core model and task graph based application model that are used in our approach were already described in Section 3.3.1.

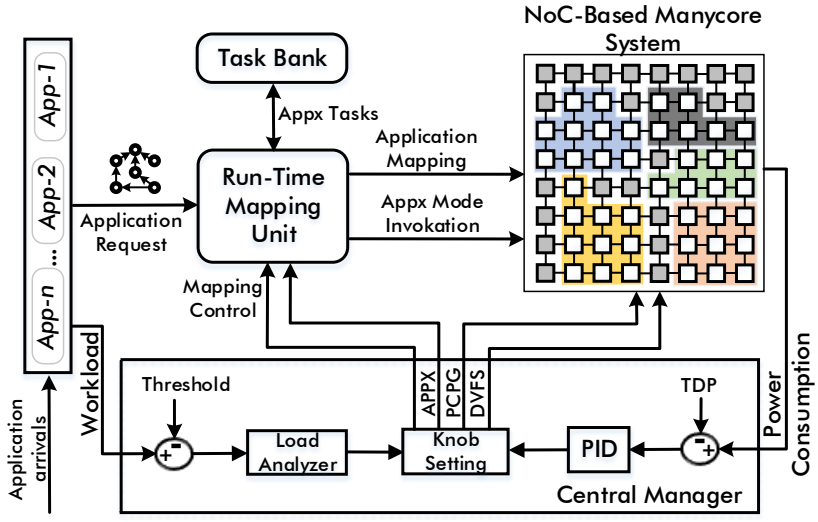



Figure 5.2: System architecture

### Central Manager

Resource allocation decisions are made by the central manager, which consists of a PID controller - for power knob settings and *load analyzer* - for performance (approximation) knob settings. The power controller used in this framework is the same as presented in Chapter 3.3.1 and elaborated in [Haghbayan et al., 2014] and [Rahmani et al., 2015a]. The power controller monitors instantaneous power consumption, compares it against the upper bound on power to estimate power headroom available. This is used to determine DVFS and PG knob settings that will be acted upon. Every incoming application's execution request will be served, upon availability of power budget and free cores onto which the application can be mapped. Unavailability of power budget and/or free cores due to applications that are already running on the chip forces the incoming applications to wait in the queue longer. This time elapsed between an application's execution request and beginning of the actual execution is *waiting time* (also referred as turn-around time [Fattah et al., 2012b]). We use moving average of the waiting time of incoming applications, *Average waiting time* (AWT), to quantify workload intensity and performance requirements. A longer AWT intuitively represents higher workload - either with higher number of application requests (application arrival rate) and/or longer per-application latency of applications that are currently running on the chip. The *load analyzer* compares AWT against a pre-determined and parameterizable threshold to determine workload intensity and thus APPX knob settings accordingly. Upon a performance violation i.e., AWT exceeding the threshold, APPX is invoked, setting the mode of execution to approximate. Enforcing the mode switching is detailed in the following.

### Mode Switching

When the execution mode is accurate, the run-time mapping unit (RMU) maps the accurate version of the incoming application. We assume approximable applications to be presented with at least one variable accuracy implementation - to support switching the mode of execution from accurate to the provided approximate

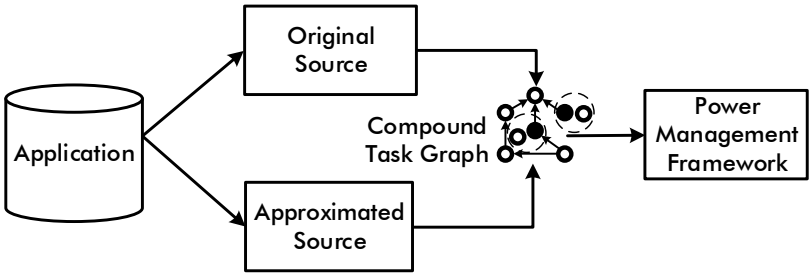



Figure 5.3: Workflow of proposed approach. Solid filled tasks are approximable.

version. Such approximable applications are represented using a compound task graph, where one or more tasks of the application have one or more possible approximate versions. Formulation of the compound task graphs is shown in Figure 5.3 - the approximate tasks are shown in solid fill. Each applications' task(s) that has an alternative approximate version (if any) is embedded within the compound task graph. For example, in Figure 5.3, there are two approximable tasks - both accurate and their approximate versions are shown together (highlighted). For all incoming approximable applications, approximate task(s) are buffered into the *Task Buffer*. Depending on the mode of execution/APPX knob setting, RMU chooses the version of task to be included at the application mapping stage, while the other versions are buffered. With the invocation of APPX knob, mode of execution is switched to approximate from accurate. This presents two possible scenarios for mode switching viz., i) incoming applications - map approximate versions and ii) currently running applications - switch the mode of execution by task replacement. For incoming applications, the RMU maps them in their approximate mode, by including the approximable tasks instead of accurate tasks, until the mode is switched back to accurate. For applications that are already running on the chip, RMU identifies the corresponding approximate task of the specific application from the *Task Buffer* and replaces the accurate task with the approximate task.

For evaluation, applications are modeled compromising concurrent tasks that execute periodically, repeating the task in a loop. Such streaming nature of computation with incoming samples of data per iteration allows to relax certain aspects of computation when new batch of data arrives every iteration. When mode of execution is switched to approximate, the RMU waits until the current iteration of accurate task's computation is finished. Once the task is completed, the RMU loads the corresponding approximate task on to the chip, replacing the accurate task. Figure 5.4 demonstrates a simple use case of task replacement during mode switching. The example shown has three tasks 1, 2 and 3 out of which task2 is approximable. Upon invocation of APPX knob triggering approximate mode of execution, mode switching happens in the following sequence. i) The RMU finds the approximate version of task2, $task2\_appx$, from the *Task Buffer*. ii) It waits until data from task2 (data23) is received at task3. iii) $task2\_appx$ is loaded by fetching the instruction stream into the cache. iv) After the data is received at task3,

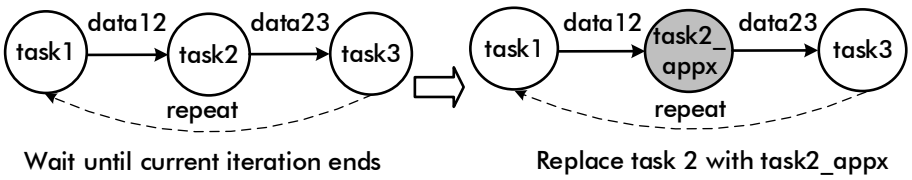



Figure 5.4: Example demonstrating mode switching for an application with 3 tasks

the execution of task2 will now start from new instruction stream of $task2\_appx$. Algorithmic flow of the control strategy is detailed in Paper III, which is attached in Part II of the dissertation.

## 5.4 Evaluation

For evaluation purposes, a mix of approximable and non-approximable applications were chosen. Multiple instances of k-means, k-nearest neighbors, linear regression and least squares applications were used for simulation. Each application has two levels of approximation viz., Apx-1 and Apx-2. Applications simulated are summarized in Table 5.1. These workloads largely rely on iterative computational methods, such that accuracy of the result converges towards optimal solution with more number of iterations. Since an accurate solution may not exist and lower convergence could still offer an acceptable result, they become suitable candidates for approximation. Levels Apx-1 and Apx-2 for linear regression and least squares applications are realized through loop perforation [Sidiroglou et al., 2011a], by skipping 10% and 25% of computations over input data respectively. For k-means clustering and k-nearest neighbors, we use relaxed convergence (RC). We compromise on number of flips for k-means, and coverage of neighbors and training data sets respectively for k-nearest neighbors. In these two applications, Apx-1 and Apx-2 correspond to relaxing 1% and 5% of convergence respectively.

The proposed framework which combines power knobs - DVFS and PG with APPX knob is compared against power management with i) only DVFS, ii) only PG and iii) combination of DVFS and PG [Rahmani et al., 2015b]. Figure 5.5 shows the average waiting time with each of the above strategies, simulated over 100 applications. All the strategies could service incoming applications within minimal waiting time when the arrival rate is lower. As the number of incoming applications increase, AWT increases with DVFS, PG and DVFS+PG. These strategies initially map incoming applications, and subsequently throttle down system

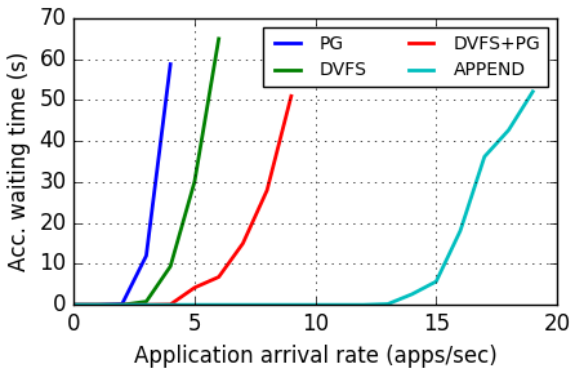



Figure 5.5: Average waiting time with the proposed approach

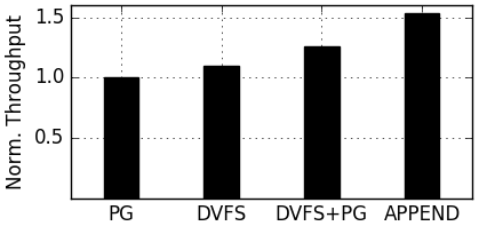



Figure 5.6: Normalized throughput with the proposed approach

resources to stay within the power budget. Further, intensive applications running on the chip with longer latency compounds the AWT. With the proposed approach, workload intensity and threshold of AWT are monitored - which triggers approximate mode of execution. Invocation of APPX knob replaces accurate tasks with approximate tasks to reduce computational workload, resulting in higher performance of currently running applications. As the applications finish their execution relatively faster, power budget and free cores become available for incoming applications. Hence, the AWT with APPEND is minimal, even at a higher application arrival rate and workload intensity. Consequently, the overall throughput i.e., time taken to finish the execution of all the applications is also significantly higher using the APPX knob, compared against other traditional power knob based strategies. Figure 5.6 shows the normalized throughput of all the aforementioned approaches. Lower AWT and higher application service rate of APPEND consequently reflects in higher throughput. Normalized throughput gain for APPEND is about l.5 $\times$ better than that of PG and DVFS knobs alone and about l.2 $\times$ better than that of MOC (combination of DVFS and PG), showing a significant performance gain, within the power budget. APPEND is equally effective in power capping, similar to the other power knobs. Instantaneous power consumption and effectiveness in power capping with APPEND are presented in the attached Paper III. It should be noted that the performance gains with APPX knob comes at the expense of accuracy loss. The relative mean square error along with the performance gains for each application simulated are shown in Table 5.1. Since individual knobs of DVFS, PG and the MOC (DVFS+PG) are not approximation aware, accuracy is retained in these cases.

## 5.5 Exploiting Error Sensitivity

The APPEND framework used two coarse grained levels of approximation to switch to, upon invocation of APPX knob. Subject to system dynamics and application's characteristics, such a coarse grained invocation of approximation at times might lead to over/under compensation decisions on accuracy trade-offs. Further, different applications exhibit different levels of tolerance to errors and performance gains with approximation. Such behavior presents with the opportunity

Table 5.1: Applications simulated

| Application | Norm. Perf | | Relative Error (%) | |
|---|---|---|---|---|
| | Apx-1 | Apx-2 | Apx-1 | Apx-2 |
| Lin. Reg | 1.1 | 1.2 | 6 | 13 |
| k-means | 1.9 | 5.6 | 1 | 5 |
| k-NN | 1.1 | 1.3 | 3 | 8 |
| Least Sq. | 1.07 | 1.26 | 5 | 12 |

for selecting the i) appropriate applications as candidates for approximation and ii) their level of approximation to switch to - in order to maximize performance gains while minimizing error induced. For instance, approximating a certain application might have better performance gains than that of the others, while approximating to a specific level might result in higher performance-per-error. This is quantified through application's sensitivity to error, which is defined as:

$$Sensitivity = \frac{Perf_i - Perf_{i-1}}{Error_i - Error_{i-1}} \tag{5.1}$$

where $Perf_i$ and $Error_i$ are the performance and error induced at $i^{th}$ level of approximation. *Sensitivity* of an application for any two given levels of accuracy would be high when the performance gained by lowering accuracy is high or when the accuracy loss in performance improvement is lower. Sensitivity metric can be used to identify appropriate applications for mode switching among concurrently running approximable applications.

**APPEND+**

The APPEND framework is extended with control decisions on the APPX knob settings which consider sensitivity metric along with power/performance dynamics. The extended framework, referred to as APPEND+, is elaborated in Paper IV. The overall control strategy is enhanced with a parameterizable number of fine-grained approximation levels that can be chosen from. For evaluation, additional micro kernels of FFT, matrix multiplication, low pass filter, and early warning score system (EWS) [Azimi et al., 2017] and fall detection [Gia et al., 2018] from IoT domain were used. Approximation strategies for the machine learning kernels are already described in Section 5.4. Among the other kernels, we approximate the computation involving exponential functions with relaxed memoization and low precision storage of twiddle factors for FFT. We reduce the number of coefficients by 10% per each level of approximation for low pass filter. We reduce the number of samples of heart rate sensor and compromise data fusion for EWS. We

Table 5.2: Application Sensitivity

| Application | | Sensitivity | | | |
|---|---|---|---|---|---|
| | | Level-1 | Level-2 | Level-3 | Level-4 |
| Machine Learning | Linear Regression | 10.1 | 0.09 | 0.13 | 0.23 |
| | K-means | 6.51 | 0.28 | 0.44 | 0.64 |
| | K-NN | 0.07 | 0.06 | 0.06 | 0.43 |
| Signal Processing | FFT | 0.01 | 0.01 | 0.05 | 0.07 |
| | LPF | 0.6 | 5.5 | 2.9 | 1.84 |
| | Vector Multiplication | 0.24 | 1.59 | 0.31 | 0.83 |
| IoT | EWS | 0.95 | 0.26 | 0.58 | 0.3 |
| | Fall Detection | 1.02 | 0.86 | 0.72 | 0.14 |

reduce the sampling rate of accelerometer and simplify the logic of magnitude vector computation for fall detection. These applications and their sensitivity metrics are summarized in Table 5.2. Since most of these applications operate iteratively over input vectors of data, loop perforation and task skipping were used to realize approximate versions. Each application is annexed with 4 levels of approximations, with increasing levels of accuracy relaxed. They are further elaborated on software approximation used, normalized performance and relative error induced in the attached Paper IV.

We simulate the system over a period in which 200 instances of different applications are serviced. For evaluation, we consider effectiveness in power capping, AWT and normalized throughput metrics. Instantaneous power consumption and power capping for TDP and TSP baselines are elaborated in the attached Paper IV. Figure 5.7 shows the AWT of different strategies using different combinations of power/performance knobs, including APPEND and APPEND+. APPEND+ has the best AWT, preceded by APPEND, MOC, PG and DVFS in order. As demonstrated in Section 5.4, DVFS, PG and MOC actuation have higher AWTs as the application request rate starts increasing - largely due to the limitations of traditional power knobs. APPEND+ on the other hand have a near-zero AWT for as long as $5\times$ more than DVFS and PG knobs alone, and $3\times$ more than that of MOC. Figure 5.8 shows the normalized throughput for different strategies for two baselines - i) using TDP as the upper bound and ii) using TSP (as described in Chapter 3) as the variable upper bound. With different levels of approximation to choose from, lower AWTs and higher application service rates, performance gains with APPEND+ are higher. Normalized throughput is relatively higher with TSP, in comparison with using TDP. The significance of efficient power budgeting and utilization described in Chapters 3 and 4 is reflected here. Furthermore, APPEND+ has the possibility of aggressively approximating (if any) suitable applications with higher sensitivity (performance per error), which translates into lower per-application latency, lower AWT and higher overall throughput.

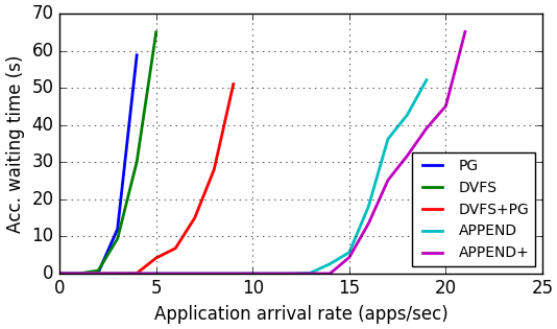



Figure 5.7: Normalized throughput with the proposed approach

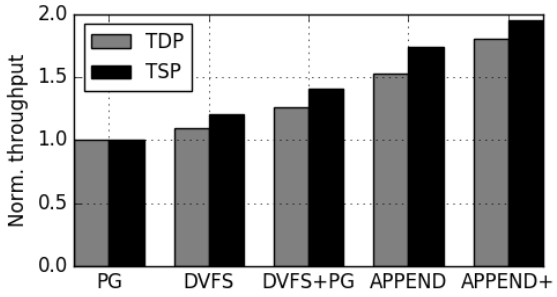



Figure 5.8: Normalized throughput with the proposed approach

### Summary

In this Chapter, we presented the idea of using approximation as a dynamic knob to address the performance loss incurred in power actuation. We use functional approximations to dynamically switch between accurate and approximate modes of executions subject to system dynamics and applications requirements. We evaluated the proposed approach against existing state-of-the-art power management frameworks to establish the performance gains within the power budgets using approximation, for an acceptable loss of accuracy. Further, we enhanced the proposed approach considering error sensitivity of each application to minimize the error induced per performance gained. The contents of this chapter are based on original publications, which are attached as Paper III and IV.

# Chapter 6

# Co-ordinated Approximation

Chapter 5 presented the idea of using approximation to minimize performance loss incurred in power actuation, by reducing the workload. This approach is mainly targeted at maximizing the performance within the power cap, fully utilizing the available power budget. Such strategy for approximation suits throughput applications running on platforms that can provide required high performance. On the other hand, another spectrum of latency sensitive applications running on embedded processors demand a certain level of performance, sustained over longer periods within minimal power consumption [Lo et al., 2014]. Such scenarios require co-optimization of power and performance, to make appropriate resource allocation decisions based on system and application constraints. Based on the insights presented in Chapter 5, approximation can be opportunistically used to i) provide the required performance and/or ii) minimize power consumption. The idea is to lower the resources to the reduced (approximated) workloads, which can be used to i) provide low power operation while guaranteeing the required level of performance or ii) provide high performance with the same amount of resources and power budgets. In this chapter, we present coordinated approximation to make power/performance co-optimization decisions to provide performance guarantees within minimal power consumption.

## 6.1 Approximation on Heterogeneous Platforms

Heterogeneous multi-processor (HMPs) platforms provide cores with different power-performance characteristics that can be leveraged for optimizing resource utilization. In this chapter, we specifically consider heterogeneous multi-core processing (HMP) systems, which could typically comprise of cores with different power-performance characteristics including high-performance and/or power efficient CPUs, throughput processors (GPUs), customizable logic (FPGAs) and domain specific accelerators etc [Hardkernel co., 2014] [Lindholm et al., 2008]. While heterogeneous architectures expose different options for execution, run-

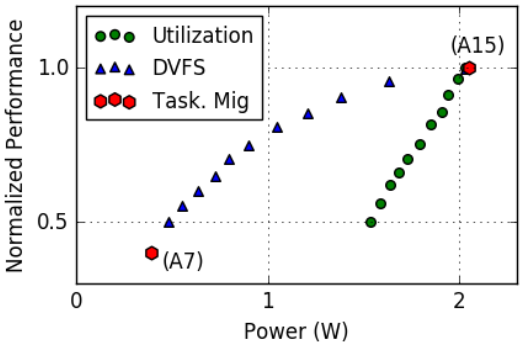



Figure 6.1: Power-performance behavior with different knobs.

time management techniques are required to exploit such behavior to maximize resource efficiency. Although, different constraints exacerbate the run-time management challenge, including i) diverse performance requirements of applications ii) fixed and limited power and energy budgets iii) varying workload characteristics and iv) core-level heterogeneity. Approximation on heterogeneous systems opens up both power-performance and accuracy-performance Pareto-spaces, with the choice of using approximation to i) maintain the required performance within lower power or to ii) achieve high performance by utilizing available power budget, or iii) a combination of both. The choice among these opportunities depends simultaneously on applications' requirements and system constraints, and resource allocation strategy.

### 6.1.1 Power-Performance Co-optimization

Existing resource management techniques largely employ knobs such as Dynamic Voltage and Frequency Scaling (DVFS) [Cochran et al., 2011], task migration [Muthukaruppan et al., 2013b], power gating and CPU quota scaling [Rahmani et al., 2015a,Rahmani et al., 2016] etc., for power optimization. Performance requirements of individual applications, dynamic nature of varying workload characteristics and power and energy budgets will drive the choice of suitable combination of knobs. Both power and performance being effected with any given combination of actuation knobs, a co-optimization strategy becomes more relevant in this context. Within heterogeneous architectures, actuation of each of these knobs has a different effect on power and performance. We demonstrate such behavior using a micro-kernel of least squares curve fitting over 1 million pairs of floating point data, executed on a heterogeneous combination of ARM A7 (low power) and A15 (high performance) cores [Hardkernel co., 2014]. Figure 6.1 shows the power-performance behavior using DVFS and CPU utilization on an A15 core, along with task migration onto an A7 core. Here, frequency is scaled in steps of 100 MHz and CPU utilization is scaled in steps of 5% of quota allocated to achieve different

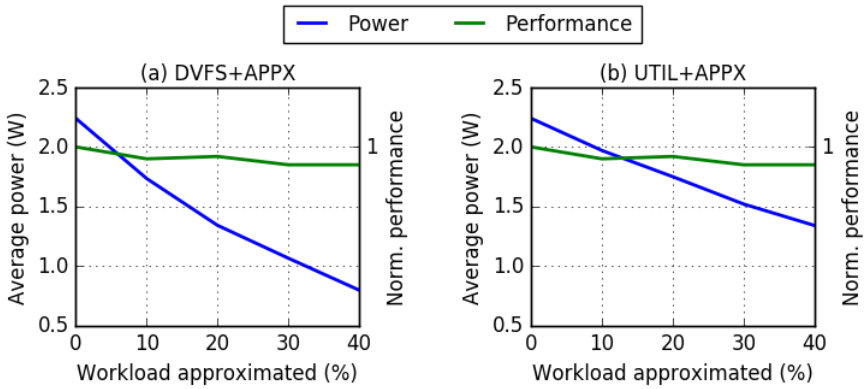



Figure 6.2: Power-performance behavior with joint actuation of approximation and power knobs. (a) DVFS and APPX - frequency scaled from 1.8GHz to 1GHz in steps of 0.2GHz, (b) CPU Utilization and APPX - utilization scaled down in steps of 10%. In both cases, APPX corresponds to 0%-40% of loops skipped over 1 million pairs of least squares kernel.

power states. This shows a wider range of Pareto-space that can be explored to achieve different levels of performance within different power budgets.

For maximizing performance within the fixed power budgets, these techniques propose joint actuation of power knobs [Cochran et al., 2011, Vega et al., 2013, Rahmani et al., 2015a, Ma and Wang, 2012, Muthukaruppan et al., 2013b] and exploitation of application characteristics [Hoffmann et al., 2012, Gaspar et al., 2016]. These strategies minimize the effect of power actuation on performance, yet inevitably compromise on performance for thermal safety. This specifically affects a class of streaming applications from domains such as machine learning, artificial intelligence, multimedia processing, computer vision and Internet-of-Things [Jouppi et al., 2017, Schulz et al., 2017]. These applications may i) include user interaction, ii) serve as intermediate results to other computational kernels (e.g., computer vision in self-driving vehicles), making them latency-sensitive. Typical power actuation techniques cannot provide real-time performance guarantees required to ensure user satisfaction and responsiveness. At the same time, such applications are relatively error resilient (i.e., approximable) due to their algorithmic tolerance nature, redundant input data and perceptive end results. As described in Chapter 5, functionally approximate kernels present a Pareto space of accuracy-performance trade-offs that can be leveraged at run-time to achieve better performance within lower power and/or energy resources, for an acceptable loss in accuracy [Baek et al., 2010, Sidiroglou et al., 2011b]. Figure 6.2 shows the effect of using approximation (APPX) in combination with power knobs viz., DVFS and Utilization in sustaining a fixed level of performance, even with lower resources. The application and experimental setup are the same as described in case of pre-

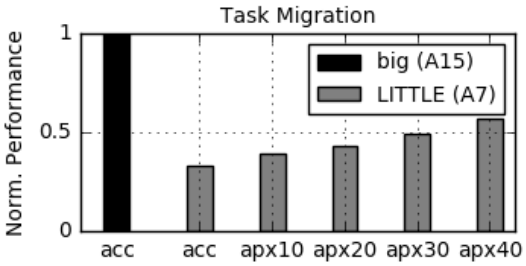



Figure 6.3: Joint actuation of approximation and task migration. apx10-apx40 represents 10%-40% of loops skipped over 1 million pairs of least squares kernel.

vious example, shown in Figure 6.1. Traditional power actuation with DVFS and CPU utilization alone degrades performance as (V,F) levels and utilization are lowered for power actuation. When using them in combination with APPX, resources are lowered on top of reduced workloads, which avoids the performance degradation, yet provides low power operation. As shown in Figure 6.2 (a) and (b), with every progressive lowering of frequency levels and CPU time slices, workloads are also simultaneously reduced through approximation. As a result, power consumption is reduced while performance is still sustained, as approximation covers the possible performance loss with scaled down resources. Another alternative is to use APPX for improving performance within the same power budget, which becomes relevant particularly on wimpy cores in a heterogeneous architecture. Figure 6.3 shows the normalized performance with task migration alone, in comparison with the combination of task migration and APPX. The application and experimental setup is the same as used in previous example. Upon migrating the application from high performance cluster (A15) to low power (A7) cluster, both power consumption and performance are reduced significantly. However, joint actuation of APPX with task migration can provide a relatively higher performance within the same power budget. As shown in Figure 6.3, different levels of approximation can further provide more options for minimizing performance degradation, yet restore the power savings with task migration.

### 6.1.2 Coordinating APPX with Power Knobs

While joint actuation of approximation with power knobs opens up opportunities for meeting performance within power budgets, disciplined tuning of these combinations is necessary for maximizing resource efficiency. We demonstrate the effect of combining approximation (APPX) knob with power actuation through an example, using the control strategy presented in [Kanduri et al., 2016]. For simplicity, we use only DVFS as the power knob, and loop perforation [Sidiroglou et al., 2011b] based approximation as the APPX knob. As a test case, we use least squares curve fitting application over 24000 pairs of inputs, and 10% of the loops skipped to realize the approximate version. Figure 6.4 (a) shows the power consumption

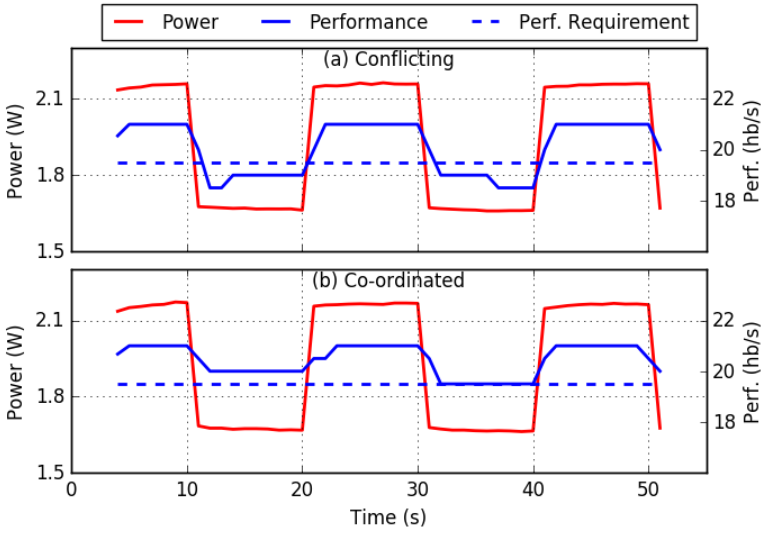



Figure 6.4: Power-performance actuation combining power knobs with approximation. (a) Conflicting decisions, (b) Coordinated decisions.

and corresponding performance (measured in heartbeats/s, as described in Section 6.2.1) of the application. We assume $2W$ as the threshold for power actuation, 10 seconds (for presentation ease) as the control period for resource management decisions, along with a target performance requirement (the red line). At $t = 10s$, power exceeds the threshold, triggering DVFS to reduce power, while performance requirements are satisfied. Subsequently, power consumption is reduced during the control period, also reducing the performance below the requirement.

At $t = 20s$, performance requirements are not met - triggering the APPX knob, while power consumption is below the threshold - triggering DVFS (upscaling). With the uncoordinated actuation decisions performed by [Kanduri et al., 2016], performance is restored with some accuracy loss, while power consumption again exceeds the threshold during the following control period. At $t = 30s$, performance requirements are met, hence the application is switched to accurate execution; DVFS is triggered again to lower power consumption. A similar oscillation between accurate-approximate executions and high-low frequencies will continue in the subsequent control periods. Although the performance violation at $t = 10s$ could be recovered with approximation at $t = 20s$, the lack of coordination between DVFS and approximation either over-compensates or under-compensates the knobs' actuation. This occurs because of the reactive nested loop structure of such control policies (as in [Kanduri et al., 2016]), where power and performance actuation happens independently, and is mutually agnostic. Figure 6.4 (b) shows power and performance for the same scenario, with a coordination among DVFS and approximation. At $t = 10s$, DVFS is triggered to address power violation. At the same time, considering the performance loss with power actuation, approx-

imation is triggered pro-actively by predicting the potential loss in performance due to DVFS (downscaling). As a result, performance requirements are met during the subsequent control period despite the voltage/frequency (VF) downscaling. At $t = 30s$, VF is upscaled with availability of power headroom and the same is indicated to the performance manager. This allows the performance manager to make informed decisions on performance actuation, which in turn restores the accurate mode of execution. As a conclusion, while combination of approximation and DVFS addresses the performance loss due to power actuation to an extent, tight coordination between power and performance actuation provides better results in i) meeting performance requirements often, and ii) minimizing the accuracy loss due to approximation.

### 6.1.3 Rationale

In a realistic scenario, the resource management policy has to consider i) a wider set of power knobs viz., DVFS, CPU quota assignment, task migration and fine-grained levels of approximation, and ii) variable workload characteristics having applications entering and leaving the system with an unknown trend, and each one being characterized by a specific performance requirement. When considering these aspects, knob actuation for maximizing performance within minimal power consumption and accuracy loss becomes dramatically complex and challenging. To address this issue, we propose a novel run-time resource management strategy for heterogeneous multi-core systems running dynamic and unknown workloads, exploiting approximation. Our strategy coordinates power knobs with approximation knob to meet both performance requirements and the power constraints. Subject to system dynamics and applications' requirements, we switch application's mode of execution from accurate to approximate to meet i) power constraints - by lowering resources to approximate kernels, and/or ii) performance requirements - by opportunistically reducing workloads. In either case, the coordinated usage of approximation knob complements the traditional power knobs, thereby overcoming their limitations and conflicting decisions. Design and implementation of the proposed framework and resource management policy, and evaluation are described in the following sections.

## 6.2 Dynamic Approximation Framework

We present a run-time resource management framework for heterogeneous systems that uses approximation dynamically along with other power knobs and coordinates both power and performance allocation decisions simultaneously. This section will describe design of the framework, strategy for run-time approximation and the proposed resource management policy.

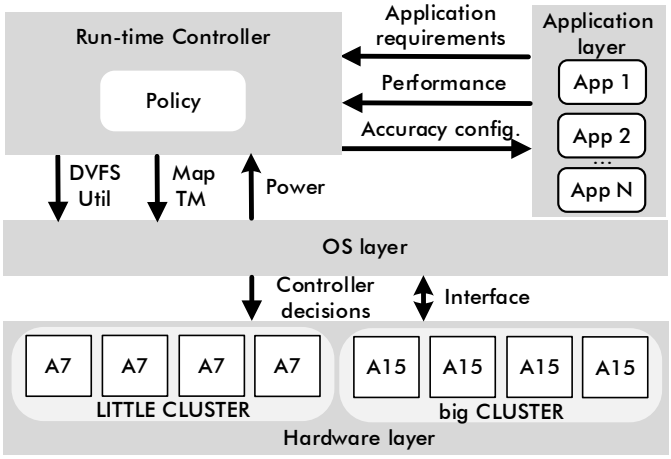



Figure 6.5: System architecture.

### 6.2.1 System Architecture

Overview of the proposed resource management framework, comprising hardware, software and interfaces, is shown in Figure 6.5. Workflow of the proposed approach and components of the system architecture will be described in the following. We specifically consider the widely used ARM big.LITTLE asymmetric multi-core model which is composed of a *big* cluster of high-performance power-hungry cores, and an alternative *LITTLE* cluster of power efficient wimpy cores. The processor is constrained by an upper bound on power consumption (TDP) [Pagani et al., 2014] specified for the chip's thermal safety. It should be noted that in previous chapters, TSP has been used as the dynamic upper bound, considering a many-core system baseline. Larger number of homogeneous active cores' spatial alignment (mapping) effects the utilizable power budget - making TSP relevant for such architectures, whereas TDP is a practical choice for embedded processors [Muthukaruppan et al., 2013a]. On-board power sensors are used for power monitoring, while voltage regulators are used to enable a per-cluster DVFS to support power management. The Operating System (OS) loaded onto the processor provides different utilities to control resource allocation decisions including i) task/application mapping (MAP) on a specific core(s) within a cluster(s), ii) task migration (TM) between different clusters, iii) CPU quota assignment (Util) to set the percentage of time/resources the core will exclusively provide per task/application, iv) DVFS to set frequency levels of a cluster and v) accuracy configuration.

### Performance Monitoring

Incoming applications are assumed to express their performance requirements, which will be used to make resource allocation decisions. To enable both expression and measurement of performance, we use the HeartBeat API [Hoffmann et al., 2010] - a library for extracting application level performance metrics. We annotate each application with the hooks provided by HeartBeats API to log the performance of a specific block of code, task or the entire application. The region of interest over which performance is measured may vary among applications. In this context, we specifically consider applications that are executed iteratively within a computationally intensive loop, where the main compute kernel is continuously repeated over batches of new inputs with every iteration. Streaming [Papazoglou and Georgakopoulos, 2003] and on-line data intensive [Lo et al., 2014] applications are good examples of such workloads. Performance of annotated applications is measured in terms number of iterations computed in a unit time i.e., heart beats per second ($hb/s$), which is proportional to the amount of data processed over a time window. The same metric is used for expression of user/application-level performance requirements, preferably as an acceptable range of throughput to be guaranteed. Expressing and measuring performance requirements through annotated applications is shown as a listing in Figure 6.6. Each application is annotated with APIs of the designed controller framework and HeartBeats library. `attach_ctrl(perf_req)` attaches the controller to the application and expresses `perf_req` as the level of required performance, while `log_heartbeat` measures the performance over the specified code block. Run-time accuracy configuration and controller strategy are described in the following sections.

### Run-time Controller and Accuracy Configuration

The *run-time controller* monitors power consumption and performance requirements to make decisions on appropriate settings of the aforementioned power/performance knobs. Incoming applications are assumed to be entering and leaving the system in an unknown sequence, causing a significant workload variation. The controller accesses system-level power sensors and enforces decisions on actuation knobs through the OS interfaces. Similarly, application level performance measures and accuracy configuration are enforced through the HeartBeats API. Unlike the traditional power knobs, invoking approximation dynamically requires the application to be configurable for approximation. We assume each application's main compute kernel to be possibly provided with an alternative software approximated implementation using strategies such as loop perforation [Sidiroglou et al., 2011b], relaxed convergence [Rinard, 2006] or reduced input data sampling [Sidiroglou et al., 2011c] etc. This allows configuring accuracy level of the application at run-time by passing a parameter as determined by the controller. As shown in Figure 6.6, `accuracy` of the current iteration of the kernel is de-

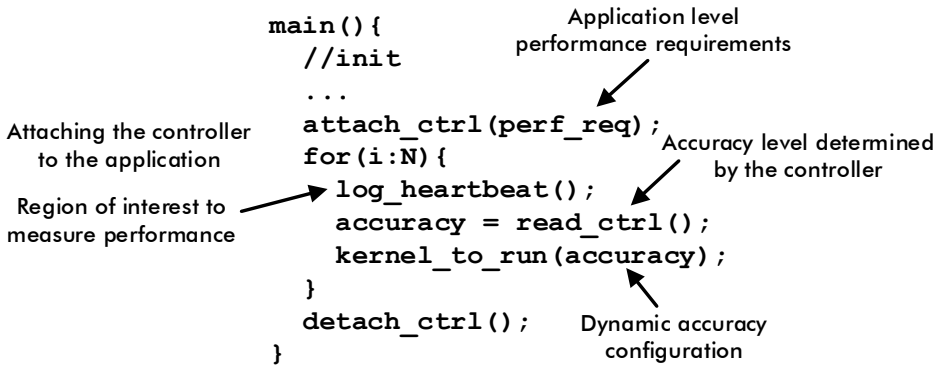



Figure 6.6: Annotated application model for dynamic approximation.

termined by the controller and is read by the application (`read_ctrl`) before executing the main compute kernel. The configured accuracy is parsed within the `kernel_to_run` to dynamically generate an approximate version with desired level of accuracy (here, as per the accuracy level determined by the controller). Our approach thus explores and benefits from fine-grained levels of dynamic approximation, rather than being confined to fixed levels. Decisions of the run-time controller are driven by the *Policy* that governs resource allocation. We design and implement the policy for coordinated and efficient resource allocation, which is described in the following sections. The overall structure of this framework is similar to the ones used in [Muthukaruppan et al., 2013b, Tan et al., 2015, Pathania et al., 2014] - however, the key distinctive aspect of our proposed system is using the APPX knob set by the controller to determine the accuracy levels of the applications at run-time. It should be noted that the previous chapters have used PID controllers, given the fine-grained levels of power actuation in a simulation environment. In the context of this work though, we use a rule-based controller - considering coarse grained power actuation decisions that are driven by the operating system. Design and implementation of the resource allocation policy are described in the following section.

### 6.2.2 Proposed Policy

The run-time resource management policy has been designed similar to a finite state machine as shown in Figure 6.7. The policy is invoked over every *epoch* - a configurable short-term time interval suitable to make resource allocation decisions. An overview of each individual phase is presented in the following subsections and the details on the decision strategy are further elaborated.

**Idle.** This phase corresponds to the unloaded system or to a thoroughly balanced one, satisfying power and performance requirements, eventually. In this phase, the controller monitors relevant events causing significant variations in system dy-

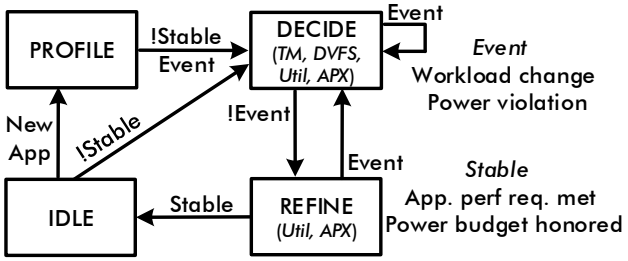



Figure 6.7: Overview of proposed policy.

namics. Such an event typically occurs when a new application enters the system, requesting system resources, or a currently running application leaves the system, freeing the resources. In such events resource allocation decisions are to made, by transitioning into the *decide* phase.

**Decide.** In this phase, resource allocation decisions on core selection, DVFS, CPU quota, task migration and approximation are made. When a new application enters the system, it is conservatively mapped on a LITTLE core; moreover, each time an application enters/exits, the running workload in each cluster is re-mapped to make utilization uniform on the various cores. Actuation decisions are based on power and throughput measurements from the OS-level and application-level monitoring interfaces. Knob settings are determined to be proportional to applications' performance requirement and amount of available power headroom. Failing to meet an application's performance requirement or violating the upper bound on power budget are two key instances when decision making becomes critical. Upon such events, the *decide* phase makes the choice between combinations of power knobs alone (DVFS, CPU quota and/or task migration) or power knobs combined with approximation, subject to whichever yields better performance within acceptable power constraints and accuracy loss (the logical flow of this strategy is presented in a more detailed way in the next subsection). We use estimation models (detailed in Section 6.2.2) to predict near-optimal knob settings during the *decide* phase. They minimize the knob setting space to be pruned, allowing a faster arrival at a near-optimal solution. Further, estimating performance loss/gain over every power actuation decision allows to establish coordination among conflicting power-approximation knobs settings. If performance is not met or the power budget is violated, this strategy will also act on approximation. At the end, selected resource allocation decisions are enforced. If task migration has been performed, this significantly alters application's performance thus the *decide* phase is repeated; otherwise the subsequent *refine* phase will take place.

**Refine.** The *refine* phase monitors power and performance metrics and fine-tunes any coarse-gained knob settings enforced during the *decide* phase. This provides precise control over resource allocation, primarily using CPU quota and APPX knobs. Further, any possible aberrations, or over/under compensation decisions

made during the *decide* phase, will be converged towards required knob settings in the subsequent *refine* phase. We iteratively refine the decisions made in the decide phase, if necessary, by using the feedback loop provided by the power and performance measures such that a stable resource allocation is eventually reached. The *refine* phase will last as long as needed to ensure the performance target, for all running applications, is met within the power budget. The policy's decisions can be classified into two relevant conditions viz., power violation and power budget being honored. The decision strategy in either cases are explained as follows.

**Power Violation.** Recovery actions have to be taken by acting mainly on the big cluster since it is responsible for significant power consumption. Among the current workload, application(s) that have a minimal performance penalty (set to $< 10\%$) upon migrating to the LITTLE cluster are chosen. Each selected application is then migrated from the big cluster to the LITTLE one. Approximation is invoked simultaneously, by setting an accuracy level that is proportional to the estimated loss in performance with task migration. This ensures that a potential performance loss with migrating to the low-power LITTLE cluster is pro-actively addressed with reducing the workload through approximation. If no application with tolerable performance loss upon task migration is available, DVFS is used for reducing power consumption.. Target VF levels are determined by the ratio of current power consumption and budget $P/TDP$, using the estimation model in Equation 6.1. For each application, the possible performance loss with lowered VF levels is estimated with the model in Equation 6.4. Approximation is triggered over such applications that could suffer performance loss with the new VF levels. The level of approximation is set proportional to the estimated performance loss. In this scenario, DVFS and APPX are triggered coordinately such that any possible performance loss incurred in power actuation is pro-actively and proportionally covered up through approximation, using the estimation models.

**Power Budget Honored.** In this scenario, we analyze each cluster separately. We start on the LITTLE and first identify possible applications that do not meet target performance at the highest resources. If any, such applications are migrated on the big cluster and the current decision phase ends and it will be restarted on the subsequent control period to have an updated measure of the throughput in the new configuration. After that, on each cluster the application having the highest assigned CPU quota is chosen as the most demanding. The VF level is then determined such that performance requirements of this selected application are met at maximum CPU quota. As presented in [Muthukaruppan et al., 2013b, Pathania et al., 2014], selecting the minimum necessary VF level at maximum CPU quota minimizes power consumption among all configurations offering the same level of performance. Performance of other applications is estimated with the newly determined VF level. The applications which have met the performance requirements and were approximated in the previous decision phases are recovered back to accuracy, subjective and proportional to the availability of surplus power budget and current performance levels. Among the other applications, CPU quota of appli-

cations that have already met the performance requirements (if any) are lowered proportionally within their performance requirements. For applications that do not meet the performance requirements with new VF settings, CPU quota is increased proportional to the target throughput if possible. In case of persistent performance violation, approximation is invoked, setting the level of approximation proportional to the loss in performance, as estimated.

### Power and Performance Estimates

We have employed performance and power estimation models adapted from literature (e.g. [Pathania et al., 2014, Rexha et al., 2017, Tan et al., 2015]) and experimentally verified on the considered board, Odroid XU-3 [Hardkernel co., 2014]. We represent power consumption $P_j$ of a cluster $j$ running a set of $N$ applications $App_i$ as a function of the overall cluster utilization $U_j$ and the related $V_j, F_j$. In particular, as shown in [Rexha et al., 2017,Tan et al., 2015,Pathania et al., 2014] for the considered architecture for each $V_j, F_j$ pair, a fast/almost-accurate estimation of power can be obtained as a function of $U_j$:

$$P_{V_j,F_j} = a_{V_j,F_j} \cdot U_j + b_{V_j,F_j} \tag{6.1}$$

where $a_{V_j,F_j}$ and $b_{V_j,F_j}$ are empirically derived at each $V_j/F_j$ pair. Utilization $U_j$ of a cluster $j$ indicates the amount of time in a given control period for which the included cores are busy:

$$U_j = \sum_{i=0}^{N} U_i \tag{6.2}$$

where $N$ is the number of cores in the cluster. Moreover, $U_i$ measure is directly provided by the OS, whereas the assigned CPU quota is directly proportional to the utilization. Then, power consumption $P_t$ of the system is the sum of values of the two clusters. As shown in previous works (e.g. [Pathania et al., 2014,Tan et al., 2015]), performance ($Perf_i$) of an application $i$ has an almost linear relationship with the CPU quota $Q$, VF setting of the cluster it is mapped on, expressed as:

$$Perf(i) = \alpha Q(App_i) F_i \tag{6.3}$$

where $\alpha$ is a variable parameter that depends on application characteristics and cluster where the application is running. Thus, we estimate the performance penalty with actuation of power knobs viz., CPU quota assignment, VF level and task migration from a configuration $old$ to the $new$ one as:

$$Penalty(App_i) = \frac{Perf_{new}}{Perf_{old}} = \frac{\alpha_{new} Q_{new}(App_i) F_{new}}{\alpha_{old} Q_{old}(App_i) F_{old}} \tag{6.4}$$

## 6.3 Evaluation

**Experimental Setup.** For the experimental evaluation, we considered an Odroid XU3 board running Linux Ubuntu 15.04. The board hosts a Samsung Exynos 5422 device featuring 4 ARM A7 (LITTLE) and 4 ARM A15 (big) with frequencies spanning between $200-1400$MHz and $200-2000$MHz, respectively. Setting a specific frequency level though OS drivers automatically scales corresponding voltage level. TDP was set to $5W$, considering a restriction on-chip temperature to 80°C. The controller has been implemented in C++ and runs as a user space process. It uses Linux drivers to access on-board sensors for power measures and to apply DVFS settings. Application's execution and allocation of CPU quota are enabled through CGroups Linux library. Communication between applications and the controller is established through Linux shared memory mechanism. An in-house implementation of HeartBeats API uses this shared memory space to write control signals related to setting approximation level and to read application's throughput. This provides a fine-grained control on levels of approximation with a negligible overhead on switching the mode of execution. Further, the approximation level is passed only as parameter to the kernel function, dynamically creating a range of approximate versions without any compilation and memory overhead. Finally, the epoch for the controller invocation is parameterizable, and in our experiments we set it to $1s$ to clearly see the effects of knobs actuation on throughput of selected applications during the immediate subsequent controller's invocation. This epoch may be tuned subject to the platform and types of considered applications. We chose four applications from machine learning domain due to their error resilient nature of computation. Table 6.1 shows application characteristics and used types of approximation.

**Results.** To test our proposed solution, we created a dynamic workload scenario ranging from 1 to 4 concurrently running applications, emulating unknown workloads. We compare it against similar state-of-the-art approaches, namely ApxKnob [Kanduri et al., 2016] and HierCtrl [Muthukaruppan et al., 2013b]. These approaches exploit the uncoordinated usage of the APPX knob with other power knobs, and the coordinated usage of power knobs without any approximation, respectively. All systems are fed with the same dynamic workload scenarios. Figure 6.9 shows a comparison of power consumption, and Figure 6.8 reports each application's performance against the set requirements over the workloads (with

Table 6.1: Simulated workload.

| Application | Input | Approximation |
|---|---|---|
| LeastSq | 1 million pairs | loop perforation |
| KNN | 25k points and 25 test cases | loop perforation and task skipping |
| kMeans | 50k points into 3 clusters | relaxed convergence |
| LinReg | 1 million pairs | relaxed convergence |

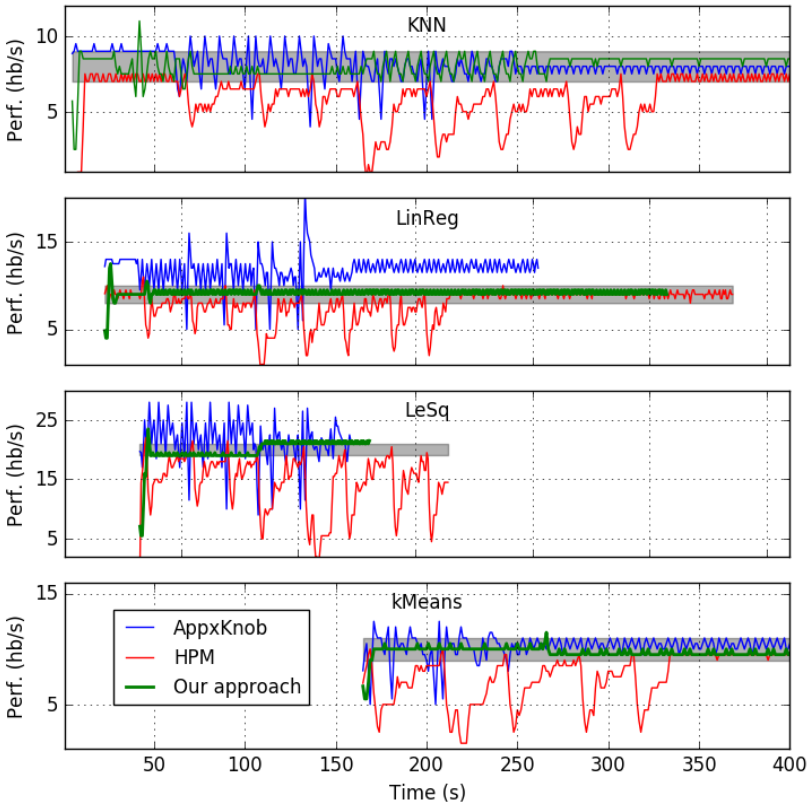



Figure 6.8: Run-time performance of applications (hb/s), in order of their arrival, with different policies - AppxKnob [Kanduri et al., 2016], HMP [Muthukaruppan et al., 2013b] and the proposed approach. The shaded grey region shows (an acceptable range of) performance requirements expressed by applications.

a $-1/+1hb$ tolerance band). Table 6.2 lists the average power consumption and percentage of power violations for each approach. ApxKnob violates power budget the most, at 35% of the overall execution period, mainly with conflicting power-performance decisions resulting in constant oscillation between high-low power-performance states. HierCtrl has a much lower power violation at 3.5%, as different controllers converge eventually towards stable configurations. Our solution has a negligible 0.5% power violation with pro-active and coordinated knob actuation, benefiting from the estimation models. Table 6.2 also reports for every approach, the performance guarantees in terms of the percentage of execution time during which the requested performance is achieved for each application. The loss in accuracy with approximation knob is presented as the weighted sum of execution time and approximation level set, also shown in Table 6.2. ApxKnob meets the performance requirements mostly by leveraging approximation, and largely due to uninhibited power usage. Although the loss in accuracy for performance

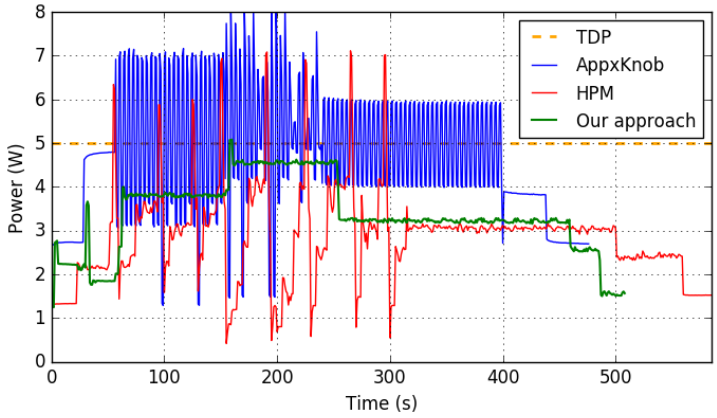



Figure 6.9: Run-time power consumption (W) with different policies viz., AppxKnob [Kanduri et al., 2016], HMP [Muthukaruppan et al., 2013b] and the proposed approach. TDP is set to 5W.

guarantees is acceptable, this approach is limited by frequent thermal violations. HierCtrl relies on degrading throughput requirements to address power emergencies, resulting in dramatic performance loss, particularly under heavy workloads. Although HierCtrl achieves the lowest average power consumption and has a lower power violation rate, performance losses are significant. Our proposed approach, on the other hand, balances both performance requirements and power constraints, in particular, it is effective in delivering the requested performance, within minimal power consumption and an acceptable loss of accuracy. The effectiveness of approximation knob in power management although is trivial, requires disciplined tuning to achieve performance gains within power limits. Figure 6.9 shows that ApxKnob over compensates for performance (resulting in power budget violation), while HierCtrl violates performance requirements (with conservative power actuation). Our approach tunes approximation effectively to obtain the requested performance while minimizing the necessary power consumption. This can also be observed in Figure 6.8, where our approach consistently provides precisely the required performance, while ApxKnob and HierCtrl often over/under provision, resulting in their respective power and performance violations.

Table 6.2: Power, performance and approximation behavior.

| **Technique** | **Avg. Power (W)** | **Power Violation (%)** | **Perf. Violation (%)** | | | | **Approximation (%)** | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | **LR** | **LeSq** | **KNN** | **kM** | **LR** | **LeSq** | **KNN** | **kM** |
| ApxKnob [Kanduri et al., 2016] | 4.6 | 35.8 | 1.6 | 15.3 | 3.5 | 4.5 | 0 | 2 | 5.3 | 2.6 |
| HPM [Muthukaruppan et al., 2013b] | 2.82 | 3.5 | 28.6 | 89.7 | 44.2 | 42.6 | 0 | 0 | 0 | 0 |
| **Our Appr.** | 3.34 | 0.58 | 0.62 | 3.04 | 1.85 | 1.3 | 0.1 | 5 | 6.4 | 3.9 |

**Summary**

This chapter extends the ideas presented in the previous chapters to utilize approximation in the context of power/performance management on heterogeneous systems. We designed and implemented a resource management strategy that coordinates between approximation and other traditional power knobs for guaranteeing performance requirements within minimal power consumption. The proposed policy is implemented and evaluated on real hardware testbed, and compared against state-of-the-art run-time management frameworks to demonstrate the efficiency of coordinated approximation. The contents presented in this chapter are based on original publication, attached as Paper V.

# Chapter 7

# Conclusion

Major motivations for this research, particularly when put together are - i) the *increase in performance demands* of emerging applications from domains such as machine learning, artificial intelligence and internet-of-things etc., coupled with ii) *diminishing performance gains* from technology node scaling due to increase in power densities.Architectural solutions could alleviate the challenge of maximizing performance within fixed power budgets, particularly with heterogeneity and domain specific acceleration. However, considering dynamic scenarios such as workload variation, diversity in application requirements and characteristics, run-time management techniques can maximize the resource efficiency and take better advantage of underlying hardware. This dissertation thus focuses on run-time techniques to manage system resources efficiently in terms of i) ensuring thermal safety, ii) honoring power budgets, while iii) improving performance and/or guaranteeing the required performance and iv) minimizing power consumption and/or utilize available power budget efficiently.

## 7.1 Thesis Summary

Chapter 1 introduces the context of the research pursued in this dissertation, motivation, significance and relevance of the problem. This is followed by the mentions on contributions, research outcomes and the organization of this dissertation.

Chapter 2 provides background on run-time resource management, existing strategies for power and performance actuation, traditional power knobs, their impacts on performance and the need for balancing application requirements and system dynamics simultaneously.

Chapter 3 focuses on power budgeting, discussing aspects of fixed versus variable power budgeting techniques and presents the effect of application mapping on power budgets in the context of many-core systems. This chapter introduces dark silicon patterning - aligning inevitable dark cores alongside active cores to minimize temperature accumulation among neighboring cores. As a result, active cores

reach the critical temperature after utilizing a higher amount of power budget, in comparison with tightly packed mapping strategies. While application mapping strategies are already considered to be relevant in run-time management, this part of the dissertation adapts existing approaches for the dark silicon scenario to improve power budgets.

Chapter 4 targets improving performance within thermally safe limits by efficiently utilizing available power budget and thermal headroom. It extends upon the insights from dark silicon patterning from Chapter 3, which provides surplus power budgets and/or lower on-chip temperatures. This chapter presents performance boosting techniques leveraging power and thermal headroom created through dark silicon patterning, using appropriate power and thermal feedback controllers. While performance boosting techniques such as computational sprinting, core unfolding and selective frequency scaling do exist, this dissertation adapts the combination of efficient power budgeting followed by necessary boosting to maximize performance gains.

Chapter 5 addresses the challenge of maximizing performance under power cap, considering application's error resilience characteristics. It introduces the idea of using accuracy-performance trade-offs dynamically subject to applications' performance requirements and system constraints of power budget. The proposed approach uses approximation as a dynamic knob which can be actuated alongside the traditional power knobs, to cover up for the performance loss that might incur in power actuation. While there are a range of software and hardware approximation techniques for performance and energy gains, this strategy emphasizes on adapting those techniques to make justifiable and opportunistic decisions on accuracy trade-offs. Further, applications' sensitivity to error are considered to minimize the accuracy loss upon triggering approximate mode of execution.

Chapter 6 extends the idea of using approximation to manage both power and performance simultaneously, specifically on heterogeneous platforms. It highlights the need for coordination among power and performance allocations, and presents models for estimating the effect of resource scaling decisions on power and performance mutually. The proposed approach explores both power-performance and accuracy-performance Pareto-spaces to make decisions that ensures performance guarantees, while minimizing power consumption and accuracy loss. While most of the existing approximation techniques rely on static configurations, the techniques presented in this chapter dynamically configure software approximations to either gain/ensure performance and/or lower resources on top of reduced workloads to minimize power consumption - making them particularly relevant in the context of run-time resource management.

## 7.2 Open Directions

This dissertation attempts to address the problem of maximizing performance of applications while minimizing power consumption and ensuring thermal safety through run-time management techniques.

One of the main challenges in this approach is understanding and adapting to dynamic characteristics of applications. Different workload characteristics have different performance requirements (latency sensitivity vs. throughput), exhibit diverse phase behaviors (compute intensive vs. memory) and a range of error resilience. Expressing such application level requirements in a unified manner, preferably through programming constructs or application level APIs can provide insights into appropriate application requirements. Such an expression enables translating the requirements into quantifiable amount of resources, easing the allocation phases of management policies. One of the open research directions is to improve the understanding and expression of application level characteristics and requirements, which could be efficiently used vertically across the computing stack.

Another challenge lies in comprehending the power-performance-accuracy trade-offs of a given application on a target hardware platform. Similar to different applications, heterogeneous hardware provides different power-performance options for execution. Understanding this behavior and choosing the appropriate type of execution units is non-trivial. Although several approaches to profile, analyze and learn application-core bias exist, a formal methodology to match applications to hardware and determine the amount of resources from the off-set can further improve the resource efficiency. Another open research direction is for a formalized design methodology for identifying application-core bias and thus enable on-the-fly match between applications and underlying hardware, and application requirements and amount of resources to be allocated.

Considering concurrent workloads and the way each application utilizes available resources, the subsequent power-performance metrics vary significantly. Understanding this behavior is increasingly complicated even with thorough off-line profiling. Yet, analyzing such behavior of resource utilization of any given application(s) under concurrent workloads will provide hints on efficacy of run-time management strategies. On-line prediction and estimation techniques, control theoretic and machine learning models for run-time resource management offers lot of promise in this context. An interesting research direction is to further extend online learning strategies to provide a fully autonomous resource allocation strategy, highlighting the notion of self-awareness in computer systems.

# Bibliography


[Big, 2011] (2011). ARM Ltd. *http://www.arm.com/products/tools/development-boards/versatile-express/index.php*.

[pac, 2011] (2011). Pack & cap: adaptive dvfs and thread packing under power caps. In *MICRO*, pages 175–185.

[TGG, 2017] (2017). TGG: Task Graph Generator. `http://sourceforge.net/projects/taskgraphgen/`. Last update: 2015-08-02.

[Akram et al., 2016] Akram, S., Sartor, J. B., and Eeckhout, L. (2016). Dvfs performance prediction for managed multithreaded applications. In *Performance Analysis of Systems and Software (ISPASS), 2016 IEEE International Symposium on*, pages 12–23. IEEE.

[Akram et al., 2017] Akram, S., Sartor, J. B., and Eeckhout, L. (2017). Dep+burst: Online dvfs performance prediction for energy-efficient managed language execution. *IEEE Trans. Comput.*, 66(4):601–615.

[Ansel et al., 2009] Ansel, J. et al. (2009). PetaBricks: a language and compiler for algorithmic choice. *ACM SIGPLAN Notices*.

[Arden et al., 2010] Arden, W., Brillouët, M., Cogez, P., Graef, M., Huizing, B., and Mahnkopf, R. (2010). More-than-moore white paper. *Version*, 2:14.

[Arora et al., 2015] Arora, M., Manne, S., Paul, I., Jayasena, N., and Tullsen, D. M. (2015). Understanding idle behavior and power gating mechanisms in the context of modern benchmarks on cpu-gpu integrated systems. In *High Performance Computer Architecture (HPCA), 2015 IEEE 21st International Symposium on*, pages 366–377. IEEE.

[Azimi et al., 2017] Azimi, I., Anzanpour, A., Rahmani, A. M., Liljeberg, P., and Tenhunen, H. (2017). Self-aware early warning score system for iot-based personalized healthcare. In *eHealth 360Âř*, pages 49–55. Springer.

[Baek and Chilimbi, 2010] Baek, W. and Chilimbi, T. M. (2010). Green: A Framework for Supporting Energy-Conscious Programming using Controlled Approximation. In *PLDI*, pages 198–209.

[Baek et al., 2010] Baek, W. et al. (2010). Green: A Framework for Supporting Energy-Conscious Programming using Controlled Approximation. In *PLDI*.

[Bhadauria et al., 2009] Bhadauria, M., Weaver, V. M., and McKee, S. A. (2009). Understanding PARSEC performance on contemporary CMPs. In *Proc. of the Int. Symp. on Workload Characterization (IISWC)*, pages 98–107, Washington, DC, USA.

[Bornholt et al., 2014] Bornholt, J., Mytkowicz, T., and McKinley, K. S. (2014). Uncertain<T>: A First-Order Type for Uncertain Data. pages 51–66.

[Chaudhuri et al., 2011] Chaudhuri, S., Gulwani, S., Lublinerman, R., and Navidpour, S. (2011). Proving programs robust. pages 102–112.

[Cochran et al., 2011] Cochran, R. et al. (2011). Pack & cap: adaptive dvfs and thread packing under power caps. In *MICRO*.

[Conoci et al., 2018] Conoci, S., Di Sanzo, P., Ciciani, B., and Quaglia, F. (2018). Adaptive performance optimization under power constraint in multi-thread applications with diverse scalability. In *Proceedings of the 2018 ACM/SPEC International Conference on Performance Engineering*, ICPE '18, pages 16–27, New York, NY, USA. ACM.

[de Souza Carvalho et al., 2010] de Souza Carvalho, E. L., Calazans, N. L. V., and Moraes, F. G. (2010). Dynamic task mapping for MPSoCs. *IEEE Design & Test of Computers*, 27(5):26–35.

[Dennard et al., 1974] Dennard, R., Gaensslen, F., Rideout, V., Bassous, E., and LeBlanc, A. (1974). Design of ion-implanted MOSFET's with very small physical dimensions. *IEEE Journal of Solid-State Circuits*, 9(5):256–268.

[Ding et al., 2014] Ding, Y., Yedlapalli, P., and Kandemir, M. (2014). Qos aware dynamic time-slice tuning. In *Workload Characterization (IISWC), 2014 IEEE International Symposium on*, pages 84–85. IEEE.

[Du et al., 2015] Du, Z., Fasthuber, R., Chen, T., Ienne, P., Li, L., Feng, X., Chen, Y., and Temam, O. (2015). ShiDianNao : Shifting Vision Processing Closer to the Sensor. ISCA '15.

[Esmaeilzadeh, 2015] Esmaeilzadeh, H. (2015). Approximate acceleration: A path through the era of dark silicon and big data. In *Proceedings of the 2015 International Conference on Compilers, Architecture and Synthesis for Embedded Systems*, CASES '15, pages 31–32, Piscataway, NJ, USA. IEEE Press.

[Esmaeilzadeh et al., 2012a] Esmaeilzadeh, H., Blem, E., St. Amant, R., Sankaralingam, K., and Burger, D. (2012a). Dark Silicon and the End of Multicore Scaling. *IEEE Micro*, 32(3):122–134.

[Esmaeilzadeh et al., 2012b] Esmaeilzadeh, H. et al. (2012b). Neural Acceleration for General-Purpose Approximate Programs. In *MICRO*.

[Esmaeilzadeh et al., 2012c] Esmaeilzadeh, H., Sampson, A., Ceze, L., and Burger, D. (2012c). Architecture support for disciplined approximate programming. volume 40, page 301.

[Fattah et al., 2013] Fattah, M., Daneshtalab, M., Liljeberg, P., and Plosila, J. (2013). Smart hill climbing for agile dynamic mapping in many-core systems. In *Proc. Design Automation Conf. (DAC)*, pages 1–6.

[Fattah et al., 2012a] Fattah, M. et al. (2012a). Cona: Dynamic application mapping for congestion reduction in many-core systems. In *Proc. of ICCD*, pages 364–370.

[Fattah et al., 2014] Fattah, M., Liljeberg, P., Plosila, J., and Tenhunen, H. (2014). Adjustable contiguity of run-time task allocation in networked many-core systems. In *Proc. Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 349–354.

[Fattah et al., 2012b] Fattah, M., Ramirez, M., Daneshtalab, M., Liljeberg, P., and Plosila, J. (2012b). CoNA: Dynamic application mapping for congestion reduction in many-core systems. In *Proc. Int. Conf. on Computer Design (ICCD)*, pages 364–370.

[Fazzino et al., 2008] Fazzino, F., Palesi, M., and Patti, D. (2008). Noxim: Network-on-chip simulator. *URL: http://sourceforge.net/projects/noxim*.

[Gaspar et al., 2016] Gaspar, F. et al. (2016). A framework for application-guided task management on heterogeneous embedded systems. *ACM TACO*, 12(4):42.

[Gaspar et al., 2014] Gaspar, F., Ilic, A., Tomás, P., and Sousa, L. (2014). Performance-aware task management and frequency scaling in embedded systems. In *Computer Architecture and High Performance Computing (SBAC-PAD), 2014 IEEE 26th International Symposium on*, pages 65–72. IEEE.

[Gaspar et al., 2015] Gaspar, F., Taniça, L., Tomás, P., Ilic, A., and Sousa, L. (2015). A framework for application-guided task management on heterogeneous embedded systems. *ACM Trans. Archit. Code Optim.*, 12(4):42:1–42:25.

[Gia et al., 2018] Gia, T. N., Sarker, V. K., Tcarenko, I., Rahmani, A. M., Westerlund, T., Liljeberg, P., and Tenhunen, H. (2018). Energy efficient wearable sensor node for iot-based fall detection systems. *Microprocessors and Microsystems*, 56:34–46.

[Goulding-Hotta et al., 2011] Goulding-Hotta, N., Sampson, J., Venkatesh, G., Garcia, S., Auricchio, J., Huang, P., Arora, M., Nath, S., Bhatt, V., Babb, J.,

Swanson, S., and Taylor, M. (2011). The GreenDroid Mobile Application Processor: An Architecture for Silicon's Dark Future. *IEEE Micro*, 31(2):86–95.

[Gupta et al., 2013a] Gupta, P., Agarwal, Y., Dolecek, L., Dutt, N., Gupta, R. K., Kumar, R., Mitra, S., Nicolau, A., Rosing, T. S., Srivastava, M. B., et al. (2013a). Underdesigned and opportunistic computing in presence of hardware variability. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 32(1):8–23.

[Gupta et al., 2011] Gupta, V., Mohapatra, D., Park, S. P., Raghunathan, A., and Roy, K. (2011). IMPACT: IMPrecise adders for low-power approximate computing. *Proceedings of the International Symposium on Low Power Electronics and Design*, pages 409–414.

[Gupta et al., 2013b] Gupta, V., Mohapatra, D., Raghunathan, A., and Roy, K. (2013b). Low-power digital signal processing using approximate adders. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 32(1):124–137.

[Haghbayan et al., 2017a] Haghbayan, M.-H. et al. (2017a). Performance/reliability-aware resource management for many-cores in dark silicon era. *IEEE Tran. on Computers*.

[Haghbayan et al., 2015] Haghbayan, M.-H., Kanduri, A., Rahmani, A.-M., Liljeberg, P., Jantsch, A., and Tenhunen, H. (2015). MapPro: Proactive Runtime Mapping for Dynamic Workloads by Quantifying Ripple Effect of Applications on Networks-on-Chip. In *Int. Symp. on Networks-on-Chip (NOCS)*, pages 1–8.

[Haghbayan et al., 2017b] Haghbayan, M.-H., Miele, A., Rahmani, A. M., Liljeberg, P., and Tenhunen, H. (2017b). Performance/reliability-aware resource management for many-cores in dark silicon era. *IEEE Transactions on Computers*, 66(9):1599–1612.

[Haghbayan et al., 2017c] Haghbayan, M.-H., Rahmani, A. M., Liljeberg, P., Jantsch, A., Miele, A., Bolchini, C., and Tenhunen, H. (2017c). Can dark silicon be exploited to prolong system lifetime? *IEEE Design & Test*, 34(2):51–59.

[Haghbayan et al., 2014] Haghbayan, M.-H., Rahmani, A.-M., Weldezion, A., Liljeberg, P., Plosila, J., Jantsch, A., and Tenhunen, H. (2014). Dark silicon aware power management for manycore systems under dynamic workloads. In *Proc. Int. Conf. on Computer Design (ICCD)*, pages 509–512.

[Hardkernel co., 2014] Hardkernel co. (2014). Odroid XU3. `http://www.hardkernel.com`.

[Hegde and Shanbhag, 1999] Hegde, R. and Shanbhag, N. (1999). Energy-efficient signal processing via algorithmic noise-tolerance.

[Hennessy and Patterson, 2012] Hennessy, J. and Patterson, D. (2012). Computer architecture, fifth edition: A quantitative approach (the morgan kaufmann series in computer architecture and design). *Elsevier*.

[Hindman et al., 2011] Hindman, B., Konwinski, A., Zaharia, M., Ghodsi, A., Joseph, A. D., Katz, R. H., Shenker, S., and Stoica, I. (2011). Mesos: A platform for fine-grained resource sharing in the data center. In *NSDI*, volume 11, pages 22–22.

[Hoffmann et al., 2010] Hoffmann, H. et al. (2010). Application heartbeats: a generic interface for specifying program performance and goals in autonomous computing environments. In *Proc. Int. Conf. on Autonomic computing*, pages 79–88. ACM.

[Hoffmann et al., 2012] Hoffmann, H. et al. (2012). Dynamic knobs for responsive power-aware computing. *ACM SIGPLAN Notices*.

[Hu et al., 2004] Hu, Z., Buyuktosunoglu, A., Srinivasan, V., Zyuban, V., Jacobson, H., and Bose, P. (2004). Microarchitectural techniques for power gating of execution units. In *Proceedings of the 2004 International Symposium on Low Power Electronics and Design*, ISLPED '04, pages 32–37, New York, NY, USA. ACM.

[Huang et al., 2006] Huang, W. et al. (2006). Hotspot: A compact thermal modeling methodology for early-stage vlsi design. *IEEE Tran. on VLSI Systems*, 14(5):501–513.

[Intel Corporation, 2011] Intel Corporation (2011). Intel Corporation. Intel Xeon Processor - Measuring Processor Power, revision 1.1. In *White paper, Intel Corporation*.

[Intel Corporation, 2014] Intel Corporation (2014). Intel Corporation. Fourth Generation Mobile Processor Family Data Sheet. In *White paper, Intel Corporation*.

[Joao et al., 2013] Joao, J. A., Suleman, M. A., Mutlu, O., and Patt, Y. N. (2013). Utility-based acceleration of multithreaded applications on asymmetric cmps. In *Proceedings of the 40th Annual International Symposium on Computer Architecture (ISCA)*, pages 154–165.

[Jouppi et al., 2017] Jouppi, N. P. et al. (2017). In-datacenter performance analysis of a tensor processing unit. *SIGARCH Comput. Archit. News*, 45(2):1–12.

[Kahng and Kang, 2012] Kahng, A. B. and Kang, S. (2012). Accuracy-configurable adder for approximate arithmetic designs. *Proceedings of the 49th Annual Design Automation Conference (DAC)*, page 820.

[Kanduri et al., 2015a] Kanduri, A. et al. (2015a). Dark silicon aware runtime mapping for many-core systems: A patterning approach. In *Proc. of ICCD*, pages 573–580.

[Kanduri et al., 2015b] Kanduri, A. et al. (2015b). Dark silicon aware runtime mapping for many-core systems: A patterning approach. In *ICCD*.

[Kanduri et al., 2016] Kanduri, A. et al. (2016). Approximation knob: Power capping meets energy efficiency. In *In Proc. of ICCAD*, pages 1–8. IEEE.

[Kanduri et al., 2018] Kanduri, A., Miele, A., Rahmani, A. M., Liljeberg, P., Bolchini, C., and Dutt, N. (2018). Approximation-aware coordinated power/performance management for heterogeneous multi-cores. In *Proceedings of the 55th Annual Design Automation Conference*, page 68. ACM.

[Kapadia and Pasricha, 2015] Kapadia, N. and Pasricha, S. (2015). Varsha: Variation and reliability-aware application scheduling with adaptive parallelism in the dark-silicon era. In *DATE*, pages 1060–1065.

[Kedem et al., 2011] Kedem, Z. M., Mooney, V. J., Muntimadugu, K. K., and Palem, K. V. (2011). An approach to energy-error tradeoffs in approximate ripple carry adders. *Proceedings of the International Symposium on Low Power Electronics and Design*, pages 211–216.

[Khdr et al., 2017] Khdr, H., Pagani, S., Sousa, E., Lari, V., Pathania, A., Hannig, F., Shafique, M., Teich, J., and Henkel, J. (2017). Power density-aware resource management for heterogeneous tiled multicores. *IEEE Transactions on Computers*, (3):488–501.

[Kim and Shanbhag, 2014] Kim, E. P. and Shanbhag, N. R. (2014). Energy-Efficient Accelerator Architecture for Stereo Image Matching Using Approximate Computing and Statistical Error Compensation. *Signal and Information Processing (GlobalSIP), 2014 IEEE Global Conference on*.

[Kim et al., 2008] Kim, W., Gupta, M. S., Wei, G.-Y., and Brooks, D. (2008). System level analysis of fast, per-core dvfs using on-chip switching regulators. In *Proc. of HPCA*, pages 123–134.

[Kondo et al., 2014] Kondo, M., Kobyashi, H., Sakamoto, R., Wada, M., Tsukamoto, J., Namiki, M., Wang, W., Amano, H., Matsunaga, K., Kudo, M., et al. (2014). Design and evaluation of fine-grained power-gating for embedded microprocessors. In *Design, Automation and Test in Europe Conference and Exhibition (DATE), 2014*, pages 1–6. IEEE.

[Li et al., 2009] Li, S., Ahn, J. H., Strong, R., Brockman, J., Tullsen, D., and Jouppi, N. (2009). McPAT: An integrated power, area, and timing modeling

framework for multicore and manycore architectures. In *Proc. Int. Symp. on Microarchitecture (MICRO)*, pages 469–480.

[Li and Yeung, 2006] Li, X. and Yeung, D. (2006). Exploiting Soft Computing for Increased Fault Tolerance. *Workshop on Architectural Support for Gigascale Integration*, (June).

[Li and Yeung, 2007] Li, X. and Yeung, D. (2007). Application-level correctness and its impact on fault tolerance. *Proceedings - International Symposium on High-Performance Computer Architecture*, pages 181–192.

[Lindholm et al., 2008] Lindholm, E., Nickolls, J., Oberman, S., and Montrym, J. (2008). Nvidia tesla: A unified graphics and computing architecture. *IEEE micro*, 28(2).

[Lingamneni et al., 2013] Lingamneni, Basu, A., Enz, Palem, and Piguet (2013). Improving energy gains of inexact DSP hardware through reciprocative error compensation. *Dac '13*, page 20.

[Liu et al., 2014] Liu, Y., Draper, S. C., and Kim, N. S. (2014). Sleepscale: run-time joint speed scaling and sleep states management for power efficient data centers. In *Computer Architecture (ISCA), 2014 ACM/IEEE 41st International Symposium on*, pages 313–324. IEEE.

[Lo et al., 2014] Lo, D., Cheng, L., Govindaraju, R., Barroso, L. A., and Kozyrakis, C. (2014). Towards energy proportionality for large-scale latency-critical workloads. In *Proceeding of the 41st Annual International Symposium on Computer Architecuture (ISCA)*, pages 301–312.

[Lo et al., 2015] Lo, D., Cheng, L., Govindaraju, R., Ranganathan, P., and Kozyrakis, C. (2015). Heracles: Improving resource efficiency at scale. In *Proceedings of the 42Nd Annual International Symposium on Computer Architecture (ISCA)*, pages 450–462.

[Lukefahr et al., 2014] Lukefahr, A., Padmanabha, S., Das, R., Dreslinski, Jr., R., Wenisch, T. F., and Mahlke, S. (2014). Heterogeneous microarchitectures trump voltage scaling for low-power cores. In *Proceedings of the 23rd International Conference on Parallel Architectures and Compilation*, PACT '14, pages 237–250, New York, NY, USA. ACM.

[Lungu et al., 2009] Lungu, A., Bose, P., Buyuktosunoglu, A., and Sorin, D. J. (2009). Dynamic power gating with quality guarantees. In *Proceedings of the 2009 ACM/IEEE international symposium on Low power electronics and design*, pages 377–382. ACM.

[Ma and Wang, 2012] Ma, K. and Wang, X. (2012). PGCapping: Exploiting Power Gating for Power Capping and Core Lifetime Balancing in CMPs. In *Proc. Int. Conf. on Parallel Architectures and Compilation Techniques (PACT)*, pages 13–22.

[Madan et al., 2011] Madan, N., Buyuktosunoglu, A., Bose, P., and Annavaram, M. (2011). A case for guarded power gating for multi-core processors. In *Proceedings of the 2011 IEEE 17th International Symposium on High Performance Computer Architecture*, HPCA '11, pages 291–300, Washington, DC, USA. IEEE Computer Society.

[Majumdar et al., 2017] Majumdar, A., Piga, L., Paul, I., Greathouse, J. L., Huang, W., and Albonesi, D. H. (2017). Dynamic gpgpu power management using adaptive model predictive control. In *2017 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 613–624. IEEE.

[Misailovic et al., 2014] Misailovic, S. et al. (2014). Chisel: Reliability- and accuracy-aware optimization of approximate computational kernels. In *OOPSLA*, pages 309–328.

[Mishra et al., 2014] Mishra, A. K., Barik, R., and Paul, S. (2014). iact: A software-hardware framework for understanding the scope of approximate computing. In *Workshop on Approximate Computing Across the System Stack (WACAS)*.

[Mittal, 2016] Mittal, S. (2016). A survey of techniques for approximate computing. *ACM Comput. Surv.*, 48(4):62:1–62:33.

[Moore, 1965] Moore, G. (1965). ramming more components onto integrated circuits. *Electronics, 38(8)*.

[Moreau et al., 2015] Moreau, T., M, W., Nelson, J., Sampson, A., Esmaeilzadeh, H., Ceze, L., and Oskin, M. (2015). Snnap : Approximate computing on programmable socs via neural acceleration.

[Muthukaruppan et al., 2013a] Muthukaruppan, T. et al. (2013a). Hierarchical power management for asymmetric multi-core in dark silicon era. In *DAC*.

[Muthukaruppan et al., 2013b] Muthukaruppan, T., Pricopi, M., Venkataramani, V., Mitra, T., and Vishin, S. (2013b). Hierarchical power management for asymmetric multi-core in dark silicon era. In *Proc. Design Automation Conf. (DAC)*, pages 1–9.

[Nair, 2015] Nair, R. (2015). Big data needs approximate computing: technical perspective. *Communications of the ACM*, 58(1):104–104.

[Naveh et al., 2011] Naveh, A., Rajwan, D., Ananthakrishnan, A., and Weissmann, E. (2011). Power management architecture of the 2nd generation intel® coreâĎć microarchitecture, formerly codenamed sandy bridge. In *Hot Chips*, volume 23, page 0.

[Pagani et al., 2015] Pagani, S. et al. (2015). seboost: Selective boosting for heterogeneous manycores. In *Proc. of CODES+ISSS*, pages 104–113.

[Pagani et al., 2017] Pagani, S. et al. (2017). Thermal safe power (tsp): Efficient power budgeting for heterogeneous manycore systems in dark silicon. *IEEE Transactions on Computers*, 66(1):147–162.

[Pagani et al., 2014] Pagani, S., Khdr, H., Munawar, W., Chen, J.-J., Shafique, M., Li, M., and Henkel, J. (2014). TSP: Thermal Safe Power: Efficient Power Budgeting for Many-core Systems in Dark Silicon. In *Proc. Int. Conf. on Hardware/Software Codesign and System Synthesis (CODES)*, pages 10:1–10:10.

[Palomino et al., 2016] Palomino, D., Shafique, M., Susin, A., and Henkel, J. (2016). Thermal optimization using adaptive approximate computing for video coding. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2016*, pages 1207–1212. IEEE.

[Papazoglou and Georgakopoulos, 2003] Papazoglou, M. P. and Georgakopoulos, D. (2003). Introduction: Service-oriented computing. *Commun. ACM*, pages 24–28.

[Pathania et al., 2014] Pathania, A. et al. (2014). Integrated CPU-GPU power management for 3D mobile games. In *Proc. of DAC*, pages 1–6.

[Petrucci et al., 2015] Petrucci, V., Laurenzano, M. A., Doherty, J., Zhang, Y., Mosse, D., Mars, J., and Tang, L. (2015). Octopus-man: Qos-driven task management for heterogeneous multicores in warehouse-scale computers. In *High Performance Computer Architecture (HPCA), 2015 IEEE 21st International Symposium on*, pages 246–258. IEEE.

[Pricopi et al., 2013] Pricopi, M., Muthukaruppan, T. S., Venkataramani, V., Mitra, T., and Vishin, S. (2013). Power-performance modeling on asymmetric multi-cores. In *Proceedings of the 2013 International Conference on Compilers, Architectures and Synthesis for Embedded Systems*, CASES '13, pages 15:1–15:10, Piscataway, NJ, USA. IEEE Press.

[Raghavan et al., 2012] Raghavan, A. et al. (2012). Computational sprinting. In *Proc. of HPCA*, pages 1–12. IEEE.

[Rahmani et al., 2015a] Rahmani, A. et al. (2015a). Dynamic power management for many-core platforms in the dark silicon era: A multi-objective control approach. In *ISLPED*.

[Rahmani et al., 2016] Rahmani, A., Liljeberg, P., Hemani, A., Jantsch, A., and Tenhunen, H. (2016). *The Dark Side of Silicon*. Springer, 1st edition edition.

[Rahmani et al., 2015b] Rahmani, A.-M. et al. (2015b). Dynamic power management for many-core platforms in the dark silicon era: A multi-objective control approach. In *Proc. of ISLPED*, pages 219–224.

[Rahmani et al., 2017] Rahmani, A. M. et al. (2017). Reliability-aware runtime power management for many-core systems in the dark silicon era. *IEEE Tran. on VLSI Systems*, 25(2):427–440.

[Rahmani et al., 2015c] Rahmani, A.-M., Haghbayan, M.-H., Kanduri, A., Weldezion, A., Liljeberg, P., Plosila, J., Jantsch, A., and Tenhunen, H. (2015c). Dynamic Power Management for Many-Core Platforms in the Dark Silicon Era: A Multi-Objective Control Approach. In *Proc. Int. Symp. on Low Power Electronics and Design (ISLPED)*, pages 1–6.

[Rao et al., 2017] Rao, K., Wang, J., Yalamanchili, S., Wardi, Y., and Handong, Y. (2017). Application-specific performance-aware energy optimization on android mobile devices. In *High Performance Computer Architecture (HPCA), 2017 IEEE International Symposium on*, pages 169–180. IEEE.

[Rexha et al., 2017] Rexha, H. et al. (2017). Core Level Utilization for Achieving Energy Efficiency in Heterogeneous Systems. In *Proc. of PDP*, pages 401–407.

[Rinard, 2006] Rinard, M. (2006). Probabilistic Accuracy Bounds for Fault-Tolerant Computations that Discard Tasks. pages 324–334.

[Rinard et al., 2004] Rinard, M., Cadar, C., Dumitran, D., Roy, D. M., Leu, T., and Beebee, W. S. J. (2004). Enhancing server availability and security through failure-oblivious computing. page 21.

[Rotem, 2012] Rotem, E. o. (2012). Power-management architecture of the intel microarchitecture code-named sandy bridge. *IEEE Micro*, 32(2):20–27.

[Saez et al., 2012] Saez, J. C., Fedorova, A., Koufaty, D., and Prieto, M. (2012). Leveraging core specialization via os scheduling to improve performance on asymmetric multicore systems. *ACM Trans. Comput. Syst.*, 30(2):6:1–6:38.

[Samadi et al., 2014] Samadi, M., Jamshidi, D. A., Lee, J., and Mahlke, S. (2014). Paraprox : Pattern-Based Approximation for Data Parallel Applications. pages 35–50.

[Samadi et al., 2013] Samadi, M., Lee, J., and Jamshidi, D. (2013). Sage: Self-tuning approximation for graphics engines.

[Sampson et al., 2011] Sampson, A. et al. (2011). {EnerJ}: Approximate Data Types for Safe and General Low-power Computation. In *PLDI*.

[San Miguel and Badr, 2014a] San Miguel, J. and Badr, M. (2014a). Load Value Approximation.

[San Miguel and Badr, 2014b] San Miguel, J. and Badr, M. (2014b). Load Value Approximation.

[Schlachter et al., 2015] Schlachter, J., Camus, V., and Enz, C. (2015). Near/Sub-Threshold Circuits and Approximate Computing: The Perfect Combination for Ultra-Low-Power Systems. *Proc. if ISVLSI*, pages 476–480.

[Schulz et al., 2017] Schulz, P. et al. (2017). Latency critical iot applications in 5g: Perspective on the design of radio interface and network architecture. *IEEE Communications Magazine*, 55(2):70–78.

[Semiconductor Industry Association, 2013] Semiconductor Industry Association (2013). International technology roadmap for semiconductors (ITRS), 2013 edition.

[Sha et al., 2018] Sha, S., Wen, W., Ren, S., and Quan, G. (2018). M-oscillating: Performance maximization on temperature-constrained multi-core processors. *IEEE Transactions on Parallel and Distributed Systems*.

[Shafique et al., 2014a] Shafique, M., Garg, S., Henkel, J., and Marculescu, D. (2014a). The eda challenges in the dark silicon era. In *Proc. of DAC*, pages 1–6.

[Shafique et al., 2014b] Shafique, M., Garg, S., Mitra, T., Parameswaran, S., and Henkel, J. (2014b). Dark silicon as a challenge for hardware/software co-design: Invited special session paper. In *Proc. of ACM International Conference on Hardware/Software Codesign and System Synthesis*, page 13.

[Sidiroglou et al., 2011a] Sidiroglou, S. et al. (2011a). Managing performance vs. accuracy trade-offs with loop perforation. pages 124–134.

[Sidiroglou et al., 2011b] Sidiroglou, S. et al. (2011b). Managing performance vs. accuracy trade-offs with loop perforation. In *FSE*.

[Sidiroglou et al., 2011c] Sidiroglou, S., Misailovic, S., Hoffmann, H., and Rinard, M. (2011c). Managing performance vs. accuracy trade-offs with loop perforation. pages 124–134.

[Sondag and Rajan, 2009] Sondag, T. and Rajan, H. (2009). Phase-guided thread-to-core assignment for improved utilization of performance-asymmetric multi-core processors. In *Proceedings of the 2009 ICSE Workshop on Multicore Software Engineering*, IWMSE '09, pages 73–80, Washington, DC, USA. IEEE Computer Society.

[Sozzo et al., 2016] Sozzo, E. D., Durelli, G. C., Trainiti, E. M. G., Miele, A., Santambrogio, M. D., and Bolchini, C. (2016). Workload-aware power optimization strategy for asymmetric multiprocessors. In *Proceedings of the 2016 Conference on Design, Automation & Test in Europe*, DATE '16, pages 531–534, San Jose, CA, USA. EDA Consortium.

[Tan et al., 2015] Tan, C. et al. (2015). Approximation-aware scheduling on heterogeneous multi-core architectures. In *Proc. of ASP-DAC*, pages 618–623.

[Tang et al., 2016] Tang, K., Tiwari, D., Gupta, S., Huang, P., Lu, Q., Engelmann, C., and He, X. (2016). Power-capping aware checkpointing: On the interplay among power-capping, temperature, reliability, performance, and energy. In *Dependable Systems and Networks (DSN), 2016 46th Annual IEEE/IFIP International Conference on*, pages 311–322. IEEE.

[Taylor, 2012] Taylor, M. (2012). Is dark silicon useful? harnessing the four horsemen of the coming dark silicon apocalypse. In *Proc. Design Automation Conference (DAC)*, pages 1131–1136.

[Thwaites et al., 2014] Thwaites, B. et al. (2014). Rollback-Free Value Prediction with Approximate Loads.

[Tomusk et al., 2016] Tomusk, E., Dubach, C., and O'boyle, M. (2016). Selecting heterogeneous cores for diversity. *ACM Trans. Archit. Code Optim.*, 13(4):49:1–49:25.

[Tong et al., 2000] Tong, J. Y. F., Nagle, D., Rutenbar, R., et al. (2000). Reducing power by optimizing the necessary precision/range of floating-point arithmetic. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 8(3):273–286.

[Van Craeynest et al., 2012] Van Craeynest, K., Jaleel, A., Eeckhout, L., Narvaez, P., and Emer, J. (2012). Scheduling heterogeneous multi-cores through performance impact estimation (pie). In *ACM SIGARCH Computer Architecture News*, volume 40, pages 213–224. IEEE Computer Society.

[Varatkar and Shanbhag, 2006] Varatkar, G. and Shanbhag, N. (2006). Energy-efficient Motion Estimation using Error-Tolerance. *ISLPED'06 Proceedings of the 2006 International Symposium on Low Power Electronics and Design*.

[Vega et al., 2013] Vega, A. et al. (2013). Crank it up or dial it down: Coordinated multiprocessor frequency and folding control. In *MICRO*, pages 210–221.

[Venkataramani et al., 2013] Venkataramani, S., Chippa, V. K., Chakradhar, S. T., Roy, K., and Raghunathan, A. (2013). Quality programmable vector processors for approximate computing. *46th Annual IEEE/ACM International Symposium*, pages 1–12.

[Venkatesh et al., 2010] Venkatesh, G., Sampson, J., Goulding, N., Garcia, S., Bryksin, V., Lugo-Martinez, J., Swanson, S., and Taylor, M. B. (2010). Conservation cores: Reducing the energy of mature computations. In *Proceedings of the Fifteenth Edition of ASPLOS on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XV, pages 205–218, New York, NY, USA. ACM.

[Wang and Skadron, 2012] Wang, L. and Skadron, K. (2012). Dark vs. Dim Silicon and Near-Threshold Computing Extended Results. In *University of Virginia Department of Computer Science Technical Report TR-2013-01*.

[Wang and Skadron, 2013] Wang, L. and Skadron, K. (2013). Implications of the power wall: Dim cores and reconfigurable logic. *IEEE Micro*.

[Wu et al., 2016] Wu, Q., Deng, Q., Ganesh, L., Hsu, C.-H., Jin, Y., Kumar, S., Li, B., Meza, J., and Song, Y. J. (2016). Dynamo: facebook's data center-wide power management system. In *Computer Architecture (ISCA), 2016 ACM/IEEE 43rd Annual International Symposium on*, pages 469–480. IEEE.

[Ye et al., 2013] Ye, R., Wang, T., Yuan, F., Kumar, R., and Xu, Q. (2013). On reconfiguration-oriented approximate adder design and its application. *IEEE/ACM International Conference on Computer-Aided Design, Digest of Technical Papers, ICCAD*, pages 48–54.

[Yun et al., 2015] Yun, J., Park, J., and Baek, W. (2015). Hars: A heterogeneity-aware runtime system for self-adaptive multithreaded applications. In *Proceedings of the 52Nd Annual Design Automation Conference*, DAC '15, pages 107:1–107:6, New York, NY, USA. ACM.

[Zadeh, 1994] Zadeh, L. A. (1994). Fuzzy logic, neural networks, and soft computing. *Commun. ACM*, 37(3):77–84.

[Zhan and Reda, 2013] Zhan, X. and Reda, S. (2013). Techniques for energy-efficient power budgeting in data centers. In *Proceedings of the 50th Annual Design Automation Conference*, page 176. ACM.

[Zhang et al., 2013] Zhang, Y., Peng, L., Fu, X., and Hu, Y. (2013). Lighting the dark silicon by exploiting heterogeneity on future processors. In *Proceedings of the 50th Annual Design Automation Conference*, DAC '13, pages 82:1–82:7, New York, NY, USA. ACM.

# Part II

# Original Publications

# Paper I

# Dark Silicon Aware Runtime Mapping for Many-core Systems: A Patterning Approach

Anil Kanduri, Hashem Haghbayan, Amir Rahmani, Axel Jantsch, Pasi Liljeberg, Hannu Tenhunen

Published in Proccedings of 33rd IEEE International Conference on Computer Design (ICCD), 2015, New York, USA, pp: 573-580.

# Dark Silicon Aware Runtime Mapping for Many-core Systems: A Patterning Approach

Anil Kanduri[1], Mohammad-Hashem Haghbayan[1], Amir-Mohammad Rahmani[1,3],
Pasi Liljeberg[1], Axel Jantsch[2], and Hannu Tenhunen[1,3]
[1]Department of Information Technology, University of Turku, Turku, Finland
[2]TU Wien, Austria
[3]Department of Industrial and Medical Electronics, KTH Royal Institute of Technology, Stockholm, Sweden
*Email: {spakan, mohhag, amirah, pakrli}@utu.fi, axel.jantsch@tuwien.ac.at, hannu@kth.se*

***Abstract*—Limitation on power budget in many-core systems leaves a fraction of on-chip resources inactive, referred to as dark silicon. In such systems, an efficient run-time application mapping approach can considerably enhance resource utilization and mitigate the dark silicon phenomenon. In this paper, we propose a dark silicon aware runtime application mapping approach that patterns active cores alongside the inactive cores in order to evenly distribute power density across the chip. This approach leverages dark silicon to balance the temperature of active cores to provide higher power budget and better resource utilization, within a safe peak operating temperature. In contrast with exhaustive search based mapping approach, our agile heuristic approach has a negligible runtime overhead. Our patterning strategy yields a surplus power budget of up to 17% along with an improved throughput of up to 21% in comparison with other state-of-the-art run-time mapping strategies, while the surplus budget is as high as 40% compared to worst case scenarios.**

***Keywords*—Dark Silicon; Power Budgeting; Runtime Mapping**

## I. Introduction

Power density of many-core systems is alarmingly increasing for every technology node generation, attributed to slack voltage scaling that is not on par with technology node scaling and exponential rise in leakage power [1]. Increase in power density leads to thermal issues [2], forcing the chip's functionality to fail. To ensure a safe operation, it is critical for the chip to perform within a fixed upper bound on power budget [3]. In order to stay within in this limit, a certain section of the chip has to remain inactive - a growing phenomenon termed as Dark Silicon [4]. Compute intensity of future applications such as deep machine learning, virtual reality, big data etc., demands further technology node scaling, subsequently leading to further rise in power density and dark silicon. ITRS projections have predicted that by 2020, designers would face up to 90% of dark silicon, meaning that only 10% of the chip's hardware resources are useful at any given time [5]. Increasing dark silicon directly reflects on performance to a point that multi-core and many-core scaling provides zero gain [4].

Many-core systems face a new set of challenges at the verge of dark silicon to continue providing the expected performance and efficiency [6]. Run-time application mapping policy is one of the key factors that can influence performance and energy efficiency of many-core systems [7]. Mapping is the process of choosing a preferable set of cores on the chip to run tasks of an application, minimizing congestion and maximizing performance [8], [9]. With dynamic workload characteristics and un-predictable sequence and arrival of applications, mapping decisions had to be made at run-time for current and future many-core systems. Thus far, designers have been using mapping strategies assuming that all the cores of a chip are active and available. However with dark silicon scenario, this completely alters as not all the cores can be active at a given time and the number of cores that can be active varies depending on activity of other working cores [10]. Existing run-time mapping algorithms are dark silicon agnostic and will not be able to provide the high performance they used to, as they do not consider the availability of active cores and are likely to violate safe power budget. All of these factors necessitates a dark silicon aware run-time mapping strategy to continue achieving performance gain and energy efficiency through technology node scaling.

Thermal Design Power (TDP) is a standard design time metric that has been used to determine a safer upper bound on chip's power consumption. Safer operation of a chip is guaranteed as long as power consumption stays within TDP [11]. TDP is single fixed upper bound that is pessimistically estimated assuming that all the cores are active and are operating at a worst case voltage and frequency. With dark silicon phenomenon, a variable number of cores will be inactive (dark), depending on current set of applications running, ambient temperature and most importantly, number of simultaneously active cores. Thus, the safe upper bound on power budget varies in run-time, as opposed to the conservative upper bound of TDP which leads to under-utilization of available power budget resulting in dark silicon [10]. A sensible way to avoid conservative limit of TDP is to use a variable and realistic upper bound on power consumption, Thermal Saturation Power (TSP), as proposed by Pagani *et al.* [10]. TSP is modeled as a function of simultaneously active cores, their alignment, effect of temperature of a core on its neighbors and ambient temperature. At any given time instance, the amount of power the chip can consume to safely operate will depend on the alignment of active cores, which is determined by application mapping. It can be deduced that for same set of applications and same number of active cores, a certain mapping can result in a higher power budget over other mappings, based on appropriate alignment of active and dark cores. Subsequently, the surplus budget gained through mapping can be utilized to power up more cores, minimizing dark silicon and thus offering higher performance. This is explained through an example presented in Figure 1.

A contiguous mapping of an application with 6 tasks

978-1-4673-7166-7/15/$31.00 

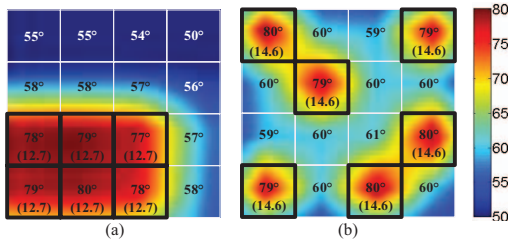



Fig. 1: Effect of mapping on Power Budget [10]

is shown in 1 (a). Since all the active cores are tightly packed, the heat dissipated by every active core affects its neighbors hazardously. As a result, the cores reach their critical temperature of (80°C) after consuming 12.7W of power. This configuration of mapping can effectively utilize 12.7×6 = 76.2W of power, which would be its power budget. Utilizing power beyond this budget would heat up the cores beyond their critical temperature leading to permeant failure. The mapping shown in Figure 1(b) is sparsely distributed in comparison with the previous mapping. In this case, as the cores are separated out, the heating effect of one active core on the other is minimized. Therefore, the cores can now consume 14.6W with reaching their critical temperature. Effectively, the chip's power budget in this case is 14.6×6 = 87.6W, which is 11.2W (14.9%) more than the previous case, establishing the impact of mapping on providing a better power budget. This surplus budget can in turn be used to: i) activate more cores, ii) run current tasks much faster, and iii) run more tasks without reaching critical temperatures.

Although it has been shown that optimal mapping can provide higher power budget [12][13], to the best of our knowledge, no methodical approach exists on how to pattern the active cores alongside dark cores such that they result in higher power budget. In this paper, we propose a dark silicon aware run-time mapping approach which aligns active cores along with dark cores that can evenly distribute heat dissipation across the chip. The surplus budget we gain through mapping rises the upper bound on power consumption which is used to activate more cores, directly mitigating the dark silicon phenomenon. This way, we maximize the utilization on power budget to improve performance and energy efficiency. To the best of our knowledge, ours is the first work to consider dark silicon scenario for run-time application mapping and to pattern the active and dark cores to improve the power budget. The key contributions of this work are as follows:

- A dark silicon aware runtime application mapping approach that aligns active cores with dark cores to offer higher power budget.
- A closed-loop power budgeting platform that keeps the maximum power consumption under safe operational power (i.e., TSP) which varies at runtime.

The rest of the paper is organized as follows: In Section II, related work on dynamic mapping and power budgeting is presented. Our proposed mapping strategy and overview of the system are detailed in Section IV. Experimental setup and results are presented in Section V. Finally, Section VI concludes the paper and discusses potential future work.

## II. Related Work

State-of-the-art dynamic application mapping strategies have targeted benefits in terms of network performance, minimizing congestion, system throughput, power optimization, etc [14][8][15], without any fixed upper bound on power consumption. The main focus of mapping strategies so far is on inter-task communication and their objective is maximizing performance, forcing them to map contiguously [16][17][18]. Precisely, these algorithms do not consider the issue of dark silicon. However, power budget (and thus performance) limitations of future many-core systems emphasizes to re-structure the objective of mapping towards improving power budget by considering dark silicon. Conventional mapping policies advocate avoiding dispersion and fragmentation and do not consider the issue of dark silicon which changes the impact of dispersion and fragmentation on system performance [19]. A non-contiguity mapping through geometrical partitioning of the network is presented in [20], which shows that penalties on performance can be minimized by mapping communicating tasks on nearby cores and the rest in proximity, but not necessarily contiguous. They have established that non-contiguous not necessarily affects system performance, although they do not exploit this fact to attack dark silicon. A patterning approach to avoid congestion between packets routed from same row or column is proposed in [21]. They limit the number of tasks to one per row and one per column and do not consider any power budget.

A feedback based power management system is proposed in [3]. They are limited to restricting the violation of TDP through a PID controller and they use a conventional dynamic mapping approach [8], which is not dark silicon aware. An online learning approach is presented in [22] that employs various power management techniques whenever there are hot-spots identified in the system with changing workloads. It is restricted to multi-core systems and does not consider dark silicon. Thermal aware system analysis and calibration is presented in [23] at a lower level of abstraction, yet they do not propose effective task allocation based on their detailed analysis. Liu *et al.* [24] present an energy and thermal aware mapping strategy for NoC-based systems. Their approach is limited to design time (static mapping) which is based on heuristic that estimates temperature, resulting into a near exhaustive search to find optimal nodes. A proactive estimation of potential hot-spots through temperature sensors is presented in [25]. They mitigate identified hot-spots through thread migration and dynamic voltage scaling. This method suffers from performance degradation by voltage down scaling and overhead in migration. Bao *et al.* [26] present a case for balancing the heat dissipation of the chip evenly through a temperature aware mapping. Their mapping is based on voltage down scaling when needed as per critical temperatures of cores, putting a limit on its performance. Power capping of the cores through dynamic voltage and frequency scaling by identifying hot-spots at runtime is presented in [27]. The authors try to minimize the power consumption of a specific section of chip and thus to balance heat distribution, however, no run-time mapping strategy is either presented in their work. On the whole, thermal aware mappings stay within upper bound of power budget and avoid hot-spots, but do not utilize the available budget effectively, neither improve it.

The phenomenon of dark silicon and effects of utilization wall were identified in pioneering works in [4][28]. Thus far,

most common practice of ensuring safe chip functionality has been by estimating TDP [11] in a conservative manner. Recent upgrades on AMD and Intel Corporations' CPUs have the option of a configurable TDP, however they are limited to a maximum of 3 modes, without any fine grained control [29][30]. Pagani *et al.* have proposed an adaptive way of setting the upper bound on power consumption by expressing it as a function of simultaneously active cores in [10]. They provide a light weight C library to estimate TSP for a given mapping and also the worst case TSP for a given number of active cores. Despite these, they do not provide any insight on which mapping would offer a better power budget. The importance of spatial alignment of dark cores along with active cores and its effects on power budget have been presented in [12]. It shows gain of patterning in two perspectives viz., better power budget and lower operating temperatures. [13] quantitatively showed that different patterns of dark silicon result in different power budgets and temperature profiles of the chip. However, both these works do not propose any method on how to pattern the dark tiles to get such higher power budgets. Taking all these things together into perspective, it is necessary for a run-time mapping algorithm that patterns active and dark cores to maximize utilization of power budget, considering dark silicon.

## III. Motivation

The heat dissipated by a core $C_i$ is a 3-tuple ($P_i$, $T_n$, $T_{amb}$), where $P_i$ is the power dissipated by the core, $T_n$ is the temperature of neighbouring cores and $T_{amb}$ is the ambient temperature. When active cores function at full throttle, they dissipate power and heat that is proportional to the power. If temperature goes beyond the safer limit, chip's functionality would fail permanently. Generally, dynamic thermal management techniques such as clock and power gating, voltage and frequency down scaling, increased fan speed etc., are triggered to manage any such sudden phases of chip's overheating. This would reduce activity inside the chip and let the chip back to steady state temperature over a period of time. Although uneven heat distribution can be managed with these, it hampers the performance by a great deal. A better way for even distribution of heat is to pattern the dark cores along with working cores through runtime mapping.

The impact of spatial alignment of active cores on power budget is explained through a motivational example, presented in Figure 2. Three applications App1, App2 and App3 with 9, 12 and 7 tasks respectively are assumed to be running on the system. Conventional mapping approaches offer lower inter-task communication latency by greedily mapping all the applications contiguously. Mapping of the 3 applications contiguously on a NoC-based many-core system with 144 cores is shown in Figure 2(a). The power budget (TSP) of this system as computed by TSP library is 66W. A non-contiguous and spread-out mapping of the same applications (as well tasks) is shown in Figure 2(b). This mapping provides a power budget (TSP) of 74.6W, as calculated by TSP library. An improvement of 8.6W in power budget can be observed for the spread-out, patterned mapping as opposed to tightly packed and contiguous mapping. Contiguous mapping avoids dispersion, but it leads to poor thermal profile of the chip due to prorogation of heat among neighboring applications and tasks and thus resulting in lower power budget. Contrastingly, spatially distributed mapping of applications offers higher power budget as effect of heat among different applications is negligible. Also, active

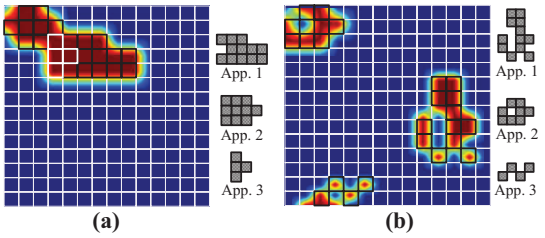



Fig. 2: Thermal profiles of contiguous and spatially distributed mappings

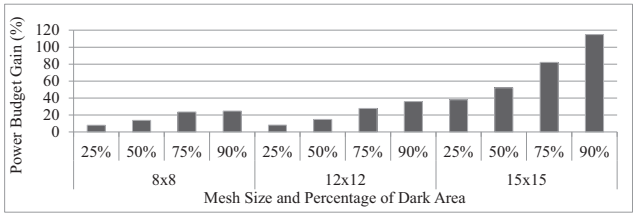



Fig. 3: Surplus power budget with increasing dark silicon

cores are patterned along with inactive cores such that heat effects of neighboring cores running the same application are minimized. Based on the spatial alignment and a minor compromise on dispersion, we could gain up to 13% of power budget. It is to be noted that the inactive cores are the inevitable dark cores - instead of leaving them out naively, we use them to balance out heat distribution and gain power budget. In the mapping presented in Figure 2(a), the budget of 66W is used to power 22 cores, giving a per-core budget of 3W. Assuming a per-core budget of 3W, the mapping configuration in Figure 2(b) could power 74.6/3 = 24.7 cores. The surplus budget we could gain via patterning could thus be used to power up 2.7 more cores, reducing dark silicon by 12.2%. This benefit could be better realized with increased size of the many-core system, technology node scaling and more dark silicon which is expected in near future. The average packet latency in each of the patterned mappings for App1, App2 and App3 is 1%, 3.5% and 3.8% more than that of the contiguous mappings. Although there is a minor penalty in terms of latency, the gain in power budget outweighs the odds against it.

Figure 3 shows the surplus power budget that was gained for different mapping configurations over the worst case TSP budget. We ran random mappings with fixed amount of inactive (dark) cores ranging from 25% to 90% of darkness, over mesh sizes of 8x8, 12x12 and 15x15 with 22nm technology. For the same number of active cores, we collected the worst case TSP budgets. The difference between a random configuration and the worst case budget is presented as percentage gain in power budget. It is evident that the gain in power budget increases with increase in amount of dark silicon, to an extent where we can get a surplus of up to 114% when 90% of core is dark. It is to be noted that this surplus is purely from a random mapping generated to represent the power budget gain that can possibly be achieved, although the realistic gain could be variable.

## IV. The Proposed Mapping and Patterning Approach

### A. System Architecture

The top level abstraction of the system we implemented in our approach is shown in Figure 4. Application Repository

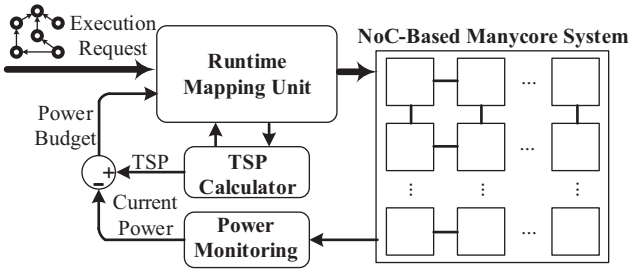



Fig. 4: Architecture Implementing the proposed mapping strategy

holds the applications modeled as task graphs and are released onto the system for execution over time. Runtime Mapping Unit (RMU) monitors the power profile of incoming applications issued by Application Repository, to ensure that upper bound on power budget is not violated by mapping a new application. RMU estimates the power of incoming application and checks if the chip currently has enough power budget to run the new application. An example of such estimation can be found in [3]. The application is forwarded onto the system if there is available budget. In case of un-availability, the application waits until the system can allocate enough budget, perhaps with currently running application(s) leaving the system after finishing their execution. TSP Calculator receives current mapping configuration of the system as input and computes the realistic upper bound on power budget, the thermal safe power (TSP). The RMU feeds this new budget value to the chip and updates the maximum power budget of the chip to the TSP provided by the TSP Calculator.

Application mapping is to find a free region for an application on the chip, which is of polynomial time complexity [31]. However, we have constraints on finding an optimal region such as power consumption, communication latency, dispersion, patterning the dark cores etc., making mapping an NP-hard problem. Surplus budget gained from mapping in Figure 2(b) can be attributed to the mapping policy. Two major factors that distinguishes it from contiguous mapping are spatial distribution of applications and sparsity among tasks of each individual application. In view of these factors, we split our mapping approach into two phases viz., selecting a region that is spatially dispersed from current set of applications running on chip and mapping tasks of the application sparsely such that active and inactive cores are patterned in the selected region.

Finding an optimal region for an application starts with finding an optimal node, the *first node*, around which an application can be mapped. In our approach, we prioritize nodes (thus regions) that are spatially far from currently active cores as *first nodes*. Tightly packed up active cores are the major reason for hop-spots which eventually lead to under utilization of power budget. Hence, after region selection, we map tasks of the application in a sparse pattern that will reduce the probability of heat accumulating at specific regions that potentially turn out as hot-spots. Our approach tries to attain an even distribution of heat at both region selection (through optimal first node selection) and mapping (through patterning).

### B. First Node Selection

In contrast to reactive strategies, we exploit MapPro [9] which pro-actively calculates an optimal first node for incoming applications of every possible size by assigning the square that can fit the application. This way, we totally eliminate the time spent in first node selection, which automatically reflects in the overall execution time. The choice of a particular first node and thus a particular square region is based on our objective to find a region that has: i) free nodes to allocate for incoming application with minimal internal congestion and ii) minimal effect in terms of temperature on other regions. To quantify these objectives, we use Vicinity Counter (*VC*) defined in [9] which addresses the aforementioned first objective. We then combine it with a new parameter viz., Distance Factor (*DF*) which addresses the second objective.

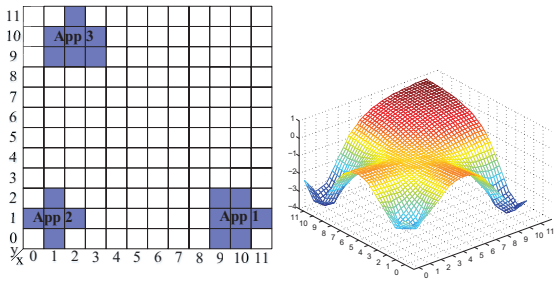
VC parameter expresses the number of free nodes, which also represents the number of occupied nodes. Using this parameter, we consider the affect of occupied nodes in the region on internal congestion, by quantifying the location of occupied nodes. For instance, a node that is occupied in the inner most square close to the first node has more affect on internal congestion than the ones that are occupied in outer squares, far from the first node. Therefore, we quantify this by pegging the weight of an occupied node with its distance from the central node, by assigning a higher penalty to occupied nodes closer to central node and relatively lower penalty to the ones that are far. The VC value represents the availability and congestion around a chosen node, and makes it easier to select between nodes with the same VC value, by considering congestion.

***Definition:*** *Distance Factor, $DF_{ij}$, for a node located at (i,j) is the weighted sum of impact of distance from all the other occupied nodes located at (x,y) such that $(x, y) \in Mesh$.*

$$DF_{i,j} = \sum Wn_{i,j} \times (e^{-\alpha(d_{ij-xy})}) \quad (1)$$

where $Wn_{ij}$ is the weight of node $n_{ij}$, $d_{ij-xy}$ is distance from nodes located at (i,j) and (x,y) and $\alpha$ is the mesh size. Thus the DF of a node represents the effect of heat from concurrently running applications on the chip. This helps in selection of a region that is less probable to generate any potential hotspots. In addition, the issue of two or more nodes having a same VC value can be resolved by examining their corresponding DF values, and the node with the lower (best) DF value is chosen.

When an un-occupied node becomes occupied by a new task, it starts dissipating power and thus heat. Initially, the heat is concentrated on the node itself, over time it starts impacting its neighbors. However, the effect of heat dissipated by this node on neighboring nodes gradually decreases as we move towards farther nodes. Inspired by the surface tension phenomenon [32], where energy distribution of a flat surface turns into a curved surface with applied surface pressure, we model the effect of heat transfer in a similar fashion. The temperature effect of every active core decreases exponentially, but not linearly, with distance from the active (hot) core. In other words, greater the distance from an active core, lesser the effect of heat from it. We illustrate this effect in Figure 5, for the chip running 3 applications App1, App2 and App3. The regions where applications are mapped (deeper zones) have a DF as low as -4, while the regions that are far away (shallow zones) from them have a DF of 1. Also, as we traverse towards the far off shallow zones, the DF value improves indicating the




(a) x (b) y

Fig. 5: Effect of occupied cores on Distance Factor of un-occupied cores

importance of distributing the application spatially across the chip. This effect also depends on mesh size such that smaller mesh results in greater impact, due to more proximity among hot and its neighboring nodes. In order to minimize this heating effect from active cores of other applications, we prioritize nodes that are as far as possible from such active cores.

The algorithmic flow of first node selection is shown in Algorithm 1 which is inspired by our previous work presented in [9]. When a new application enters the system, power budget required for it is estimated. If the system has enough budget to be allocated for the new application, then the application request is serviced by finding required first node (lines 1-3). The node with maximum VC value, $maxVC$, among all nodes is chosen as the first node, and thus also selecting the square centered at $maxVC$. The chosen first node and the application are passed to patterning phase (line 4). Mapping the tasks based on patterning is explained in the following section in Algorithm 2. The chip's mapping configuration changes after mapping one application. As a result, the VC and DF values for the remaining un-occupied nodes need to be updated for every newly occupied node. Once the application is mapped, we calculate the new node with maximum value of VC $maxVC$ for different radii and also update the DF values of un-occupied nodes. This way, we pro-actively calculate the first node for next incoming application and avoid the overhead caused in first node selection. VC values for square of different groups are updated (lines 6-23) to determine the new $maxVC$ node. In case of a conflict when more than one node has the same $maxVC$, it is resolved by comparing their DF values (lines 16-18). Thus the VC and DF values are pro-actively updated with the intention of servicing the next incoming application immediately, without any overhead. Nodes are released from the system when an application leaves after finishing its execution. The VC and DF values are once again updated according to the changed mapping configuration of the chip, with the exit of an application.

### C. Dark Silicon Patterning

An optimal region for mapping an incoming application is chosen via first node selection. The mapping receives the first node ($fn$) as an input and builds a polygon $P$ which is the set of all un-occupied nodes around the selected first node. We choose nodes among the region $P$ such that the tasks are run on nodes that are sparsely aligned. Sparsity of a node $n_{ij}$ represents the number of free nodes that are neighboring

**Algorithm 1** The mapping algorithm

**Inputs:** $newApp$:New application, $budget$:Available power budget;
**Outputs:** $Q$:Mapping;
**Constants:** $M$ :Size of the mesh, $groups$ : Number of square groups = $[(\sqrt{M}-1)/2]$,$maxRadius$:Maximum radius of the square ($(\sqrt{M}-1)/2$), $\alpha$:Mesh size parameter $\sqrt{M}$,$P_{avg}$: Average power consumption per node;
**Global Variables:** $VC$ : Vicinity Count of a node, $maxVC$ : Node with the maximum VC, $firstNode$ : Selected first node for mapping, $DF$ : Distance factor;

**Body:**

1: $appPredictedPower \leftarrow |newApp| \times P_{avg}$;
2: **if** $appPredictedPower \leq budget$ **then**
3: $firstNode \leftarrow maxVC_{[(\sqrt{appSize}-1)/2]}$
4: $Q \leftarrow pattern(firstNode, newApp)$;
5: //Updating VC and DF values after mapping
6: **for** each $n_{xy} \in newApp$ **do**
7: **for** each core $n_{ij}$ located in Row $i$ and Column $j$ **do**
8: $r'$ = $maximum(|i-x|, |j-y|)$;
9: $DF_{ij} -= e^{-\alpha r'}$;
10: **for** $r$ = 1 to $maxRadius$ **do**
11: **if** $r - r' \geq 0$ **then**
12: $VC^r_{ij}$ -= $r$ - $r'$;
13: **if** $VC^r_{ij} > maxVC_r$ **then**
14: $maxVC_r \leftarrow VC^r_{ij}$;
15: **else**
16: **if** $VC^r_{ij} = maxVC_r$ and $DF_{ij} > DF_{maxVC_r}$ **then**
17: $maxVC_r \leftarrow VC^r_{ij}$;

it in all four cardinal directions (North, East, West, South). The Sparsity Factor ($SF_{ij}$) for a node $n_{ij}$ located at $(i, j)$ is expressed as:

$$SF_{i,j} = \sum_{i=1}^{4}\sum_{j=1}^{4} F(i+i', j+j') \quad (2)$$

where $i'$ = [0, 1, 1, -1], $j'$ = [-1, 0, 1, 1]. $F(i, j)$ denotes if a node located at $(i, j)$ is free or not, such that

$$F(i,j) = \begin{cases} 1 & \text{if } n_{i,j} \text{ is unoccupied} \\ 0 & \text{if } n_{i,j} \text{ is occupied} \end{cases}$$

Based on the Sparsity Factor, we prioritize the nodes that are more sparse. Sparse nodes ensure that they have less effect on neighbor's temperature. As a result, the dense nodes are assigned least priority and are patterned in a way that they provide necessary cooling effect needed by their active neighbors. While mapping, we sort the tasks of the incoming application as per communication volume. We choose the task with highest communication volume and map it on to the first node, the most sparse node. We proceed to the task with next highest communication expense and map it onto the node with the highest $SF$ among the nodes in the selected square $S$. We continue similarly, in order of communication of tasks and sparsity of nodes so that tasks with higher communication gets mapped onto nodes with higher sparsity, until all tasks of the application are mapped. This way, the less sparse nodes (i.e., the dense nodes) which are tightly packed with active cores as neighbors gets least priority. These denser nodes eventually remain un-occupied among the other occupied nodes of the region, minimizing the probability of creating potential hot-

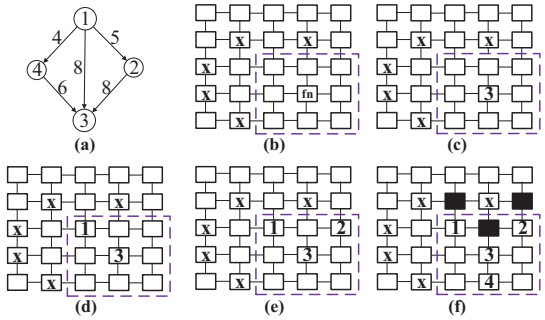



Fig. 6: Example for pattern based mapping

spots.

The patterning is explained through an example presented in Figure 6. Assuming an application with 4 tasks, as in Figure 6(a), has arrived, the selected first node $fn$ and the corresponding square region is shown in Figure 6(b). The most expensive task of the applications is task 3, which is mapped on most sparse node, the $fn$. This automatically effects the SF values of nodes surrounding it. The next in order of communication is tasks 1, and the next node with higher SF is chosen and task 1 is mapped on it. In the similar fashion, the remaining tasks are mapped subsequently, as in Figure 6(c)-(f). It is to be observed that after the application is mapped, one node in the square is left such that it has SF = 0, which would be of least priority for any incoming application. On the same lines, there are other nodes with SF = 1, which also attained a relatively lower priority for getting mapped. These nodes are the candidates that would potentially be dark, and provide the cooling effect needed by their active neighbors. Thus, we pattern the dark cores among active cores to minimize the probability of heat getting accumulating at any single point on the chip. Conversely, had all the tasks of the application been mapped contiguously, they would have reached their critical temperatures by consuming only lower amount of power and eventually trigger dark cores else where on the chip. However, in the case where we patterned the inevitable dark cores, the cores will reach critical temperatures only after utilizing available power budget to a better extent, or offer more budget to activate more cores.

The algorithmic flow of patterning is shown in Algorithm 2. Patterning starts with a selected first node, $firstNode$ and chooses the square region ($Square^{size}_{firstNode}$) that can fit the application ($App$) (line1). The application is sorted into tasks ($Tasks$) as per their communication volume (line 2). The most expensive task is mapped onto the node with maximum Sparsity Factor, $maxSF$ (lines 5-6). The mapped task is removed from the list of tasks to be mapped and the mapped node is removed from the list available nodes in the square (lines 7-8). This procedure is repeated until all the tasks of the application are mapped. Intuitively, the $SF$ value for nodes in selected square changes with every occupied and thus new $maxSF$ is computed for every unmapped task.

## V. Evaluation

We simulated our proposed mapping strategy over applications modeled as task graphs of different sizes ranging from 4 to 35 tasks, generated using [33]. Communication volumes among these tasks are distributed randomly using Gaussian elimination. We simulated traffic patterns of these

**Algorithm 2** Patterning

**Inputs:** $App$ : Application,$firstNode$ : Selected First Node.
**Global Variables:** $S$ Selected square, $SF$ : Sparsity Factor for each node in a square, $maxSF$ : Node with the maximum SF in a square, $currentNode$ : Node onto which current task is being mapped;
**Global Constants:** $Tasks$ : Vector of tasks of the application $App$,$size$: Radius of square close to size of $App$

**Body:**

1: $S \leftarrow Square^{size}_{firstNode}$;
2: $Tasks$ = $sort(App)$;
3: **while** $App \neq \emptyset$ **do**
4: **for** each $t_i \in Tasks$ **do**
5: $currentNode \leftarrow maxSF$;
6: $map\ (t_i) \rightarrow maxSF$;
7: $App - t_i$ ;
8: $S - currentNode$ ;

applications using our in-house cycle-accurate many-core platform implemented in SystemC. The specifications of Niagara-2 like in-order cores obtained from McPAT [34] are used as the baseline for processing elements. The communication network infrastructure between processing elements is provided by a pruned version of Noxim [35] that uses mesh topology and XY routing. Parameters related to technology node scaling are extracted from Lumos framework [36], an open source framework which quantifies power-performance characteristics of many-core systems with technology node scaling. We used TSP library [10] to calculate the Thermal Safe Power. Proposed dynamic mapping is implemented by a Central Manager ($CM$), which is the node $n_{(0,0)}$ of the mesh in our many-core platform. In the many-core platform we implemented, (overview as in Figure4), a random sequence of applications enter the system and are buffered into a FIFO. Applications are serviced in a first-come-first-serve policy, subject to availability of enough power budget. If enough power budget can be allocated, the application is scheduled by the Central Manager ($CM$). A suitable first node for mapping the incoming application is chosen by the ($CM$), followed by mapping all the remaining tasks of the application.

We evaluate our dark silicon patterning approach (from here on referred to as *PAT*) against the combination of *SHiC* [8] and *CoNA* [19] (from here on referred to as *SC*), for first node selection and mapping respectively. These two approaches are state-of-the-art strategies that prioritize regions with free nodes and contiguity among nodes selected for mapping, with the primary objective of minimizing communication latency. Since we relaxed the constraint on contiguity among individual applications, we chose to compare against these works to quantify the effect of our patterning approach against contiguity. In view of applications that require a conservative upper bound on power, we also compare $PAT$ against worst case power budget that can be offered with a given number of active cores ($TSP_{wc}$). Given a fixed number and sequence of applications, we compare power budgets offered by different mappings based on different mapping strategies. The entry sequence of applications is maintained the same for different mapping approaches for a fair comparison. In case of $TSP_{wc}$, we compare the power budget given by [10] for the number of active cores in the mapping generated by *PAT*. We run our simulations over different network sizes of $16 \times 16$ and $20 \times 20$. In addition, we emulate a varying dark silicon behavior

ranging from 50% darkness through 90% by adjusting initial upper bound on power consumption (TDP). We set the TDP to 177.7W and 277.7W respectively for $16 \times 16$ and $20 \times 20$ network sizes using 22nm technology, keeping the power density constant [36]. ITRS projections present the fact that by the year 2020, computer systems would face 90% dark silicon and that many-core platforms would be in upwards of 512 cores to gain maximum peak performance [4]. Hence, we considered the case of the chip being 90% dark, while also including contemporary projection of 50% dark area. We limit the number of applications entering the system to be in accordance with dark areas. We evaluate the proposed approach over power budget provided per mapping and corresponding throughput. The power budget is computed using TSP library for every mapping, traced with entry of a new application. The ambient temperature is set to 45°C and the safe operating temperature beyond which chip's functionality fails is set to 80°C.

The average (arithmetic mean) and best case percentage gains in power budgets of different mapping configurations using *PAT* strategy over *SC* for different network sizes are presented in Table I. *PAT* achieves a surplus power budget when compared to *SC*, as the active cores are optimally arranged, balancing heat distribution across the chip. In contrast, *SC* tries to map contiguously, leading to tightly packed active cores which get heated up already at lower power consumption, resulting in a lower power budget. In addition to a better power budget, the chip always operates under safe peak operating temperature (80°C), since the budgets are computed through TSP library which manages the upper bound on power avoiding hazardous hotspots. It can be observed that the surplus budget achieved in case of *PAT* increases with increase in amount of dark silicon on the chip. With 90% of the chip being dark, utilizing the remaining fewer number of cores that can originally be powered (active) becomes crucial. *PAT* performs better in such scenarios, given the wider choice of dark cores that can be patterned, while *SC* remains dark silicon agonistic. The gain also increases with increase in network size, once again due to increase in scope of the chip area that can be patterned. Moreover, this is in line with the power budget gain obtained for randomly distributed tasks compared to worst case budget generated by TSP library, as in Figure 3.

In view of future applications like big data, artificial intelligence etc., requiring many-core platforms of larger network sizes, and the issue of dark silicon predicted to grow worse, patterning becomes quintessential to extract higher performance as expected from a many-core system. The potential of patterning could be further better realized with highly scaled up networks and subsequent increase in dark silicon. Nevertheless, the gain in power budget can still be realized at contemporary many-core platforms of relatively smaller network sizes. We simulated $12 \times 12$ network for 90% dark area to observe significant gain in power budget in case of *PAT* compared to *SC* and $TSP_{wc}$. The average and best case (BC) gain in power budget and throughput achieved by *PAT* over *SC* and $TSP_{wc}$ are shown in Figure 7. Some of the low power and safety critical applications follow a strict and conservative approach on setting a single upper bound on power budgets. We account for such applications by comparing our proposed approach against power budget that could result from a worst case mapping configuration for a given number of active cores, generated by [10]. The surplus gained when compared to worst

TABLE I: Surplus Power Budget (in %) of PAT over SC

| Network Size | 90% dark | | 75% dark | | 50% dark | |
|---|---|---|---|---|---|---|
| | Avg. | Best | Avg. | Best | Avg. | Best |
| 16×16 | 5.74 | 13.9 | 4.15 | 11.3 | 2.19 | 7.68 |
| 20×20 | 6.54 | 17.17 | 5.06 | 8.55 | 2.63 | 4.28 |

TABLE II: Surplus Power Budget of PAT over $TSP_{wc}$

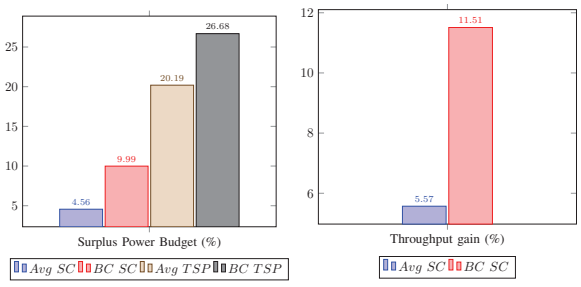
| Network Size | 90% dark | | 75% dark | | 50% dark | |
|---|---|---|---|---|---|---|
| | Avg. | Best | Avg. | Best | Avg. | Best |
| 16×16 | 32.33 | 34.92 | 22.02 | 24.14 | 11.73 | 13.20 |
| 20×20 | 38.70 | 40.83 | 22.40 | 27.4 | 12.50 | 13.33 |

case power budgets for different network sizes and darker chip areas is shown in Table II. Understandably, PAT offers better power budget against worst case values. Although the comparison is against worst case budgets, it still establishes the impact of patterning on mitigating dark silicon to a large extent.

Since a surplus in power budget is gained through *PAT*, it can be utilized to power up more number of cores without violating any safe upper bound on power consumption, which reflects in throughput. The proposed first node selection method chooses diverse regions for different applications which has no impact on individual application's latency. However, few applications using *PAT* might have a lower individual latency at times, due to the compromise on contiguity among tasks of a patterned application. Despite the occasional latency, the overall throughput of the system using PAT still remains higher compared to that of *SC*, as the gain achieved in terms of power budget would (over) compensate for the latency. Gain in throughput for *PAT* compared to *SC* for different mesh sizes and dark regions is presented in Table III. Throughput gain depends largely on surplus budget gained, which in turn can be used to activate more cores. Thus, throughput achieved using PAT strategy follows a similar trend to that of surplus power budget achieved.

TABLE III: Throughput gain for PAT over SC

| Network Size | 90% dark | | 75% dark | | 50% dark | |
|---|---|---|---|---|---|---|
| | Avg. | Best | Avg. | Best | Avg. | Best |
| 16×16 | 7.27 | 15.64 | 4.59 | 13.92 | 2.42 | 8.58 |
| 20×20 | 8.5 | 20.99 | 5.88 | 10.21 | 2.89 | 4.54 |




(a) Surplus Power Budget (in %) (b) Gain in throughput

Fig. 7: Surplus budget and throughput gain of PAT over SC and $TSP_{wc}$ for $12 \times 12$ mesh

## VI. Conclusion and Future Work

In this paper, we presented a dark silicon aware run-time mapping strategy for achieving a better power budget. We implemented our proposed approach in two phases viz., first node selection and patterning based mapping, where we evenly distribute tasks across the chip area to balance heat distribution. This lets the active cores to utilize relatively more power before they reach a maximum limit beyond which chip's functionality is harmed due to thermal violation. As a result, different applications utilize surplus power budgets better to activate more cores and thus gain in performance. Noticeably, the surplus power budget reflects in better resource utilization and throughput, while ensuring thermal safety of the chip, using a software run-time technique, with no hardware overhead. We observed that gain in terms of power budget and throughput increases with increase in network sizes and amount of dark silicon on the chip, stressing the importance of dark silicon aware mappings moving into the future workload characteristics and increase in dark silicon.

We considered our many-core platform to be homogeneous, although having a heterogenous combination of cores with different power-performance characteristics could have more potential. The surplus budget could be better used to activate even more number of cores at lower frequencies and also to run heterogeneous workloads on cores that suit them. Implementing our technique and allocation of surplus budget for a heterogeneous many-core platform is planned for future work.

## Acknowledgment

The authors acknowledge the financial support by the Academy of Finland project entitled "MANAGE: Data Management of 3D Systems for the Dark Silicon Age", University of Turku graduate school (UTUGS) and EU COST Actions IC1103: Manufacturable and Dependable Multicore Architectures at Nanoscale (MEDIAN). The authors want to thank Mr. Mohammad Fattah for setting up NoC based system simulator with dynamic mapping feature. The authors also mention that they used the source code of Lumos from University of Virginia to extract the power data for different technologies.

## References

[1] N. Goulding-Hotta et al. The GreenDroid Mobile Application Processor: An Architecture for Silicon's Dark Future. *IEEE Micro*, 31(2), 2011.

[2] Joonho Kong et al. Recent thermal management techniques for microprocessors. *ACM Comput. Surv.*, 2012.

[3] M.-H. Haghbayan et al. Dark Silicon Aware Power Management for Manycore Systems under Dynamic Workloads. In *ICCD*, 2014.

[4] H. Esmaeilzadeh et al. Dark Silicon and the End of Multicore Scaling. *IEEE Micro*, 32(3), 2012.

[5] Semiconductor Industry Association et al. International technology roadmap for semiconductors (ITRS), 2011 edition. 2011.

[6] A.-M. Rahmani et al. Dynamic Power Management for Many-Core Platforms in the Dark Silicon Era: A Multi-Objective Control Approach. In *Proc. Int. Symp. on Low Power Electronics and Design (ISLPED)*, pages 1–6, 2015.

[7] Carvalho de Souza et al. Dynamic task mapping for MPSoCs. *Design & Test of Computers, IEEE*, 27(5):26–35, 2010.

[8] M. Fattah et al. Smart hill climbing for agile dynamic mapping in many-core systems. In *DAC*, 2013.

[9] M.-H. Haghbayan et al. Mappro: Proactive runtime mapping for dynamic workloads by quantifying ripple effect of applications on networks-on-chip. In *International Symposium on Networks-on-Chip (NOCS)*, 2015.

[10] S. Pagani et al. TSP: Thermal Safe Power: Efficient Power Budgeting for many-core systems in dark silicon era. In *Proc. of CODES+ISSS*, 2014.

[11] Intel Corporation. Intel Xeon Processor - Measuring Processor Power, revision 1.1. In *White paper, Intel Corporation*, April, 2011.

[12] M. Shafique et al. Dark Silicon As a Challenge for Hardware/Software Co-design. In *Proc. of CODES+ISSS*, pages 13:1–13:10, 2014.

[13] Muhammad Shafique et al. The EDA challenges in the dark silicon era. In *Proc. of DAC 2014*, pages 1–6.

[14] Chen-Ling Chou and R. Marculescu. Contention-aware application mapping for network-on-chip communication architectures. In *Proc. of ICCD*, pages 164–169, 2008.

[15] M. Fattah et al. Adjustable contiguity of run-time task allocation in networked many-core systems. In *ASP-DAC*, pages 349–354, 2014.

[16] C. Chen-Ling et al. Energy- and Performance-Aware Incremental Mapping for Networks on Chip With Multiple Voltage Levels. *IEEE Tran. on Computer-Aided Design of Integrated Circuits and Systems*, 27(10), 2008.

[17] A.M. Bender et al. Communication-aware processor allocation for supercomputers: Finding point sets of small average distance. *Algorithmica*, 50(2):279–298, January 2008.

[18] E. Carvalho et al. Heuristics for Dynamic Task Mapping in NoC-based Heterogeneous MPSoCs. In *RSP*, pages 34–40, 2007.

[19] M. Fattah et al. CoNA: Dynamic application mapping for congestion reduction in many-core systems. In *Proc. of ICCD*, pages 364–370, 2012.

[20] M. Deveci et al. Exploiting geometric partitioning in task mapping for parallel computers. In *Proc. of Parallel and Distributed Processing Symposium, 2014*, pages 27–36. IEEE, 2014.

[21] S. Shintaro et al. Pattern-based systematic task mapping for many-core processors. In *International Conference on Networking and Computing*, pages 173–178. IEEE, 2010.

[22] A. Coskun et al. Temperature management in multiprocessor socs using online learning. In *Proc. of DAC*, pages 890–893. IEEE, 2008.

[23] L. Thiele et al. Thermal-aware system analysis and software synthesis for embedded multi-processors. In *Proc. of DAC*, pages 268–273. ACM, 2011.

[24] Y. Liu et al. Energy and thermal aware mapping for mesh-based noc architectures using multi-objective ant colony algorithm. In *Proc. of ICCRD*, volume 3, pages 407–411, 2011.

[25] A. Coskun et al. Utilizing predictors for efficient thermal management in multiprocessor socs. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 28(10):1503–1516, 2009.

[26] Min Bao et al. Temperature-aware task mapping for energy optimization with dynamic voltage scaling. In *Proc. of DDECS*, pages 1–6, 2008.

[27] T. Komoda et al. Power capping of CPU-GPU heterogeneous systems through coordinating DVFS and task mapping. In *ICCD*, pages 349–356, 2013.

[28] MB Taylor. Is dark silicon useful?: harnessing the four horsemen of the coming dark silicon apocalypse. *Proc. of DAC, 2012*, pages 1131–1136.

[29] Intel Corporation. Fourth Generation Mobile Processor Family Data Sheet. In *White paper, Intel Corporation*, July, 2014.

[30] AMD. AMD Kaveri APU A10-7800. Accessed: 2015-02-28.

[31] Dobkin et al. Searching for empty convex polygons. *Algorithmica*, 5(1-4):561–571, 1990.

[32] Harvey E. White. *Modern College Physics*. Van Nostrand, 1948.

[33] TGG: Task Graph Generator. *URL: http://sourceforge.net/projects/taskgraphgen/*, 2010.

[34] S. Li et al. McPAT: An Integrated Power, Area, and Timing Modeling Framework for Multicore and Manycore Architectures. In *MICRO*, 2009.

[35] F. Fazzino et al. Noxim: Network-on-chip simulator. *URL: http://sourceforge.net/projects/noxim*, 2008.

[36] L. Wang and K. Skadron. Dark vs. Dim Silicon and Near-Threshold Computing Extended Results. In *University of Virginia Dept. of CS Technical Report TR-2013-01*, 2012.

# Paper II

# adBoost: Thermal Aware Performance Boosting through Dark Silicon Patterning

Anil Kanduri, Hashem Haghbayan, Amir Rahmani, Muhammad Shafique, Axel Jantsch and Pasi Liljeberg

Published in IEEE Transactions on Computer, 2018, Volume 67, Issue 8, pp: 1063-1077

# adBoost: Thermal Aware Performance Boosting Through Dark Silicon Patterning

Anil Kanduri, *Student Member, IEEE*, Mohammad-Hashem Haghbayan, *Student Member, IEEE*, Amir M. Rahmani, *Senior Member, IEEE*, Muhammad Shafique, *Senior Member, IEEE*, Axel Jantsch, *Member, IEEE*, and Pasi Liljeberg, *Member, IEEE*

**Abstract**—Increasing power densities of many-core systems leaves a fraction of on-chip resources inactive, referred to as dark silicon. Efficient management of critical interlinked parameters - power, performance and temperature can improve resource utilization and mitigate dark silicon. In this paper, we present a run-time resource management system for thermal aware performance boosting using a dark silicon aware run-time application mapping strategy. The mapping policy patterns inactive cores among active cores for relatively lower and even distribution of operating temperatures. This provides enough thermal headroom for boosting the frequency of active cores upon performance surges and allows sustained boosting periods, improving the performance further. We design a controller for thermal aware performance boosting that decides on efficient allocation utilization of power budget and thermal headroom obtained from patterning. Our strategy yields up to 37 percent better throughput, 29 percent lower waiting time and up to 2 $\times$ longer boosting periods, in comparison with other state-of-the-art run-time mapping policies.

**Index Terms**—Dark silicon, run-time mapping, dynamic power and thermal management

## 1 Introduction

Power densities of may-core systems are increasing with disproportional voltage and technology node scaling, contributing to higher on-chip temperatures [1], [2]. As a result, a section of the chip has to remain inactive to function within safe thermal limits. This inactive circuitry of the chip is termed as Dark Silicon [3]. Computer systems cap the power consumption at a fixed upper bound to ensure that the on-chip temperature due to power dissipation is within a safe limit. Thermal design power (TDP) is a widely used design time estimate of the upper bound on power consumption that ensures thermal safety. TDP is a conservative estimate with the assumption that the chip is operating at worst case voltage and frequency levels, and workloads [4]. An efficient alternative to TDP is Thermal Safe Power (TSP), a variable run-time estimate of upper bound on power, calculated dynamically as a function of number of active cores and their spatial alignment [5]. Considering the mutual thermal effect of neighboring active and inactive cores, application mapping can determine the maximum utilizable power budget. Using TSP instead of TDP presents a scenario where utilizing the same power budget can result in different temperatures for different mappings. State-of-the-art application mapping policies that handle thread-to-core allocation prefer i) a contiguous mapping scheme to minimize inter-core communication latency [6] and ii) the set of cores chosen for mapping an application to be regular geometric structures such that potential incoming applications do not suffer with a dispersed set of available cores and to avoid inter-application congestion. At lower technology nodes, conventional mapping approaches result in denser mappings which accumulate temperature faster, limiting the maximum utilizable power budget within safe thermal limits. In contrast, sparse mappings that compromise on contiguity can provide higher utilizable power budget before reaching the critical temperature. With shrinking transistor sizes and increasing power densities, mapping policies have to be adapted to maximize performance through efficient power budget utilization.

### 1.1 Implications of Mapping on Power Density

We demonstrate the effect of mapping on temperature accumulation through the following example. Consider an application with 8 threads, run on a 16-core system arranged in a 4×4 mesh, as shown in Fig. 1a. Each core is modeled based on Niagara-2 architecture at 22nm technology, power and performance metrics are extracted from Lumos analytical framework [7]. The application is executed on our in-house cycle accurate simulator (see Section 4). All the 8 threads are mapped in a dense manner on 8 cores and the remaining cores are power gated. Each thread consumes an average power of 5.95 W, resulting in total active power of 47.6 W (5.95×8). We assume the critical temperature of the chip to be 80 °C. The final steady state temperature of the system after running the application for 100 ms is shown in Fig. 1a. This mapping configuration has

- *A. Kanduri, M.-H. Haghbayan and P. Liljeberg are with the University of Turku, Turku 20500, Finland. E-mail: {spakan, mohhag, pakrli}@utu.fi.*
- *M. Shafique and A. Jantsch are with the TU Wien1040, Vienna, Austria. E-mail: {muhammad.shafique, axel.jantsch}@tuwien.ac.at.*
- *A. M. Rahmani is with the University of California, Irvine, CA 92617, also with the TU Wien1040, Vienna, Austria. E-mail: amirr1@uci.edu.*

*Manuscript received 31 July 2017; revised 26 Dec. 2017; accepted 26 Jan. 2018. Date of publication 14 Feb. 2018; date of current version 7 July 2018. (Corresponding author: Anil Kanduri.)*
*Recommended for acceptance by Y. Xie.*
*For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.*
*Digital Object Identifier no. 10.1109/TC.2018.2805683*

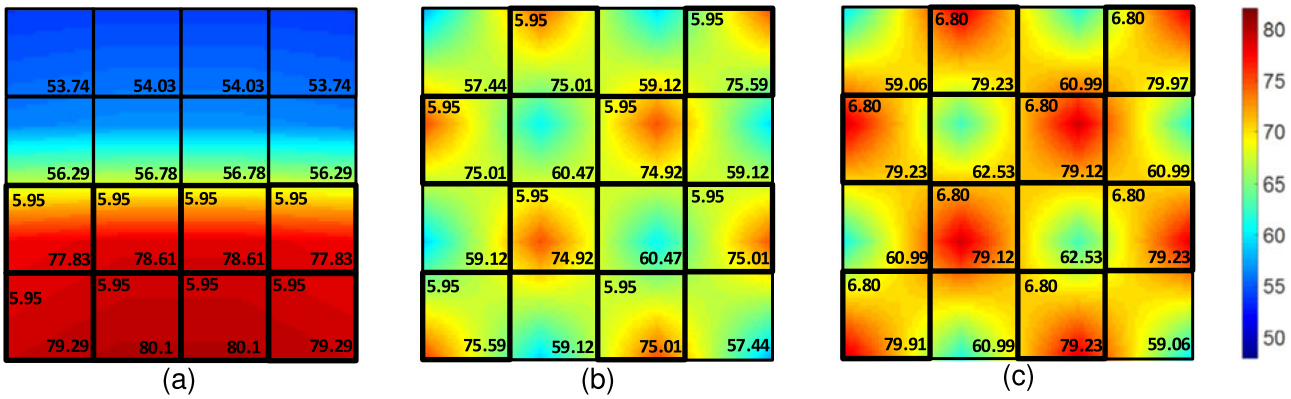



Fig. 1. Effect of mapping on temperature and power budget for a 16-core system arranged in 4x4 mesh. Each block represents a tile with power consumption (W) in top left corner and final steady state temperature in bottom right corner (a) Dense mapping, (b) Sparse mapping that consumes same power as dense mapping at lower temperature, (c) Sparse mapping that utilizes more power than dense mapping at same temperature

resulted in the chip reaching its critical temperature after consuming 47.6 W of power. For the same application and set up, consider the mapping configuration in Fig. 1b. The same application is mapped sparsely, such that each active core has an idle neighbor. With cooler neighboring cores, temperature accumulation of active cores is slowed down. For same amount of power consumption, the sparse mapping has resulted in almost 4°C lower final steady state temperatures, showing the effect of mapping on temperature.

With critical temperature set to 80°C, available thermal headroom can be utilized i) to power up more cores, ii) to run active cores at a higher frequency i.e., boosting, iii) to lower overall on-chip temperatures. This allows both per-application performance to be improved by boosting, and per-chip throughput by activating more cores. Consider the final steady state temperature of the same application with sparse mapping configuration as discussed earlier in Fig. 1c, which is boosted within thermally safe limits. With the sparse mapping configuration, the chip has reached critical temperature after each core consuming 6.8 W of power. The total power that can be consumed with sparse mapping (6.8 $\times$8=54.4 W) is higher than that of the dense mapping, yielding a 6.8 W of surplus power budget. The sparse mapping configuration in Fig. 1b has 5.96 percent lower temperature and the one in Fig. 1c has 14.2 percent of surplus power budget, over the dense mapping configuration. Assuming that each thread consumes 5.95 W of power as in Fig. 1a, the surplus budget can be used to activate at least one more core on the chip, increasing the utilization to 12.5 percent, thus mitigating dark silicon. We establish that the performance and utilization of applications can be improved by considering sparsity in application mapping, which provides power headroom for activating more cores and/or thermal headroom for boosting the frequency of active cores. We refer to the sparse mapping approach which aligns cool inactive cores among hot active cores as Dark silicon patterning [8].

### 1.2 Impact of Dark Silicon Aware Mapping and Boosting on Performance

The thermal headroom provided with dark silicon patterning can be leveraged to boost the frequency over short bursts of time to improve performance while minimizing the likeliness of thermal violation. We refer to this technique as boosting, distinguishing from conventional frequency scaling for performance enhancement [9]. We define boosting period as the time taken for a power violation to translate into a thermal violation. Boosting period largely depends on available thermal headroom, on-chip temperatures, the rate at which temperature accumulates and frequency to be boosted to. In addition to these factors, power-performance characteristics of an application and the combination of other concurrent applications also effect the boosting period. With hot active cores neighbored by cool inactive cores, dark silicon aware mapping is likely to result in higher thermal headroom and importantly a lower rate at which temperature accumulates with power. Dark silicon aware mapping thus improves the performance gains from boosting, which necessitates considering mapping and boosting together at run-time. We provide insights into the compound effect of dark silicon aware mapping on boosting with the following example.

*Motivational Example.* Consider a synthetic application of 8 threads, executed on a similar set up mentioned in Section 1.1 with both dense and sparse mappings. Fig. 2 shows the rate at which transient temperature accumulates for the application with dense and sparse mappings over 100 ms of execution time. The thermal profile was extracted using HotSpot 6.0 [10], with its default configuration. In both cases, the application started execution with same initial temperature of 70 °C. Temperature accumulates faster with dense mapping, with each active core's temperature compounded by neighboring core's temperature. In contrast, temperature accumulates relatively slower with sparse mapping, due to the interleaved inactive cores minimizing the effect of temperature among active cores. After 100 ms of execution, there is about 2 °C of difference between the temperatures of both the mappings. Assume a performance

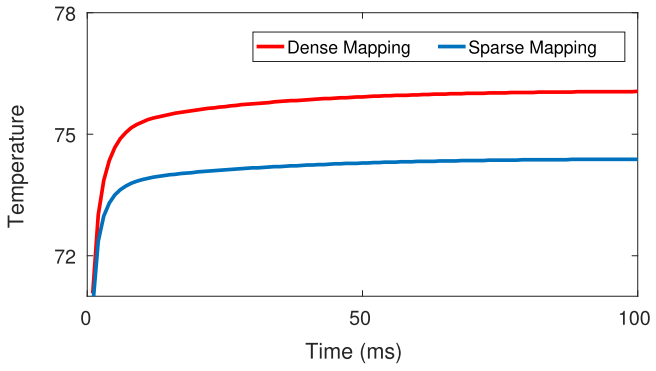



Fig. 2. Effect of mapping on temperature accumulation.

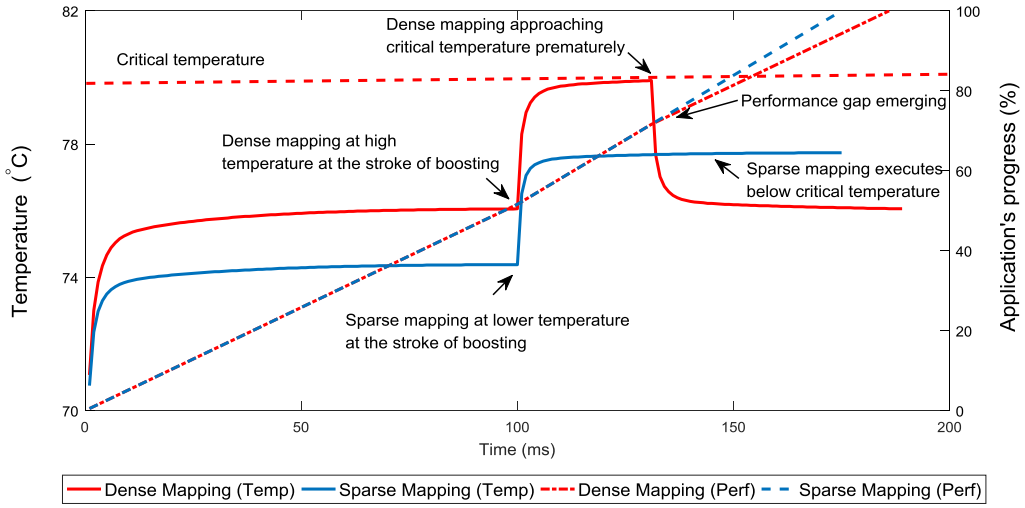



Fig. 3. Effect of mapping on boosting and performance. Thermal profile and progress of execution are shown for dense and sparse mappings.

surge requirement at 100 ms, which is met by boosting the frequency of active cores from 3 GHz to 3.75 GHz. The sudden increase in frequency causes increase in power and temperature, as shown in Fig. 3. At time t=100 ms, the dense mapping is operating at about 76 °C, while sparse mapping is operating at 74 °C. With dense mapping being at a higher temperature when boosting is invoked, subsequent temperature accumulation towards critical temperature is faster, lowering the period of boosting. The dense mapping reaches the critical temperature of 80 °C at 131 ms, after sustaining a boosting period of 31 ms. The frequency is reduced back to 3 GHz to prevent potential thermal violation. The application finished execution eventually after 189 ms. On the other hand, the sparse mapping, which is executing at a lower temperature when boosting is invoked, sustains 75 ms of boosting - about 2x longer than that of the denser mapping. As a result, there is a significant performance gain during the period of boosting and the application finishes execution by 175 ms. The sparse mapping configuration could sustain 41 percent longer period of over-boosting which reflects in 8 percent of performance as compared to the dense mapping. The sparsely mapped application remained well below the critical temperature, providing thermal headroom for longer periods and/or higher levels of boosting. Our analysis demonstrates that mapping applications sparsely followed by boosting maximizes performance. This also avoids frequent oscillation between frequency up and down scaling that densely mapped applications suffer with, due to faster temperature accumulation.

### 1.3 Our Contributions

In our previous work [8], we proposed dark silicon patterning, a run-time mapping technique to increase utilizable power budgets by balancing heat distribution evenly across the chip. We used the surplus power budget to improve chip throughput by activating more cores. Dark silicon patterning reduces the effect of temperature accumulation among neighboring active cores, offering thermal headroom for active cores' frequency to be boosted before reaching critical temperatures. In this work, we focus on improving performance within thermal constraints by utilizing the thermal headroom from dark silicon patterning to boost the frequency of active cores upon performance requirements. We propose *adBoost*, a thermal aware adaptive boosting technique that considers both mapping and boosting decisions together for creating enough thermal headroom for sustainable periods of boosting. We split our approach into 2 steps viz., 1) dark silicon patterning based mapping and power management - to create power and thermal headroom, followed by 2) thermal aware adaptive boosting - to efficiently utilize thermal headroom created. We augment our previous run-time controller for power management that supports dynamic application mapping and power allocation with the proposed boosting controller, *adBoost*. The *adBoost* controller receives feed-back on instantaneous temperature from the system to make decisions frequency scaling decisions upon performance requirements for efficient utilization of available thermal headroom obtained with patterning. Existing works on power budgeting [11], dark silicon aware power management [12], [13], dark silicon aware interconnect designs [14], [15], frequency boosting [16] have considered each of their focus aspects and scenarios independently, without considering the effect of each technique on subsequent inter-related aspects. In our work, we integrate application mapping - to maximize power budgets and minimize on-chip temperatures, power allocation - to efficiently utilize the surplus budget provided with mapping, and boosting - to leverage thermal headroom offered with sparse mapping for performance improvement, honoring thermal constraints. Our contributions are as follows:

- A run-time mapping technique for dark silicon patterning to evenly distribute heat across the chip that improves power budget.
- Design, analysis and implementation of thermal aware performance boosting mechanism through frequency scaling.
- Feedback controllers for efficient allocation of surplus power and thermal headroom to improve per-chip throughput and/or per-application latency.
- Design and implementation of a system manager that integrates the run-time mapping unit - for surplus power budget, power management - for power allocation decisions and boosting mechanism - for utilizing the thermal headroom available.

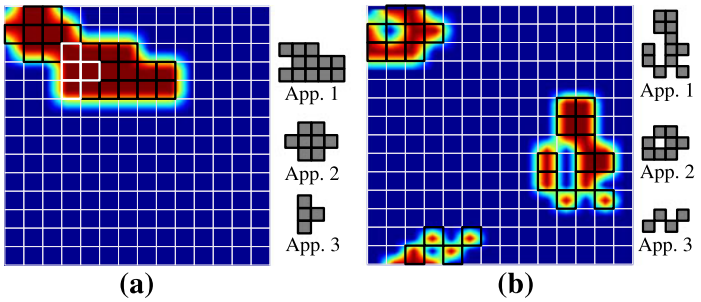



Fig. 4. Effect of inter-application mapping on temperature accumulation. (a) Contiguous mapping of 3 applications, with their individual mapping configurations. (b) Sparse mapping of 3 applications, with their individual mapping configurations and percentage of packet latency penalty for mapping sparsely, normalized against contiguous mappings.

## 2 Dark Silicon Aware Mapping

We prefer run-time mapping as a pro-active choice for even distribution of heat and surplus power budget gains, as opposed to reactive dynamic power management techniques. Our dark silicon aware mapping solution considers spatial alignment of active and dark cores with the objective of maximizing utilizable power budget. In the following sections, we elaborate our mapping strategy.

### 2.1 Inter-Application Sparsity

While conventional mapping approaches prefer contiguity and regular geometric structures for performance and minimal congestion, we propose sparsity for even heat distribution and minimizing hot spots. Two factors that distinguishes a sparse mapping from a contiguous mapping are spatial distribution of applications and sparsity among tasks of each application. Consider the heat dissipated by a core $C_i$, which is a 3-tuple ($P_i$, $T_n$, $T_{amb}$), where $P_i$ is the power dissipated by the core, $T_n$ is neighbouring cores' temperature and $T_{amb}$ is the ambient temperature. Contiguous mappings accumulate heat faster with neighboring cores' temperature effecting the active cores (higher $T_n$), where as the sparse mapping we propose benefits from cooler inactive neighboring cores (lower $T_n$).

We demonstrate the effect of inter-application spatial alignment on temperature and power budget through another example, presented in Fig. 4. Three applications App1, App2 and App3 with 12, 8 and 4 tasks respectively are running on the system. Thermal profiles of these 3 applications on a 144-core system with a contiguous mapping is shown in Fig. 4a. The power budget of this mapping configuration computed using TSP library [5] is 66W. Thermal profiles of the same applications running on the same system with sparse mapping configuration (at both inter-application and intra-application levels) is shown in Fig. 4b. This mapping configuration provides a power budget of 74.6W, as calculated by TSP library. An improvement of 8.6W in power budget can be observed for the spread-out, patterned mapping compared to tightly packed and contiguous mapping. Contiguous mapping results in higher on-chip temperatures and lower utilizable power budget, with temperature accumulating among neighboring active cores and applications. Sparse mapping on the other hand has lower on-chip temperatures and higher utilizable power budget, with both inter-application and inter-task sparsity. Assuming that some of the applications require performance surges at run-time, boosting them for sustained periods depends on both neighboring active cores within the applications as well as other (spatially closer) concurrently running applications. In view of these factors, we split our mapping approach into two phases viz., selecting a region that is spatially dispersed from current set of applications running on chip and mapping tasks of the application sparsely such that active and inactive cores are patterned in the selected region. *Latency Vs Performance Gain*: The contiguous mapping (Fig. 4a) has a per-core budget of 3 W, powering up 22 cores with 66 W. Assuming this as the baseline, the sparse mapping (Fig. 4b) could power 74.6/3 = 24.7 cores with the surplus budget gained through patterning, mitigating dark silicon by 12.2 percent. The average packet latency in each patterned applications App1, App2 and App3 is 1, 3.5 and 3.8 percent (shown in Fig. 4b) more than that of the contiguous mappings respectively. Despite the dispersion and increased inter-task communication latency, sparse mapping has power budget gain of about 12 percent compared to dense mapping and thermal headroom for potentially boosting the applications. It is also to be noted that the inactive cores are the inevitable dark cores - instead of leaving them out naively, we use them to balance out heat distribution and gain power budget.

### 2.2 First Node Preferences

We split our mapping approach into two phases - first node selection, followed task allocation, as proposed in [6]. *First node selection* is to choose a free core on the chip around which the tasks of an application can subsequently be mapped to fulfill power, performance and/or latency objectives. Based on our analysis presented in Fig. 4, we prefer nodes that are far from active cores as suitable candidates for first node - to minimize the compound effect of temperature. Our objective is to find a *free* and *sparse* first node which has: i) enough number of free nodes around it to allocate tasks of the incoming application with minimal internal congestion and ii) minimal effect of temperature due to neighboring (active) cores.

*Free First Node.* We modify the first node selection strategy *MapPro,MapPro*, in order to meet our first objective of finding a *free* first node. MapPro uses Vicinity Counter (*VC*) as a metric to represent number of free cores around a given node within in a given region. The node with a relatively higher *VC* value is preferred as the first node such that it can fit all the tasks of an application within the region around it. It should be noted that *VC* of node is relative to the size of the region i.e., the same node has different VC values over different regions. For example, a node within a 5×5 (25 cores) sized region of free cores has 24 free cores around it, whereas the same node within a 3×3 (9 cores) sized region has 8 free cores around it. The *VC* parameter also represents internal congestion around a given node by considering the number of occupied nodes around it - intuitively reflecting the likeliness of congestion, if mapped around that node. Implementation details of MapPro are elaborated in [17].

*Sparse First Node.* To meet our second objective of minimizing mutual heat effect among active cores of concurrently running applications, we prefer nodes that are farther from (if any) currently active cores as candidates for the first node.

The temperature effect of every active core on other cores of the chip decreases exponentially, with distance from the active (hot) core. In other words, the greater the distance from an active core, the lesser the effect of heat from it. In order to minimize this heating effect from active cores of other applications, we prioritize nodes that are as far as possible from such active cores. Inspired by the surface tension phenomenon [18], where energy distribution of a flat surface turns into a curved surface with applied surface pressure, we model the effect of temperature of an active core on other cores of the chip in a similar fashion. We define Distance Factor (*DF*) to quantify the effect of temperature of an active cores on other cores on the chip, as follows: *Definition: Distance Factor, $DF_{ij}$, for a node located at (i,j) is the weighted sum of impact of distance from all the other occupied nodes located at (x,y) such that* $(x, y) \in Mesh$

$$DF_{i,j} = \sum Wn_{i,j} \times (e^{-\alpha(d_{ij-xy})}), \tag{1}$$

where $Wn_{ij}$ is the weight of node $n_{ij}$, $d_{ij-xy}$ is distance from nodes located at (i,j) and (x,y) and $\alpha$ is the mesh size. We augment MapPro's choice of *free first node* with the *sparse first node* to find a suitable candidate for the *first node*. The algorithmic flow of first node selection is shown in Algorithm 1 which is inspired by our previous work presented in [17]. Whenever a new application enters the system, the average power budget required for its execution is estimated. Subject to the availability of power budget, the application is serviced by finding a suitable first node based on *VC* and *DF* values (lines 1-3). Number of tasks of the application determines the region size over which *VC* values will be computed. The node with maximum *VC* value, $maxVC$, among all nodes is chosen as the first node, and thus also selecting the polygon (preferably square) centered at $maxVC$. The chosen first node and the application are passed to patterning phase (line 4). Mapping the tasks based on patterning is explained in the following section and Algorithm 2. Mapping a new application alters the *VC* and *DF* values for all the nodes on the chip, which need to be updated. After mapping an application, we calculate the new *VC* and *DF* values for all the nodes, to be used for choosing the first node for subsequent applications. VC values for different sizes of regions are updated (lines 6-23) to determine the new $maxVC$ node. Pro-active *VC* and *DF* calculation eases first node selection for next incoming application (based on number of tasks of the application) and reduces first node selection overhead. Conflict between nodes having same VC value is resolved by considering corresponding *DF* values (lines 16-18). Status of the nodes running an application is reset to *free*, once the application leaves after finishing its execution. The *VC* and *DF* values are updated as per the altered mapping configuration.

### 2.3 Dark Silicon Patterning

We prefer mapping tasks of the application in a sparse manner to minimize potential hot-spots. We define the likeliness of an active core effecting the temperature of its neighbors through its sparsity. Sparsity of a node $n_{ij}$ represents the number of free nodes that are neighboring it in four cardinal directions (North, East, West, South). The Sparsity Factor ($SF_{ij}$) for a node $n_{ij}$ located at $(i, j)$ is expressed as:

$$SF_{i,j} = \sum_{i=1}^{4} \sum_{j=1}^{4} F(i + i', j + j'), \tag{2}$$

where $i'$ = [0, 1, 1, -1], $j'$ = [-1, 0, 1, 1]. $F(i, j)$ denotes if a node located at $(i, j)$ is free or not, such that

$$F(i,j) = \begin{cases} 1 & \text{if } n_{i,j} \text{ is unoccupied} \\ 0 & \text{if } n_{i,j} \text{ is occupied.} \end{cases}$$

**Algorithm 1.** The Mapping Algorithm

**Inputs:** $newApp$: New application, $budget$: Avail. power budget;
**Outputs:** $Q$: Mapping;
**Constants:** $M$: Size of the mesh, $groups$ : Number of square groups = $[(\sqrt{M}-1)/2]$, $maxRadius$: Maximum radius of the square ($(\sqrt{M}-1)/2$), $\alpha$:Mesh size parameter $\sqrt{M}$, $P_{avg}$: Average power consumption per node;
**Global Variables:** $VC$ : Vicinity Count of a node, $maxVC$ : Node with the maximum VC, $firstNode$ : Selected first node for mapping, $DF$ : Distance factor;

**Body:**
1: $appPredictedPower \leftarrow |newApp| \times P_{avg}$;
2: **if** $appPredictedPower \leq budget$ **then**
3: $firstNode \leftarrow maxVC_{[(\sqrt{appSize}-1)/2]}$
4: $Q \leftarrow pattern(firstNode, newApp)$;
5: //Updating VC and DF values after mapping
6: **for** each $n_{xy} \in newApp$ **do**
7: **for** each core $n_{ij}$ located in Row $i$ and Column $j$ **do**
8: $r' = maximum(|i - x|, |j - y|)$;
9: $DF_{ij} -= e^{-\alpha r'}$;
10: **for** $r$ = 1 to $maxRadius$ **do**
11: **if** $r - r' \geq 0$ **then**
12: $VC^r_{ij}$ -= $r$ - $r'$;
13: **if** $VC^r_{ij} > maxVC_r$ **then**
14: $maxVC_r \leftarrow VC^r_{ij}$;
15: **else**
16: **if** $VC^r_{ij} = maxVC_r$ and $DF_{ij} > DF_{maxVC_r}$ **then**
17: $maxVC_r \leftarrow VC^r_{ij}$;

**Algorithm 2.** Patterning

**Inputs:** $App$: Application, $firstNode$: Selected First Node.
**Global Variables:** $S$ Selected square, $SF$: Sparsity Factor for each node in a square, $maxSF$: Node with the maximum SF in a square, $currentNode$: Node onto which current task is being mapped;
**Global Constants:** $Tasks$ : Vector of tasks of the application $App$, $size$: Radius of square close to size of $App$

**Body:**
1: $S \leftarrow Square^{size}_{firstNode}$;
2: $Tasks = sort(App)$;
3: **while** $App \neq \emptyset$ **do**
4: **for** each $t_i \in Tasks$ **do**
5: $currentNode \leftarrow maxSF$;
6: $map\ (t_i) \rightarrow maxSF$;
7: $App - t_i$ ;
8: $S - currentNode$ ;

Following the first node selection, we build a polygon $P$ (preferably a square), which is a set of all the un-occupied nodes around the selected first node. Mapping an application

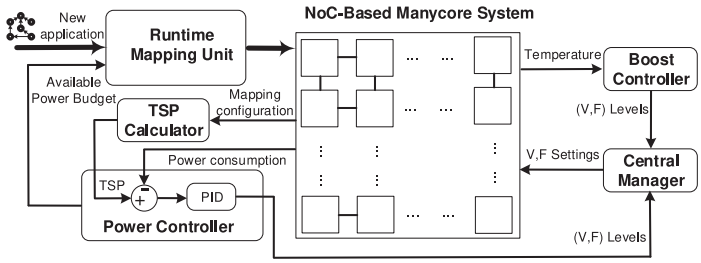



Fig. 5. System architecture.

in this region $P$ would have lesser effect of temperature from other applications. Within the selected region, we prioritize the nodes that are more sparse to map tasks of the application. As a result, sparse nodes are occupied first, leaving out the denser nodes to be patterned in a way that they provide necessary cooling effect needed by their active neighbors. While mapping, we sort the tasks of the application as per their communication volume. We choose the task with highest communication volume and map it on to the first node, the most central and sparse node such that proximity with other intercommunicating tasks is higher. We proceed to the task with next highest communication volume by sorting all the remaining tasks as per their total communication volume and map it onto the node with the highest $SF$ among the nodes in the selected region $P$. We continue in a similar order of tasks - based on communication volume and similar order of nodes - based on sparsity, such that tasks with higher communication gets mapped onto nodes with higher sparsity, until all tasks of the application are mapped. Mapping tasks with higher communication volume in the order of sparsity allows corresponding communicating tasks to be in proximity of the high communication volume tasks, minimizing the weighted Manhattan distance for inter-task communication. Throughout the mapping procedure, dense nodes have least priority, which would remain un-occupied among the other occupied nodes of the region. This provides the necessary cooling effect for improved power budget utilization.

The algorithmic flow of patterning is shown in Algorithm 2. Mapping starts at $firstNode$, selected from Algorithm 1 by choosing square region ($Square^{size}_{firstNode}$) such that application ($App$) can fit within this region (line1). Tasks ($Tasks$) of the application are sorted as per their communication volume (line 2). The most expensive task as per communication is mapped onto the node with maximum sparsity factor ($SF$), $maxSF$ (lines 5-6). The mapped task is removed from the list of tasks to be mapped and the mapped node is removed from the list available nodes in the square (lines 7-8). This procedure is repeated until all the tasks of the application are mapped. The $SF$ value for nodes in selected square changes with every occupied and thus new $maxSF$ is computed for every unmapped task.

## 3 System Design

Our system design consists of three phases - i) run-time mapping that maximizes utilizable power budget, ii) power controller that utilizes surplus power budget to activate more cores (if possible) and iii) boosting controller that increases frequency beyond the base frequency subject to thermal safety. The hierarchical view of the system architecture is shown in Fig. 5. We first present the work flow of the framework, followed by details on individual modules in the following sections. *Work flow.* The run-time mapping unit (RMU) is invoked upon on arrival of new applications and is responsible for task-to-core allocation. The RMU chooses appropriate first node and subsequently finalizes task-to-core mapping. The TSP Calculator estimates safe upper limit for power consumption, based on number of simultaneously active cores using the mapping configuration. Once a new application is mapped and/or a currently running application exited the system, upper bound on power is re-evaluated using the TSP calculator. The upper bound is set until further variation in mapping due subsequent entry/exit of an application. The power controller handles power actuation decisions by monitoring current power consumption and comparing against safe limit and communicates the DVFS and power gating settings to the central manager. Power controller is invoked over the parametric power epoch $E_{power}$. The boost controller makes boosting decisions upon performance requirements of applications and sends the corresponding frequency level and application information to the central manager. The boost controller is invoked over the boosting epoch $E_{boost}$. It is to be noted that power controller decisions are based on power headroom available whereas boost controller decisions are based on thermal headroom. DVFS settings from the power controller are proportional to the surplus power budget available. Boost controller's decisions rely on temperature feedback, irrespective of power headroom. We set $E_{power}$ as $4 \times E_{boost}$, such that boosting decisions are made conservatively to avoid thermal violation. This minimizes the oscillation of voltage and frequency scaling decisions among power and boost controllers. Both power controller and boost controller act independently over their respective epochs and communicate their decisions to the central manager. The central manager enforces voltage and frequency levels and power gating settings as per the decisions made by power controller and the boost controller by arbitrating between them. Fig. 6 shows modular work flow of different control units of our approach. Individual modules of the system are detailed in following sections.

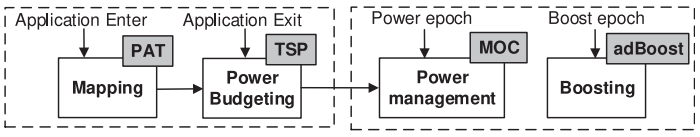



Fig. 6. Workflow of proposed framework.

### 3.1 Run-Time Mapping Unit

Applications arriving dynamically are serviced by the Run-time Mapping Unit (RMU) by finding active cores around a suitable first node, followed by task-to-core mapping. The RMU presented in Fig. 5 uses the pro-active approach, MapPro [17], for first node selection and dark silicon patterning based mapping (PAT) [8] for application mapping. The objectives and algorithmic flow of patterning based mapping have been elaborated in Section 2.3. Detailed description of on first node selection and dark silicon patterning are presented in our previous works [17] and [8] respectively. Each task of an incoming application holds computation and communication factors, indicating their relative amount of computation and inter-task communication

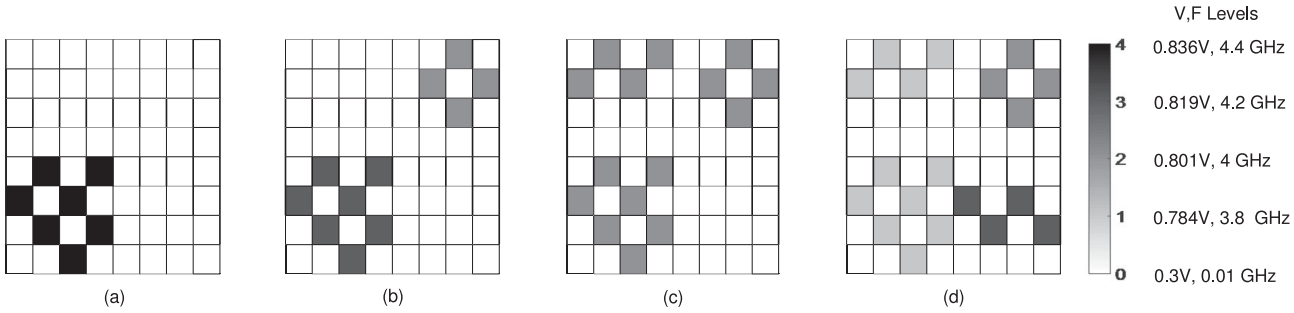



Fig. 7. Varying DVFS levels using the power controller with dynamic workloads. (a)-(d): Mapping and DVFS level of applications, ranging from 1 to 4 concurrent applications of 7, 4, 4 and 4 tasks per application respectively.

volumes. RMU estimates the average power consumption of an incoming application as the accumulated sum of power consumption of task-per-core, under nominal conditions. RMU receives the available power budget from the power controller and compares it with the estimated average power consumption of the incoming application. RMU decides to map the new application with sufficient power budget being available, while the application waits in the queue in case of limited power budget. A possibility of avoiding longer application waiting time is by scheduling a sub-set of independent tasks that would fit within available power budget. This approach requires application level information on inter-task dependencies, and availability and choice of appropriate set of independent tasks that would fit within the limited power budgets.

### 3.2 Power Controller

The power controller is responsible for power management and allocation decisions, as shown Fig. 5. The TSP Calculator (in Fig. 5) receives current mapping configuration to dynamically calculate the thermal safe power (TSP). TSP is evaluated whenever there is a change in mapping configuration i.e., upon arrival of new application or exit of a currently running application. We use the light weight open source TSP library for power budget estimation [11]. The power controller estimates available power budget as the difference between the monitored power consumption and the safe upper bound determined by TSP Calculator. We use a PID controller to determine voltage and frequency level settings of cores, such that they are proportional to the available power budget. In case of a power violation, i.e., power consumption exceeding TSP, the power controller scales down the (V,F) levels proportionally to reduce the power consumption to a safe limit. With power constraints honored i.e., power consumption below TSP, the power controller allocates the surplus budget in two possible ways, i.e., either to map new applications (if any), or to throttle currently running applications. When there is a new application waiting in the queue for execution, the surplus budget is used to map the new application, provided that the available power budget is enough to accommodate the new application. With no new application arrival or insufficient amount surplus budget to fit the new application, the available budget is used to throttle currently running applications. In this scenario, the power controller scales up the (V, F) levels of the running application(s), corresponding to the amount of power budget available. In case of throttling current applications, the power controller also considers applications' priority level and network characteristics to determine suitable candidates among concurrently applications to actuate DVFS. In case of up/downscaling, voltage and frequency levels determined by the power controller are received by the central manager, which enforces DVFS settings for power actuation. Algorithmic implementation, decisions on power allocation and choice of suitable candidates for DVFS are detailed in our previous work [19].

*Dark and Dim Silicon Considerations.* We present a working scenario with dynamic entry and exit of applications, creating a variable workload. This practically leads to both dark silicon and dim silicon baselines, as shown in Figs. 7a, 7b, 7c, and 7d, with entry of four applications consecutively. In Fig. 7a, one application (App1) of 7 tasks being mapped at high (V,F) levels. As another application (App2) of 4 tasks enters, power budget is estimated and has to be shared among the two applications. This results in downscaling the App1 to accommodate both App1 and App2 within the safe limits. As new applications continue to arrive, the power budget is allocated accordingly using DVFS. Fig. 7d shows the scenario where there are 4 concurrent applications running on the chip. Available power budget has to be shared among all the applications by throttling down certain applications as per their priority requirements and network characteristics. With several concurrent applications, the system execution is inclined towards dim silicon, whereas relatively smaller number of concurrent applications presents dark silicon scenario. Power budget gains from patterning the applications is higher with dark silicon, as compared to a dim silicon baseline.

### 3.3 adBoost Controller

The boost controller is responsible for boosting the frequency of active cores, subject to application requirements and thermal safety. The controller is invoked over an epoch $E_{boost}$. Transient and steady state per-core temperature values are calculated using HotSpot 6.0 with its default configuration over every epoch $E_{boost}$. Trace file containing per-core power values is obtained from the power controller. In every epoch, it monitors on-chip per-core temperatures to estimate the thermal headroom $\triangle T$, calculated as the difference between critical temperature ($T_{crit}$) and maximum temperature ($T_{max}$) among active cores. When an application requests for performance improvement, the controller evaluates $\triangle T$ to handle boosting process. Fig. 8 shows the control flow of the boost controller subject to the thermal headroom available. The algorithmic flow of boosting is presented in Algorithm 3. $\triangle T > \mathbf{0}$ When there is enough

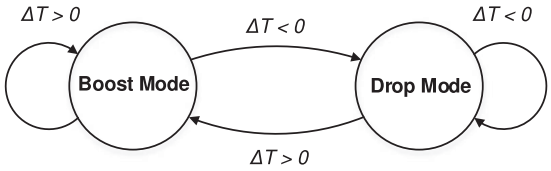



Fig. 8. Control flow of boosting. $\triangle T$ is the thermal headroom. Boost and drop modes represents up and down scaling of frequency.

thermal headroom ($T_{crit} > T_{max}$), the controller can boost the frequency upon an application's request for performance. With a specific application's request boosting, the controller chooses to increase the frequency of cores running the application by one *level*. Frequency of each core among the set of cores running the application is boosted within in one epoch (line 9, Algorithm 3). If the performance requests still persist, frequency of chosen cores is boosted progressively in steps of one *level* during the subsequent epochs, provided that there is thermal headroom. Boosting stops upon reaching the maximum possible frequency ($F_{max}$) through successive steps of boosting up to $B$ levels. When more than one application makes a request for boosting, the application running on cores with relatively lower temperatures is chosen.

**Algorithm 3.** Boosting

**Inputs:** $Temp[]$ : Per-core temperature vector, $APPS$, $APPS_{boost}$ : Applications - currently running on the chip and requested for boosting
**Global Variables:** $T_{max}$ Maximum temperature on the chip, $F$ : Frequency, $c_i$: Core $i$
**Global Constants:** $T_{crit}$: Critical temperature, $E_{boost}$: Boosting epoch, $level$: Step size of boosting frequency, $F_{max}$, $F_{min}$: Maximum and minimum operating frequencies

**Body:**
1: **for** every $E_{boost}$ **do**
2: $Temp[] \leftarrow$ updateTemperature();
3: $T_{max} \leftarrow \max(Temp[])$;
4: $\triangle T \leftarrow T_{crit}$ - $T_{max}$ ;
5: **if** $\triangle T > 0$ **then**
6: **for** $APPS_{boost}$ **do**
$targetApp \leftarrow$ minTemp($APPS_{boost}$);
7: **if** $targetApp \rightarrow$ F $\neq F_{max}$ **then**
8: **for** $c_i \in targetApp$ **do**
9: Boost($c_i, level$) ;
10: **else**
11: **for** $APPS$ **do**
$targetApp \leftarrow$ maxTemp($APPS_{boost}$);
12: **if** $targetApp \rightarrow$ F $\neq F_{min}$ **then**
13: **for** $c_i \in targetApp$ **do**
14: Down($c_i, level$) ;

$\triangle T < 0$ With no thermal headroom ($T_{crit} < T_{max}$), the controller lowers the frequency of active cores and any applications' performance requests cannot be handled. The controller chooses application running on cores with maximum temperature to downscale the frequency by one *level*. Lowering the frequency continues progressively in steps of one *level* over the subsequent epochs, until the temperature violation is resolved. Similar to the boosting, frequency of each core among the set of cores running the chosen application is scaled down (line 14, Algorithm 3). Once there is enough thermal headroom, the boosting process resumes normally as per application requirements.

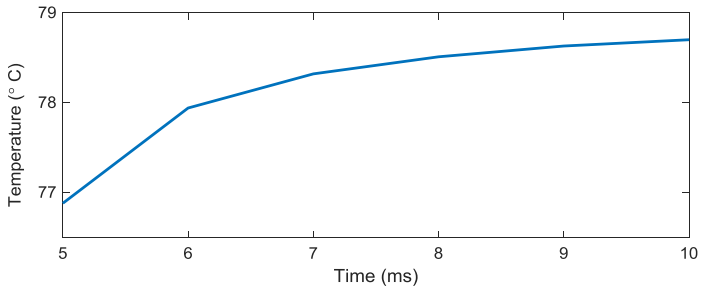



Fig. 9. Rate of temperature accumulation.

The critical temperature, epoch of boosting ($E_{boost}$), number of boosting frequency levels ($B$) and the size of each level ($level$) are parameterizable. Increase in frequency with boosting will increase the temperature subsequently over time. Fig. 9 shows the rate at which temperature accumulates upon increasing the frequency from 4 GHz to 4.2 GHz. We observe that temperature accumulation is steeper in the initial phases and eventually saturates over a period of 5 ms. For fine-grain control over thermal headroom, we set our boosting epoch length to half of the time period over which temperature would saturate with a 200 MHz rise in frequency. To ensure thermal safety, we prioritize applications running on cooler cores and conservatively boost only one application by 200 MHz within one epoch. The number of epochs over which temperature does not reach $T_{crit}$ with boosting represents period of boosting. Intuitively, a dense mapping with hot neighboring cores would approach the $T_{crit}$ faster, whereas a sparse mapping would offer longer boosting periods.

## 4 Evaluation

The system model and specifications are summarized in Table 1 and are elaborated as follows. *Application Model:* We evaluated the proposed mapping and boosting approach over applications that are modeled as task graphs, generated using TGG [20]. Each application is a directed graph of tasks, where each task holds information on computation and communication with other tasks, and priority factor to indicate need for performance surges. Inter-task communication is considered while mapping a specific task, by sorting all the other tasks as per their communication volume with it. This avoids mapping tasks with higher communication volume on farther cores with sparse mappings.

*System Model*. We simulated application traffic patterns using our in-house cycle-accurate many-core platform implemented in SystemC. Specifications of Niagara-2 like in-order cores obtained from McPAT [21] are used as the baseline for processing elements. The communication network

TABLE 1
Simulation Details

| Parameter | Specification |
|---|---|
| Application Model | Synthetic task graph, 4-18 tasks [20] |
| Core Model | Niagara-2 like, in-order |
| Power Model | McPAT [21] and Lumos [7], 16nm |
| Thermal Model | HotSpot 6.0 [10] |
| Network Model | Noxim NoC, mesh topology, X-Y routing [22] |

infrastructure between processing elements is provided by a pruned version of Noxim [22] that uses mesh topology and XY routing. Technology node scaling parameters are extracted from Lumos [7], an open source framework that quantifies power-performance characteristics of many-core systems with transistor scaling. We used TSP library [5] to calculate the Thermal Safe Power. In our framework (overview as in Fig. 5), a random sequence of applications enter the system and are buffered into a FIFO. A Central Manager ($CM$) assigned to node $n_{(0,0)}$ of many-core network implements the required system resource control and allocation operations including run-time mapping, power budget calculation, power controller and boost controller functions. Each of the control routines are implemented as software modules, requiring no additional specialized hardware. Our framework is designed to run as a user-space process which can communicate with the operating system to enforce resource management decisions. Applications are serviced in a first-come-first-serve policy, subject to availability of power budget. A suitable first node for mapping the incoming application is chosen by the ($CM$), followed by mapping all the remaining tasks of the application. Assuming a base frequency of 4 GHz in our evaluation, we set the step size of each boosting frequency level to 5 percent rise on the base frequency, resulting in a step size of 200 MHz. Fig. 9 shows the rate at which temperature accumulates upon boosting from 4GHz to 4.2GHz. Steep rise followed by a saturation in temperature is noticeable over a time period of 5 ms. We sampled this time taken, to avoid sharp rise in temperature before making a consecutive boosting decision. It should be noted that the rate of temperature accumulation varies with initial conditions viz., temperature of chosen cores at the instance of boosting, simultaneously active cores neighboring the boosted cores and change in level of frequency. In our design, we allow the boosting epoch and boosting frequency level as parametric values for adaptability of the policy. For evaluation purposes, we chose an epoch $E_{boost}$ of length 2.5 ms, with 4 levels of boosting such that each level corresponds to 200MHz, assuming the critical temperature $T_{crit}$ as 80 °C. A dynamic setting of boosting epoch and boosting level would be possible to determine optimal boosting period. However, this requires fine-grained monitoring and control of temperature, and exhaustive search over concurrent applications, making it NP-hard problem. Within the scope of our work, we considered a static setting which is a light weight working solution.

*Evaluation Metrics.* We evaluate our dark silicon patterning approach *PAT* that prefers sparsity against *CoNA*, that prefers contiguity [23]. Since we relaxed contiguity among individual applications, we chose to compare PAT against CoNA to quantify the effect of our patterning approach. As a baseline, we consider the system architecture presented in Section 3 which uses the power controller presented in Section 3.2, [19] for power actuation using traditional DVFS and power gating knob. For comparison of dense and sparse mappings, we use CoNA and PAT mapping policies respectively in the run-time mapping unit (RMU) module. We also compare the same mapping strategies from a boosting perspective, simulating them with the boost controller enabled (from here on referred as *PAT++* and *CoNA++*). In this case, we augmented the system baseline comprising of power controller using traditional DVFS and power gating with the adBoost controller as the policy for boosting module. We simulate the system over a period in which 100 applications enter and leave the system. The entry sequence of applications is kept the same for different mapping approaches for a fair comparison. We evaluate both the mapping strategies with and without boosting, over different sizes of the network (64-core and 144-core) and different number of simultaneously active cores. We use a combination of higher and lower number of applications arriving for execution, resulting in variable number concurrent applications and thus active cores and dark cores. This leads to two scenarios of increasing dark silicon, precisely emulating 25 and 50 percent dark cores, for evaluation. Throughout the evaluation, we refer to these combinations of sizes and active cores - (a) 64 cores, 25 percent inactive, (b) 144-cores, 25 percent inactive, (c) 64-cores, 50 percent inactive, (d) 144-cores, 50 percent inactive. Given the fixed number and sequence of applications, we compare power budgets, average waiting time (AWT), thermal profiles and throughput offered by CoNA and PAT. With the boosting controller enabled, we also compare average period of boosting for CoNA++ and PAT++. In every case, we simulated the platform at 16 nm technology and a critical temperature of 80 °C.

### 4.1 Power Budgets and Performance

Fig. 10 shows the power budgets offered by CoNA and PAT for 64-core and 144-core systems over different number of active cores. The power budgets are calculated using the TSP library [5] upon entry of a new application. Figs. 10a and 10b have 25 percent inactive cores while Figs. 10c and 10d have 50 percent inactive cores during execution. Power budgets offered by PAT are higher than that of CoNA due to the sparsity of patterned mapping. At lower network sizes and with higher number of active cores, power budget gains from PAT are relatively lower due to limited number of dark cores that can be patterned across higher number of active cores. The amount of power budget gain with PAT is higher with lower number of active cores i.e., with higher percentage of dark silicon. For instance, gains for both 64-cores and 144-core systems with 50 percent inactive cores is significant when compared to that of 64-cores and 144-cores systems with 25 percent inactive cores. This indicates that as the amount of dark silicon increases with technology node scaling, PAT provides higher power budgets. The surplus gain obtained at every instance of an application's entry in turn would be utilized to accommodate incoming applications if possible, which reflects in higher performance and better power budget utilization. Power budget gains continue to increase with increase in the number of active cores. The relative power budget gain among different mappings however depends on number of dark cores among total cores that can be patterned for power budget gains.

Applications that entered the system when there is not enough power budget are forced to wait in the queue until sufficient power budget becomes available. We refer to this time elapsed between application entering the system and starting its execution as the average waiting time (AWT). With CoNA offering lower power budgets compared to PAT, the system often does not have enough budget to service incoming applications, increasing the AWT. PAT offers higher power budgets which can be utilized to accommodate

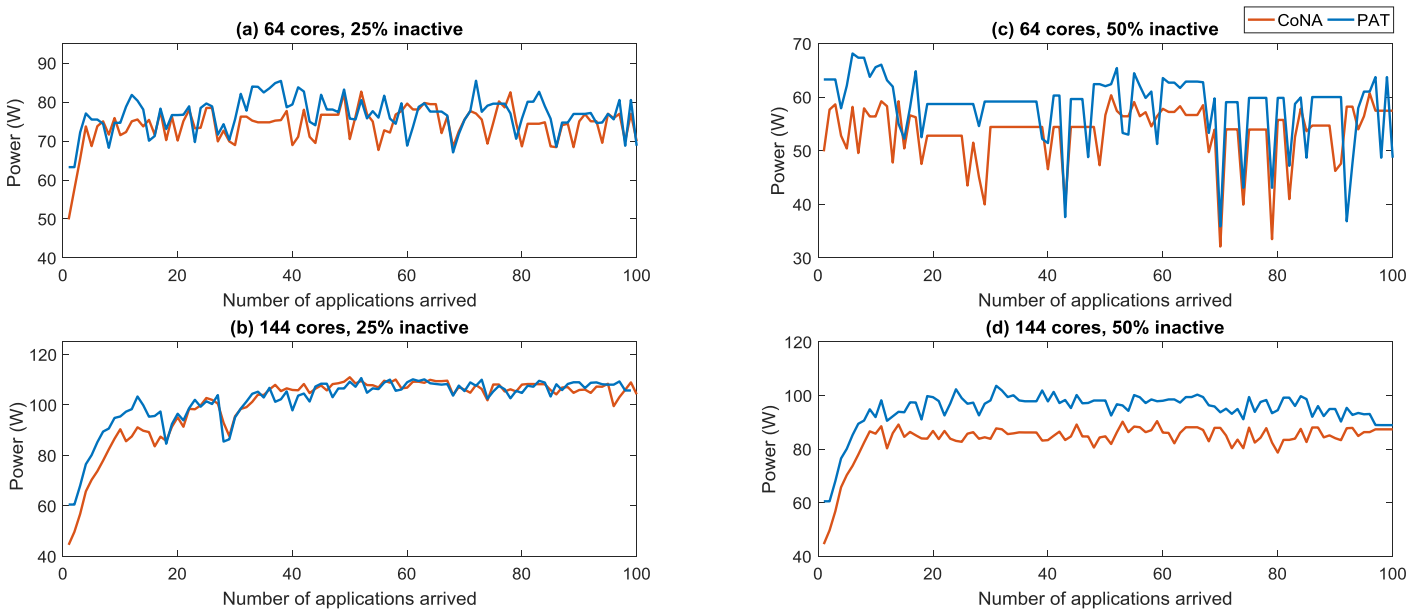



Fig. 10. Power budgets of CoNA and PAT with varying number of active cores. (a) 64-cores, with 25 percent inactive. (b) 144-cores, with 25 percent inactive. (c) 64-cores with 50 percent inactive. (d) 144-cores with 50 percent inactive. Power budget is calculated using TSP calculator upon arrival/exit of an application, over 100 applications that enter and leave the system.

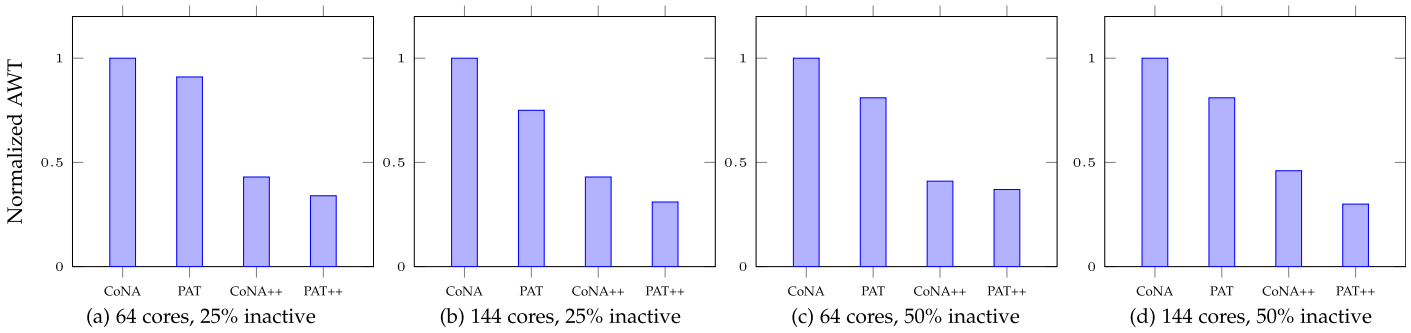



Fig. 11. Average waiting times (AWT) of CoNA and PAT with varying number of active cores. (a) 64-cores with 25 percent inactive. (b) 144-cores with 25 percent inactive. (c) 64-cores with 50 percent inactive. (d) 144-cores with 50 percent inactive. AWT is calculated as the average time spent by each application in the queue before it starts execution, over 100 applications that enter and leave the system.

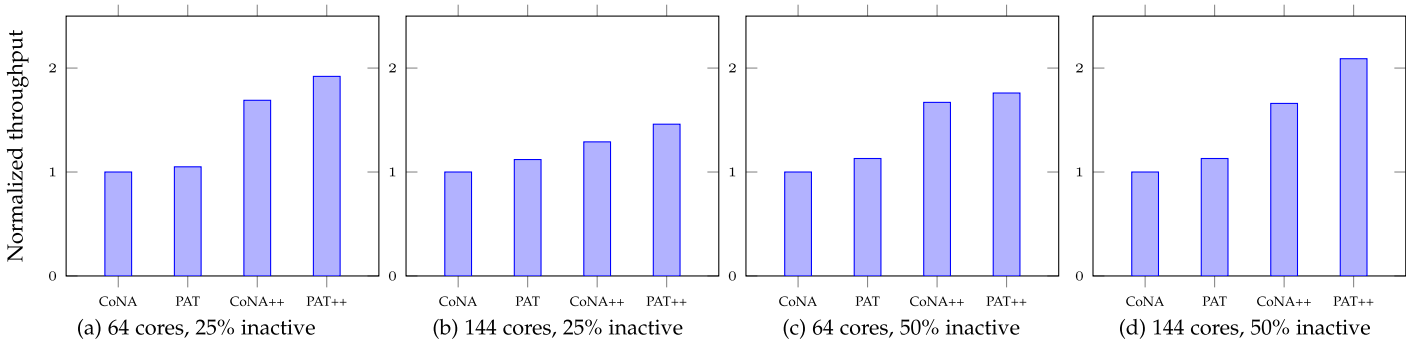



Fig. 12. Throughput gain of PAT over CoNA with varying number of active cores. Normalized throughput is calculated as time taken for 100 applications to enter and leave the system. (a) 64-core system with 25 percent inactive cores. (b) 144-core system with 25 percent inactive cores. (c) 64-core system with 50 percent inactive cores. (d) 144-core system with 50 percent inactive cores.

incoming applications more frequently. Thus, PAT minimizes the time spent by the applications waiting in the queue, which eventually reflects in higher performance. Fig. 11 shows the normalized AWT of CoNA, PAT, CoNA++ and PAT++. The gain in AWT for PAT over CoNA is evident across different sizes of network and active cores due to the higher power budget. CoNA++ has a better AWT than PAT, with high performance from frequency boosting which results in accommodating incoming applications. PAT++ has better AWT compared to all the other strategies with both higher power budget gain and longer periods of boosting due to patterned mapping. Lower AWT implies applications' request for execution being serviced quicker, reflecting a lower overall execution time. Fig. 12 shows normalized throughput gains for CoNA and PAT - with and without boosting. Throughput is calculated as the time taken for 100 applications to enter and leave the system. PAT and PAT++ have higher throughput compared to CoNA and CoNA++, as expected due to the

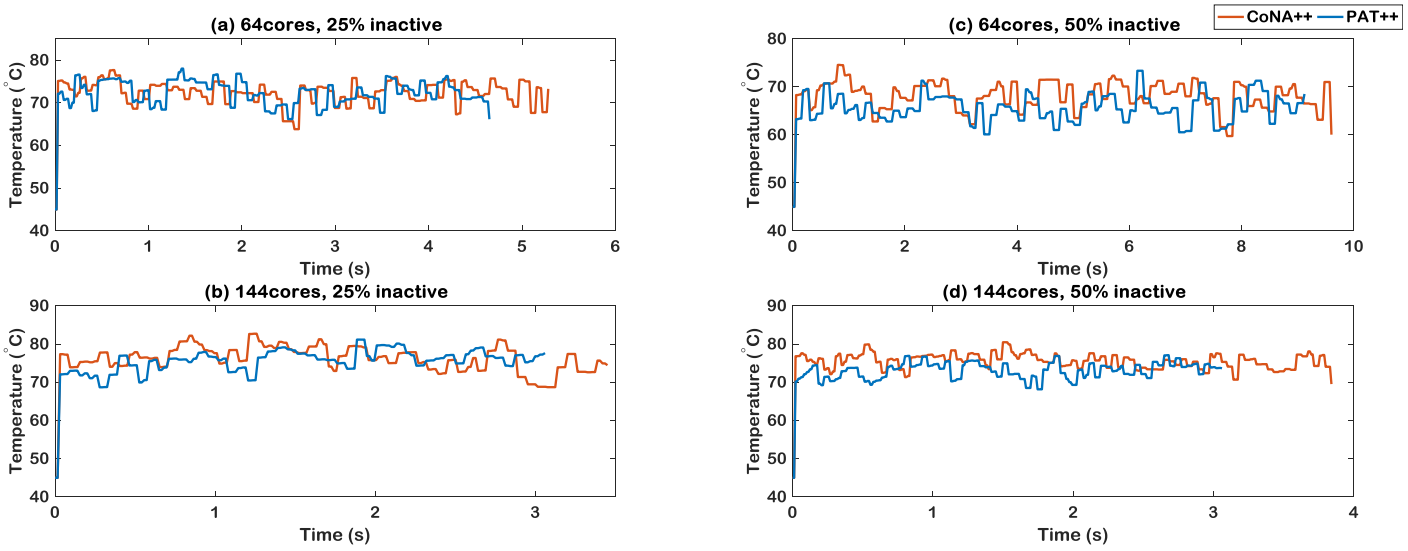



Fig. 13. Temperature profiles of CoNA++ and PAT++ for simulation time of 100 applications entering and leaving the system. (a) 64-cores with 25 percent inactive. (b) 144-cores with 25 percent inactive. (c) 64-cores with 50 percent inactive. (d) 144-cores with 50 percent inactive.

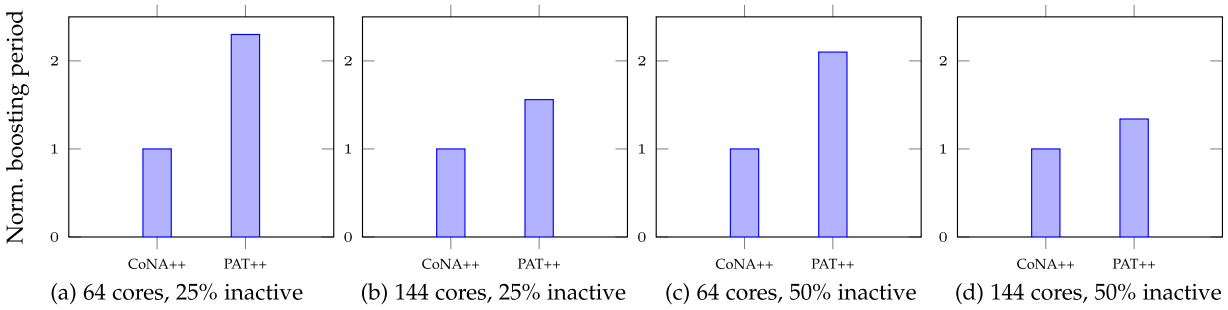



Fig. 14. Boosting period gains for PAT++ over CoNA++. Boosting period is calculated as time the simulation ran in boosting mode per overall execution time for 100 applications to enter and leave the system. (a) 64-cores with 25 percent inactive. (b) 144-cores with 25 percent inactive. (c) 64-cores with 50 percent inactive. (d) 144-cores with 50 percent inactive.

surplus power budgets. Similar to the power budget gains, throughput gains also are significant with increase in dark silicon. Throughput gain of PAT over CoNA can be attributed to two factors - i) higher power budget offered for a given application, which improves per-application performance and ii) higher power budget allocation for the chip over several concurrent applications, which improves per-chip throughput. The surplus power budget gained with entry of a sequence of applications is accumulated and utilized whenever possible to accommodate new applications waiting in the queue. Within the same amount of time, PAT would be able to execute more applications than CoNA and/or execute the same number of applications as CoNA within a lesser time. It should be noted that for same sequence of incoming applications, AWT decreases with increase in number of active cores, as there are higher amount of resources available. However, increase in number of active cores can be a result of either larger mesh sizes or relatively fewer dark cores. The gains for PAT over CoNA and PAT++ over CoNA++ also follows the trend of AWT in respective cases, reflecting the effect of AWT on throughput. In addition to these factors, throughput gain with PAT++ over CoNA++ can be attributed to more frequent thermal violations of CoNA++ with boosting. Every time the temperature approaches 80°C, frequency of active cores has to be lowered for thermal safety, which in turn reduces some performance. With PAT++, thermal constraints are honored, retaining a higher period of boosting and performance.

### 4.2 Thermal Profile and Boosting

Fig. 13 shows thermal profiles of CoNA++ and PAT++ for different mesh sizes and active cores, showing the maximum per-core temperature observed on the chip. PAT++ is relatively cooler when compared to CoNA++ due to the sparsity of active cores. CoNA++ on the other hand is more susceptible to accumulate temperature and form hot-spots with tightly packed active cores. Since the surplus power budget from PAT++ is utilized either to map new incoming applications or to boost the frequency (or both), the difference between temperatures is relatively small. Operating the chip exclusively at lower temperatures with the same power budget instead of utilizing the surplus budget to increase performance could a designer's choice.

Fig. 14 shows periods of boosting for CoNA++ and PAT++ over different mesh sizes and active cores. Boosting period is calculated as the accumulated sum of the number of boosting epochs ($E_{boost}$) elapsed during the overall simulation time. PAT++ has a relatively longer period of boosting compared to CoNA++ in all the cases. PAT++ follows sparse mapping of an application and also disperses concurrent applications which result in lower temperatures. Boosting active cores that are already operating lower temperatures allows the increase in frequency to be sustained for longer intervals, before the temperature eventually might approach critical temperature. Both higher power budgets and lower operating temperatures from dark silicon patterning favor PAT++ in sustaining longer boosting

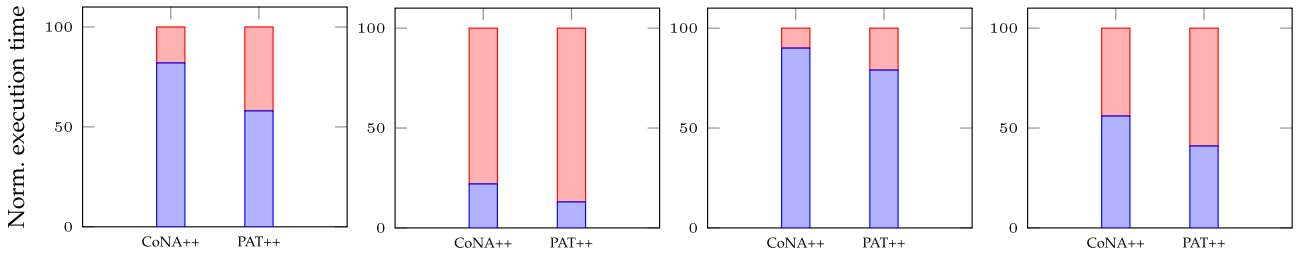



Fig. 15. Boosting period for CoNA++ and PAT++. Normalized execution time is split into normal execution (in blue) and boosted execution (in red) for (a) 64-cores with 25 percent inactive. (b) 144-cores with 25 percent inactive. (c) 64-cores with 50 percent inactive. (d) 144-cores with 50 percent inactive.

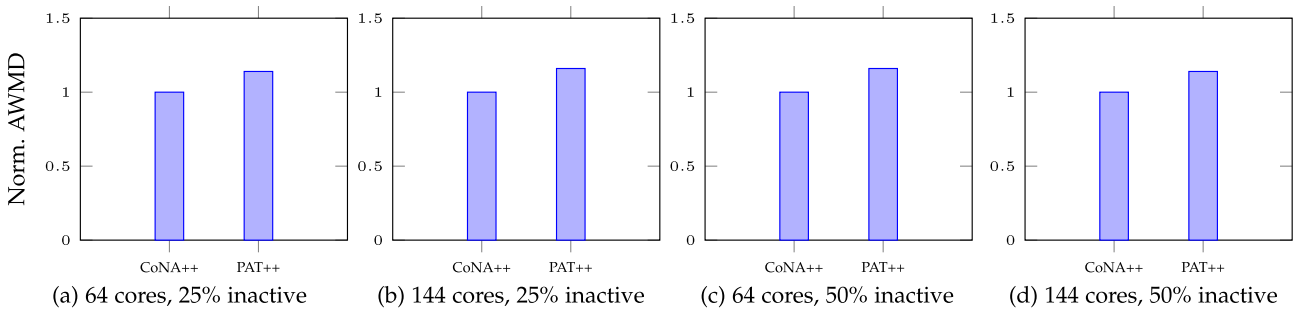



Fig. 16. Average weighted manhattan distance (AWMD) of PAT over CoNA. (a) 64-core system with 25 percent inactive cores. (b) 144-core system with 25 percent inactive cores. (c) 64-core system with 50 percent inactive cores. (d) 144-core system with 50 percent inactive cores.

periods. Additional performance extracted from the longer boosting periods reflect in throughput gains from PAT++ over CoNA++. Boosting in case of CoNA++ results in a lower gain, given the contiguous nature of mapping that accumulates temperature, resulting in active cores that are relatively hot. Boosting the cores which are already at higher temperatures would further contribute to approach the critical temperature faster. As on-chip temperatures approach critical temperature, frequency of boosted cores has to be scaled down towards normal mode of execution, limiting the effective boosting period and thus performance gains. Another consideration to be made for power budget utilization is the benefits obtained from boosting. Fig. 15 shows the normalized period of boosting per overall execution time. It is to be noted that higher period of boosting might not necessarily translate into equivalent higher performance. For instance, the period of boosting is higher in case of (b) 144-cores, 25 percent inactive - with higher number of active cores, and lower in case of (c) 64-cores, 50 percent inactive - with lower number of active cores. With higher number of active cores, the incidence of thermal violation is more frequent. The controller thus rolls between normal and boost modes of execution. There are several phases of - boosting followed by frequency downscaling, increasing the period of boosting. In contrast, with fewer active cores, the temperature barely approaches critical temperature. This gives an uninhibited boosting scenario, such that maximum operating frequency is reached sooner. Despite not having a longer boosting period, gain in performance is achieved through effective utilization of boosting.

### 4.3 Overhead with Sparse Mapping and Controllers

The total number of hops traversed by all packets sent and received by tasks of an application, referred as Weighted Manhattan Distance (WMD) represents the penalty levied by inter-core communication. Although sparse mapping provides high power budget and low operating temperatures, dispersed tasks of an application increases the WMD, compared to contiguous mappings such as CoNA. Fig. 16 shows the average weighted Manhattan distance (AWMD) of PAT over CoNA for different mesh sizes and active cores. The normalized AWMD is higher for PAT when compared to CoNA, indicating a higher latency. Despite the latency, PAT still offers better performance over CoNA, with higher power budgets that compensate for the additional packet latency. Our timing model includes additional packet latency induced with sparse mappings for fair comparison. With the same sequence of workloads, patterned mapping configurations per each application remain similar in each of the cases in Figs. 16a, 16b, 16c, and 16d, although the location of first node might vary subject to concurrently running applications. This results in a similar amount of additional hops traversed and thus similar AWMD values with sparse mappings. The additional NoC energy overhead incurred in inter-task communication due to additional hops with sparse mappings is 14, 16, 14 and 16 percent respectively, which is also proportional to the performance overhead (AWMD). The proactive first node finding uses a look up table to find appropriate first node for any possible sized incoming application by continuously keeping track of density factors of each node. Updating the density factors is of $O(n \times M \times \sqrt{M})$ complexity, where $n$ is the number of tasks of an application (or application size) and $M$ is the mesh size. The patterning algorithm has complexity of $O(n^2)$, where $n$ is the size of the application. The complexity of power controller and boost controllers are $O(M)$ and $O(n)$ respectively, where $M$ is the mesh size and $n$ is number of tasks within an application. The run-time overhead induced with the controller includes latency elapsed in transmitting the frequency settings from the central manager to the

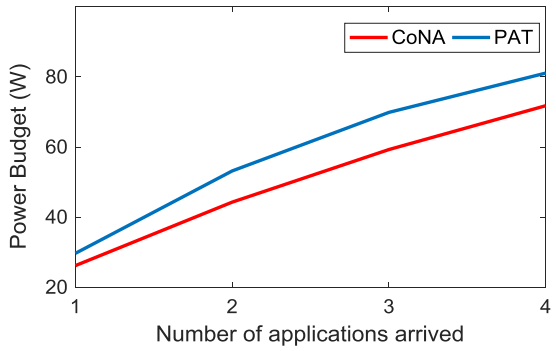



Fig. 17. Power budgets with CoNA and PAT mappings. Applications concurrently run: streamcluster, ferret, bodytrack and blackscholes

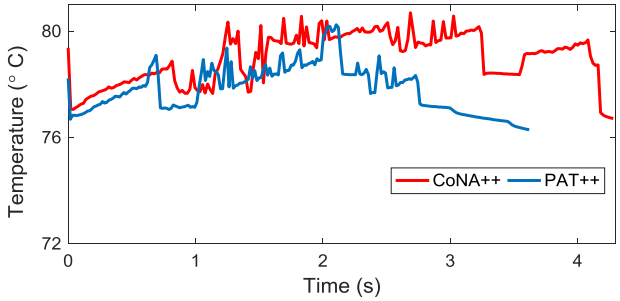



Fig. 18. Thermal profiles with CoNA++ and PAT++. Applications concurrently run: streamcluster, ferret, bodytrack and blackscholes.

respective cores. This depends on the Manhattan distance between the central manager and the active core, such that the worst case latency is equal to $\sqrt{2} \times M \times hopLatency$, where $M$ is the size of the network and $hopLatency$ is the per-hop latency of a packet. The overhead of finding first node, mapping, power management and boosting decisions and enforcement of voltage and frequency settings are included in the execution timing model.

### 4.4 Gains with other Workloads

We also evaluate our proposed approach on real workloads, apart from the synthetic task graphs. We chose 4 applications from PARSEC benchmark suite viz., `blackscholes`, `bodytrack`, `ferret` and `streamcluster` and native sim-small input data. We extracted execution traces of the applications using Intel Pin tool and power traces using McPAT. Technology node specifications are obtained from Lumos [24], validated against Niagara-2 in-order cores at 16 nm technology node. The applications are concurrently run on a 64-core system over the same in-house platform platform used for evaluating the synthetic workloads, described earlier. Power budgets achieved at run-time with entry of each new application using CoNA and PAT are shown in Fig. 17. With sparse mapping providing necessary cooling effect among active cores, PAT offers upto 17 percent more power budget, compared to CoNA. Thermal profiles of the same set of applications using the adBoost controller over CoNA++ and PAT++ are shown in Fig. 18. It can be observed that PAT++ maintains a lower temperature profile with patterned mapping, which subsequently provides more thermal headroom for PAT++ compared to CoNA++, which has a limited scope for boosting. The subsequent performance gains with PAT++ over CoNA++ along with PAT over CoNA are shown in Fig. 19. The

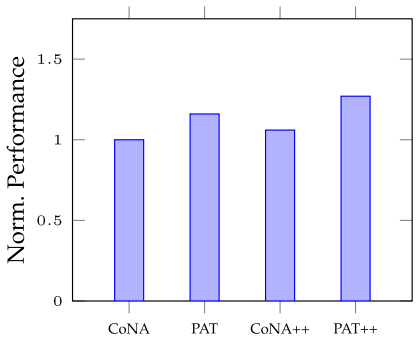



Fig. 19. Normalized performance for different mappings. CoNA and PAT represent performance using power controller alone, CoNA++ and PAT++ represent performance also using boost controller. Applications concurrently run: streamcluster, ferret, bodytrack and blackscholes.

additional power budget and relatively lower temperatures offered by sparse mapping allows PAT++ to be boosted for upto 18 percent longer compared against CoNA++.

## 5 Related Work

*Application Mapping.* State-of-the-art dynamic application mapping strategies have focused on performance aspects through minimizing congestion and spatial alignment [6], [25], [26], without considering power budgets. The emphasis of mapping techniques is constrained to inter-task communication and contiguous mapping with performance objectives alone [27], [28], [29]. These algorithms do not consider maximizing performance within given power budgets. The limitations on future many-core systems with dark silicon challenge requires re-structuring of mapping objectives towards improving power budgets by considering dark silicon, and thus performance. Contiguous mapping avoiding dispersion and fragmentation for better performance was proposed in [23]. However, power budget limitation with dark silicon changes the implications of dispersion and fragmentation on system performance. A non-contiguous mapping through geometrical partitioning of the network is presented in [30], minimizing performance penalties by mapping communicating tasks on nearby cores and the rest in proximity. This establishes that non-contiguous mappings not necessarily affects system performance, which can be exploited to attack dark silicon. Patterning inevitable dark cores around active neighbors for improving power budget utilization was presented in our previous work [8]. In the similar vein, maximizing performance through adaptive mapping within a given power budget, by exploiting process variations, reliability and thermal considerations were presented in [15], [31], [32], [33], [34], [35]. This emphasizes the significance of application mapping on maximizing performance within fixed power budgets.

*Dark Silicon and Power Budgeting.* Dark silicon phenomenon and its implications on performance are well established in [3], [36]. Precisely, power densities of computer systems are rising with technology scaling, which leads to thermal violations. Common practise of ensuring thermal safety is to cap the power consumption at a fixed upper bound - thermal design power (TDP) [4]. A slightly flexible option for power budgeting were AMD and Intel Corporations' CPUs with configurable TDP options, however without any fine grained control [37], [38]. Selectively

mapping applications to restrict power consumption to a fixed limit was proposed in [13], using a feedback controller. There are other reactive strategies that trigger dynamic power and thermal management techniques to honor TDP [12], [15], [19], [39], [40], [41], [42]. Pagani et al. have proposed more realistic and dynamic estimate of power budget by expressing it as a function of simultaneously active cores i.e., mapping configuration [5]. They provide a light weight C library to estimate thermal safe power (TSP) for a given mapping configuration along with the possible worst case TSP for a given number of active cores. The effect of application mapping on power budget, by aligning dark cores among active cores have been presented in [43]. It shows a two fold gain from patterning - better power budget utilization and lower operating temperatures. [44] quantified the advantages dark silicon patterning which results in higher power budgets and lower temperature profiles of the chip. However, principles for dark silicon patterning based mapping, utilizing surplus power budget from patterning, allocation of the same for boosting performance have remained unexplored. To summarize, most of the strategies mentioned above were reactive to power/thermal violations but do not directly address the performance penalties or maximizing performance within given power budgets.

*Boosting.* The initial idea of throttling an application's frequency to maximize performance per watt was proposed by Murali et al. [45]. The goal is to choose appropriate pareto-optimal space among instructions per second and power per instructions, depending on amount of parallelism in an application. This feature was later implemented in Intel's architecture as the TurboBoost feature which increases the frequency of active cores beyond the base frequency in short bursts of time for performance gains [46]. Raghavan et al. have proposed the concept of computational sprinting - exceeding the power limitations of a chip over short bursts of time by activating cores which are otherwise dark [47]. They focus on accelerating particularly the parallel sections in applications which can have performance gains with sprinting. Further, they also use frequency sprinting on top of parallel sprinting, relying on heat dissipation during a cool down period that follows the boosting period. Selectively boosting an application while scaling down the frequency of other concurrent applications is proposed in [16]. Most of these works emphasize on boosting the frequency, while having enough thermal headroom and power budget remain the key requirements to facilitate boosting. In our work, we first address the issue of creating surplus power budget through dark silicon patterning, followed by efficient allocation of the available thermal headroom to then boost the frequency of active cores.

## 6 Conclusion

We have presented *adBoost* controller for thermal aware performance boosting in many-core systems. *adBoost* benefits from dark silicon patterning (PAT), a mapping strategy to evenly distribute temperature across the chip in order to improve utilizable power budget. PAT maps tasks and applications in a sparse manner by patterning inevitable cooler dark cores around hotter active cores, allowing them to utilize power efficiently before reaching critical temperature. Our power controller efficiently allocates the surplus power budget from patterning to map new applications or throttle current applications while *adBoost* controller scales the frequency upon performance requirements. Lower on-chip temperatures with PAT provides sustainable periods of boosting, improving both per-application latency and per-chip throughput. In comparison with the state-of-the-art contiguous mapping strategies, the combination of *adBoost* and PAT offers higher power budget, lower temperatures and sustain longer periods of boosting. These put together reflect in better throughput and efficient utilization of power budget. Our strategy can also be extended for heterogeneous architectures, provided the prior knowledge on performance-per-watt of every application/task. This allows the choice of more suitable set of heterogeneous cores at the mapping phase, which reflects on lower on-chip temperatures using power efficient cores. This can be leveraged to boost the frequency of low-power cores for improved performance and/or use the thermal headroom created with the low-power cores to boost the performance of other power hungry cores.

## Acknowledgments

We acknowledge financial support from the Marie Curie Actions of the European Union's H2020 Programme.

## References

[1] A. Kanduri, A. M. Rahmani, P. Liljeberg, A. Hemani, A. Jantsch, and H. Tenhunen, "A perspective on dark silicon," in *The Dark Side of Silicon*. Berlin, Germany: Springer, 2017, pp. 3–20.
[2] A. Rahmani, P. Liljeberg, A. Hemani, A. Jantsch, and H. Tenhunen, *The Dark Side of Silicon*. Berlin, Germany: Springer, 2016.
[3] H. Esmaeilzadeh, E. Blem, R. S. Amant, K. Sankaralingam, and D. Burger, "Dark silicon and the end of multicore scaling," in *Proc. Annu. Int. Symp. Comput. Archit.*, 2011, pp. 365–376.
[4] "Intel Xeon Processor - Measuring Processor Power, revision 1.1," in *White paper, Intel Corporation*, April, 2011.
[5] S. Pagani, et al., "TSP: Thermal safe power: Efficient power budgeting for many-core systems in dark silicon," in *Proc. Int. Conf. Hardware/Softw. Codesign Syst. Synthesis*, 2014, Art. no. 10.
[6] M. Fattah, M. Daneshtalab, P. Liljeberg, and J. Plosila, "Smart hill climbing for agile dynamic mapping in many-core systems," in *Proc. ACM/EDAC/IEEE Des. Autom. Conf.*, 2013, pp. 1–6.
[7] L. Wang and K. Skadron, "Dark versus Dim Silicon and Near-Threshold computing extended results," in Univ. Virginia, Charlottesville, VA, Tech. Rep. TR-2013-01, 2012.
[8] A. Kanduri, M.-H. Haghbayan, A.-M. Rahmani, P. Liljeberg, A. Jantsch, and H. Tenhunen, "Dark silicon aware runtime mapping for many-core systems: A patterning approach," in *Proc. IEEE Int. Conf. Comput. Des.*, 2015, pp. 573–580.
[9] W. Kim, M. S. Gupta, G.-Y. Wei, and D. Brooks, "System level analysis of fast, per-core DVFS using on-chip switching regulators," in *Proc. Int. Symp. High Perform. Comput. Archit.*, 2008, pp. 123–134.
[10] W. Huang, S. Ghosh, S. Velusamy, K. Sankaranarayanan, K. Skadron, and M. R. Stan, "Hotspot: A compact thermal modeling methodology for early-stage vlsi design," *IEEE Tran. VLSI Syst.*, vol. 14, no. 5, pp. 501–513, May 2006.
[11] S. Pagani, H. Khdr, J.-J. Chen, M. Shafique, M. Li, and J. Henkel, "Thermal safe power (TSP): Efficient power budgeting for heterogeneous manycore systems in dark silicon," *IEEE Trans. Comput.*, vol. 66, no. 1, pp. 147–162, Jan. 2017.
[12] A. M. Rahmani, M.-H. Haghbayan, A. Miele, P. Liljeberg, A. Jantsch, and H. Tenhunen, "Reliability-aware runtime power management for many-core systems in the dark silicon era," *IEEE Tran. VLSI Syst.*, vol. 25, no. 2, pp. 427–440, Feb. 2017.
[13] M.-H. Haghbayan, et al., "Dark silicon aware power management for manycore systems under dynamic workloads," in *Proc. IEEE 32nd Int. Conf. Comput. Des.*, 2014, pp. 509–512.

[14] J. Zhan, et al., "Designing energy-efficient NoC for real-time embedded systems through slack optimization," in *Proc. ACM/EDAC/IEEE Des. Autom. Conf.*, 2013, Art. no. 37.
[15] J. Zhan, Y. Xie, and G. Sun, "NoC-sprinting: Interconnect for fine-grained sprinting in the dark silicon era," in *Proc. ACM/EDAC/IEEE Des. Autom. Conf.*, 2014, pp. 1–6.
[16] S. Pagani, M. Shafique, H. Khdr, J.-J. Chen, and J. Henkel, "seBoost: Selective boosting for heterogeneous manycores," in *Proc. Int. Conf. Hardware/Softw. Codes. Syst. Synthesis (CODES +ISSS)*, 2015, pp. 104–113.
[17] M.-H. Haghbayan, et al., "Mappro: Proactive runtime mapping for dynamic workloads by quantifying ripple effect of applications on networks-on-chip," in *Proc. IEEE/ACM Int. Symp. Netw.-on-Chip*, 2015, pp. 1–8.
[18] H. E. White, *Modern College Physics*. New York, NY, USA: Van Nostrand, 1948.
[19] A.-M. Rahmani, et al., "Dynamic power management for many-core platforms in the dark silicon era: A multi-objective control approach," in *Proc. IEEE/ACM Int. Symp. Low Power Electron. Des.*, 2015, pp. 219–224.
[20] TGG: Task Graph Generator, 2010. [Online]. Available: http://sourceforge.net/projects/taskgraphgen/
[21] S. Li, et al., "McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures," in *Proc. Annu. IEEE/ACM Int. Symp. Microarchitecture*, 2009, pp. 469–480.
[22] F. Fazzino, M. Palesi, and D. Patti, Noxim: Network-on-chip simulator, 2008. [Online]. Available: http://sourceforge. net/projects/noxim
[23] M. Fattah, et al., "Cona: Dynamic application mapping for congestion reduction in many-core systems," in *Proc. IEEE 30th Int. Conf. Comput. Des.*, 2012, pp. 364–370.
[24] Lumos Framework, [Online]. Available: http://liangwang.github.io/lumos/, Accessed on: May 20, 2014.
[25] C.-L. Chou and R. Marculescu, "Contention-aware application mapping for network-on-chip communication architectures," in *Proc. Int. Conf. Comput. Des.*, 2008, pp. 164–169.
[26] M. Fattah, P. Liljeberg, J. Plosila, and H. Tenhunen, "Adjustable contiguity of run-time task allocation in networked many-core systems," in *Proc. Asia South Pacific Des. Autom. Conf.*, 2014, pp. 349–354.
[27] C.-L. Chou, U. Y. Ogras, and R. Marculescu, "Energy-and performance-aware incremental mapping for networks on chip with multiple voltage levels," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 27, no. 10, pp. 1866–1879, Oct. 2008.
[28] M. A. Bender, et al., "Communication-aware processor allocation for supercomputers: Finding point sets of small average distance," *Springer Algorithmica*, vol. 50, no. 2, pp. 279–298, 2008.
[29] E. Carvalho, N. Calazans, and F. Moraes, "Heuristics for dynamic task mapping in noc-based heterogeneous MPSoCs," in *Proc. IEEE/IFIP Int. Workshop Rapid Syst. Prototyping*, 2007, pp. 34–40.
[30] M. Deveci, et al., "Exploiting geometric partitioning in task mapping for parallel computers," in *Proc. IEEE 28th Int. Parallel Distrib. Process. Symp.*, 2014, pp. 27–36.
[31] B. Raghunathan, Y. Turakhia, S. Garg, and D. Marculescu, "Cherry-picking: Exploiting process variations in dark-silicon homogeneous chip multi-processors," in *Proc. Des. Autom. Test Eur. Conf. Exhib.*, 2013, pp. 39–44.
[32] N. Kapadia and S. Pasricha, "VARSHA: Variation and reliability-aware application scheduling with adaptive parallelism in the dark-silicon era," in *Proc. Des. Autom. Test Eur. Conf. Exhibit.*, 2015, pp. 1060–1065.
[33] M.-H. Haghbayan, et al., "Performance/reliability-aware resource management for many-cores in dark silicon era," *IEEE Trans. Comput.*, vol. 66, no. 9, pp. 1599–1612, Sep. 2017.
[34] Y. Liu, Y. Ruan, Z. Lai, and W. Jing, "Energy and thermal aware mapping for mesh-based noc architectures using multi-objective ant colony algorithm," in *Proc. IEEE Int. Conf. Comput. Res. Develop.*, vol. 3, 2011, pp. 407–411.
[35] M. Bao, A. Andrei, P. Eles, and Z. Peng, "Temperature-aware task mapping for energy optimization with dynamic voltage scaling," in *Proc. IEEE Workshop Des. Diagnostics Electron. Circuits Syst.*, 2008, pp. 1–6.
[36] M. Taylor, "Is dark silicon useful?: Harnessing the four horsemen of the coming dark silicon apocalypse," in *Proc. Des. Autom. Conf.*, 2012, pp. 1131–1136.
[37] "Intel Corporation. Fourth Generation Mobile Processor Family Data Sheet," in *White paper, Intel Corporation*, July, 2014.
[38] AMD Kaveri APU A10-7800, [Online]. Available: http://www.phoronix.com/scan.php?page=articleitem=amd_a_45watt&num=1, Accessed on: Feb. 28, 2015.
[39] A. K. Coskun, T. S. Rosing, and K. C. Gross, "Temperature management in multiprocessor socs using online learning," in *Proc. ACM/IEEE Des. Autom. Conf.*, 2008, pp. 890–893.
[40] L. Thiele, L. Schor, H. Yang, and I. Bacivarov, "Thermal-aware system analysis and software synthesis for embedded multiprocessors," in *Proc. ACM/EDAC/IEEE Des. Autom. Conf.*, 2011, pp. 268–273.
[41] T. Komoda, et al., "Power capping of CPU-GPU heterogeneous systems through coordinating DVFS and task mapping," in *Proc. Int. Conf. Comput. Des.*, 2013, pp. 349–356.
[42] A. K. Mishra, et al., "A case for dynamic frequency tuning in on-chip networks," in *Proc. Annu. IEEE/ACM Int. Symp. Microarchitecture*, 2009, pp. 292–303.
[43] M. Shafique, R. Das, S. Eachempati, R, Iyer, N. Vijaykrishnan, and C. R. Das, "Dark silicon as a challenge for hardware/software co-design: Invited special session paper," in *Proc. Int. Conf. Hardware/Softw. Codesign Syst. Synthesis*, 2014, Art. no. 13.
[44] M. Shafique, S. Garg, J. Henkel, and D. Marculescu, "The EDA challenges in the dark silicon era," in *Proc. Des. Autom. Conf.*, 2014, pp. 1–6.
[45] M. Annavaram, E. Grochowski, and J. Shen, "Mitigating amdahl's law through EPI throttling," in *Proc. Int. Symp. Comput. Archit.*, 2005, pp. 298–309.
[46] E. O. Rotem, "Power-management architecture of the intel microarchitecture code-named sandy bridge," *IEEE Micro*, vol. 32, no. 2, pp. 20–27, Mar./Apr. 2012.
[47] A. Raghavan, et al., "Computational sprinting," in *Proc. Int. Symp. High-Perform. Comput. Archit.*, 2012, pp. 1–12.

**Anil Kanduri** received MSc (Tech) degree in embedded computing from the University of Turku, Finland, in 2014. Since then, he has been working towards the PhD degree in computer systems in the department of Information Technology, University of Turku. His research interests include energy efficient computer architectures, run-time management and approximate computing. He is a student member of the IEEE.

**Mohammad-Hashem Haghbayan** received the BS degree in computer engineering from Ferdowsi University of Mashhad, Iran, the MS degree in computer architecture from the University of Tehran, Iran, and the PhD degree in information and communication technology from the University of Turku, Finland. He is currently a post-doctoral researcher in the Department of Future Technologies in University of Turku, Finland. His research interests include machine-learning algorithms, high-performance energy-efficient architectures, power management techniques, and online/off-line testing. He has several years of experience working in industry as well as developing research tools. He is a student member of the IEEE.

**Amir M. Rahmani** received the master's degree from the Department of ECE, University of Tehran, Iran, in 2009, the PhD degree from the Department of IT, University of Turku, Finland, in 2012, and the MBA degree from the Turku School of Economics and European Institute of Innovation & Technology (EIT) ICT Labs, in 2014. He is currently Marie Curie Global Fellow with the University of California Irvine and TU Wien (Austria). He is also an adjunct professor (Docent) in embedded parallel and distributed computing with the University of Turku, Finland. His research interests span self-aware computing, energy-efficient many-core systems, runtime resource management, healthcare internet of things, and Fog/Edge computing. He has served on a large number of technical program committees of international conferences, such as DATE, GLSVLSI, DFT, ESTIMedia, CCNC, MobiHealth, and others, and guest editor of the special issues in journals such as the *Journal of Parallel and Distributed Computing, Future Generation Computer Systems*, the *Springer Mobile Networks and Applications*, Sensors, Supercomputing, etc. He is the author of more than 150 peer-reviewed publications. He is a senior member of the IEEE.

**Muhammad Shafique** (M'11, SM'16) received the PhD degree in computer science from the Karlsruhe Institute of Technology, in January 2011. He is a full professor in the Institute of Computer Engineering, Department of Informatics, TU Wien, Austria. He is directing the Group on Computer Architecture and Robust, Energy-Efficient Technologies (CARE-Tech). He was a senior research group leader in the Karlsruhe Institute of Technology (KIT), Germany for more than 5 years. Before, he was with Streaming Networks Pvt. Ltd. developing video coding algorithms and optimizations for VLIW-based Processors. His research interests include computer architecture, power-/energy-efficient systems, robust computing covering various dependability/reliability/resilience aspects, emerging computing trends like neuromorphic and approximate computing, neurosciences, emerging technologies and nanosystems, hardware security, adaptive and self-learning, FPGAs, MPSoCs, and embedded systems. He received the prestigious 2015 ACM/SIGDA Outstanding New Faculty Award, six gold medals in educational career, and several best paper awards and nominations at prestigious conferences like CODES+ISSS, DATE, DAC, and ICCAD, Best Master Thesis Award, and Best Lecturer Award. He has given several Invited Talks, Tutorials, and 2 Keynotes. He has also organized many special sessions at premier conferences (like DAC, ICCAD, DATE, and Codes+ISSS) and served as the guest editor of the *IEEE Design and Test Magazine (D&T)* and the *IEEE Transactions on Sustainable Computing (T-SUSC)*. He has served as the TPC co-chair of ESTIMedia and LPDC, general chair of ESTIMedia, and Track co-chair at DATE 2017-18 and FDL 2017. He has served on the program committees of several IEEE/ACM conferences like ISCA, ICCAD, DATE, CASES, FPL, and ASPDAC. He is a senior member of the IEEE and IEEE Signal Processing Society (SPS), and a member of ACM. He holds one US patent and more than 150 papers in premier journals and conferences.

**Axel Jantsch** received the dipl Ing and DR tech degrees from the Technical University of Vienna, Austria, in 1988 and 1992. He was with Siemens Austria, Vienna, as a system validation engineer from 1995 to 1997. Since 1997, he has been an associate professor in the Royal Institute of Technology (KTH), Stockholm, Sweden. Since 2000, has been a Docent, and since December 2002, a full professor of Electronic System Design in the Department of Electronic Systems. He has published more than 200 papers in international conferences and journals. He has served on a large number of technical program committees of international conferences, such as FDL, DATE, CODES ISSS, SOC, NOCS, and others, and one book in the areas of VLSI design and synthesis, system level specification, modeling and validation, HW/SW codesign and cosynthesis, reconfigurable computing, and networks on chip. He received the Alfred Schrodinger Scholarship from the Austrian Science Foundation while a guest researcher with KTH between 1993 and 1995. He has served on a large number of technical program committees of international conferences, such as FDL, DATE, CODES ISSS, SOC, NOCS, and others. He has been the TPC chair of SSDL/ FDL 2000, the TPC co-chair of CODES ISSS 2004, the general chair of CODES ISSS 2005, and the TPC co-chair of NOCS 2009. From 2002 to 2007, he was a subject area editor for the Journal of System Architecture. He is a member of the the IEEE.

**Pasi Liljeberg** received the MSc and PhD degrees in electronics and information technology from the University of Turku, Turku, Finland, in 1999 and 2005, respectively. He received adjunct professorship in embedded computing architectures in 2010. Currently he is working as a full professor with the University of Turku in the field of Embedded Systems and Internet of Things. At the moment his research interests include computer architectures, fog computing, energy efficient architectures, approximate and adaptive computing, biomedical engineering, health technology and Internet of Things. In that context he has established and leading the Internet-of-Things for Healthcare (IoT4Health) research group. Liljeberg is the author of more than 260 peer-reviewed publications. He is a member of the IEEE.

# Paper III

# Approximation Knob: Power Capping Meets Energy Efficiency

Anil Kanduri, Hashem Haghbayan, Amir Rahmani,Pasi Liljeberg, Axel Jantsch, Nikil Dutt and Hannu Tenhunen

Published in Proceedings of 35th IEEE/ACM International Conference on Computer Aided Design (ICCAD), 2016, Austin, TX, USA, Article No: 122.

# Approximation Knob: Power Capping Meets Energy Efficiency

Anil Kanduri[1], Mohammad-Hashem Haghbayan[1], Amir M. Rahmani[1],
Pasi Liljeberg[1], Axel Jantsch[2], Nikil Dutt[3], and Hannu Tenhunen[1,4]
[1]University of Turku, Finland, [2]Vienna University of Technology, Austria
[3]University of California, Irvine, USA, [4]KTH Royal Institute of Technology, Sweden
{spakan, mohhag, amirah, pakrli}@utu.fi, axel.jantsch@tuwien.ac.at, dutt@ics.uci.edu, hannu@kth.se

## ABSTRACT

Power Capping techniques are used to restrict power consumption of computer systems to a thermally safe limit. Current many-core systems employ dynamic voltage and frequency scaling (DVFS), power gating (PG) and scheduling methods as actuators for power capping. These knobs are oriented towards power actuation, while the need for performance and energy savings are increasing in the dark silicon era. To address this, we propose approximation (APPX) as another knob for close-looped power management, lending performance and energy efficiency to existing power capping techniques. We use approximation in a pro-active way for long-term performance-energy objectives, complementing the short-term reactive power objectives. We implement an approximation-enabled power management framework, APPEND, that dynamically chooses an application with appropriate level of approximation from a set of variable accuracy implementations. Subject to the system dynamics, our power manager chooses an effective combination of knobs - APPX, DVFS and PG, in a hierarchical way to ensure power capping with performance and energy gains. Our proposed approach yields 1.5× higher throughput, improved latency upto 5×, better performance per energy and dark silicon mitigation compared to state-of-the-art power management techniques over a set of applications ranging from high to no error resilience.

## Keywords

Dynamic Power Management; Power Capping; Approximate Computing

## 1. INTRODUCTION

Multi-core and many-core architectures have become conventional to meet performance requirements of emerging applications. Building denser chips leads to high power density and thermal issues, given the limited cooling solutions. The chip has to function within an upper bound on power consumption, *thermal design power* (TDP) to ensure reliable operation and lower chip temperatures. Power Capping techniques are used to restrict the power consumption below TDP, forcing a section of the chip to be powered off - the powered off area is known as Dark Silicon [5, 21]. Power Capping uses dynamic power management (DPM) techniques such as adaptive Dynamic Voltage and Frequency Scaling (DVFS) [24], Near Threshold Computing (NTC) [25], (Per-core/cluster) Power Gating (PCPG/PG) [15], scheduling techniques like controlled degree of parallelism [14], thread packing [4], task mapping [13] and task migration [17]. We refer to these different means for actuating power consumption as *knobs*. The hierarchy of different power knobs in order of their simplicity and temporal effect on performance is shown in Figure 1(a). These knobs are oriented at power actuation and are effective in power capping, making orthogonal compromises on power and performance. With the amount of simultaneously usable logic decreasing under dark silicon regime, it is necessary to extract higher performance per energy. This creates a need for energy knobs that can offer i) performance and energy gains to complement the existing power knobs, ii) long-term objectives and control.

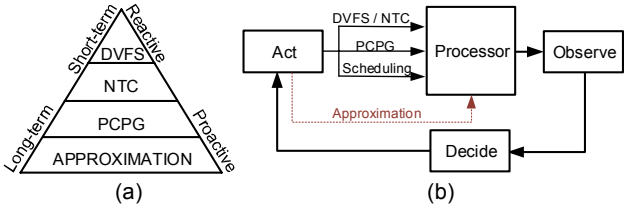



**Figure 1: Power Management Knobs**

Approximate Computing has emerged as an alternative paradigm that can offer the required performance and energy gains, by trading off some accuracy. Approximation leverages inherent error resilience of applications from several domains such as streaming, image and video processing, machine learning, big data analytics etc. Such applications can tolerate inaccurate results due to their NP-hard and iterative nature, making approximation a better choice to improve performance per energy in the dark silicon era. Recent works on approximation includes techniques such as loop perforation [22], choice of variable accuracy algorithms using logic simplification and task skipping[16, 2. 1], relaxed convergence [11] etc.. with approximation at both software and hardware levels. Most of these techniques are open-looped (or closed looped with error tolerance as a constraint), compromising accuracy. In a similar vein, power capping based on DVFS and PG knobs are closed-looped, compromising performance or energy. Putting together the existing power capping techniques with open-looped approximation, we propose a closed-loop power management framework that uses approximation, APPX, as another knob for power capping with performance and energy gains. We use loop perforation and relaxed convergence to provide multiple versions of the same application with different accuracy levels. For evaluation, we used machine learning algorithms which run iteratively on input data sets. Under loop perfo-


*ICCAD '16, November 07-10, 2016, Austin, TX, USA*

DOI: http://dx.doi.org/10.1145/2966986.2967002

ration [22], we skip some parts of the input data, reducing the number of iterations, inducing error in the eventual result. The chosen algorithms produce a good enough result with reduced computations, while further iterations try to converge towards an optimal solution. Under relaxed convergence [1]. we terminate the computation early, compromising the possibility of a more convergent solution. We present a power management framework that dynamically switches between accuracy levels of the application, choosing from the set of implementations provided. Invocation and actuation of the APPX knob determines the level of accuracy selected. We design a power controller that monitors instantaneous power consumption, workload intensity and utilization metrics of the chip and decides on appropriate actuation of DVFS, PG and APPX in a hierarchical manner. Figure 1(b) shows a top level view of the approximation knob (APPX) in conjunction with other power knobs. While DVFS and PG are used as short-term power knobs, APPX is used as a long-term energy knob, improving performance and energy efficiency, simultaneously ensuring power capping. In this paper. we propose an approximation-enabled power management approach. APPEND, for power capping and throughput enhancement. To the best of our knowledge, ours is the first approach to use approximation in the context of a closed-loop power capping for network-on-chip (NoC) based many-core systems. The contributions of this paper are as follows.

- The approximation knob (APPX) that lends performance and energy gains to power knobs by altering accuracy level of applications from a set of variable accuracy implementations
- A power management framework for power capping in many-core systems with dynamic in-flow of applications using DVFS, PG and APPX knobs
- An accuracy-aware run-time mapping technique that switches between levels of accuracy by mapping approximate tasks or replacing accurate tasks with them
- A run-time control algorithm for power, performance, throughput and energy saving decisions by actuating DVFS, PG and APPX in a hierarchical way.

## 2. BACKGROUND AND MOTIVATION

Power knobs based on voltage and frequency scaling such as DVFS and NTC are simple, yet effective for power capping due to the dependence of power on V and F. Transistor scaling results in increased leakage power, as the operational voltage approaches threshold voltage ($V_t$). limiting the scope of DVFS. A special case of voltage scaling is NTC, where supply voltage is intentionally reduced beyond threshold voltage for ultra low power operation. This compromises performance heavily, replacing dark silicon with dim silicon [25]. Power Gating (PG) reduces the static power at the expense of performance, since only fewer cores are simultaneously powered up. DVFS and PG are triggered based on a reactive strategy with short-term objective of power capping, consequently restricting performance and energy gains to a short-term. As opposed to conventional power knobs, approximation provides performance and energy gains for loss of accuracy. We demonstrate the impact of different power knobs on performance of many-core systems using sparse vector multiplication as an example. With applications entering and leaving the system dynamically, performance is determined by *service time* of an application, which is the sum of *wait time* - the time elapsed between application request and starting of the execution and *run-time* - the time consumed in executing the application on chip [14]. Dynamic workload characteristics contribute to

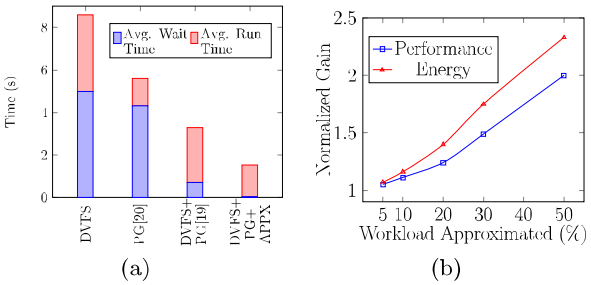



**Figure 2: Performance and Accuracy trade-offs with APPX knob. (a) Application Service Time. (b) Accuracy-Energy trade-offs**

power violations, forcing actuation of power knobs. We simulate the application for 4 different power knobs viz., DVFS, PG, DVFS+PG [20], and DVFS+PCPG+APPX, using the experimental platform, detailed in Section 5. For APPX, we used loop perforation to skip 10%-50% of input data to generate variable accuracy versions of sparse vector multiplication. The average service time for different knob combinations is shown in Figure 2(a).

In case of using DVFS and PG knobs [7], per-application run-time increases forcing incoming applications to wait longer, resulting in high service time. The combination of DVFS and PG has relatively better service time using the power management algorithm, as in [20]. With the APPX knob in combination with DVFS and PG. the service time is the lowest, indicating high performance and energy gain within the given power budget. The APPX knob loads applications with relaxed accuracy that have lower workloads and thus low run-time. Consequently, more resources are available for incoming applications, improving the wait-time and the overall service time. Despite effective power capping and possibility of increasing the number of simultaneously active cores, performance still suffers with DVFS and PG when compared to that of APPX. Hence, we propose a hierarchical management for effective combination of these knobs to complement each other. The performance gains of APPX knob come at the expense of accuracy, traded in loop perforation. Figure 2(b) shows the gain in performance and energy for amount of the workload relaxed. It is to be noted that amount of workload traded is not the same as loss in accuracy, which is subjective to input data sets.

## 3. KNOB ACTUATION SCENARIOS

In our work, we primarily monitor power consumption and workload intensity, along with utilization and network intensities, to make knob actuation decisions. We use accumulated wait-time ($AWT$) of application requests made, for monitoring the workload. For a set of applications $App_1$, $App_2$,..$App_N$ with wait-times $w_1$, $w_2$, ...$w_N$, $AWT$ is:

$$AWT = \sum_{i=1}^{N} w_i \quad (1)$$

The longer an application waits before being serviced, the higher the workload intensity. The power-objective is to restrict the power consumption to below TDP and workload-objective is to restrict the AWT to a parameterizable threshold $AWT_{th}$. We demonstrate possible scenarios that require knob(s) actuation under diverse power consumption and workload intensities. Figure 3 summarizes these scenarios, representing power consumption, workload intensity and knob actuations employed over a span of execution. Each scenario (a-f) shows power consumption with respect to TDP, applications waiting in the queue, applications that

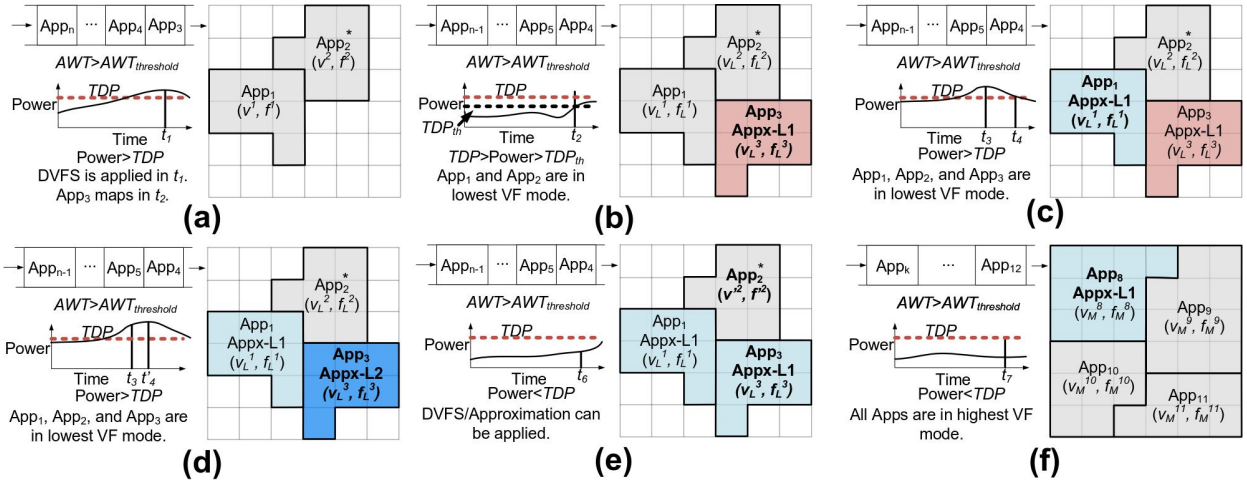



**Figure 3: Knob Actuation Scenarios**

are mapped on the chip with their respective voltage and frequency levels. Voltage and frequency levels of each mapped application (App1, App2,..Appn) are represented as $((v^1,f^1), (v^2,f^2), ...(v^n,f^n))$, with $(v_L,f_L)$ being lowest and $(v_M,f_M)$ being the maximum levels of voltage and frequency respectively. Primary criteria for knob actuation are TDP violation (i.e., power > TDP) and high request rate of incoming applications (i.e., $AWT > AWT_{th}$).

**Scenario (a):** Two applications App1 $(v^1,f^1)$ and App2 $(v^2,f^2)$ are currently on the system at their respective voltage and frequencies. At time instance $t_1$. a power violation ($power > TDP$) occurs, along with high request rate ($AWT > AWT_{th}$). The power manager employs the DVFS knob at first to stay within the power budget. The remaining un-occupied cores are power gated using the PG knob. Power-gated cores represent the amount of dark silicon accumulated on the chip. **Scenario (b):** App1 and App2 are now running at their lowest voltage and frequency levels $(v_L^1,f_L^1)$ and $(v_L^2,f_L^2)$ due to triggering of DVFS at $t_1$. At this stage. power consumption is approaching TDP, indicating a potential violation of power budget. Also, the request rate of applications is high. Anticipating the possibility of TDP violation, the power manager decides to switch the mode of incoming applications to approximate. Since App1 and App2 are already down-scaled in terms of DVFS and remaining cores are power gated, the power manager triggers APPX knob. At time instance $t_2$, a new application App3 arrives and is hence mapped onto the system in approximate mode as App3-Appx-L1 (shown in plum). As App3 is directly mapped in its approximate version, there is no overhead of mode switching. **Scenario (c):** At time instance $t_3$, power violation occurs along with a high request rate of applications. All the applications are operating at their lowest possible voltage and frequencies and rest of the cores are power gated. App3 is already being executed at Appx-L1 while App1 is in accurate mode. Since DVFS and PG are used. the power manager now switches the mode of execution of App1, whereas App2 is not approximable. App1 is switched to approximate mode, resulting in App1-Appx-L1, with some overhead (indicated in blue). These actuations may restrict the power within TDP, as shown at $t_4$. **Scenario (d):** Contradictorily to the previous scenario, the actuations may still not be able to prevent the power violation nor address the high request rate. as shown at instance $t'_4$. In this case, we further switch the mode of execution to another level of approximation. Since App1 is already executing at Appx-L1, App3-Appx-L1 is switched to App3-Appx-L2, the next level of approximation. This incurs a marginal overhead (indicated in blue) in switching between modes of execution . **Scenario (e):** With preceding knob actuations, power consumption at instance $t_6$ is well below TDP. This gives enough budget to utilize, prompting an increase in voltage and frequency levels and/or increase in accuracy of computations. We make this a user-defined quality of service (QoS) design parameter, where one can prioritize either throughput or accuracy. Accordingly, either voltage and frequency levels are up-scaled or accuracy level of the approximate application is switched one level ahead. In this scenario, we show App2 being up-scaled to $(v'^2,f'^2)$ and App-3-Appx-L2 being switched to level-1 of approximation, App-3-Appx-L1. **Scenario (f):** Power consumption is well below TDP, leaving ample budget to utilize. This allows cranking up voltage and frequency levels of applications, hence all the applications are operating at their maximum required voltage and frequency levels $(v_M,f_M)$. However, at instance $t_7$, the request rate is high, although TDP is honored. In this case, to decrease the AWT, App-8 is switched to approximate mode at level-1, App8-Appx-L1.

# 4. SYSTEM DESIGN

Fulfilling the objectives of power capping while maintaining better throughput requires a power management framework that *monitors* critical chip parameters of power consumption, performance, utilization and network intensity, *decides* on required optimization and *acts* upon the optimization in an observe-decide-act (ODA) loop. We design our power management framework in such an ODA loop fashion for a NoC-based many-core systems. Subject to application requests, the workload, power consumption, utilization, network intensity of the chip varies. Our power management framework monitors these metrics and in case of power violations and higher workload intensities, it decides on actuation of different knobs. The system architecture is shown in Figure 4, and detailed below.

## 4.1 System Architecture

We present our power management framework for NoC based many-core system that supports dynamic arrival and servicing of applications. We use applications that are modeled as directed task graphs, where each task runs concurrently with other tasks. Each task has its computational intensity and its communication volume with other tasks. Applications are classified as approximable and non-approximable. Approximable applications are those that are modeled as compound task graphs, such that one or more

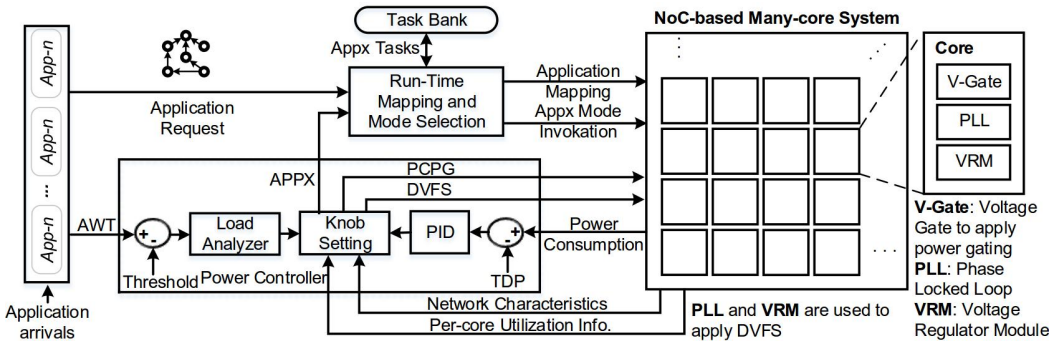



**Figure 4: Power Management Framework**

tasks of the application can be replaced by their approximate versions. Incoming applications arrive at the application repository and make an application request to be serviced. The run-time mapping and mode selection unit (RMSU) is responsible for servicing these requests by allocating on-chip resources per application. RMSU handles application mapping, a one-to-one function of allocating a core-per-task for all tasks of the application. In our framework, we use pro-active application mapping *MapPro*, presented in [10]. Parallel execution of multiple incoming applications is possible with RMSU support. RMSU communicates with power controller to send mapping information of current set of applications running on-chip and for actuation decisions, described in sections below.

### 4.1.1 Power Manager

Power Manager is the central controller that monitors system metrics and decides of actuation of different knobs. Actuation decisions of the power manager are based on per-core power consumption from power meters, per-core utilization from performance counters, network intensity from router buffers, amount of workload and their characteristics from application repository (i.e., approximable or non-approximable). Each core on the network is equipped with per-core power and utilization meters and we trace network intensity as a moving average of packet flow through the buffers at each router. These metrics are sent to the power manager, forming the power monitoring phase. We accumulate wait-time of all the applications that have made an execution request and are currently waiting in the queue to be serviced. We send the accumulated wait-time (AWT), as in Equation 1, to the power manager, forming the workload monitoring phase. We set another parameterizable threshold $TDP_{th}$, a metric that indicates potential TDP violation, such that $0.66 \times TDP < TDP_{th} < TDP$. Actuation decisions of our power manager for DVFS and PG knobs are based on power controller presented in [20]. DVFS and PG actuations are applied to the chip, as shown in Figure 4. In case of high workload ($AWT > AWT_{th}$), the power manager invokes APPX knob, and chooses the application(s) that can be switched to approximate mode. The difference between AWT and the threshold $AWT_{th}$ determines the level of approximation. The power manager sends the APPX knob invocation, the application chosen to be approximated and the level of approximation to the RMSU. For evaluation, we currently use two levels of approximation in increasing order of accuracy trade-offs. Alternatively, several fine-grained levels of accuracy trade-offs could be used.

**Mode Switching**: When the current mode of execution is accurate, the RMSU originally maps the accurate version of the task graph. If the application is approximable, the RMSU buffers approximate tasks of the application into the *Task Buffer*. Formulation of the compound task graphs is shown in Figure 5. We generate compound task graphs that

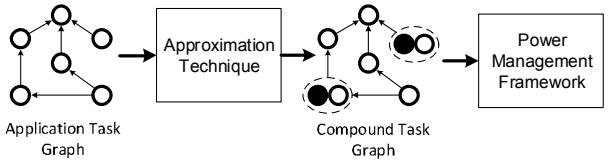



**Figure 5: Compound Task Graph - Workflow**

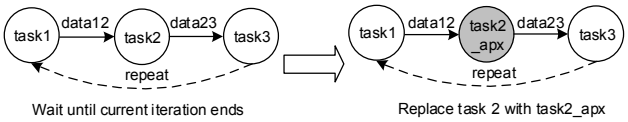



**Figure 6: Mode Switching**

include multiple versions of tasks that are approximable, shown in dotted lines. Depending on APPX knob setting, RMSU chooses the version of task to be included in the application mapping, while the other versions are buffered. With the invocation of APPX knob, there are two possible scenarios for mode switching viz., i) mapping approximate task graphs and ii) switching mode of execution of applications currently running by task replacement. In the former case, the RMSU maps every incoming application in its approximate version by including the approximable tasks instead of accurate tasks, until the mode is switched back to accurate. In the latter case, power manager chooses the application(s) and level of approximation to switch to. Based on these, RMSU identifies the corresponding approximate task from the *Task Buffer* and replaces the accurate task with the approximate task. We modeled applications for evaluation as data dependent concurrent tasks that execute periodically. The computational process repeats until the end of execution with a specified periodicity. Streaming and signal processing applications are good examples which execute periodically over incoming samples of data, where it is possible to relax certain aspects of computation when new data arrives every period. When RMSU has to replace an accurate task with approximate, it lets the current iteration of accurate task's computation to finish execution. It waits until data from the accurate task is received at its destination end task (if any). Once the data transfer is completed, the RMSU loads the approximate task on to the chip, replacing the accurate task. Figure 6 shows the process of task replacement during mode switching. The example has three tasks 1, 2 and 3 out of which task2 is approximable. On invocation of APPX knob, the switching happens in the following sequence. i) The RMSU finds the approximate task *task2_appx* from the *Task Buffer*. ii) It waits until data from task2 (data23) is received at task3. iii) *task2_appx* is loaded by fetching the instruction stream into the cache. iv) After the data is received at task3, the execution of task2 will now start from new instruction stream of *task2_appx*. De-

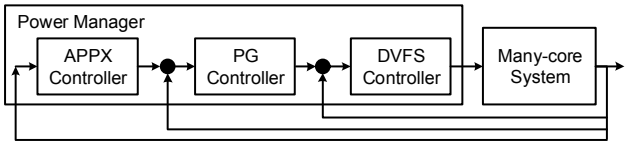



**Figure 7: Hierarchy of Knobs**

**Algorithm 1** Power Management Algorithm

**Inputs:** $P$: Instantaneous Power, $AWT$: Accumulated Waiting Time;
**Outputs:** $RMSU.inMode$, $RMSU.runMode$, $RMSU.level$: Mode switching commands for RMSU;
**Constants:** $TDP$: Power budget, $AWT_{th}$: $AWT$ Threshold
**Global Variables:** $App_R$: Applications currently running, $App_X$: Applications approximable, $App_Y$: Applications running in approximate mode;

**Body:**

```
1: while AWT ≥ AWT_th do
2:    while P ≥ TDP do
3:       DVFS_down(Apps_R);
4:       if success then break;
5:       PCPG(cores_un);
6:       if success then break;
      RMSU.inMode = APPX;
7:    for App_X do
8:       RMSU.runMode(App_X, level-1);
9:    if (success) then break;
10:   for App_Y do
11:      RMSU.runMode(App_Y, level-2);
12:   if success then break;
13: while AWT < AWT_th do
14:   while P < TDP do
15:      DVFS_up(Apps_R);
16:      if fail then break;
17:      powerup(cores_un);
18:      if fail then break;
19:   RMSU.inMode = ACC;
20:   if fail then break;
```

pending on size of the instruction cache used, instructions of task2 may require flushing, however this is subject to hardware platform. Since the computational process of the application is periodic in nature, data is not changed with mode switching and moving the data or flushing the data cache is not needed. The state of the application is hence preserved at the end of the period. It is to be noted that task migration [17] has an appreciable overhead in moving both instructions and data, which is a widely used approach in dynamic power and thermal management. In comparison, the mode switching overhead is lesser, as it involves insertion of new instructions alone and does not need any data accesses. The overhead incurred in mode switching is elaborated in Section 5.

**Mode Switching Vs Dynamic Knobs:** Hoffman *et al.* [11] have used dynamic knobs that reside in program space to scale accuracy of applications. However, this restricts accuracy trade-offs only to applications with possibility of specific relaxed execution that do not require compile time support. Approximation techniques that use logic simplification and minimization with a different implementation need compile time support, and cannot take advantage of dynamic knobs approach. Mode switching overcomes this limitation, widening the scope to several application domains and approximation techniques. Dynamic knobs can be treated as a best case sub-set of mode switching.

## 4.2 Power Management Algorithm

We employ the DVFS and PG knobs synergistically with APPX in a hierarchical way, as shown in Figure 7. Triggering and disciplined tuning of these knobs together for power capping is handled by a power management and mode

**Algorithm 2** Mode Switching Algorithm

**Inputs:** $inMode$: Execution mode for incoming applications, $runMode$: Execution mode of currently running applications; $App_{in}$: Incoming applications,$App'_{in}$: Approximate version of incoming application, $App_R$: Application currently running;
**Outputs:** $Map$: Mapping configuration of incoming application;
**Constants:** $TDP$: Power budget, $AWT_{th}$ : $AWT$ Threshold

**Body:**

```
1: if newApp then
2:    taskBuffer.push(App.T', App.T'');
3:    if inMode = APPX then
4:       Map(App'_in);
5:    else
6:       Map(App_in);
7: for App_R do
8:    if App_R.runMode = level-1 then
9:       wait until current iteration of T finishes;
10:      switch(T,taskBuffer(T'));
11:   else if App_R.runMode = level-2 then
12:      wait until T' finishes;
13:      switch(T',taskBuffer(T''));
```

switching algorithms. Our hierarchical power management algorithm is presented in Algorithm 1. In cases of power violation, it first applies DVFS knob to downscale the voltage and frequency of applications that are currently running on the chip. For voltage down/up scaling (in Algorithm 1 lines 3-4, 12-14), we use the approach presented in [20]. If power violation persists, the PG knob is used to power gate all the cores that are currently un-occupied by any tasks on the chip. Although, this leads to increase in dark silicon, and further accumulation of the same happens when workload intensity is higher. In case of AWT exceeding the threshold, APPX knob is invoked by switching the mode to approximate execution. A decision to map any incoming applications in their approximate mode is indicated to the RMSU. The applications currently running on the chip that are approximable are switched to their approximate mode. A mapping command is sent to the RMSU indicating the application(s) to be switched and level of approximation. If AWT is still violated, applications that are already running in level-1 approximation are further switched to level-2 of approximation. The level of approximation can be set in a fine-grained manner by using N-levels of accuracy trade-offs. For instance, Palomino *et. al* have used a 4-level approximation in video encoding to optimize chip's temperature [18]. In this work, we use two levels of approximation to demonstrate energy gains. If TDP is honored, voltage and frequency levels of applications that are currently down-scaled are cranked up. If the TDP is still honored after voltage upscaling, un-occupied cores that are power gated are powered up. While AWT is honored, mode of incoming applications is changed back to accurate. If AWT is honored further, a mapping command indicating the application and new level of approximation (i.e., more accurate level) is sent to the RMSU. Mapping and mode switching decisions made by RMSU are presented in Algorithm 2. The RMSU receives mode of execution for incoming applications from the power manager. Upon arrival of a new application, the RMSU buffers approximate tasks of the application (if any) into the Task Buffer. By default, RMSU maps accurate version of an application on to the chip if the *inMode* is ACC, while the approximate mode is mapped when *inMode* is APPX. RMSU receives mode switching commands for current set of applications running on the chip. The *runMode* command indicates the application whose task has to be replaced and the level of approximation. When *runMode* of a specified application is set to level-1. the RMSU fetches corresponding approximate task ($T'$) from the Task Buffer and replaces the accurate task with approximate task. In case of *runMode* of an application being set to level-2. RMSU repeats the afore-

mentioned switching process, with an approximate task of level-2. When the power and workload intensity are manageable within their thresholds, we switch back to ACC mode by indicating the same to the RMSU. Inspired by the technique presented in [19], we avoid oscillating between ACC and APPX by setting intermediary thresholds $TDP_{th}$ and $AWT_{th}$. We switch from ACC to APPX mode when the power and workload are approaching their thresholds, while switching back to ACC mode is invoked only when these parameters approach intermediary thresholds. This ensures that switching back to ACC mode is done when power and workloads are maintained well under their thresholds and minimizes the chance of oscillating between the two modes frequently. Setting intermediary thresholds is a designer's decision, with a choice between quality and throughput.

# 5. EVALUATION

We assess the efficiency of our approximation-enabled power management approach, APPEND, against state-of-the-art dynamic power management/capping techniques PG [7] (based on PCPG) and and MOC [20] (based on per-core DVFS and PG). We chose widely used data-triggered on-line learning applications that fall under classification and estimation. They are inter-disciplinary, being used in a range of periodic applications like recognition. mining and automation that are performance and energy demanding. These workloads are based on iterative methods of computation, meaning that the accuracy of result converges towards optimal solution with more number of iterations. Since an accurate solution may not exist and lower convergence could still offer an acceptable result, they become candidates for approximation. The list of applications used for evaluation is presented in Table 1. We normalize the performance gain of approximate tasks of level-1 (AP-1) and level-2 (AP-2) in comparison with their accurate versions. For level-1 and level-2 of approximation with loop perforation (LP) for linear regression and least squares. we skip 10% and 25% of computations on input data respectively. For k-means clustering and k-nearest neighbors, we use relaxed convergence (RC). We compromise on number of flips, coverage of neighbors and training data sets respectively for these applications. We set the limits of relaxation on convergence to 1% and 5% respectively for two levels of approximation. For each application and level of approximation, we present normalized energy gain when compared to accurate tasks. It can observed that performance and energy gain increases with amount of accuracy traded. These gains come at the loss of accuracy, ranging from 3% to 13% over different applications and data sets. Accuracy trade-offs and overheads incurred in switching mode of execution from accurate to approximate are reasoned in Section 5.2.1.

## 5.1 Simulation Environment

Applications are modeled as task graphs, as described in Section 4. We implement each application on interval-core based Sniper simulator, annexed with McPAT for modeling power [23]. We used Nehalem-like processing elements with 32KB of instruction and data caches. We model each application as a combination of concurrent tasks, preserving data flow nature. We use loop perforation and relaxed convergence in case of approximate tasks. We extract execution time, average power and energy consumption per each task. We normalize these values as *compute factor* metric for each node in the task graph. along with amount of data flow as the *communication volume* between tasks. The task graph for each application is thus a directed network of nodes that holds execution time, communication volume, average static and dynamic power consumption for accurate and approximate tasks. We use our in-house cycle accurate

**Table 1: Applications' Energy-Accuracy Trade-offs**

| App | Appx | Norm. Perf | | Norm. Energy | | % Error | |
|---|---|---|---|---|---|---|---|
| | | AP-1 | AP-2 | AP-1 | AP-2 | AP-1 | AP-2 |
| Linear Regression | LP | 1.1 | 1.2 | 1.16 | 1.27 | 6 | 13 |
| K-Means | RC | 1.93 | 5.6 | 1.9 | 5.27 | 1 | 5 |
| K-NN | RC | 1.11 | 1.33 | 1.09 | 1.22 | 3 | 8 |
| Least Squares | LP | 1.07 | 1.26 | 1.11 | 1.25 | 5 | 12 |

simulator implemented in SystemC to evaluate the proposed power management framework. We extended Noxim [6] NoC simulator using its network infrastructure for interconnects. The power characteristics of processing elements (PE) are modeled based on metrics extracted from McPAT and Lumos [25]. Lumos is an analytical framework that quantifies power-performance characteristics with technology node scaling for many-core systems. We used Lumos for physical scaling parameters, voltage scaling and TDP metric for different network sizes. We added the support for dynamic arrival and servicing of applications through the run-time mapping unit. The mapping unit receives commands from power controller. implemented as a software module. The test-bed is a rectangular network with X-Y routing. The $tile_{(0,0)}$ of the mesh acts as the central manager that is responsible for keeping track of mapping information. The network size is 12×12 and the chip area is $138mm^2$. For the *first node* selection in the runtime mapping process, we use MapPro [10] method. For the DVFS purpose, we use 15 VF levels with voltage in the range of 0.8V-1.2V. The frequency of the on-chip communication network (e.g., routers) is set to the maximum level (similar to [7] and [20]). The TDP value is set to 90W, based on the chip's power density.

## 5.2 Evaluation Metrics and Results

For evaluation purposes, we simulate the system over a period in which 200 applications are serviced. The evaluation metrics are: i) *Power Consumption*: Power consumption of the system over the period of execution, honoring TDP by capping the power, ii) *Accumulated Wait-time*: Accumulated value of wait-time of applications before the application request is serviced. and iii) *Throughput*: Time consumed to service 200 applications Our pre-requisite objective is to cap the power consumption such that TDP constraint is honored throughout the period of execution. Figure 8 shows the power consumption of DVFS, PG, MOC and APPEND, along with TDP constraint. over the execution time for servicing 200 applications. TDP violation is more frequent with PG and DVFS knobs, while TDP is honored for most of the execution period with MOC and APPEND, with APPEND being better of the two. Noticeable issue is that while honoring TDP, APPEND maintains power consumption closest to TDP when compared to other knob combinations, reflecting better utilization of available power budget. This indicates mitigation of dark silicon and can be attributed to hierarchical usage of power knobs in APPEND's power controller. Moreover, we actuate power knobs - DVFS and PG by monitoring power consumption over an epoch $e_1$ and trigger the approximation knob pro-actively over epoch $e_2$ with $e_2$ being five times longer than $e_1$. This eliminates possible random actuations or oscillations between different modes of execution. With better utilization, APPEND is able to service applications faster, reducing the run-time and consequently wait-time of incoming applications. Figure 9 shows the accumulated wait-time ($AWT$) for different power capping actuators over the period of execution. We present $AWT$ as a function that is directly related to rate of application requests made. Similar to power capping, APPEND has the best $AWT$, preceded by MOC, PG and DVFS. DVFS and PG based actuations have higher AWTs already when the application request rate reaches 3 per second. MOC has a

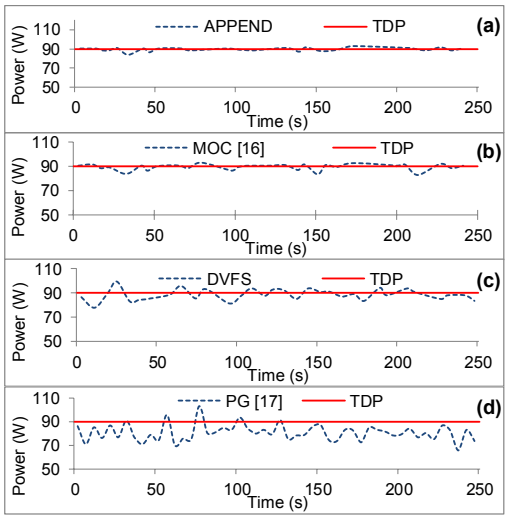



**Figure 8: Instantaneous Power Consumption**

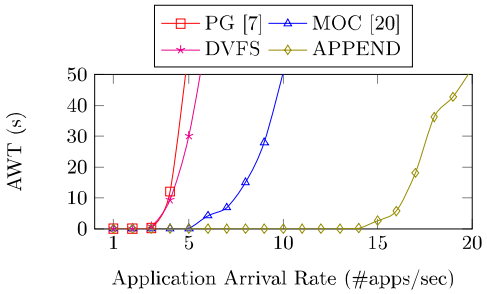



**Figure 9: Accumulated waiting time**

relatively high AWT when application request rate is 5 per second. However, APPEND has a near-zero AWT for as long as 5× more than DVFS and PG and 3× more than that of MOC. This demonstrates the ability of APPEND to service applications faster despite high workloads, when compared to the other knobs. APPEND has *AWT* greater than zero when the application request rate reaches 14 per second. Also, *AWT* accumulation is more steeper in case of other knobs than that of APPEND, indicating a substantial rise in their wait-times with high request rates. The minimal *AWT* and high service rate of APPEND also results in high throughput and energy efficiency. Normalized gain in throughput for all knob combinations is shown in Figure 10. APPEND has a throughput that is 1.5× better than PG and 1.2× better than MOC, showing a significant gain in performance and energy while power capping is strictly maintained. Employing APPX knob allows APPEND to minimize execution time of applications running on the chip. With applications leaving the system faster, more resources (cores) become available for incoming applications and reduces their wait-time. APPEND benefits from AWT and throughput mutually improving each other.

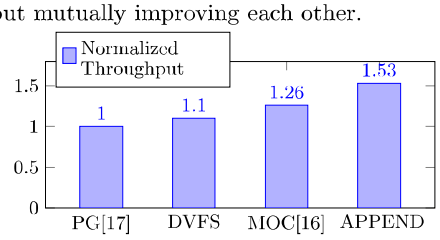



**Figure 10: Normalized throughput**

**Table 2: Applications - Behavior and Overhead**

| App | Instr. | | Instr. Sim (M) | | L1-I Accesses (M) | | Overhead % (Norm.) |
|---|---|---|---|---|---|---|---|
| | Acc | Appx | Acc | Appx | Acc | Appx | |
| Linear Regression | 1105 | 1150 | 93 | 75 | 9 | 7 | 1.2 |
| K-Means | 1018 | 1021 | 449 | 102 | 57 | 14 | 0.3 |
| K-NN | 1457 | 1513 | 140 | 105 | 37 | 24.7 | 0.3 |
| Least Squares | 770 | 815 | 53 | 50 | 5 | 3.7 | 1.2 |

### 5.2.1 *Error and Overhead Analysis*

Behavioral patterns of accurate (Acc) and approximate (Appx) versions of each application are shown in Table 2. The number of instructions of each applications (Instructions), number of instructions simulated (Instr. Sim (M) in million), number of L1-instruction cache accesses (L1-I Accesses (M)) (in million), and normalized overhead (in %) are presented in Table 2. Number of instructions are slightly higher for approximate tasks due to conditional branching involved. However, these instructions eventually result in reduced overall workload and hence improve performance. For each application, we used 1 million elements in training set in increasing steps of 100000 data points per period. Number of simulated instructions depend on training and test data sets used. and are variable in case of different sizes of data used. With loop perforation and relaxed convergence, input data elements are skipped, resulting in fewer instructions required to be simulated. Normalized energy savings, performance gain and loss in accuracy for each application are shown in Table 1. For loop perforated (LP) applications, we used squared distance from accurate solution to calculate the error and for relaxed convergence (RC), we set 1% and 5% as limits for convergence. Switching execution from accurate to approximate version incurs some overhead due to monitoring and triggering the approximate version. For every approximate task, the switching of execution mode involves a conditional branching instruction(s). The overhead incurred during this transformation included in the approximate task's *compute factor*. For the applications we used, the normalized overhead penalty incurred ranged between 0.3% up to 1.2%. This overhead is negligible when compared to the workload reduced by approximation and thus levies no significant performance penalty. Further, loading an approximate task involves moving new instruction stream to the instruction cache, with a possibility of increase in the number of DRAM accesses. However, this depends on the number of application instructions and the size of L1-instruction cache. For instance, a larger application coupled with smaller L1-instruction cache presents a worst case scenario that would force the system to evict accurate task and fetch the approximate task from main memory. Although, in our testbed, we used L1-I cache of 32KB and all the applications have instructions up to as many as 1500. The worst case penalty in terms of communication for switching from accurate version of a task to the approximate version can be calculated as follows:

$$penalty = \frac{size_{app}}{size_{pkt}} \times (P_L + (n \times r_L) + MC_L + DRAM_L) \quad (2)$$

where $size_{app}$ and $size_{pkt}$ are application and packet sizes, $P_L$ is packetizing latency i.e., time consumed to packetize data, access the network interface and inject packets into the network, $n$ is the number of hops from a core to nearest memory controller, $r_L$ is router channel latency, $MC_L$ and $DRAM_L$ are access latencies of memory controller and off-chip memory. We demonstrate worst case overhead penalty of mode switching for a video encoding application which was used by Holmbacka *et al.* as an example that they used to demonstrate overhead for task migration [12]. For experimental many-core platform Intel SCC, the core, net-

work and off-chip memory frequencies are 533MHz, 800MHz and 400MHz, respectively. The worst case mode switching penalty using SCC for the video encoding application of size 6KB is 1.5ms. For the same application, penalty in task migration is 10.6ms, 7× more than the mode switching overhead, to move both instructions of 6KB and data of 16KB. Task migration overhead can still be higher when more data is to be moved, while mode switching needs no movement of data. It should be noted that these values are subjective to the platform on which they are executed, while the relative difference in overheads between mode switching and task migration might hold good.

# 6. RELATED WORK

**Power Capping**: Adaptive Power capping techniques monitor the power consumption and actuate power knobs in a closed loop, in case of TDP violation. A PID controller based power management is presented in [7], where knob settings are actuated for power capping as per normalized gain of PID. Vega et. al propose a power capping algorithm using DVFS, PCPG and core folding, with all power knobs tightly coupled [24]. They suggest that combinatorial usage of different power knobs is effective for system level power capping decisions. Cochran *et al.* have used thread packing i.e., allocation of threads per core as a power knob along with adaptive DVFS [4]. PGCapping was presented in [15] that uses PCPG and DVFS in a hierarchical way for power capping and life time balancing. Kapadia *et al.* have used Degree-of-parallelism (DoP) as a knob for power management and to improve system reliability. Application mapping i.e., spatial alignment of active cores for improving power budget and thus power capping limit was proposed in [13] and [8]. A multi-objective power capping approach was presented in [20] which uses combination of DVFS and PCPG based on network and workload characteristics. Chen et. al have proposed using resource allocation at data center level as another knob for power actuation [3]. They use history based prediction for potential workload to determine CPU resource allocation. While all the above techniques use TDP as upper bound, Pagani et. al have proposed an adaptive way way of setting the upper bound on power consumption, thermal safe power (TSP), as a function of spatial alignment of active components. [9].

**Approximation**: Ansel *et al.* have used variable accuracy implementations of same algorithm, with language and compiler support to choose one among different implementations for exploring energy-accuracy trade-offs. [1]. Back and Chilimbi have proposed approximation at software level with a choice between accurate and approximate versions of blocks of code using Green compiler [2]. Hoffman *et al.* have proposed using energy-accuracy trade-offs in context of power capping by translating static parameters of an application into dynamic knobs such as convergence for drop in accuracy [11]. However, other approximations at algorithmic level such as logic simplification cannot be translated into dynamic knobs. Escaping infinite loops and skipping iterations of long bottleneck loops was proposed by Sidiriglou *et al.* as Loop Perforation [22]. All these techniques explore ways to compute approximately, keeping quality control, energy and performance gains in view. However, they do not use approximation for actuating power consumption in a closed-loop way.

# 7. CONCLUSIONS AND FUTURE WORK

In this work, we proposed approximation as another knob for power capping and management in many-core systems. We used the APPX knob hierarchically with other power knobs of DVFS and PG to gain performance and energy within the power budget, for some accuracy trade-offs. We presented a power management framework, APPEND, that monitors power consumption and workload intensities to dynamically replace accurate tasks with approximate tasks. We used multiple variable accuracy implementations of the same application to support dynamic switching between tasks. APPEND thoroughly honors TDP and yields higher throughput and lower wait-times, in comparison with state-of-the-art power management techniques. Our proposed approach requires application design support and restricts the accuracy trade-offs to a fixed scale. Making the system aware of possibilities in accuracy scaling for a fine-grained and dynamic approximation is planned for future work.

# 8. REFERENCES


[1] J. Ansel et al. PetaBricks: a language and compiler for algorithmic choice. *ACM SIGPLAN Notices*, 2009.

[2] W. Baek et al. Green : A Framework for Supporting Energy-Conscious Programming using Controlled Approximation. In *PLDI*, 2010.

[3] H. Chen et al. Dynamic server power capping for enabling data center participation in power markets. In *ICCAD*, 2013.

[4] R. Cochran et al. Pack & cap: adaptive dvfs and thread packing under power caps. In *MICRO*, 2011.

[5] H. Esmaeilzadeh et al. Dark silicon and the end of multicore scaling. In *ISCA*, 2011.

[6] F. F. et al. Noxim: Network-on-chip simulator. *URL: http://sourceforge.net/projects/noxim*, 2008.

[7] M.-H. H. et al. Dark Silicon Aware Power Management for Manycore Systems under Dynamic Workloads. In *ICCD*, 2014.

[8] M. S. et al. Dark Silicon As a Challenge for Hardware/Software Co-design. In *CODES+ISSS*, 2014.

[9] S. P. et al. TSP: Thermal Safe Power: Efficient Power Budgeting for many-core systems in dark silicon era. In *CODES+ISSS*, 2014.

[10] M. Haghbayan et al. MapPro: Proactive Runtime Mapping for Dynamic Workloads by Quantifying Ripple Effect of Applications on NoCs. In *NOCS*, 2015.

[11] H. Hoffmann et al. Dynamic knobs for responsive power-aware computing. *ACM SIGPLAN Notices*, 2012.

[12] S. Holmbacka et al. A task migration mechanism for distributed many-core operating systems. *Journal of Supercomputing*, 68(3), 2014.

[13] A. Kanduri et al. Dark silicon aware runtime mapping for many-core systems: A patterning approach. In *ICCD*, 2015.

[14] N. Kapadia et al. VARSHA: Variation and Reliability-aware Application Scheduling with Adaptive Parallelism in the Dark-silicon Era. In *DATE*, 2015.

[15] K. Ma and X. Wang. PGCapping: Exploiting power gating for power capping and core lifetime balancing in CMPs. *PACT*, 2012.

[16] S. Misailovic et al. Chisel: Reliability- and accuracy-aware optimization of approximate computational kernels. In *OOPSLA*, 2014.

[17] T. Muthukaruppan et al. Hierarchical power management for asymmetric multi-core in dark silicon era. In *DAC*, 2013.

[18] D. Palomino et al. Thermal optimization using adaptive approximate computing for video coding. *DATE*, 2016.

[19] A. Rahmani et al. Design and management of high-performance, reliable and thermal-aware 3D networks-on-chip. *IET Circ., Dev. & Sys.*, 2012.

[20] A. Rahmani et al. Dynamic power management for many-core platforms in the dark silicon era: A multi-objective control approach. In *ISLPED*, 2015.

[21] A. Rahmani et al. *The Dark Side of Silicon*. 2016.

[22] S. Sidiroglou et al. Managing performance vs. accuracy trade-offs with loop perforation. In *FSE*, 2011.

[23] E. Trevor et al. Sniper: Exploring the level of abstraction for scalable and accurate parallel multi-core simulations. In *SC*, 2011.

[24] A. Vega et al. Crank it up or dial it down: Coordinated multiprocessor frequency and folding control. In *MICRO*, 2013.

[25] L. Wang and K. Skadron. Dark vs. dim silicon and near-threshold computing extended results. *Univ. of Virginia, Dept of Comp.Sci Technical Report*, 1, 2012.

Paper IV

# Accuracy-Aware Power Management for Many-Core Systems Running Error-Resilient Applications

Anil Kanduri, Hashem Haghbayan, Amir Rahmani,Pasi Liljeberg, Axel Jantsch, Hannu Tenhunen and Nikil Dutt.

Published in IEEE Transactions on VLSI Systems (TVLSI), 2017, Volume 25, Issue 10, pp: 2749 - 2762

# Accuracy-Aware Power Management for Many-Core Systems Running Error-Resilient Applications

Anil Kanduri, *Senior Member, IEEE*, Mohammad-Hashem Haghbayan, *Senior Member, IEEE*, Amir M. Rahmani, *Member, IEEE*, Pasi Liljeberg, *Member, IEEE*, Axel Jantsch, *Member, IEEE*, Hannu Tenhunen, *Member, IEEE*, and Nikil Dutt, *Fellow, IEEE*

***Abstract*— Power capping techniques based on dynamic voltage and frequency scaling (DVFS) and power gating (PG) are oriented toward power actuation, compromising on performance and energy. Inherent error resilience of emerging application domains, such as Internet-of-Things (IoT) and machine learning, provides opportunities for energy and performance gains. Leveraging accuracy-performance tradeoffs in such applications, we propose approximation (APPX) as another knob for close-looped power management, to complement power knobs with performance and energy gains. We design a power management framework, APPEND+, that can switch between accurate and approximate modes of execution subject to system throughput requirements. APPEND+ considers the sensitivity of the application to error to make disciplined alteration between levels of APPX such that performance is maximized while error is minimized. We implement a power management scheme that uses APPX, DVFS, and PG knobs hierarchically. We evaluated our proposed approach over machine learning and signal processing applications along with two case studies on IoT—early warning score system and fall detection. APPEND+ yields 1.9× higher throughput, improved latency up to five times, better performance per energy, and dark silicon mitigation compared with the state-of-the-art power management techniques over a set of applications ranging from high to no error resilience.**

***Index Terms*— Approximate computing, dark silicon, Internet-of-Things (IoT), power management, runtime mapping.**

## I. Introduction

EMERGING application domains, such as Internet-of-Things (IoT), cyber–physical systems (CPSs), big data analytics, and so on, are compute intensive and power hungry [1]. Transistor scaling supported building denser chips that provide higher compute intensity to meet performance requirements of these applications, while keeping the power density constant. With transistor scaling reaching its physical limit, operating voltage approaches its threshold and cannot be further scaled down gracefully with transistor scaling [2]. This leads to rise in power density and subsequently thermal violation. Performance surges of emerging application domains, smaller chip areas, and limited cooling solutions contribute to high power densities and frequent thermal violations, potentially damaging the chip's functionality. To avoid thermal violations, the chip has to function within dissipatble (safe) limits of power. This forces a section of chip to be powered off temporally—this inactive portion is termed as dark silicon [3]. Dark silicon phenomena reduce performance, energy efficiency, and utilization of on-chip resources [2].

Power capping techniques are used to restrict power consumption of the chip to a fixed and safer limit, typically a design time estimate called thermal design power (TDP), beyond which thermal violations may occur [4]. Dynamic power capping and management techniques typically function in an observe-decide-act loop, observe instantaneous power consumption and temperature accumulation, decide on power actuation, and act on the decisions through power knobs [5]. Dynamic voltage and frequency scaling (DVFS), power gating (PG) [6], near threshold computing [7], and adaptive scheduling [8] are widely used knobs for power management. Combinatorial actuation of different power knobs can honor thermal and power constraints, although performance and/or energy gains can be minimal [9]. DVFS knob would be limited as the voltage approaches its threshold and cannot be scaled down any further and also suffers with increase in leakage power. PG knob addresses the issue of static power and is not limited as DVFS. However, the reduction in static power comes at the expense of performance, since only fewer cores are simultaneously powered up. Both DVFS and PG are triggered as a reaction to power violations, which might work for instantaneous power reduction in short term. Despite power capping benefits, they do not offer any substantial gains on performance or energy efficiency.

Approximate computing is emerging as an alternative for dark silicon mitigation, by trading off accuracy for performance and energy gains. Applications from several domains are inherently error resilient, based on their nature of computation and/or input data. For example, algorithms used in multimedia signal processing, machine learning, numerical methods, and so on can be iterative or NP-hard, making them

Manuscript received August 16, 2016; revised December 5, 2016 and February 20, 2017; accepted March 21, 2017. Date of publication April 28, 2017; date of current version September 25, 2017. This work was supported by the Marie Curie Actions of the European Union's H2020 Programme. *(Corresponding author: Anil Kanduri.)*

A. Kanduri, M.-H. Haghbayan, and P. Liljeberg are with the University of Turku, 20500 Turku, Finland (e-mail: spakan@utu.fi; mohhag@utu.fi; pakrli@utu.fi).

A. M. Rahmani is with the University of California Irvine, Irvine, CA 92617 USA, and also with TU Wien, 1040 Vienna, Austria (e-mail: amirr1@uci.edu).

A. Jantsch is with TU Wien, 1040 Vienna, Austria (e-mail: axel.jantsch@tuwien.ac.at).

H. Tenhunen is with the University of Turku, 20500 Turku, Finland, and also with the KTH Royal Institute of Technology, 16440 Stockholm, Sweden (e-mail: hannu@kth.se).

N. Dutt is with the University of California Irvine, Irvine, CA 92617 USA (e-mail: dutt@uci.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TVLSI.2017.2694388

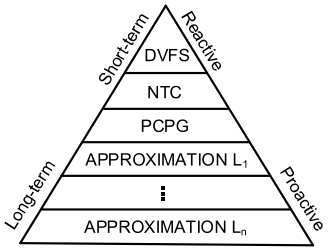



Fig. 1. Power management knobs.

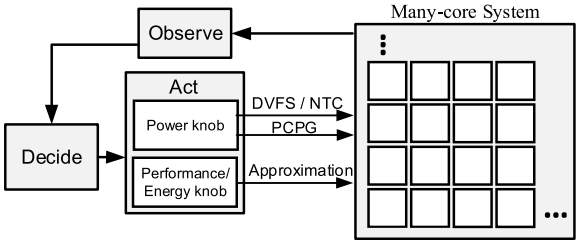



Fig. 2. Approximation knob.

tolerant to inaccurate computations. Similarly, IoT systems deal with continuous sensory data originating from analog or noisy sources, where relaxing certain computations has less effect on eventual result. Approximation (APPX) leverages inherent error resilience of such applications to reduce computational workload and increase energy efficiency.

Alongside soft computing applications, several IoT applications rely on a smart gateway for edge and/or cloud-based processing [10]. With several sensor nodes mapped to a single edge or cloud processor, performance and power challenges continue to effect IoT applications indirectly. Traditional power knobs alone would not suffice to ensure power capping while maintaining higher performance and energy efficiency. To fill the performance and energy gap of power knobs, we propose APPX as another knob for power capping and management. We couple the short-term reactive power knobs, DVFS and PG, with the long-term proactive energy and performance knob, APPX, in a hierarchical manner for power capping, while ensuring performance and energy gains, at the expense of accuracy. Fig. 1 shows power knobs classified in the order of their temporal effect and overhead.

In this paper, which is a major extension of our recent work published in [11], we propose an APPX-enabled power management framework APPEND+ that uses DVFS and PG knobs primarily for power capping and APPX knob for performance. Fig. 2 shows the top-level view of APPX as another knob for power management along with conventional power knobs. We design a power manager that makes decisions on the actuation of DVFS and PG knobs in case of power violation and APPX knob in case of throughput violation. The key idea of this paper is to switch the mode of execution of an application from accurate to approximate upon performance requirements with APPX knob invocation. In our previous work, we switch from accurate to approximate mode of execution among approximable applications, subject to system requirements [11]. At times, this strategy either over compensates for performance surges by approximating beyond the requirement, or falls short of meeting the performance requirement. We fill this gap using *sensitivity metric* for each application at each level of APPX that is used to make mode switching decisions. We use a set of variable accuracy implementations of an approximable task, with each approximate task identified by its sensitivity metric, which is represented as the performance gained per error induced. To enable selection of suitable candidates for mode switching, we prune this set to choose a candidate task for replacing the accurate task that offers maximal performance gain within minimal error. We present a run-time mapping and mode switching algorithm for replacing accurate tasks with approximate tasks from the set of variable accuracy implementations. Our contributions based on our prior work [11] are as follows:

1) APPX knob for closed loop power and performance management;
2) a classification algorithm for identifying approximable tasks that maximizes performance by pruning application space based on sensitivity metric;
3) a run-time mapping and mode switching technique for replacing accurate tasks with approximate tasks;
4) a power management framework APPEND+ that uses DVFS, PG, and APPX hierarchically for power capping and throughput improvement;
5) a case study of IoT applications, fall detection and early warning score (EWS), to evaluate APPEND+.

## II. Related Work

### A. Power Capping

Power capping techniques monitor the power consumption and actuate power knobs in a closed loop, in case of power consumption exceeding TDP. A PID controller-based power capping is presented in [12], where knob settings are actuated as per normalized gain of PID. Vega *et al.* [13] propose a power capping algorithm using DVFS, per-core PG (PCPG), and core folding, with all power knobs tightly coupled. They suggest that combinatorial usage of different power knobs is effective for system level power capping decisions. Cochran *et al.* [14] have used thread packing, i.e., allocation of threads per core as a power knob along with adaptive DVFS. PGCapping was presented in [6] that uses PCPG and DVFS in a hierarchical way for power capping and life time balancing. Kapadia and Pasricha [8] have used degree-of-parallelism as a knob for power management and to improve system reliability. Application mapping, i.e., spatial alignment of active cores for improving power budget and thus power capping limit, is proposed in [15] and [16]. A multiobjective power capping approach is presented in [15] and [16], which uses the combination of DVFS and PCPG based on network and workload characteristics. Chen *et al.* [18] have proposed using resource allocation at data center level as another knob for power actuation. They use history-based prediction for potential workload to determine CPU resource allocation. While all the above-mentioned techniques use TDP as upper bound, Pagani *et al.* [4] have proposed an adaptive way of setting the upper bound on power consumption, thermal safe power (TSP), as a function of spatial alignment of active

components. All these techniques focus exclusively on power capping and combinatorial usage of power knobs, but do not consider their implications on performance.

### B. Approximation

Ansel *et al.* [19] have used variable accuracy implementations of the same algorithm, with language and compiler support to choose one among different implementations for exploring energy-accuracy tradeoffs. Baek and Chilimbi [20] have proposed APPX at software level with a choice between accurate and approximate versions of blocks of code using Green compiler. Hoffmann *et al.* [21] have proposed using energy-accuracy tradeoffs in context of power capping by translating static parameters of an application into dynamic knobs such as convergence for drop in accuracy. However, other APPXs at algorithmic level, such as logic simplification, cannot be translated into dynamic knobs. Escaping infinite loops and skipping iterations of bottleneck loops that consume longer execution time were proposed by Sidiroglou-Douskos *et al.* [22] as loop perforation. All these techniques explore ways to compute approximately for energy and performance gains, within acceptable quality. However, they do not use APPX for closed-loop power actuation.

### C. IoT

In the context of IoT applications, the need for computational capacity at a sensor node level would not suffice. To meet real-time performance requirements, data collected over sensory nodes are processed at a smart gateway [10]. The gateway acts as an intermediate edge layer between the sensor front end and the cloud server back end [23]. Typical gateway can be a multicore platform, which still faces with power and energy consumption challenges [24]. Some of the IoT applications are concerned with actuation mechanism based on sensory data analysis, such as identifying sudden changes in input data. Such applications present with an opportunity to relax the accuracy of computation, which in turn can be used to conserve energy and accelerate the performance.

## III. Approximation Knob

Approximate execution of an application provides variable performance and energy gains with the amount of error induced. We present the Pareto space of accuracy-performance tradeoffs with two example applications viz., sparse matrix multiplication and $k$-means clustering. We considered two $10\,000 \times 10\,000$ sparse matrices that are multiplied approximately by skipping inner most loop that performs multiplied accumulation. Fig. 3(a) shows normalized performance and energy gains with the number of inner most loops that are skipped (workload reduced) to reduce the number of computations. With 50% of workload reduced, performance and energy efficiency doubles. For $k$-means clustering, we use 10 000 random input data points to be classified into 50 clusters. We used relaxed convergence as the APPX, by early termination of the clustering algorithm with nonzero flips. Fig. 3(b) shows the gain in performance and energy for error induced by

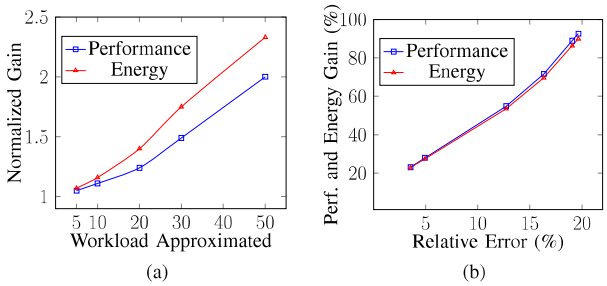



Fig. 3. Performance and energy gains with APPX. (a) Workload-performance tradeoffs for matrix multiplication. (b) Accuracy-performance tradeoffs for $k$-means clustering.

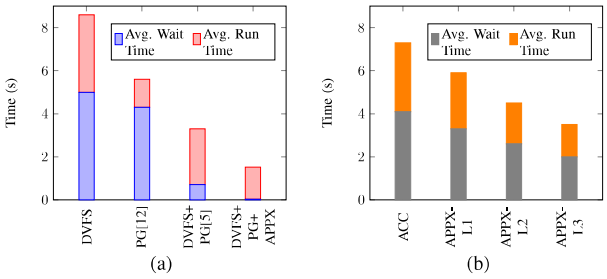



Fig. 4. Performance gains of different knobs for k-means clustering simulated on 16-core system. (a) Application service time for different knobs. (b) Application service time for different levels of APPX knob.

relaxing the convergence from 5%–10%. Relative performance gains increase as the error induced increases. Both these examples establish performance and energy gains with APPX. Specifically, they provide an insight on performance gains per error induced, which can be exploited while implementing the APPX knob.

We demonstrate the impact of different power knobs on the performance of many-core systems with applications dynamically entering and leaving the system using $k$-means clustering as an example. The performance of the system is determined by service time of an application, which is the sum of wait time, the time elapsed between application request and starting of the execution, and runtime, the time consumed in executing the application on chip [8]. Dynamic workload characteristics contribute to power violations, forcing actuation of power knobs. We simulate the application for four different power knobs viz., DVFS, PG, DVFS + PG (referred as MOC) [5], and DVFS + PCPG + APPX, using the experimental platform, detailed in Section VII. The APPX knob has $k$ levels of APPX, where $k$ is a parametrizable entity. In this case, we chose four levels of APPX.

The average service time for different knob combinations is shown in Fig. 4(a). In case of using DVFS and PG knobs [12], per-application runtime increases forcing incoming applications to wait longer, resulting in high service time. The combination of DVFS and PG has relatively better service time using the power management algorithm, as in [5] and [17]. With the APPX knob in combination with DVFS and PG, the service time is the lowest, indicating high performance and energy gain within the given power budget. The APPX knob

loads applications with relaxed accuracy that have lower workloads and thus low runtime. Consequently, more resources are available for incoming applications, improving the wait time and the overall service time. Fig. 4(b) shows application service time of APPX knob over different levels of APPX. The gain in performance with increasing level of APPX is trivial. Despite effective power capping and possibility of increasing the number of simultaneously active cores, performance still suffers with DVFS and PG when compared with that of APPX. Hence, we propose a hierarchical management for effective combination of these knobs to complement each other.

## IV. Accuracy-Performance Tradeoffs: A Case for IoT

IoT applications deal with real world sensor data that are analog and involve noisy components. Performance requirements for IoT systems are usually high while energy budget is limited [1]. Collection and classification of raw data into meaningful clusters, filtering, computation for actuation decisions, and communicating external world are major functionalities of an IoT system. Every stage in this process has a variable tolerance to inaccurate computations, presenting opportunities for APPX. Leveraging this fine-grained error resilience, APPX can provide better performance-per-energy in IoT systems. We explore the possibilities of accuracy-performance tradeoffs in IoT domain using two case studies on health monitoring applications viz., EWS system and fall detection.

### A. EWS System

An EWS system is used in health care for monitoring vital signs of a patient to proactively alert medical support. A method for EWS is proposed in [25] that uses three types of sensors—medical, environmental, and activity. Data collected from the sensors are preprocessed by using a Butterworth filter to remove noise components and false alarms. Sensory data from different sources are fused to extract useful details. Every physiological data sensed are allocated a score based on the range the sample belongs to and its implication on patient's health deterioration. The final score is calculated as a combination of scores of all the individual sensor nodes' data. When the final EWS exceeds a fixed threshold, a warning signal is transmitted seeking for medical attention. This system deals with heterogeneous data generated by different sensors and involves computations on unclassified, redundant, and noisy data. To extract meaningful insights, the EWS uses data acquisition, filtering, fusion, classification, and analysis, followed by computation and transmission. Although this system is mission critical, there are several intermediary stages where inaccurate computations can be tolerated. For example, heart rate measured is classified into one of the several ranges of heart rate data and a score is assigned according to the range it belongs to, but the exact value of heart rate is not used. Medical sensors trace several samples of data per second on an average, while a median of these data is good enough to represent all the samples. Relaxing such computations and data points that do not affect final EWS can enhance performance of the system within a lower energy budget.

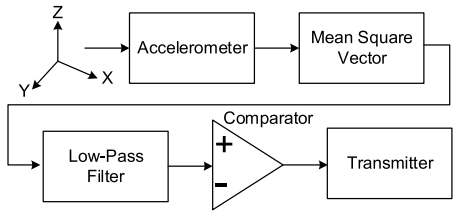



Fig. 5. Fall detection.

### B. Fall Detection

Another example of IoT application that is data and compute intensive is fall detection [26]. Fall detection mechanism identifies whether a person using the wearable detector falls hazardously on the ground. Specifically, fall detection is used in the context of patient and elderly people monitoring, to bring attention and support upon a fall. Typical fall detection employs camera and gyroscopic or accelerometer sensors for identifying a fall with respect to inertial position. We use fall detection based on a accelerometer, as proposed in [26]. The accelerometer data in three dimensions are used to calculate signal magnitude vector as the square root of the sum of the squares of signal component in each axis. This is fed to a low pass filter to generate discrete signal of positioning. Unusual spikes in the filtered data when compared with a fixed threshold represent the possibility of a fall, which is then transmitted to support system infrastructure for further assistance. The fall detection mechanism is shown in Fig. 5. A major conundrum in fall detection is in identifying the abnormal spikes in positioning signal—whether to analyze the accelerometer data tightly coupled to the sensor or to transmit the data to a cloud computer. Analyzing the data in a simpler microcontroller has performance penalties while transmitting filtered data to a high-end cloud computer consumes more energy. Expensive floating point computations on high sampled accelerator data, such as multiplications, square root, and filtering, can be relaxed to gain performance. We have run the fall detector mechanism over three sample persons for 8 h of a day. Sensory data are collected at 100 samples/s and fall detection is executed at a gateway between sensor nodes and cloud server. These test cases show that accelerometer signals generate data that are usually redundant, indicating that skipping some of the sensory data samples would induce only a tolerable error. Reducing the sampling of sensor and filter length by half produced results that are similar to accurate computations. Relaxing accuracy of data analysis can thus improve fall detection's performance.

## V. Knob Actuation Scenarios

We primarily monitor power consumption, workload intensity, and sensitivity of applications to make knob actuation decisions. The threshold for power consumption is TDP. Power consumption exceeding TDP indicates a power violation. Furthermore, we also set another parameterizable threshold $TDP_{th}$, a metric that indicates possibility of a potential TDP violation, such that $0.66 \times \text{TDP} < \text{TDP}_{th} < \text{TDP}$. We use accumulated wait time (AWT) of application requests

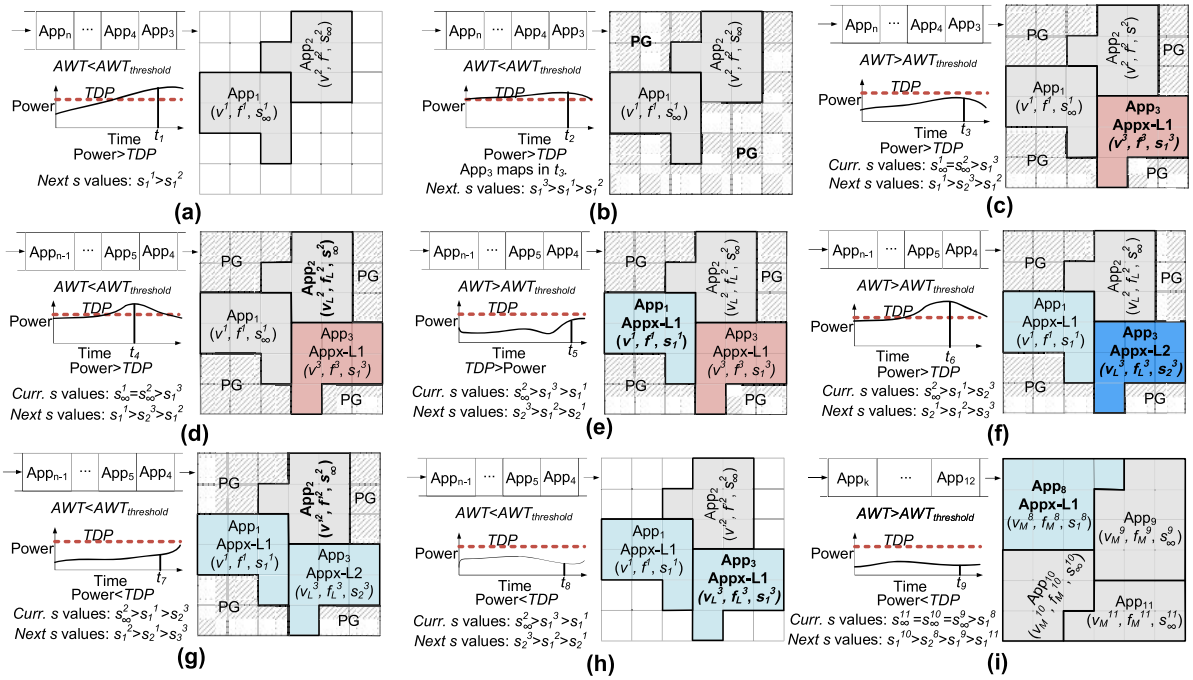



Fig. 6. Knob actuation scenarios. Each scenario shows applications running on a 36-core system along with power consumption and workload intensities.

made for monitoring the workload. For a set of applications $\text{App}_1, \text{App}_2, \ldots, \text{App}_N$ with wait times $w_1, w_2, \ldots, w_N$, AWT is given as

$$\text{AWT} = \left(\sum_{i=1}^{N} w_i\right)/N. \tag{1}$$

Longer application wait time indicates the higher workload intensity. A parametrized metric $\text{AWT}_{\text{th}}$ is set as threshold for workload intensity. The power objective is to restrict the power consumption to TDP and throughput objective is to restrict the AWT to $\text{AWT}_{\text{th}}$. For the actuation of APPX knob, we use sensitivity metric of an application to choose an application and its corresponding level of APPX.

### A. Application Sensitivity

The performance and energy gains varies for different applications over different levels of APPX. This presents a case where approximating one application might yield more performance gain than that of the others. We identify each application with a sensitivity metric as performance that could be gained per error induced. The motivation behind this is to choose an application that results in higher performance gain for the amount of error induced.

We define an application's sensitivity metric as

$$\text{Sensitivity} = \frac{\text{Perf}_i - \text{Perf}_{i-1}}{\text{Error}_i - \text{Error}_{i-1}} \tag{2}$$

where $\text{Perf}_i$ and $\text{Error}_i$ represent performance and error induced at $i$th level of APPX. Sensitivity of an application for any two given levels of accuracy would be high when the performance gained by lowering accuracy is high or when the accuracy loss in performance improvement is lower. Subject to application characteristics and input data, sensitivity of an application varies through different levels of APPX. Sensitivity metric of an application presents a wider Pareto-space of accuracy-performance tradeoffs that can be explored in choosing an application to be approximated and the level of APPX. Sensitivity metric identifies tasks that will result in the highest performance gain among the set of tasks currently running, and prioritizes these tasks as candidates for switching their mode of execution to approximate. This enables fine-grained control on APPX knob, appropriate choices for mode switching, and performance and energy gain at lower relative error, and limits the possibility of overcompensation with APPX. Details on sensitivity metric used for evaluation purpose are detailed in Section VII. We demonstrate possible scenarios that require knob(s) actuation under diverse power consumption and workload intensities. Fig. 6 summarizes these scenarios, representing power consumption, workload intensity, and knob actuations employed over a span of execution. Each scenario (a)–(i) shows power consumption with respect to TDP, applications waiting in the queue, applications that are mapped on the chip with their respective voltage and frequency levels, and application's sensitivity. Voltage and frequency levels of each mapped application $(\text{App}_1, \text{App}_2, \ldots, \text{App}_n)$ are represented as $[(v^1, f^1, s^1), (v^2, f^2, s^2), \ldots, (v^n, f^n, s^n)]$. The lowest and highest levels of voltage and frequency are $(v_L, f_L)$ and $(v_M, f_M)$, respectively. The corresponding lowest and highest levels of sensitivities are represented as $s^1_\infty$ (lowest level is $s_\infty$ as error is 0) and $s^1_m$. The sensitivity for each application at different levels of APPX is shown as $s^n_i$, where $n$ is the application number and $i$ is the level of APPX. Criteria for knob actuation are TDP violation (power > TDP) and high request rate of incoming applications ($AWT > \text{AWT}_{\text{th}}$).

1) *Scenario (a):* Two applications $\text{App}_1$ $(v^1, f^1, {}^1_\infty)$ and $\text{App}_2$ $(v^2, f^2, s^2_\infty)$ are currently running in the

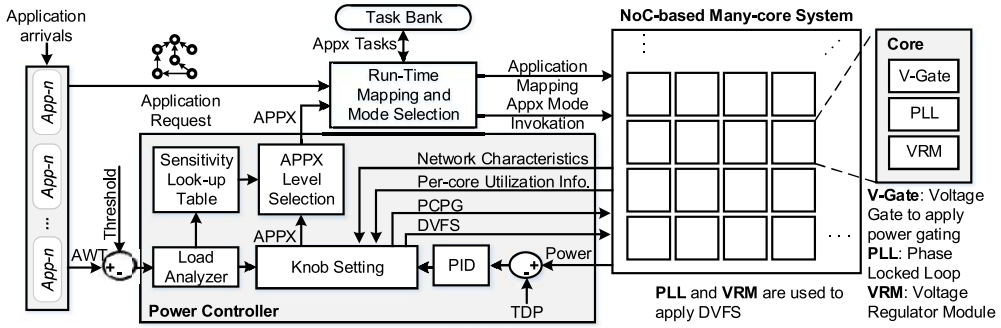



Fig. 7. Power management framework.

accurate mode of execution. At time $t_1$, a power violation (power > TDP) occurs, while the throughput is under control (AWT < $\text{AWT}_{\text{th}}$). DVFS knob is triggered over $\text{App}_1$ and $\text{App}_2$ to lower the power consumption within TDP.

2) *Scenario (b):* $\text{App}_1$ and $\text{App}_2$ are now running at downscaled voltage and frequency levels ($v^1$, $f^1$, $s^1_\infty$) and ($v^2_L$,$f^2_L$, $s^2_\infty$) with DVFS invocation at $t_1$. Although power consumption is relatively lower than at $t_1$, power violation still persists. So, PG knob is invoked to power gate the remaining unoccupied cores (power gated cores are shaded). This would potentially violate throughput constraint, since subsequently arriving applications do not have any free cores to be mapped onto.
3) *Scenario (c):* Power consumption is below TDP and there is no power violation. Application request rate has increased and there is a throughput violation (AWT > $\text{AWT}_{\text{th}}$), due to power knobs triggered previously. APPX knob is invoked to counter throughput violation. $\text{App}_3$, arrived at $t_3$, has a higher sensitivity than $\text{App}_1$ and $\text{App}_2$, and hence is mapped in its approximate version [$\text{App}_3(v^3, f^3, s^3_1)$]. Intuitively, we are reducing the execution time of $\text{App}_3$ so that there are free cores for potentially incoming applications.
4) *Scenario (d):* At time $t_4$, the power consumption approaches $\text{TDP}_{\text{th}}$, indicating the possibility of power violation. DVFS knob is triggered to avoid potential power violation over $\text{App}_2$, which is chosen as candidate for voltage downscaling based on its network and compute characteristics [$\text{App}_2(v^2_L, f^2_L, s^2_\infty)$].
5) *Scenario (e):* At time $t_4$, power is under control, while throughput violation persists. $\text{App}_3$ is running in approximate mode at level-1. APPX knob is invoked again to address throughput violation, with a choice among switching $\text{App}_2$ to its next level, i.e., level-2 of APPX, or $\text{App}_1$ to level-1 of APPX. The sensitivity metric for $\text{App}_1$ at level-1 $s^1_1$ is higher than the other two applications at level-2 ($s^2_2$, $s^3_2$), hence $\text{App}_1$ is chosen to switch to level-1 of APPX [$\text{App}_1(v^1_L, f^1_L, s^1_1)$].
6) *Scenario (f):* At time $t_5$, power is under control, while request rate is still high. All the approximable applications are running in approximate mode at level-1. APPX knob is invoked again, with a choice among the applications for maximum performance gain. The sensitivity metric for $\text{App}_3$ at level-2 of APPX is higher ($s^3_2 > s^2_2 > s^1_2$ ), and is thus chosen to switch the level of APPX further to level-2 (shown in bold) [$\text{App}_3(v^3_L, f^3_L, s^3_2)$].
7) *Scenario (g):* At time $t_5$, power consumption is below TDP, so the DVFS knob is invoked to upscale the voltage and frequencies of some cores. Among the three applications, $\text{App}_2$ is chosen for upscaling, as it benefits the most based on its network and compute characteristics.
8) *Scenario (h):* At time $t_8$, power consumption is below TDP, and the application request rate is also below its threshold. DVFS knob is invoked to upscale voltage and frequencies of some more cores. Since $\text{App}_2$ is previously upscaled, $\text{App}_1$ is now chosen as the candidate that benefits from upscaling. Since the throughput constraint is maintained, APPX knob is invoked. $\text{App}_3$, which has a higher sensitivity, is chosen to switch a level up in accuracy, going into level-1 of APPX from level-2 [$\text{App}_3(v^3_L, f^3_L, s^3_1)$]. This invocation can be influenced by a user-defined parameter to upscale voltage instead of switching up the level of APPX.
9) *Scenario (i):* At time $t_9$, power consumption is well below TDP, allowing more power to be consumed safely. DVFS knob is invoked to upscale the voltage and frequencies of all active cores to their maximum values. The application request is still higher than threshold, despite voltage upscaling and hence APPX knob is invoked. $\text{App}_8$ has the highest sensitivity among running applications, hence it is chosen to switch mode of execution to approximate at level-1 [$\text{App}_8(v^8_M, f^8_M, s^8_1)$].

## VI. System Design

We design our power management framework for NoC-based many-core systems supporting dynamic arrival of applications. Our power management framework monitors per-core power consumption and utilization, network intensity, incoming application request rate, and sensitivities of applications to error to actuate different knobs accordingly. The top-level view of our system architecture is shown in Fig. 7.

### A. Application Modeling

We model individual computational blocks of an application as a task. Each task is identified by its compute intensity,

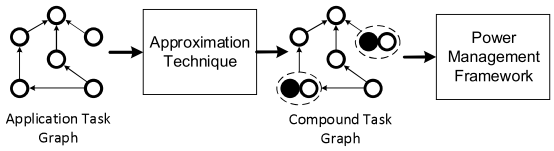



Fig. 8. Compound task graph—workflow.

communication volume with other tasks, and power consumption. Applications are modeled as directed graphs, with each node representing a task running concurrently with other tasks. Analysis of applications' power-performance characteristics and task graph formation is implemented as an off-line function. Incoming applications are classified as approximable and nonapproximable. Approximable applications have one or more tasks that can be replaced by their approximate versions. Such applications are modeled as compound task graphs, as shown in Fig. 8. We generate compound task graphs that include multiple versions of approximable tasks (in dotted lines), and the approximate tasks are shown in solid fill.

### B. System Architecture

*1) Runtime Mapping and Mode Selection:* Incoming applications are queued in the application repository and make a request to be executed on the chip. The run-time mapping and mode selection unit (RMSU) responds to the application request by selecting free cores and maps the application in a task-per-core manner. In addition, if the application is approximable, RMSU buffers the approximate tasks from the compound task graph into the task bank. The task bank is implemented as a memory structure that holds pointers to the addresses of approximable tasks, to provide easier access to approximable tasks without a significant overhead. We use proactive application mapping *MapPro*, presented in [27]. Upon application mapping, RMSU sends the core allocation information to the power controller for potential actuation decisions. Servicing an application depends on the availability of free cores on the chip, which in turn depends on performance and power consumption of active cores. The number of outstanding application requests weighed with the time before they get serviced, and AWT (1) is sent to the power controller for knob actuation decisions.

*2) Power Controller:* Power controller is the central manager that monitors power consumption, incoming application request rate, and system metrics for power and performance knob actuation decisions. Every core on the chip is provided with power and processor utilization sensors. We use the combination of processor utilization, packet injection rate, and buffer utilization to prune the design space for the selection of candidates that are more suitable for voltage down/up scaling. Selection of appropriate candidates for employing the DVFS and PG knobs is elaborated in our previous work [5], [17].

Thus, we monitor power consumption, processor utilization, network congestion, and network intensity at runtime, forming the monitor phase of the power management framework. Actuation decisions of the power controller are based on parameters received from the monitor phase. We feed the difference between power consumption of the chip and TDP to a PID controller. Output of the PID controller is proportional to the difference between power consumption and TDP and determines voltage and frequency levels to be downscaled to avoid power violation. In case of power consumption being below TDP, voltage and frequency would be upscaled for better power utilization. Knob Setting block of the power controller receives the new voltage and frequency levels from the PID controller, along with processor utilization, buffer utilization, and packet injection rate from the monitor phase. Based on utilization and network parameters, knob setting block decides the cores to which voltage and frequency levels are to be updated. The PID controller's output is also used to decide the number of cores to be power gated. Both DVFS and PG actuations are applied to the chip, as shown in Fig. 7.

Load analyser compares application request rate, represented by AWT, and the threshold, $AWT_{th}$, to determine throughput violation ($AWT > AWT_{th}$). The *Knob Setting* uses this information and invokes APPX knob. Sensitivity metric over different levels of APPX for all the applications is summarized into a lookup table. Each application has $k$ (parametrized) levels of APPX and sensitivities associated with each level. The level of APPX of an application that is currently running on the chip determines current sensitivity factor, while the ones that are preceding and succeeding are the previous and next sensitivity factors. The lookup table is pruned to find the application that has the highest sensitivity factor in its next level of APPX. For example, consider $App_1$ running at level-1 of APPX and $app_2$ running at level-2 of APPX. If APPX is to be invoked, the next level of APPX for $app_1$ is level-2 and for $app_2$ is level-3. So, the sensitivities of $app_1$ at level-2 (succeeding the current level-1) and $app_2$ at level-3 (succeeding the current level-2) are compared to find the application with highest next sensitivity value. The chosen application and corresponding level of APPX are forwarded to the RMSU. The RMSU retrieves the approximable tasks of the chosen application with the level specified by the APPX Level Selection from the task bank. The accurate task is then replaced with the approximate task retrieved from the task bank. For evaluation, we currently use four levels of APPX in increasing order of accuracy-performance tradeoffs. Alternatively, several fine-grained levels of accuracy tradeoffs could be used.

*Mode switching:* APPX knob invocation triggers switching the mode of execution of an approximable application. Depending on APPX knob setting, RMSU chooses the version of task to be included in the application mapping, while the other versions are buffered. With the invocation of APPX knob, there are two possible scenarios for mode switching: 1) mapping approximate task graphs and 2) switching mode of execution of applications currently running by task replacement. In the former case, the RMSU maps every incoming application in its approximate version by including the approximable tasks instead of accurate tasks, until the mode is switched back to accurate. The level of APPX is specified by the power manager. For applications that are currently running on the chip, power manager chooses the application(s) and the level of APPX to switch to. Based on these, RMSU identifies the corresponding approximate task specified by the power

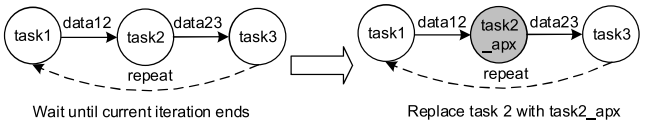



Fig. 9. Mode switching.

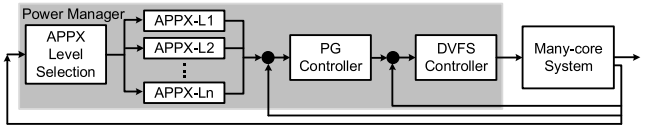



Fig. 10. Hierarchy of knobs.

manager from the task bank and replaces the accurate task with the approximate task. We modeled applications for evaluation as data-dependent concurrent tasks that execute periodically. The computational process repeats until the end of execution with a specified periodicity. Streaming and signal processing applications are good examples, which execute periodically over incoming samples of data, where it is possible to relax certain aspects of computation when new data arrive every period. In a similar manner, IoT applications work on real world sensory data, preprocessing, filtering, and computation over continuous intervals in a batch. Precisely, in these applications, the computational task remains the same while new data arrive after every interval. When the RMSU has to replace an accurate task with approximate, it lets the current iteration of accurate task's computation to finish execution. It waits until data from the accurate task are received at its destination end task (if any). Once the data transfer is completed, the RMSU loads the approximate task on to the chip, replacing the accurate task. Fig. 9 shows the process of task replacement during mode switching. The example has three tasks 1–3 out of which task2 is approximable. On invocation of APPX knob, the switching happens in the following sequence: 1) the RMSU finds the approximate task $task2\_appx$ from the task bank; 2) it waits until data from task2 (data23) is received at task3; 3) $task2\_appx$ is loaded by fetching the instruction stream into the cache (I-cache); and 4) after the data is received at task3, the execution of task2 will now start from new instruction stream of $task2\_appx$. Depending on the size of instruction cache used, instructions of task2 may require flushing, however, this is subject to hardware platform. Since the computational process of the application is periodic in nature, data are not changed with mode switching, and moving the data or flushing the data cache (D-cache) is not needed. The state of the application is hence preserved at the end of the period. The mode switching overhead is elaborated in Section VII.

### C. Power Management Algorithm

We employ the DVFS and PG knobs synergistically with APPX in a hierarchical way, as shown in Fig. 10. Triggering and tuning of these knobs together for power capping and performance maintenance is handled by power management algorithm, as listed in Algorithm 1. We define three epochs $e_1$, $e_2$, and $e_3$ for actuation of APPX, PG, and DVFS knobs, respectively, such that $e_1 > e_2 > e_3$. At every epoch $e_1$, application request rate is monitored. The difference between AWT and its threshold is used to choose settings for APPX knob. Settings for the APPX knob viz., application and level of APPX, are determined by level selection algorithm, as listed in Algorithm 2. The extent of throughput violation ($\Delta T$) determines the mode of incoming applications. If ($\Delta T > 1$), it reflects a steeper request rate of applications. Then, the mode is set to in, indicating to the RMSU that newly incoming applications are to be mapped in their approximate mode at level-1. If ($1 > \Delta T > 0$), the mode is set to run, i.e., to switch the execution mode of currently running applications only. In this case, based on sensitivity factors of currently

**Algorithm 1** Power Management Algorithm

**Inputs:** $P$: Power Consumption, $AWT$: Accumulated Wait Time;
**Outputs:** $APPX$, $PCPG$, $DVFS$, $mode$: Knob actuations and Mode switching commands for RMSU;
**Constants:** $TDP$: Power budget, $AWT_{th}$: *AWT* Threshold, $e1$, $e2$, $e3$: Knob actuation epochs, $SFLT$: Sensitivity factor look-up table
**Global Variables:** $currSF[\,]$, $nextSF[\,]$, $prevSF[\,]$ : Sensitivity factor vectors of current, next and previous levels of approximation of running applications; $cores_{un}$, $cores_{gate}$, $cores_{pref}$ : Cores - Unoccupied, power gated and preferable for DVFS.
**Variables:** $App_R$, $App_X$, $App_Y$: Applications - currently running, approximable, running in approximate mode; $SF$: Sensitivity factor;
**Body:**

```
1: for epoch = e1 do
2:    ΔT = AWT − AWT_th;
3:    (app, level, mode) = appxChoose(ΔT, currSF);
4:    APPX(app, level, mode);
5:    for epoch = e2 do
6:       ΔP = P − TDP;
7:       if ΔP ≥ 0 then
8:          PCPG_gate(cores_un);
9:       else
10:         PCPG_un(cores_gate);
11:      for epoch = e3 do
12:         ΔP = TDP − P;
13:         if ΔP ≥ 0 then
14:            DVFS_down(cores_pref);
15:         else
16:            DVFS_up(cores_pref);
```

**Algorithm 2** APPX Level Calculation Function [$appxChoose()$]

**Inputs:** $\Delta T$, $currSF(SF[\,])$;
**Outputs:** $app$: Application chosen for approximation, $level$: level of approximation, $mode$: Mode for incoming applications;
**Body:**

```
1: if ΔT ≥ 0 then
2:    if ΔT ≥ 1 then
3:       mode = in;
4:    else
5:       mode = run;
6:    SV[ ] ← nextSF(SFLT[ ]);
7:    sort(SV[ ]);
8:    (app, level) = max(SV[ ]);
9: else
10:   SV[ ] ← prevSF(SFLT[ ]);
11:   sort(SV[ ]);
12:   (app, level) = min(SV[ ]);
13: return (app, level, mode);
```

TABLE I
APPLICATIONS' ENERGY-ACCURACY TRADEOFFS

| Application | | Norm. Perf. | | | | Norm. Energy | | | | %Error | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Level-1 | Level-2 | Level-3 | Level-4 | Level-1 | Level-2 | Level-3 | Level-4 | Level-1 | Level-2 | Level-3 | Level-4 |
| Machine Learning | Linear Regression | 1.01 | 1.09 | 1.15 | 1.20 | 1 | 1.08 | 1.15 | 1.2 | 0.01 | 0.05 | 0.08 | 0.1 |
| | K-means | 1.23 | 1.54 | 1.71 | 1.92 | 1.22 | 1.53 | 1.69 | 1.89 | 3.53 | 12.74 | 16.34 | 19.66 |
| | K-NN | 1.08 | 1.1 | 1.15 | 1.57 | 1.11 | 1.13 | 1.18 | 1.75 | 1.13 | 2.1 | 5.85 | 28.8 |
| Signal Processing | FFT | 1.01 | 1.01 | 1.05 | 1.07 | 1.04 | 1.06 | 1.08 | 1.11 | 2.17 | 4.14 | 7.6 | 13.34 |
| | LPF | 1.1 | 2.0 | 3.34 | 10.5 | 1.2 | 2.0 | 3.5 | 9.33 | 15.72 | 16.6 | 19.35 | 30.09 |
| | Vector Multiplication | 1.07 | 1.18 | 1.25 | 1.68 | 1.06 | 1.15 | 1.2 | 1.6 | 8.53 | 12.39 | 18.2 | 30.0 |
| IoT | EWS | 1.11 | 1.15 | 1.33 | 1.49 | 1.22 | 1.29 | 1.37 | 1.57 | 10.4 | 14.5 | 17.49 | 27.72 |
| | Fall Detection | 1.1 | 1.39 | 1.7 | 2.06 | 1.1 | 1.4 | 1.75 | 2.33 | 9.18 | 11.38 | 14.1 | 22.9 |

**Algorithm 3** Mode Switching Algorithm

**Inputs:** $app$: Application chosen for mode switching, $level$: Level of approximation, $mode$ : Mode of mapping a new application
**Outputs:** $Map$: Mapping configuration of incoming application;
**Variable:** $newApp$: Incoming application, $t^{level}$: Approximable task and its level of approximation;
**Body:**

```
if mode = in then
    if newApp then
        for t^a ∈ newApp do
            TaskBank.push(t^a);
            switch(t^1, t);
            Map(newApp);
if mode = run then
    wait until current iteration of t finishes;
    switch(TaskBank→ app.t^level, app.t);
```

running applications, the sensitivity factor vector for the next level of APPX is looked up from the sensitivity factor lookup table (SFLT). This vector is sorted to find the application with highest sensitivity among running applications. The chosen application and level of APPX are returned to the power manager for invocation of APPX knob. In case the throughput is not violated ($\Delta T < 0$), the APPX level of a chosen application is shifted up, to a relatively accurate level. The sensitivity factors of preceding level of APPX of currently running applications are looked up from SFLT. These are sorted to find the application that is least sensitive to error. This application is chosen for the invocation of APPX knob, and the application and its level of APPX are returned to the power manager. APPX knob is invoked by the power manager by sending the knob settings received from the level selection algorithm to the RMSU. This is presented in a listing in Algorithm 3. When the mode is in, new incoming application's tasks are buffered into the task bank. The new application is mapped in its approximate mode at level-1. When the mode is set to run, the task and its level of APPX specified by the power manager are retrieved from the task bank. Accurate version of this task is replaced by the approximate task. Epochs $e_2$ and $e_3$ are smaller than $e_1$ and are concerning power violations. Power violations are monitored at epoch $e_2$. If power consumption exceeds TDP, PCPG knob is actuated by PG cores that are currently unoccupied on the chip. Conversely, when power consumption is below TDP, cores that are previously power gated are powered up. At epoch $e_3$, power violations are addressed by DVFS knob. Cores that benefit relatively higher from DVFS actuation are the preferable cores. DVFS knob is actuated over these preferable cores. Similar to the PCPG knob, voltage and frequency levels of preferable cores are upscaled when power consumption is below TDP. The actuation of PCPG and DVFS knobs is based on our prior work on multiobjective power management framework [5], [17].

## VII. EVALUATION

In this section, we assess the efficiency of our APPX-enabled power management approaches APPEND+ and APPEND, with and without considering sensitivity of application. We compare our approach against the state-of-the-art dynamic power management/capping techniques PG [12] which is based on PCPG, and MOC [5] which is based on per-core DVFS and PCPG.

### A. Application Setup

For evaluation purpose, we choose interdisciplinary error-resilient application domains of machine learning and signal processing. Furthermore, we selected two applications from IoT domain, given the nature of input data and computations involved. The applications used for the evaluation of APPEND+ are presented in Table I. The chosen machine learning applications are data-triggered on-line learning techniques that fall under classification and estimation. They are interdisciplinary, specifically with IoT-based applications, being used in recognition, mining, synthesis, and automation that are performance and energy demanding. These workloads are based on the iterative methods of computation, meaning that the accuracy of result converges toward an optimal solution with a more number of iterations. Since an accurate solution may not exist and lower convergence could still offer an acceptable result, they become candidates for APPX. For evaluation purpose, we choose four levels of APPX, level-1 through level-4. We normalize the performance and energy gains of approximate tasks from level-1 to level-4 against their accurate versions. Table I shows the normalized gain in performance and energy, and relative error induced with APPX for different applications and levels of APPX. The applications tested are error resilient in general. We chose APPXs that result in soft errors, ensuring there are no critical errors or exceptions. For linear regression, we use training data of 1 million data samples and a test it over 1000 samples.

TABLE II
APPLICATION SENSITIVITY

| Application | | Sensitivity | | | |
|---|---|---|---|---|---|
| | | Level-1 | Level-2 | Level-3 | Level-4 |
| Machine Learning | Linear Regression | 10.1 | 0.09 | 0.13 | 0.23 |
| | K-means | 6.51 | 0.28 | 0.44 | 0.64 |
| | K-NN | 0.07 | 0.06 | 0.06 | 0.43 |
| Signal Processing | FFT | 0.01 | 0.01 | 0.05 | 0.07 |
| | LPF | 0.6 | 5.5 | 2.9 | 1.84 |
| | Vector Multiplication | 0.24 | 1.59 | 0.31 | 0.83 |
| IoT | EWS | 0.95 | 0.26 | 0.58 | 0.3 |
| | Fall Detection | 1.02 | 0.86 | 0.72 | 0.14 |

We use loop perforation to skip 5% to 15% of computations on input training data. The error is calculated as relative to accurate regression. For $k$-means clustering and $k$-nearest neighbors, we use relaxed convergence. We compromise on the number of flips, coverage of neighbors, and training data sets, respectively, for these applications. We set the limits of relaxation on convergence from 3% to 10% for four levels of APPX. For the fast Fourier transform, we approximate the computation involving exponential functions with relaxed memorization and storing the twiddle factors in lower precision. We compute the complex exponential for one iteration and reuse it for subsequent samples, despite the inputs to exponential function not being the same. For a low-pass filter, we use a Blackman window with 50 coefficients. We reduce the number of coefficients up to 5 for relaxed execution. We use sparse vector multiplication because of its broader usage and application in several other fields. For this application, we simplify the logic of product calculation that replaces accumulated multiplications of rows and columns with a single multiplication of means of a row and column. For IoT case study on EWS, we reduce the number of samples of heart rate sensor and compromise data fusion. We run the accurate and approximate versions on data sets collected from three different subjects. For fall detection, we reduce the sampling rate of an accelerometer and the number of filter coefficients, and simplify the logic of magnitude vector computation. We use real-time accelerator data collected from a subject wearing the fall detector, over different physical activities of walking, sprinting, and resting.

We used the normalized performance gain and relative error induced upon mode switching for each application over four levels of APPX. We calculated the error sensitivity as gain in performance per error, as described in (2). Table II shows the sensitivity metrics for the applications used over different levels of APPX. It should be noted that normalized gain and sensitivity with increasing level of APPX are distinct. The sensitivity metric represents the amount of performance gained per amount accuracy lost by moving a level of APPX further. For example, linear regression has sensitivity of 10.1 at level-1, while the sensitivity at level-2, 0.09, is much lower than the previous level. This intuitively means that changing the level of APPX for this application from level-1 to level-1 either has a lower performance gain or higher error penalty or both. Similarly, $k$-means at level-1 has much higher sensitivity than that of the other levels. The normalized gain at level-1 for $k$-means is 1.23, however, the error, 0.01, (see Table I) is extremely small, making the sensitivity high. APPEND+ considers the sensitivity metric of all the applications currently running on the chip to make decisions on which application and which level of APPX are to be chosen for APPX knob actuation. APPEND is oblivious to the sensitivity metric and chooses applications in a naive manner and corresponding level of APPX in a sequentially increasing order.

### B. Simulation Environment

Applications are modeled as task graphs, as described in Section VI. We implement each application such that one task is allocated one core on interval-core-based Sniper simulator, annexed with McPAT for modeling power [28]. We used Gainestown architecture that has Nehalem-like processing elements (PEs) with 32 kB of instruction and data caches. We model the application into concurrent tasks, preserving data flow nature. We use loop perforation and relaxed convergence in case of approximate tasks. We extract execution time, average power, and energy consumption per task. We normalize these values as compute factor metric for each node in the task graph, along with the amount of data flow as the communication volume between tasks. The task graph for each application is thus a directed network of nodes that holds execution time, communication volume, average static, and dynamic power consumption for accurate and approximate tasks. The performance and power values extracted from these simulations provide the relative performance and power gains of a task when the execution is switched to approximate from accurate. We use this ratio to model the power and throughput gains while simulating, so that the APPEND+ framework can be adaptive for all hardware platforms irrespective of architecture. The Sniper simulator provides PEs with other variants of microarchitecture. The relative performance gains for approximate versions over the accurate versions may hold good over different hardware platforms, unless the architecture is highly customized. We use our in-house cycle accurate simulator implemented in SystemC to evaluate the proposed power management framework. We extended Noxim [29] NoC simulator using its network infrastructure for interconnects. The power characteristics of PEs are modeled based on metrics extracted from McPAT and Lumos [30]. Lumos is an analytical framework that quantifies power-performance characteristics with technology node scaling for many-core systems. We used Lumos for physical scaling parameters, voltage scaling, and TDP metric for different network sizes. We added the support for dynamic arrival and servicing of applications through the run-time mapping unit. The mapping unit receives commands from power controller, implemented as a software module. The test bed is a rectangular network with X-Y routing. The $tile_{(0,0)}$ of the mesh acts as the central manager that is responsible for keeping track of mapping information. The network size is $12 \times 12$ and the chip area is 138 mm$^2$. For the first node selection in the run-time mapping process, we use MapPro [27] method. For the DVFS purpose, we use 15 VF levels with voltage in the range of 0.8–1.2 V. The frequency of the on-chip communication network (e.g., routers) is set to the maximum level (similar to [12] and [5]). The TDP value is

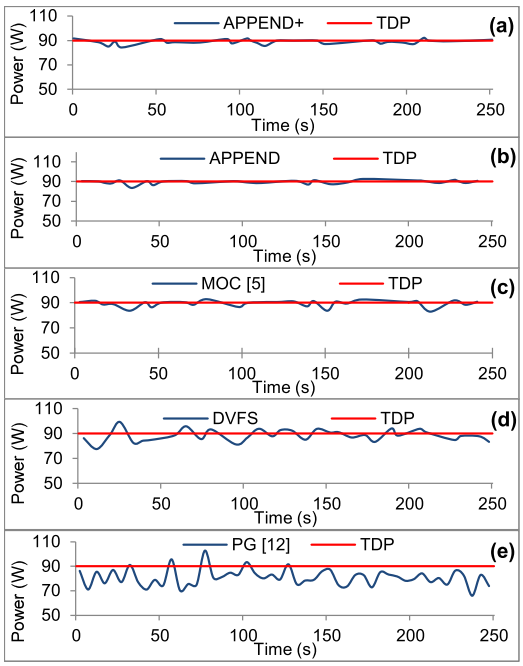



Fig. 11. Instantaneous power consumption of different knobs (a–e) on a 144-core system. (a) APPEND+, (b) APPEND, (c) MOC, and (d) DVFS, (e) PG. Power is capped at TDP.

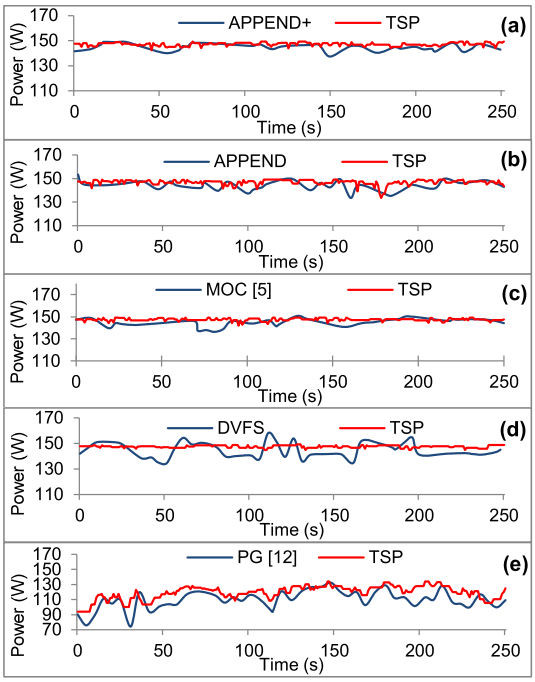



Fig. 12. Instantaneous power consumption of different knobs (a–e) on a 144-core system. (a) APPEND+, (b) APPEND, (c) MOC, (d) DVFS, and (e) PG. Power is capped at TSP calculated at run-time.

set to 90 W, calculated based on the chip's power density. We also evaluate our approach for power capping under a variable power budget, TSP [4]. TSP is calculated as a function of simultaneously active cores, which vary at runtime based on application arrival and mapping. TSP provides a relatively higher power budget than the conservative design time estimate of TDP. We estimate TSP online and use it as the upper bound on power budget for evaluating APPEND+. We implemented the APPEND+ technique over integrated simulation framework, as summarized earlier. It is possible to implement the same as an operating system level policy, provided the hardware platform supports sensing and actuation of power.

### C. Evaluation Metrics and Results

For evaluation purposes, we simulate the system over a period in which 200 applications are serviced. The evaluation metrics are as follows.

1) Power consumption: Power consumption of the system over the period of execution, honoring TDP by capping the power.
2) AWT: Accumulated value of wait time of applications before the application request is serviced.
3) Throughput: Time consumed to service 200 applications.

Our prerequisite goal is to cap the power consumption such that TDP constraint is honored throughout the period of execution. Fig. 11 shows the power consumption of DVFS, PG, MOC, APPEND, and APPEND+, along with TDP constraint, over the execution time for servicing 200 applications. TDP violation is more frequent with PG and DVFS knobs, while TDP is honored for most of the execution period with MOC, APPEND, and APPEND+. Fig. 12 shows the power consumption of the system with TSP as the upper bound on power.

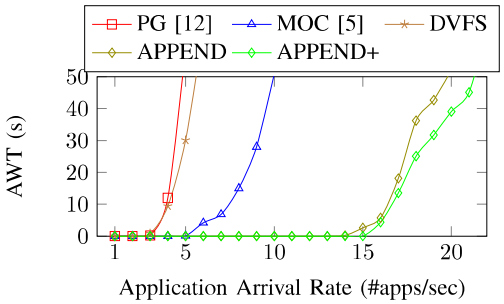



Fig. 13. Accumulated waiting time.

Unlike TDP, TSP varies at runtime offering a flexibility in power capping. DVFS knob violates the TSP limit and is not at efficient power capping. PG honors TSP constraint, however, the power consumption always remains lower than (but not closer to) TSP, indicating a lower utilization of available power budget. MOC, APPEND, and APPEND+ meet the power capping requirement and have a better power budget utilization. APPEND and APPEND+ maintain power consumption closest to TDP when compared with other knob combinations, reflecting better utilization of available power budget. This indicates mitigation of dark silicon and can be attributed to hierarchical usage of power knobs in APPEND+'s power controller. Moreover, we actuate power knobs, DVFS and PG, by monitoring power consumption over an epoch $e_1$ and trigger the APPX knob proactively over epoch $e_2$ with $e_2$ being five times longer than $e_1$. This eliminates possible random actuations or oscillations between different modes of execution. With better utilization, APPEND and APPEND+ are able to service applications faster, reducing the runtime

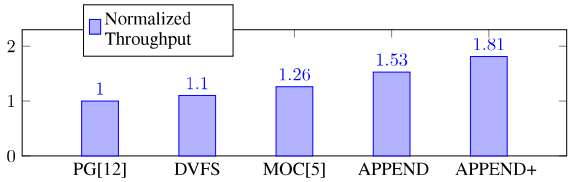



Fig. 14. Normalized throughput (TDP).

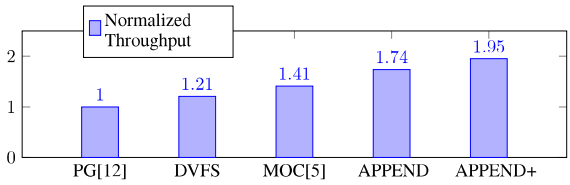



Fig. 15. Normalized throughput (TSP).

and consequently wait time of incoming applications. Fig. 13 shows the AWT for different power capping actuators over the period of execution. We present AWT as a function that is directly related to the rate of application requests made. Similar to power capping, APPEND+ has the best AWT, preceded by APPEND, MOC, PG, and DVFS. DVFS- and PG-based actuations have higher AWTs already when the application request rate reaches 3 per second. MOC has a relatively high AWT when application request rate is 5 per second. However, APPEND and APPEND+ have a near-zero AWT for as long as five times more than DVFS and PG and three times more than that of MOC. This demonstrates the ability of APPEND and APPEND+ to service applications faster despite high workloads, when compared with the other knobs. APPEND+ has AWT greater than zero when the application request rate reaches 15 per second. Also, AWT accumulation is more steeper in case of other knobs than that of APPEND and APPEND+, indicating a substantial rise in their wait times with high request rates. The minimal AWT and high service rates of APPEND and APPEND+ also results in high throughput and energy efficiency. Normalized gain in throughput for all knob combinations with TDP as upper bound and TSP as upper bound is shown in Figs. 14 and 15, respectively. APPEND+ followed by APPEND has the higher throughput, that is up to 1.9× better than PG and 1.4× better than MOC, showing a significant gain in performance and energy while power capping is strictly maintained. Employing APPX knob allows APPEND+ to minimize execution time of applications running on the chip. With applications leaving the system faster, more resources (cores) become available for incoming applications and reduce their wait time. APPEND benefits from AWT and throughput mutually improving each other. It is also to be noted that throughput gain of APPEND+ is relatively higher with TSP than that of TDP. The same trend can be seen with throughput of all the other knobs too, reflecting better utilization of power under TSP constraint. APPX knob can be implemented exclusively in software, making the APPEND+ framework scalable and adaptable across different hardware platforms. In the context of IoT applications, APPEND+ can be used both at the edge and cloud layers which can deliver real-time high performance at the front end. The sensitivity lookup table can be used only to store sensitivity metrics of applications that are currently running on the chip, limiting the size of the lookup table without affecting scalability.

TABLE III
APPLICATIONS—BEHAVIOR AND OVERHEAD

| App | Instr. | | Instr. Sim (M) | | L1-I Accesses (M) | | Overhead % (Norm.) |
|---|---|---|---|---|---|---|---|
| | Acc | Appx | Acc | Appx | Acc | Appx | |
| Linear Regression | 1105 | 1150 | 93 | 75 | 9 | 7 | 1.2 |
| K-Means | 1018 | 1021 | 449 | 102 | 57 | 14 | 0.3 |
| K-NN | 1457 | 1513 | 140 | 105 | 37 | 24.7 | 0.3 |
| FFT | 1147 | 1159 | 251.4 | 249 | 37 | 36 | 0.8 |
| LPF | 721 | 721 | 197 | 18 | 19.9 | 19 | 0.1 |
| Vector Multiplication | 775 | 779 | 33.2 | 22.7 | 20.3 | 19.8 | 0.5 |
| EWS | 1017 | 1032 | 66.1 | 4.1 | 4.3 | 4.1 | 0.3 |
| Fall Detection | 795 | 801 | 41.9 | 20.05 | 3.4 | 2.23 | 0.1 |

*1) Error and Overhead Analysis:* Behavioral patterns of accurate (Acc) and approximate (Appx) versions of each application are shown in Table III. The approximate versions are at level-4 of APPX, to reflect the maximum overhead caused and maximum performance gained. The number of instructions of each application (Instructions), the number of instructions simulated [Instr. Sim (M) in million], the number of L1-instruction cache accesses [L1-I Accesses (M)] (in million), and normalized overhead (in %) are presented in Table III. These metrics are extracted from individual application simulations (accurate and approximate) on Sniper. A number of instructions are slightly higher for approximate tasks due to conditional branching involved. However, these instructions eventually result in reduced overall workload and hence improve performance. For each application, we used 1 million elements in training set in increasing steps of 100 000 data points per period. A number of simulated instructions depend on training and test data sets used, and are variable in case of different sizes of data used. With loop perforation and relaxed convergence, input data elements are skipped, resulting in fewer instructions required to be simulated. Switching execution from accurate to approximate version incurs some overhead due to monitoring and triggering the approximate version. For every approximate task, the switching of execution mode involves a conditional branching instruction(s). The overhead incurred during this transformation included in the approximate task's compute factor. For the applications we used, the normalized overhead penalty incurred in mode switching ranged between 0.3% up to 1.2%. This overhead is negligible when compared with the workload reduced by APPX and thus levies no significant performance penalty. In terms of power overhead, the task bank and sensitivity lookup tables used in APPEND+ framework are simple memory structures with fewer access during execution of an application. The power consumption of these components is insignificant compared with the total system power. Loading an approximate task involves moving new instruction stream to the instruction cache, with a possibility of increase in the number of DRAM accesses. However, this

depends on the number of application instructions and the size of L1-instruction cache. For instance, a larger application coupled with smaller L1-instruction cache presents a worst case scenario that would force the system to evict accurate task and fetch the approximate task from main memory. In our test bed, we used L1-I cache of 32 kB and all the applications have instructions up to as many as 1500. The worst case penalty in terms of communication for switching from accurate version of a task to the approximate version can be calculated as follows:

$$\text{penalty} = \frac{\text{size}_{\text{app}}}{\text{size}_{\text{pkt}}} \times (P_L + (n \times r_L) + \text{MC}_L + \text{DRAM}_L) \tag{3}$$

where $\text{size}_{\text{app}}$ and $\text{size}_{\text{pkt}}$ are the application and packet sizes, $P_L$ is the packetizing latency, $n$ is the number of hops from a core to nearest memory controller, $r_L$ is the router channel latency, and $\text{MC}_L$ and $\text{DRAM}_L$ are the access latencies of memory controller and off-chip memory. We demonstrate the theoretical worst case overhead penalty of mode switching for a video encoding application run on Intel SCC as an example [31]. The experimental many-core platform Intel SCC has the off-chip memory, core, and network frequencies of 400, 533, and 800 MHz, respectively. The worst case mode switching penalty on SCC for the video encoding application of size 6 kB using the formula in (3) is 1.5 ms. For the same application, penalty in task migration is 10.6 ms, seven times more than the mode switching overhead, to move both instructions of 6 kB and data of 16 kB. Task migration overhead can still be higher when more data are to be moved, while mode switching needs no movement of data. It should be noted that these values are subjective to the platform on which they are executed, while the relative difference in overheads between mode switching and task migration might hold good.

## VIII. Conclusion

In this paper, we proposed APPX as another knob for power management in many-core systems. We implemented APPX knob based on application's sensitivity to error such that performance gain is maximized within minimal error. We developed power managing schemes to combine DVFS and PG knobs with APPX knob to meet system requirements in power capping, performance, and energy efficiency. We presented a power management framework, APPEND+, that monitors chip's power and performance requirements at runtime and triggers different knob actuations accordingly. We evaluated APPEND+ against other state-of-the-art power management techniques, over machine learning, signal processing, and IoT applications. APPEND+ improves performance and energy efficiency with the APPX knob and sensitivity aware actuation of the APPX knob, while power capping is maintained with the combination of APPX, DVFS, and PG knobs.

## References


[1] W. Arden *et al.*, *More-Than-Moore White Paper Version 2*, 2010, p. 14.

[2] A. M. Rahmani *et al.*, *Dark Side of Silicon*. Springer, 2016.

[3] H. Esmaeilzadeh, E. Blem, R. S. Amant, K. Sankaralingam, and D. Burger, "Dark silicon and the end of multicore scaling," in *Proc. ISCA*, 2011, pp. 365–376.

[4] S. Pagani *et al.*, "TSP: Thermal safe power: Efficient power budgeting for many-core systems in dark silicon," in *Proc. CODES+ISSS*, 2014, p. 10.

[5] A.-M. Rahmani *et al.*, "Dynamic power management for many-core platforms in the dark silicon era: A multi-objective control approach," in *Proc. ISLPED*, 2015, pp. 219–224.

[6] K. Ma and X. Wang, "PGCapping: Exploiting power gating for power capping and core lifetime balancing in CMPs," in *Proc. PACT*, 2012, pp. 13–22.

[7] L. Wang and K. Skadron, "Implications of the power wall: Dim cores and reconfigurable logic," *IEEE Micro*, vol. 33, no. 5, pp. 40–48, Sep./Oct. 2013.

[8] N. Kapadia and S. Pasricha, "VARSHA: Variation and reliability-aware application scheduling with adaptive parallelism in the dark-silicon era," in *Proc. DATE*, 2015, pp. 1060–1065.

[9] T. Komoda, S. Hayashi, T. Nakada, S. Miwa, and H. Nakamura, "Power capping of CPU-GPU heterogeneous systems through coordinating DVFS and task mapping," in *Proc. ICCD*, 2013, pp. 349–356.

[10] A.-M. Rahmani *et al.*, "Smart e-Health gateway: Bringing intelligence to Internet-of-Things based ubiquitous healthcare systems," in *Proc. CCNC*, 2015, pp. 826–834.

[11] A. Kanduri *et al.*, "Approximation knob: Power capping meets energy efficiency," in *Proc. ICCAD*, 2016, pp. 1–8.

[12] M.-H. Haghbayan *et al.*, "Dark silicon aware power management for manycore systems under dynamic workloads," in *Proc. ICCD*, 2014, pp. 509–512.

[13] A. Vega, A. Buyuktosunoglu, H. Hanson, P. Bose, and S. Ramani, "Crank it up or dial it down: Coordinated multiprocessor frequency and folding control," in *Proc. MICRO*, 2013, pp. 210–221.

[14] R. Cochran, C. Hankendi, A. K. Coskun, and S. Reda, "Pack & Cap: Adaptive DVFS and thread packing under power caps," in *Proc. MICRO*, 2011, pp. 175–185.

[15] A. Kanduri, M.-H. Haghbayan, A.-M. Rahmani, P. Liljeberg, A. Jantsch, and H. Tenhunen, "Dark silicon aware runtime mapping for many-core systems: A patterning approach," in *Proc. ICCD*, 2015, pp. 573–580.

[16] M. Shafique, S. Garg, T. Mitra, S. Parameswaran, and J. Henkel, "Dark silicon as a challenge for hardware/software co-design," in *Proc. CODES+ISSS*, 2014, pp. 1–10.

[17] A. M. Rahmani, M.-H. Haghbayan, A. Miele, P. Liljeberg, A. Jantsch, and H. Tenhunen, "Reliability-aware runtime power management for many-core systems in the dark silicon era," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 25, no. 2, pp. 427–440, Feb. 2017.

[18] H. Chen, C. Hankendi, M. C. Caramanis, and A. K. Coskun, "Dynamic server power capping for enabling data center participation in power markets," in *Proc. ICCAD*, 2013, pp. 122–129.

[19] J. Ansel *et al.*, "PetaBricks: A language and compiler for algorithmic choice," *ACM SIGPLAN Notices*, vol. 44, no. 6, pp. 38–49, 2009.

[20] W. Baek and T. M. Chilimbi, "Green: A framework for supporting energy-conscious programming using controlled approximation," in *Proc. PLDI*, 2010, pp. 198–209.

[21] H. Hoffmann, S. Sidiroglou, M. Carbin, S. Misailovic, A. Agarwal, and M. Rinard, "Dynamic knobs for responsive power-aware computing," *ACM SIGPLAN Notices*, vol. 46, no. 3, pp. 199–212, 2012.

[22] S. Sidiroglou-Douskos, S. Misailovic, H. Hoffmann, and M. Rinard, "Managing performance vs. accuracy trade-offs with loop perforation," in *Proc. FSE*, 2011, pp. 124–134.

[23] T. N. Gia, M. Jiang, A.-M. Rahmani, T. Westerlund, P. Liljeberg, and H. Tenhunen, "Fog computing in healthcare Internet of Things: A case study on ECG feature extraction," in *Proc. CIT*, 2015, pp. 356–363.

[24] C. Tan, A. Kulkarni, V. Venkataramani, M. Karunaratne, T. Mitra, and L. S. Peh, "LOCUS: Low-power customizable many-core architecture for wearables," in *Proc. CASES*, 2016, p. 11.

[25] A. Anzanpour, A. M. Rahmani, P. Liljeberg, and H. Tenhunen, "Internet of Things enabled in-home health monitoring system using early warning score," in *Proc. MobiHealth*, 2015, pp. 174–177.

[26] A. Odunmbaku, A.-M. Rahmani, P. Liljeberg, and H. Tenhunen, "Elderly monitoring system with sleep and fall detector," in *Proc. HealthyIoT*, 2015, pp. 473–480.

[27] M. H. Haghbayan, A. Kanduri, A. M. Rahmani, P. Liljeberg, A. Jantsch, and H. Tenhunen, "MapPro: Proactive runtime mapping for dynamic workloads by quantifying ripple effect of applications on networks-on-chip," in *Proc. NOCS*, 2015, p. 26.

[28] T. E. Carlson, W. Heirmant, and L. Eeckhout, "Sniper: Exploring the level of abstraction for scalable and accurate parallel multi-core simulation," in *Proc. SC*, 2011, pp. 1–12.

[29] F. Fazzino, M. Palesi, and D. Patti. (2008). *Noxim: Network-on-Chip Simulator*. [Online]. Available: http://sourceforge. net/projects/noxim

[30] L. Wang and K. Skadron, "Dark vs. dim silicon and near-threshold computing extended results," Dept. Comput. Sci., Univ. Virginia, Charlottesville, VA, USA, Tech. Rep. CS-2013-01, 2012.
[31] S. Holmbacka, M. Fattah, W. Lund, A.-M. Rahmani, S. Lafond, and J. Lilius, "A task migration mechanism for distributed many-core operating systems," *J. Supercomput.*, vol. 68, no. 3, pp. 1141–1162, 2014.

**Anil Kanduri** (SM'14) received the B.Tech. degree in electronics and communications from Jawaharlal Nehru Technological University at Kakinada, Kakinada, India and the M.Sc. (Tech) degree in embedded computing from the University of Turku, Turku, Finland, where he is currently pursuing the Ph.D. degree with the Department of Information Technology.

His current research interests include energy efficient computer architectures, run-time management, and approximate computing.

**Mohammad-Hashem Haghbayan** (SM'14) received the B.A. degree in computer engineering from the Ferdowsi University of Mashhad, Mashhad, Iran, and the M.S. degree in computer architecture from the University of Tehran, Tehran, Iran. He is currently pursuing the Ph.D. degree with the University of Turku, Turku, Finland.

He has several years of experience working in industry as well as developing research tools before starting his Ph.D. His current research interests include high-performance energy-efficient architectures, power management techniques, and online/offline testing.

**Amir M. Rahmani** (M'08) received the M.Sc. degree from the University of Tehran, Tehran, Iran, in 2009, the Ph.D. degree from the University of Turku, Turku, Finland, in 2012, and the MBA degree from the Turku School of Economics, European Institute of Innovation and Technology ICT Labs, in 2014.

He is currently a Marie Curie Global Fellow with the University of California Irvine, Irvine, CA, USA, and with Technische Universität Wien, Vienna, Austria, and also an Adjunct Professor (Docent) in embedded parallel and distributed computing from the University of Turku. He has authored over 120 peer-reviewed publications.

**Pasi Liljeberg** (M'09) received the M.Sc. and Ph.D. degrees in electronics and information technology from the University of Turku, Turku, Finland, in 1999 and 2005, respectively.

He is currently a Full Professor in Embedded Computing Architectures with the Embedded Computer Systems Laboratory, University of Turku. His current research interests include parallel and distributed systems, Internet-of-Things, e-Health, embedded computing architecture, fog computing, 3-D multiprocessor system architectures, cyber-physical systems, and reconfigurable system design.

**Axel Jantsch** (M'97) received the Dipl.Ing. and Dr.Tech. degrees from the Technische Universität Wien (TU Wien), Vienna, Austria, in 1988 and 1992, respectively.

From 1997 to 2002, he was an Associate Professor with the KTH Royal Institute of Technology, Stockholm, Sweden, where he was also a Full Professor of Electronic Systems Design from 2002 to 2014. Since 2014, he has been a Professor with the Institute of Computer Technology, TU Wien. He has authored over 300 articles and one book in the areas of VLSI design and synthesis, HW/SW codesign and cosynthesis, networks-on-chip, and self-awareness in cyber-physical systems.

**Hannu Tenhunen** (M'83) received the Diploma degree from the Helsinki University of Technology, Espoo, Finland, in 1982, and the Ph.D. degree from Cornell University, Ithaca, NY, USA, in 1986.

In 1985, he joined the Signal Processing Laboratory, Tampere University of Technology, Tampere, Finland, as an Associate Professor and later served as a Professor and a Department Director. Since 1992, he has been a Professor with the KTH Royal Institute of Technology, Stockholm, Sweden, where he also served as a Dean. His current research interests include VLSI architectures and systems, especially network-on-chip systems.

**Nikil Dutt** (S'84–M'89–SM'96–F'08) received the Ph.D. degree in computer science from the University of Illinois at Urbana–Champaign, Champaign, IL, USA, in 1989.

He is currently a Chancellor's Professor of Computer Science with the Department of Electrical Engineering and Computer Science, University of California Irvine, Irvine, CA, USA. His current research interests include embedded systems, electronic design automation, computer architecture, systems software, formal methods, and brain-inspired computing.

Dr. Dutt is an ACM Distinguished Scientist. He received the IFIP Silver Core Award.

# Paper V

## Approximation-Aware Coordinated Power/Performance Management of Heterogeneous Multi-cores

Anil Kanduri, Antonio Miele, Amir Rahmani, Pasi Liljeberg, Cristiana Bolchini and Nikil Dutt

Published in 55th ACM/IEEE Design Automation Conference (DAC), 2018, San Francisco, USA, Article No: 68

# Approximation-Aware Coordinated Power/Performance Management for Heterogeneous Multi-cores

Anil Kanduri [†], Antonio Miele[‡], Amir M. Rahmani[§¶], Pasi Liljeberg[†], Cristiana Bolchini[‡], Nikil Dutt[§]
[†] University of Turku, Finland [‡] Politechnico de Milano, Italy [§] University of California, Irvine, USA [¶] TU Wien Austria
{spakan@utu.fi, antonio.miele@polimi.it, amirr1@uci.edu, pakrli@utu.fi, cristiana.bolchini@polimi.it, dutt@ics.uci.edu}

## ABSTRACT

Run-time resource management of heterogeneous multi-core systems is challenging due to i) dynamic workloads, that often result in ii) conflicting knob actuation decisions, which potentially iii) compromise on performance for thermal safety. We present a run-time resource management strategy for performance guarantees under power constraints using functionally approximate kernels that exploit accuracy-performance trade-offs within error resilient applications. Our controller integrates approximation with power knobs - DVFS, CPU quota, task migration - in coordinated manner to make performance-aware decisions on power management under variable workloads. Experimental results on Odroid XU3 show the effectiveness of this strategy in meeting performance requirements without power violations compared to existing solutions.

## CCS CONCEPTS

• **Hardware → On-chip resource management**; • **Computer systems organization →** *Multicore architectures*;

## KEYWORDS

On-chip resource management, approximate computing

## 1 INTRODUCTION

Heterogeneous multi-core systems that are resource constrained exacerbate the run-time management challenge with i) diverse performance requirements of applications ii) fixed power budgets iii) dynamic workload characteristics iv) core-level heterogeneity. Existing resource management techniques use Dynamic Voltage and Frequency Scaling (DVFS) [2], task migration [15], power gating and CPU quota scaling [18, 19] etc., for power optimization. Such techniques propose joint actuation of power knobs [2, 14, 15, 18, 25] and application characteristics exploitation [7, 9] to maximize performance within the fixed power budgets. These strategies minimize the effect of power actuation on performance, yet inevitably compromise on performance for thermal safety. This specifically affects a class of streaming applications from domains such as machine learning, artificial intelligence, multimedia processing, computer vision and Internet-of-Things [12, 21]. These applications may include user interaction and serve as intermediate results to other computational kernels (e.g., computer vision in self-driving vehicles), making them latency-sensitive. Typical power actuation techniques cannot provide real-time performance guarantees required to ensure user satisfaction and responsiveness. At the same time, such applications are relatively error resilient (i.e., approximable) due to their algorithmic tolerance nature, redundant input data and perceptive end results. Approximate computing [4] has become a popular choice for energy and performance gains exploiting the error resilience of certain application domains. Functionally approximate kernels present a Pareto space of accuracy-performance trade-offs that can be leveraged at run-time to achieve better performance within lower power and/or energy resources, for an acceptable loss in accuracy [1, 22]. In this work, we propose a novel run-time resource management strategy for heterogeneous multi-core systems running dynamic and unknown workloads, exploiting approximation together with traditional power knobs. Our strategy jointly coordinates with power knobs and approximation knob to scale accuracy of the applications as per available resources, when power/performance requirements cannot be met with traditional knobs. Subject to system dynamics and applications' requirements, we switch application's mode of execution from accurate to approximate to meet i) power constraints - by lowering resources to approximate kernels, and/or ii) performance requirements - by opportunistically reducing workloads. Coordinated usage of approximation knob complements the traditional power knobs to overcome their limitations and conflicting decisions. The major contributions of our work are:

- Coordinated actuation of approximation knob with other power knobs to satisfy both power and performance constraints subject to system dynamics,
- Resource allocation policy for heterogeneous multi-core systems running unknown/dynamic workloads using the coordinated actuation strategy through DVFS, CPU quota assignment, task migration and approximation,
- Evaluation of our strategy on a real test-bed of Odroid XU3 8-core ARM big.LITTLE platform, over a set of error resilient machine learning applications.

**Organization:** Motivation for coordinated actuation of approximation with other power knobs and related work are presented in Sections 2 and 3, highlighting the novelty of our solution. Section 4 discusses the background on system architecture. The proposed resource management strategy for power-performance actuation is described in Section 5. Section 6 presents an evaluation of our approach with selective workloads on Odroid XU3 HMP platform. Finally, Section 7 concludes the paper with future work.


*DAC '18, June 24–29, 2018, San Francisco, CA, USA*

ACM ISBN 978-1-4503-5700-5/18/06…$15.00
https://doi.org/10.1145/3195970.3195994

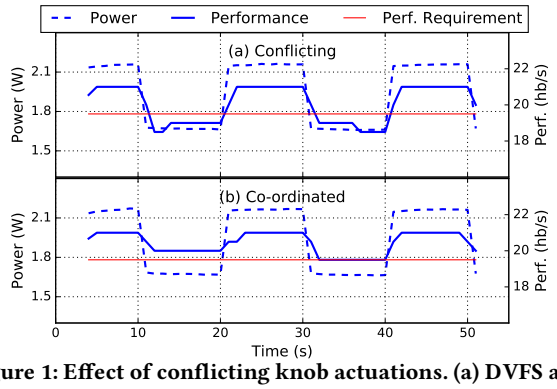



**Figure 1: Effect of conflicting knob actuations. (a) DVFS and APX are triggered without co-ordination [13]. (b) DVFS and APX are tuned with co-ordination.**

## 2 MOTIVATION

We demonstrate the effect of combining approximation (APX) knob with power actuation through an example, using the control strategy presented in [13]. For simplicity, we use only DVFS as the power knob, and loop perforation [22] based approximation as the APX knob. As a test case, we use least squares curve fitting application over 24000 pairs of inputs, and 10% of the loops skipped to realize the approximate version. Figure 1(a) shows the power consumption and corresponding performance (measured in heartbeat/s, as described in Section 4) of the application. We assume $2W$ as the threshold for power actuation, 10 seconds (for presentation ease) as the control period for resource management decisions , along with a target performance requirement (the red line). At $t = 10s$, power exceeds the threshold, triggering DVFS to reduce power, while performance requirements are satisfied. Subsequently, power consumption is reduced during the control period, also reducing the performance below the requirement. At $t = 20s$, performance requirements are not met - triggering the APX knob, while power consumption is below the threshold - triggering DVFS (upscaling). With the uncoordinated actuation decisions performed by [13], performance is restored with some accuracy loss, while power consumption again exceeds the threshold during the following control period. At $t = 30s$, performance requirements are met, hence the application is switched to accurate execution; DVFS is triggered again to lower power consumption. A similar oscillation between accurate-approximate executions and high-low frequencies will continue in the subsequent control periods. Although the performance violation at $t = 10s$ could be recovered with approximation at $t = 20s$, the lack of coordination between DVFS and approximation either over-compensates or under-compensates the knobs' actuation. This occurs because of the *reactive nested loop structure of such control policies (as in [13]), where power and performance actuation happens independently, and is mutually agnostic.*

Figure 1(b) shows power and performance for the same scenario, with a coordination among DVFS and approximation. At $t = 10s$, DVFS is triggered to address power violation. At the same time, considering the performance loss with power actuation, approximation is triggered pro-actively by predicting the potential loss in performance due to DVFS (downscaling). As a result, performance requirements are met during the subsequent control period despite DVFS downscaling. At $t = 30s$, VF is upscaled with availability of power headroom and the same is indicated to the performance manager. This allows the performance manager to make informed decisions on performance actuation, which in turn restores the accurate mode of execution. As a conclusion, while combination of approximation and DVFS addresses the performance loss due to power actuation to an extent, tight coordination between power and performance actuation provides better results in i) meeting performance requirements often, and ii) minimizing the accuracy loss due to approximation. In a realistic scenario, the resource management policy has to consider 1) a wider set of power knobs viz., DVFS, CPU quota assignment, task migration and fine-grained levels of approximation, and 2) variable workload characteristics having applications entering and leaving the system with an unknown trend, and each one being characterized by a specific performance requirement. When considering these aspects, knob actuation for maximizing performance within minimal power consumption and accuracy loss becomes dramatically complex and challenging. Our goal is therefore to address this issue by enabling coordinated decision making on power knobs and approximation actuation.

## 3 RELATED WORK

**Resource Management.** We primarily focus on run-time resource management techniques proposed for heterogeneous multi-core systems, targeting big.LITTLE architecture. HMP is the last version of Linux scheduler for big.LITTLE architectures designed by Samsung [26]. It maps compute-intensive applications on the big cluster and I/O bounded ones on the LITTLE cluster. However, it neither considers performance requirements expressed by the user, nor acts on CPU quota and DVFS. *Race-to-idle* policy is not efficient in the case of asymmetric processors for neither power dissipation nor energy consumption, as shown in [11]. Therefore, advanced strategies are required especially when considering also performance requirements. Numerous works enhanced the HMP idea with advanced run-time resource management to meet performance requirements while concurrently optimizing energy and power consumption [3, 6, 10, 15]. The most representative one is [15] - using task mapping, DVFS, CPU quota and task migration, which eventually coordinate and converge towards meeting performance requirements within power limits.

**Run-time Approximation.** [24] has proposed a static scheduling algorithm considering various approximate versions of tasks in an application to maximize performance within a given error bound, while honoring power constraints. It is assumed that approximate versions of tasks (based on loop perforation) are provided before hand and the scheduler relies on an off-line heuristic to determine scheduling decisions of the approximate task. [13] proposes the use of approximation as a performance knob in covering up for the performance lost during power capping in many-core architectures. They assume two approximation levels to be provided for each task and switch the execution mode from accurate to approximate and vice-versa based on performance surges. No tight coordination between approximation and other power knobs is the limit of this work, as shown in Figure 1. Run-time approximation for thermal management of a single video application through content-aware error resilience characterization has been proposed in [16]. It defines four approximate levels for reducing power and temperature opportunistically over specific cases of video applications, however

**Table 1: Qualitative comparison of proposed solution w.r.t. existing approaches.**

| Technique | Goal | Multi-Programmed | Unknown Workload | Knobs | | | | APX Levels | Coord. |
|---|---|---|---|---|---|---|---|---|---|
| | | | | DVFS | CPU quota | Task Mig. | APX | | |
| HierCtrl [15] | Performance within power cap | ✓ | ✓ | ✓ | ✓ | ✓ | ✘ | ✘ | +/- |
| ApxSched [24] | Energy efficiency and QoS | ✓ | ✘ | ✓ | ✓ | ✓ | ✓ | 5 | ✘ |
| ApxTherm [16] | Thermal optimization | ✘ | ✘ | ✓ | ✘ | ✘ | ✓ | 4 | ✘ |
| ApxKnob [13] | Power capping | ✓ | ✓ | ✓ | ✓ | ✘ | ✓ | 2 | +/- |
| **Our Appr.** | **Performance within power cap** | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | **Fine-grained** | ✓ |

it does not use approximation from a resource management perspective, nor it deals with multi-programmed workloads. Finally, [23] proposes a proactive control of knobs for approximating a given program, satisfying a set of system parameters for minimum loss in accuracy. This technique is based on an off-line learning, where the approximation knob takes into account, for a given set of inputs, the introduced error and the achieved performance improvement; moreover, no resource management is considered.

Table 1 reports and compares the most representative among the mentioned existing solutions, by listing their supported features. The last line in the table shows our proposed approach, which considers a more comprehensive scenario compared to the others, addressing the resource management and application approximation in a more fine-grained and tightly coordinated way.

## 4 SYSTEM ARCHITECTURE

Figure 2 presents our system overview comprising hardware, software and interface. The specific processing platform deploys the well-known ARM big.LITTLE asymmetric multi-core model, composed of a cluster of high-performance power-hungry *big* cores, and an alternative cluster of low-power *LITTLE* cores. The processor is constrained by a Thermal Design Power (TDP) [5] - the conservative maximum power specified for the chip's thermal safety, and is provided with per-cluster DVFS and power sensors to support power management. The loaded Operating System (OS) utilities provide control over resource allocations through: task migration between different clusters, task mapping on a specific core within a cluster, and CPU quota assignment to set the percentage of time/resources the core will devote per application. The overall system we propose is similar to the ones used in [15, 17, 24] - the key distinctive aspect of our approach is the APX knob set by the controller to determine the accuracy levels of the applications at run-time.

In this work, we target applications with a computationally intensive loop, where the main kernel is continuously repeated. Applications may enter and leave the system with an unknown trend, thus causing a highly variable workload. The applications are assumed to be enhanced with the HeartBeat API [8] for a throughput measure (in terms of loop iterations or *heartbeats*, performed per unit of time, $hb/s$, which is proportional to the amount of processed data per unit of time). This also allows expression of user-level performance requirements, in terms of the minimum throughput to be guaranteed. Finally, we assume the compute kernel to be possibly implemented with an approximation strategy, e.g., the loop perforation [22], that can be tuned at run-time.

The last component in Figure 2 is a run-time controller (similar to [15]), capable of accessing the described system-level sensors and actuation knobs through the OS interfaces, and the application-level ones through the HeartBeats API. The proposed run-time

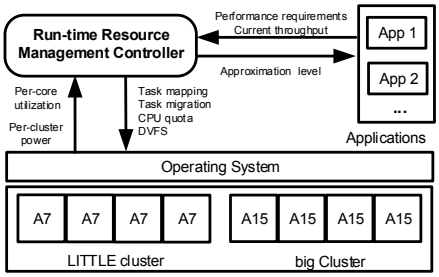



**Figure 2: System Architecture.**

resource management policy implemented as the controller module is presented in the following section.

## 5 THE PROPOSED POLICY

The run-time resource management policy functions in *idle*, *decide* and *refine* phases, implemented as a finite state machine shown in Figure 3, and described as follows.

### 5.1 Policy phases

**Idle.** This phase corresponds to the unloaded system or to a thoroughly balanced one, satisfying power and performance requirements, eventually. When in this phase, the controller monitors relevant events causing significant variations in system dynamics. Such an event typically occurs when a new application enters the system (requesting system resources), or a currently running application leaves it (freeing them). In such events resource allocation decisions are to made, by transitioning into the *decide* phase.

**Decide.** Resource allocation decisions on core selection, DVFS, CPU quota, task migration and approximation are made in this phase. When a new application enters the system, it is conservatively mapped on a LITTLE core. Each time an application enters/exits, the running workload on each cluster is re-mapped to make utilization uniform on the various cores. Actuation decisions are based on power and throughput measurements from the OS-level and application-level monitoring interfaces. Knob settings are determined to be proportional to applications' performance requirement and power budget available. Failing to meet an application's performance requirement or violating the upper bound on power budget are two instances when decision making becomes critical. Upon such events, the controller decides between combinations of power knobs alone (DVFS, CPU quota and/or task migration) and/or power knobs combined with approximation, subject to whichever yields better performance within acceptable power constraints and accuracy loss (the logical flow of this strategy is presented in a more detailed way in the next subsection). We use estimation models (detailed in Section 5.3) to predict near-optimal knob settings during the *decide* phase. They minimize the knob setting space to be pruned, allowing a faster arrival at a near-optimal

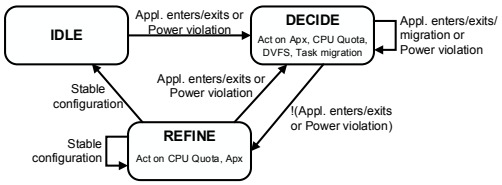



**Figure 3: Work flow of the designed policy.**

solution. Estimating performance loss/gain over every power actuation decision allows to establish coordination among conflicting power-approximation knobs settings. At the end, selected resource allocation decisions are enforced. If task migration has been performed, this significantly alters application's performance thus the *decide* phase is repeated; otherwise the subsequent *refine* phase will take place.

**Refine.** The *refine* phase monitors power and performance metrics and fine-tunes any coarse-gained knob settings enforced during the *decide* phase. This provides precise control over resource allocation, primarily using CPU quota and APX knobs. Further, any possible aberrations, or over/under compensation decisions made during the *decide* phase, will be converged towards required knob settings in the subsequent *refine* phase so that a stable resource allocation is eventually reached. The *refine* phase will last as long as needed to ensure the performance target, for all running applications, is met within the power budget.

## 5.2 The decision strategy

The algorithmic flow of the *decide* phase is shown in Algorithm 1. Actuation decisions are invoked in two cases - a power violation, i.e., measured power consumption ($P$) exceeding $TDP$ (Lines 1–13) and power constraints honored (Lines 14–45), detailed next.

**Power Violation.** Recovery actions have to be taken by acting mainly on the big cluster since it is the responsible of relevant power consumption. Application(s) that have a minimal performance penalty (set to $< 10\%$) upon migrating to the LITTLE cluster are chosen, if any (Line 3), and each selected application is migrated from the big cluster to the LITTLE one. Approximation is invoked simultaneously, by setting an accuracy level that is proportional to the estimated loss in performance with task migration (Lines 3–5). This ensures that a potential performance loss with migrating to the low-power LITTLE cluster is pro-actively addressed with reducing the workload through approximation. If no application with tolerable performance loss upon task migration is available, DVFS is used for reducing power consumption (Lines 7–13). Target voltage and frequency (VF) levels are determined by the ratio of current power consumption and budget $P/TDP$, using the estimation model in Equation 1 (presented next – Line 7). For each application, the possible performance loss with lowered VF levels is estimated with the model in Equation 4 (Line 10). Approximation is triggered over such applications that could suffer performance loss with the new VF levels. The level of approximation is set proportional to the estimated performance loss (Lines 12–13). In this scenario, DVFS and APX are triggered coordinately such that any possible performance loss incurred in power actuation is pro-actively and proportionally covered up through approximation, using the estimation models.

**Power Budget Honored.** We analyze each cluster separately (Line 15). We start on the LITTLE and we first identify possible applications that do not meet target performance at the highest resources. If any, such applications are migrated on the big cluster and the current decision phase ends and it will be restarted on the subsequent control period to have an updated measure of the throughput in the new configuration (Lines 18–22).

After that, on each cluster the application having the highest assigned CPU quota is chosen as the most demanding (Line 25). The VF level is then determined such that performance requirements of this selected application are met at maximum CPU quota; indeed as shown in [15, 17] selecting the minimum necessary VF level at maximum CPU quota minimizes power consumption among all configurations offering the same level of performance (Lines 26–27). Performance of other applications is estimated with the newly determined VF level (Line 29). The applications which have met the performance requirements and were approximated in the previous decision phases are recovered back to accuracy, subjective and proportional to the availability of surplus power budget and current performance levels (Lines 30–32). Among the other applications, CPU quota of applications that have already met the performance

**Algorithm 1** Decide phase workflow.

**Inputs:** $Apps$ : Applications currently running
**Global Variables:** $U$ : Overall utilization of all the cores
$V_b$, $F_b$, $V_l$, $F_l$ : VFs of big and LITTLE clusters
$P$: Measured power consumption
**Global Constants:** $TDP$: Power band limit
$migPenalty$: Performance penalty threshold upon task migration

**Body:**

1: **if** $P \geq TDP$ **then**
2: **for** every $app_i$ in $Apps \in big$ **do**
3: **if** $\frac{app_i.maxPerfOnLITTLE}{app_i.targetPerf} \geq 1 - migPenalty$ **then**
4: $TaskMigration(app_i, LITTLE)$ //it forces decide phase re-execution
5: $APX(app_i, migPenalty)$
6: **else**
7: $V_r, F_r \leftarrow powerEstimate(V_l, F_l, U, P/TDP)$
8: $setDVFS(V_r, F_r)$
9: **for** every $app_i$ in $Apps$ **do**
10: $app_i.perfEstimate \leftarrow perfEstimate(V_r, F_r, app_i.quota)$
11: **if** $app.perfEstimate < 1$ **then**
12: $loss \leftarrow 1 - \frac{app_i.perfEstimate}{app_i.perfTarget}$
13: $APX(app_i, loss)$
14: **else**
15: **for** each cluster $c \in \{big, LITTLE\}$ **do**
16: $Apps_c \leftarrow get\_appls\_on\_cluster(c)$
17: $redecide \leftarrow false$
18: **if** $c = LITTLE$ **then**
19: **for** every $app_i$ in $Apps_c$ **do**
20: **if** $app_i.perf \leq app_i.targetPerf$ **then**
21: $TaskMigration(app_i, big)$
22: $redecide \leftarrow true$
23: **if** $redecide = true$ **then**
24: exit() //it forces decide phase re-execution
25: $appH \leftarrow get\_appl\_with\_max\_CPUquota(Apps_c)$
26: $V_r, F_r \leftarrow powerEstimate(V_c, F_c, U, appH.perf)$
27: $setDVFS(V_r, F_r)$
28: **for** every $app_i$ in $Apps_c$ **do**
29: $app_i.perfEstimate \leftarrow perfEstimate(V_r, F_r, app_i.quota)$
30: **if** $app.perfEstimate > 1$ **then**
31: **if** $app_i.apx > 0$ **then**
32: $APX(app_i, app_i.perf)$
33: **else**
34: $q_{old} \leftarrow app_i.quota$
35: $q \leftarrow perfEstimate(V_r, F_r, app_i.quota)$
36: $setQuota(app_i, q)$
37: $U \leftarrow U - q_{old} + q$
38: **else**
39: $q_{old} \leftarrow app_i.quota$
40: $q \leftarrow perfEstimate(V_r, F_r, app_i.quota)$
41: $setQuota(app_i, q)$
42: $U \leftarrow U - q_{old} + q$
43: $app_i.perfEstimate \leftarrow perfEstimate(V_r, F_r, u)$
44: **if** $app_i.perfEstimate < 1$ **then**
45: $APX(app_i, app_i.perf)$

requirements (if any) are lowered proportionally within their performance requirements (Lines 33–37). For applications that do not meet the performance requirements with new VF settings, CPU quota is increased proportional to the target throughput if possible (Lines 39–42). In case of persistent performance violation, approximation is invoked, setting the level of approximation proportional to the loss in performance, as estimated (Lines 43–45).

### 5.3 Power and Performance Estimates

Within the discussed Algorithm 1, we used performance and power estimation models adapted from [17, 20, 24] and experimentally verified on the considered board. We represent power consumption $P_j$ of a cluster $j$ running a set of $N$ applications $App_i$ as a function of the overall cluster utilization $U_j$ and the related $V_j, F_j$. As presented in [17, 20, 24], a fast/almost-accurate estimation of power for each $V_j, F_j$ pair can be obtained as a function of $U_j$:

$$P_{V_j,F_j} = a_{V_j,F_j} \cdot U_j + b_{V_j,F_j} \tag{1}$$

where $a_{V_j,F_j}$ and $b_{V_j,F_j}$ are empirically derived at each $V_j/F_j$ pair. Utilization $U_j$ of a cluster $j$ indicates the amount of time in a given control period for which the included cores are busy:

$$U_j = \sum_{i=0}^{N} U_i \tag{2}$$

where $N$ is the number of cores in the cluster. Moreover, $U_i$ measure is directly provided by the OS, whereas the assigned CPU quota is directly proportional to the utilization. Then, power consumption $P_t$ of the system is the sum of values of the two clusters. As shown in previous works (e.g.[17, 24]), performance ($Perf_i$) of an application $i$ has an almost linear relationship with the CPU quota $Q$, VF setting of the cluster it is mapped on, expressed as:

$$Perf(i) = \alpha Q(App_i)F_i \tag{3}$$

where $\alpha$ is a variable parameter that depends on application characteristics and cluster where the application is running. Thus, we estimate the performance penalty with actuation of power knobs viz., CPU quota assignment, VF level and task migration from a configuration *old* to the *new* one as:

$$Penalty(App_i) = \frac{Perf_{new}}{Perf_{old}} = \frac{\alpha_{new}Q_{new}(App_i)F_{new}}{\alpha_{old}Q_{old}(App_i)F_{old}} \tag{4}$$

## 6 EXPERIMENTAL RESULTS

**Experimental Setup.** For the experimental evaluation, we considered an Odroid XU3 board running Linux Ubuntu 15.04. It features Samsung Exynos 5422 with 4 ARM A7 (LITTLE) and 4 ARM A15 (big) cores, operating within $200-1400$MHz and $200-2000$MHz frequencies respectively. TDP was set to $5W$, considering a restriction on-chip temperature to 80°C. The controller has been implemented in C++ and runs as a user space process. It uses Linux drivers to access on-board sensors for power measures and to apply DVFS settings. Application's execution and allocation of CPU quota are enabled through CGroups Linux library. Communication between applications and the controller is established through Linux shared memory mechanism. An in-house implementation of HeartBeats API uses this shared memory space to periodically 1) write the application throughput information - to enable the controller's throughput monitoring and 2) read approximation levels determined by the controller - to dynamically switch the application accuracy level. Approximation level is passed as parameter to the kernel function, dynamically creating a range of fine-grained approximate versions without any compilation and memory overhead. For experimentation, we used four applications from machine learning domain, listed in Table 2 and the type of software approximations used. Larger number of iterations over input data yields results that converge towards optimal result due to algorithmic nature of these applications. We only temporarily trade the accuracy to meet power/performance requirements, while accurate execution is enabled when enough resources are available. The epoch for the controller invocation is parametrizable and can be tuned as per platform and applications' characteristics. In our evaluation, we set the epoch to 1$s$ to collect stable throughput measurements upon knob actuation, given the streaming nature of the chosen applications.

**Table 2: Simulated workload.**

| Application | Input | Approximation |
|---|---|---|
| LeastSq | 1 million pairs | loop perforation |
| KNN | 25k points and 25 test cases | loop perforation and task skipping |
| kMeans | 50k points into 3 clusters | relaxed convergence |
| LinReg | 1 million pairs | relaxed convergence |

**Evaluation.** To test our proposed solution, we created a dynamic workload scenario ranging from 1 to 4 concurrently running applications, emulating unknown workloads. We compare it against similar state-of-the-art approaches, namely ApxKnob [13] and HierCtrl [15]. These approaches exploit the uncoordinated usage of the APX knob with other power knobs, and the coordinated usage of power knobs without any approximation, respectively. All systems are fed with the same dynamic workload scenarios. Figure 4 shows a comparison of power consumption, and Figure 5 reports each application's performance against the set requirements over the workloads (with a $-1/+1hb$ tolerance band). Table 3 lists the average power consumption and percentage of power violations for each approach. ApxKnob violates power budget the most, at 35% of the overall execution period, mainly with conflicting power-performance decisions resulting in constant oscillation between high-low power-performance states. HierCtrl has a much lower power violation at 3.5%, as different controllers converge eventually towards stable configurations. Our solution has a negligible 0.5% power violation with pro-active and coordinated knob actuation, benefiting from the estimation models. Table 3 also reports for every approach, the performance guarantees in terms of the percentage of execution time during which the requested performance is achieved for each application. The loss in accuracy with approximation knob is presented as the weighted sum of execution time and approximation level set, also shown in Table 3. ApxKnob meets the performance requirements mostly by leveraging approximation, and largely due to uninhibited power usage. Although the loss in accuracy for performance guarantees is acceptable, this approach is limited by frequent thermal violations. HierCtrl relies on degrading throughput requirements to address power emergencies, resulting in dramatic performance loss, particularly under heavy workloads. Although HierCtrl achieves the lowest average power consumption

**Table 3: Power, performance and approximation behavior.**

| Technique | Avg. Power (W) | Power Violation (%) | Perf. Violation (%) | | | | Approximation (%) | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | LR | LeSq | KNN | kM | LR | LeSq | KNN | kM |
| ApxKnob [13] | 4.6 | 35.8 | 1.6 | 15.3 | 3.5 | 4.5 | 0 | 2 | 5.3 | 2.6 |
| HPM [15] | 2.82 | 3.5 | 28.6 | 89.7 | 44.2 | 42.6 | 0 | 0 | 0 | 0 |
| **Our Appr.** | 3.34 | 0.58 | 0.62 | 3.04 | 1.85 | 1.3 | 0.1 | 5 | 6.4 | 3.9 |

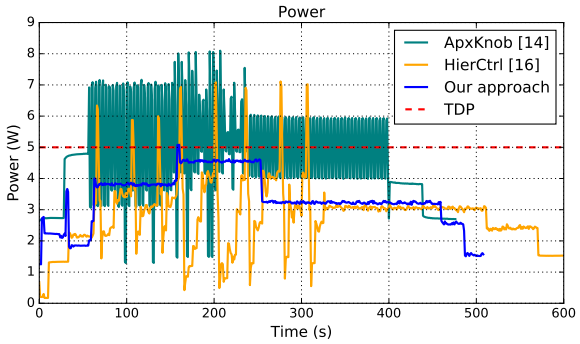



**Figure 4: Power consumption measures.**

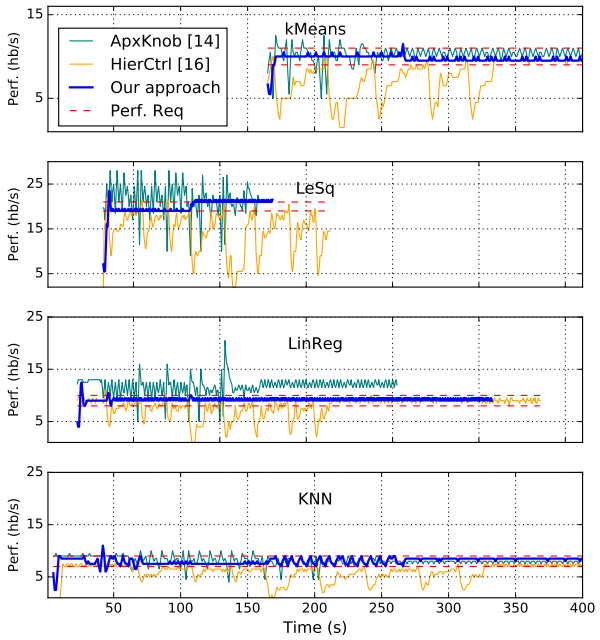



**Figure 5: Performance measures for different applications.**

and has a lower power violation rate, performance losses are significant. Our proposed approach, on the other hand, balances both performance requirements and power constraints, in particular, it is effective in delivering the requested performance, within minimal power consumption and an acceptable loss of accuracy. The effectiveness of approximation knob in power management although is trivial, requires disciplined tuning to achieve performance gains within power limits. Figure 5 shows that ApxKnob over compensates for performance (resulting in power budget violation), while HierCtrl violates performance requirements (with conservative power actuation). Our approach tunes approximation effectively to obtain the requested performance while minimizing the necessary power consumption. This can also be observed in Figure 5, where our approach consistently provides precisely the required performance, while ApxKnob and HierCtrl often over/under provision, resulting in their respective power and performance violations.

## 7 CONCLUSIONS

We presented a run-time resource management policy for asymmetric multi-cores that integrates functional approximation with other power knobs viz., DVFS, CPU quota assignment and task migration in a disciplined manner, to make performance-aware decisions on power management. The designed solution is able to minimize the penalty of power actuation decisions leading to performance loss with coordinated invocation of approximation together with other power knobs. Experimental results performed on an Odroid XU3 board demonstrates the effectiveness of the proposed solution w.r.t. the most representative existing approaches. Future work will focus on considering the management of multi-thread applications possibly targeting also the GPU resource.

## ACKNOWLEDGMENT

We acknowledge financial support by the Marie Curie Actions of the European Union's H2020 Programme.

## REFERENCES

[1] W. Baek et al. Green: A Framework for Supporting Energy-Conscious Programming using Controlled Approximation. In *PLDI*, 2010.
[2] R. Cochran et al. Pack & cap: adaptive dvfs and thread packing under power caps. In *MICRO*, 2011.
[3] B. Donyanavard et al. SPARTA: Runtime Task Allocation for Energy Efficient Heterogeneous Many-cores. In *Proc. of CODES+ISSS*, pages 27:1–27:10, 2016.
[4] H. Esmaeilzadeh et al. Neural Acceleration for General-Purpose Approximate Programs. In *MICRO*, 2012.
[5] S. Pagani et al. TSP: Thermal Safe Power: Efficient Power Budgeting for many-core systems in dark silicon era. In *CODES+ISSS*, 2014.
[6] F. Gaspar et al. Performance-Aware Task Management and Frequency Scaling in Embedded Systems. In *In Proc. of SBAC-PAD*, pages 65–72, 2014.
[7] F Gaspar et al. A framework for application-guided task management on heterogeneous embedded systems. *ACM TACO*, 12(4):42, 2016.
[8] H. Hoffmann et al. Application heartbeats: a generic interface for specifying program performance and goals in autonomous computing environments. In *Proc. Int. Conf. on Autonomic computing*, pages 79–88. ACM, 2010.
[9] H. Hoffmann et al. Dynamic knobs for responsive power-aware computing. *ACM SIGPLAN Notices*, 2012.
[10] S. Holmback et al. Performance Monitor Based Power Management for big.LITTLE Platforms. In *Proc. HIPEAC Workshop on Energy Efficiency with Heterogeneous Computing*, pages 1–6, 2015.
[11] C. Imes and H. Hoffmann. Minimizing Energy Under Performance Constraints on Embedded Platforms: Resource Allocation Heuristics for Homogeneous and single-ISA Heterogeneous Multi-cores. *SIGBED Rev.*, 11(4):49–54, January 2015.
[12] Norman P Jouppi et al. In-datacenter performance analysis of a tensor processing unit. *SIGARCH Comput. Archit. News*, 45(2):1–12, June 2017.
[13] A. Kanduri et al. Approximation knob: Power capping meets energy efficiency. In *In Proc. of ICCAD*, pages 1–8. IEEE, 2016.
[14] K. Ma and X. Wang. PGCapping: Exploiting power gating for power capping and core lifetime balancing in CMPs. *PACT*, 2012.
[15] T.S. Muthukaruppan et al. Hierarchical power management for asymmetric multi-core in dark silicon era. In *Proc. of DAC*, pages 1–9, 2013.
[16] D. Palomino et al. Thermal optimization using adaptive approximate computing for video coding. In *In Proc. of DATE*, pages 1207–1212, 2016.
[17] A. Pathania et al. Integrated CPU-GPU power management for 3D mobile games. In *Proc. of DAC*, pages 1–6, 2014.
[18] A. Rahmani et al. Dynamic power management for many-core platforms in the dark silicon era: A multi-objective control approach. In *ISLPED*, 2015.
[19] A.M. Rahmani, P. Liljeberg, A. Hemani, A. Jantsch, and H. Tenhunen. *The Dark Side of Silicon*. Springer, 1st edition edition, 2016.
[20] H. Rexha et al. Core Level Utilization for Achieving Energy Efficiency in Heterogeneous Systems. In *Proc. of PDP*, pages 401–407, 2017.
[21] P. Schulz et al. Latency critical iot applications in 5g: Perspective on the design of radio interface and network architecture. *IEEE Communications Magazine*, 55(2):70–78, 2017.
[22] S. Sidiroglou et al. Managing performance vs. accuracy trade-offs with loop perforation. In *FSE*, 2011.
[23] X. Sui et al. Proactive control of approximate programs. In *Proc. of ASPLOS*, pages 607–621. ACM, 2016.
[24] C. Tan et al. Approximation-aware scheduling on heterogeneous multi-core architectures. In *Proc. of ASP-DAC*, pages 618–623, 2015.
[25] A. Vega et al. Crank it up or dial it down: Coordinated multiprocessor frequency and folding control. In *MICRO*, pages 210–221, 2013.
[26] K. Yu et al. Power-aware task scheduling for big.LITTLE mobile processor. In *Proc. Int. SoC Design Conf.*, pages 208–212, 2013.

# Turku Centre for Computer Science
# TUCS Dissertations

1. **Marjo Lipponen**, On Primitive Solutions of the Post Correspondence Problem
2. **Timo Käkölä**, Dual Information Systems in Hyperknowledge Organizations
3. **Ville Leppänen**, Studies on the Realization of PRAM
4. **Cunsheng Ding**, Cryptographic Counter Generators
5. **Sami Viitanen**, Some New Global Optimization Algorithms
6. **Tapio Salakoski**, Representative Classification of Protein Structures
7. **Thomas Långbacka**, An Interactive Environment Supporting the Development of Formally Correct Programs
8. **Thomas Finne**, A Decision Support System for Improving Information Security
9. **Valeria Mihalache**, Cooperation, Communication, Control. Investigations on Grammar Systems.
10. **Marina Waldén**, Formal Reasoning About Distributed Algorithms
11. **Tero Laihonen**, Estimates on the Covering Radius When the Dual Distance is Known
12. **Lucian Ilie**, Decision Problems on Orders of Words
13. **Jukkapekka Hekanaho**, An Evolutionary Approach to Concept Learning
14. **Jouni Järvinen**, Knowledge Representation and Rough Sets
15. **Tomi Pasanen**, In-Place Algorithms for Sorting Problems
16. **Mika Johnsson**, Operational and Tactical Level Optimization in Printed Circuit Board Assembly
17. **Mats Aspnäs**, Multiprocessor Architecture and Programming: The Hathi-2 System
18. **Anna Mikhajlova**, Ensuring Correctness of Object and Component Systems
19. **Vesa Torvinen**, Construction and Evaluation of the Labour Game Method
20. **Jorma Boberg**, Cluster Analysis. A Mathematical Approach with Applications to Protein Structures
21. **Leonid Mikhajlov**, Software Reuse Mechanisms and Techniques: Safety Versus Flexibility
22. **Timo Kaukoranta**, Iterative and Hierarchical Methods for Codebook Generation in Vector Quantization
23. **Gábor Magyar**, On Solution Approaches for Some Industrially Motivated Combinatorial Optimization Problems
24. **Linas Laibinis**, Mechanised Formal Reasoning About Modular Programs
25. **Shuhua Liu**, Improving Executive Support in Strategic Scanning with Software Agent Systems
26. **Jaakko Järvi**, New Techniques in Generic Programming – C++ is more Intentional than Intended
27. **Jan-Christian Lehtinen**, Reproducing Kernel Splines in the Analysis of Medical Data
28. **Martin Büchi**, Safe Language Mechanisms for Modularization and Concurrency
29. **Elena Troubitsyna**, Stepwise Development of Dependable Systems
30. **Janne Näppi**, Computer-Assisted Diagnosis of Breast Calcifications
31. **Jianming Liang**, Dynamic Chest Images Analysis
32. **Tiberiu Seceleanu**, Systematic Design of Synchronous Digital Circuits
33. **Tero Aittokallio**, Characterization and Modelling of the Cardiorespiratory System in Sleep-Disordered Breathing
34. **Ivan Porres**, Modeling and Analyzing Software Behavior in UML
35. **Mauno Rönkkö**, Stepwise Development of Hybrid Systems
36. **Jouni Smed**, Production Planning in Printed Circuit Board Assembly
37. **Vesa Halava**, The Post Correspondence Problem for Market Morphisms
38. **Ion Petre**, Commutation Problems on Sets of Words and Formal Power Series
39. **Vladimir Kvassov**, Information Technology and the Productivity of Managerial Work
40. **Frank Tétard**, Managers, Fragmentation of Working Time, and Information Systems

41. **Jan Manuch**, Defect Theorems and Infinite Words
42. **Kalle Ranto**, $Z_4$-Goethals Codes, Decoding and Designs
43. **Arto Lepistö**, On Relations Between Local and Global Periodicity
44. **Mika Hirvensalo**, Studies on Boolean Functions Related to Quantum Computing
45. **Pentti Virtanen**, Measuring and Improving Component-Based Software Development
46. **Adekunle Okunoye**, Knowledge Management and Global Diversity – A Framework to Support Organisations in Developing Countries
47. **Antonina Kloptchenko**, Text Mining Based on the Prototype Matching Method
48. **Juha Kivijärvi**, Optimization Methods for Clustering
49. **Rimvydas Rukšėnas**, Formal Development of Concurrent Components
50. **Dirk Nowotka**, Periodicity and Unbordered Factors of Words
51. **Attila Gyenesei**, Discovering Frequent Fuzzy Patterns in Relations of Quantitative Attributes
52. **Petteri Kaitovaara**, Packaging of IT Services – Conceptual and Empirical Studies
53. **Petri Rosendahl**, Niho Type Cross-Correlation Functions and Related Equations
54. **Péter Majlender**, A Normative Approach to Possibility Theory and Soft Decision Support
55. **Seppo Virtanen**, A Framework for Rapid Design and Evaluation of Protocol Processors
56. **Tomas Eklund**, The Self-Organizing Map in Financial Benchmarking
57. **Mikael Collan**, Giga-Investments: Modelling the Valuation of Very Large Industrial Real Investments
58. **Dag Björklund**, A Kernel Language for Unified Code Synthesis
59. **Shengnan Han**, Understanding User Adoption of Mobile Technology: Focusing on Physicians in Finland
60. **Irina Georgescu**, Rational Choice and Revealed Preference: A Fuzzy Approach
61. **Ping Yan**, Limit Cycles for Generalized Liénard-Type and Lotka-Volterra Systems
62. **Joonas Lehtinen**, Coding of Wavelet-Transformed Images
63. **Tommi Meskanen,** On the NTRU Cryptosystem
64. **Saeed Salehi**, Varieties of Tree Languages
65. **Jukka Arvo**, Efficient Algorithms for Hardware-Accelerated Shadow Computation
66. **Mika Hirvikorpi**, On the Tactical Level Production Planning in Flexible Manufacturing Systems
67. **Adrian Costea**, Computational Intelligence Methods for Quantitative Data Mining
68. **Cristina Seceleanu**, A Methodology for Constructing Correct Reactive Systems
69. **Luigia Petre**, Modeling with Action Systems
70. **Lu Yan**, Systematic Design of Ubiquitous Systems
71. **Mehran Gomari**, On the Generalization Ability of Bayesian Neural Networks
72. **Ville Harkke**, Knowledge Freedom for Medical Professionals – An Evaluation Study of a Mobile Information System for Physicians in Finland
73. **Marius Cosmin Codrea**, Pattern Analysis of Chlorophyll Fluorescence Signals
74. **Aiying Rong**, Cogeneration Planning Under the Deregulated Power Market and Emissions Trading Scheme
75. **Chihab BenMoussa**, Supporting the Sales Force through Mobile Information and Communication Technologies: Focusing on the Pharmaceutical Sales Force
76. **Jussi Salmi**, Improving Data Analysis in Proteomics
77. **Orieta Celiku**, Mechanized Reasoning for Dually-Nondeterministic and Probabilistic Programs
78. **Kaj-Mikael Björk**, Supply Chain Efficiency with Some Forest Industry Improvements
79. **Viorel Preoteasa**, Program Variables – The Core of Mechanical Reasoning about Imperative Programs
80. **Jonne Poikonen**, Absolute Value Extraction and Order Statistic Filtering for a Mixed-Mode Array Image Processor
81. **Luka Milovanov**, Agile Software Development in an Academic Environment
82. **Francisco Augusto Alcaraz Garcia**, Real Options, Default Risk and Soft Applications
83. **Kai K. Kimppa**, Problems with the Justification of Intellectual Property Rights in Relation to Software and Other Digitally Distributable Media
84. **Dragoş Truşcan**, Model Driven Development of Programmable Architectures
85. **Eugen Czeizler**, The Inverse Neighborhood Problem and Applications of Welch Sets in Automata Theory

86. **Sanna Ranto**, Identifying and Locating-Dominating Codes in Binary Hamming Spaces
87. **Tuomas Hakkarainen**, On the Computation of the Class Numbers of Real Abelian Fields
88. **Elena Czeizler**, Intricacies of Word Equations
89. **Marcus Alanen**, A Metamodeling Framework for Software Engineering
90. **Filip Ginter**, Towards Information Extraction in the Biomedical Domain: Methods and Resources
91. **Jarkko Paavola**, Signature Ensembles and Receiver Structures for Oversaturated Synchronous DS-CDMA Systems
92. **Arho Virkki**, The Human Respiratory System: Modelling, Analysis and Control
93. **Olli Luoma**, Efficient Methods for Storing and Querying XML Data with Relational Databases
94. **Dubravka Ilić**, Formal Reasoning about Dependability in Model-Driven Development
95. **Kim Solin**, Abstract Algebra of Program Refinement
96. **Tomi Westerlund**, Time Aware Modelling and Analysis of Systems-on-Chip
97. **Kalle Saari**, On the Frequency and Periodicity of Infinite Words
98. **Tomi Kärki**, Similarity Relations on Words: Relational Codes and Periods
99. **Markus M. Mäkelä**, Essays on Software Product Development: A Strategic Management Viewpoint
100. **Roope Vehkalahti**, Class Field Theoretic Methods in the Design of Lattice Signal Constellations
101. **Anne-Maria Ernvall-Hytönen**, On Short Exponential Sums Involving Fourier Coefficients of Holomorphic Cusp Forms
102. **Chang Li**, Parallelism and Complexity in Gene Assembly
103. **Tapio Pahikkala**, New Kernel Functions and Learning Methods for Text and Data Mining
104. **Denis Shestakov**, Search Interfaces on the Web: Querying and Characterizing
105. **Sampo Pyysalo**, A Dependency Parsing Approach to Biomedical Text Mining
106. **Anna Sell**, Mobile Digital Calendars in Knowledge Work
107. **Dorina Marghescu**, Evaluating Multidimensional Visualization Techniques in Data Mining Tasks
108. **Tero Säntti**, A Co-Processor Approach for Efficient Java Execution in Embedded Systems
109. **Kari Salonen**, Setup Optimization in High-Mix Surface Mount PCB Assembly
110. **Pontus Boström**, Formal Design and Verification of Systems Using Domain-Specific Languages
111. **Camilla J. Hollanti**, Order-Theoretic Mehtods for Space-Time Coding: Symmetric and Asymmetric Designs
112. **Heidi Himmanen**, On Transmission System Design for Wireless Broadcasting
113. **Sébastien Lafond**, Simulation of Embedded Systems for Energy Consumption Estimation
114. **Evgeni Tsivtsivadze**, Learning Preferences with Kernel-Based Methods
115. **Petri Salmela**, On Commutation and Conjugacy of Rational Languages and the Fixed Point Method
116. **Siamak Taati**, Conservation Laws in Cellular Automata
117. **Vladimir Rogojin**, Gene Assembly in Stichotrichous Ciliates: Elementary Operations, Parallelism and Computation
118. **Alexey Dudkov**, Chip and Signature Interleaving in DS CDMA Systems
119. **Janne Savela**, Role of Selected Spectral Attributes in the Perception of Synthetic Vowels
120. **Kristian Nybom**, Low-Density Parity-Check Codes for Wireless Datacast Networks
121. **Johanna Tuominen**, Formal Power Analysis of Systems-on-Chip
122. **Teijo Lehtonen**, On Fault Tolerance Methods for Networks-on-Chip
123. **Eeva Suvitie**, On Inner Products Involving Holomorphic Cusp Forms and Maass Forms
124. **Linda Mannila**, Teaching Mathematics and Programming – New Approaches with Empirical Evaluation
125. **Hanna Suominen**, Machine Learning and Clinical Text: Supporting Health Information Flow
126. **Tuomo Saarni**, Segmental Durations of Speech
127. **Johannes Eriksson**, Tool-Supported Invariant-Based Programming

128. **Tero Jokela**, Design and Analysis of Forward Error Control Coding and Signaling for Guaranteeing QoS in Wireless Broadcast Systems
129. **Ville Lukkarila**, On Undecidable Dynamical Properties of Reversible One-Dimensional Cellular Automata
130. **Qaisar Ahmad Malik**, Combining Model-Based Testing and Stepwise Formal Development
131. **Mikko-Jussi Laakso**, Promoting Programming Learning: Engagement, Automatic Assessment with Immediate Feedback in Visualizations
132. **Riikka Vuokko**, A Practice Perspective on Organizational Implementation of Information Technology
133. **Jeanette Heidenberg**, Towards Increased Productivity and Quality in Software Development Using Agile, Lean and Collaborative Approaches
134. **Yong Liu**, Solving the Puzzle of Mobile Learning Adoption
135. **Stina Ojala**, Towards an Integrative Information Society: Studies on Individuality in Speech and Sign
136. **Matteo Brunelli**, Some Advances in Mathematical Models for Preference Relations
137. **Ville Junnila**, On Identifying and Locating-Dominating Codes
138. **Andrzej Mizera**, Methods for Construction and Analysis of Computational Models in Systems Biology. Applications to the Modelling of the Heat Shock Response and the Self-Assembly of Intermediate Filaments.
139. **Csaba Ráduly-Baka**, Algorithmic Solutions for Combinatorial Problems in Resource Management of Manufacturing Environments
140. **Jari Kyngäs**, Solving Challenging Real-World Scheduling Problems
141. **Arho Suominen**, Notes on Emerging Technologies
142. **József Mezei**, A Quantitative View on Fuzzy Numbers
143. **Marta Olszewska**, On the Impact of Rigorous Approaches on the Quality of Development
144. **Antti Airola**, Kernel-Based Ranking: Methods for Learning and Performace Estimation
145. **Aleksi Saarela**, Word Equations and Related Topics: Independence, Decidability and Characterizations
146. **Lasse Bergroth**, Kahden merkkijonon pisimmän yhteisen alijonon ongelma ja sen ratkaiseminen
147. **Thomas Canhao Xu**, Hardware/Software Co-Design for Multicore Architectures
148. **Tuomas Mäkilä**, Software Development Process Modeling – Developers Perspective to Contemporary Modeling Techniques
149. **Shahrokh Nikou**, Opening the Black-Box of IT Artifacts: Looking into Mobile Service Characteristics and Individual Perception
150. **Alessandro Buoni**, Fraud Detection in the Banking Sector: A Multi-Agent Approach
151. **Mats Neovius**, Trustworthy Context Dependency in Ubiquitous Systems
152. **Fredrik Degerlund**, Scheduling of Guarded Command Based Models
153. **Amir-Mohammad Rahmani-Sane**, Exploration and Design of Power-Efficient Networked Many-Core Systems
154. **Ville Rantala**, On Dynamic Monitoring Methods for Networks-on-Chip
155. **Mikko Pelto**, On Identifying and Locating-Dominating Codes in the Infinite King Grid
156. **Anton Tarasyuk**, Formal Development and Quantitative Verification of Dependable Systems
157. **Muhammad Mohsin Saleemi**, Towards Combining Interactive Mobile TV and Smart Spaces: Architectures, Tools and Application Development
158. **Tommi J. M. Lehtinen**, Numbers and Languages
159. **Peter Sarlin**, Mapping Financial Stability
160. **Alexander Wei Yin**, On Energy Efficient Computing Platforms
161. **Mikołaj Olszewski**, Scaling Up Stepwise Feature Introduction to Construction of Large Software Systems
162. **Maryam Kamali**, Reusable Formal Architectures for Networked Systems
163. **Zhiyuan Yao**, Visual Customer Segmentation and Behavior Analysis – A SOM-Based Approach
164. **Timo Jolivet**, Combinatorics of Pisot Substitutions
165. **Rajeev Kumar Kanth**, Analysis and Life Cycle Assessment of Printed Antennas for Sustainable Wireless Systems
166. **Khalid Latif**, Design Space Exploration for MPSoC Architectures

167. **Bo Yang**, Towards Optimal Application Mapping for Energy-Efficient Many-Core Platforms
168. **Ali Hanzala Khan**, Consistency of UML Based Designs Using Ontology Reasoners
169. **Sonja Leskinen**, m-Equine: IS Support for the Horse Industry
170. **Fareed Ahmed Jokhio**, Video Transcoding in a Distributed Cloud Computing Environment
171. **Moazzam Fareed Niazi**, A Model-Based Development and Verification Framework for Distributed System-on-Chip Architecture
172. **Mari Huova**, Combinatorics on Words: New Aspects on Avoidability, Defect Effect, Equations and Palindromes
173. **Ville Timonen**, Scalable Algorithms for Height Field Illumination
174. **Henri Korvela**, Virtual Communities – A Virtual Treasure Trove for End-User Developers
175. **Kameswar Rao Vaddina**, Thermal-Aware Networked Many-Core Systems
176. **Janne Lahtiranta**, New and Emerging Challenges of the ICT-Mediated Health and Well-Being Services
177. **Irum Rauf**, Design and Validation of Stateful Composite RESTful Web Services
178. **Jari Björne**, Biomedical Event Extraction with Machine Learning
179. **Katri Haverinen**, Natural Language Processing Resources for Finnish: Corpus Development in the General and Clinical Domains
180. **Ville Salo**, Subshifts with Simple Cellular Automata
181. **Johan Ersfolk**, Scheduling Dynamic Dataflow Graphs
182. **Hongyan Liu**, On Advancing Business Intelligence in the Electricity Retail Market
183. **Adnan Ashraf**, Cost-Efficient Virtual Machine Management: Provisioning, Admission Control, and Consolidation
184. **Muhammad Nazrul Islam**, Design and Evaluation of Web Interface Signs to Improve Web Usability: A Semiotic Framework
185. **Johannes Tuikkala**, Algorithmic Techniques in Gene Expression Processing: From Imputation to Visualization
186. **Natalia Díaz Rodríguez**, Semantic and Fuzzy Modelling for Human Behaviour Recognition in Smart Spaces. A Case Study on Ambient Assisted Living
187. **Mikko Pänkäälä**, Potential and Challenges of Analog Reconfigurable Computation in Modern and Future CMOS
188. **Sami Hyrynsalmi**, Letters from the War of Ecosystems – An Analysis of Independent Software Vendors in Mobile Application Marketplaces
189. **Seppo Pulkkinen**, Efficient Optimization Algorithms for Nonlinear Data Analysis
190. **Sami Pyöttiälä**, Optimization and Measuring Techniques for Collect-and-Place Machines in Printed Circuit Board Industry
191. **Syed Mohammad Asad Hassan Jafri**, Virtual Runtime Application Partitions for Resource Management in Massively Parallel Architectures
192. **Toni Ernvall**, On Distributed Storage Codes
193. **Yuliya Prokhorova**, Rigorous Development of Safety-Critical Systems
194. **Olli Lahdenoja**, Local Binary Patterns in Focal-Plane Processing – Analysis and Applications
195. **Annika H. Holmbom**, Visual Analytics for Behavioral and Niche Market Segmentation
196. **Sergey Ostroumov**, Agent-Based Management System for Many-Core Platforms: Rigorous Design and Efficient Implementation
197. **Espen Suenson**, How Computer Programmers Work – Understanding Software Development in Practise
198. **Tuomas Poikela**, Readout Architectures for Hybrid Pixel Detector Readout Chips
199. **Bogdan Iancu**, Quantitative Refinement of Reaction-Based Biomodels
200. **Ilkka Törmä**, Structural and Computational Existence Results for Multidimensional Subshifts
201. **Sebastian Okser**, Scalable Feature Selection Applications for Genome-Wide Association Studies of Complex Diseases
202. **Fredrik Abbors**, Model-Based Testing of Software Systems: Functionality and Performance
203. **Inna Pereverzeva**, Formal Development of Resilient Distributed Systems
204. **Mikhail Barash**, Defining Contexts in Context-Free Grammars
205. **Sepinoud Azimi**, Computational Models for and from Biology: Simple Gene Assembly and Reaction Systems
206. **Petter Sandvik**, Formal Modelling for Digital Media Distribution

**207.** **Jongyun Moon**, Hydrogen Sensor Application of Anodic Titanium Oxide Nanostructures
**208.** **Simon Holmbacka**, Energy Aware Software for Many-Core Systems
**209.** **Charalampos Zinoviadis**, Hierarchy and Expansiveness in Two-Dimensional Subshifts of Finite Type
**210.** **Mika Murtojärvi**, Efficient Algorithms for Coastal Geographic Problems
**211.** **Sami Mäkelä**, Cohesion Metrics for Improving Software Quality
**212.** **Eyal Eshet**, Examining Human-Centered Design Practice in the Mobile Apps Era
**213.** **Jetro Vesti**, Rich Words and Balanced Words
**214.** **Jarkko Peltomäki**, Privileged Words and Sturmian Words
**215.** **Fahimeh Farahnakian**, Energy and Performance Management of Virtual Machines: Provisioning, Placement and Consolidation
**216.** **Diana-Elena Gratie**, Refinement of Biomodels Using Petri Nets
**217.** **Harri Merisaari**, Algorithmic Analysis Techniques for Molecular Imaging
**218.** **Stefan Grönroos**, Efficient and Low-Cost Software Defined Radio on Commodity Hardware
**219.** **Noora Nieminen**, Garbling Schemes and Applications
**220.** **Ville Taajamaa**, O-CDIO: Engineering Education Framework with Embedded Design Thinking Methods
**221.** **Johannes Holvitie**, Technical Debt in Software Development – Examining Premises and Overcoming Implementation for Efficient Management
**222.** **Tewodros Deneke**, Proactive Management of Video Transcoding Services
**223.** **Kashif Javed**, Model-Driven Development and Verification of Fault Tolerant Systems
**224.** **Pekka Naula**, Sparse Predictive Modeling – A Cost-Effective Perspective
**225.** **Antti Hakkala**, On Security and Privacy for Networked Information Society – Observations and Solutions for Security Engineering and Trust Building in Advanced Societal Processes
**226.** **Anne-Maarit Majanoja**, Selective Outsourcing in Global IT Services – Operational Level Challenges and Opportunities
**227.** **Samuel Rönnqvist**, Knowledge-Lean Text Mining
**228.** **Mohammad-Hashem Hahgbayan**, Energy-Efficient and Reliable Computing in Dark Silicon Era
**229.** **Charmi Panchal**, Qualitative Methods for Modeling Biochemical Systems and Datasets: The Logicome and the Reaction Systems Approaches
**230.** **Erkki Kaila**, Utilizing Educational Technology in Computer Science and Programming Courses: Theory and Practice
**231.** **Fredrik Robertsén**, The Lattice Boltzmann Method, a Petaflop and Beyond
**232.** **Jonne Pohjankukka**, Machine Learning Approaches for Natural Resource Data
**233.** **Paavo Nevalainen**, Geometric Data Understanding: Deriving Case-Specific Features
**234.** **Michal Szabados**, An Algebraic Approach to Nivat's Conjecture
**235.** **Tuan Nguyen Gia**, Design for Energy-Efficient and Reliable Fog-Assisted Healthcare IoT Systems
**236.** **Anil Kanduri**, Adaptive Knobs for Resource Efficient Computing

Turku Centre *for* Computer Science

http://www. tucs.fi

tucs@abo.fi

**University of Turku**
*Faculty of Science and Engineering*
- Department of Future Technologies
- Department of Mathematics and Statistics

*Turku School of Economics*
- Institute of Information Systems Science

**Åbo Akademi University**
*Faculty of Science and Engineering*
- Computer Engineering
- Computer Science

*Faculty of Social Sciences, Business and Economics*
- Information Systems

ISBN 978-952-12-3775-1
ISSN 1239-1883

Anil Kanduri

# Adaptive Knobs for Resource Efficient Computing

Anil Kanduri

Adaptive Knobs for Resource Efficient Computing

Anil Kanduri

Adaptive Knobs for Resource Efficient Computing