TUCS

Thomas Canhao Xu – 许粲昊

Hardware/Software Co-Design for Multicore Architectures

Turku Centre *for* Computer Science

TUCS Dissertations
No 147, September 2012

Hardware/Software Co-design for Multicore Architectures

Thomas Canhao Xu

To be presented, with the permission of the Faculty of Mathematics and Natural Science of the University of Turku, for public criticism in Auditorium Beta on September 12th, 2012, at 12 noon.

University of Turku
Department of Information Technology
20014 Turku, Finland

2012

Supervisors

Prof. Hannu Tenhunen
Department of Information Technology
University of Turku
20014 Turku
Finland

Adj. Prof. Pasi Liljeberg
Department of Information Technology
University of Turku
20014 Turku
Finland

Reviewers

Prof. Dr. Martin Radetzki
Institut für Technische Informatik
Universität Stuttgart
Pfaffenwaldring 5b, D-70569 Stuttgart
Germany

Principal Scientist Martti Forsell
VTT Technical Research Center of Finland
PL 1100
90571 Oulu
Finland

Opponent

Prof. Per Gunnar Kjeldsberg
Department of Electronics and Telecommunications
Norwegian University of Science and Technology
O.S. Bragstads Plass 2A, NO-7491 Trondheim
Norway

ISBN 978-952-12-2772-1
ISSN 1239-1883

Abstract

The integration of multiple cores on a single chip leads to the concept of chip multiprocessor. We have already witnessed multicore processors emerging with increasing number of cores and complex on-chip interconnect in the past few years. Network-on-Chip (NoC) architecture was proposed as a promising solution for future multicore processors with hundreds or even thousands of cores. In this regard, hardware/software co-design of NoC based multicore architectures are presented in this dissertation.

Three dimensional (3D) integration has the potential to increase device density, providing higher efficiency compared with two dimensional (2D) integration. Moreover, the combination of 3D integration and NoC architecture provides the benefits of both. Here, 2D/3D multicore processors with integrated and split core/cache architectures are analyzed based on non uniform cache architecture. In addition, a 3D multicore design with on-chip dynamic random access memories is also introduced to alleviate the memory bandwidth wall.

There are many hardware resources in a multicore processor, for example caches and memory controllers. If the resources, e.g. memory controllers, are attached to all nodes, the utilization of resources can be low, and therefore leading to a poor system efficiency. One solution is to distribute a limited number of resources. However, in this case, multiple requesters have to share a resource, leading to possible traffic contention. To alleviate the problem of performance degradation by reduced amount of resources, intelligent placement of resources for a mesh-based on-chip networks is introduced. Three hardware resources are used as case studies, including through silicon vias, memory controllers and cores/caches.

Two operating system scheduling algorithms are presented in order to improve performance and efficiency of multicore systems. We propose a minimal average access time scheduler to reduce on-chip communication latencies for 2D multicore processors. A greedy heuristic approximation scheduling algorithm is presented for resource constrained 3D multicore processors. Current parallel applications are designed and optimized for conventional bus or crossbar based multicore architectures. Without the collaboration of software, the processing ability of NoC based multicore systems can be limited. Three applications are analyzed for the NoC platform, including H.264, FFT and two hierarchical N-Body methods. Optimization suggestions are given both on hardware and software.

Dedicated to my father Xu Zejin
who gave so much hope and support for my education but
unfortunately did not live long enough to share this moment

献给我的父亲许泽金
他在教育上给予我极高的期望和支持但是
很遗憾没能活着见证这一刻

Acknowledgements

The research presented in this dissertation was carried out ar the Department of Information Technology, University of Turku and Turku Center for Computer Science (TUCS), during September 2008 and August 2012. I would like to express my gratitude to many people and organization, this dissertation would not be possible without their support.

First and foremost, I would like to gratitude my supervisors, Professor Hannu Tenhunen and Adjunct Professor Pasi Liljeberg, for their support and guidance to me. Throughout these years at Turku, they have been an incredible source of encouragement, wisdom and inspiration. Their mentorship and unique insights were immense assets to my research and life. I am truly fortunate to have had the opportunity to learn from and work with them.

I would like to thank Professor Martin Radetzki from Universität Stuttgart and Principal Scientist Martti Forsell from VTT for the detailed reviews and constructive comments on this dissertation.

Most importantly, none of this would have been possible without the love and patience of my family. I am blessed to have parents that are instrumental to my successes. They were my first teachers and role models, and sacrificed so much to support my education. I am indebted to my father for his love. Although he is no longer with us, he is forever remembered. I am sure he shares our joy and happiness in the heaven. As a typical perfect Chinese mother, I have no suitable word that can fully describe her everlasting love and support to me. I would like to express my deepest gratitude and love for their dedication and the many years of support that provided the foundation for this work.

I am truly grateful for all the support from my wife, for her understanding and love during the past few years. I remember her constant encouragement when I encountered difficulties and I remember, most of all, her delicious dishes. Her support and encouragement was in the end what made this dissertation possible. Meanwhile, I have enjoyed many delightful times with my daughter, although now she is too young to understand my appreciation.

Finally, I would like to express my sincerest gratitude for the financial support from Academy of Finland, Turku Center for Computer Science, Nokia Foundation and Finnish Foundation for Technology Promotion (Tekniikan edistämissäätiö).

致谢

本文的研究是二零零八年九月至二零一二年八月在图尔库大学信息科技学院以及图尔库计算机中心完成的。我想在此对一些人和组织表示谢意，没有他们的帮助本文不可能完成。

首先，我要感谢我的导师，Hannu Tenhunen教授和Pasi Liljeberg副教授，感谢他们的支持和指导。在图尔库的这些年，他们是我创新和智慧的源泉。他们的教导和独特的眼光是我人生无尽的宝藏。我感到非常幸运能有机会向他们学习，和他们共事。

我感谢斯图加特大学的Martin Radetzki教授和VTT的基础科学家Martti Forsell，感谢他们对本文的详细审阅和富有建设性的意见和建议。

最重要的是，如果没有家庭的爱和耐心支持，本文是不可能完成的。我很幸运拥有这样的父母，他们在我成功的道路上功不可没。他们是我的良师益友、道德榜样，并且牺牲了很多来支持我的教育。我非常感激我父亲的爱，虽然他已经不在这个世界，他将永远被记住。我肯定他会在天国与我们共享这一美好的时刻。作为一个典型的中国母亲，我找不到合适的词语来完全描述她对我持久的爱和支持。我的父母多年来付出许多，这奠定了本文的基础，我想对他们表达我最深的感谢和爱。

我真心感谢我的妻子在这些年对我的爱和理解。我记得她在我遇到困难时候不断的支持，还有她厨艺方面的天赋。她的支持和鼓励使得本文最终能够顺利完成。同时，我也享受了很多跟我女儿在一起的美好时光，虽然她现在太小，还没法懂得我这份心意。

最后，我要对以下机构的资金支持表达我最诚挚的感谢：芬兰科学院，图尔库计算机中心，诺基亚基金会，以及芬兰技术促进基金会。

List of original publications

The work presented in this dissertation is based or partially based on the following publications.

Journals:

1. Thomas Canhao Xu, Tapio Pahikkala, Pasi Liljeberg, Juha Plosila, and Hannu Tenhunen;
   Optimized Network-on-Chip Architectures for Data Parallel Fast Fourier Transform;
   Springer Telecommunication System, accepted.

2. Tapio Pahikkala, Antti Airola, Thomas Canhao Xu, Pasi Liljeberg, Tapio Salakoski, and Hannu Tenhunen;
   Parallelized Online Regularized Least-Squares for Adaptive Embedded Systems;
   International Journal of Embedded and Real-Time Communication Systems (IJERTCS), Volume 3, Issue 2, April-June 2012, Pages 73-91;
   Published by IGI Global, DOI: 10.4018/jertcs.2012040104.

3. Thomas Canhao Xu, Pasi Liljeberg, and Hannu Tenhunen;
   An Optimized Network-on-Chip Design for Data Parallel FFT;
   Procedia Engineering, Volume 30, March 2012, Pages 313-318;
   Published by Elsevier, DOI: 10.1016/j.proeng.2012.01.866.

4. Thomas Canhao Xu, Pasi Liljeberg, and Hannu Tenhunen;
   Exploring DRAM Last Level Cache for 3D Network-on-Chip Architecture;
   Advanced Materials Research, Volumes 403-408, November 2011, Pages 4009-4018;
   Published by Trans Tech Publications,
   DOI: 10.4028/www.scientific.net/AMR.403-408.4009.

5. Thomas Canhao Xu, Alexander Wei Yin, Pasi Liljeberg, and Hannu Tenhunen;
   A Study of 3D Network-on-Chip Design for Data Parallel H.264 Coding;
   Microprocessors and Microsystems, Volume 35, Issue 7, October 2011, Pages 603-612;
   Published by Elsevier, DOI: 10.1016/j.micpro.2011.06.009.

6. Masoud Daneshtalab, Masoumeh Ebrahimi, Thomas Canhao Xu, Pasi Liljeberg, and Hannu Tenhunen;
A Generic Adaptive Path-based Routing Method for MPSoCs;
Journal of Systems Architecture, Volume 57, Issue 1, Special Issue On-Chip Parallel And Network-Based Systems, January 2011, Pages 109-120;
Published by Elsevier, DOI: 10.1016/j.sysarc.2010.08.002.

Springer LNCS series:

7. Thomas Canhao Xu, Pasi Liljeberg, and Hannu Tenhunen;
Study of Hierarchical N-Body Methods for Network-on-Chip Architectures;
In Proceedings of the 17th International Euro-Par Conference, Parallel Processing Workshops (Euro-Par), LNCS 7156/2012, pp.365-374, 29 August-02 September 2011, Bordeaux, France;
Published by Springer, DOI: 10.1007/978-3-642-29740-3_41.

8. Thomas Canhao Xu, Pasi Liljeberg, and Hannu Tenhunen;
A Greedy Heuristic Approximation Scheduling Algorithm for 3D Multicore Processors;
In Proceedings of the 17th International Euro-Par Conference, Parallel Processing Workshops (Euro-Par), LNCS 7155/2012, pp.281-291, 29 August-02 September 2011, Bordeaux, France;
Published by Springer, DOI: 10.1007/978-3-642-29737-3_32.

9. Thomas Canhao Xu, Pasi Liljeberg, and Hannu Tenhunen;
A Minimal Average Accessing Time Scheduler for Multicore Processors;
In Proceedings of the 11th International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP), LNCS 7017/2011, pp.287-299, 24-26 October 2011, Melbourne, Australia;
Published by Springer, DOI: 10.1007/978-3-642-24669-2_28.

Conference and workshop proceedings:

10. Thomas Canhao Xu, Tapio Pahikkala, Antti Airola, Pasi Liljeberg, Juha Plosila, Tapio Salakoski, and Hannu Tenhunen;
Implementation and Analysis of Block Dense Matrix Decomposition on Network-on-Chips;
In Proceedings of the 14th IEEE International Conference on High Performance Computing and Communications (HPCC), accepted.

11. Thomas Canhao Xu, Pasi Liljeberg, Juha Plosila, and Hannu Tenhunen;
Exploration of Heuristic Scheduling Algorithms for 3D Multicore Processors;
In Proceedings of the 15th International Workshop on Software and Compilers for Embedded Systems (SCOPES), pp.22-31, 15-16 May 2012, Schloss Rheinfels, St. Goar, Germany;
Published by ACM, DOI: 10.1145/2236576.2236579.

12. Thomas Canhao Xu, Pasi Liljeberg, and Hannu Tenhunen;
Explorations of Optimal Core and Cache Placements for Chip Multiprocessor;
In Proceedings of the 29th IEEE Norchip Conference (Norchip), pp.1-6, 14-15 November 2011, Lund, Sweden;
Published by IEEE, DOI: 10.1109/NORCHP.2011.6126728.

13. Thomas Canhao Xu, Pasi Liljeberg, and Hannu Tenhunen;
Optimal Memory Controller Placement for Chip Multiprocessor;
In Proceedings of the 9th IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES/ISSS), pp.217-226, 9-14 October 2011, Taipei, Taiwan;
Published by ACM, DOI: 10.1145/2039370.2039405.

14. Alexander Wei Yin, Thomas Canhao Xu, Bo Yang, Pasi Liljeberg, and Hannu Tenhunen;
Change Function of 2D/3D Network-on-Chip;
In Proceedings of the 11th IEEE International Conference on Computer and Information Technology (CIT), pp.181-188, 31 August-02 September 2011, Pafos, Cyprus;
Published by IEEE, DOI: 10.1109/CIT.2011.38.

15. Thomas Canhao Xu, Pasi Liljeberg, and Hannu Tenhunen;
Optimal Number and Placement of Through Silicon Vias in 3D Network-on-Chip;
In Proceedings of the 14th IEEE International Symposium on Design and Diagnostics of Electronic Circuits & Systems (DDECS), pp.105-110, 13-15 April 2011, Cottbus, Germany;
Published by IEEE, DOI: 10.1109/DDECS.2011.5783057.

16. Tapio Pahikkala, Antti Airola, Thomas Canhao Xu, Pasi Liljeberg, Tapio Salakoski, and Hannu Tenhunen;
A Parallel Online Regularized Least-Squares Machine Learning Algorithm for Future Multi-Core Processors;
In Proceedings of the 2011 International Conference on Pervasive and Embedded Computing and Communication Systems (PECCS), pp.590-599, 5-7 March 2011, Vilamoura, Algarve, Portugal;
Published by SciTePress.

17. Thomas Canhao Xu, Pasi Liljeberg, and Hannu Tenhunen;
Process Scheduling for Future Multicore Processors;
In Proceedings of the 5th International Workshop on Interconnection Network Architecture: On-Chip, Multi-Chip (INA-OCMC), pp.15-18, 24-26 January 2011, Heraklion, Crete, Greece;
Published by ACM, DOI: 10.1145/1930037.1930042.

18. Thomas Canhao Xu, Liang Guang, Alexander Wei Yin, Bo Yang, Pasi Liljeberg, and Hannu Tenhunen;

An Analysis of Designing 2D/3D Chip Multiprocessor with Different Cache Architecture;
In Proceedings of the 28th IEEE Norchip Conference (Norchip), pp.1-6, 15-16 November 2010, Tampere, Finland;
Published by IEEE, DOI: 10.1109/NORCHIP.2010.5669433.

19. Bo Yang, Liang Guang, Thomas Canhao Xu, Alexander Wei Yin, Tero Säntti, and Juha Plosila;
Multi-application Multi-step Mapping Method for Many-core Network-on-Chips;
In Proceedings of the 28th IEEE Norchip Conference (Norchip), pp.1-6, 15-16 November 2010, Tampere, Finland;
Published by IEEE, DOI: 10.1109/NORCHIP.2010.5669454.

20. Thomas Canhao Xu, Bo Yang, Alexander Wei Yin, Pasi Liljeberg, and Hannu Tenhunen;
3D Network-on-Chip with On-chip DRAM: An Empirical Analysis for Future Chip Multiprocessor;
In Proceedings of the 2010 International Conference on Computer, Electrical, and Systems Science, and Engineering (ICCESSE), pp.18-24, 27-29 October 2010, Paris, France;
Published by WASET.

21. Bo Yang, Liang Guang, Thomas Canhao Xu, Tero Säntti, and Juha Plosila;
Multi-application Mapping Algorithm for Network-on-Chip Platforms;
In Proceedings of the 26th IEEE Convention of Electrical and Electronics Engineers in Israel (IEEEI), pp.540-544, 17-20 November 2010, Eilat, Israel;
Published by IEEE, DOI: 10.1109/EEEI.2010.5662160.

22. Thomas Canhao Xu, Pasi Liljeberg, and Hannu Tenhunen;
A Study of Through Silicon Via Impact to 3D Network-on-Chip Design;
In Proceedings of the 2010 International Conference on Electronics and Information Engineering (ICEIE), pp.V1-333-V1-337, 1-3 August 2010, Kyoto, Japan;
Published by IEEE, DOI: 10.1109/ICEIE.2010.5559865.

23. Bo Yang, Thomas Canhao Xu, Tero Säntti, and Juha Plosila;
Tree-model Based Mapping for Energy-efficient and Low-latency Network-on-Chip;
In Proceedings of the 13th IEEE International Symposium on Design and Diagnostics of Electronic Circuits & Systems (DDECS), pp.189-192, 14-16 April 2010, Vienna, Austria;
Published by IEEE, DOI: 10.1109/DDECS.2010.5491789.

24. Thomas Canhao Xu, Alexander Wei Yin, Pasi Liljeberg, and Hannu Tenhunen;

Operating System Processor Scheduler Design for Future Chip Multiprocessor;
In Proceedings of the 23rd International Conference on Architecture of Computing Systems (ARCS), pp.1-7, 22-23 February 2010, Hannover, Germany;
Published by VDE VERLAG, DOI: 10.1145/1930037.1930042.

25. Thomas Canhao Xu, Alexander Wei Yin, Pasi Liljeberg, and Hannu Tenhunen;
A Study of 3D Network-on-Chip Design for Data Parallel H.264 Coding;
In Proceedings of the 27th IEEE Norchip Conference (Norchip), pp.1-6, 16-17 November 2009, Trondheim, Norway;
Published by IEEE, DOI: 10.1109/NORCHP.2009.5397851.

26. Alexander Wei Yin, Thomas Canhao Xu, Pasi Liljeberg, and Hannu Tenhunen;
Explorations of Honeycomb Topologies for Network-on-Chip;
In Proceedings of the 6th IFIP International Conference on Network and Parallel Computing (NPC), pp.73-79, 19-21 October 2009, Gold Coast, Australia;
Published by IEEE, DOI: 10.1109/NPC.2009.34.

Other publications:

27. Thomas Canhao Xu, Pasi Liljeberg, and Hannu Tenhunen;
Embedded Software System Architecture for MyGoogle-on-Chip;
TUCS Technical Report, No.922;
Published by TUCS.

Contents

1 Introduction **1**
1.1 Dissertation Organization . . . 5
1.2 Dissertation Contributions . . . 6

2 Background **9**
2.1 Network-on-Chip . . . 9
2.2 Three Dimensional Network-on-Chip . . . 11
2.2.1 3D NoC Router and Routing Algorithm . . . 12
2.3 Simulation Platform . . . 13
2.3.1 Example Configuration . . . 16
2.3.2 Example Workload . . . 16

3 Cache and Memory in 3D Network-on-Chip **19**
3.1 2D/3D Multiprocessor with Different Cache Architecture . . . 20
3.1.1 The Architecture of Cache . . . 20
3.1.2 Integrated and Split Core/Cache . . . 21
3.1.3 Thermal Issue in 3D NoCs . . . 22
3.1.4 Cache Access Latencies . . . 23
3.1.5 Wire Delay . . . 24
3.1.6 Experimental Evaluation . . . 26
3.2 3D Network-on-Chip with Dynamic Random Access Memory . . . 28
3.2.1 Modeling of the 3D NoC . . . 30
3.2.2 Analysis of the Impact of On-chip DRAM . . . 31
3.2.3 Experimental Evaluation . . . 35

4 Resource Placement **39**
4.1 Theoretical Background . . . 40
4.2 Divide and Conquer . . . 41
4.2.1 Placement for 16 Resources . . . 42
4.2.2 Placement for 8 Resources . . . 42
4.3 Placement for Through Silicon Via . . . 45
4.3.1 Cost Considerations . . . 46
4.3.2 Performance with Reduced Number of TSVs . . . 47
4.3.3 3D NoC Models . . . 48
4.3.4 Evaluation for 4×4 NoC with Five Layers . . . 49

4.3.5 Evaluation for 8×8 NoC with Two Layers 51
4.4 Placement for Memory Controller 52
4.4.1 Motivation 53
4.4.2 Analysis of Different Placements of Memory Controllers . . 55
4.4.3 Placement for the Memory Controllers 56
4.4.4 Generic Placement Method 60
4.4.5 Routing Algorithm 63
4.4.6 Address Mapping 63
4.4.7 Experimental Evaluation 64
4.5 Placement for Core and Cache 68
4.5.1 Modern Chip Multiprocessor Design 68
4.5.2 Area of Core and Cache 69
4.5.3 Ring and Network Connection 70
4.5.4 Placements of Caches 71
4.5.5 Discussion 73
4.5.6 Experimental Evaluation 75

5 Application Study for Network-on-Chip 79
5.1 H.264 80
5.1.1 The H.264 Coding Standard 81
5.1.2 The Parallelism of H.264 Encoding 83
5.1.3 Case Study 84
5.1.4 The Baseline 2D NoC Design for H.264 85
5.1.5 3D NoC Designs for H.264 86
5.1.6 Experimental Evaluation 88
5.2 Hierarchical N-Body 92
5.2.1 Modeling of the Network-on-Chip 93
5.2.2 The Hierarchical N-Body Methods 94
5.2.3 Experimental Evaluation 95
5.3 FFT 99
5.3.1 The FFT Algorithm 99
5.3.2 FFT Traffic Pattern 100
5.3.3 An Optimized Network-on-Chip Design 101
5.3.4 Experimental Evaluation 103

6 Operating System Scheduling 105
6.1 A Minimal Average Access Time Scheduler 106
6.1.1 Evaluation of Scheduler under NoC-based CMP 107
6.1.2 The Scheduler for NoC-based CMP Architecture 109
6.1.3 Discussion 116
6.1.4 Case Studies 118
6.1.5 Experimental Evaluation 119
6.2 A Greedy Heuristic Approximation Scheduling Algorithm 123
6.2.1 3D NoC with Through Silicon Via Constraints 123
6.2.2 The Scheduling Algorithm 125

6.2.3 Experimental Evaluation 128

7 Conclusion 131

7.1 Future Work 133

Appendix A Example Simulation Output and Post-processing 135

List of Figures

1.1 AMD and Intel multicore processors. 3

2.1 An example of 4x4 NoC with mesh topology. 10
2.2 The Tilera TILE multicore processor with 100 cores [102]. 11
2.3 Schematic diagram of a 3D NoC with one processor layer (Px) and one cache layer (Cx), layers are fully connected by TSVs. The heatsink is attached on top of the processor layer 12
2.4 Router model in GEMS/Simics. 15
2.5 Five stage pipeline. 16

3.1 UCA and NUCA for one processor (P) with eight cache banks (Cx). 21
3.2 Comparison of 2D NoC design with core and cache within the same tile (a), and 3D NoC with core and cache in different tiles and layers (b). 22
3.3 An example of remote cache access. 24
3.4 Total latency in cycles with increasing hop count. 26
3.5 Normalized average cache hit latency in cycles. 27
3.6 Normalized average link utilization in flits/cycle. 28
3.7 Normalized average network latency in cycles. 28
3.8 3D NoC with one processor layer (Px), one cache layer (Cx) and one memory layer, layers are fully connected by through silicon vias (TSVs, not shown in figure). 30
3.9 Compare of different processor and memory sub system organization. 31
3.10 2D off-chip memory and 3D on-chip memory organizations. 34
3.11 Normalized average link utilization with different configurations. . 36
3.12 Normalized executed cycles with different configurations. 37

4.1 A placement of 16 resources (grey nodes with P) in an 8×8 NoC, white nodes are requesters, number means minimal hops required for a requester to reach a resource. 41
4.2 Different placements of 4 resources in a 4×4 NoC. 42
4.3 An optimal placement of 16 resources (P) in an 8×8 NoC. 43
4.4 Different placements of 2 resources in a 4×4 NoC. 44
4.5 Two placements of 8 resources in an 8×8 mesh. 45
4.6 The TSV yield with increasing TSV numbers (x). 47
4.7 3D NoC with layers half and quarter connected by TSVs. 47

4.8 The comparison of average hop count among full, half and quarter connected layers. 48
4.9 Normalized average network latency with different number of pillars, for a 4×4 NoC with five layers. 50
4.10 Normalized power performance product with different number of pillars, for a 4×4 NoC with five layers. 50
4.11 Normalized average network latency with different number of pillars, for a 8×8 NoC with two layers. 52
4.12 Normalized power performance product with different number of pillars, for a 8×8 NoC with two layers. 52
4.13 Memory request rate for 16-core NoC running FFT. 54
4.14 Three examples of memory controller accesses of different placements of memory controllers, node 6 is colored as dark gray, nodes with light gray are connected with a memory controller. 55
4.15 Different placement of memory controllers, M means a memory controller, number means minimal hops required for a processor to reach a nearest memory controller. (a) to (e) are studied in [2] . 57
4.16 Hop counts to all memory controllers in three placements, dark gray node is a active PE sending memory requests, memory controllers are colored as light gray. 58
4.17 Three cases for memory message congestion. 59
4.18 Three placements of 8 memory controllers in an 8×8 mesh. 61
4.19 The memory request and response traffic for Placement 2 in Figure 4.18. 61
4.20 ANL-MReq for FFT, with different placements of memory controller. 65
4.21 Normalized ANL with different placements of memory controller. . 66
4.22 Normalized ALU with different placements of memory controller. . 67
4.23 Normalized PPP with different placements of memory controller. . 67
4.24 Intel Westmere-EX architecture, presented by Intel at Hot Chips 2010. 70
4.25 An example of 8-core CMP, with Bi-directional ring interconnect (a, Cx as Core x), and access latency (shown as hop count) for Core 0 (b) and Core 1 (c) of each cache slice. 71
4.26 Examples of worst case placement (a), best case placement (b), and access latency Core 1 (c) of each cache slice of the best case. . 72
4.27 Examples of worst case placement (a), and best case placement (b), for a 10-core 10-cache configuration. 74
4.28 Best case placements for a 12-core CMP (a), and 14-core CMP (b). 74
4.29 Normalized average network latency in cycles. 76
4.30 Normalized execution time. 77

5.1 H.264 encoding process. 82
5.2 H.264 decoding process. 82
5.3 Slice level parallelization of a frame. 84
5.4 Dependency in a video sequence, with 2 B frames. 84

5.5 Frame level parallelization of a video sequence. 84
5.6 2D NoC design based on SNUCA. 85
5.7 Two 2-layer 3D NoC designs with PEs distributed in layers (3D-DL) and on top (3D-top). Four pillars are shown for clarity. 86
5.8 The comparison of hop count among 2D, 3D-DL and 3D-top. . . . 87
5.9 Chip temperature of 2D and two 3D configurations. 87
5.10 Normalized average network latency with different chip and video configurations. 90
5.11 Normalized execution time with different chip and video configurations. 90
5.12 Normalized power consumption with different chip and video configurations. 91
5.13 Normalized average network latency with different number of pillars. 91
5.14 Normalized performance power product with different number of pillars. 92
5.15 An 8×8 mesh-based NoC with memory controllers attached to up and down sides. 94
5.16 Network request rate for 64-core NoC running Barnes. 96
5.17 Network request rate for 64-core NoC running FMM. 97
5.18 Performance for Barnes and FMM. 98
5.19 Network request rate for 16-core NoC running FFT. The time is segmented in 216K-cycle/percentage. 100
5.20 A diagram showing NoC model used here. 102
5.21 Normalized average network latency with different number of pillars. 104

6.1 Node traffic distribution (a) and memory controller traffic distribution (b) over FFT. 108
6.2 Node traffic distribution (a) and memory controller traffic distribution (b) over SPECjbb. 109
6.3 A 8×8 mesh-based NoC with 16 memory controllers attached to top and bottom of the design. 110
6.4 Two core allocation schemes for 15 threads. 111
6.5 Different core/memory allocation schemes, arrows are memory accesses. 112
6.6 Comparison of different core/memory allocation schemes. 113
6.7 A fragmented situation with 36 cores occupied (gray), and 28 cores free (white). 116
6.8 The scheduling system between CMP, OS and application. 117
6.9 LU decomposition algorithm with blocking. 118
6.10 Normalized performance metrics with different allocation strategies, for FFT. 120
6.11 Normalized ANL, ALU, ET and CHL with different allocation strategies, for FFT. 121
6.12 Normalized ANL, ALU, ET and CHL with different allocation strategies, for LU. 122

6.13 Normalized ANL, ALU, ET and CHL with different allocation strategies, for H.264. 122
6.14 Gray nodes are attached to a pillar. 124
6.15 The node selection steps for the algorithm. 127
6.16 The execution of our algorithm, starting from node 19 and selected 16 nodes. 128
6.17 Performance for FFT and Radix. 129

List of Tables

2.1 System configuration parameters 17

3.1 Comparison of wire characteristics 25
3.2 Processor and memory bandwidth for one channel 29
3.3 Highest network latency for a NoC node 32
3.4 Comparison of processors with memory controller and memory channel 34
3.5 Memory sub system configurations for different processors 35

4.1 Comparison of computation complexity of exhaustive enumeration and "divide and conquer". 45
4.2 Comparison of processors with memory controllers 56
4.3 Characteristics of different memory controller (MC) placements. AHC/1MC and AHC/All-MC are shown as 1MC and All-MC respectively. The Adjacent Memory Controllers (AMC) are shown in $Number\ of\ Sets \times Number\ of\ nodes\ in\ each\ Sets$. 60
4.4 Comparison of mesh size and placement possibilities, with $2\sqrt{n}$ memory controllers 62
4.5 Comparison of computation complexity of exhaustive enumeration and "divide and conquer", for several memory controller placements. 63
4.6 Comparison of different Chip Multiprocessors. LLC for Last Level Cache. 69
4.7 Comparison of AHP_{All} for different interconnect and placement . . 75

5.1 Input video sequences 89
5.2 Time distribution and scalability of the Barnes-Hut Method 95
5.3 Time distribution and scalability of the Fast Multipole Method . . 97
5.4 Top Point-to-Point traffics. 101

6.1 ACTs and AMTs for different allocation strategies 114
6.2 Allocation possibilities for an 8×8 mesh 115

Chapter 1

Introduction

The concept of multiprocessor has existed for several decades. Depending on the memory organization, multiprocessor systems can be categorized in two groups: shared memory and distributed memory. Distributed memory systems are more complex to design and implement, therefore Symmetric shared memory Multiprocessor Systems (SMP) are still used commonly. In SMP systems, two or more processors are connected to a single shared main memory and the memory has a symmetric relationship to all processors. Earlier SMP systems often rely on multiple-processor/multiple-socket, that is, each processor is mounted to a socket on a motherboard. To improve the performance of processors, microelectronic engineers developed smaller and faster transistors and gates, architectural engineers developed novel design methods, e.g. pipeline, superscalar, out-of-order execution, instruction level parallelism and so on. However, the constraints of chip clock frequency, power consumption and heat dissipation have pushed the chip designers to integrate multiple cores rather than to improve single-core performance. The integration of multiple cores on a single processor chip leads to the concept of Chip Multiprocessor (CMP). This is especially important for embedded devices, since the power consumption and heat dissipation are the most important factors in designing these devices. We have already witnessed CMPs emerging with increasing number of cores and complex on-chip interconnect in the past few years.

Portable embedded devices are very popular in the market of consumer electronics. These devices include mobile phones, tablets, MP3/MP4 players, Global Positioning System (GPS) automotive assistants, digital cameras, camcorders and so on. These devices are capable of processing geometry, audio and video data, capturing image and video, and output high-definition signals. More processing power is needed in these devices. For example, the quality of images and videos are growing constantly, from Color Graphics Adapter (CGA, 320×200) to High Definition (HD, 1280×720), Full HD (FHD, 1920×1080) and even Quad FHD (QFHD, 3840×2160). High-definition videos that have HD or even FHD resolution can have an uncompressed data-rate of 210 to 932 Mbps, and a compressed data-rate of 25 to 100 Mbps. Other features are added to these devices as well, for example, face recognition, machine learning, digital zooming, digital image/video stabilization and other realtime post-processing of multimedia contents. These tasks are highly parallel, and can benefit from multicore processing in terms of both performance and efficiency. In recent years, the processors of smart phones and tablet computers are emerging from single-core to dual-core or even quad-core. For example, the NVIDIA Tegra System-on-Chip (SoC) series CMP [1] are designed for embedded devices [78]. The newest Tegra 3 SoC contains five cores, including a quad-core ARM Cortex A9 and a companion core for low-power processing. The processor operates at 1.4 GHz for single-core, and 1.3 GHz for quad-core. There's also a twelve-core Graphics Processing Unit (GPU) on Tegra 3.

The fast developing Integrated Circuit (IC) manufacturing technology has provided the industry with billions of transistors on a single chip [50]. At the same time, the number of Intellectual Property (IP) blocks integrated on an Application-Specific IC (ASIC) has been increasing which leads to an exponential rise in the complexity of their interaction. If this trend holds, the traditional digital system design methods will encounter critical challenges and performance bottlenecks. One of the most well known and critical problems is the communication bottleneck. Most bus-based SoCs have the bus based communication architecture, such as simple or hierarchical buses. In contrast with the increasing chip capacity, bus based systems do not scale well with the system size in terms of bandwidth, clock frequency and power consumption [28].

In small scale SoCs and CMPs, crossbar based architectures are advantageous since they have a simple topology and are easy to design and implement. Besides, the latencies are usually fixed and messages can be broadcasted. However, the scalability of crossbar is limited in large systems, therefore it may not be suitable for these systems. of Figure 1.1a and 1.1b respectively show the AMD Istanbul 6-Core architecture and Intel Nehalem 8-Core architecture.

In these architectures, the components (cores, caches and memory controllers) share the transmission medium, but only one device can drive the bus at a time. As the amount of components increases, bus based communication architectures are no longer feasible since the intrinsic parasitic resistance and capacitance can be quite high in relatively longer bus lines. Every additional core connected to the bus will increase this para-

[1]NVIDIA and Tegra are trademarks or registered trademarks of NVIDIA or its subsidiaries. Other names and brands may be claimed as the property of others.

sitic capacitance, which in turn increases the propagation delay and power consumption. With the increases of the bus length and/or the number of cores, the delay in bit transfer over the bus will reach an intolerable number of clock cycles [40]. Moreover, energy inefficiency is another critical limitation in bus based systems because of data are transferred in the broadcast manner. Therefore, it is claimed that bus based architectures are only feasible in chips that integrate fewer than five processors [16, 68]. For example, crossbar interconnect is used in Intel Nehalem and Westmere architectures. Each core has its own data path (around 1,000 wires) to the central interconnect. This provides high bandwidth for a small number of components. The latency grows as the number of components grows, e.g. the latency to the last level cache with six cores is 22% higher than that in four cores [112]. The crossbar has to be redesigned and it becomes more complex. Another research show that high-radix crossbars are not feasible due to the area overhead and the frequency limitations [88].

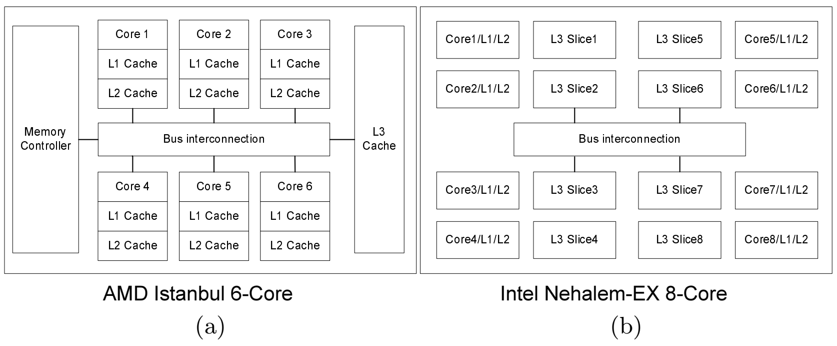


Figure 1.1: AMD and Intel multicore processors.

Due to the limited scalability of crossbar switches, academics and industries have moved to new approaches. Sun SPARC T3 applied a hierarchical crossbar to alleviate the scaling problem. While IBM Power 7, Intel Sandy Bridge, Nehalem-EX and Westmere-EX rely on a ring interconnect. Previous research have focused on the performance and efficiency of crossbar/ring/mesh interconnects. For example, Cheolmin Park et al. [81] presented a 1.2 TB/s ring interconnect implementation, providing on-chip communication for 8 cores, 8-port parallel-access 24 MB cache, and 2 system interfaces. To improve performance and reduce power consumption, hyper ring, hierarchical ring and hybrid ring have been proposed [20, 95]. Jesus Camacho Villanueva et al. [110] compared the performance differences between crossbar, ring and mesh interconnects. Results have shown that in terms of the average execution time of 7 scientific applications, the mesh network has reduced 19% compared with the ring, and 26% compared with the crossbar.

To address the aforementioned problems and to improve the system performance, Network-on-Chip (NoC, or on-chip network) was proposed as a promising solution in the field of SoCs and CMPs [28, 54, 16]. It endeavours to bring network communication methodologies into on-chip communications. The design approach of NoC is to create a communication infrastructure beforehand and then map the computational resources to it via resource dependent interfaces. Processing Elements (PEs) in a NoC are connected by routers and network links, and data are transferred in the form of network packets. This modular approach also provides more efficient communication by leveraging computer network principles. Designers have to work with individual transistors, gates and other small components several decades ago. Later on, they work with IP blocks, large components such as Arithmetic Logic Units (ALUs) and hardware accelerators. In the future, the design of processors can move to a even higher level of system engineering. For example, the IP blocks are placed in optimal positions in an on-chip network, accelerating the process of designing a new CMP.

On-chip network is a solution for solving the communication of device modules for future CMPs and SoCs. However, as the die area grows, traditional Two Dimensional (2D) chip interconnection will result long global wire lengths, which can cause high wire delays, high power consumptions and low system performances [100]. Besides 2D chips usually have large die size in multiprocessor implementations. AMD announced its twelve-core x86 processor as two dies on a chip, each 346mm^2 [6]. The chip is manufactured from 2D Multi-Chip Module (MCM) technology. Needless to say, the size of the die is an obstacle to the progress of IC manufacturing. To continue the progress of Moore's law, Three Dimensional (3D) integration is introduced. Multiple chip dies are stacked vertically, like a multi-floor building. The communication between die layers are done by pillars, like the elevators in a multi-floor building. 3D integration has the potential to increase device density, providing shorter wire lengths and faster on-chip communication compared with 2D integration. Chapter 3 studies two aspects of 2D/3D NoC: designing 2D/3D CMP with different cache architecture, and 3D NoC with on-chip Dynamic Random Access Memory (DRAM).

It is expected that future multicore processors based on NoC could contain hundreds of even thousands of cores. There are many hardware resources in such a chip, for example, cores, caches, routers, links, memory controllers and so on. Each node can include all the necessary hardware resources for best performance and availability. However, in terms of efficiency, this can be uneconomical because some resources can be wasted due to low utilization. To solve this problem and improve system efficiency, one method is to distribute a limited number of a certain resource. Notwithstanding, in this case, multiple requesters have to share a resource, leading to possible traffic contention and/or performance bottleneck. Intelligent placement of hardware resources in a mesh-based NoC is introduced in Chapter 4. Optimal or sub-optimal placement of the resources is determined, so that performance degradation caused by reduced amount of resources is alleviated. For example, the manufacturing cost of the through silicon via in a 3D chip is expensive. However, the communication between different layers in a 3D chip relies on these connections. An optimal placement of reduced number of through silicon vias is critical. Other hardware resources of different communication patterns, including memory controllers and cores/caches, are studied in this dissertation as well.

The application support for NoC architecture is very important. The hardware performance may be limited without optimized software system. For example, in H.264 data parallel coding, video stream data are distributed to processors. Multiple video stream data can be processed simultaneously in data parallel coding. However, data dependencies among coding threads can be a major bottleneck of H.264 coding. Hence, by reducing the number of inter-processor communication or improve the efficiency of the communication, we can achieve better scalability of data parallel processing in a NoC. Chapter 5 studies several applications optimized for NoC architecture, including data parallel H.264 coding, hierarchical N-Body methods, fast Fourier transform and so on.

The scheduling of processes and threads is more and more important for multicore systems [36]. It is demonstrated that in an eight-core AMD Bulldozer system, the performance can improve up to 20 % with proper scheduling [113]. For even larger scale systems such as hundred-core chips, it is obvious that scheduling of multi-threaded tasks to achieve better or even near optimal efficiency is crucial in the future. For example, in terms of power saving, since dynamic and static power are important factors of a chip, hardware-software cooperated strategy is crucial. For a NoC-based CMP, power/clock gating can be applied to unused components to save both dynamic and static power, dynamic voltage and frequency scaling can be applied to individual routers and links to save power. Two scheduling algorithms designed for 2D/3D NoC architectures are proposed in Chapter 6.

1.1 Dissertation Organization

The dissertation is divided into seven chapters. Chapter 2 gives an overview of the on-chip networks, including two dimensional and three dimensional networks. The simulation platform used in this dissertation is also introduced, including the hardware configuration and workload. Several example outputs are demonstrated. The different designs of cache architecture in 2D/3D NoCs are presented in Chapter 3. A 3D NoC design with on-chip dynamic random access memory is also illustrated in this Chapter. In Chapter 4, the optimal placement of resources in a mesh network is introduced. To reduce computation complexity, this dissertation proposes a divide and conquer method. Three cases of resource placements, namely through silicon via, memory controller and core/cache are used to study the effectiveness of this method. Chapter 5 presents several studies of application implementation for the NoC platform. Advices on platform specific optimization are given. The scheduling of threads/processes in a NoC environment is discussed in Chapter 6. Two schedulers, focused for 2D and 3D NoCs are proposed. Chapter 7 concludes the dissertation.

1.2 Dissertation Contributions

The main contributions of this dissertation are:

1. Different cache and memory designs in 2D/3D NoC architectures are analyzed in [119] and [123]. The split core/cache architecture for 3D NoCs is proposed and analyzed to reduce latency and improve performance. Experimental results show significantly reduced average network latency and average link utilization in the split core/cache design compared with the conventional integrated core/cache design in 2D NoCs. Furthermore, we introduce a novel 3D NoC architecture with on-chip dynamic random access memory, in which different layers are dedicated to different functionalities such as processors, cache or memory. Memory bandwidth is significantly improved in this architecture.

2. The problem of resource placement in a Network-on-Chip architecture is considered in [120], [122], [127] and [121]. In on-chip network system, there are many hardware resources, e.g. cores, caches, routers, links, memory controllers and so on. It might be not economical to equip each node with a hardware resource. To improve system efficiency, one solution is to distribute a limited number of a certain resource. Multiple requesters have to share a resource, leading to possible traffic contention and/or performance bottleneck. To alleviate the problem of performance degradation by reduced resources, intelligent placement of resources is introduced. Three hardware resources are used as case studies: TSV, memory controller and core/cache.

3. The adaptation of applications in a NoC environment is studied [124, 131]. Without the collaboration of software application, the hardware performance of the NoC could be limited. Several applications

originally designed for traditional system architectures are analyzed, including data parallel H.264 coding, hierarchical N-Body methods and fast Fourier transform. The analysis are based on low-level communication patterns. Optimization suggestions are given for NoC architectures.

4. The Operating System scheduling is one of the most important design issues for future CMP systems. Two scheduling algorithms for 2D and 3D NoC architectures are presented [130, 129, 126]. A minimal average accessing time scheduling algorithm for 2D NoC is proposed to reduce on-chip communication latencies and improve performance. The impact of memory access and inter process communication in scheduling are analyzed. Another scheduling algorithm, namely greedy heuristic approximation scheduling algorithm, is proposed for 3D NoCs. The limitation of through silicon via is considered in this algorithm.

Chapter 2

Background

Over the last forty years, the IC technology has provided the ability of integrating increasing number of elements in planar form. Driven by the need of implementing a large number of elements on a single chip, the IC technology has gone through the process of Programmable Logic Devices (PLD) and General-Purpose Logic (GPL) devices, Field Programmable Gate Arrays (FPGA), ASIC, and eventually to System-on-Chip (SoC).

In the early 1960's, the GPL devices were the first implementation format used in the semiconductor industry. Soon after that, scientist developed the first memory devices and then the first microprocessor on chip in 1971. The successful combination of GPL devices, memories and microprocessors has formed the heart of digital systems and a revolutionary milestone known as embedded systems. In late 1970s, the Micro Controller Unit (MCU) was developed. The idea of a MCU is to integrate a whole microprocessor system on a single chip. Since then, the hardware implementation of digital systems has followed the development from PLDs to FPGAs and ASICs. In 1990s, the introduction of IP blocks and virtual components brought in a new implementation format called system-on-chip where a single chip contained mostly reusable IP based logic blocks. As the scale increased, normal bus-based SoC has encountered several severe challenges such as deep submicron effect, global synchronization problem, power and thermal management, verification and productivity gap [28, 54, 16]. In order to solve these problems, the architecture of Network-on-Chip (NoC) was proposed in the SoC community at the beginning of the 21st century.

2.1 Network-on-Chip

During the last decade, NoC has been proved to be an promising solution for the concerns in SoC and embedded systems in terms of data parallelization [65]. Being able to integrate a large number of components on a single chip, NoC serves as an ideal on-chip platform for multicore systems. A component can be a cache bank, a processor core or even an FPGA block [28, 54, 16]. Data packets are produced or consumed by a component at a given time. Idle components can be turned off to save energy. The parallelization is achieved by the simultaneous operation of the PEs.

Figure 2.1 shows a mesh-based NoC with 16 nodes/tiles (N). Each PE contains a Network Interface (NI) and a core with private L1 cache (Core/L1) and shared L2 cache (L2). The Router (R) includes a Routing Computation Unit (RCU), a Virtual Channel Allocator (VCA), a Switch Allocator (SA), a Crossbar Switch (CS), several Virtual Channels (VC) and input buffers. The communication among PEs is achieved via the transmission of network packets. This modular approach also provides efficient communication and high bandwidth. Moreover, in a NoC, since there is no bus arbitration needed, more transactions can occur simultaneously and thus the latency of the packets is reduced and the throughput of the system is increased. As the links in NoC are based on point-to-point mechanism, the communication among cores can be pipelined to further improve the system performance. Apparently, compared with traditional bus-based CMP designs, NoCs have advantages since it can be scaled easier. Notice that when using "network" in this dissertation, we refer to mesh network by default.

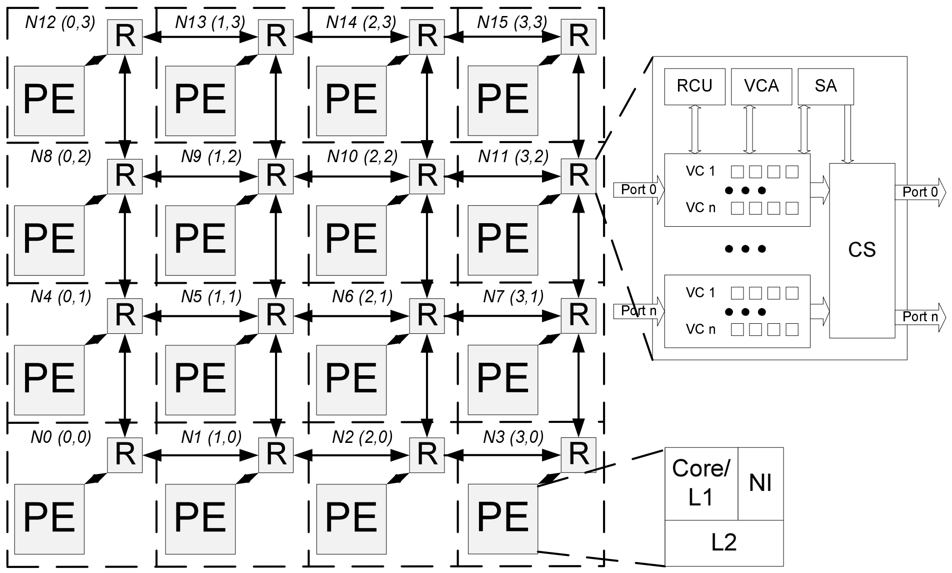


Figure 2.1: An example of 4x4 NoC with mesh topology.

Intel has demonstrated an 80 tile, 100M transistor, 275mm^2 2D NoC prototype under 65nm processing technology [107]. Recently, an experimental CMP containing 48 cores on a chip has been manufactured by Intel using 4×6 network-based 2D mesh topology with 2 cores per tile [51], each core of this chip is a standard x86 core. The Tilera company has TILE processor family, which includes TILE64, TILEPro and TILE-Gx members [102]. The basic architecture of these processor are the same: an array of 16 to 100 general purpose Reduced Instruction Set Computing (RISC) processor cores (tiles) in a on-chip mesh interconnect. Each tile consists a core with related L1 and L2 caches. The memory controllers are integrated on the chip as well.

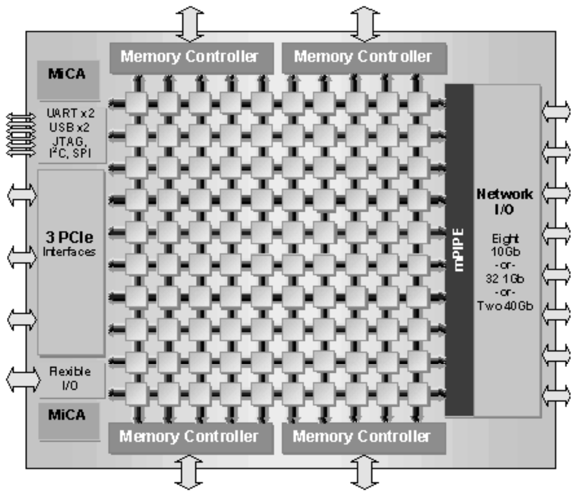


Figure 2.2: The Tilera TILE multicore processor with 100 cores [102].

Figure 2.2 shows the architecture diagram of TILE-Gx processor. Each tile consists of a 64-bit VLIW core with private L1 cache (32KB instruction and 32KB data) and shared L2 cache (256KB per tile). Four 64-bit DDR3 memory controllers, duplexed to multiple ports, connect the tiles to the main memory. The L2 caches and the memory are shared by all processors. The processor operates at 1.0 to 1.5GHz, with typical power consumption of 10 to 55W. The I/O controllers are integrated on chip to save costs of other peripheral chips. The mesh network provides bandwidth up to 200Tbps.

2.2 Three Dimensional Network-on-Chip

3D NoC is an emerging research topic exploring the architecture of 3D ICs that stack several NoC dies vertically [85]. It is shown that a 3D architecture can reduce wire length as much as the square root of the number of stacked layers [56]. Since the wire delay is related to the square of the wire length (linear if repeaters are used), 3D NoC provides higher performance and lower power consumption than its 2D counterpart [56]. Traditional stacking technologies such as System-in-Package (SiP) and Package-on-Package (PoP) have been integrated into manufacturing technology. Recent researches have focused on Through Silicon Via (TSV) [84]. TSV is a viable solution in building 3D chips by stacking IC layers together using vertical interconnects. These interconnects are formed through the die to enable communication among different die layers [101]. Layers with different functions, e.g. processor layer, cache layer, controller layer, Radio Frequency (RF) layer and analogue layer can be implemented in a 3D NoC. This enables the mapping of components to be optimized to a much larger degree. The average on-chip wire length is considerably reduced in this architecture. Figure 2.3 shows a 3D NoC model with one layer of processors and one layer of caches.

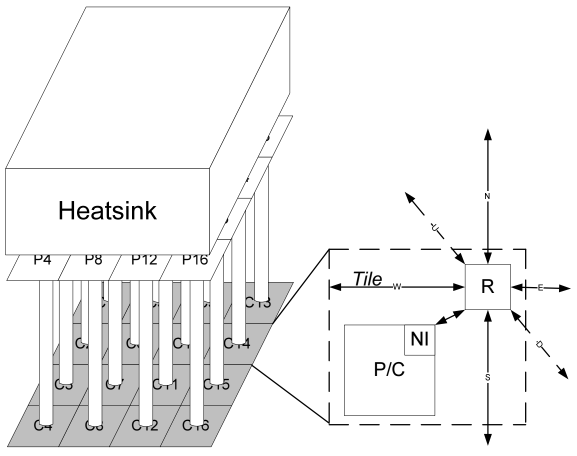


Figure 2.3: Schematic diagram of a 3D NoC with one processor layer (Px) and one cache layer (Cx), layers are fully connected by TSVs. The heatsink is attached on top of the processor layer

2.2.1 3D NoC Router and Routing Algorithm

As is shown in Figure 2.1, routers in mesh-based 2D NoCs have five ports to connect to five directions, namely, North, East, West, South and Local PE. For the vertical communication between different layers, routers in 3D NoC have two more ports and the corresponding virtual channels, buffers and crossbar to connect the Up and Down inter-layer connections. It is noteworthy that in a two-layer 3D NoC design, as shown in Figure 2.3, not all routers require seven ports, e.g. router of P4 has only East, North, Down and Local PE ports.

Adaptive routing algorithm, such as Routing Information Protocol (RIP), Enhanced Interior Gateway Routing Protocol (EIGRP), Open Shortest

Code 1 The pseudo code of the routing algorithm.

```
procedure Route(network, flit):
    for each flit f in network:
        if f.source[Z] = f.destination[Z]
            use X-Y deterministic routing

        direction[f] := X
        do send f to next node
            if f.position[X] = pillar.position[X]
                direction[f] := Y
            if f.position[X, Y] = pillar.position[X, Y]
                direction[f] := Z
        until f.source[Z] = f.destination[Z]

        send f to next node according to X-Y routing
```

Path First (OSPF) and Border Gateway Protocol (BGP), is used widely in off-chip networks [108, 49], however deterministic routing is favorable for on-chip networks because the implementation is easier. Here, a dimensional ordered routing (DOR) [99] based deterministic routing algorithm is selected and modified to fit the 3D topologies. When a node N_{source} sends a flit to a node $N_{\text{destination}}$, the flit will first travel along the X direction, then it will be routed in the Y direction. As the flit reaches the pillar, it will be vertically routed in the Z direction to the layer of the destination node. X-Y deterministic routing is used when the flit arrived the destination layer, in which a flit is first routed to the X direction and last the Y direction. The pseudo code of the routing algorithm is shown in Code 1.

2.3 Simulation Platform

Since the NoC hardware/software environment is still limited, simulator for NoC, especially a full system simulation environment is very important. There are several partial function simulators for NoC. Nostrum from KTH simulates the network model [61], which is a wormhole switching, Virtual Channel (VC) based network. Popnet and Garnet from Princeton [33, 3], Noxim from University of Catania functions similar as Nostrum [80]. Router and link power model by ORION is developed by Princeton as well [111]. Cacti, an integrated cache timing, power, and area model, brought by HP Labs, has been used widely for researching multiprocessor cache system [94]. Simics is a full system instruction-set simulator developed by Virtutech [73] and used to run unmodified applications of any supported hardware architecture (e.g. SPARC on x86). However Simics does not simulate cache coherence, the memory is perfect and can access data without any cycle delay. The aforementioned simulators are special function simulator.

GEMS/Simics is a full system simulator developed by University of Wisconsin based on Simics processor model [75]. It features GEMS cache/ memory model, ORION router/link power model and Garnet detailed wormhole virtual channel pipeline router model. With GEMS/Simics, it is possible to run unmodified operating system and software applications for a destination platform, such as the original Sun Solaris/SPARC and SPEC benchmark programs based on Solaris. The hardware system architecture can be defined in details before running, with SMP/CMP support, L1/L2 cache size, memory size, the latency and timing of cache and memory, memory/cache coherence protocol, network interconnection, various buffers, routing protocol, router model and so on.

GEMS/Simics is able to simulate several cache coherence protocols. Different protocols can have impact on system performance. In multiprocessor systems, each block in the cache system have several states concerning cache-memory data consistency, for example:

- Modified (M). This means the data block in cache has been modified, it is inconsistent with memory. The block has to be written to memory before any read operation.
- Shared (S). This means the data block in cache has not been modified; it holds the most recent consistent data with memory. Same cache data might exist in other processors of the system.
- Invalid (I). This means the data block in cache is invalid, valid data must be retrieved either from other caches or memory.
- Exclusive (E). This means the data block in cache has not been modified; it holds the most recent consistent data with memory. Data only exist in one cache.
- Owned (O). This means the data block in cache has not been modified; it holds the most recent consistent data with memory. Same cache data might exist in other processors of the system. But the data in memory can be inconsistent; only one cache can hold this state. Shared state data must be hold in other caches.

According to these states, GEMS/Simics can simulate SMP and CMP architectures with either MESI or MOESI cache coherence protocols. The main memory is shared in GEMS/Simics. Here, a SMP machine mimics each processor node of a processor, a private L1 cache and a private L2 cache, each processor has a memory and directory controller. Interconnection latency between processor nodes in an SMP system is typically higher than CMP. A CMP machine mimics each processor node of a processor, a private L1 cache and a share L2 cache between all nodes. It is possible to implement multiple CMPs within a system (SMP-CMP). MESI is used widely because it supports write-back cache policy. In a write-back cache dirty data writes are not synchronized to the memory compared with a write-through cache. MOESI is an elaborated MESI protocol; it has been used in AMD64 architecture. The protocol is designed to deliver higher performance but is much more complex.

GEMS/Simics uses a simple 2D based network topology, with Garnet detailed wormhole router model and ORION router/link power model. Garnet in GEMS/Simics consists of a fixed and flexible pipeline models. It is based on the state-of-the-art virtual channel router (Figure 2.4); the router can have different number of input and output ports, as well as input buffers. There are several other parts of the router such as virtual channel allocator, switch allocator and crossbar switch. The default routing computation algorithm is X-Y deterministic. It is possible to implement other routing protocol such as an adaptive one.

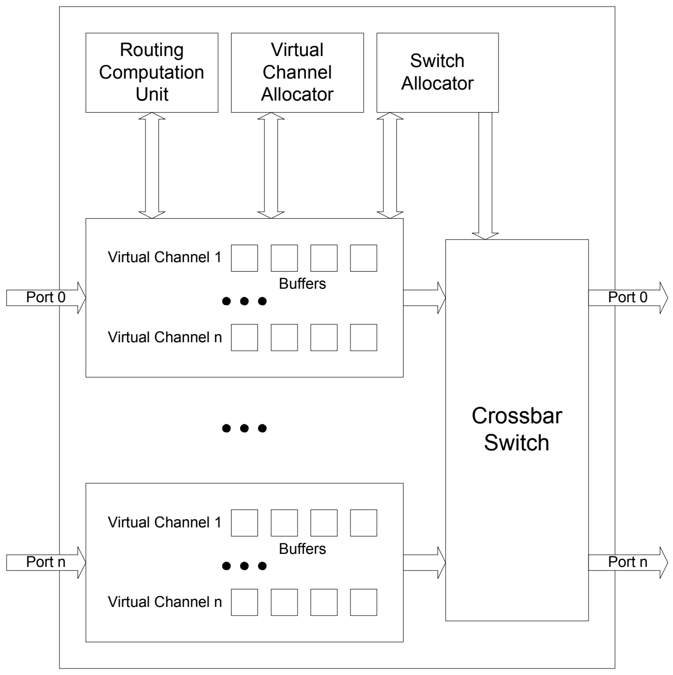


Figure 2.4: Router model in GEMS/Simics.

The router has a classic five stage pipeline. A head flit arrives at an input port as the Buffer Write (BW) stage and a request is sent to the

Route Computation (RC) at the same time. The output port is therefore calculated. The flit then arbitrates for a corresponding VC in the VC Allocation (VA). If succeed, Switch Allocation (SA) stage will take place which arbitrates for crossbar switch ports. Switch Traversal (ST), which means crossbar traversal, happens after. Link Traversal (LT) is the last stage for transferring to the next node. Route computation and VC allocations are not required for flits other than the head flit (such as body and tail). The tail flit has to de-allocate the VC. Five stages of the pipeline are shown in Figure 2.5.

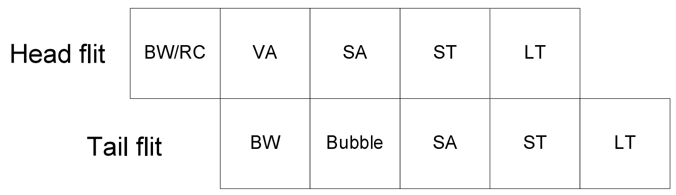


Figure 2.5: Five stage pipeline.

2.3.1 Example Configuration

The original GEMS/Simics has limited functionality. However, since the source code is available, we have made several modifications to meet our experimental requirements in this thesis. Here, we configure a simulation platform. We use a 16-node network which models a single-chip CMP for our 2D NoC. A full system simulation environment with 16 nodes, each with a core and related cache, has been implemented. The simulations are run on the Solaris 9 operating system based on SPARC instruction set in-order issue structure. Each processor core is attached to a wormhole router and has a private write-back L1 cache. The L2 cache shared by all processors is split into banks. The size of each cache bank is 1MB; hence the total size of shared cache is 16MB. The simulated memory/cache architecture mimics Static Non-Uniform Cache Architecture (SNUCA) [58]. MOESI, a two-level distributed directory cache coherence protocol is used in our memory hierarchy, where each L2 bank has its own directory. The detailed configurations of processor, cache and memory configurations can be found in Table 2.1.

2.3.2 Example Workload

Several standard multiprocessor benchmark programs are used to evaluate system overall performance. These programs include SPLASH-2 [116], PARSEC [18, 17], SPECjbb [98] and TPC-H [104].

Table 2.1: System configuration parameters

Processor configuration	
Instruction set architecture	UltraSPARCIII+ in-order
Number of processors	16
Processor frequency	2GHz
Cache configuration	
L1 cache	Private, split instruction and data cache, each cache is 16KB. 4-way associative, 64-bit line, 3-cycle access time
L2 cache	Shared, unified 16MB (16 banks, each 1MB), 64-bit line, 6-cycle access time
Cache coherence protocol	MOESI
Cache hierarchy	SNUCA
Memory configuration	
Size	4GB DRAM, 4KB/page
Access latency	260 cycles
Requests per processor	16 outstanding
Network configuration	
Router scheme	Wormhole
Flit size	128 bits

SPLASH-2, developed by Stanford in 1995, is one of the mostly commonly used shared memory parallel benchmark program for scientific uses. It consists of programs of high scalability. Concurrency and load balance are considered seriously in SPLASH-2, it is shown that Barnes, FFT, Volrend, Water-Spatial, Ocean, Water-Nsquared, FMM and Raytrace can achieve high scalability from one to sixty-four processors. SPLASH-2 has been used over ten years as the state-of-the-art parallel benchmark program for share memory systems.

PARSEC, developed by Princeton and Intel in 2008, is expected to be the next generation of benchmark suite for shared memory CMP systems. It consists of programs that are believed to be optimized for modern multicore architectures. Both SPLASH-2 and PARSEC are used widely in evaluating performance of parallel computer systems. Other benchmark suites, e.g. ALPBench [62], is designed for special purpose applications such as multimedia processing.

SPECjbb is Standard Performance Evaluation Corporation's (SPEC) standard Java server benchmark. It evaluates the performance of a Java server by simulating a three-tier client/server system (client, business logic engine, server). The program tests the performance of the shared memory system and the scalability of multiprocessors. The clients are simulated by driver threads, and the database storage are represented by binary trees of objects. We choose Sun JRE-SE 1.4.2 as the Java runtime environment.

The Transaction Processing Performance Council's (TPC) Benchmark H (TPC-H) is a decision support benchmark. The benchmark examines large amount of data from the database. It executes business oriented ad-hoc queries and concurrent data modifications. The data and queries give answers to critical business questions. The result reported by TPC-H is called the TPC-H Composite Query-per-Hour Performance Metric. It reflects the capability of the system to process queries. We configure MySQL v5.0.67 with 1GB of reference database, query 1 is used.

Chapter 3

Cache and Memory in 3D Network-on-Chip

Network-on-Chip has become a widely accepted on-chip communication architecture which provides a promising solution to integrate a large number of components on a single chip. However, with the increasingly higher performance demands for on-chip systems, NoCs are facing several critical challenges such as wire delay and power consumption. Therefore, in this section, we explore different cache and memory designs in 2D/3D NoC architectures.

In Section 3.1, integrated core/cache and split core/cache architectures have been analyzed. We present benchmark results using a cycle accurate full system simulator. Experiments show that by using the proposed architecture, compared with the conventional integrated core/cache design, the average network latency and average link utilization are reduced by 5.01% and 26.07% respectively.

In Section 3.2, observing that the memory bandwidth of the communication between on-chip components and off-chip memory has become a critical problem even in NoC based systems, we propose a novel 3D NoC with on-chip Dynamic Random Access Memory (DRAM) in which different layers are dedicated to different functionalities such as processors, cache or memory. Results show that by using our proposed architecture, average link utilization has reduced by 10.25% for SPLASH-2 workloads. Our proposed design costs 1.12% less execution cycles than the traditional design on average.

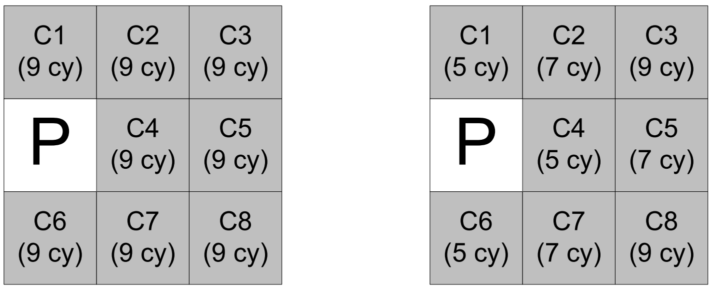
3.1 2D/3D Multiprocessor with Different Cache Architecture

In this section, we investigate 2D/3D NoC designs based on Non Uniform Cache Architecture. We model a 16-core CMP based on NoC architecture. We analyze two cache architectures; the integrated core/cache architecture in 2D NoC and the split core/cache architecture in 3D NoC. The impact of area and latencies in both architectures are analyzed. We present the performance of two systems using a full system simulator.

Previous researches have focused on several implementations for 3D NoCs. Feihui Li et al. [70] presented an implementation alternative in which cores and caches were placed in interlaced locations in all the three dimensions. In [82], Dongkook Park et al. illustrated a design to span the components in 3D NoC across all the layers so that the wire latency was reduced. Implementing caches in 3D has been explored for traditional processors [106]. Kiran Puttaswamy et al. [89] have investigated the on-chip 3D cache integration technology using Through Silicon Vias (TSVs). Their experiments show that the latency in a uni-core chip with 3D cache can be reduced by 21.5% comparing with its 2D couterpart.

3.1.1 The Architecture of Cache

Modern commercial multicore processors are designed for multi-bank multi-level Uniform Cache Architecture (UCA). The worst case wire delay in UCA may result in extremely low performance since the access times are uniform for all the caches. Thus, it is obvious that system performance can be improved if the cache access time varies from one cache bank to another so that closer cache banks have smaller number of access clock cycles. This technique is called Non Uniform Cache Architecture (NUCA) [14, 47]. Based on how data are mapped to the caches, NUCA implementation can be categorized into two groups, namely Static NUCA (SNUCA) and Dynamic NUCA (DNUCA). In SNUCA, data are mapped to cache banks statically. It is claimed that designing a NoC based SNUCA system saves channel overhead ranging from 20.9% to 5.9% compared with its private per-bank channels counterpart [58]. A DNUCA design maps the data to the cache dynamically so that frequently accessed data are migrated to closer banks for the sake of less access cycles. However, DNUCA is more complex than SNUCA since it involves data mapping, searching and replacement. Here, we explore systems using SNUCA. Figure 3.1a shows an UCA system with uniform 9-cycle latency, while in Figure 3.1b, closer cache banks, e.g. C1, C4 and C6 have lower access latencies.



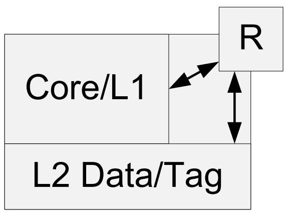
(a) UCA (b) NUCA

Figure 3.1: UCA and NUCA for one processor (P) with eight cache banks (Cx).

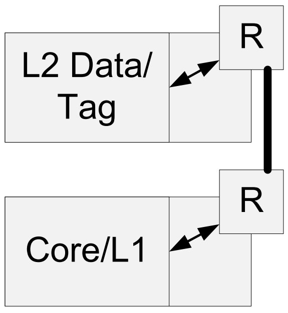
3.1.2 Integrated and Split Core/Cache

The floorplan of modern multicore chips such as third-generation Sun SPARC [105], IBM Power 7 [48], AMD Istanbul [6] shows the design choices of the PE. The total area of Sun SPARC chip is 396mm^2 with 65nm fabrication technology. Each core has an area of 14.45mm^2. We simulate the characteristics of a 16MB, 16 banks, 64-bit line size, 16-way associative, 65nm L2 cache by CACTI [94]. Results show that the total area of cache banks is 193.38mm^2. Each cache bank, including data and tag, occupies 12.09mm^2. A 5-port router is estimated to be 0.23mm^2 scaled to 65nm, as we calculated. We estimate that the area for a tile of the 2D NoC is around 14.45 + 12.09 + 0.23 = 26.77mm^2 (as shown in Figure 3.2a, each PE has a core with private L1 cache and shared L2 cache). Considering a 2D NoC with 16 tiles, the total area is about 428.32mm^2, comparable with modern CMPs [105, 48, 6].

Figure 3.2b shows a 3D NoC design with split core/cache. We adopt a 7-port router for our 3D NoC model with TSVs (bold line in Figure 3.2b) as the upward/downward communication links [125]. It is noteworthy that routers are quite small compared with processors and cache banks, e.g. scaled to 65nm, as we calculated, a 7-port 3D router is estimated to be only 0.41mm^2. Furthermore, not all routers in a 3D NoC require seven ports. Therefore, the area for a core tile of the 3D NoC is around 14.45 + 0.41 = 14.86mm^2, the area for a cache tile is around 12.09 + 0.41 = 12.50mm^2 (the router occupies less than 2.76% and 3.28% of tile area, respectively). The split core/cache design did not reduce the functionality of the system, but rather reduce the wire length between tiles, since the footprint of each tile is smaller. The wire length will be analyzed in following sections.



(a) 2D integrate Core/Cache



(b) 3D split Core/Cache

Figure 3.2: Comparison of 2D NoC design with core and cache within the same tile (a), and 3D NoC with core and cache in different tiles and layers (b).

3.1.3 Thermal Issue in 3D NoCs

Since the processors consume overwhelming majority of power in a chip, it is expected that stacking multiple processor layers could be unwise for heat dissipation. According to a research [124], by stacking multiple processor layers, heat dissipation is a major problem for some part of the chip even if processors are interlaced vertically. Without direct contact with heatsinks, the peak chip temperature of 3D design raises by 29℃ compared with the 2D design, which is not feasible for some applications [124]. However, by stacking more cache layers instead of processor layers, the thermal constraint is supposed to be alleviated. Gian Luca Loi et. al. shows that even for 18 stacked layers (1 of processors, 1 of cache and 16 of memory), the maximum temperature for a 3D chip increases only 10℃ comparing with 2D chip [72]. It is estimated that 15% lower core frequency of a 3D chip could compensate the thermal drawback [72].

On account of the aforementioned analysis, we choose a 3D NoC model with one layer of processors and one layer of caches as shown in Figure 2.3. In consideration of heat dissipation, the processor layer should be on top of the chip (near heatsink). The processor layer is a 4×4 mesh of Sun SPARC cores with private L1 cache. Based on the aforementioned analysis, the total area of the core layer is around 237.76mm^2. The cache layer has a total area of 200.00mm^2. Since the cache layer is smaller than the core layer, other system components, e.g. I/O and memory controller can be implemented in the cache layer. As we expected, the footprint of the 3D NoC is much smaller, just 55.51% of the 2D NoC.

3.1.4 Cache Access Latencies

We note that data are mapped into banks statically in SNUCA. The low order bits of the index determine the bank number. Considering a perfect application with average access probability to cache banks, for our 16-bank cache, each bank has an access probability of $^{1}/_{16}$. However, real-world applications tend to have different cache access patterns, i.e. the access probability to certain banks might be higher than others.

We first consider a cache hit of the cache architecture shown in Figure 3.2a. The core requests a data, the data is right in the local L2 cache, thus accessing the data only involve the in-tile wire latency between the core and cache, the operation did not pass through local router. In the cache architecture shown in Figure 3.2b, the closest L2 cache of the core requires one network hop using TSV, which involves link latency of the TSV and the latency of routers. For a router in the NoC, there are several parts (e.g. the routing computation unit, the virtual channel allocator, the switch allocator and the crossbar switch) that will affect latency, depending on the number of pipeline stages. Here, we use a router of two pipeline stages. It is noteworthy that the TSV is quite short (e.g. $50\mu m$) compared with links between adjacent routers which are usually several millimeters.

Equation 1 shows the latency of a local cache access in 2D NoC. Because of core and L2 cache are in the same tile (Figure 3.2a), the latency involves only two in-tile short links (L_{Link_delay1}) and the router (L_{Router_delay}).

Equation 1 $L_{L2D} = 2 \times L_{Link\_delay1} + L_{Router\_delay}$

By splitting the core and cache in different layers, the latency of a local cache access in a 3D NoC is shown in Equation 2. In this situation, one more router delay and the delay caused by TSV (L_{TSV_delay}) are added. It is obvious that this local cache access is slower than 2D NoC.

Equation 2 $L_{L3D} = 2 \times L_{Link\_delay1} + 2 \times L_{Router\_delay} + L_{TSV\_delay}$

Assuming X-Y deterministic routing, Equation 3 shows the latency required for a remote cache access, in case the required data for a core is in another cache bank. Figure 3.3 shows an example of a remote cache access, in which core in the node N2 accesses data in cache bank of node N15. This access costs two in-tile link delays, four tile-tile link delays (L_{Link_delay2}) and five router delays.

Equation 3 $L_{R2D} = 2 \times L_{Link\_delay1} + (n_{hop} + 1) \times L_{Router\_delay} + n_{hop} \times L_{Link\_delay2}$

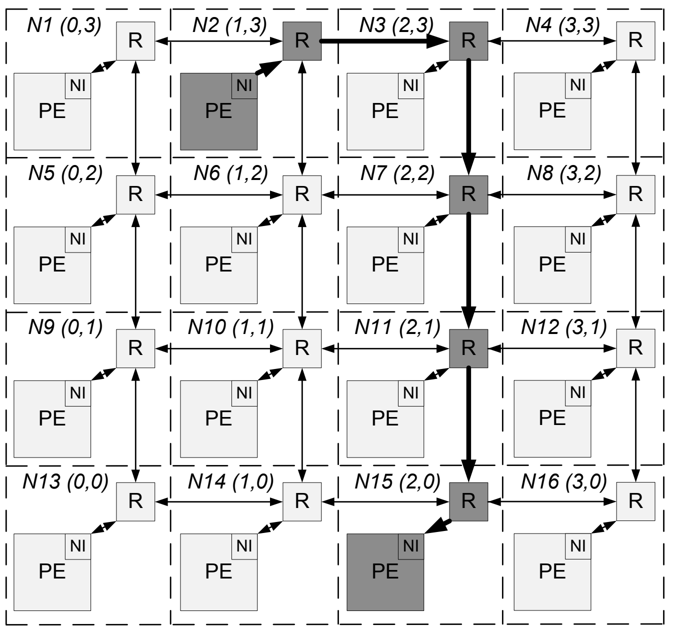


Figure 3.3: An example of remote cache access.

Equation 4 $L_{R3D} = 2 \times L_{Link\_delay1} + (n_{hop} + 2) \times L_{Router\_delay} + n_{hop} \times L_{Link\_delay2'} + L_{TSV\_delay}$

In a 3D NoC, as shown in Equation 4, accessing a remote cache causes one more router delay and the delay caused by TSV as well. Notice that the n_{hop} does not include vertical hops, since the vertical delay is considered in L_{TSV_delay}. However, in the split core/cache architecture, shown in Figure 3.2b, the tile-tile link delays ($L_{Link\_delay2'}$) are lower compared with integrated core/cache architecture, due to the shorter tile-tile links contributed by smaller tile footprint. Based on the aforementioned assumptions, each cache bank in our NoC has an access probability of $^1/_{16}$. Thus despite a local cache access in the split core/cache design is slower than its counterpart, the possibility for a remote cache access is $^{15}/_{16}$, which should be faster. We also note that in a larger NoC, e.g. 8×8, the possibility for remote cache accesses is higher ($^{63}/_{64}$). The benefit for split core/cache architecture in 3D NoC could be improved further.

3.1.5 Wire Delay

Over the last 40 years, the development of microelectronics technology has led to the improvements of system performance in many perspectives including clock frequency, feature size, power consumption and etc., thanks to the evolutionary device scaling. However, scaled chip wiring suffers from the increased resistance and capacitance if metal height is not reduced with conductor spacing. Thus, RC parasitics play an increasingly important role in overall chip performance as feature size scales [44]. According to a report from ITRS, the increasing wire latency and RC delay of long wires have often become bottlenecks of system performance [7].

One of the major advantages of adopting 3D architecture is that it can improve system bandwidth and throughput, and reduce wire length. Ideally, without considering the inter-layer vias and repeaters along long wires, the average wire length is expected to drop by a factor of $\sqrt{N_{layers}}$, where N_{layers} is the number of layers in the 3D architecture. Both wire resistance and capacitance would drop proportionately, thus power would drop by a factor of $\sqrt{N_{layers}}$ and RC delay would drop by a factor of N_{layers} [29]. Repeaters are often used in long wires to reduce the delay so that it increases linearly with wire length. However, repeaters are connected to the power networks and thus consume power when in use. Therefore the application of repeaters is a trade-off.

In this research, we assume that each tile in our NoC platform is of a square shape, and the length of an edge for a tile in the 2D NoC is 5.17mm. In 3D NoC, the length of an edge for a tile is 3.85mm and 3.54mm for tiles with core only and with cache only, respectively. In consideration of the area of the router, we calculate the delay for these wires between routers, since the inter-router latency will be determined by the physical length of the link. For inter-router long wires in voltage-mode transmission, the wire delay is significant. Repeater insertion can considerably reduce the wire delay, while incurring a large power overhead. We simulated 3 links with different lengths in Cadence Spectre. All links are driven by voltage-mode signaling with repeaters. The repeater interval is 0.5mm. The drivers, repeaters and receivers are synthesized in 65nm technology, with $V_{dd} = 0.8$V and temperature of 50°C. The link width, spacing and thickness are $0.14\mu m$, $0.14\mu m$ and $0.35\mu m$, respectively. Table 3.1 summarizes the simulation results. The size of routers are considered. We assume that the network runs at 2.5GHz, with a cycle of 400ps. The total power of the wire is calculated as one bit switching at a time. Results show that the reductions in wire length lead to considerably reductions in both delay and power consumption.

Table 3.1: Comparison of wire characteristics

Wire	**Length**	**Delay**	**Cycle**	**Total P**
2D	4.69mm	2.19ns	6	28.59nW
3D Core	3.21mm	1.58ns	4	20.58nW
3D Cache	2.90mm	1.42ns	4	17.48nW

As aforementioned, the TSV links are very short, e.g. around $50\mu m$. The transmission can be completed within one cycle under 2.5GHz frequency. Based on these assumptions and analyzes, according to *Equation 1* and *2*, it is calculated that 4 cycles are needed for a local cache access in 2D NoC, while 7 cycles in 3D NoC. Apparently, the local cache access is much slower in the split core/cache design. Figure 3.4 shows the total latency of a remote cache access in 2D/3D cache architectures. The total latency of 2D NoC for one hop cache access is lower than 3D (12 and 13 respectively). With increasing hop count, the total latency of cache accesses in 3D NoC becomes much lower than 2D, e.g. the longest cache access in a 4×4 NoC requires 6 hops, inducing 9 more cycles in 2D. For a larger NoC, e.g. 8×8, the longest cache access requires 14 hops, inducing 25 more cycles in 2D.

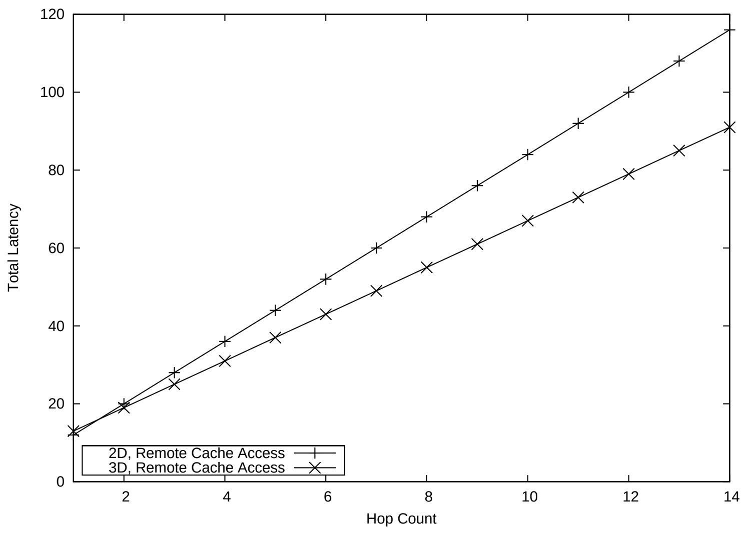


Figure 3.4: Total latency in cycles with increasing hop count.

3.1.6 Experimental Evaluation

In this section, we present the experimental evaluation under different NoC configurations. Workloads used here include FFT and LU from SPLASH-2 [116], Swaptions and x264 from PARSEC [18]. The simulation platform is based on a cycle-accurate 3D NoC simulator which can produce detailed evaluation results (Section 2.3). We use a 16-node network for our 2D NoC. The 3D architecture has one layer for processors and another layer for caches (Figure 2.3). The size of each cache bank is 1MB. MOESI cache coherence protocol is used in our memory hierarchy in which each L2 bank has its own directory.

Figure 3.5 shows that on average, the cache hit latency has decreased 3.74% by spliting the cache from the core in a NoC platform. Application with higher cache access frequencies, e.g. FFT and LU, has significant lower cache hit latency (6.34% and 4.23% lower, respectively), compared with the original integrated core/cache design. The savings are mainly from the shorter wires in the 3D split core/cache design. Notice that since the NoC has other latencies, overall cache hit latency has not improved that much.

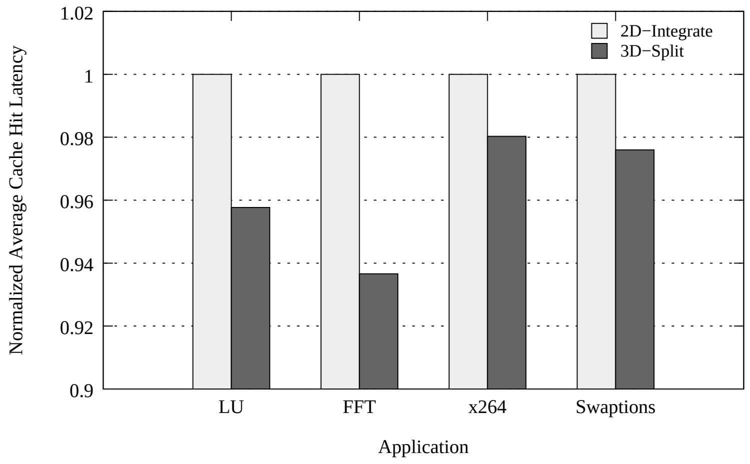


Figure 3.5: Normalized average cache hit latency in cycles.

As is shown in Figure 3.6, the proposed split core/cache design outperforms the traditional design in terms of average link utilization. Average link utilization is calculated with the number of flits transferred between NoC resources per cycle. Under the same configuration and workload, lower utilization means mitigated network load, which is favorable. Comparing with the integrated core/cache design, the average link utilization for our proposed design is reduced by 26.07%, on average. Swaptions and x264 have the most significant reduction of average link utilization, 26.18% and 32.98% respectively.

As illustrated in Figure 3.7, the 3D split core/cache design outperforms the 2D design in terms of average network latency. The improvement is more notable in FFT and x264, with 9.75% and 6.18% reduced latency, respectively, compared with the integrated core/cache design. This is primarily due to the reduced number of cycles required for tile-tile data transmission in the 3D design. The reduction of wire lengths translate directly into network latency savings. On average, comparing with the 2D design, the network latency in 3D design is reduced by 5.01%.

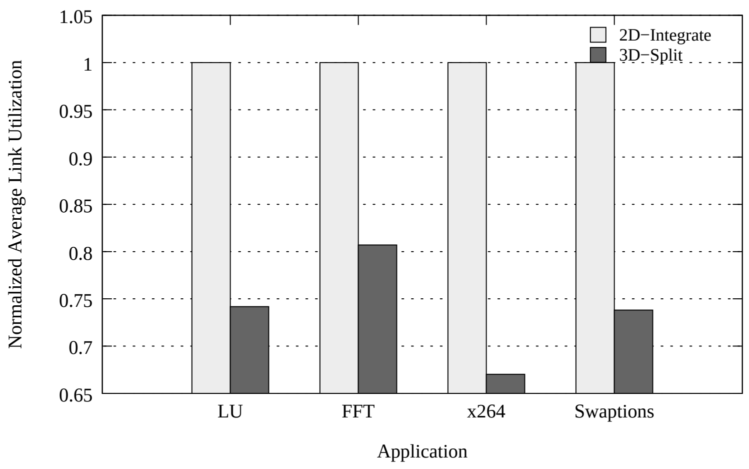


Figure 3.6: Normalized average link utilization in flits/cycle.

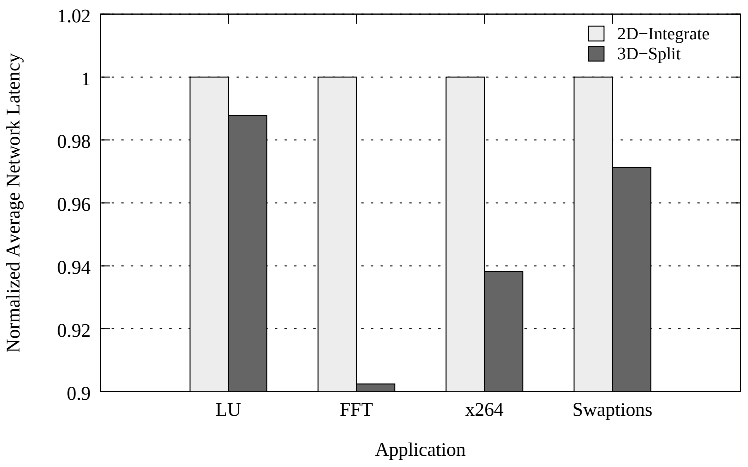


Figure 3.7: Normalized average network latency in cycles.

3.2 3D Network-on-Chip with Dynamic Random Access Memory

There is a great concern about memory bandwidth, the number of memory requests are growing with the number of cores. In the era of Pentium 3, the processor has only one core, thus the requirement for memory bandwidth is relatively low. As the number of processor core grows, the requirement of memory bandwidth grows as well. As it is shown in Table 3.2, Core 2 Duo doubles the requirement of memory bandwidth to fit the requests of two cores. The system performance will decline if memory bandwidth cannot sustain the rate requested by processor cores. By plugging two identical

Dual In-line Memory Modules (DIMMs) on the motherboard, dual channel can be configured to provide double bandwidth. In the dual channel, data is transfered in a 128-bit flavor instead of conventional 64-bit in one cycle. Triple channel is introduced with Double-Data-Rate 3 Synchronous DRAM (DDR3 SDRAM) memory, providing 192-bit data transfer in a clock cycle. Configured with triple channel DDR3-8500 memory, the maximum theoretical memory bandwidth for Intel Core i7 980X is thus 25.6GB/s [53].

Table 3.2: Processor and memory bandwidth for one channel

Processor	**Core**	**Typical memory**	**Typical BW**
Pentium 3	1	PC-133 SDRAM	1.066 GB/s
Pentium 4	1/2	PC-1600 DDR	1.6 GB/s
Core 2 Duo	2	PC2-3200 DDR2	3.2 GB/s
Core 2 Quad	4	PC2-6400 DDR2	6.4 GB/s
Core i7 980X	6	PC3-8500 DDR3	8.5 GB/s

Increasing the memory bandwidth by using DDR4 seems to be a solution, quadrupling or even quintupling the number of memory channels is another solution. However, as mentioned earlier, triple channel configuration requires at least three DIMMs, which increases cost, fault rate and power consumption. Another constraint is the pin count limitation. It is predicted by the ITRS roadmap that pin count will increase by about 10% each year only, comparing with the number of cores that is expected to double every 18 months [7].

There have been several researches in the field of processor-memory bandwidth. Brian M. Rogers et. al. [90] developed a mathematical model to evaluate the impact of memory bandwidth on CMP scaling in different technologies. However, the authors focus only on the theoretical studies in this work. In [114], the organization and performance of 3D memory in NoC are analyzed. They assumed a simple NoC model with uniform random traffic and local traffic. Gabriel H. Loh presented a novel 3D-stacked memory architecture for CMP [71]. It is claimed that a 1.75x speedup is achieved over previous approaches. Nevertheless the paper presumed a conservative quad-core configuration.

In this section, however, we investigate the empirical design of 3D NoC with memory on chip. By integrating the memory module on chip using 3D IC technology, overall system performance is expected to improve due to reduced latency and increased bandwidth. We model a 64-core 3D NoC with 3D on-chip DRAM memory, analyze the memory bandwidth and latency with different memory sub-system implementations, present the performance of our proposed approach and traditional system using a full system simulator.

3.2.1 Modeling of the 3D NoC

It is expected that since the processors consume overwhelming majority of power in a chip, stacking multiple processor layers could be unwise for heat dissipation. Therefore, a feasible design method is to have only one processor layer. Unlike the previous section, here, we model the 3D NoC with 32nm fabrication technology. The total area of Sun SPARC chip is 396mm^2 with 65nm fabrication technology. Scaled to 32nm, each core has an area of 3.4mm^2. Results from CACTI show that the total area of cache banks is 204.33mm^2, for a 64MB cache with 64 banks, 64-bit line size, 4-way associative under 32nm. Each cache bank, including data and tag, occupies 3.2mm^2. We also simulate the characteristics of a 1GB, 8 banks, 32nm DRAM memory by CACTI [94]. It is revealed that the total area of the memory is 212.79mm^2.

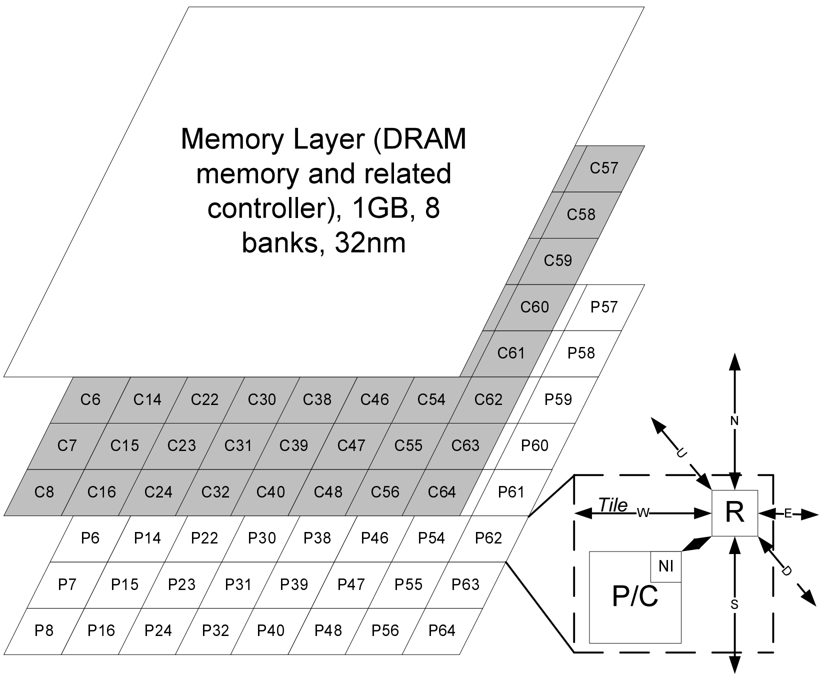

Memory Layer (DRAM memory and related controller), 1GB, 8 banks, 32nm
C57
C58
C59
C60
C61
C62
C6 C14 C22 C30 C38 C46 C54
C7 C15 C23 C31 C39 C47 C55 C63
C8 C16 C24 C32 C40 C48 C56 C64
P57
P58
P59
P60
P61
P6 P14 P22 P30 P38 P46 P54 P62
P7 P15 P23 P31 P39 P47 P55 P63
P8 P16 P24 P32 P40 P48 P56 P64
Tile
W
R
E
N
S
U
D
NI
P/C

Figure 3.8: 3D NoC with one processor layer (Px), one cache layer (Cx) and one memory layer, layers are fully connected by through silicon vias (TSVs, not shown in figure).

On account of the aforementioned analysis, we use a 3D NoC model with one layer of processor, one layer of cache and several layers of memory. In consideration of heat dissipation, the processor layer should be close to the heatsink. The top layer is a 8×8 mesh of Sun SPARC cores. The cache layer has a 8×8 mesh of cache banks. It is noteworthy that routers are quite small compared with processors and cache banks, e.g. scaled to 32nm, as we calculated, a 7-port 3D router is estimated to be only 0.096mm^2. As mentioned earlier, not all routers in a 3D NoC require seven ports, e.g. router of P8 in Figure 3.8 has only East, North, Local PE and Up ports. The total area of the chip is supposed to be around 230mm^2. Figure 3.8 shows the above-mentioned 3D NoC with three layers, however more layers of memory can be stacked. Notice that processor-cache and cache-memory are fully connected by TSVs.

3.2.2 Analysis of the Impact of On-chip DRAM

The different designs of traditional off-chip memory system are shown in Figure 3.9. In Figure 3.9a, both the memory controller and the memory are off-chip with only one memory channel. This was a default configuration with early CMP systems. When reading data from or writing data to the memory, a transmission delay is incurred. The delay consists of two parts: the delay between processor and memory controller, and the delay between memory controller and DRAM module. For modern systems these delays are usually hundreds of cycles (e.g. 200-300). Figure 3.9b illustrates a CMP system with on-chip memory controller and dual channel DRAM memory. The latency between processor cores and memory controller is reduced significantly while the memory bandwidth is doubled. By increase the number of memory controller, as shown in Figure 3.9c, the performance of memory sub system can be improved further, notwithstanding this configuration is used rarely due to pin count limitations.

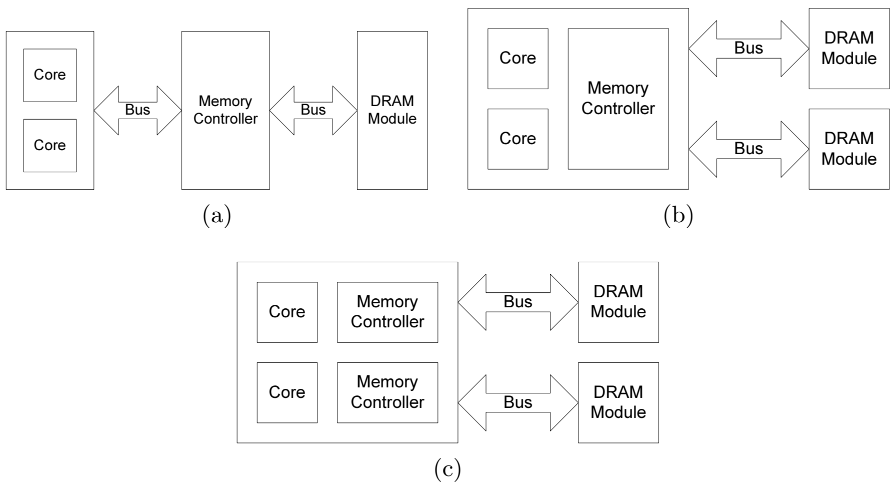


Figure 3.9: Compare of different processor and memory sub system organization.

To analyze the effect of memory architecture to a NoC, we first consider a smaller 5×5 mesh with the focus on network latency which is one of the most important performance factors for NoCs. A SystemC based cycle accurate NoC simulator Noxim [80] has been extended. We use workload trace of FFT from SPLASH-2 [116]. The trace has 2.11M packets, with 78.79M cycles executed. We gather the trace from Simics [73] configured as a 5×5 NoC. The NoC has 25 PEs, in which each PE has a private L1 cache and a shared L2 cache, the two memory controllers are attached in the center.

Table 3.3: Highest network latency for a NoC node

64	36	45	31	40
44	32	25	39	54
45	98	83	48	28
43	50	46	51	66
45	56	62	44	65

Table 3.3 shows the highest network latencies for different nodes. Obviously, the two central nodes have the highest network latency (98 and 83, compared with 25 to 66 of other nodes), due to the concentrated memory traffic from all nodes. The performance of the NoC could be affected with higher latencies. The memory sub-system might be a limitation of scalability in future multicore systems. Memory-on-Chip is a feasible way to break the bottleneck of the memory sub-system. Here, we explore the following approaches.

Memory data bus

A standard single-channel DDR2 SDRAM has a bus width of 8 bytes. Dual-channel technology utilizes two memory channels which result in a 16 bytes bus width, and double the memory bandwidth. Intel Core i7 brings triple-channel architecture, with 24 bytes bus width. It is noteworthy that pin-count grows with channel-count, 373 of 1366 pins in the Intel Core i7 processor are dedicated to one memory controller with three channels [50]. By taking the bus completely on-chip, a much wider bus, e.g. 64 bytes, with the same size of cache line, is possible and the bandwidth improves significantly.

Frequency of processor-memory bus

The frequency of off-chip memory bus is quite slow comparing with common processor frequencies. The bus is used for the communication between on-chip memory controller and memory. In the era of Intel Pentium, the frequency of the processor was 66 to 200 Mhz, and at the same time the SDRAM itself was 66 to 133 Mhz. However, the frequency of a modern processor could be over 3Ghz, while even with DDR2/3 SDRAM, the clock does not grow so much. The typical frequency of a DDR2/3 SDRAM is 100 to 266 Mhz (200 to 533 Mhz for DDR2 and 400 to 1066 Mhz for DDR3, due to dual/quadruple clock rate). Higher bus clock rate is not feasible due to power and signal noise limitations with current technology. It is possible to achieve core clock frequency for the processor-memory bus, with 3D stacked memory design. More than ten times of bus bandwidth is predicted.

Memory access latency (MAL)

By stacking multiple layers of DRAM onto a 3D chip, access latencies are expected to reduce due to shorter wire lengths. DRAM is organized into a grid of single-transistor bit-cells, and the grid is divided into rows and columns. On the higher level, a DRAM bank consists of the grid and accompanying logic. A DRAM rank consists of several banks. When the memory controller accesses data in a DRAM, t_{RCD} (the number of clock cycles needed between a row address strobe and a column address strobe), t_{CAS} (the number of clock cycles needed to access a column) and t_{RP} (the number of clock cycles needed to precharge a row) are major factors. For a DDR-400 memory, the bus frequency is 200Mhz, each cycle takes 5ns, the typical number of clock cycles for t_{RCD}, t_{CAS} and t_{RP} are 3, 3 and 3 respectively (15ns each). By stacking the DRAM ranks in a 3D fashion, the length of internal buses and bitlines are reduced, and hence the access latencies of the memory are reduced.

Equation 5 $t_{MAL} = t_{Bus\_delay} + t_{DRAM\_delay}$

As aforementioned, the area of a 1GB DRAM module is about 212.79mm^2 under 32nm technology, therefore the length of a side for the square module is 14.58mm. Figure 3.10 depicts that a request traveling from module 1 to 4 in 2D off-chip memory will take at least 29.17mm of wire length, by going through module 2 and 3. In 3D on-chip memory, since the distance between stacked layers are shorter, around $50\mu m$, the wire delay between multiple layers can be neglected. Researches have shown that based on this architecture, memory access time has improved by 32% [71].

The latency for an off-chip DRAM is typically 200-300 cycles. In a system with 2Ghz processor, the time is 100-150 ns. By bringing the DRAM on-chip, this latency can be reduced significantly, thus we ignore this latency. According to Equation 5, the total latency from memory controller to DRAM will be reduced from (250+9×10) = 340 to (0+9×10×0.68) = 61.2.

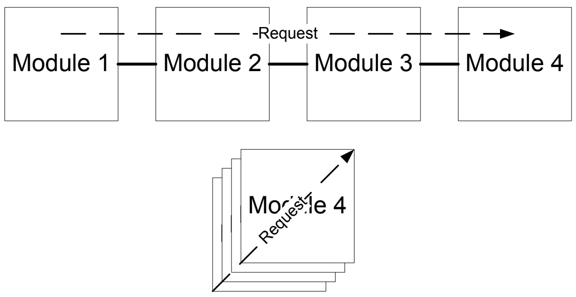


Figure 3.10: 2D off-chip memory and 3D on-chip memory organizations.

Memory controllers

Many conventional architectures employ a limited number of memory controllers due to pin count limitations. With only one memory controller, 373 of 1366 pins in the Intel Core i7-900 processor are dedicated to that [50]. The ratio between the number of cores and the number of memory controllers is 6:1 (Table 3.4). The Tilera Tile64 processor [115] implemented a 2D 8×8 mesh with four on-chip memory controllers and off-chip memory architecture. The ratio between core and memory controller is thus 16:1 (Table 3.4). It is not realistic to have a memory controller for each PE in 2D architectures. However, for 3D stacked DRAM NoC, since die layers can be connected with layer-layer TSVs [125], one memory controller per core is possible. The number of transistors required for a memory controller is quite small compared with billions of total transistors for a chip. It is presented that a DDR2 memory controller is about 13,700 gates with application-specific integrated circuit and 920 slices with Xilinx Virtex-5 field-programmable gate array [38].

Table 3.4: Comparison of processors with memory controller and memory channel

Processor	Core	Memory controllers
AMD MagnyCours	6	1 DDR3, 533Mhz (133x4), 2 channels
Intel Nehalem	4	1 DDR3, 533Mhz (133x4), 3 channels
IBM Power 7	8	2 DDR3, 533Mhz (133x4), 8 channels
Tilera Tile64	64	4 DDR2, 200Mhz (100x2), 1 channel

Mixing of all techniques

With higher bus frequency, wider bus width, shorter wire length and more memory controllers, memory bandwidth can be improved significantly.

Equation 6 *Bandwidth = Clock × Data Rate × Rate Multiplier × Bus Width × Channel × Controller*

The bandwidth of memory is defined in Equation 6. According to that, the bandwidth of a modern single-channel single-controller DDR2 memory with 200Mhz bus frequency is 200×2×2×8×1×1 = 6.4GB/s. By stacking the memory on-chip, with native clock rate of the core (2Ghz), 64-byte bus width and 64 memory controllers, the theoretical maximum bandwidth would reach: 2000×1×2×64×1×64 = 16,384GB/s! It is noteworthy that the memory runs in synchronous mode, i.e. the memory and the I/O bus are with the same frequency. We observed that the 3D stacked DRAM has a lower power consumption comparing with off-chip DRAMs, due to that 3D on-chip connection is more power efficient than off-chip bus I/Os. Higher frequencies can be achieved with lower power consumption, or lower frequencies for power constrained applications. Table 3.5 shows the comparison of memory sub system of modern systems and our proposed system.

Table 3.5: Memory sub system configurations for different processors

Processor	**Typical BW**	**BW per core**	**Latency**
AMD M-Cours	17.1GB/s	2.85GB/s/core	250+(7-7-7)
Intel Nehalem	25.6GB/s	6.4GB/s/core	250+(7-7-7)
IBM Power 7	136.8GB/s	17.1GB/s/core	250+(7-7-7)
Tilera Tile64	12.8GB/s	0.2GB/s/core	250+(3-3-3)
Our proposed	**16,384GB/s**	**256GB/s/core**	**0+(2-2-2)**

3.2.3 Experimental Evaluation

In this section, we present the experimental evaluation under different memory configurations. Applications are selected from SPLASH-2 [116]. The simulation platform is based on a cycle-accurate 3D NoC simulator which can produce detailed evaluation results (Section 2.3). We use a 128-node network which models a single-chip CMP for our experiments. The 3D architecture here has one layer for processors, one layer for shared cache memories and five layers of DRAM memory (one layer for logic) (for simplicity, Figure 3.8 shows only three layers). A full system simulation environment with 64 processors and 64 L2 cache nodes has been implemented.

The size of each cache bank node is 1MB. MOESI cache coherence protocol is used. Orion [111], a power simulator for interconnection networks, is used to evaluate detailed power characteristics. A wormhole router is modeled in Orion, with corresponding input/output ports, buffers and the crossbar. Power consumption of routers is analyzed.

The normalized full system simulation results are shown in Figure 3.11 and 3.12. As is shown in Figure 3.11, our proposed design outperforms the traditional design in terms of average link utilization. Average link utilization is calculated with the number of flits transferred between NoC resources per cycle. Under the same configuration and workload, lower utilization means mitigated network load, which is favorable. Comparing with the traditional design, the average link utilization for our proposed design is reduced by 10.25%, on average. FFT and Cholesky have the most significant reduction of average link utilization, 13.34% and 12.81% respectively.

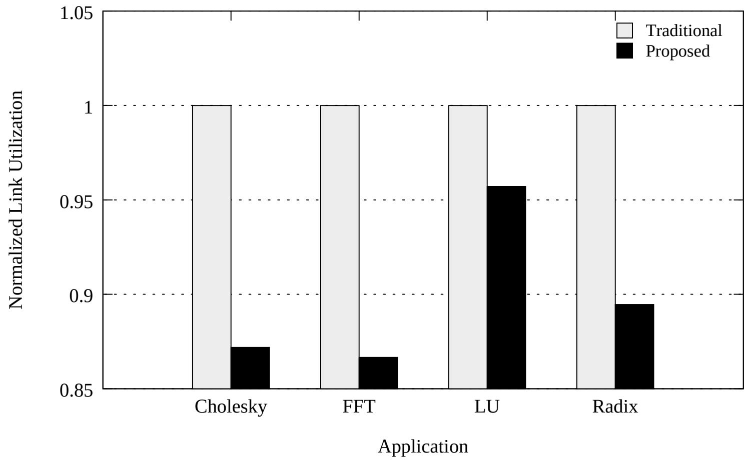


Figure 3.11: Normalized average link utilization with different configurations.

The results in Figure 3.12 show that our proposed design outperforms the traditional design in terms of executed cycles under all workloads. On average, our proposed design costs 1.12% less cycles than the traditional design, and the cycle reduction reaches 2.29% for LU workload and 1.77% for Radix respectively. The improvements of executed cycles can be interpreted as the result of the increased memory bandwidth and reduced memory access latency which commensurate with the number of memory accesses. The improvement of executed cycles is less remarkable comparing with average link utilization since local operations (e.g. core and cache) are not related with network operations. We notice that the bandwidth improvement enables just a modest performance gain in application level.

This is due to the fact that most applications are not sensitive to memory bandwidth. However, future applications may have higher requirements to memory bandwidth and therefore can benefit more from this design.

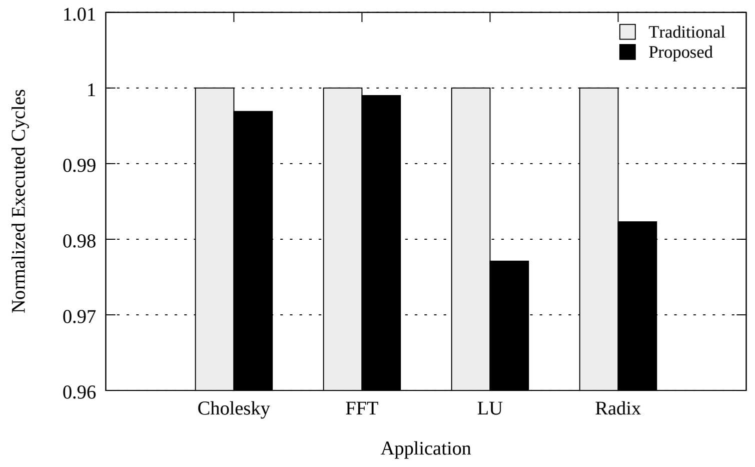


Figure 3.12: Normalized executed cycles with different configurations.

Chapter 4

Resource Placement

Future multicore processors is expected to have hundreds of even thousands of cores. In such a huge system, there are many hardware resources, e.g. cores, caches, routers, links, memory controllers and so on. It might be not economical to equip each node with a hardware resource. The utilization of resources can be low, and therefore leading to poor system efficiency. To improve system efficiency, one solution is to distribute a limited number of a certain resource. In this case, multiple requesters have to share a resource, leading to possible traffic contention and/or performance bottleneck. To alleviate the problem of performance degradation by reduced amount of resources, intelligent placement of resources is introduced, so that the optimal position of the resource is determined. We study optimal resource placement in a mesh-based NoC. Three hardware resources are used as case studies: Through Silicon Via (TSV), memory controller and core/cache.

In Section 4.1, theoretical background of resource placement is introduced. In Section 4.2, a generic "Divide and Conquer" method for solving the optimal resource placement problem is proposed. In Section 4.3, we analysis the impact of TSV placement in 3D NoCs. The adoption of a 3D NoC design depends on the performance and manufacturing cost of the chip. Therefore, a study of TSV, that connects different layers of a 3D chip, is crucial. We discuss the number of TSVs required for a 3D NoC. Different placements of layer-layer connections are explored. We configure two 3D NoCs: (*i*) a 4×4 mesh with five layers, and (*ii*) an 8×8 mesh with two layers. Experiment results show that under different workloads, the average network latencies of (*i*) in two different configurations are reduced by 5.24% and 2.18% respectively, compared with the design with quarterly connected TSVs. The average network latencies of (*ii*) in two different configurations are reduced by 14.78% and 7.38% respectively, compared with the alternative design. With optimal placement of TSVs, the vertical connections are reduced significantly with acceptable latency penalty.

In Section 4.4, we analyze and compare different placements of memory controllers for CMPs. As the number of cores increases, the memory bandwidth between on-chip components and off-chip memory has become a critical problem. The integration of more memory controllers on chip is one feasible way to solve this problem. However, the physical location of memory controllers in a mesh-based NoC have a significant impact on system performance. We investigate the placement of multiple memory controllers in an 8×8 NoC. Several metrics have been analyzed. An optimal memory controller placement is found and evaluated. By using applications selected from SPLASH-2, PARSEC, TPC and SPEC as benchmarks, it is shown that the average network latency, average link utilization and performance power product in our optimal placement are reduced by 7.63%, 10.44% and 13.94% compared with the conventional two-sides placement, respectively. The goal is to give a solid theoretical foundation to future CMP design.

In Section 4.5, optimal core and cache placements for modern CMPs is studied. As the number of cores increases, traditional on-chip interconnects such as bus and crossbar suffer from poor scalability and low efficiency. Ring based design has been proposed and implemented to mitigate these problems. However, the continuing growth of number of cores will render the ring interconnect infeasible. Network based designs are therefore proposed for future CMPs for better scalability. We explore the interconnect of a state-of-the-art CMP. We analyse and compare the implementation of the ring-based and the network-based interconnect. The placement of cores and caches in a network is proved crucial for system performance. We investigate optimal core/cache placement for CMPs. Results show that by using the optimal network interconnect, compared with the ring interconnect, the average network latency and execution time are reduced by 11.93% and 19.53% respectively, for four configurations and two applications.

4.1 Theoretical Background

The different placement of hardware resources can have a dramatic impact on the performance of the NoC. Here, for theoretical analysis, we define hop count (Manhattan Distance, or MD) as the number of routers a flit has to go through from a requester to a hardware resource. Average Hop Count (AHC) is selected as a metric in evaluating the performance of a placement configuration. AHC is calculate as, the average number of the minimal number of hops to a hardware resource, for all potential requesters. Figure 4.1 shows a worst case placement of 16 hardware resources in an 8×8 mesh. In this configuration, all the resources are placed on one side of the chip, causing high delays in the other side of the chip (AHC = 168/64 =

2.625, 168 is the accumulation of all MDs). Furthermore, due to the poor distribution of pillars, there could be traffic contentions.

P	P	P	P	P	P	P	P
P	P	P	P	P	P	P	P
1	1	1	1	1	1	1	1
2	2	2	2	2	2	2	2
3	3	3	3	3	3	3	3
4	4	4	4	4	4	4	4
5	5	5	5	5	5	5	5
6	6	6	6	6	6	6	6

Figure 4.1: A placement of 16 resources (grey nodes with P) in an 8×8 NoC, white nodes are requesters, number means minimal hops required for a requester to reach a resource.

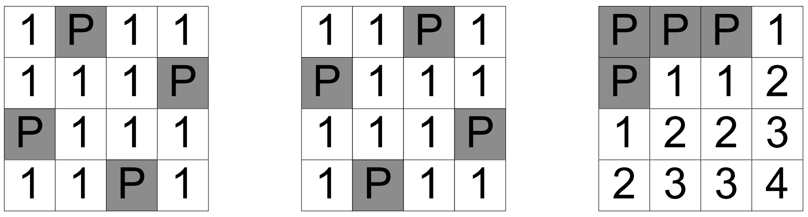
To find a placement with minimum AHC, exhaustive simulation is a possible solution. However, it is noteworthy that this method is only feasible for a small size NoC, while a heuristic-guided search is required to deal with the computational complexity from a larger search space [2], e.g. an 8×8 mesh with a quarter of limited resources has $\binom{64}{16} = 488{,}526{,}937{,}079{,}580$ different placement possibilities, $\binom{64}{32} = 1.83\text{e}{+}18$ for half resource numbers!

The situation may become more complex in certain conditions, e.g. the memory controller. We note that the memory traffic in modern systems are distributed to all the controllers evenly, according to the physical addresses. In this case, calculating the average MD from a node to all the controllers is more meaningful. In the following sections, we propose two methods in solving the resource placement problem for a mesh on-chip network.

4.2 Divide and Conquer

The divide and conquer algorithm is an essential algorithm in computer science. It works by breaking down a problem into multiple sub-problems, until these sub-problems can be solved within reasonable computation time. The results of the sub-problems are then combined, giving the solution of the original problem. The divide and conquer method has been used in many problems, such as quick sort, fast Fourier transform and map reduce. However, this technique is not possible for all problems, and it requires analysis to prove the effectiveness and efficiency of the algorithm.

As aforementioned, the design space for placing 16 resources in an 8×8 mesh is too large. We explore the possibilities starting from a smaller NoC by writing a program that exhaustively enumerate all placement possibilities and output the combinations of lowest hop count (Code 2). It is noteworthy that because of the huge design space, the time required for exhaustive simulation is practical for a NoC below 6×6. For larger NoCs, we divide it into several smaller NoCs and solve them (divide and conquer), e.g. an 8×8 NoC with 16 pillars is divided into four 4×4 NoCs, each with 4 pillars. The design space is then much smaller ($\binom{16}{4} = 1{,}820$), and the time required for enumeration process is feasible. After enumerating a 4×4 mesh, we found that there are 7,129 configurations that satisfy the lowest average hop count for half resource configuration (8 resources), 2 configurations that satisfy the lowest average hop count for quarter resource configuration (4 resources). Figure 4.2 shows 3 representative designs for best case and worst case resource placement. We note that the two best case placements are actually the same, and there are several other equal worst case placements.



(a) 16/Best1 (b) 16/Best2 (c) 16/Worst

Figure 4.2: Different placements of 4 resources in a 4×4 NoC.

4.2.1 Placement for 16 Resources

By expanding the 4×4 mesh four times, we select the design of Figure 4.2a as our placement strategy for 16 resources. Figure 4.3 shows this configuration. In this configuration, all non-native PEs (those not directly attached to a resource) require just one hop to access a resource, which results a minimal average hop count ($AHC = 48/64 = 0.75$). It is notable that the resources are well distributed, e.g. there are only two resources per row/column and no adjacent resources, providing a good traffic variance.

4.2.2 Placement for 8 Resources

We use the same metrics as aforementioned to evaluate the optimal placement of 8 resources theoretically. An 8×8 NoC with 8 resources is divided into four 4×4 NoCs, each with 2 resources ($\binom{16}{2} = 120$). After enumeration,

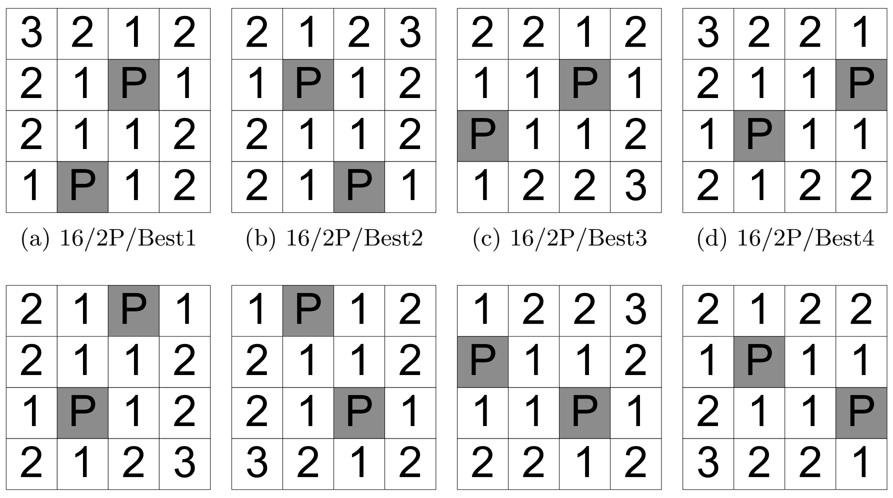
Code 2 The pseudo code of calculating average Manhattan Distance between a requester and a resource, for a combination of resource placement.

```
void calculate_amd()
{
 for(i=0;i<Combinations;i++)  //Calculate for all combinations
 {
  for(j=0;j<R_Num;j++) R[j]=R_Comb[i*R_Num+j];
                              //R_Comb is an array of all resource
                                placement combinations
  for(k=0;k<Net_x*Net_y;k++)  //Calculate for all requesters in the mesh
  {
   min_dist=Max_Hop;
   for(l=0;l<R_Num;l++)       //Calculate all resources
   {
    tmp=MD(k, R[l]);          //Calculate the MD from a requester to a
                                resource
    if (min_dist>tmp) min_dist=tmp;
   }
   distance[k]=min_dist;      //Find the minimal distance for a requester
  }
  for(m=0;m<Net_x*Net_y;m++) total_dist+=distance[m];
  final_amd[i]=total_dist/(Net_x*Net_y);
                              //Final average Manhattan Distance for a
                                combination
  total_dist=0;
 }
}
```

1	P	1	1	1	P	1	1
1	1	1	P	1	1	1	P
P	1	1	1	P	1	1	1
1	1	P	1	1	1	P	1
1	P	1	1	1	P	1	1
1	1	1	P	1	1	1	P
P	1	1	1	P	1	1	1
1	1	P	1	1	1	P	1

Figure 4.3: An optimal placement of 16 resources (P) in an 8×8 NoC.

we found that there are 8 configurations that satisfy the lowest average hop count for 2 resources. Figure 4.4 shows these configurations.



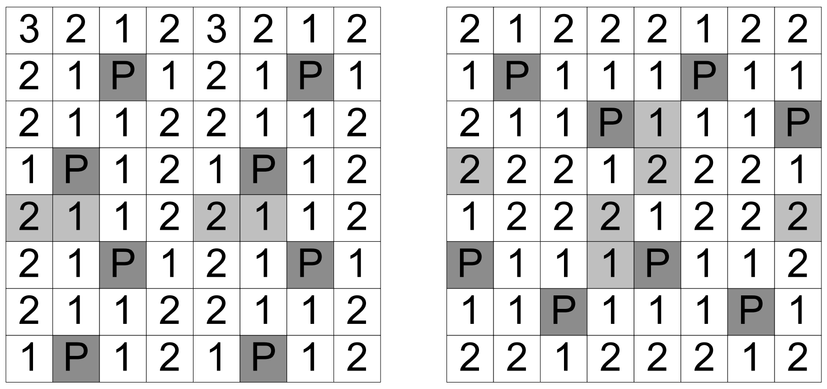
(a) 16/2P/Best1 (b) 16/2P/Best2 (c) 16/2P/Best3 (d) 16/2P/Best4

(e) 16/2P/Best5 (f) 16/2P/Best6 (g) 16/2P/Best7 (h) 16/2P/Best8

Figure 4.4: Different placements of 2 resources in a 4×4 NoC.

We expanded the optimal 4×4 placement four times in the 16-resource configuration, however it is more complex for the 8-resource configuration. Figure 4.5a shows a placement with four duplications of Figure 4.4a. The minimal hop count required for four requesters to reach a resource is reduced by one for the lower two duplications (light-gray numbers). However, by combining different placements of Figure 4.4, the hop counts reduction can be even larger. Figure 4.5b shows a placement, in which two duplications of Figure 4.4h are placed on top, and two duplications of Figure 4.4g are placed on bottom. Thus, hop count is reduced by one in six nodes, improving the placement of Figure 4.5a. Furthermore, other metrics in Figure 4.5b are better than in Figure 4.5a. For instance, in Figure 4.5a, there are two resources in all even rows/columns. However, in Figure 4.5b, resources are well distributed in the vertical axis. For a dimensional ordered deterministic routing, e.g. XYZ routing, better distribution of resources in all directions can alleviate traffic contention on resources. We note that to find the best placement, the problem can be described as: Choose 4 placements from Figure 4.4, the order of choice is important, and repetition is allowed. The design space is reduced to $8^4 = 4,096$.

With exhaustive simulation of 4,096 combinations, we discover that Figure 4.5b represents one of the best placement for 8 resources in an 8×8 mesh. Table 4.1 shows the computation complexity of exhaustive enumeration and our “divide and conquer” method. Our idea can be expanded to NoCs with regular sizes and regular number of resources, e.g. an 8×8 mesh with 4 resources (divided into 4 meshes, each 4×4 with 1 resource), a 9×9



(a) Placement 1 (b) Placement 2

Figure 4.5: Two placements of 8 resources in an 8×8 mesh.

mesh with 18 resources (divided into 9 meshes, each 3×3 with 2 resources). NoCs with irregular sizes and irregular number of resources, e.g. a 13×13 mesh with 53 resources, are not suitable for our method.

Table 4.1: Comparison of computation complexity of exhaustive enumeration and "divide and conquer".

Problem	**Exhaustive**	**Divide**	**Conquer**
8×8, 16 Resources	4.885e+14	1,820	16
8×8, 8 Resources	4.426e+9	120	4,096

4.3 Placement for Through Silicon Via

Through Silicon Via (TSV) is the most promising solution for building 3D chips. There are several types of TSVs, e.g. data transmission, control, power distribution and thermal dissipation. Here, a pillar is defined as a bunch of TSVs, including TSVs for data, control and power distribution. On the assumption that the power supply voltage is 1V, a practical aspect ratio for TSVs is between 10:1 to 5:1, in which data transmission TSVs are dominant ones [118] [7].

Obviously, maximum performance can be achieved by full layer-layer connection, i.e. all routers between layers are connected by pillars. However, as the number of tiles grow, it might not be practical to assume that each tile will be connected with corresponding TSVs because of the limitation of manufacturing cost and chip area. Assuming that a flit in a NoC is 128 bits, full layer-layer connection for a 4×4 NoC would require 128×4×4 = 2,048 TSVs for parallel data signals, full layer-layer connection for an 8×8 NoC would require $128 \times 8 \times 8 = 8,192$ TSVs for parallel data signals.

In addition, other TSVs are required for control, power distribution and thermal dissipation.

4.3.1 Cost Considerations

There are three major tasks of 3D integration processing, *a*) TSV formation for vertical interconnects; *b*) wafer thinning and backside processing; and *c*) stacking of chips. It is shown in [109] that based on IMEC 3D cost model, TSV processing cost is the dominating cost for a 3D wafer. Assuming a CMOS processing technology of 65nm with 200mm silicon wafers, 46% to 65% costs are spent on TSV processing, with annual production volume varies from 10,000 to 300,000 wafers.

There are different integration methodologies for the implementation of TSVs. The manufacturing process determines integration density and cost. Two main methods, namely via middle (VM) and via last (VL), are compared. In VM, the TSVs are fabricated before the interconnect metal layers and after the devices. In VL, the TSVs are fabricated by processing the back side of the wafer after wafer-thinning. Based on the aforementioned technologies, the diameter of a TSV processed by VM is around 5μm compared with 35μm by VL, the pitch between TSVs processed by VM is around 10μm compared with 60μm by VL. Although higher density can be achieved by VM, the processing cost of VL is considerably lower [109]. It is demonstrated that the manufacturing cost raises significantly beyond 1,000 TSVs per chip stack for VL, while it raises significantly beyond 10,000 TSVs for VM [109]. Another research conducted by the International Technology Roadmap for Semiconductors (ITRS) have shown that the maximum feasible number of TSVs in a high performance 3D chip will reach 1,000 in 2012, and increase by 1,000 in every two years [7]. Obviously, in terms of chip area and manufacturing cost, the total number of TSVs should be limited to a practical value.

It is shown that the yield for processing each TSV is about 99.99% under current technology [63]. The yield is acceptable for a smaller number of TSVs. As the number of TSVs grows, e.g. 10,000 TSVs per chip, the possibility of a faulty chip becomes higher. Figure 4.6 shows the accumulated yield for processing TSVs for a perfect chip die. It is depicted that the yield is only 36.79% for 10,000 TSVs. As the number of TSVs reduces to 2,000 or 1,000, the yield is much higher (81.87% and 90.48% respectively). We deduce that the die yield will reduce with the increasing number of TSVs manufactured on a chip. The total manufacturing cost for building a 3D chip can be unacceptable for the market with full TSV connection between layers.

Here, in order to reduce the number of total pillars and evaluate the placement for fewer pillars, we choose 16 and 8 pillars for an 8$\times$8 mesh

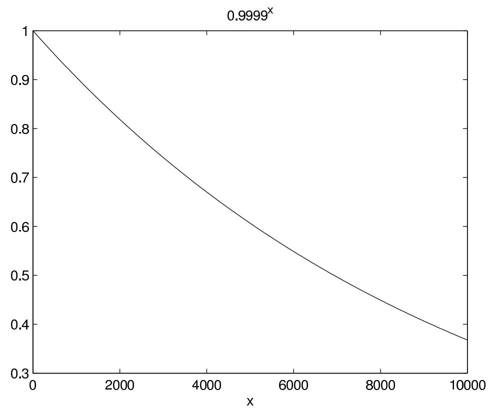


Figure 4.6: The TSV yield with increasing TSV numbers (x).

network (one fourth and one eighth of total nodes). The total number of TSVs in these two configurations is expected to be between 1,000 and 2,000.

4.3.2 Performance with Reduced Number of TSVs

It is cheaper and easier to manufacture a 3D chip with fewer pillars between layers, however the performance of the chip is affected. Multiple nodes have to share a pillar, high congestion could be created on a pillar, leading to communication bottlenecks. To minimize traffic contention between layers, the placement of pillars should be considered carefully. Figure 4.7 presents half and quarter TSV connections between layers.

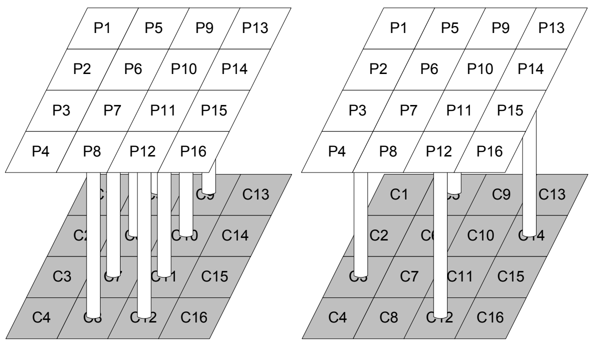


Figure 4.7: 3D NoC with layers half and quarter connected by TSVs.

Figure 4.8 shows the comparisons of the average hop counts among different architectures. Although the maximum and minimum hop count for all configurations are the same (10 and 1), the average hop counts in half and quarter configurations are respectively 5% and 11.875% higher than full TSV configuration. Average network latency, one of the most important factors of a NoC, is affected by the average hop count. Besides, fewer TSVs will increase contention of inter-layer communication; the situation can be fatal with a high flit injection rate. Detailed simulation analysis will be given in the following section.

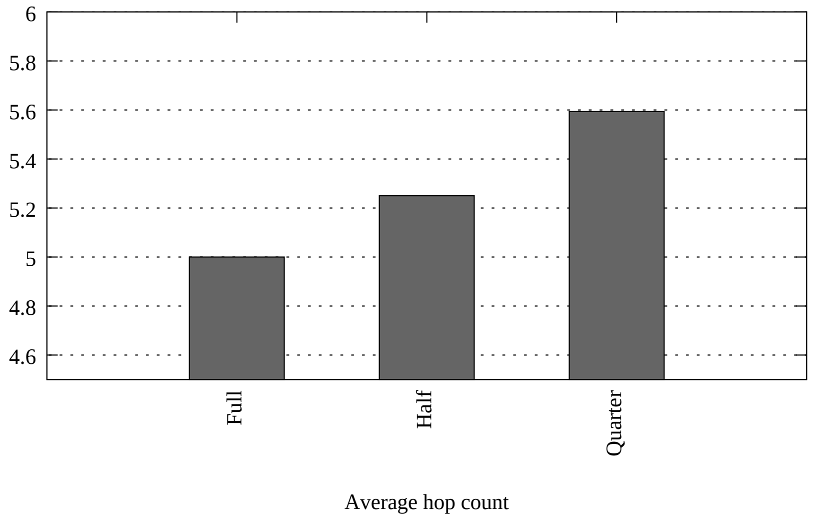


Figure 4.8: The comparison of average hop count among full, half and quarter connected layers.

4.3.3 3D NoC Models

Here, we use two 3D NoC models. One of them has five layers, based on 65nm processing technology. The top layer is a 4×4 mesh of Sun SPARC cores, each core is 14mm^2 based on 65nm technology. Other four layers are cache layers. The total area of a 64MB cache with 64 banks, 64-bit line size, 4-way associative under 65nm is estimated to be 843.50mm^2. Each cache bank, including data and tag, occupies 13.18mm^2. Therefore, each cache layer has a 4×4 mesh of cache banks. Because of routers are quite small, e.g. a 7-port 3D router under 90nm is estimated to be 0.76mm^2 [82], total area of the chip is supposed to be below 300mm^2.

Another 3D NoC model has two layers, based on 32nm processing technology. The top layer is an 8×8 mesh of 64 Sun SPARC cores, each core has an area of 3.4mm^2. The area of cache similar as aforementioned under 32nm is 204.33mm^2. Each cache bank, including data and tag, occupies 3.2mm^2. The cache layer has an 8×8 mesh of cache banks. It is noteworthy that routers are quite small compared with processors and cache banks, e.g. scaled to 32nm, as we calculated, a 7-port 3D router is estimated to be only 0.096mm^2. The total area for one layer of the 3D NoC is supposed to be below 300mm^2.

4.3.4 Evaluation for 4×4 NoC with Five Layers

In this section, we present the experimental evaluation under different number and placement of pillars, for a 4×4 NoC with five layers. Applications are selected from SPLASH-2 [116] and PARSEC [18]. SPECjbb [98] and TPC-H [104] are used as synthetic benchmarks.

We use an 80-node network which models a single-chip CMP for our experiments. The 3D architecture has five layers, in which the first layer is processors, other layers are shared cache memories. A full system simulation environment with 16 processors and 64 L2 cache nodes has been implemented. The MESI cache coherence protocol is used. Orion [111], a power simulator for interconnection networks, is used to evaluate detailed power characteristics. A wormhole router is modeled in Orion, with corresponding input/output ports, buffers and the crossbar. Power consumption of routers is analyzed.

The normalized full system simulation results are shown in Figures 4.9 and 4.10. As illustrated in Figure 4.9, the design of full pillar connection outperform others in terms of network latency. The improvement is more notable in Cholesky (chol), FMM and TPC-H, with 9.69%, 8.04% and 8.8% reduced latency, respectively, compared with the quarter design. This is primarily due to the reduced hop count of processor-cache data access in the full pillar connection than the other designs. We notice that performance difference for some of the applications, e.g. FFT and Radix (radi), is not that significant, the network latency is reduced by 1.09% and 2.93% respectively comparing with the quarter design. The reason is that these applications have a lighter network load compared with the other applications. On average, compared with the quarter design, the network latency is reduced by 5.24% and 2.18% for full and half designs, respectively.

As a combination of both performance and power consumption, the Performance Power Product (PPP) is a more meaningful metric because performance is usually a trade-off for power consumption. Performance is measured in terms of application execution time. Systems with a lower PPP (PPP > 0) generally have a better trade-off between the performance and power. The results in Figure 4.10 show that the PPP improves more significant from quarter pillar design to full pillar design, when compared to performance improvements. Configuration with full pillars performs 32.42%, 41.65%, 35.82% and 44.34% better than quarter pillars in Cholesky, Ocean, Radix and TPC-H respectively. On average, the PPP is reduced by 25.11% and 15.85% for full and half designs respectively compared with quarter design. We noticed that by using more pillars, the traffic contention between layers is alleviated. Therefore overall network power consumption is reduced with reduced execution time, resulting a significantly improved PPP. However, as aforementioned, the design and implementation of ver-

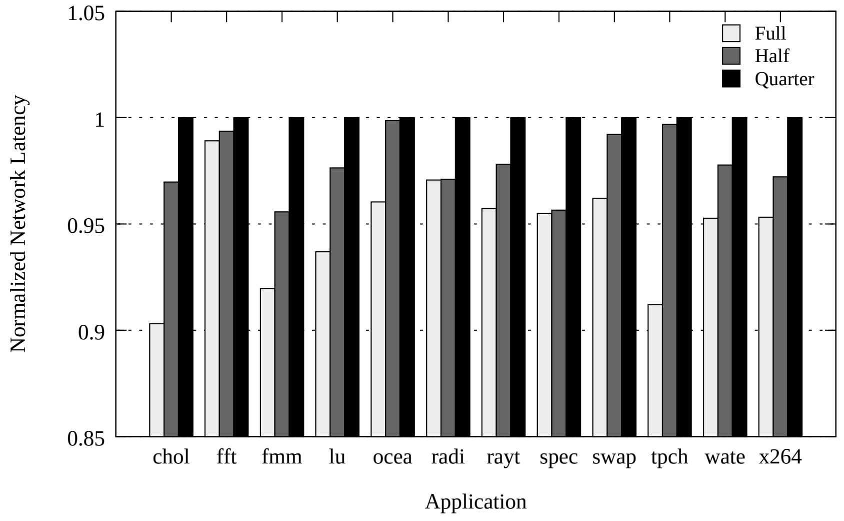


Figure 4.9: Normalized average network latency with different number of pillars, for a 4×4 NoC with five layers.

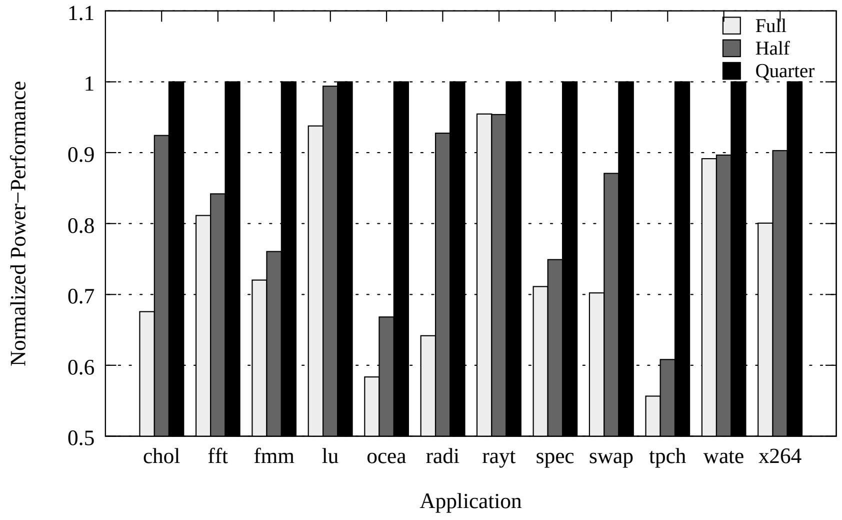


Figure 4.10: Normalized power performance product with different number of pillars, for a 4×4 NoC with five layers.

tical links are very expensive, manufacturing cost is a trade-off for this power-performance improvement. It is always desirable to have a balanced number of vertical links.

4.3.5 Evaluation for 8×8 NoC with Two Layers

Here, we present the experimental evaluation under different number and placement of pillars, for a 8×8 NoC with two layers. Applications are selected from SPLASH-2 [116] and PARSEC [18]. SPECjbb [98] and TPC-H [104] are used as synthetic benchmarks.

We use a 64-core multiprocessor which models a single-chip CMP for our experiments. The 3D architecture has two layers; the first layer is processors, the second layer consists of shared cache memories. A full system simulation environment with 64 processor nodes and 64 cache nodes, has been implemented. MOESI is used in our memory hierarchy. Orion [111] is used to evaluate detailed power characteristics.

The normalized full system simulation results are shown in Figures 4.11 and 4.12. For 16-pillar configuration we use Figure 4.3, for 8-pillar configuration we use Figure 4.5b. Average network latency represents the average number of cycles required for the transmission of all network messages. For each message, the number of cycle is calculated as, from the injection of a message header into the network at the source node, to the reception of a tail flit at the destination node. As illustrated in Figure 4.11, the 64-pillar connection outperforms others in terms of average network latency. The improvement is more notable in Cholesky, Swaptions and x264, with 21.40%, 17.47% and 18.91% reduced latency, respectively, compared with the 8-pillar design. This is primarily due to the reduced average hop count of processor-cache data accesses in the 64-pillar connection compared to the other designs. We notice that performance differences for some of the applications, e.g. LU, Radix, Raytrace and SPECjbb, are less significant, the network latencies are reduced by 12.61%, 12.32%, 12.11% and 10.65% respectively comparing with the 8-pillar design. The reason is that these applications have a lower network load compared with the other applications. On average, compared with the 8-pillar design, the network latency is reduced by 14.78% and 7.38% for 64-pillar and 16-pillar designs, respectively.

The results in Figure 4.12 show that the PPP improves more significant from 16-pillar design to 64-pillar design, when compared to performance improvements. Configuration with 64-pillars performs 37.76%, 37.99%, 44.90% and 38.44% better than 8-pillars in Cholesky, FMM, Ocean and TPC-H respectively. On average, the PPP is reduced by 32.69% and 24.27% for 64-pillar and 16-pillar designs respectively compared with the 8-pillar design. Similar as the previous section, by using more pillars, the traffic contention between layers is alleviated. However, a balanced number of vertical links should be determined by the system designer.

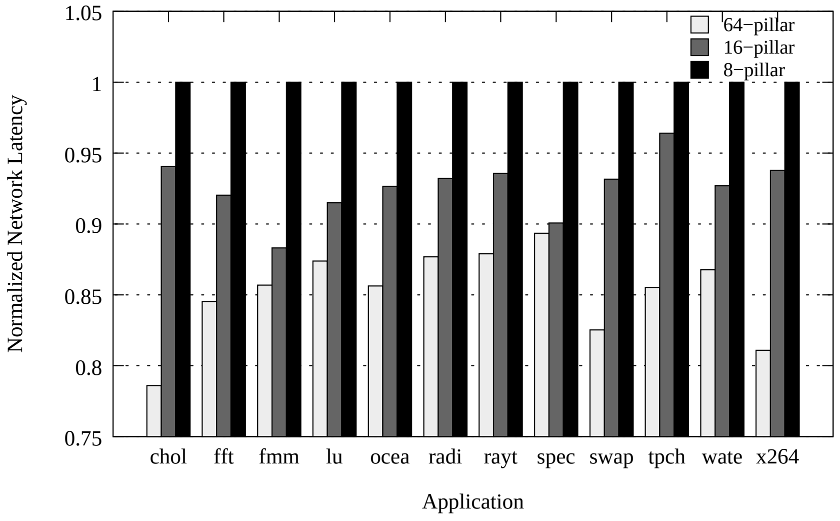


Figure 4.11: Normalized average network latency with different number of pillars, for a 8×8 NoC with two layers.

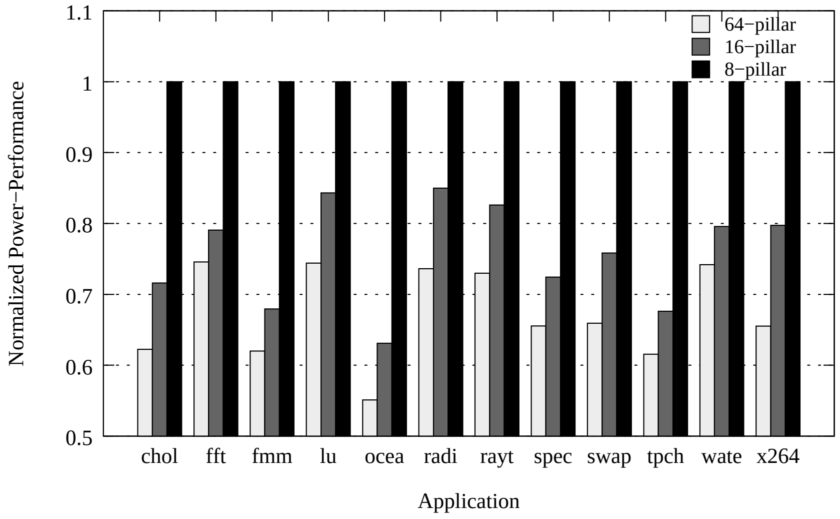


Figure 4.12: Normalized power performance product with different number of pillars, for a 8×8 NoC with two layers.

4.4 Placement for Memory Controller

As the number of processor cores increases, there is a great concern about memory bandwidth, since the number of memory accesses are growing with the number of cores. Intel Core 2 Duo doubled the memory bandwidth requirement to fit the requests of two cores. The system performance will decline if memory bandwidth cannot sustain the rate requested by processor cores. Dual channel and triple channel can be applied to provide double or triple memory bandwidth. However, these configurations require multiple memory modules, which increases cost, fault rate and power consumption. Another way of increasing memory bandwidth is to apply more memory controllers. Nevertheless, the number of memory controllers is limited due to pin count limitations. Intel Core i7-900 processor has only one memory controller, 27.31% of its total pins are dedicated to that [50]. It is not realistic to have a dedicated memory controller for each core.

With a limited number of memory controllers, quality of service support for the on-chip network can reduce the interconnect pressure via efficient distribution of memory traffic [66]. Fair arbitration between requests arriving at the memory controller has been proposed in [76] [77]. The placement of memory controllers can have a significant impact to the system performance and efficiency as well. In [2], intelligent memory controller placement and routing of core-memory traffic have been advocated. Based on their conclusion, the Diamond placement (Section 3) is the optimal placement for an 8×8 mesh with 16 memory controllers. Here, we argue that there are better placement possibilities compared with the Diamond. A good placement of memory controllers should have the following features: *a*) low average hop count for memory messages from all processors; *b*) the memory messages are distributed in the on-chip network evenly, e.g. less hot-spots and congestions. The memory messages are the messages between memory controllers and last level caches, in case of a cache miss event. We propose the "divide and conquer" method for large mesh networks to avoid huge design space. The permutation of all possibilities is not necessary. The metric of average hop count, Distance-m, Adjacent-n, memory controllers per row/column and adjacent memory controllers are analyzed and compared with several placements. We find the optimal placement of 16 memory controllers in an 8×8 mesh theoretically. A general strategy for other mesh sizes are discussed. To confirm our study, we model a NoC with 8×8 mesh, discuss the performance with different placements using a full system simulator, under multiple workloads.

4.4.1 Motivation

In our study, the on-chip memory controllers are shared by all processors to provide large physical memory space to each processor. Each of the controller controls a part of the physical memory, and each processor can access the memory controlled by any of the controllers. A physical address will be mapped to a memory controller according to its address bits and cache line address (See Section 4.4.6 for details). In this case, memory traffic is distributed to all the controllers evenly. This represents a typical design of modern processors [59].

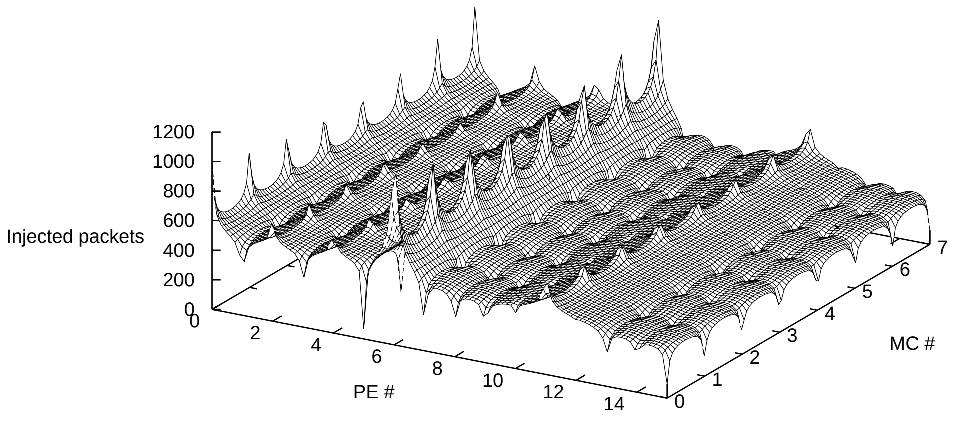


Figure 4.13: Memory request rate for 16-core NoC running FFT.

An unoptimized placement of memory controllers can cause hotspots and traffic contentions/delays. As a result, average network latency, one of the most important factors of a NoC, is increased and overall performance is degraded. Figure 4.13 shows the memory request rate of each processing element (PE) when running FFT in a 16-core NoC under GEMS/Simics simulation environment. The detailed system configuration can be found in Chapter 2 (except for the number of cores and number of memory controllers etc., we use a 4×4 mesh with 16 cores and 8 memory controllers). The traffic trace has generated 59,665 packets, with 22.1M cycles executed. In Figure 4.13, the X axis is the number of PE, the Y axis is the number of memory controllers. The traffic is shown for all the 16 PEs and 8 memory controllers. It is revealed that despite the memory traffic is distributed to all the memory controllers evenly, 73.53% of memory request traffic are generated by five nodes (N6 27.63%, N0 13.27%, N11 11.62%, N4 10.73% and N2 10.28%).

To access the memory controllers, assuming X-Y deterministic routing, Equation 4.1 shows the latency required for a memory request generated by a node (without considering the latencies of the memory controller and the memory itself). The latency involves in-tile links (Between NI and PE, L_{Link_delay1}), router (L_{Router_delay}), tile-tile links (L_{Link_delay2}) and the number of hops required to reach the destination (n_{hop}).

$$L_M = L_{Link\_delay1} + (n_{hop} + 1) \times (L_{Router\_delay} + L_{Link\_delay2}) \qquad (4.1)$$

Figure 4.14 shows the different memory controller accesses of two placements. In Figure 4.14a, memory controllers are placed on upper and lower sides of the NoC. This represents a typical design such as Intel and Tilera.

In this case, the average delay for a memory access is: L_{Link_delay1} + *3.5* × *(*L_{Router_delay} + L_{Link_delay2}*)*. In Figure 4.14b and 4.14c, memory controllers are placed on the other positions of the NoC. The average delays for a memory accesses are thus reduced to: L_{Link_delay1} + *3* × *(*L_{Router_delay} + L_{Link_delay2}*)* and L_{Link_delay1} + *2.5* × *(*L_{Router_delay} + L_{Link_delay2}*)*, respectively. We note that memory access time reduction can be different for different nodes under two placements, e.g. nodes 4, 6 and 11 (see Figure 2.1) benefit more from configuration of Figure 4.14c, while the average delay remains the same for nodes 0 and 2. In brief, we think an optimal placement of memory controllers is essential in designing future multicore processors.

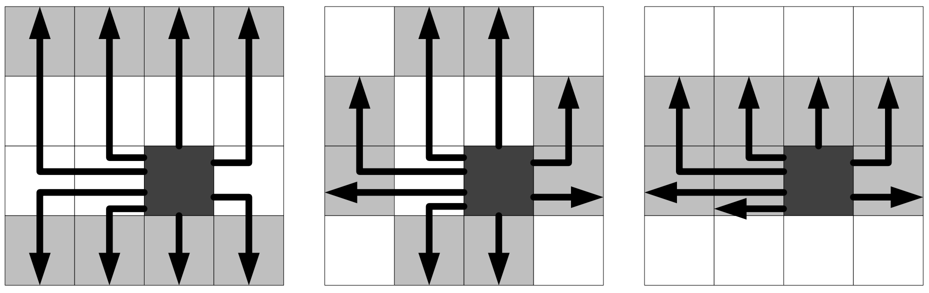

(a) Upper/Lower (b) Four sides (c) Center

Figure 4.14: Three examples of memory controller accesses of different placements of memory controllers, node 6 is colored as dark gray, nodes with light gray are connected with a memory controller.

4.4.2 Analysis of Different Placements of Memory Controllers

The number of transistors required for a memory controller is quite small compared with billions of total transistors in a chip. Many architectures employ a limited number of memory controllers due to the pin count limitation. It is predicted by the ITRS roadmap that pin count will increase by about 10% each year only, comparing with the number of cores that is expected to double every 18 months [7]. As aforementioned, 373 of 1366 pins in the Intel Core i7-900 processor are dedicated to one memory controller [50]. Moreover, about 50% of total pins are used for power supply (e.g. VDD, VCC, VSS and VTT). The ratio between the number of cores and the number of memory controllers is 6:1. The Tilera Tile64 processor [115] is implemented as an 8×8 mesh with four on-chip memory controllers and off-chip memory architecture. The ratio between core and memory controller is thus 16:1. Table 4.2 shows several modern multicore processors with different numbers of memory controllers.

Table 4.2: Comparison of processors with memory controllers

Processor	**Cores**	**Memory controllers**
AMD MagnyCours	6	1 DDR3
Intel Nehalem	4	1 DDR3
IBM Power 7	8	2 DDR3
Tilera Tile64	64	4 DDR2 in 16 ports

As aforementioned, multi channel configuration will increase cost, fault rate and power consumption. Increasing the number of on-chip memory controllers is a practical way for future multicore processors. However, it is not realistic to have a memory controller for each processor core in a NoC. A NoC with n cores and m memory controllers has $\binom{n}{m}$ possible permutations for the placement of memory controllers. Here, we assume that a square shaped NoC with radix $\sqrt{n}$ has $2\sqrt{n}$ memory controllers. If the chip has pin count limitations, these memory controllers can be replaced with memory ports and multiplexed to a smaller number of memory controllers.

The following definitions will be used for subsequent sections.

Definition 4. 1 *A NoC $N(P(X,Y),M)$ consists of a PE mesh $P(X,Y)$ of width X, length Y; and on-chip memory controllers M.*

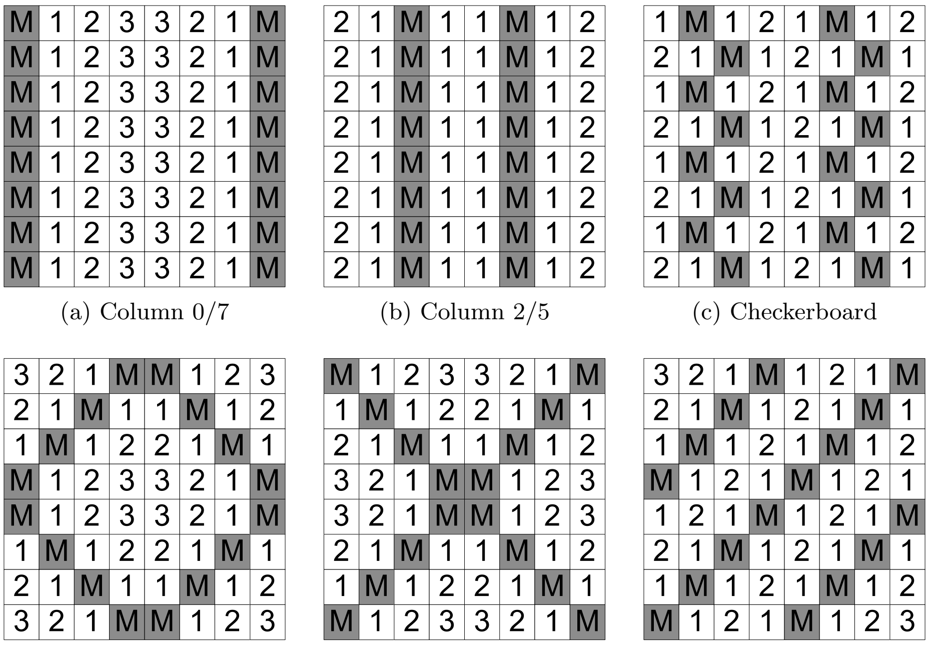
Definition 4. 2 *Each PE is denoted by a coordinate (x,y), where $0 \le x \le X-1$ and $0 \le y \le Y-1$. Each PE contains a processing core, a private L1 cache and a shared L2 cache.*

Definition 4. 3 *The Manhattan Distance (hop count) between $n_i(x_i,y_i)$ and $n_j(x_j,y_j)$ is MD(n_i,n_j), MD(n_i,n_j)=$|x_i - x_j| + |y_i - y_j|$.*

Definition 4. 4 *Two nodes $n_1(x_1,y_1)$ and $n_2(x_2,y_2)$ are adjacent if $|x_1 - x_2| + |y_1 - y_2| = 1$.*

4.4.3 Placement for the Memory Controllers

The placement of memory controllers can have a dramatic impact on the performance of the NoC. It is shown in [2] that the Diamond placement reduces average network latency of a NoC by 10% on average, compared with Column 0/7 (Figure 4.15). Optimal placement of memory controllers can be found through exhaustive simulation. However, due to the computational complexity of a large search space, this method is only feasible for small size NoCs, e.g. an 8×8 mesh with 16 memory controllers has $\binom{64}{16}$ = 488,526,937,079,580 different placement possibilities, $\binom{64}{32}$ = 1.83e+18 for 32 memory controllers!



M	1	2	3	3	2	1	M
M	1	2	3	3	2	1	M
M	1	2	3	3	2	1	M
M	1	2	3	3	2	1	M
M	1	2	3	3	2	1	M
M	1	2	3	3	2	1	M
M	1	2	3	3	2	1	M
M	1	2	3	3	2	1	M

(a) Column 0/7

2	1	M	1	1	M	1	2
2	1	M	1	1	M	1	2
2	1	M	1	1	M	1	2
2	1	M	1	1	M	1	2
2	1	M	1	1	M	1	2
2	1	M	1	1	M	1	2
2	1	M	1	1	M	1	2
2	1	M	1	1	M	1	2

(b) Column 2/5

1	M	1	2	1	M	1	2
2	1	M	1	2	1	M	1
1	M	1	2	1	M	1	2
2	1	M	1	2	1	M	1
1	M	1	2	1	M	1	2
2	1	M	1	2	1	M	1
1	M	1	2	1	M	1	2
2	1	M	1	2	1	M	1

(c) Checkerboard

3	2	1	M	M	1	2	3
2	1	M	1	1	M	1	2
1	M	1	2	2	1	M	1
M	1	2	3	3	2	1	M
M	1	2	3	3	2	1	M
1	M	1	2	2	1	M	1
2	1	M	1	1	M	1	2
3	2	1	M	M	1	2	3

(d) Diamond

M	1	2	3	3	2	1	M
1	M	1	2	2	1	M	1
2	1	M	1	1	M	1	2
3	2	1	M	M	1	2	3
3	2	1	M	M	1	2	3
2	1	M	1	1	M	1	2
1	M	1	2	2	1	M	1
M	1	2	3	3	2	1	M

(e) XCross

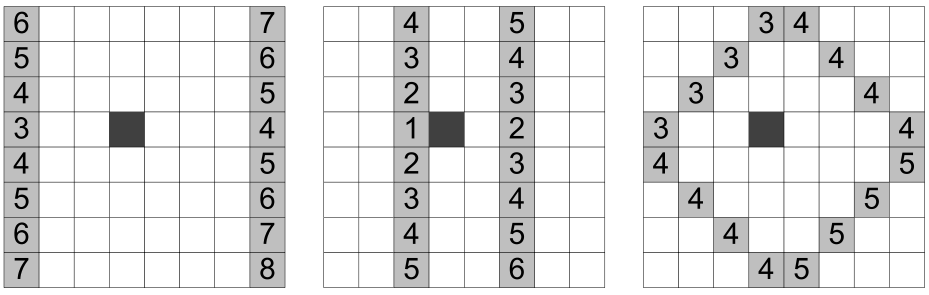
3	2	1	M	1	2	1	M
2	1	M	1	2	1	M	1
1	M	1	2	1	M	1	2
M	1	2	1	M	1	2	1
1	2	1	M	1	2	1	M
2	1	M	1	2	1	M	1
1	M	1	2	1	M	1	2
M	1	2	1	M	1	2	3

(f) Slash

Figure 4.15: Different placement of memory controllers, M means a memory controller, number means minimal hops required for a processor to reach a nearest memory controller. (a) to (e) are studied in [2]

Figure 4.15 shows six intuitive, typical placement possibilities. The values of AHC for a core accessing a nearest memory controller (1MC) are shown in Table 4.3 (AHC/1MC). We note that the memory traffic in our system are distributed to all the controllers evenly, according to the physical addresses. In this case, calculating the average Manhattan Distance (MD) from a node to all the controllers is more meaningful. Figure 4.16 shows hop counts required for transmitting memory requests from a PE to the memory controllers. In Figure 4.16a, the AHC for all memory controllers is 5.5. This value is reduced to 4 and 3.5 in Figure 4.16c and 4.16b respectively. A summary of the numbers are listed in Table 4.3 as AHC/All-MC. It is revealed that Column 2/5 has the lowest number of AHC/All-MC.

However, based on deterministic X-Y routing, if the memory controllers are in the same column, the request messages could get congested, e.g. in Column 0/7 and Column 2/5, eight memory controllers are in the same column (Figure 4.17a). In this situation, all memory messages have to be sent to the next hop node in the first place (16X in the figure). After the first column of controllers, 50% of the messages have to pass the next two nodes (8X in the figure). It would then follow that if the memory controllers are in the same row, the reply messages could get congested (Figure 4.17b). This explains why the Diamond placement (Figure 4.15d) shows the better performance, despite it has higher AHC/1MC and AHC/All-MC values compared with the Column 2/5 and Checkerboard [2]. Memory controllers in Diamond and XCross are distributed better (2 for each row/column, shown in Table 4.3, and the memory request traffic are depicted in Figure 4.17c) than the Checkerboard.



(a) Column 0/7 (b) Column 2/5 (c) Diamond

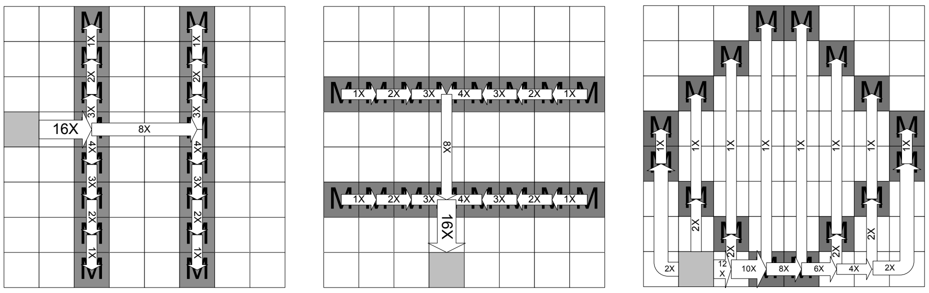
Figure 4.16: Hop counts to all memory controllers in three placements, dark gray node is a active PE sending memory requests, memory controllers are colored as light gray.

Placements presented in Figure 4.15a to 4.15e are studied in [2]. However, we believe that the Slash placement, shown in Figure 4.15f, could be even better than Diamond and XCross, because it provides better AHC and does not worsen the distribution of memory controllers. In fact, in terms of adjacent memory controllers, the XCross placement has four adjacent controllers in the center, which could cause congestion in the central region of the NoC. The Diamond placement has two adjacent controllers in four sides, therefore the congestion should be alleviated. In the Slash configuration, there are no neighboring memory controllers, hence the congestion should be alleviated further. The detailed data of adjacent memory controllers are given in Table 4.3. Besides the aforementioned metrics, the distance between a processor core and memory controllers can be used for analysis as well. We summarize these into three groups of sub-problems.

- Distance-m problem, which considers placing memory controllers such that each processor core in the NoC either has a memory controller directly attached or it is at a Manhattan Distance of at most m from at least one processor core having a memory controller.

- Adjacent-n problem, which considers placing memory controllers such that each node without a memory controller is adjacent to n memory controllers.

- The third problem is a combination of the distance-m and the adjacency-n problems. A node without a memory controller must be a MD of at most m from n memory controllers.



(a) Column 2/5 (b) Row 2/5 (c) Diamond

Figure 4.17: Three cases for memory message congestion.

In Figure 4.15a, any node without a memory controller can reach one memory controller within a MD of three, at least two memory controllers within a MD of four. We note that the distance-m adjacent-n problem can lead to further optimization of the system. For example, modifying the mapping mechanism of virtual addresses to physical memory addresses in the operating system, so that the nearest memory controllers are used [9]. Table 4.3 shows the summary for the placements presented in this section.

We examine further possibilities by using the "divide and conquer" method, and starting from a smaller NoC. It is noteworthy that because of the huge design space, Dennis Abts et al. used several intuitive configurations with a genetic algorithm and random simulations to prune the design space [2]. However, they have missed several possibilities which we will examine in this section. The 8×8 NoC with 16 memory controllers is divided into 4 small NoCs, each 4×4 with 4 memory controllers. The computation complexity is thus reduced from $\binom{64}{16}$ = 488,526,937,079,580 to $\binom{16}{4}$ = 1,820.

We select the design of Figure 4.3 as our placement strategy for 16 memory controllers. In this configuration, all non-native PEs (those not directly attached to a memory controller) require just one hop to access a memory controller, which results in a minimal AHC/1MC. The AHC/All-MC is the same with several other configurations, e.g. Diamond, XCross and Slash. In terms of other metrics, the memory controllers are well distributed, there are only two memory controllers per row/column in our Optimal placement. No adjacent memory controllers can be found in the Optimal placement as well. These provide good memory variance. Any node without a memory controller in Figure 4.3 can reach one memory

Table 4.3: Characteristics of different memory controller (MC) placements. AHC/1MC and AHC/All-MC are shown as 1MC and All-MC respectively. The Adjacent Memory Controllers (AMC) are shown in *Number of Sets*×*Number of nodes in each Sets.*

Placement	**1MC**	**All-MC**	**Distance-m, Adjacent-n**
Column 0/7	1.50	6.125	1 MC in 3 hops, 2 MCs in 4 hops
Column 2/5	1.00	4.875	1 MC in 2 hops, 2 MCs in 3 hops
Checkerboard	1.00	5.125	1 MC in 2 hops, 2 MCs in 3 hops
Diamond	1.25	5.250	1 MC in 3 hops, 2 MCs in 3 hops
XCross	1.25	5.250	1 MC in 3 hops, 2 MCs in 3 hops
Slash	1.06	5.250	1 MC in 3 hops, 2 MCs in 3 hops
Optimal	0.75	5.250	1 MC in 1 hop, 2 MCs in 2 hops

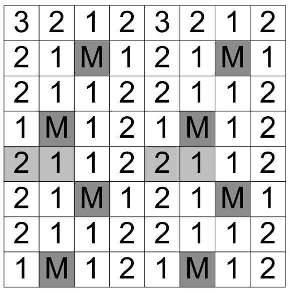
Placement	**MC per row/col**	**AMC**
Column 0/7	8	2×8
Column 2/5	8	2×8
Checkerboard	4	0
Diamond	2	4×2
XCross	2	1×4
Slash	2	0
Optimal	2	0

controller within a MD of one, at least two memory controllers within a MD of two. We note that by minimizing AHC/1MC, the overall theoretical metrics are the best among all strategies.

4.4.4 Generic Placement Method

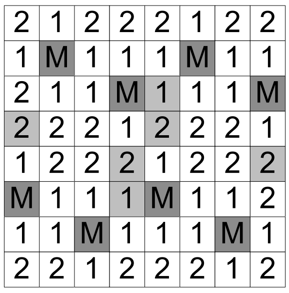
Our method can be applied to other numbers of memory controllers as well, e.g. an 8×8 mesh with 8 memory controllers. In this case, a 4×4 mesh with 2 memory controllers is enumerated ($\binom{16}{2} = 120$, compared with $\binom{64}{8} = 4{,}426{,}165{,}368$). Figure 4.4 shows these configurations.

We expand the best 4×4 placement four times in 16 memory controllers, however it is more complex for the 8 memory controller configuration. Figure 4.18a shows a placement with four instances of Figure 4.4a. The minimal hop count required for four nodes to reach a memory controller is reduced by one for the lower two instances (light-gray numbers). However, by combining different placements in Figure 4.4, the reduction for hop counts can be even better. Figure 4.18b shows a placement, in which two instances of Figure 4.4h are placed on top, and two instances of Figure 4.4g are placed on bottom. In Figure 4.18c, two instances of Figure 4.4f are placed on the right side, one of Figure 4.4h is placed on top-left, and one of Figure 4.4g is placed on bottom-left. The hop counts in these two figures are reduced by one in six and five nodes respectively, better than Figure 4.18a. Apparently, the AHC/1MC in Figure 4.18b and 4.18c are better than Figure 4.18a, while the AHC/All-MC in the three configurations remain the same (5.125).



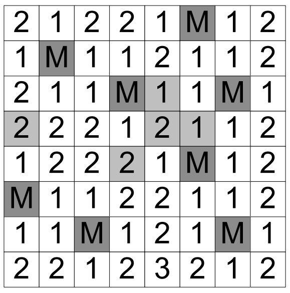
3	2	1	2	3	2	1	2
2	1	M	1	2	1	M	1
2	1	1	2	2	1	1	2
1	M	1	2	1	M	1	2
2	1	1	2	2	1	1	2
2	1	M	1	2	1	M	1
2	1	1	2	2	1	1	2
1	M	1	2	1	M	1	2

(a) Placement 1



2	1	2	2	2	1	2	2
1	M	1	1	1	M	1	1
2	1	1	M	1	1	1	M
2	2	2	1	2	2	2	1
1	2	2	2	1	2	2	2
M	1	1	1	M	1	1	2
1	1	M	1	1	1	M	1
2	2	1	2	2	2	1	2

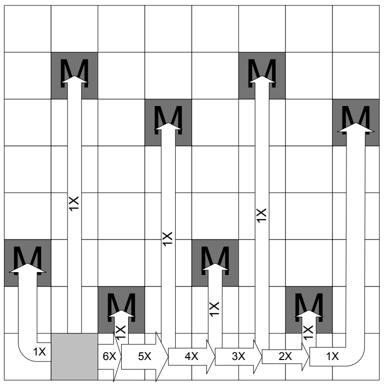
(b) Placement 2



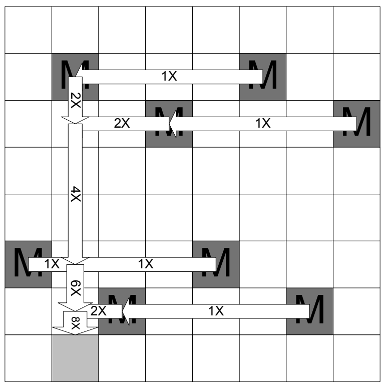
2	1	2	2	1	M	1	2
1	M	1	1	2	1	1	2
2	1	1	M	1	1	M	1
2	2	2	1	2	1	1	2
1	2	2	2	1	M	1	2
M	1	1	2	2	1	1	2
1	1	M	1	2	1	M	1
2	2	1	2	3	2	1	2

(c) Placement 3

Figure 4.18: Three placements of 8 memory controllers in an 8×8 mesh.



(a) Request messages



(b) Response messages

Figure 4.19: The memory request and response traffic for Placement 2 in Figure 4.18.

Besides, other metrics in Figure 4.18b are better than Figure 4.18a. For instance, in Figure 4.18a, there are two memory controllers in all even rows/columns, meaning that the memory requests/responses could be congested on these rows or columns. However, in Figure 4.18b, memory controllers are evenly distributed in the vertical axis. In this case, the request messages will be distributed evenly in the horizontal axis (Figure 4.19a), the response messages will be concentrated on the four rows (Figure 4.19b). The situation is different in Figure 4.18c: there are two rows with two memory controllers in the horizontal axis (four rows with one, two with zero), and two columns with two memory controllers in the vertical axis (four columns with one, two with zero). In this case, the request and response messages will be evenly balanced in two axes.

The combination should take other metrics into consideration as well, e.g. Distance-m, Adjacent-n and adjacent memory controllers. We note that to find the best combination, the problem can be described as: Choose 4 placements from Figure 4.4, the order of choice is important, and repetition is allowed. The design space is reduced to $8^4 = 4096$.

Table 4.4: Comparison of mesh size and placement possibilities, with $2\sqrt{n}$ memory controllers

Mesh size	**Placement possibilities**	**Exec time**
3×3	84	0.007s
4×4	12870	0.060s
5×5	3268760	23.343s
6×6	1.252e+9	14755.577s
7×7	6.752e+11	N/A
8×8	4.885e+14	N/A
9×9	4.567e+17	N/A
10×10	5.360e+20	N/A

Although we study memory controller placement for an 8×8 mesh in this section, the idea can be generalized. The time required for exhaustive simulation is practical for a NoC below the size of 6×6. Table 4.4 shows placement possibilities and execution time for exhaustive simulation with different mesh sizes. The simulation platform is 3.372GHz Intel Core 2 Duo processor with 2GB of memory. We observe that the execution time is too long beyond 6×6, thus the data is shown as "N/A".

For a larger NoC, we divide it into several smaller NoCs and solve them. The time required for enumerating a smaller NoC is feasible:

- A 6×6 NoC with 12 memory controllers is a combination of four 3×3 NoCs, each with 3 memory controllers ("divide", $\binom{9}{3} = 84$). In this case, simulation shows 10 best placements. The design space for "conquer" is $10^4 = 10,000$.
- A 10×10 NoC with 20 memory controllers is a combination of four 5×5 NoCs, each with 5 memory controllers ("divide", $\binom{25}{5}$ = 53,130). In this case, simulation shows 34 best placements. The design space for "conquer" is $34^4 = 1,336,336$.

Table 4.5 lists the computation complexity of two methods, for several chip configurations. Our idea can be expanded to NoCs with regular sizes and regular number of memory controllers. It is noteworthy that NoCs with large prime radix numbers (13×13 and 19×19 etc.), or with irregular number of memory controllers (11, 17 etc.) are more difficult to solve, however these configurations are very rare in real world.

Table 4.5: Comparison of computation complexity of exhaustive enumeration and "divide and conquer", for several memory controller placements.

Problem	**Exhaustive**	**Divide**	**Conquer**
6×6, 12MC	1.252e+9	84	10,000
8×8, 8MC	4.426e+9	120	4,096
10×10, 20MC	5.360e+20	53,130	1,336,336

4.4.5 Routing Algorithm

Adaptive routing is used widely in off-chip networks, however deterministic routing is favorable for on-chip networks because the implementation is less complex. Dimensional ordered routing (DOR) [99] algorithms, e.g. X-Y and Y-X, are typical deterministic routing algorithms to avoid deadlocks. In X-Y routing, when a node sends a flit to another node, the flit will first travel along the X direction, then it will be routed in the Y direction. A Class-based Deterministic Routing (CDR) is proposed to further reduce network contention [2]. The algorithm takes advantage of both X-Y and Y-X routing, where the path is determined by message class. Memory request packets use one routing algorithm, while reply packets take another.

Intuitively, Column 0/7 and Column 2/5 benefit more from CDR compared with the other placements. Since the memory controller nodes are in the same column, inducing high contention in one axis of X-Y and Y-X routing. It is claimed that CDR improves performance by 56% for uniform traffic comparing with the baseline X-Y routing for Column 0/7 placement [2]. The performance improvement is very limited in the Diamond placement, since the memory controllers are well distributed in rows and columns and there are only four sets of two adjacent nodes. Here, we propose an optimal placement of memory controllers, in which the distribution of traffic and adjacent nodes are comparable or better than the Diamond placement. Therefore we use X-Y routing instead of CDR.

4.4.6 Address Mapping

In our system, the memory controllers are shared by all processors. For a physical address (an unsigned 64-bit integer), the following algorithm determines which memory controller it maps to. The memory controller accepts requests from the processors and provides the arbitration interface to determine which request will be processed. The request is then mapped to a memory address location and converted to Dynamic Random Access Memory (DRAM) commands.

Code 3 The pseudo code of mapping physical addresses to memory controllers.

```
int Address_to_MC(addr)
{
 b1=Data_Block_Bits;  //Data block width in bits
 b2=Data_Block_Bits+Memory_Bits-1;
 if(b2>=63)           //Memory address width (64) minus 1
  return (addr>>b1);
 else
  return ((addr&(~((64)~0<<(b2+1))))>>b1);
}
```

4.4.7 Experimental Evaluation

In this section, we present the experimental evaluation under different placements of memory controllers. Applications are selected from SPLASH-2 [116], TPC-H [104] and PARSEC [18]. To evaluate the performance of different memory controller placements, we model a NoC with 16 memory controllers. The NoC consists of an 8×8 mesh of PEs. Each PE consist of a processor core with a private L1 cache and a shared L2 cache. We use a 64-node network which models a single-chip CMP for our experiments. The size of each cache bank node is 1MB; hence the total size of shared L2 cache is 64MB. MOESI is used in our memory hierarchy. A power simulator for interconnection networks is used to evaluate detailed power characteristics [57]. The simulator models a wormhole router with corresponding input/output ports, buffers and the crossbar. Power consumption of both routers and links are analyzed.

We evaluate performance in terms of Average Network Latency (ANL), Average Link Utilization (ALU) and Performance Power Product (PPP). The results are illustrated in Figure 4.20, 4.21, 4.22 and 4.23. The ANL is measured in two ways: one for memory request messages (ANL-MReq), and another for all network messages (ANL-All). ANL-MReq represents the average number of cycles required for the transmission of memory request messages. ANL-All represents the average cycles required for the transmission of all network messages, e.g. cache data and coherence messages. The number of cycles of each message is calculated as, from the injection of the message header into the network at the source node, to the reception of the tail flit at the destination node.

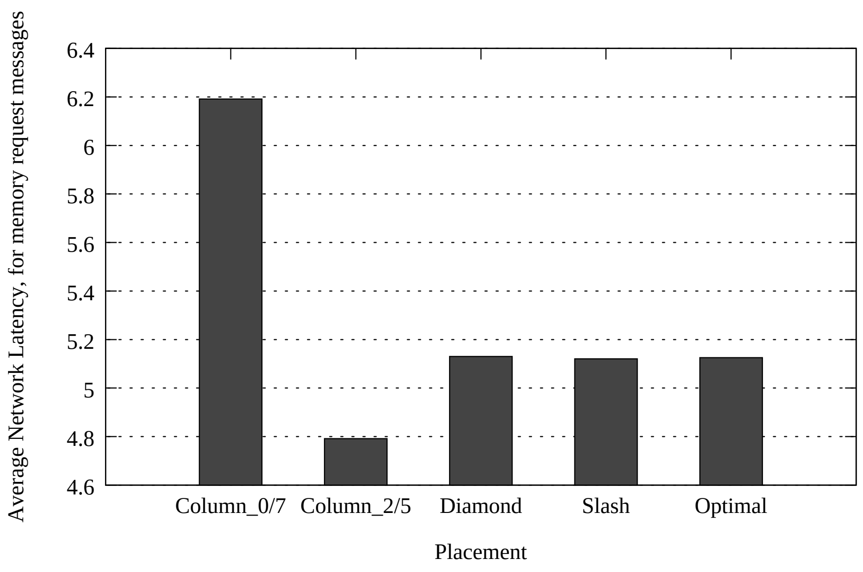


Figure 4.20: ANL-MReq for FFT, with different placements of memory controller.

As is shown in Figure 4.20, in terms of memory request messages, the Column 2/5 placement outperforms the other placements. The ALN-MReq in Column 2/5 is reduced by 22.6% compared with Column 0/7. The improvement is because of the lower value of AHC/All-MC in Column 2/5. We note that the experimental results conform to our theoretical analysis in Table 4.3. In spite of the lower ALN-MReq in Column 2/5, only processor(cache)-memory traffic is considered in this case.

The data in Figure 4.21 contains additional traffic, for example, processor-processor communication. It is noteworthy, the processor-processor communication makes up the major portion of the traffic. According to our traffic trace, more than 90% of the traffic are processor-processor related. Both Slash and Optimal placements outperform the other placements in terms of ANL. For example, comparing with the Column 0/7 placement, the ANL for all applications of the Slash placement is reduced by 6.04% (7.63% for the Optimal placement). This is primarily due to the reduced hop count of memory data traffic in the Slash and Optimal placements, compared with that in the Column 0/7 counterpart. Although Column 2/5 has lower number of ANL-MReq, in terms of ANL, it does not benefit from that advantage. As aforementioned, the adjacent memory controllers in Column 2/5 can cause traffic congestion.

The ALU is defined as the number of flits transferred between NoC resources per cycle. The average load of all virtual channels are calculated.

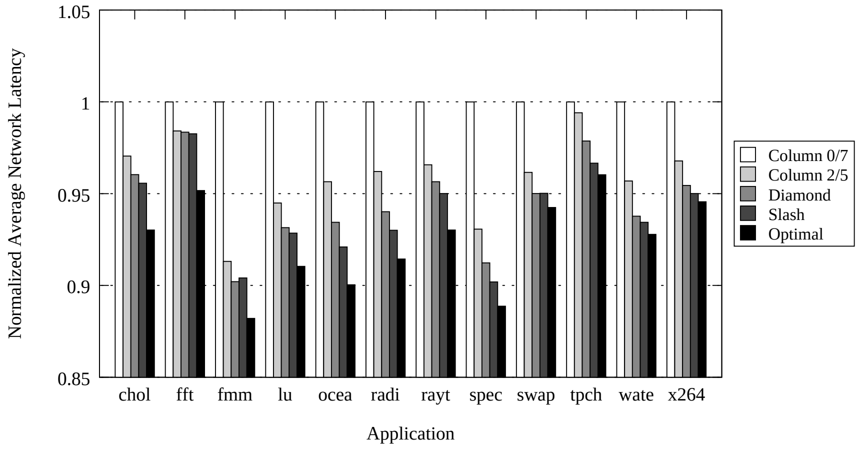


Figure 4.21: Normalized ANL with different placements of memory controller.

Under the same configuration and workload, lower utilization means alleviated network load, which is favorable. It is depicted in Figure 4.22 that comparing with the traditional Column 0/7 design, the ALU for our proposed Optimal placement is reduced by 10.44%, on average. FMM, SPECjbb and swaptions have the most significant reduction of ALU, 14.79%, 13.38% and 12.14% respectively. We note that the ALUs in Column 0/7 and Column 2/5 are nearly the same, while there's a significant reduction in the other three placements. This is due to the fact that the memory request messages could get congested in the 8 adjacent memory controllers of the two columns in these two placements (Column 0 and 7 in Column 0/7, Column 2 and 5 in Column 2/5). Besides, the congestion may happen in surrounding nodes as well (Figure 4.17a). On average, the ALU for our proposed Optimal placement is reduced by 2.87% compared with the Diamond placement.

As a combination of both performance and power consumption, the PPP is a more meaningful metric because performance is usually a trade-off for power consumption. We measure performance in terms of execution time of an application. Power consumption of both links and routers are calculated. Systems with a lower PPP generally have a better trade-off between the performance and power. The results in Figure 4.23 show that the PPP improves more significantly, when compared to ANL and ALU improvements. For example, configuration with Optimal placement performs 17.56%, 15.60%, and 16.38% better than Column 0/7 in FMM, Ocean and SPECjbb respectively. On average, the PPP is reduced by 13.94% and 10.90% for Optimal and Slash designs respectively compared with the Column 0/7 design. By using better placements, we notice that the execution time is reduced slightly (e.g. 5.88% reduced execution time of FFT in Opti-

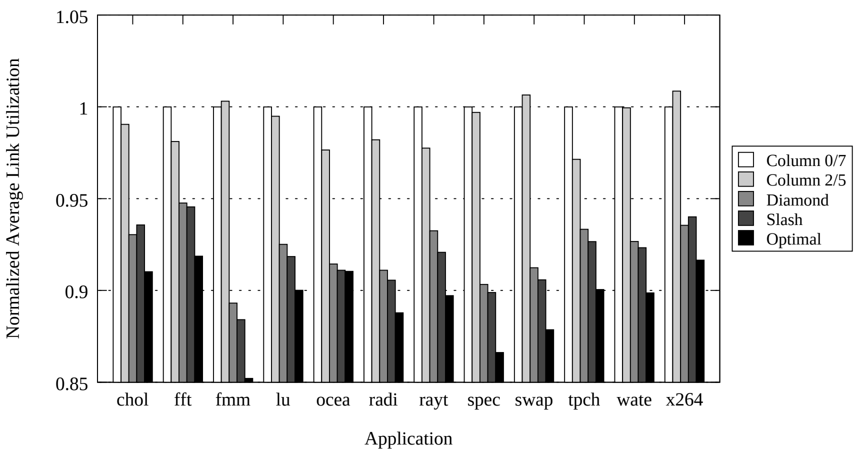


Figure 4.22: Normalized ALU with different placements of memory controller.

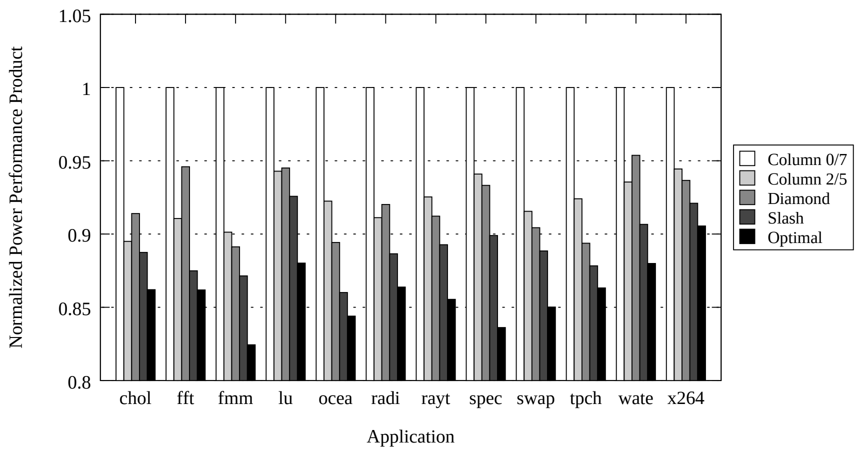


Figure 4.23: Normalized PPP with different placements of memory controller.

mal compared with Column 0/7), while the power consumption is reduced further (e.g. 8.43% reduced power consumption of routers and links of FFT in Optimal compared with Column 0/7). We also note that the Column 2/5 placement provides lower execution time in some cases (Cholesky, FFT, LU, Radix etc.), due to its lower AHC/All-MC. However, as aforementioned, the concentrated memory controllers in Column 2/5 and Diamond can cause hotspots in certain region of the on-chip network. In conclusion, the system efficiency will improve significantly with an optimal placement of memory controllers.

4.5 Placement for Core and Cache

Multicore processors are dominating the market nowadays. Normally, each core has its own computational units, registers and private caches. A last level cache will be shared by all cores. Intel Pentium-D is one of the first CMP which embedded two dies on a processor package. Quad-core CMPs such as Intel Core i5, i7 [50] and AMD Phenom II [5] are mainstream for desktop systems. The integration of more cores on a CMP is under intensive research. For example, AMD has announced its twelve-core Opteron CMP with two dies on a chip, each of which has six cores [6]. The newest Xeon, based on Westmere-EX architecture, has been fabricated by Intel with ten cores [52]. Sun and IBM have their own CMP design as well [105, 48]. It is predictable that in the future, the integration of more cores on a single chip will move on.

In this section, we investigate the placement of cores and caches in a 2D mesh network interconnect. We model a CMP based on state-of-the-art processors. We analyze the impact of different placements of cores/caches in a mesh network. Optimal placements of several configurations have been found and discussed. We present the performance using a full system simulator.

4.5.1 Modern Chip Multiprocessor Design

As aforementioned, there have been different interconnect topologies for CMPs. Conventional designs are focused on crossbar or hierarchical crossbar. While recently, designers are moving from crossbar to ring interconnect to improve scalability. Ring has natural ordering properties, making design, validation and routing simpler. Table 4.6 shows the key features of state-of-the-art CMPs. Take the Westmere-EX for example (Figure 4.24), it uses a bi-directional ring to connect the last level caches to the cores. The ring consists of 4 different sub-rings, namely request, acknowledge, snoop and data. The data ring is pipelined and uses 32 bytes data width. For each processor cycle, data can move one hop on the ring for either directions. These 4 rings are responsible for the cache/memory coherency. It is reported that [27], although the sizes of last level cache in Nehalem and Sandy Bridge are the same (8MB), the access latency is significantly reduced from 35 cycles in Nehalem (Minimal latency is 35 cycles if the frequencies of core and uncore are the same) to 26-31 cycles in Sandy Bridge (Depending on the hop count to the cache slices). Several factors contribute to the reduced latency. For example, the cache architecture of Nehalem is uniform and divided into multiple levels. The access latency of the 8MB last level cache is determined by the longest access latency to the furthest data arrays and tags. While in Sandy Bridge, the 8MB cache is split into

4 slices, each 2MB. The access latencies to access the cache slices are decreased. Each slices has different access latency, making it a non-uniform cache architecture.

Table 4.6: Comparison of different Chip Multiprocessors. LLC for Last Level Cache.

CMP	**Cores**	**LLC**	**Transistors**
AMD MagnyCours [6]	12 (2×6)	12MB	1,808M
IBM Power 7 [48]	8	32MB	1,200M
Intel Nehalem [50]	4	8MB	731M
Intel Westmere [50]	6	12MB	1,170M
Intel SandyBridge [52]	4	8MB	995M
Intel Nehalem-EX [50]	8	24MB	2,300M
Intel Westmere-EX [52]	10	30MB	2,600M
Sun SPARC T3 [105]	16	6MB	1,000M

CMP	**Process**	**Die Size**	**Interconnect**
AMD MagnyCours [6]	45nm	692mm^2	Crossbar
IBM Power 7 [48]	45nm	567mm^2	Ring
Intel Nehalem [50]	45nm	263mm^2	Crossbar
Intel Westmere [50]	32nm	239mm^2	Crossbar
Intel SandyBridge [52]	32nm	216mm^2	Ring
Intel Nehalem-EX [50]	45nm	684mm^2	Ring
Intel Westmere-EX [52]	32nm	513mm^2	Ring
Sun SPARC T3 [105]	40nm	377mm^2	Hierarchical Crossbar

Routing in a ring is more complex than routing in a crossbar. Furthermore, the scaling of the ring has limitations. As data travel across the ring, it can block accesses from other agents. With each additional agent, the available bandwidth will be reduced and the average hop count will increase. In this case, as aforementioned, on-chip network has been introduced as a better solution. The scalability of on-chip network is better compared with crossbar and ring. Each network node can include different component, e.g. a node with only Core/L1 cache, and a node with only L2 cache, creating a heterogeneous network.

4.5.2 Area of Core and Cache

As is shown in Figure 4.24, cores and caches in the Westmere-EX have similar area. For example, a 32nm Westmere core has an area of about 17mm^2. The last level cache in Westmere-EX is split into 10 slices, each 3MB. Notice that each slice is about the same size as a core. Here, we assume that:

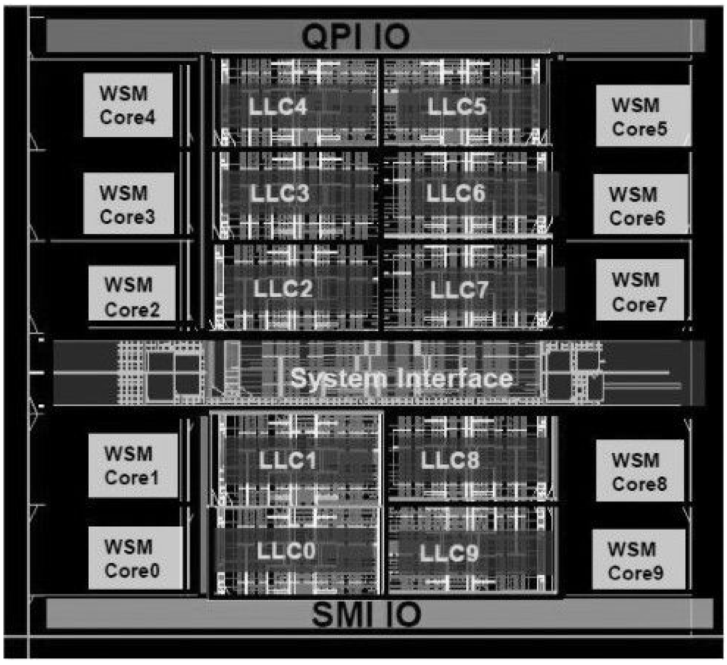


Figure 4.24: Intel Westmere-EX architecture, presented by Intel at Hot Chips 2010.

- The area of a core and a cache slice are the same,
- The last level cache consists of multiple slices, the number of cores and cache slices is equal.

In this case, an 8-core CMP has 8 cores and 8 cache slices. These assumption fits the design principles of Intel and IBM [50, 52, 48].

4.5.3 Ring and Network Connection

In a ring connection, the average hop count accessing cache slices for all processors are the same. For example, Figure 4.25a shows an abstract model of 8-core Nehalem-EX. The cores (C0 to C7) are connected to the central bi-directional ring (the connections between cores and ring are not shown in figure). Figure 4.25b and 4.25c demonstrate the hop counts required for accessing each cache slice of C0 and C1, respectively. The latencies for accessing each cache slice is different. Considering a perfect application with average access to cache slices, for the 8-slice cache, each slice has an access probability of ${}^{1}/_{8}$. Real-world applications are most likely to have different cache access patterns, i.e. the access frequency to certain slices might be higher than others. In this case, the addresses of data kept in the cache arrays can be distributed among the cache slices with a uniform distributed hash function [27]. However, a system with billions of cache accesses tend to have a balanced access to all slices. We define AHP_{Cn} as the Average Hop Count accessing all cache slices for a core Cn:

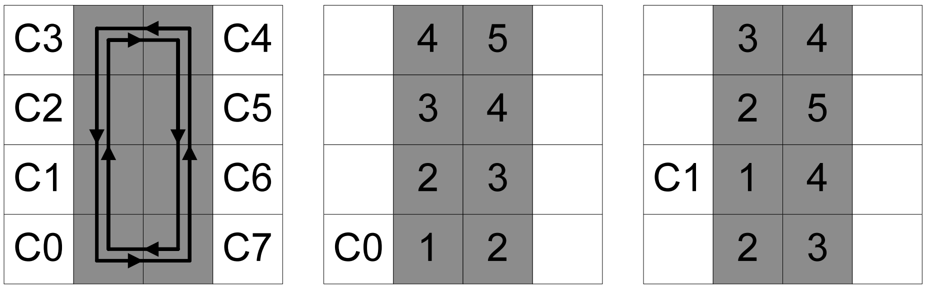
$$AHP_{Cn} = \frac{\sum_{i=0}^{m} Distance(C_n, Cache_i)}{m+1}$$

Such that: $m + 1 =$ (Number of cache slices)

Notice that the average hop counts for C0 (AHP_{C0}) and C1 (AHP_{C1}) to all cache slices are the same, which is 3. We define AHP_{All} as the average value of all AHP_{Cn}:

$$AHP_{All} = \frac{\sum_{i=0}^{n} AHP_{Ci}}{n+1}$$

Obviously, the AHP_{All} for Figure 4.25a is 3 since all cores have the same average latency to the cache.



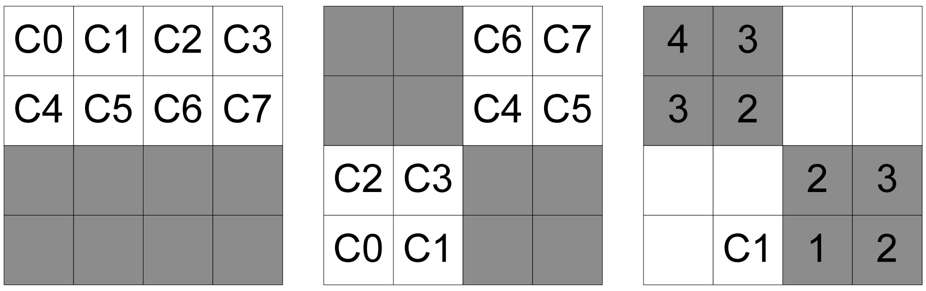
(a) Bi-directional ring (b) Hop count for C0 (c) Hop count for C1

Figure 4.25: An example of 8-core CMP, with Bi-directional ring interconnect (a, Cx as Core x), and access latency (shown as hop count) for Core 0 (b) and Core 1 (c) of each cache slice.

By implementing a network interconnect, the AHPs can be non-uniform for different cores. Manhattan Distance, or MD, is used to calculate the hop count in a network-based interconnect. For example, AHP_{C0} in a network is the same as in a ring, however AHP_{C1} in a network is just 2.5 compared with 3 in a ring. We calculate that in Figure 4.25a, the AHP_{All} for a network interconnect is 2.75, 8.3% lower than that in a ring.

4.5.4 Placements of Caches

The different placement of cache slices can have a dramatic impact on the performance of the CMP. Figure 4.26a shows a worst case placement of 8 cache slices. In this configuration, all the cache slices are placed on one side of the chip (gray nodes), causing high delays for the cores in the other side of the chip (AHP_{All}=3.25).



(a) Worst case (b) Best case (c) Hop count for C1

Figure 4.26: Examples of worst case placement (a), best case placement (b), and access latency Core 1 (c) of each cache slice of the best case.

To minimize cache access latencies for all cores, the placement of cache slices should be considered carefully. For example, an 8-core CMP with 8 cache slices has $\binom{16}{8}$=12,870 different cache placement possibilities. The value of AHP_{All} varies significantly with different placements. We explore the placements by writing a program that exhaustively enumerate all possibilities and output the AHP_{All} (Code 4). The outputs are then processed and sorted for analysis. A best case placement of 8 cache slices is illustrated in Figure 4.26b. We observe that there are 90 placements with lowest AHP_{All} (2.5), and 4 placements with highest AHP_{All} (3.25). The cache access latencies of Core 1 are demonstrated in Figure 4.26c. It is noteworthy that AHP_{C3} and AHP_{C4} are lowest among 8 cores, while AHP_{C0} and AHP_{C7} are highest.

We continue our investigation for a 10-core CMP, similar to Westmere-EX. In this case, there are $\binom{20}{10}$=184,756 different cache placement possibilities. The simulation output has 4 worst case placements and 1,240 best case placements. Figure 4.27a and 4.27b show two of these. The AHP_{All} for worst case placements and best case placements are 3.68 and 2.86 respectively, while the value is 3.5 in a ring interconnect.

The best case AHP_{All} for a 12-core CMP is similar to the 8-core, where 6 cores are placed on one side, and the other 6 cores are placed on the other side of the CMP (Figure 4.28a). The AHP_{All} for worst case placements and best case placements are 3.19 and 4.25 respectively, while the value is 4 in a ring interconnect. We find 1,860 best case placements from 2,704,156 combinations. Figure 4.28b shows a best case placement for a 14-core CMP, notice that the placement of cores and caches is similar to the 10-core configuration.

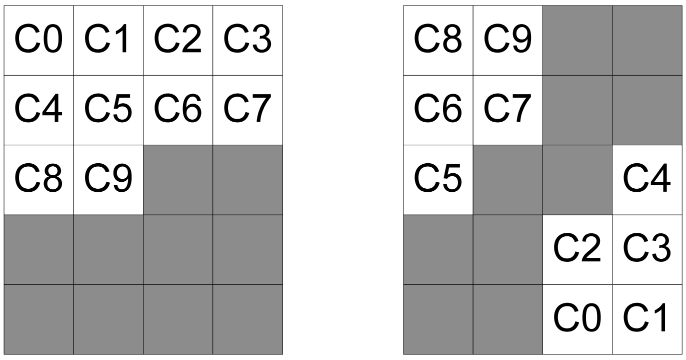
Code 4 Part of C source code of calculating AHP_{All}, for a combination of cache slices placement.

```
void calculate_ahpall()
{
 for(i=0;i<Combinations;i++)  //Calculate for all combinations
 {
  for(j=0;j<C_Num;j++) C[j]=C_Comb[i*C_Num+j];
                              //C_Comb is an array of all cache
                              placement combinations
  last_node=0;
  for(k=0;k<Net_x*Net_y-C_Num;k++) //Calculate for all caches in the CMP
  {
   Cores[k]=last_node;        //Cores is set of cores
   for(l=0;l<Net_x*Net_y;l++)
   {
    for(m=0;m<C_Num;m++)
    {
     if(Cores[k]==C_Comb[m]) Cores[k]++;
     if(k>0&&(Cores[k-1]==Cores[k])) Cores[k]++;
    }
    last_node=Cores[k];
   }

  for(x=0;x<Net_x*Net_y-C_Num;x++) //for all cores
  {
   for(y=0;y<C_Num;y++)            //for all caches
   {
    tmp+=MD(Cores[x], C_Comb[y]);  //Calculate the MD from a Core to a
                                     cache slice
   }
   avg_dist=tmp/C_Num;
   total_dist+=avg_dist;
   avg_dist=0;
   tmp=0;
  }
  final_ahpall[i]=total_dist/(Net_x*Net_y-C_Num);
                                   //Final AHP_all for a combination
  total_dist=0;
 }
}
```

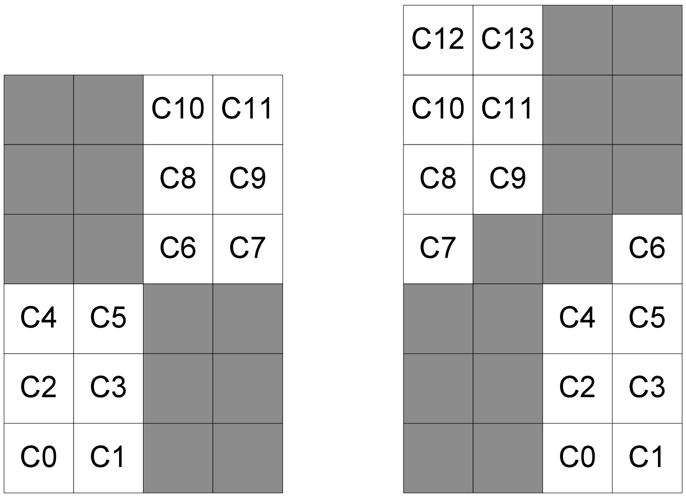
4.5.5 Discussion

To compare the difference between ring and network interconnect, we summarize and calculate the AHP_{All} for several configurations. Table 4.7 shows the data (NW for worst case placement of a network, NB for best case placement of a network, NB%Ring for percentage of NB compared with Ring, NB%NW for percentage of NB compared with NW). It is shown that, on average, the AHP_{All} for a best placement of cache slices in a network inter-



(a) Worst case (b) Best case

Figure 4.27: Examples of worst case placement (a), and best case placement (b), for a 10-core 10-cache configuration.



(a) 12-core (b) 14-core

Figure 4.28: Best case placements for a 12-core CMP (a), and 14-core CMP (b).

connect is around 25% less than that in a ring. Researches have shown that [27], due to the implementation of ring interconnect, the last level cache hit latency has reduced from 35-40 cycles (Nehalem) to 26-31 cycles (Sandy Bridge).

We use abstract models of $4\times N$, where N equals to half of the number of cores. For example, we use $4\times N$ network in "10 Core/10 Cache" configuration. This model fits well with smaller number of cores/caches. According to [67], a model shape has lower number of AHP (this AHP is

Table 4.7: Comparison of AHP_{All} for different interconnect and placement

Configuration	Ring	NW	NB	NB%Ring	NB%NW
8 Core/8 Cache	3.00	3.25	2.50	83.3%	76.9%
10 Core/10 Cache	3.50	3.68	2.86	81.7%	77.7%
12 Core/12 Cache	4.00	4.25	3.19	79.9%	75.2%
14 Core/14 Cache	4.50	4.69	3.54	78.7%	75.5%

for all nodes, assuming a homogeneous model) if it is closer to a square. Their conclusion applies to our heterogeneous model as well. Considering the "12 Core/12 Cache" configuration which has 24 nodes, the communication overhead for a 4×6 model is smaller than that in 3×8 and 2×12. In this case, we don't consider models such as 3×8 and 2×12. CMPs with a higher number of cores are not considered here, due to the fact that the number of cores and cache slices may not be equal in the future. For instance, current desktop CMP design trends tend to have smaller and faster caches for better user interface experience, while a CMP designed for server environment have larger caches.

However, it is hard to define a rule of thumb for optimal placement of cores/caches. We note that exhaustive simulation is feasible for a smaller number of nodes, e.g. a 6×6 model with 18 cores and 18 cache slices has $\binom{36}{18}$ = 9,075,135,300 different placement possibilities, a 5×8 model with 20 cores and 20 cache slices has $\binom{40}{20}$ = 137,846,528,820 possibilities. The design space grows exponentially as the number of cores increases, making the exhaustive method to be unpractical. As aforementioned, divide and conquer method can be used for a huge design space, however this method only applies with certain limitations. Heuristic-guided methods can reduce the computational complexity from a large search space, notwithstanding the non-optimal outputs.

4.5.6 Experimental Evaluation

In this section, we present the experimental evaluation under different CMP configurations. Applications are selected from SPLASH-2 [116] and PARSEC [18].

We use ring and network interconnects to model a single-chip CMP with 8, 10, 12 and 14 cores (16, 20, 24 and 28 nodes). Each processor core has a private write-back L1 cache. The L2 cache shared by all processors is split into slices/banks. The size of each cache bank is 3MB; hence the total sizes of the shared cache are 24MB, 30MB, 36MB and 42MB respectively. MOESI cache coherence protocol is implemented here. Workloads used here include FFT from SPLASH-2 and swaptions from PARSEC. The

FFT is a one-dimensional, radix-n, six-step algorithm, which is optimized to minimize inter processor communication. The communication between processors only take place at the last stage of the execution. However the coherence traffic and cache miss rate are very high. Swaptions is a workload that employs Monte Carlo simulation to compute prices.

We evaluate performance in terms of Average Network Latency (ANL) and execution time. The results are illustrated in Figures 4.29 and 4.30.

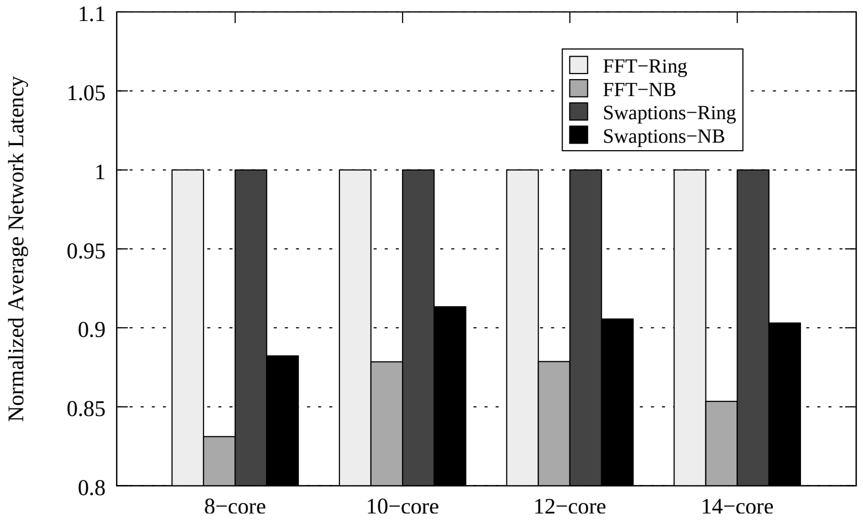


Figure 4.29: Normalized average network latency in cycles.

Figure 4.29 shows that, on the whole, the average network latency of our optimal core/cache placement in a network interconnect (*-NB in the figure) outperforms the ring interconnect (*-Ring in the figure) in four configurations. The improvement is more significant in the 8-core configuration, with 16.89% reduced latency in FFT and 11.78% reduced latency in Swaptions, respectively. This is primarily due to the fact that the AHP_{All} for NB is much lower than that in the Ring. We notice that the on-chip communication consist of cache coherent messages, cache data and memory control/data messages. Out of these messages, the cache related control/data messages are the dominant (over 90%). In this case, lower average access latency to the cache slices will transfer into lower average network latency. We also note, the improvement is more significant in application with higher cache access frequencies, for example FFT. Overall, the average network latency has reduced 11.93% for four configurations.

As depicted in Figure 4.30, in terms of execution time, the optimal core/cache placement has considerable advantage compared with the ring. Overall, the reduction of execution time is more significant than that in the average network latency. Comparing with the ring interconnect, on average,

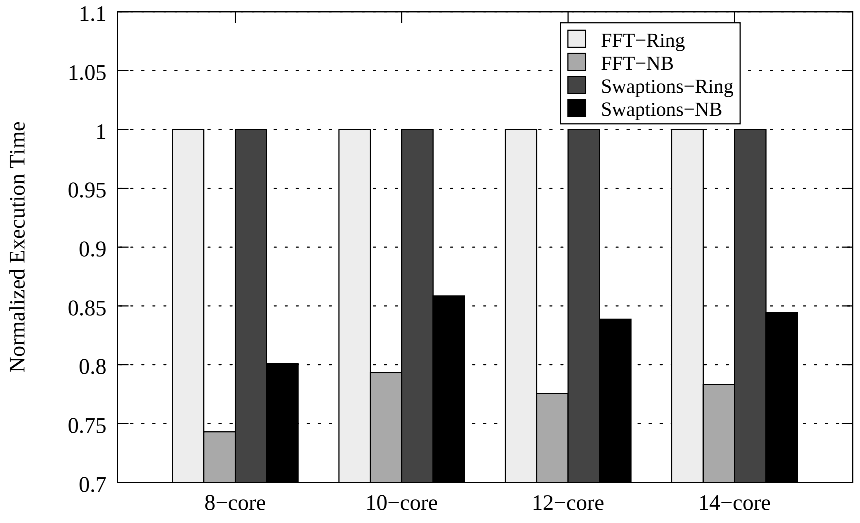


Figure 4.30: Normalized execution time.

the execution time for our proposed core/cache placement is reduced by 19.53% in four configurations. Again, the higher cache/memory traffic in FFT benefits more from reduced AHP_{All}, compared with Swaptions. Results shown that the execution time of FFT has reduced 22.62% in four configurations, while the reduction in Swaptions is 16.43%. This proves that FFT is more sensitive to core/cache latency than Swaptions as well. The savings of AHP_{All} have translated directly into the reduced execution time.

Chapter 5

Application Study for Network-on-Chip

To improve energy efficiently, a computer system should have both efficient hardware and software specially optimized for the hardware. For computers based on Network-on-Chip paradigm, the hardware chip is manufactured for research and commercial use by Intel, Tilera, Intellasys and many other companies and laboratories. However, the adaptation of software system falls far behind the hardware. Without the collaboration of software system, the processing ability of NoC can be limited, and therefore the system energy efficiency can be low. In this section, we analyze several applications originally designed for traditional architectures, and try to give optimization suggestions for NoC architecture. Arbitrary core allocation is used in this section.

In Section 5.1, we implement, analyze and compare different NoC architectures aiming at higher efficiency for MPEG-4/H.264 coding. Noticing that the inter-thread data dependencies of shared reads and writes are performance bottlenecks, we explore different Non-Uniform Cache Access (NUCA) designs using NoC architectures. Two-dimensional (2D) and three-dimensional (3D) NoCs have been analyzed with the focuses on hop counts and heat dissipation. Experiments show that under different workloads, the average network latencies in two 3D NoC architectures are reduced by 28% and 34% respectively, comparing with their 2D counterparts. It is also shown that the heat dissipation is a trade-off consideration in improving the performance of 3D chips.

In Section 5.2, we study two hierarchical N-Body methods for Network-on-Chip (NoC) architectures. The N-Body problem is a classical problem of approximating the motion of bodies. Two methods, namely Barnes-Hut (Barnes) and Fast Multipole (FMM), have been developed for a fast simulation. The two algorithms have been implemented and studied in conventional computer systems and Graphics Processing Units (GPUs). However, as a promising multicore architecture, the evaluation of N-Body methods in a NoC platform has not been well addressed. We define a NoC model based on state-of-the-art systems. Experiments show that Barnes scales better (53.7x/Barnes and 36.6x/FMM for 64 processing elements) and requires less cache than FMM. However, we observe hot-spot traffic in Barnes. Our analysis and experiment results provide a guideline for studying N-Body methods in a NoC platform.

In Section 5.3, we propose an optimized NoC design for data parallel FFT applications. NoC based architecture is proposed for future multicore processors due to its scalability. FFT is widely used in digital systems. The implementation of FFT on conventional architectures have been studied. However, the evaluation of data parallel FFT in a NoC platform has not been well addressed. We analyse data parallel FFT in terms of traffic patterns and propose an optimized NoC design. Experiments show that the execution time of our optimized design is 12.13% faster than the original.

5.1 H.264

H.264, also known as Advanced Video Coding (AVC), is the latest international standard of video stream coding [86]. It has been defined in MPEG-4 part 10. Previous standards such as MPEG-2 have been widely used for coding video streams over digital television signal and video conference system. However, new applications and services such as camera phone and on-line video services require higher coding efficiency. H.264 has been used in a wide range of applications such as Blu-ray Disc, videos from YouTube and the iTunes Store, DVB broadcast, direct-broadcast satellite television service, cable television services, and real-time video conference. Not surprisingly, H.264 is hungry for higher processing power which requires parallelism and multicore computing. However, the current portable devices are usually suffering from the limited processing ability and low efficiency, thus a new chip design is required.

Most prior researches have focused on functional partition of the H.264 coding, including conventional architectures and NoCs. We analyze the impact of NoC design, and the influence of temperature and performance of 2D/3D NoC for data parallel H.264 coding in this section. The encoding and decoding processes of H.264 have been analyzed. We discuss the parallelism of H.264, and an open-source H.264 encoding program is used as a case study. The contribution of this section lies in the NoC design method and performance evaluation of data parallel H.264 coder. We present benchmark results using a cycle accurate full system simulator based on realistic workloads. Our analysis and experiment results provide a guideline to design efficient 3D NoCs for data parallel H.264 coding applications.

5.1.1 The H.264 Coding Standard

Like the traditional coding system such as MPEG-2 and MPEG-1, H.264 is based on motion compensation and inverted coding. However, in H.264, several advanced coding technologies have been introduced, such as multiple motion estimation, inter-frame estimation and multi-frame estimation [86].

A video sequence in H.264 is constructed by multiple groups of pictures, each of which includes several frames, and each frame includes one or more slices which are built by multiple macroblocks. A macroblock is a block shaped unit (e.g. 8x8 pixels) of associated luma and chroma sample [86].

The following are the three basic frame types in H.264.

- I Frame: All macroblocks in an I frame are coded by the intra motion prediction. The I frames are not optimized for motion compensation.

- P Frame: Certain macroblocks in a P frame are coded by intra motion prediction as those in I Frames, while others are coded by inter motion prediction with a motion compensated frame as a reference. The referred frame is usually the previous frame.

- B Frame: Certain macroblocks in a B frame are coded by the inter motion prediction which refers to two other motion compensated frames. The B frames are similar to the P frames except for the difference in the number of reference frames that are used. Better video quality and compression ratio can be achieved with well predicted B frames.

Both encoding and decoding processes are defined in H.264. Figure 5.1 and Figure 5.2 illustrate these processes respectively.

A frame from the video stream which has been divided into macroblocks is inputted for encoding (Figure 5.1). The Macroblocks are encoded in either inter or intra prediction mode (mode selection, MS) depending on their frame types. In the intra prediction (IP) mode, the macroblocks use samples in the same frame for coding while in the inter prediction mode, samples from the reference frames that have been motion estimated (ME) or motion compensated (MC) are used. After being subtracted from the current macroblock, the predicted macroblock produces a difference macroblock D_n which is then transformed using discrete cosine transform (DCT) and quantized (Q). Thereafter, the generated coefficients are encoded by the entropy coder for output. To encode a macroblock using reconstructed frames, the coefficients are inversely quantized (Q^{-1}) and inversely discrete cosine transformed (IDCT) to produce the difference macroblock D'_n. The

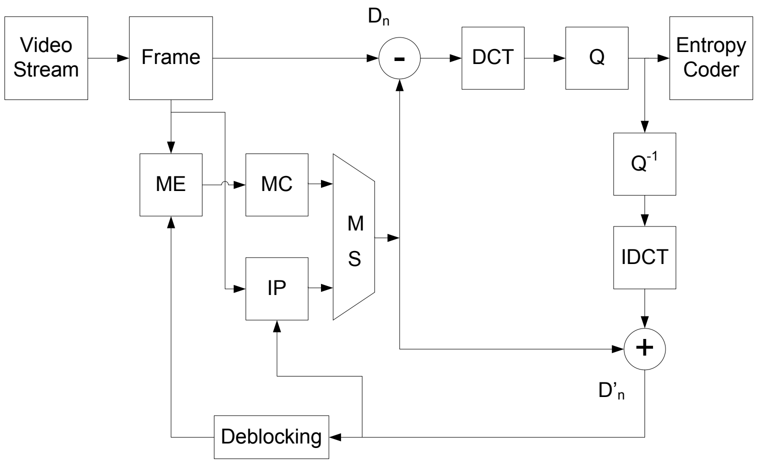


Figure 5.1: H.264 encoding process.

predicted macroblock is added to D'$_n$ and a deblocking filter is applied to minimize the distortion.

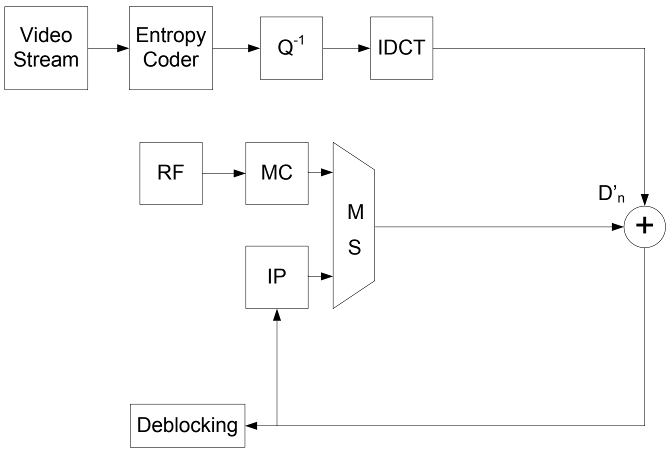


Figure 5.2: H.264 decoding process.

The input of the decoding process is the compressed video streams (Figure 5.2). The data are entropy decoded to a set of coefficients which then produce the difference macroblock D'$_n$ via Q^{-1} and IDCT. Based on the information in the video stream, reference frames (RF) can be selected for inter frame motion compensation. The compensated macroblock is added to D'$_n$ to generate the result which can be used as either intra prediction or output after deblocking. It is obvious that the decoding process of H.264 is included in the encoding process.

5.1.2 The Parallelism of H.264 Encoding

As discussed in the previous section, the decoding process of H.264 is a part of the encoding process, hence the analysis on the parallelization of the encoding process is suitable for the decoding process as well.

Functional Partition and Data Parallel

There are two approaches to design a H.264 coding system; functional partition and data parallel. In a functional partitioned system, tasks such as MC, ME, DCT and deblocking are assigned to individual PEs. Each PE is designed and optimized for a specific task to minimize its area and power consumption. Macroblocks are processed sequentially. However, this architecture is suffering from high transfer rates between PEs and unbalanced workloads. Furthermore, a functional partitioned design is usually applied to certain applications with simple control and fixed/static communication patterns like digital signal processing.

In data parallel design, in contrast to its functional partitioned counterpart, is applied to applications with enough independent data. For data parallel H.264 coding, video stream data such as macroblocks, slices and frames are distributed to PEs. Multiple video stream data can be processed simultaneously. Data dependency is a major constraint. Here, we will analyze two data parallelization methods at the slice and frame level.

Slice Level Parallelization

As is shown in Figure 5.3, the program divides a frame into a number of slices and each slice has its own independent structure. A thread is generated for each slice. After execution, a thread has to wait until other threads are completed. Apparently parallelization increases when more slices are divided. However, by dividing more slices, there will be a penalty to the quality of the video since each slice requires a few bits in the video stream. It is claimed in [103] that the peak signal-to-noise ratio decreases when more slices are divided. Furthermore, sequential portions of the program do not scale well as the number of PEs increases.

Frame Level Parallelization

Independent frame sequence is required to realize a full frame level parallelization. However, frames in H.264 are dependent on each other. For the three frame types I, P and B, an I frame need no reference frame, a P frame refers to the previous P frame and a B frame refers to the previous and next P frames.

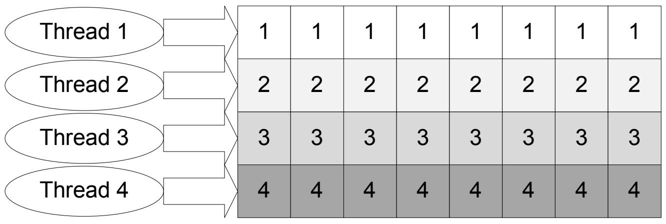


Figure 5.3: Slice level parallelization of a frame.

Take the video sequence in Figure 5.4 as an example. The first I frame refers to nothing, while the fourth frame (P) refers to the first I frame and is referred by the previous B frames (second and third) and the next P frame (seventh). Therefore, full parallelization of all frames is impossible due to the inter dependabilities in the frame sequences.

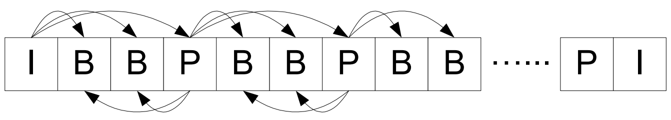


Figure 5.4: Dependency in a video sequence, with 2 B frames.

A thread Tn will be generated for each frame n. Earlier frames must be completed before later frames can be processed because the motion prediction and compensation involves previous frames (Figure 5.5).

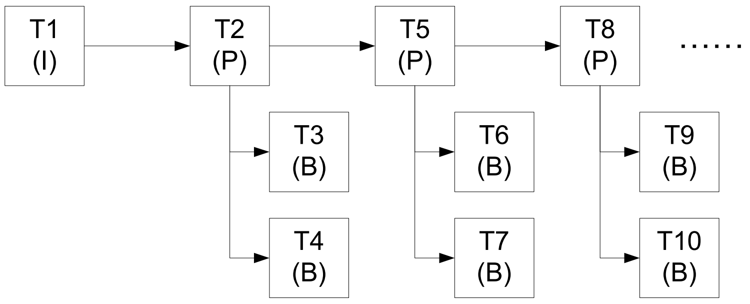


Figure 5.5: Frame level parallelization of a video sequence.

5.1.3 Case Study

In our research, the x264 has been chosen as case study for the analysis of parallelization. X264 is an open source H.264 video encoder which is a benchmark program in the Princeton Application Repository for Shared-Memory Computers (PARSEC) [18]. PARSEC benchmark suite is designed for studies of shared memory CMPs. It has been proved to be more accurate than SPLASH-2 when executing modern multithreaded applications [17]. The x264 program is multithreaded with threads including B-adapt (adaptive B-frame placement), rate-control, encoding and decoding. The coarse-grained pipeline parallelism is also achieved in x264.

It is shown in [18] that the speedup of x264 in a multicore system is limited because of the serial section in the code. Data dependency among x264 threads is heavy due to the shared reads and writes which are introduced by inter-thread communications. The shared data are deblocking pixels and reference frames that are used for the motion compensation and prediction in P and B frames. Moreover, data transfers from external memory to local cache are expensive in terms of latency and power consumption. If these shared data (e.g. the information of macroblocks) are stored and shared in a high performance on-chip network, cache miss rate and power consumption can be greatly reduced.

5.1.4 The Baseline 2D NoC Design for H.264

As aforementioned, data dependencies among threads are a major bottleneck of data parallel H.264 coding. Scalability of data parallel processing in a NoC requires minimized inter-communication between nodes. Figure 5.6 shows a NoC based 2D mesh SNUCA design which is optimized for data parallel H.264 coding. The cache banks are placed in the center of the 2D mesh topology while PEs are placed on the two sides. Each cache bank and PE is attached to a router which connects to the underlying on-chip interconnect network. The shared frame data for inter-thread communication are stored in the central cache banks.

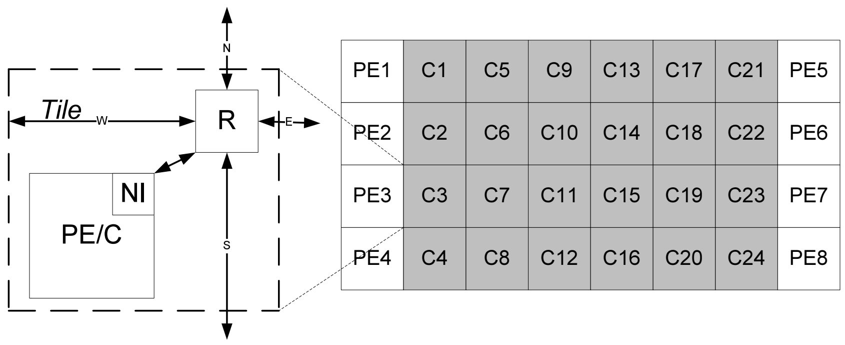


Figure 5.6: 2D NoC design based on SNUCA.

5.1.5 3D NoC Designs for H.264

In this section, two 3D NoC designs are studied, as shown in Figure 5.7. 3D-DL is achieved by simply folding the 2D design, leaving the PEs in both layers. In the 3D-top design, the PEs are gathered on the top layer for the efficient heat dissipation. Each design alternative has its pros and cons. For example, in terms of hop count, the 3D designs are lower than the 2D. Figure 5.8 shows the comparisons of the worst case and average hop counts among different architectures. The worst case hop count in the 2D design is 9, while in 3D-top with 4 pillars (top-4) it is reduced to 7 and eventually in 3D-DL with 4 pillars (DL-4) it is further decreased to 6. Similarly, the average hop counts in top-4 and DL-4 are respectively 17.5% and 21.9% less than those in the 2D design. From Figure 5.8, it is unsurprisingly obvious that the hop counts decreases with the increase in the number of pillars. The effect of pillar numbers in real workload is discussed in Section 6. Apparently, 3D-DL outperforms the 3D-top design in terms of hop count.

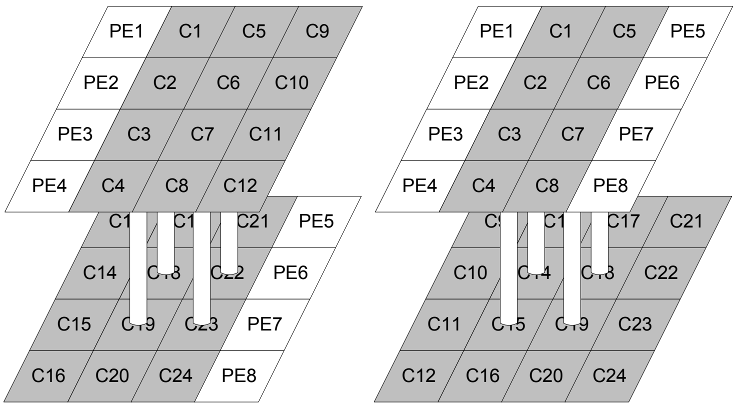


Figure 5.7: Two 2-layer 3D NoC designs with PEs distributed in layers (3D-DL) and on top (3D-top). Four pillars are shown for clarity.

However, comparing with the 3D-top design, the 3D-DL suffers from the severe constrain of thermal hot spots. In on-chip systems, PEs such as CPUs and DSPs usually produce much more heat than memory devices. As a result, the chip temperatures are usually high in the regions where the density of PEs is high. The high temperature often results in higher fault rates, process variations and shorter lifetime. In the 3D-top design, since all the PEs lie on the top layer where the heat can be easily dissipated into heat sinks, the hop spot problem is alleviated in a certain degree. On the contrary, in 3D-DL, the space between the two layers is so limited that the heat cannot dissipate efficiently.

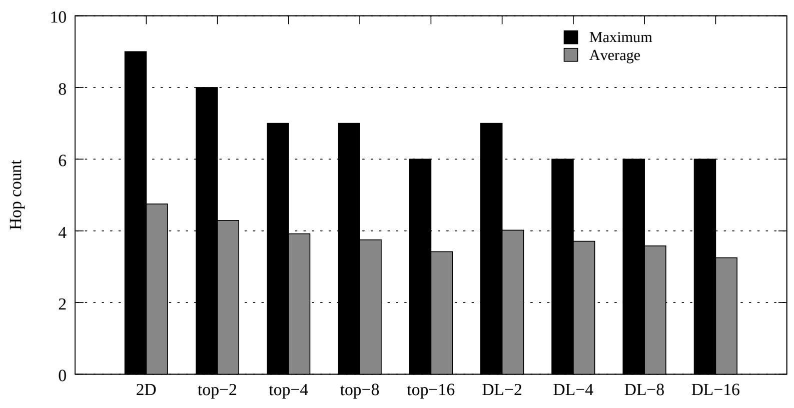


Figure 5.8: The comparison of hop count among 2D, 3D-DL and 3D-top.

On-chip temperatures of both the 2D and 3D implementations have been evaluated in our research. The HotSpot [97] 3D thermal model is used to simulate the thermal dissipation of the chip. HotSpot takes power trace and floorplan as inputs, and the corresponding transient temperatures as output. The simulator provides a thermal aware floorplanning tool which takes a list of connectivity between functional blocks such as caches and processors. The power consumption of a 512KB, 64-bit line size, 4-way associative, 90nm cache bank has been simulated by CACTI [94]. The power consumption of the processor cores is extracted from the Sun UltraSPARC T1 [60] design. Although power consumption profile differs from platform to platform, the analysis is still meaningful for 3D chip design.

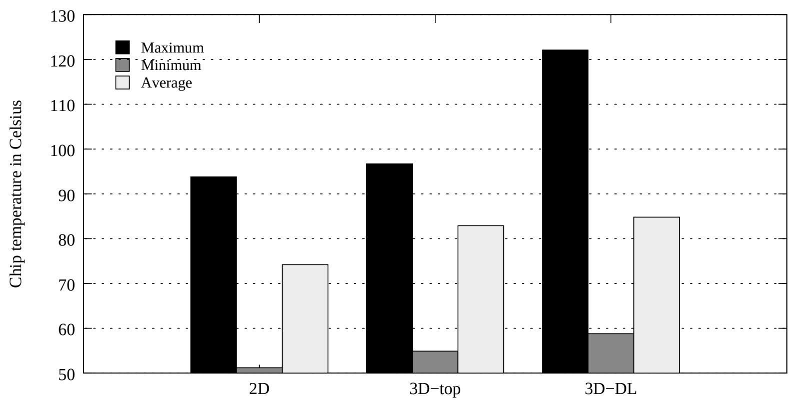


Figure 5.9: Chip temperature of 2D and two 3D configurations.

Figure 5.9 shows the evaluation results of chip temperatures in different system structures. In 3D-top design, the average chip temperature raises by 8°C comparing with the 2D design, while the maximum and minimum temperature does not have significant raise. However, in 3D-DL, the peak chip temperature raises by 29℃ comparing with the 2D design, which could be too high to be feasible for some applications. It is predictable that the temperature would be even higher by stacking more PEs vertically. To address this problem, dynamic voltage and frequency scaling (DVFS) can be used to cool down the chip or hot regions by lowering the working frequency and voltage, at the cost of performance degradation [133, 134, 15]. Obviously 3D-DL requires more scaling of voltage and frequency than 3D-top.

The above evaluation results show that the two aforementioned 3D architectures have a trade-off between network access efficiency and chip level thermal dissipation. System designers are recommended to choose either of these architectures by carefully examining the specifications of their applications and platforms.

5.1.6 Experimental Evaluation

In this section, we present the experimental evaluation for system performances based on the simulation of H.264 coding. X264 from PARSEC is used to test the encoding efficiency of different architectures including 2D, 3D-top and 3D-DL with different pillar numbers. We use a 32-node network which models a single-chip CMP for our experiments. The 3D architectures have two layers. The size of each cache bank node is 512KB. MESI cache coherence protocol is used in our memory hierarchy (see Chapter 2). Orion [111, 57], a power simulator for interconnection networks, is used to evaluate detailed power characteristics. Power consumption of both router and related links are calculated.

Three input video sequences have been selected for evaluation (Table 5.1). "simlarge" is a standard video clip from the PARSEC. It is taken from an open source movie called "Elephants Dream". This video clip models a high motion chasing scene. "Street" is a video clip which models a medium motion scene. "VConference(VConf)" is a video conference clip which models a low motion scene.

The effect of B frames has also been evaluated here. We first use the default parameters of x264 from PARSEC which have no specification of encoding B frames. We then simulate two scenarios in which 2 and 4 additional B frames have been tested respectively[1]. It is shown in Table 5.1 that different B frame configurations result in different compression ratio. The sizes of the output files with 2 extra B frames are reduced by 14.9% in "simlarge", 17.6% in "Street" and 1.5% in "VConference". But the effect

[1]The command line options are: **x264 –quite –qp 20 –partitions b8x8,i4x4 –ref 5 –direct auto –b-pyramid –weightb –mixed-refs –no-fast-pskip –me umh –subme 7 –analyse b8x8,i4x4 –threads 8 –bframes** ***n***

Table 5.1: Input video sequences

Video clip	**simlarge**	**Street**	**VConference**
Motion	High	Mid	Low
Frames	128	176	160
Resolution	640x360	704x400	640x480
RAW data	44,237KB	74,343KB	73,729KB
P only	3,171KB	2,370KB	598KB
B=2	2,759KB	2,016KB	579KB
B=4	2,758KB	2,006KB	579KB

is not obvious with the increase in the number of B frames. Therefore in our later experiments, we consider two configurations in which either only P frames or 2 B frames are used.

The full system simulation results are shown in Figure 5.10, 5.11 and 5.13. In Figure 5.10 and 5.11 we use the configuration of 4 pillars. As is shown in Figure 5.10, 3D NoC designs outperform the 2D NoCs in terms of network latency. Comparing with the 2D NoCs, the network latency of 3D-top with "simlarge" workload is reduced by 31% and 33% for P frame only and 2 B frames respectively. This is primarily due to the reduced number of hop counts of processor-cache data accesses in the 3D design, than it is in the 2D counterpart. Moreover, as mentioned before, shared data operation is intensive in x264 coding process. The improvement of "Street" is almost the same as that of "simlarge". With "VConf", the improvement is limited because of the smaller amount of shared data operations. For the 3D-DL, comparing with the 3D-top, network latencies of the three workloads are reduced by 7.1% and 10.1% for P frame only and 2 B frames respectively. However, heat dissipation is a trade-off consideration in this architecture.

The results in Figure 5.11 show that both 3D based topologies outperform 2D mesh in terms of execution time under all workloads. On average, 3D-top takes 18.8% shorter execution time than the 2D design, and the maximum time reduction reaches 25%. In 3D-DL the average execution time is reduced by 23.9%, and maximum decrease even reaches of 32%. The improvements of execution time can be interpreted as the result of network latency savings which commensurate with the number of shared cache accesses. The average execution times in the scenario with 2 B frames are 2.1% and 3.6% shorter than that in the scenario with P frame only in 3D-top and 3D-DL respectively. The improvement of execution time is less remarkable comparing with network latency since local operations are not related with network latency.

It is shown in Figure 5.12 that network latency advantages in 3D NoC designs translate into power consumption benefits. In comparison with the

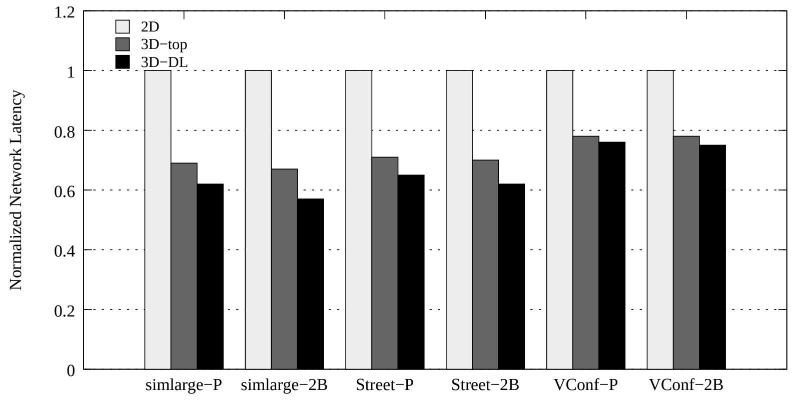


Figure 5.10: Normalized average network latency with different chip and video configurations.

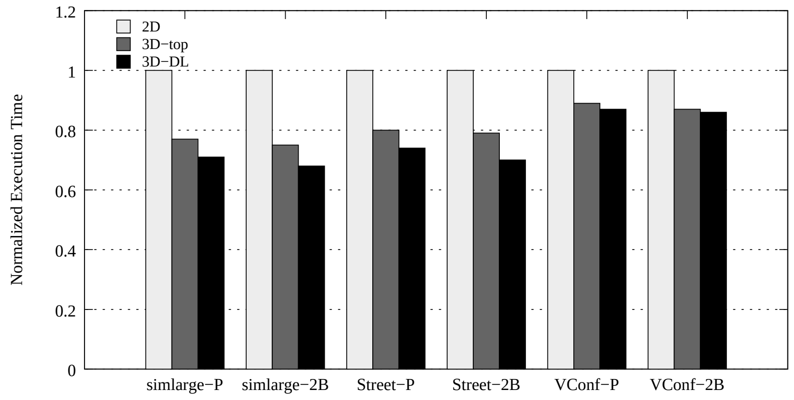


Figure 5.11: Normalized execution time with different chip and video configurations.

2D NoC, power savings for 3D NoCs are even better. Power consumption of 3D-top with "simlarge" workload is reduced by 48% and 51% for P frame only and 2 B frames respectively. The power savings are due to the structural advantages of the 3D NoCs, i.e. decreased hop count and network latency. The 3D NoC designs show lower power consumption than 2D design in "Street" and "VConf" as well. In the "Street" case, the power consumption of 3D-top and 3D-DL has decreased, on an average, by 52% and 48% over the 2D design. The improvement of power consumption in "VConf" is limited as above-mentioned.

The impact of the number of pillars is shown in Figure 5.13. The "simlarge" workload with P frame only is used for this evaluation. It is shown that network latencies in the 3D-DL and 3D-top architectures are reduced by 9% and 10% respectively by increasing the number of pillars

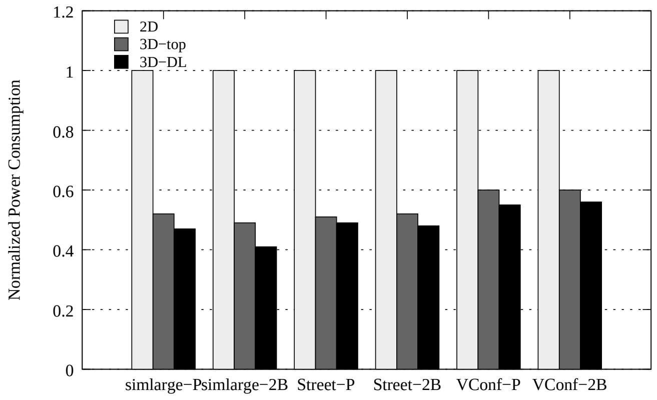


Figure 5.12: Normalized power consumption with different chip and video configurations.

from 2 to 4. This is mainly due to the decreased contention of inter-layer communications. Moreover, as shown in Figure 5.8, average processor-cache access hop count is reduced. By using 8 pillars, the latency reductions are 2.2% for 3D-DL and 4.4% for 3D-top comparing with the 4 pillars case. Maximum performance is achieved in the scenario where there are 16 pillars, i.e., each router has a pillar for the vertical communication. However, the simulation results show that in the systems with more than 4 pillars, the performance improvements are not significant.

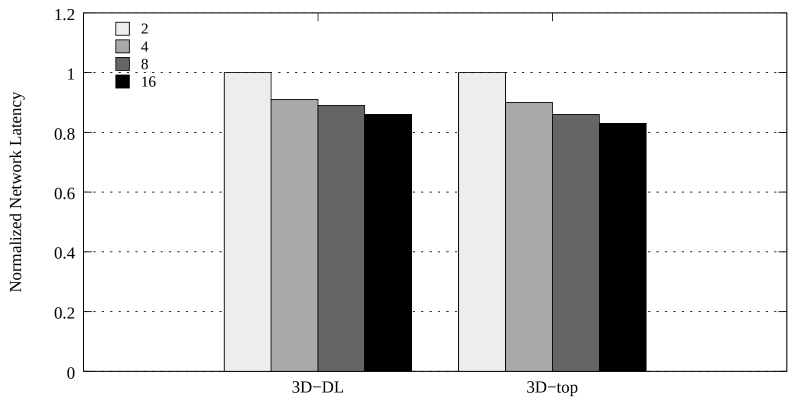


Figure 5.13: Normalized average network latency with different number of pillars.

As a combination of both performance and power consumption, the Performance Power Product (PPP) is more meaningful because performance is usually a trade-off for power consumption. Systems generally have better trade-off between the performance and power with higher PPP. The results in Figure 5.14 show that the PPP improves more significant from 2 pillars to 16 pillars, comparing with performance improvements. Configuration with 4 pillars performs 26.7% better than 2 pillars. Configuration with 8 pillars performs 14.3% better than 4 pillars. The highest efficiency is provided by full pillar connection, outperforming 8-pillar design by 15.6%. We noticed that the traffic contention between layers is alleviated by using more pillars, therefore overall network power consumption (link and router) is reduced with reduced execution time. This finding is of great importance since the design and implementation of vertical links are very expensive and therefore it is always desirable to have an optimized number of vertical links.

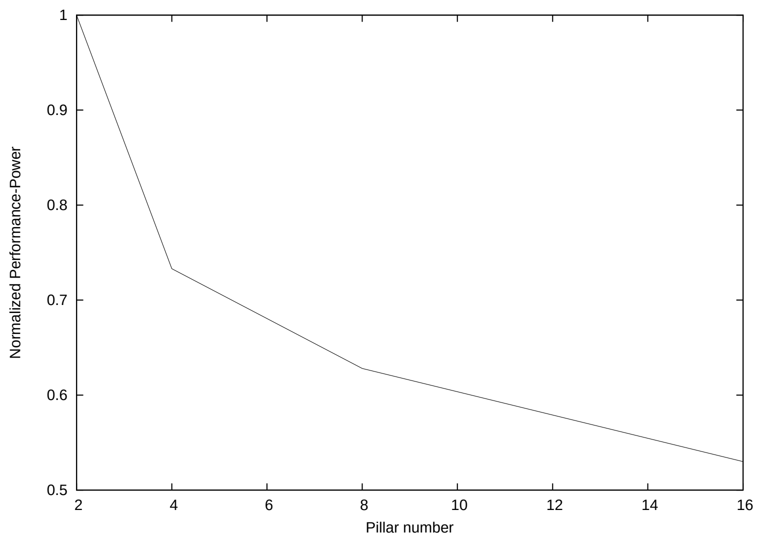


Figure 5.14: Normalized performance power product with different number of pillars.

5.2 Hierarchical N-Body

The N-Body problem is a classical problem of approximating the motion of bodies that interact with each other continuously. The bodies are usually galaxies and stars in an astrophysical system. The gravitational force of bodies is calculated according to Newton's Principia [1]. The N-Body problem is used in other computations and simulations as well, e.g. the interference of wireless cells and protein folding [87]. Several algorithms have been developed for N-Body simulation. In principle, to be precise, the simulation requires the calculation of all pairs, since the gravitational force is a long range force. However the computation complexity of this method is $O(n^2)$ [91]. J. Barnes et al. and L. Greengard introduced two fast hierarchical methods [13, 41]. A tree is built firstly according to the position of the bodies in the physical space. The interactions are calculated by traversing this tree. The computation complexities in these algorithms are reduced to $O(nlog_n)$, or even $O(n)$ in some cases.

The performance of these two algorithms has been studied in traditional cache-coherent shared address space multiprocessors, e.g. Standford DASH, KSR-1 and SGI-Challenge [45]. A simulator is used for examining the implications of the two algorithms in a multiprocessor architecture [96]. However, the previous works are based on conventional architectures, e.g. bus-based multiprocessors, physically distributed main memory or cache-only memory architecture. NVIDIA has presented a CUDA-based N-Body simulation by calculating the gravitational attractions of all body-pairs [79]. Hierarchical methods for GPGPU-based (General Purpose GPU) systems have been implemented and compared in [55] and [43]. As a promising multicore architecture in the future, the implementation of these algorithms in a NoC platform has not been well studied. To design efficient NoCs, designers need to understand the characteristics of the applications, e.g. the amount of communication among cores, caches and memory controllers, as well as the scaling of the application with the designated architecture. Here, we study and discuss two hierarchical N-Body algorithms for the NoC architecture. To validate our study, we model and analyze a 64-core NoC with 8×8 mesh, present the performance and network traces of the two algorithms using a full system simulator.

5.2.1 Modeling of the Network-on-Chip

To analyze the low-level behavior of an application, we model a NoC similar to the Tilera TILE architecture. Each processing core of the NoC is a Sun SPARC RISC core [105], which is 3.4mm^2 under 32nm. The simulation results by CACTI show that the total area of a 16MB cache with 64 banks, 64-bit line size, 4-way associative under 32nm is 64.61mm^2. Each cache bank, including data and tag, occupies 1mm^2. Routers are quite small compared with processors and caches, e.g. we calculate a 5-port router to be only 0.054mm^2 under 32nm. The number of transistors required for a memory controller is quite small compared with a chip (usually billions). It is presented that a DDR3 memory controller is about 2,000 LCs with Xilinx Virtex-5 Field-Programmable Gate Array (FPGA) [39]. The total area of the chip is estimated to be around 300mm^2, comparable to the TILE-Gx. Figure 5.15 illustrates the architecture of the aforementioned NoC.

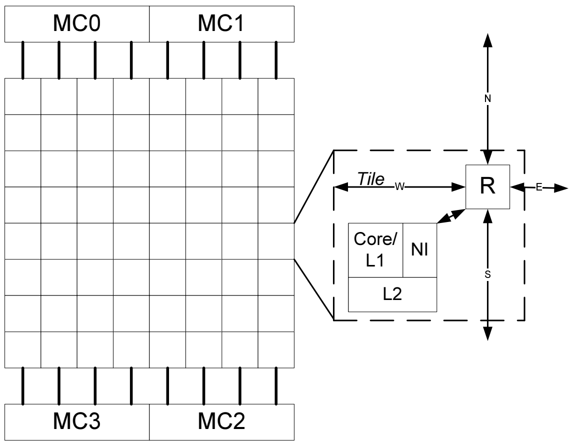


Figure 5.15: An 8×8 mesh-based NoC with memory controllers attached to up and down sides.

5.2.2 The Hierarchical N-Body Methods

In this section, we describe the two most important hierarchical N-Body algorithms that we used for analysis: the Barnes-Hut method [13] and the Fast Multiple Method (FMM) [41]. The two hierarchical methods build a structured tree in the first step. The tree is built by subdividing space cells until a certain condition, e.g. reaching the maximum number of particles in a leaf cell. The physical space is represented by a hierarchical tree. The computation of interactions is done by traversing this tree. The two algorithms differ in the steps they use to calculate the interactions of particles.

In Barnes-Hut method, for each particle, the tree is traversed to compute the forces. It starts at the root of the tree, and traverses every cell. To reduce the computation complexity of long-range interactions, the subtree is approximated by the mass of the center cell, if the cell is far away from the particle. The accuracy of this methods is thus dependent on the approximation metrics. The Barnes-Hut method only computes the interactions for particle-particle and particle-cell.

The FMM computes the interactions for cell-cell as well, compared with Barnes-Hut. If two cells are far away from each other, the interaction between them is computed by the multipole expansion of the cells. The computation complexity is thus reduced. For uniform distributions, the complexity of FMM is $O(n)$, compared with $O(nlog_n)$ in Barnes-Hut. To develop a multithreaded program for both algorithms, the space is divided into several regions where each core is assigned with a region. A tree for the regions is built for the responsible core, and each core calculates its local tree. Most of the calculation time is spent in traversals of the tree to compute the forces. In a NoC platform, the performance of the algorithms will be affected by *(a)* long distance communication of nodes; *(b)* the initial distribution of particles; *(c)* the dynamic changing of position of particles; *(d)* hot-spot traffic.

5.2.3 Experimental Evaluation

We use a 64-core network which models a single-chip NoC for our experiments. Each processor core is running at 2GHz, attached to a wormhole router and has a private write-back L1 cache (split I+D, each 32KB, 4-way, 64-bit line, 3-cycle). The 16MB L2 cache shared by all processors is split into banks (64 banks, each 256KB, 64-bit line, 6-cycle). For both methods, we use the Plummer model [31] for particle generation, instead of uniform distribution. The multithreaded part of the programs utilizes the C/pthread model.

We start by evaluating the computation time distribution and scalability of the two algorithms. Both algorithms apply the same parameters. The results are listed in Table 5.2 and 5.3. The first five rows show the computation time from 4K to 64K particles, with 64 processors. In Barnes-Hut, around 90% of the total time are spent on force calculation (84.2% in 4K to 91.1% in 64K), while the time spent on other tasks (e.g. tree building) are relatively small. The Barnes-Hut method scales very well from 1 to 64 processors. The speedups for 64 processors are 53.7x and 61.8x for total execution time and force calculation time, respectively.

Table 5.2: Time distribution and scalability of the Barnes-Hut Method

Configuration	**Total time**	**Treebuild**	**Forcecalc**	**Others**
64p/4K	19	1	16	2
64p/8K	41	2	35	4
64p/16K	87	5	76	6
64p/32K	184	8	168	8
64p/64K	385	15	351	19
4K/1p	1020	28	988	4
4K/2p	511	15	495	1
4K/4p	258	8	246	4
4K/8p	129	4	124	1
4K/16p	65	3	61	1
4K/32p	34	2	31	1
4K/64p	19	1	16	2

In Figure 5.16, the network request rates of 64 cores are illustrated. We

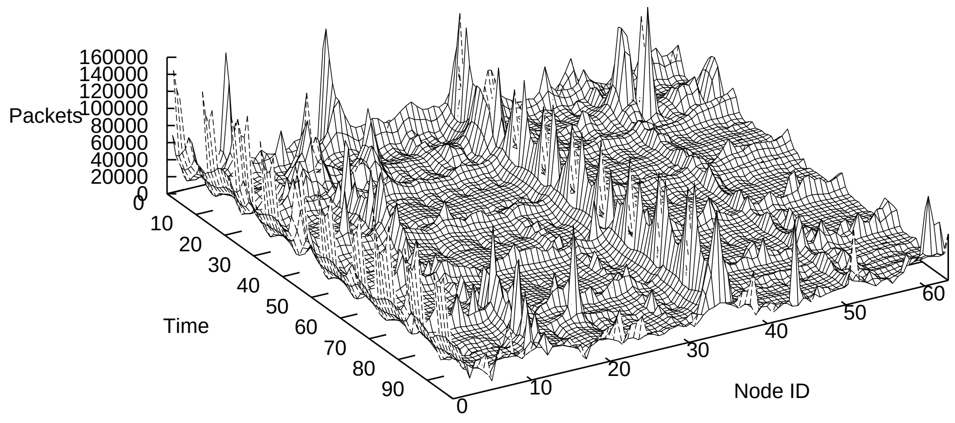


Figure 5.16: Network request rate for 64-core NoC running Barnes.

simulate 64K particles in 5 time steps. The horizontal axis is time, segmented in 12.1M-cycle percentage fragments. The traffic trace has 165.2M packets. It is observed that, several nodes, especially N0 and N34, generate more data traffic than others (e.g. N0 14.18%, N34 12.19%, N12 5.3% and N20 2.76%). This introduces heavy hot-spot traffic in certain regions of the NoC. Notice that the traffic patterns of other nodes are quite similar, they have a high traffic in the starting phase, and drop to a lower traffic after that. There are several time slices, for example 16% to 21%, when all processors are sending packets simultaneously. The reason is, the simulation has executed for 5 time steps, the positions of particles will change at the end of each time step. In terms of point-to-point traffic, several source-destination pairs, specifically originated from N0 and N34, generated a considerable amount of the traffic. We observe the top 5 pairs are: 34-62 (0.88%), 0-14 (0.63%), 0-58 (0.62%), 0-8 (0.60%) and 34-10 (0.51%). These hot-spot traffic can be alleviated with, e.g. long links between nodes, or increase the link bandwidth for hot-spot nodes.

The time spent on force calculation in the Fast Multipole method is lower than Barnes-Hut (Table 5.3), e.g. 58.8% in 4K to 70.3% in 64K. Nearly 10% of the time are spent on tree building, and about 15% on barrier. The Fast Multipole method scales worse than Barnes. The speedups for 64 processors are 36.6x and 53.3x for total execution time and force calculation time, respectively. This is primarily due to the higher number of barriers in Fast Multipole method. It is noteworthy that in spite of poor scaling, the Fast Multipole method spends less time for calculation. For example, it spends 54.3% of the total execution time in 64p/64K, compared with Barnes. In consideration of better scalability, the Barnes-Hut method could use shorter time in a systems with thousands of cores.

Table 5.3: Time distribution and scalability of the Fast Multipole Method

Conf.	**T.Time**	**T.build**	**F.calc**	**Barrier**	**L.build**	**Others**
64p/4K	17	1	10	3	2	1
64p/8K	27	2	16	6	0	3
64p/16K	54	7	30	14	2	1
64p/32K	102	11	73	13	1	4
64p/64K	209	21	147	30	4	7
4K/1p	622	75	533	0	10	4
4K/2p	316	38	270	1	3	4
4K/4p	162	20	136	1	3	2
4K/8p	83	9	71	0	1	2
4K/16p	44	4	35	2	1	2
4K/32p	26	3	16	4	0	3
4K/64p	17	1	10	3	2	1

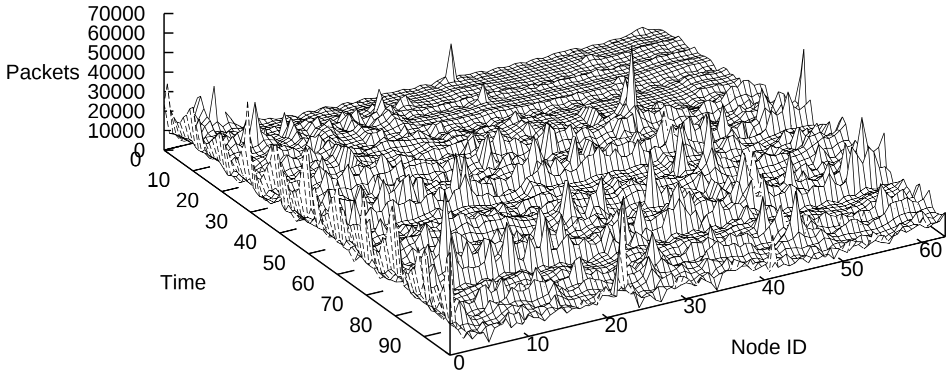


Figure 5.17: Network request rate for 64-core NoC running FMM.

Figure 5.17 shows the network request rate of each processing core when running FMM in a 64-core NoC. The horizontal axis is time, segmented in 1.69M-cycle percentage fragments. The traffic trace has 57.4M packets. It is revealed that several nodes (e.g. N0 7.6%, N46 4.15%, N13 2.72% and N7 2.71%) generate more data traffic than others. The network traffic is relatively low in the starting phase (before 30% of the time slice). After that time point, FMM shows similar traffic patterns as in Barnes. However, the hot-spot traffic in FMM is not as significant as Barnes. We note that, in terms of point-to-point traffic, a small portion of source-destination pairs generated a sizable portion of the traffic. For example, only 4 (19-60, 13-44, 60-19 and 0-29) of the pairs (in totally $64^2 = 4,096$) generated 1.42% traffic.

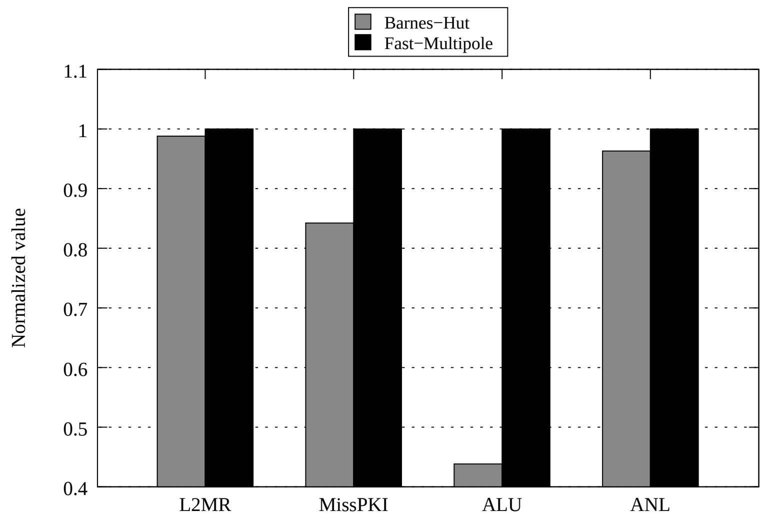


Figure 5.18: Performance for Barnes and FMM.

We evaluate other performance metrics of the two algorithms in terms of L2 cache miss rate (L2MR), misses per thousand instructions (MissPKI), Average Link Utilization (ALU) and Average Network Latency (ANL). ALU is calculated with the number of packets transferred between NoC resources per cycle. ANL represents the average number of cycles required for the transmission of all messages. The number of required cycles for each message is calculated from the injection of the message header into the network at the source node, to the reception of the tail flit at the destination node. Under the same configuration and workload, lower values of these metrics are favorable. The results are shown in Figure 5.18. We note that, in terms of L2MR and MissPKI, Barnes is lower than FMM (1.21% for L2MR and 15.77% for MissPKI respectively). This reflects, FMM requires more cache than Barnes. A system with limited cache could be unsuitable for FMM. The ALU of Barnes is only 43.83% of FMM, which means an alleviated network load. It is noteworthy that despite the fact that the value of Z axis in Figure 5.16 is twice as larger than that in Figure 5.17, each time slice in Figure 5.16 represents 12.1M cycles, compared with 1.69M cycles in Figure 5.17. Finally, the ANL of Barnes is 96.31% that of FMM, indicating that the network performance of Barnes is better, and hence having lower communication overhead.

5.3 FFT

The Fast Fourier transform (FFT) is a fast algorithm to compute the discrete Fourier transform. FFT is widely used in digital signal processing, solving partial differential equations and multiplication of large integers. Broadband wireless communication is a famous application field that heavily rely on FFT. In modern wireless communication, Orthogonal Frequency-Division Multiplexing (OFDM) is developed and widely used, due to the fact that OFDM is capable of coping with bad channel conditions (e.g. IEEE 802.11a/g/n WLAN, IEEE 802.16 WiMAX, 3GPP Long Term Evolution and DVB-H for mobile TV) [10, 132, 34, 8, 35]. In OFDM, FFT is implemented on the receiver side and inverse FFT on the sender side to achieve efficient mmodulation and demodulation. Previous generations of wireless standards, e.g. IEEE 802.11a, use an FFT of 64 points. Latest standards, e.g. 802.16, scale the FFT to the channel bandwidth. The allowed FFT subcarrier numbers are up to 2,048 in 802.16 and 8,192 in DVB-H, respectively.

There are many FFT algorithm implementations, the most common FFT is the Cooley-Tukey algorithm [26]. Other algorithms have been proposed to reduce complexity of FFT, including reducing the required multiplications and additions. A famous algorithm is the split-radix FFT, which achieves the lowest arithmetic operation count [32]. Implementing FFTs on multi-processor systems has been studied in [11] and [4]. However, the implementation and optimization of data parallel FFT in a NoC platform have not been well addressed. In our section, we analyse a data parallel FFT algorithm with on-chip traffic trace data, propose and discuss a novel optimized NoC architecture which aims to reduce the latency of long distance communications and improve the efficiency of data parallel FFT. To confirm our study, we model a NoC with 4×4 mesh, present the performance of the data parallel FFT with different NoC designs using a full system simulator (refer to Figure 2.1 for details).

5.3.1 The FFT Algorithm

We select a one-dimensional, radix-n, six-step FFT algorithm from [12]. There are two input data sets, one with n^2 complex data points is to be transformed, and the other with n^2 complex data points is referred as the roots of unity. The two data sets are organized and partitioned as $n{\times}n$ matrices, a partition of contiguous set of rows is assigned to a processor and distributed to its local cache. The six steps are: (1), Transpose the input data set matrix; (2), Perform one-dimensional FFTs on the resulting matrix; (3), Multiply the resulting matrix by roots of unity; (4), Transpose the resulting matrix. (5), Perform one-dimensional FFTs on the resulting matrix; (6), Transpose the resulting matrix. The communication among processors can be a bottleneck in the three matrix transpose steps. During the matrix transpose step, a processor transposes a contiguous sub-matrix locally, and a sub-matrix from every other processor. The transpose step requires communication of all processors. It is shown in [117] that fast data transfer between processors is the most dominant factor for this application (Step (1), (4) and (6)). Traffic hotspots and contentions could occur in an unoptimized system, and thus the overall performance is degraded.

5.3.2 FFT Traffic Pattern

Firstly, we need a detailed overview of the traffic. Figure 5.19 shows the network request rate of each PE when running FFT in a 16-core NoC under GEMS/Simics simulation environment. In Figure 5.19, the horizontal axis is time, segmented in 216K-cycle percentage fragments. The traffic trace has 1.64M packets, with 21.6M cycles executed. The traffic is shown for all the 16 nodes. It is revealed that 63.9% of data traffic are concentrated on five nodes (N0 29.6%, N8 6.7%, N11 10.0%, N13 8.7% and N15 8.8%, refer to Figure 2.1 for numbering of nodes). There is a traffic peak for all nodes during the last stage of execution (around 80% of the time). Three nodes (N0, N8 and N15) have hotspot traffic in the beginning. The top point-to-point traffics are listed in Table 5.4. A small portion of source-destination pairs generated a sizable portion of the traffic, e.g. 3.13% of the pairs (8/256) generated 32.07% traffic. Notice that traffic between N0 and N11 contributed 10.97% of total volume.

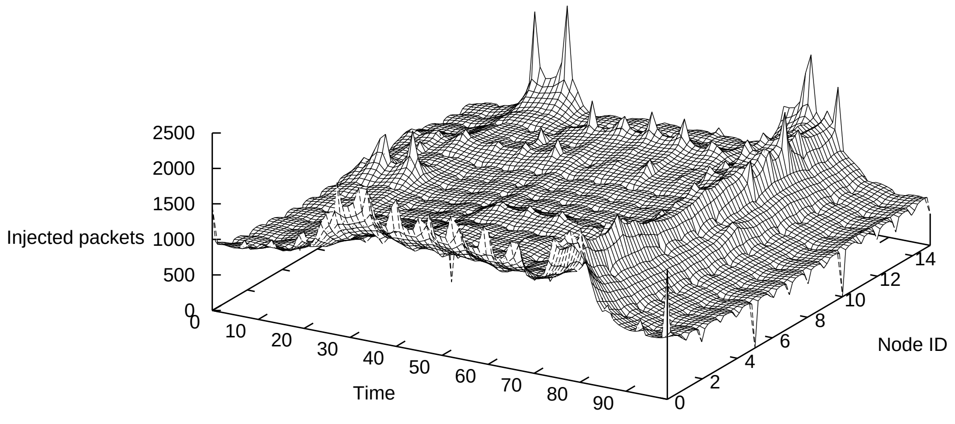


Figure 5.19: Network request rate for 16-core NoC running FFT. The time is segmented in 216K-cycle/percentage.

Table 5.4: Top Point-to-Point traffics.

Source Node	Destination Node	Traffic Percentage
0	11	7.43
0	4	4.11
0	3	3.94
15	11	3.66
13	6	3.63
11	0	3.54
0	12	3.49
8	11	2.27

5.3.3 An Optimized Network-on-Chip Design

Assuming X-Y deterministic routing, Equation 5.1 shows the access time (latency) required for a core-core communication. The latency involves in-tile links (Between NI and PE, L_{Link_delay1}), router (L_{Router_delay}), tile-tile links (L_{Link_delay2}) and the number of hops required to reach the destination (n_{hop}).

$$L = (n_{hop} + 1)\times L_{Router\_delay} + 2\times L_{Link\_delay1} + n_{hop}\times L_{Link\_delay2} \quad (5.1)$$

In order to evaluate the detailed number of cycles required for each of these metrics, we model the NoC according to Sun SPARC. Each SPARC core with private L1 cache has an area of 14.45mm^2 with 65nm fabrication technology. Results from CACTI show that (16MB, 16 banks, 64-bit line size, 16-way associative, 65nm), each cache bank, including data and tag, occupies 12.09mm^2. We calculate a 5-port router is estimated to be 0.23mm^2 under 65nm. Hence the area for a tile of the NoC is around 14.45 + 12.09 + 0.23 = 26.77mm^2. Considering a NoC with 16 tiles, the total area is about 428.32mm^2, comparable with modern chip multiprocessors, such as Sun SPARC and IBM Power 7. In this research, we assume that each tile and each router is of a square shape, and thus the length of an edge is 5.17mm and 0.48mm for a tile and a router, respectively.

We calculate the delay for the links between routers, NIs, cores and caches using Cadence Spectre, since the latency will be determined by the physical length of the link. For inter-router long links in voltage-mode transmission, the wire delay is significant. Repeaters are inserted to reduce the wire delay in long links over 0.5mm. The delays are calculated under 2.5GHz, with a cycle of 400ps. Notice that the in-tile links between NI and router are very short, e.g. Less than 0.5mm. The transmission can be completed within one cycle. For a router in the NoC, there are several

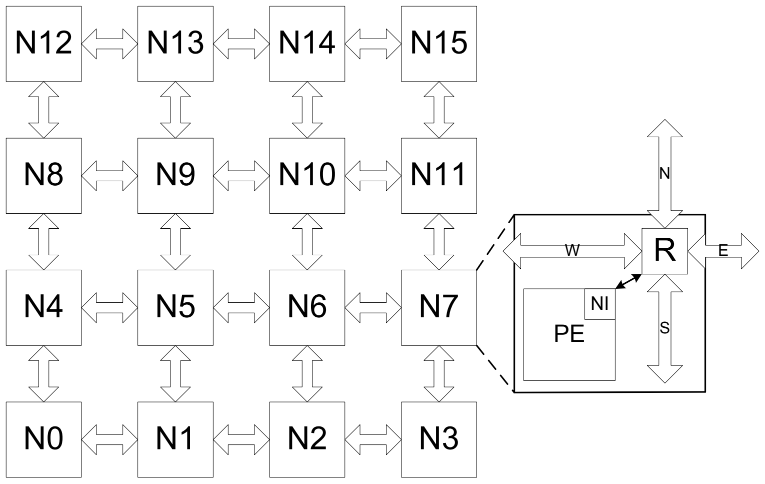


Figure 5.20: A diagram showing NoC model used here.

parts (e.g. RCU, VCA, SA and CS) that will affect latency, depending on the number of pipeline stages. Here, we use a standard router of four pipeline stages. The tile-tile links are much longer than in-tile links. For example, the length of a tile-tile link connecting two routers is 4.69mm. In consideration of the synchronization of these pipelined long links, a data transfer requires 6 cycles. Assuming a packet transmitted from N0 to N11, it will go through N1, N2, N3 and N7, resulting 5 hops (Figure 5.20). Hence the latency is calculated as: $2\times1+(5+1)\times4+5\times6=56$ cycles.

By noticing that some source-destination pairs generate significant amount of traffic, we propose direct long links as an optimization method. The delays of intermediate routers are eliminated. For instance, a long link can be placed between N0 and N11 directly. In this case, the latencies of L_{Router_delay} for N1, N2, N3 and N7 will be eliminated. However, the number of links that can be routed in a NoC is limited by the router size and the area of the links. The limitation of router area is more significant than the long links. A typical router in a 2D mesh NoC has five ports to connect to five directions, namely, North, East, West, South and Local PE. This requires a 5×5 crossbar. Researches have shown that [82], crossbar occupies over 50% of the router area. A 7×7 crossbar doubles the area compared with 5×5. Therefore, adding too many long links can be undesirable. We note that the router of N0 has only 3 out of 5 ports utilized (North, East and Local PE), which leaves 2 free ports. Other routers have free ports as well, e.g. N3 and N11 have 2 and 1 free ports, respectively. In our optimized design, we connect N0-N11 and N0-N3 with long links. Other pairs are not practical with long links, e.g. despite the fact that the communication between N0 and N4 is more frequent than N0 and N3, they are directly connected. Connecting N6 with N13 will require an expansion of the crossbar of router in N6, which is not favourable. Equation 5.2 shows the latency for a core-core communication with long links.

$$L_L = 2\times(L_{Link\_delay1} + L_{Router\_delay}) + L_{Longlink\_delay} \tag{5.2}$$

The latency of the long links ($L_{Longlink\_delay}$) between N0-N3 and N0-N11 will be much higher than L_{Link_delay2}. We calculate that the length of link between N0 and N3 is 15.03mm. Based on the aforementioned wire delay model, a data transfer requires 18 cycles under 2.5GHz. Comparing with the original communication delay (2×1+(3+1)×4+3×6=36 cycles), the delay of N0-N3 long link is reduced to 28 cycles. The savings are mainly from two routers. The length of N0-N11 long link is 24.89mm, resulting 32 cycles for a data transfer. The reduction of the communication latency for long links between N0 and N11 is higher than N0 and N3 (42/56=0.75 and 28/36=0.78 respectively). Taking into account of the 10.97% communication volume between N0 and N11, system performance can improve with our optimization. It is noteworthy that placing long links all over the NoC will incur higher design complexity of both hardware and software.

5.3.4 Experimental Evaluation

The simulation platform contains a 16-core network. Each PE is running at 2GHz, attached to a wormhole router and has a private write-back L1 cache (split I+D, each 32KB, 4-way, 64-bit line, 3-cycle). The 16MB L2 cache shared by all processors is split into banks (16 banks, each 1MB, 64-bit line, 6-cycle). We setup a system with 4GB of main memory, and the latency from the main memory to the L2 cache is 260 cycles. The MOESI cache coherence protocol is implemented.

We evaluate performance in terms of Average Network Latency (ANL), Average Link Utilization (ALU), Execution Time (ET) and Cache Hit Latencies (CHL). ANL represents the number of average cycles required for the transmission of all network messages. The number of cycles of each message is calculated as, from the injection of the message header into the network at the source node, to the reception of the tail flit at the destination node. ALU is defined as the number of flits transferred between NoC resources per cycle. Under the same configuration and workload, lower values of these metrics are favourable.

The results are depicted in Figure 5.21. Our optimized NoC architecture outperforms the original design in all metrics. For example, the ANL for the optimized NoC is 9.89% lower than the original. This is primarily due to the lower latencies between hotspot nodes, e.g. N0-N11 and N0-N3,

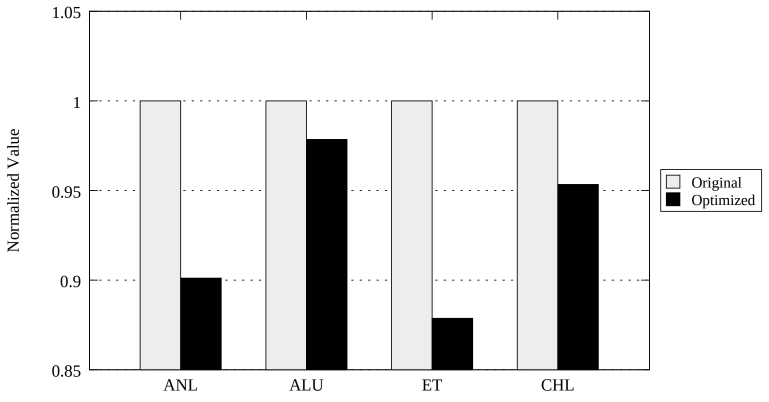


Figure 5.21: Normalized average network latency with different number of pillars.

compared with the original design. As aforementioned, the transpose steps in FFT require communication of all processors (especially the last stage, see Figure 2). We note that the communication is not evenly distributed to all processors. In this case, reducing the delay of the hopspot nodes by adding long links is a feasible method. The ALU of FFT for our optimized design is 2.15% lower than the original as well. Apparently, the improvement is not as significant as ANL. The reason is that there are only two additional links which can alleviate the overall link load. The CHL in our design is 4.66% lower than the original, because of the reduced latencies. Overall, in terms of ET, our design uses 12.13% less time than the original. This reflects the savings of ANL, ALU and CHL.

Chapter 6

Operating System Scheduling

The Operating System (OS) scheduling (or dynamic task mapping, the two terms are considered same here) is one of the most important design issues for future CMP systems. Higher performance can be achieved by proper scheduling of processes and threads. For example, the modular design of AMD Bulldozer CMP benefits from better scheduling [36, 21]. Each Bulldozer module contains two discrete integer cores with several shared resources, including the floating-point cores, the front-end instruction fetch and decode units, the L2 caches, and the associated data pre-fetchers. This shared nature of the modules requires OS and application developers to optimize their programs for better performance. Results shown that, with proper scheduling, the performance is 10% to 20% higher in three applications [113]. Dynamic task mapping is studied in MPSoC and NoC architectures as well for various optimization goals, for example energy consumption [23], network contention [24, 25], communication volume [30], inter-task dependency [74] and so on. Here, two scheduling algorithms are proposed, one focuses on optimizing memory access and inter process communication (IPC), while the other aims to reduce cache access latencies in 3D NoCs.

In Section 6.1, limitations of state-of-the-art OS scheduler are discussed, with Sun Solaris used as a case study. The contribution of this section lies in the on-chip data traffic calculation of running applications. By evaluating FFT and SPECjbb as benchmarks, it is shown that the Solaris scheduler does not provide the optimal communication scheme and thus suffers from the network latency and overall performance degradation. We define a model for future CMPs, based on which a minimal average accessing time scheduling algorithm is proposed to reduce on-chip communication latencies and improve performance. The impact of memory access and IPC in scheduling are analyzed. We explore six typical core allocation strategies.

A protocol for OS-level implementation of the algorithm has been proposed. Results show that a strategy with the minimal average accessing time of both core-core and core-memory outperforms other strategies, the overall performance for three applications (FFT, LU and H.264) has improved for 8.23%, 4.81% and 10.21% respectively comparing with other strategies.

In Section 6.2, we propose a greedy heuristic approximation scheduling algorithm for future 3D CMPs. To reduce on-chip communication delay, 3D integration with TSVs is introduced to replace the 2D counterpart. Multiple functional layers can be stacked in a 3D CMP. However, operating system process scheduling has not been well addressed for such a system. We define a model for future 3D CMPs, based on which a scheduling algorithm is proposed to reduce cache access latencies and the delay of IPC. We explore different scheduling possibilities and discuss the advantages and disadvantages of our algorithm. Experiments show that under two workloads, the execution times of our scheduling in two configurations (2 and 4 threads) are reduced by 15.58% and 8.13% respectively, compared with the other schedulings.

6.1 A Minimal Average Access Time Scheduler

The design of OS schedulers is one of the most important issues for CMPs. For large scale CMPs such as hundred-core chips, it is obvious that scheduling of multi-threaded tasks to achieve better or even optimal efficiency is crucial in the future. Several multiprocessor scheduling policies such as round robin, co-scheduling and dynamic partitioning have been studied and compared in [69]. However, these policies are designed mainly for the conventional shared bus based communication architecture. Many heuristic-based scheduling methods have been proposed [42]. These methods are based on different assumptions, e.g. the prior knowledge of the tasks and execution time of each task in a program, presented as a directed acyclic graph. Hypercube scheduling has been proposed for off-chip systems [93]. Hypercube systems are usually based on Non-Uniform Memory Access (NUMA) or cache coherent NUMA architectures [64], which are different from CMPs. It is claimed in [2] that the network latency is greatly affected by the distance between core and memory controller. Therefore, how to reduce the distances is one of the main considerations in our approach. However, the work in [2] is based on enumerating all possible permutations of memory controller placement explicitly beforehand. While in our study, we focus on the other side instead of hardware design. Task scheduling for NoC platforms is studied in [22] and [46]. The effect of memory controller placement is not considered in these papers. In this section, we propose and discuss a novel scheduler for NoC-based CMPs which aims to minimize the average network latency between memories and processing cores. With the decrease of average network latency, lower power consumption and higher performance can be achieved. To confirm our theory, we model and analyze a 64-core NoC with 8×8 mesh, present the performances with different allocation strategies using a full system simulator.

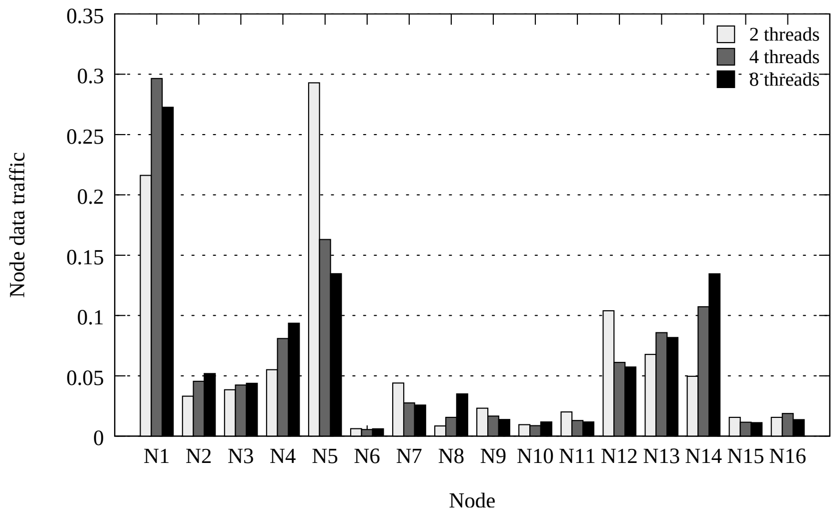
6.1.1 Evaluation of Scheduler under NoC-based CMP

In this section, full system simulation is performed to gain further insight into the scheduling issues. We use a configuration with 16 cores which models a single-chip CMP for our experiments. Each PE has a core, a private L1 cache and a shared L2 cache bank (16 banks, each 512KB). Memory controllers are connected to the top and bottom side of the chip. NUCA [14] is implemented in our memory/cache architecture. Compared with UCA, which has been used in traditional commercial multicore processors, NUCA has more flexible cache access latencies and thus improves the system performance. This is due to the fact that in UCA, the unified cache access latency is determined by the worst case wire delay. The MESI [83] cache coherence protocol is used in our memory hierarchy, in which each L2 bank has its own directory. Cache data and related messages such as *Get exclusive* and *Get shared*; and memory data and related messages such as *Memory data* have been collected and calculated.

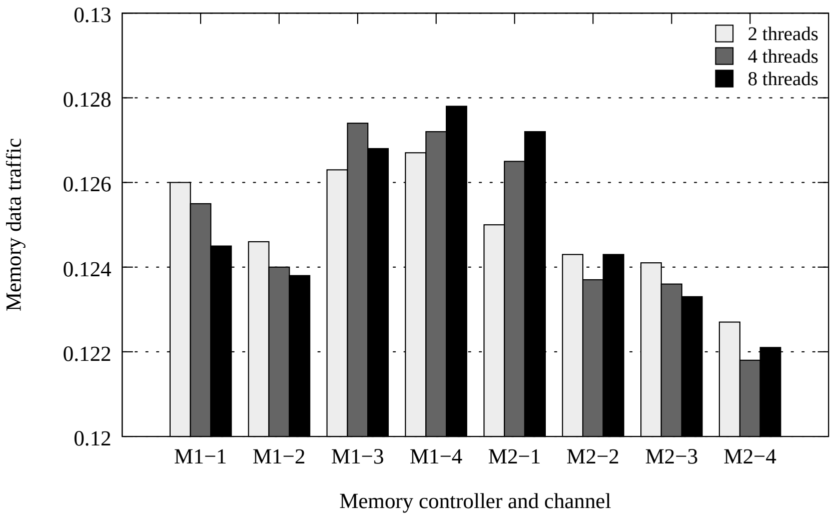
We evaluated the scheduling algorithm of Sun Solaris 9. Results are presented for FFT [116] and SPECjbb2000 [98] workloads with different numbers of threads or warehouses involved. The Java platform is based on Sun JRE-SE 1.4.2.

Figure 6.1 shows the analysis results of the study on FFT in which configurations with 2, 4 and 8 threads are tested. As anticipated, the Solaris scheduler allocates cores using a non-optimal way. Nodes N1, N5 and N12 generated the most data traffic (totally 61.3%, Figure 6.1a) with 2-thread FFT. It is noteworthy that the OS itself has to be scheduled to at least one core. Nodes N1, N4, N5 and N12-N14 are selected with 4-thread FFT, generated 79.46% of data traffic. In 8-thread FFT, nodes N1-N5, N8 and N12-N14 are selected, 90.51% data traffic are concentrated on these nodes. The traffic of memory controller is unbalanced as well. It is obvious from Figure 6.1b that memory data traffic are concentrated on channels 1, 3 and 4 of memory controller 1 and channel 1 of memory controller 2, corresponding with the above node traffic.

The results in Figure 6.2a and Figure 6.2b show the SPECjbb node and memory data traffic distribution. Nodes N1, N9 and N14 generate totally 55.17% data traffic in 2-warehouse SPECjbb, while in 4-warehouse configuration 67.38% data traffic are concentrated on N1, N4, N9, N13 and N14. 82.92% of total data traffic are distributed among N1, N3, N4, N6,



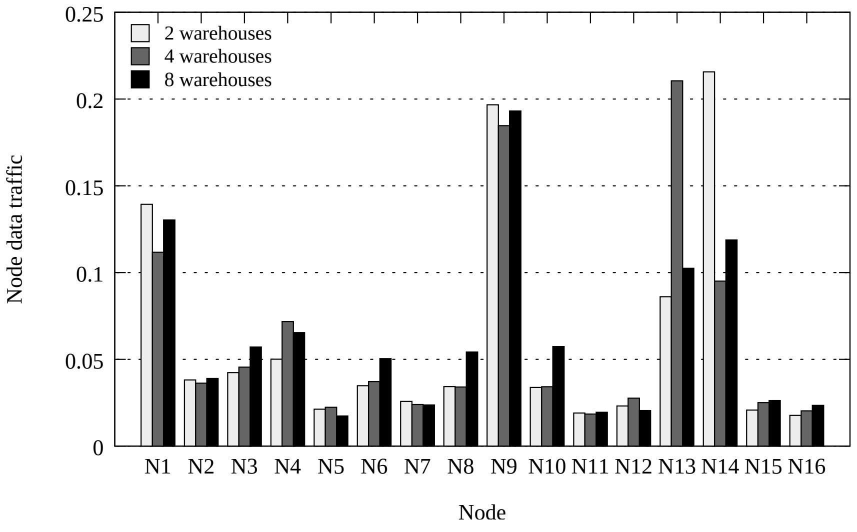
(a)



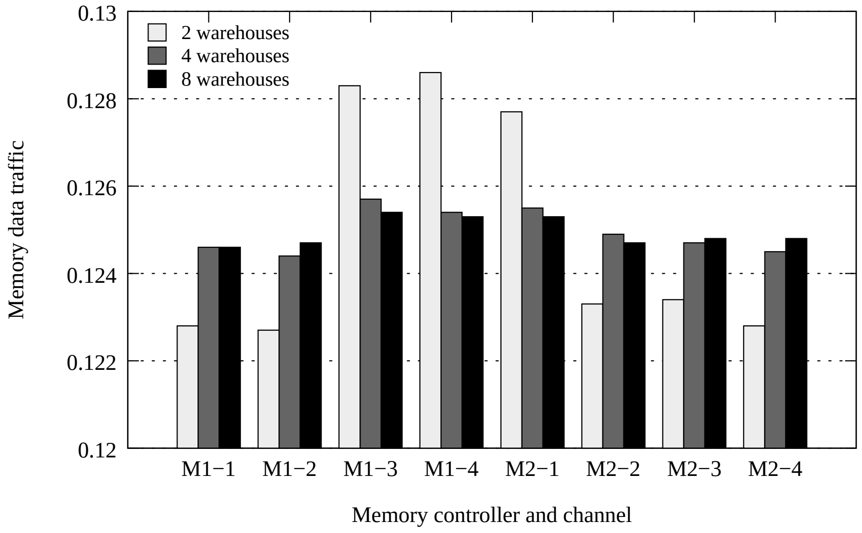
(b)

Figure 6.1: Node traffic distribution (a) and memory controller traffic distribution (b) over FFT.

N8-N10, N13 and N14. Memory traffic is similar to FFT, observable unbalanced distribution can be noticed. An unoptimized scheduling algorithm can cause hotspots and traffic contentions. As a result, average network latency, one of the most important factors of a NoC, is increased and overall performance is degraded. Obviously, without proper schedule, the communication overhead can be an obstacle for future multicore processors.



(a)



(b)

Figure 6.2: Node traffic distribution (a) and memory controller traffic distribution (b) over SPECjbb.

6.1.2 The Scheduler for NoC-based CMP Architecture

As aforementioned, the scheduler fails to maintain optimal node/memory traffic. To overcome the limitation of traditional processor scheduler, a new scheduling algorithm for NoC-based CMP is proposed. We use a CMP model as described below. The algorithm takes into consideration of on-chip topology, scheduling decisions are made based on such information.

Definition 6. 1 *A NoC $N(P(X,Y),M)$ consists of a PE mesh $P(X,Y)$ of width X, length Y; and on-chip memory controllers M (connected to the upper and lower sides of the NoC). Figure 6.3 shows a NoC of $N(P(8,8),16)$.*

Definition 6. 2 *A $N(P(X,Y),M)$ consists of $X{\times}Y$ PEs, therefore this is the maximum number of concurrent threads it can process (without consideration of hyper-threading).*

Definition 6. 3 *A task $T(n)$ with n threads requests the allocation of n cores.*

Definition 6. 4 *n_{Free} is a list of all unallocated nodes in N.*

Definition 6. 5 *$R(T(n))$ is a unallocated region in P with n cores for $T(n)$.*

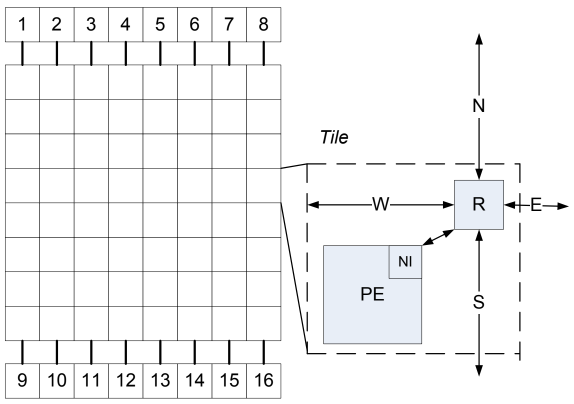


Figure 6.3: A 8×8 mesh-based NoC with 16 memory controllers attached to top and bottom of the design.

Average core access time (ACT) and average memory access time (AMT) are calculated when making scheduling decision. The aim of the algorithm is to minimize average network latency of the system, which is one of the most important factors of a NoC. ACT is defined as the number of nodes a message has to go through from a node to other nodes, $\forall i,j \in P$.

$$ACT = \frac{\sum MD(n_i,n_j)}{n} \tag{6.1}$$

Such that: $\forall i{\neq}j{\in}P$ and $n_i{\neq}n_j$

For a rectangular core allocation with $A{\times}B$ nodes, according to [67], ACT can be calculated in an easier way (Equation 6.2). For example, 4×4 and 2×8 are possible rectangular core allocations for a task with 16 threads.

However, the value of ACT in 4×4 is smaller than in 2×8 (2.5 and 3.125). In consideration of ACT, an allocation shape have a lower ACT number if it is closer to a square. Figure 6.4a and 6.4b show two core allocation schemes for a task with 15 threads. In Figure 6.4b, the number of ACT is lower than in Figure 6.4a (2.4177 and 2.4888 respectively).

$$ACT = \frac{A+B}{3} \times (1 - \frac{1}{A \times B}) \tag{6.2}$$

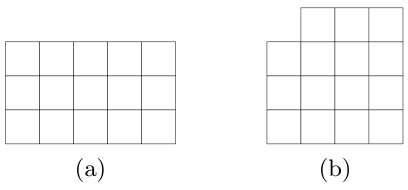


Figure 6.4: Two core allocation schemes for 15 threads.

Taking into account of memory controller placement, e.g. the memory controllers are allocated in top and bottom of the chip. The number of transistors required for a memory controller is quite small compared with billions of total transistors in a chip. The memory controllers are shared by all processors to provide a large physical memory space to each processor. Each of the controller controls a part of the physical memory, and each processor can access any part of the memory [59]. Traditionally, a physical address will be mapped to a memory controller according to its address bits and cache line address. In this case, memory traffic are distributed to all the controllers evenly. However, in our study, we assume that a physical address will be mapped to a memory controller according to its physical location of the on-chip network, i.e. the nearest controller in terms of MD [9]. We define AMT as the minimal number of nodes a message has to go through from a node to the memory controller since more than one controller can co-exist, $\forall i \in P$.

$$AMT = \frac{\sum min(MD(n_i, M))}{n} \tag{6.3}$$

Equation 6.4 shows the access time required for a core-memory communication (not considering the latencies of the memory controller and the memory).

$$L_M = L_{Link\_delay1} + (n_{hop} + 1) \times (L_{Router\_delay} + L_{Link\_delay2}) \tag{6.4}$$

Figure 6.5 shows four typical allocation of a task with 4 threads to a 16-core CMP configuration, all cores are free initially, gray nodes are allocated nodes. The worst case *ACT* is shown in Figure 6.5a, in which 4 threads are distributed in 4 corners of the CMP, thread-thread communication delay is very high. We can calculate the *ACT* is 12 according to Equation 6.1. Allocated cores are in a line in Figure 6.5b, the *ACT* is therfore reduced to 5. In Figure 6.5c and Figure 6.5d, the *ACT* is further reduced to 4.5 and 4 respectively. Obviously Figure 6.5d represent a minimal *ACT* in a 4-thread task.

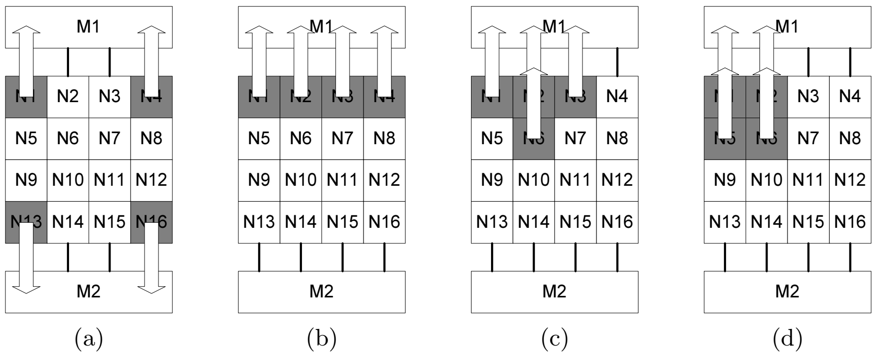


Figure 6.5: Different core/memory allocation schemes, arrows are memory accesses.

In consideration of *AMT* however, although *ACT*s in Figure 6.5a and Figure 6.5b are not optimal, the *AMT*s are only 1 for both configurations because of each allocated core is connected with the memory controller directly. For Figure 6.5c and Figure 6.5d, the *AMT* is 1.25 and 1.5 respectively. Although Figure 6.5d has the best *ACT* value, it might not be optimal in case a task has a lot of memory accesses. Each time a cache miss happen, a request to the memory subsystem is generated to fetch the required data. Lower network latency translates into higher performance.

Figure 6.6 shows six typical allocation of a task with 16 threads to a 64-core CMP configuration, all cores are free initially, gray nodes are allocated nodes. One of the worst case *ACT* configuration is shown in Figure 6.6c, in which 16 threads are distributed in four corners of the CMP, thread-thread communication delay is thus very high. We can calculate the *ACT* is 6.625 according to Equation 6.1. The square allocation shown in Figure 6.6a shows the most promising *ACT*, it is reduced to the minimum of 2.5. As aforementioned in Equation 6.2, for a rectangular core allocation, a quasi-square shape has the lowest *ACT* value. Obviously Figure 6.6a represent a minimal *ACT* in a 16-thread task.

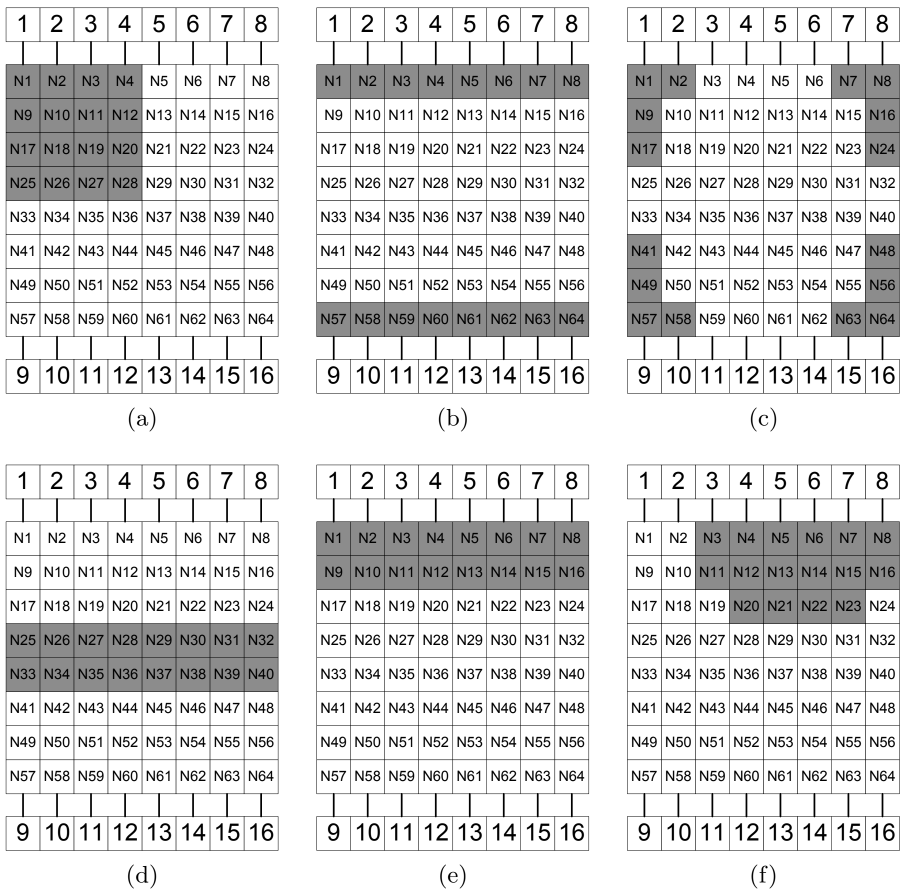


Figure 6.6: Comparison of different core/memory allocation schemes.

In consideration of AMT however, although ACT in Figure 6.6c is the worst, AMT is only 1.75. For Figure 6.6a, despite the fact that it has the best ACT value, the value of AMT is 2.5. This allocation might not be optimal in case a task has a lot of memory accesses. Each time a cache miss happen, a request to the memory subsystem is generated to fetch the required data. Lower network latency translates into higher performance.

The best case AMT is shown in Figure 6.6b, because of each allocated core is connected with the memory controller directly. Figure 6.6d shows the worst value of AMT, which is 4. Two balanced allocation strategies are shown in Figure 6.6e and 6.6f. In these strategies, despite neither ACT nor AMT beats other strategies as a single value, the average numbers of these two factors are better than other four strategies. For instance, allocated cores are in two lines, adjacent to each other in Figure 6.6e, the ACT and

AMT are therefore 3.125 and 1.5, respectively. The average value is lower than in Figure 6.6a (2.3125 and 2.5). Figure 6.6f shows another possibility, with further reduced average number of ACT and AMT. Table 6.1 summarizes these data. We note that if we want to reduce ACT (between 6.625 and 2.5), the value of AMT will increase (between 1 and 4), and vice versa.

Table 6.1: ACTs and AMTs for different allocation strategies

Strategy	**ACT**	**AMT**	**Average value**
Figure 6.6a	2.5000	2.5000	2.5000
Figure 6.6b	6.1250	1.0000	3.5625
Figure 6.6c	6.6250	1.7500	4.1875
Figure 6.6d	3.1250	4.0000	3.5625
Figure 6.6e	3.1250	1.5000	2.3125
Figure 6.6f	2.6406	1.8750	2.2578

In our case (irregularly shaped allocation with ACT and AMT constraints), given a task with n executing threads, we define the problem as deciding the best core allocation for the task by selecting a region containing of n cores. Furthermore, with multiple tasks, the problem is extended to be deciding the best core allocation for all the tasks. Overall network latency is minimized for a task, and the average communication delay for all tasks is minimized. The problem can be described as: Find a region $R(T(n))$ inside $N(P(X,Y),M)$ and a node list N_l of $R(T(n))$, which:

$$min\{\frac{ACT + AMT}{2}\} \tag{6.5}$$

Code 5 The steps of region selection algorithm.

```
1, $\forall n \in n_{Free}$, calculate all $min(MD(n,M))$.
2, $\forall n \in n_{Free}$, start with the first free node $n_i$ and calculate all other $n_j \in n_{Free}$ with $MD(n_i,n_j)$ and sort them in ascending order $MD_1 \leq MD_2 \leq MD_3 \leq \ldots MD_k$.
3, Repeat 2 for the remaining free nodes.
4, Select $R(T(n))$ from 3 which contains $N_l$ that satisfies $T(n)$ with Equation 6.5.
```

The pseudo code of the algorithm is shown in Code 5. Figure 6.6f shows the outcome of the algorithm, with minimal average value of ACT and AMT. This algorithm uses method of exhaustion, and it works always. However, it is very important to design an efficient scheduling algorithm. Our problem belongs to the set of nondeterministic polynomial time (NP) that can be solved in polynomial time with a nondeterministic machine but likely take an exponential execution time to complete in a deterministic machine. That is, to determine if an allocation strategy has the lowest combination of ACT and AMT, it suffices to enumerate the allocation possibilities and then verify if these possibilities produce the lowest value. We consider the problem to be NP-complete. It means that, despite any allocation can be verified in polynomial time, there is no known efficient way to find that allocation. It is as difficult as any other NP-complete problems. The time required to solve this problem increases very quickly as the size of the inputs grows (e.g. the number of free nodes and the number of threads in a task). As a result, it is noteworthy that exhaustive simulation is feasible only for a small size NoC, because of the high computational complexity from the large search space, e.g. an 8×8 mesh with 16 threads has $\binom{64}{16}$ = 488,526,937,079,580 different allocation possibilities! Table 6.2 shows the allocation possibilities for different thread numbers in an 8×8 NoC.

Table 6.2: Allocation possibilities for an 8×8 mesh

Threads of the task	**Allocation possibilities**
1	64
2	2,016
4	635,376
8	4,426,165,368
16	488,526,937,079,580

In real world, however, most applications have fewer threads, and there are fewer available PEs for allocation. Faulty PEs can also be excluded from the search space. Thus there might be a much smaller search space. Figure 6.7 shows a fragmented allocation, in which only 28 cores are available for a new task. In this case, there are only $\binom{28}{16}$ = 30,421,755 allocation possibilities for a 16-thread task.

Another problem is that the trade-off for spending time to find the best combination of ACT and AMT can be undeserving. If the differences between different allocation strategies are quite small, and if the search algorithm takes too much time, a near optimal allocation strategy is preferable.

Heuristical scheduling algorithms are proposed with a clear view of the behaviour of a program beforehand [42]. The longest path to a leaf is selected in the dependence directed acyclic graph [42]. However, considering millions of different applications, it is not practical to draw a graph for all applications. We extend Code 5 with greedy heuristic approximation. As aforementioned, an allocation shape closer to a square have a lower ACT number. The calculation of all combinations is unnecessary. Take Figure 6.7 for example. To schedule a task with 8 threads, we start

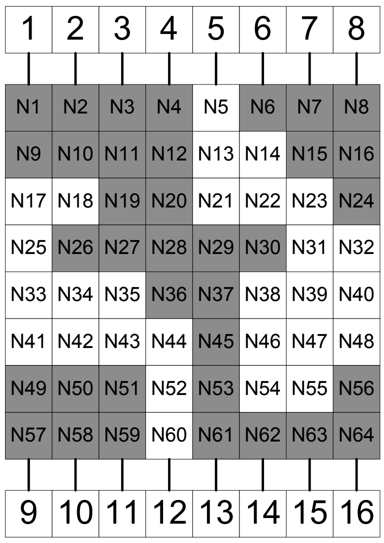

1
2
3
4
5
6
7
8
N1 N2 N3 N4 N5 N6 N7 N8
N9 N10 N11 N12 N13 N14 N15 N16
N17 N18 N19 N20 N21 N22 N23 N24
N25 N26 N27 N28 N29 N30 N31 N32
N33 N34 N35 N36 N37 N38 N39 N40
N41 N42 N43 N44 N45 N46 N47 N48
N49 N50 N51 N52 N53 N54 N55 N56
N57 N58 N59 N60 N61 N62 N63 N64
9
10
11
12
13
14
15
16

Figure 6.7: A fragmented situation with 36 cores occupied (gray), and 28 cores free (white).

from square regions which are closest to the number of nodes required for the task. In this case, we have 4 candidates ($R1(N33-N35, N41-N43), R2(N31, N32, N39, N40, N47, N48), R3(N38-N40, N46-N48)$, $R4(N38, N39, N46, N47, N54, N55)$). To select other two nodes, adjacent nodes of the region are considered. The improved algorithm is shown below.

Code 6 The steps of greedy heuristic approximation.

```
1, $\forall n \in n_{Free}$, calculate the ACT and AMT of all region, which contains nodes $\leq T(n)$.
2, Add adjacent nodes of the region in 1, if the region is smaller than the task.
3, Select $R(T(n))$ from 2 which contains $N_l$ that satisfies $T(n)$ with $min\{\frac{ACT+AMT}{2}\}$.
```

6.1.3 Discussion

Despite our goal is to find the best combination of ACT and AMT using the average value of the two, the weight of ACT and AMT should be considered as well. Different applications have their own profile: memory-intensive or IPC-intensive. Researches shown that scientific applications such as transitive closure, sparse matrix-vector multiplication, histogram and mesh adaption are memory-intensive [37]. It is also shown by Patrick Schmid et al. that video editing, 3D gaming and file compression are memory sensitive applications in daily computing, while other applications concentrate more on thread-thread communication [92].

It is difficult to determine the behavior of an application automatically beforehand, since there are millions of them and the number is still increasing. One feasible way is to add an interface between the application and the OS, the application will tell the OS if it is memory-intensive. Another way is to add a low overhead profiling module inside the OS. Program access patterns are traced dynamically. Memory management functions such as *malloc()*, *free()* and *realloc()* are obtained as histograms for evaluating the weight of AMT, thread management functions such as *pthread_create()*, *pthread_join()* and *pthread_mutex*()* are obtained as histograms for evaluating the weight of ACT. It is noteworthy that these histograms can be only used as rescheduling (thread migration, or in case of a fault PE), i.e. there are no access patterns for the first run of a program. Here, we evaluate the performance differences for several allocation strategies of three 16-thread tasks. These tasks have different IPC and memory access intensities. The detailed performance analysis is shown in later sections.

We also consider a protocol called Processor Status Protocol (PSP). The protocol is implemented between the CMP and the OS, describing the physical topology of the CMP. The physical position of cores and memory controllers and their network connection to other components is determined. The topology information of the CMP is used by the scheduling algorithm (Figure 6.8 and Code 7). The primary work of the scheduler is to evaluate the network performance of the CMP and determine the best core allocation for a task. Better efficiency can be achieved by optimized core allocation and power schemes of the scheduler.

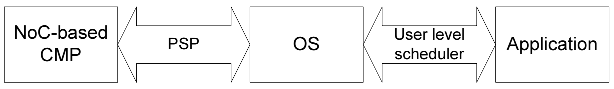


Figure 6.8: The scheduling system between CMP, OS and application.

Code 7 The steps of PSP-OS implementation.

```
1, OS start and query for CMP topology information via PSP.
2, The CMP status is returned and registered to the OS.
3, OS processor scheduler schedules threads with Code 5 according to PSP info.
4, DVFS and power/clock gating commands are sent via PSP by the scheduler.
5, User level application can send their behavior profile to the scheduler so that the scheduler will optimize its scheduling decisions.
```

6.1.4 Case Studies

FFT

The fast Fourier transform (FFT) is an algorithm to compute the continuous and discrete Fourier transform. FFT is widely used in digital signal processing. There are many FFT algorithm implementations, we select a one-dimensional, radix-n, six-step algorithm from [12], which is optimized to minimize IPC. Refer to Section 5.3 for details. It is shown in [117] that the performance is mostly determined by the data latencies between processors. Our workload contains 64K points with 16 threads.

LU

The LU decomposition/factorization is a matrix decomposition which factors a matrix into the product of a lower triangular (L) and an upper triangular matrices (U). It is used in numerical analysis to solve linear equations or to calculate the determinant. The main application fields of LU include: digital signal processing, wireless sensor networks and simulating electric field components. We select a LU decomposition kernel from [117]. This program is optimized to reduce IPC by using blocking. A dense $n \times n$ matrix M is divided into an $N \times N$ array of $B \times B$ blocks ($n = N \times B$). The implementation of blocking method in the program can exploit temporal locality on individual sub-matrix's elements.

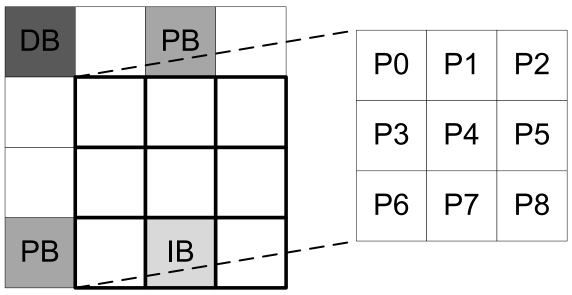


Figure 6.9: LU decomposition algorithm with blocking.

As is shown in Figure 6.9, the diagonal block (DB) is decomposed first. The perimeter blocks (PB) are updated using DB information. The matrix blocks are assigned to processors (P1, P2...) using a 2D scatter decomposition. The interior blocks (IB) are updated using corresponding PB information. It is very important that since the computation of IB involves a dense matrix multiplication of two blocks, to reduce IPC, the computation is performed by the processor that owns the block. Despite the optimization, communication of processors can still be a bottleneck. One situation is that when processors require a DB used by all processors to update their own PBs. In this case, a copy of the DB is sent to all requesting processors by the processor that updates the DB. Another case is that when processors require PBs used by all processors to update their IBs. In this case, a copy of the PB is sent to all requesting processors by the processor that updates the PB [117]. The 16-thread workload used in our experiment has an input matrix of 512×512 with 16×16 element blocks.

H.264

The H.264 is the latest standard of video stream coding, it is optimized for higher coding efficiency than previous standards. We select a data parallel H.264 coding implementation from [18]. In this program, video stream data are distributed to processors. Multiple video stream data can be processed simultaneously in data parallel coding. The program is multithreaded with frame level parallelization, and coarse granular pipeline parallelism is achieved. In terms of IPC and external memory communication, H.264 can be the toughest among three applications. Refer to Section 5.1 for further details. We select "simlarge" as our workload. This is a standard video clip from the PARSEC, taken from an open source movie [18]. This video clip models a high motion chasing scene.

6.1.5 Experimental Evaluation

In this section, a NoC with 64 processing cores is modeled for evaluation (Figure 6.3). The NoC consists of a 8×8 mesh of processing elements (PEs). Each PE consists of a processor core and a shared cache. The 16 memory controllers are connected to the two sides of the mesh network. This represents a typical NoC design, similar as Intel and Tilera. The L2 cache shared by all processors is split into banks. The size of each cache bank node is 1MB; hence the total size of shared L2 cache is 64MB.

We first evaluate performance data of FFT, in terms of cycles per instruction (CPI), misses per thousand instructions (MissPKI) and cache hit latencies. Figure 6.10 shows the normalized values for six allocation strategies for FFT (Figure 6.6). Lower numbers are better. Allocation F (Figure 6.6f) outperforms other strategies in terms of CPI and MissPKI. For example, the MissPKI for allocation F is 20.62% lower than allocation D, and 3.72% lower than allocation A. This is primarily due to the better ACT and AMT numbers in allocation F, compared with other allocations. As aforementioned, the transpose steps in FFT require communication of all processors. It is clearly that allocation B, C and D are not favorable strategies, since the ACT and AMT are high. We also note that the value of CPI does not change significantly among six strategies. In terms of cache hit latency, however, allocation A shows the most promising performance. The reason is that the value of ACT in A is lowest among six strategies. In consideration of three factors, it is obvious that allocation A and F perform better than other four. On average, for allocation F, the three metrics are reduced by 6.94% compared with other allocations. The improvements for MissPKI and cache hit latency of allocation F are reduced up to 25.97% and 13.11%, respectively. Apparently, the FFT algorithm we used here is IPC-intensive, with certain memory accesses.

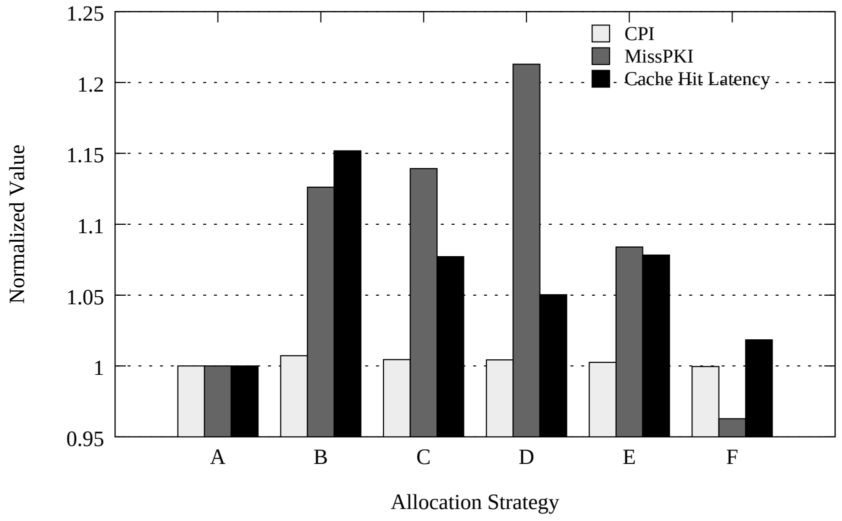


Figure 6.10: Normalized performance metrics with different allocation strategies, for FFT.

Other performance metrics for three applications are evaluated in terms of Average Network Latency (ANL), Average Link Utilization (ALU), Execution Time (ET) and Cache Hit Latencies (CHL). ANL represents the number of average cycles required for the transmission of all network messages. The number of cycle of each message is calculated as, from the injection of the message header into the network at the source node, to the reception of the tail flit at the destination node. ALU is defined as the number of flits transferred between NoC resources per cycle. Under the same configuration and workload, lower metric values are favorable.

The results are illustrated in Figure 6.11, 6.12 and 6.13, in terms of FFT, LU and H.264 respectively. Allocation F (Figure 6.6f, similarly hereinafter) outperforms the other strategies on average, in the three applications. For example, the ANL for allocation F is 10.42% lower than for allocation C, and 3.84% lower than for allocation A when considering FFT application.

This is primarily due to the better ACT and AMT numbers in allocation F, compared with the other allocations. As aforementioned, the transpose steps in FFT require communication of all processors (especially the last stage, see Figure 5.19). In this case, the ACT plays as a major role. It is clear that allocation B, C and D are not favorable strategies in this case, since the two values are high. The ACT is very high in Allocation E, compared with A and F (3.13 in E, 2.50 in A and 2.64 in F). This is the reason why ANL in E is worse than in A and F. The differences of ANL in LU is not as significant as in FFT, e.g. the value of ANL in allocation F is 7.02% lower than in allocation C, and 1.84% lower than in allocation A. The reason is that LU generates less network traffic compared with FFT. The larger difference of ANL in H.264 reflects its higher demand on core-core and core-memory communication, compared with FFT and LU.

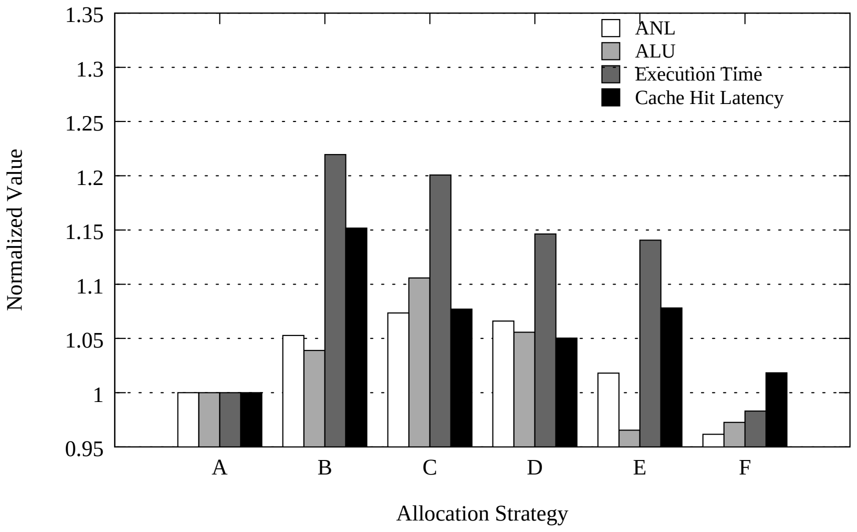


Figure 6.11: Normalized ANL, ALU, ET and CHL with different allocation strategies, for FFT.

The ALUs of FFT for allocation E and F are lower than other strategies as well, e.g. 12.69% and 12.05% lower than in allocation C, respectively. Apparently, ALU is directly related with the average number of ACT and AMT. However, as we observed in the preceding part, the ALU is affected by the traffic intensity of an application as well. In terms of ET, however, the ACT plays as a major role again. Allocations A and F shows the most promising performance, while the other strategies did not perform well. For instance, the ET of allocation F in the three applications has reduced 1.69%, 0.67% and 2.87% compared with allocation A, respectively. The CHL is more related with ACT. Allocations A and F have lower number of CHLs, while allocations with high ACT numbers (B, C, D and E) have much higher values.

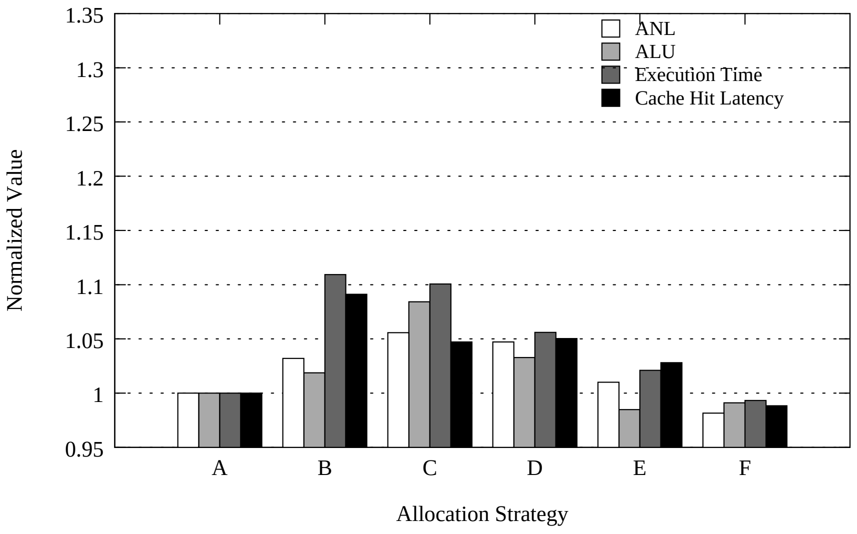


Figure 6.12: Normalized ANL, ALU, ET and CHL with different allocation strategies, for LU.

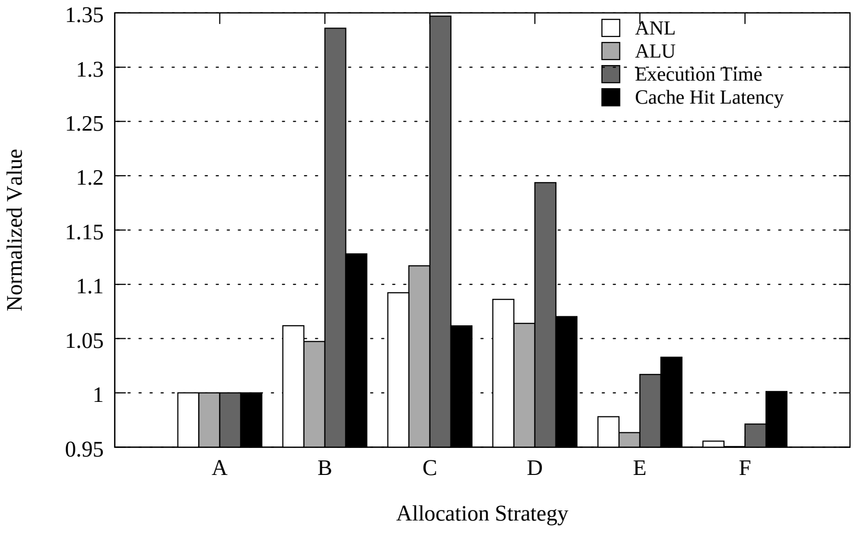


Figure 6.13: Normalized ANL, ALU, ET and CHL with different allocation strategies, for H.264.

We note that ACT is more important than AMT in most cases. This is due to, most multithreaded applications nowadays are still optimized for IPC. An application with abnormal behavior can benefit more from closer memory controllers, e.g. one with a lot of threads sending memory requests constantly to the memory controller, and without communications with each other. We also note that allocations A and F provide better performance than the other allocations in most cases. In consideration of the four metrics, on average, for allocation F, the performance is improved by 8.23%, 4.81% and 10.21% in FFT, LU and H.264, respectively, compared with the other allocations.

6.2 A Greedy Heuristic Approximation Scheduling Algorithm

The 3D integration has the potential to increase device density, providing shorter wire lengths and faster on-chip communication compared with the 2D integration. TSV is proposed as a viable solution in building 3D chips. The manufacturing process of the TSV is complex and expensive [109], therefore finding an optimal number and placement of TSVs is critical. It is presented that the balance between performance and manufacturing cost is essential in designing a 3D chip [128].

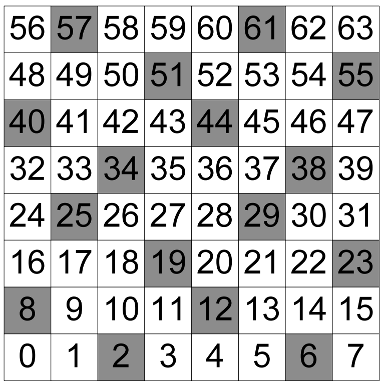
With limited resources between layers, it is obvious that better or even optimal efficiency can be achieved through appropriate scheduling of multi-threaded tasks in large scale 3D multicore processors. Task scheduling for 2D NoC platforms is studied in [22] and [46]. The impact of limited resources between layers is not considered in these papers. Here, we propose and discuss a novel greedy heuristic approximation scheduler for TSV constrained 3D multicore processors which aims to reduce the average network latency between caches and processing cores. With the decrease of the latencies, lower power consumption and higher performance can be achieved. To confirm our theory, we model and analyze a 64-core, 2-layer NoC with 8×8 meshes, present the performance of an application with different allocation strategies using a full system simulator.

6.2.1 3D NoC with Through Silicon Via Constraints

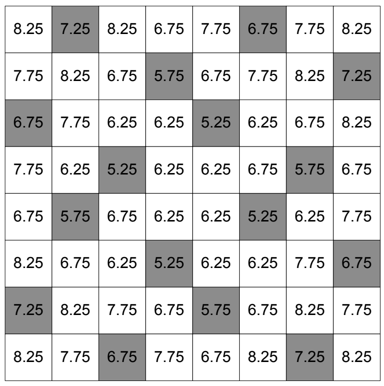
A modern multicore processor is composed of several parts, e.g. processor core, shared last level cache, I/O and memory controller. Nearly half of the die area is devoted to cores and the other half is devoted to shared caches and other circuits. A natural way of applying 3D integration is to partition all the processors to one layer and other components to the other layer. There is a significant concern for thermal hot-spots brought by packing layers vertically. It is expectable that since the processors consume overwhelming majority of power in a chip, stacking multiple processor layers would be unwise for heat dissipation. According to [128], heat dissipation is a major problem by stacking multiple processor layers even if processors are interlaced vertically. Therefore, in consideration of heat dissipation of current CMP, the processor layer should be on top of the chip.

In this section, based on the above analysis, we use a 3D multicore processor model of two layers (more memory layers can be attached). The top layer is an 8×8 mesh with 64 Sun SPARC cores under 32nm. The L2 cache is 64MB split into 64 banks, with 64-bit line size and 4-way associativity. Based on 32nm, each cache bank, including data and tag, occupies 3.2mm^2. The cache layer has an 8×8 mesh of cache banks. As aforementioned, routers are quite small compared with processors and caches. The total area of the processor is supposed to be below 300mm^2.

TSV is the most promising solution for building 3D chips. There are several types of TSVs, e.g. data signal transmission, control signal transmission, power distribution and thermal dissipation. Here, a pillar is defined as a bunch of TSVs, including TSVs for data, control and power distribution. It is obvious that the maximum performance can be achieved by full layer-layer connection, e.g. all routers are connected with up/down routers by pillars. However, as the number of tiles grow, it might not be practical to assume that each tile will be connected with corresponding TSVs because of the manufacturing cost and chip area. In [128], the placement of pillars is studied, an optimal placement of TSVs for an 8×8 mesh with 16 pillars is presented to minimize traffic contention between layers (Figure 6.14a). The overall performance and total number of TSVs are 92% and 20% compared with full layer-layer connection respectively, achieving a good balance between performance and manufacturing cost.



(a) Numbers denote the sequence of nodes.



(b) Numbere denotes average hop counts to all cache nodes

Figure 6.14: Gray nodes are attached to a pillar.

Assuming XYZ deterministic routing, Equation 6.6 shows the access time (latency) required for a core-cache communication. The latency involves in-tile links (Between NI and PE, L_{Link_delay1}), router (L_{Router_delay}), tile-tile links (L_{Link_delay2}), the number of hops required to reach the destination (n_{hop}) and the delay caused by TSV (L_{TSV_delay}). Since the delays of link, router and TSV are fixed, hop count is the most important metric in determining latency. Figure 6.14b shows the average hop counts required for a core to access the shared cache nodes (AHPC). Obviously, without proper schedule, the communication overhead can be an obstacle. For example, nodes at corners of the NoC have much higher AHPC than nodes in the center. However, nodes directly connected with a pillar usually have lower AHPC, sometimes even lower than inner nodes, e.g. the AHPC for the node 38 is 5.75, lower than 6.75 of the node 37. Scheduling a task to the node 38 is therefore preferable than 37, since the average delay to the shared caches is lower.

$$L_{CoreCache} = 2 \times L_{Link\_delay1} + (n_{hop} + 1) \times L_{Router\_delay} + n_{hop} \times L_{Link\_delay2} + L_{TSV\_delay} \tag{6.6}$$

6.2.2 The Scheduling Algorithm

Our proposed scheduling algorithm takes into consideration of on-chip topology and TSV placement, scheduling decisions are made based on these information. The aim of the algorithm is to minimize average network latency of the system, which is an important factor of system performance. We use a 3D NoC model as described below. Other definitions can be found in the previous section.

Definition 6. 6 *A 3D NoC $N(P(X,Y), C(X,Y))$ consists of a layer of PE mesh $P(X,Y)$ (width X, length Y); and a layer of cache mesh $C(X,Y)$. Layers are connected by TSVs, only a quarter of nodes are connected (Figure 6.14a).*

Definition 6. 7 *n_{Free} is a sorted list of all unallocated nodes in P, such that: $AHPC_{nFree1} \leq AHPC_{nFree2} \leq AHPC_{nFree3} \leq \ldots AHPC_{nFreek}$.*

To schedule a task efficiently, several metrics have to be considered, e.g. MD, AHPC and so on. Scheduling a task with only 1 thread is relatively easy. In this case, nodes 19, 29, 34 and 44 are considered in the first place, if they are not utilized by other applications. The reason is that these four nodes have the lowest AHPC (5.25). However, as the requested number of threads grows, other metrics have to be included. For example, a 2-thread task can be scheduled to nodes 19 and 29. In this case, the Inter Process Communication (IPC) between threads will suffer higher delay, since the messages have to go through nodes 20 and 21 according to XY routing. Another problem is fragmentation. Non-contiguous allocation of cores in a dynamic system can cause degradation of overall system performance. The

2-thread task can be scheduled to nodes 19 and 20 as well. Despite the fact that the AHPC is increased by 1 for node 20 compared with node 29, the adjacent allocation will alleviate IPC bottleneck, and reduce fragmentation.

ACT is used to calculate the overhead of a core-core communication. According to the equation, the ACT is 3 and 1 for nodes 19/29 and 19/20, respectively. The delay for a core-core communication is shown in Equation 6.7. Obviously, allocation 19/29 will incur much higher router delay and delay of tile-tile links, comparing with allocation 19/20. It is noteworthy that a core-core communication is a intra-layer communication, while a core-cache communication is a inter-layer communication.

$$L_{CoreCore} = 2\times L_{Link\_delay1} + (n_{hop}+1)\times L_{Router\_delay} + n_{hop}\times L_{Link\_delay2} \tag{6.7}$$

Code 8 The Greedy Heuristic Approximation Scheduling Algorithm

Input: A mesh based NoC N with TSV constrains, a task with n threads
Output: An allocated region R, containing n processors

```
1  Pop the first node as an initial node $u_0$ from $n_{Free}$, and push $u_0$ to $R$
2  $n_{MD}$ := $n_{Free}$ sorted as MD($u_0$,$n_{Free}$)
3  while $n_{MD}$ is not empty do
4      Pick a node $u_n(x_i, y_i)$ from $n_{MD}$ with smaller AHPC
       if several nodes with same AHPC then
5          pick a node $u_n(x_i, y_i)$ with smaller ACT in the result region
6      end
7      if several nodes with same ACT then
8          pick a node $u_n(x_i, y_i)$ which $x_i \to \frac{X}{2}$ and $y_i \to \frac{Y}{2}$
9      end
10     Pop $u_n(x_i, y_i)$ from $n_{MD}$, and push $u_n$ to $R$
11 end
```

For a rectangular core allocation with $A\times B$ nodes, according to [67], ACT can be calculated in an easier way. A scheduling algorithm should have a low computation complexity and should deliver an optimal or near-optimal results. This is due to the scheduling has to be solved online, and the time for solving the scheduling is a part of the overall system response time. It is clear that we should not try to solve the scheduling problem optimally, in case the computation complexity is too high. Given a task with n executing threads, we define the problem as determining the near-optimal core allocation for the task by selecting a region containing of n cores. The pseudo code of the algorithm is shown in Code 8.

Line 1 sets the starting node of the algorithm, which is the one with the lowest AHPC. A list n_{MD} contains nodes sorted based on MD from the starting node. The adjacent nodes are always considered first, in terms of AHPC. ACT will be calculated, in case several nodes are with the same AHPC. If ACTs for the allocation strategies are still the same, a node closer to the center of the network will be selected (considering the statistical variance of the coordinates of two nodes, Equation 6.8). This is due to the fact that nodes in the center usually have lower AHPC than nodes in the border, following steps may have better results from this heuristic.

$$Var(n) = \frac{1}{2} \times [(x_i - \frac{X}{2})^2 + (x_j - \frac{Y}{2})^2] \tag{6.8}$$

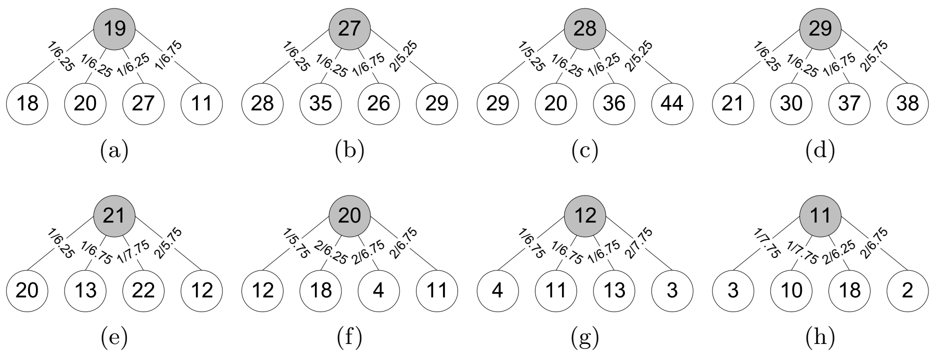


Figure 6.15: The node selection steps for the algorithm.

We analyze an example of the algorithm. Figures 6.15a to 6.15h shows the steps for node selection, starting from node 19. The number between two nodes n_i and n_j means MD(n_i,n_j) and AHPC(n_j). Note that we only show 4 child nodes in these figures. The actual list n_{MD} and n_{Free} may contain more nodes. As illustrated in Figure 6.15a, node 19 has 4 adjacent nodes and 3 of them are with the same AHPC and ACT. However, in terms of distance to the center, node 27 is selected ($Var(27) < Var(20) < Var(18)$). The selection of the next node follows the similar rule: same AHPC, same ACT, same variance. In this case we choose node 28, having a smaller node number than node 35. Figure 6.15c demonstrates that node 29 is selected due to its lowest MD and AHPC. The next step involves different ACTs: both node 21 and node 30 have the lowest AHPC, however the ACTs for the two nodes are different (2 for node 30, and 1.8 for node 21). Node 20 and 12 are selected as the sixth and seventh node, respectively, due to their lowest AHPC. The next node (11) is picked out, on account of its lower ACT than the others. It is noteworthy that the aforementioned greedy heuristic approximation algorithm generates near-optimal scheduling solution in most cases. However, in our algorithm we put adjacent nodes as the first priority, the AHPC and ACT are considered next. This strategy may generate non-optimal scheduling for certain applications.

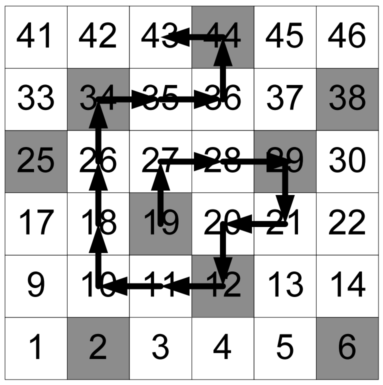


Figure 6.16: The execution of our algorithm, starting from node 19 and selected 16 nodes.

Take a 4-thread application for example. As shown in Figure 6.16, the algorithm will choose nodes 19, 27, 28 and 29 for allocation. An IPC-intensive application may suffer from the long distance communication of node 19 and 29. In this case, node 20 is a better choice than 29 since the ACT is lower. Despite our goal is to find a near-optimal scheduling using MD, AHPC and ACT, the weight of these metrics should be considered as well. Different applications have their own profile: some have higher demand of caches, some have higher volume of IPC. One solution is to add an interface between the application and the OS, the application will tell the OS its behavior. Another solution is to add a low overhead profiling module inside the OS. Program access patterns are traced dynamically, and possibly migrated for better allocations.

6.2.3 Experimental Evaluation

We use a 64-core multiprocessor for our experiments. The 3D CMP has two layers; the first layer contains PEs (each running at 2GHz with a private L1 cache, split I+D, each 16KB, 4-way associative, 64-bit line, 3-cycle), the second layer consists of shared L2 caches (unified 64 banks, each 1MB, 64-bit line, 6-cycle). The MOESI cache coherence protocol is used. We select FFT [12] and Radix [19] as applications. In FFT, the communication between processors only take place at the last stage of the execution. However the network traffic and cache miss rate are very high. The Radix sort algorithm assigns each processor a part of the sorting keys. For every iteration in the algorithm, a permutation for the keys is required to create a new array for the next iteration. This will incur all-to-all communication among processes. Hence Radix represents an application with high IPC.

Performance is evaluated in terms of Average Network Latency (ANL), Execution Time (ET) and Cache Hit Latencies (CHL). We analyze two core allocations for a 2-thread task: *T2-1* is from our algorithm, which contains nodes 19 and 27. It has lowest ACT values, however the AHPC is not optimal. *T2-2* is an alternative allocation, which contains nodes 19 and 29. In this case, the AHPC is minimized. The *T4-1*, *T4-2* and *T4-3* are three core allocations for a 4-thread task: *T4-2* contains nodes 19, 20, 27 and 28, represents lowest ACT; *T4-3* contains nodes 19, 29, 34 and 44, represents lowest AHPC. Our algorithm selects *T4-1*, it has neither lowest ACT nor AHPC numbers. However we believe it could be a good balance for the two metrics.

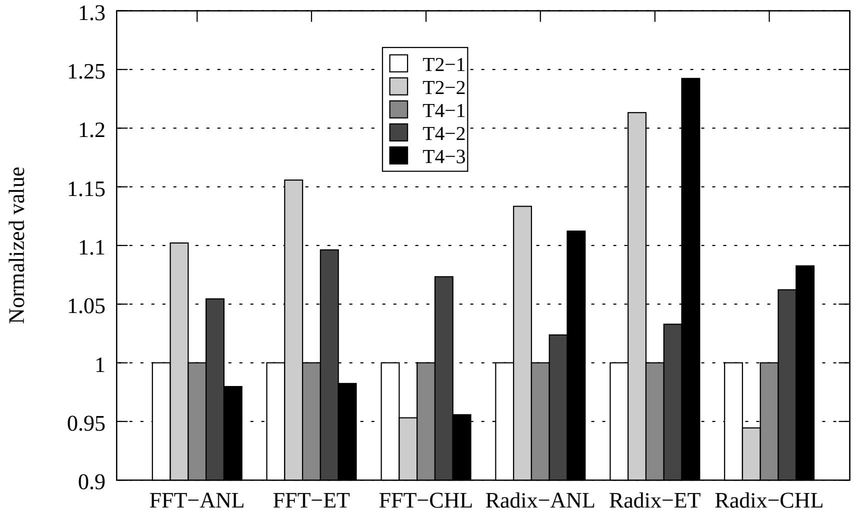


Figure 6.17: Performance for FFT and Radix.

The results are illustrated in Figure 6.17. The core allocation of our scheduling algorithm for 2 threads outperforms the other in terms of ANL. The improvement is more notable in 2-thread FFT and Radix, with 9.26% and 11.77% reduced latency, respectively, compared with the *T2-2* allocation. This is primarily due to the reduced communication overhead between two PEs. We note that the reduced AHPC in *T2-2* failed to compensate the increasing ACT, in terms of ANL. The CHL in *T2-2* directly reflects the reduced AHPC. However, the average runtime of two applications show that our algorithm spends 15.58% shorter time than *T2-2*. Considering a 4-thread task, we note that both ACT and AHPC play important roles in overall performance. For example, despite the fact that *T4-2* has lowest ACT, the ANL for two applications is 3.76% higher than in our algorithm. This leads to a higher ET as well. Allocation *T4-3* performs better than our scheduling in the 4-thread FFT. This is because of, in FFT, the communication between threads only happens at the last stage of the execution. In this case, we observe that the trade-off for ACT is worthy. However, applications that heavily rely on IPC, e.g. Radix, will suffer from the *T4-3*. The ET of *T4-3* is 24.24% longer than in *T4-1*.

Chapter 7

Conclusion

An efficient computer system should include both high efficiency hardware and software that is specially optimized for the designated hardware. The main contribution of this dissertation lies on the hardware/software co-designs of multicore architectures.

As a part of hardware designs of this dissertation, the different two-dimensional and three-dimensional Network-on-Chip designs with integrate and split core/cache architectures were explored. In the proposed system, the last level caches were separated from the processing elements and implemented on another layer of the processor using static non uniform cache architecture. Wire lengths were reduced significantly in the proposed system. Through the experimental results, it is found that the proposed split core/cache architecture outperformed the traditional design in terms of cache hit latency, average link utilization and average network latency.

Observing that current on-chip systems suffer from the critical memory bandwidth problems in the communication between on-chip components and off-chip memory, a novel three-dimensional Network-on-Chip architecture with several layers of on-chip dynamic random access memories was proposed as a solution. The architecture contained a processor layer, a cache layer and several memory layers. Results of the experiments show that the average link utilization and execution time of application were reduced compared with the design of off-chip memory.

Intelligent placement of resources was introduced in this dissertation, so that the optimal position of the resource was determined in the design phase and the performance degradation by reduced amount of resources was alleviated. Several network sizes and amount of resources were explored. A generic method, divide and conquer, has been proposed for larger networks.

Through silicon vias were used as a case study for resource placement. In a three-dimensional chip, it is obvious that the maximum performance can be achieved by full layer-layer connection. However, as more than half of the manufacturing costs are dedicated to these connections, reducing the total number of layer-layer connections is an important design issue. The performance of the chip could be affected, since multiple nodes have to share a connection. To minimize traffic contention between layers, the placement of these connections should be considered carefully. The optimal placement of memory controllers were investigated in this dissertation as well. The memory bandwidth problem could be a bottleneck for designing future multicore processors. By integrating more on-chip memory controllers, this problem can be alleviated. However, the number of on-chip memory controllers will be limited due to the restrictions of the pin count. Several metrics, including average hop count, memory controllers per row/column and adjacent memory controllers have been analyzed for optimal placement of memory controllers. As the third case study, we explored the optimal placements of cores and caches. It is found that the placement of cores and caches in mesh networks have a significant impact to the system performance. Several core/cache configurations were investigated. Experimental results shown that the optimal placement of hardware resources provided a balance between performance and manufacturing cost.

As a part of software designs of this dissertation, three applications, including H.264, N-Body and FFT, were used to study the implementation of software to on-chip networks. Different NoC design alternatives for data parallel H.264 coding have been proposed and evaluated. Our study shows that the inter-thread communication and shared data accesses are the system performance bottlenecks. Two three-dimensional design approaches with processors on the top layer and distributed in two layers have been evaluated. It is shown that, although the performance of distributed design was higher, it suffered from problem of heat dissipation. The implementation of two hierarchical N-Body methods (Barnes-Hut and Fast Multipole) in a NoC platform was studied as the second application. Both scalability and network traffic for the two methods were analyzed. The time distribution of the two methods were explored. We investigated the advantages and disadvantages of the two algorithms. The network requests rates of 64 processing cores were illustrated for both methods. Our experiments have shown that the Barnes-Hut method generates more hot-spot traffic than Fast Multipole. However, it scales better, and has lower overall pressure to the on-chip network and caches, compared with Fast Multipole. An optimized Network-on-Chip architecture for data parallel FFT was proposed in this dissertation. A one-dimensional, radix-n, six-step FFT algorithm was selected. We analysed low-level traffic pattern for FFT. Several hotspots were found. An optimization method, namely long links between hotspot nodes, was introduced. Results show that the reduced latencies have a strong impact on system performance. The execution time of our optimized design was reduced significantly compared with the original design.

The scheduling of processes and threads for future multicore platforms was explored. We investigated two scheduling algorithms for two-dimensional and three-dimensional network-on-chips. A minimal average access time scheduler was proposed to reduce overall on-chip communication latencies for two-dimensional network-on-chips. For resource constrained three-dimensional chips, we presented a greedy heuristic approximation scheduling algorithm. The constraints of through silicon vias were discussed. Experimental results shown the execution time of applications reduced significantly with proper scheduling. Our research have demonstrated that operating system scheduling is crucial for future multicore architectures.

7.1 Future Work

The research of hardware/software co-design in two-dimensional and three-dimensional Network-on-Chips has just started, and yet there are still many open problems to explore.

For example, new technologies of volatile and non-volatile memories are emerging, and can be a candidate in designing future multicore chips. These include magneto-resistive random access memory (MRAM), phase-change random access memory (PRAM), ferroelectric random access memory (FRAM) and resistive random access memory (RRAM). They can provide different design possibilities for multicore architectures.

The resource placement problem can be refined as well, by using other mathematical methods. The integer linear programming is a candidate for improving the effectiveness of resource placement. The combination of both divide and conquer and integer linear programming could open a new direction of resource placement.

The software adaptability for future multicore architectures still require more work. For example, the implementation and optimization for applications to the Network-on-Chip platform, and the modifications to operating system scheduler to reach an optimal or sub-optimal performance in such platforms.

Appendix A

Example Simulation Output and Post-processing

The following results (Code 9) are generated by the GEMS/Simics full system simulator. These results include network configuration, execution time, cache misses, executed instructions, L2 cache profiles, average link utilization, average virtual channel loads, average network latency and so on.

By adding support for system event trace, messages sent from one node to another node (unicast) or other nodes (multicast/broadcast) are gathered for analysis. A typical request message is shown in Code 10, the memory address of the message is "0x2c486c0", the type of the message is "Get Instruction", the access mode of the message is "SupervisorMode", the requestor of the message is the 14th node of the L1 cache, the destination of the message is unicast to node 9, the message is sent on cycle 5, it has no dirty bit nor prefetch option. The types of request messages are shown in Code 11. The types of a respond messages are shown in Code 12. Notice that different cache coherence protocols have different message types.

A segment of the event trace is demonstrated in Code 13. The four fields in a line is segmented by a semicolon. The first field represents the cycle number, the second represents the source node (e.g. 0-63 in a 64 node network), the third represents the destination node, and the last field represents the message type (e.g. R for Request and S for Response, number denotes the sequence in the enumeration structure). Notice that the detailed information of a message can be gathered as well, however in this case each line requires additional space. In this dissertation, the size of traces ranges from tens of megabytes to hundreds of gigabytes. The communication between cache nodes and memory controllers are presented in Code 14. The extra field at the end of a line represents the memory address.

To process these trace data, which can contain hundreds of gigabytes of information, several scripts are composed. For example, Code 15 is used for generating source data for plotting figures from the trace file, Code 16 and 17 are used for generating the inter-layer message statistics, and Code 18 is used for counting inter-layer messages.

Code 9 Sample results.

```
......
[Network Interface 0] - [inLink 84] - [outLink 0]
[Network Interface 1] - [inLink 89] - [outLink 1]
[Network Interface 2] - [inLink 94] - [outLink 2]
[Network Interface 3] - [inLink 104] - [outLink 3]
[Network Interface 4] - [inLink 109] - [outLink 4]
[Network Interface 5] - [inLink 114] - [outLink 5]
......
Elapsed_time_in_seconds: 58865
......
total_misses: 63716 [ 15959 550 183 251 117 121 118 114 117 7017
117 1560 8508 123 117 115 119 119 115 181 26499 122 115 112 109
306 114 114 191 123 115 175 ]
......
instruction_executed:  6148165072  [  148532668  199320085
199908109  199840525  200006216  199998180   199992377
199982544  199947887  189757216  199839161   197883902
188127441  199964058  199962287  199961567   199994756
200000320  200003247  199255653  28217294    199969973
199961410  199965802  199966170  199747725   200004509
200000001  199889970  199979479  199955817   198228723 ]
......
L2_cache_misses_per_transaction:  63716
L2_cache_misses_per_instruction:  1.03634e-05
L2_cache_instructions_per_misses: 96493.3
L2_cache_request_type_GETS:       28.9645%
L2_cache_request_type_GET_INSTR:  10.8011%
L2_cache_request_type_GETX:       23.0005%
L2_cache_request_type_UPGRADE:    37.234%
......
Average Link Utilization :: 0.00107376 flits/cycle
Average VC Load [0] = 0.0292597 flits/cycle
Average VC Load [1] = 0.0292669 flits/cycle
Average VC Load [2] = 0.0292714 flits/cycle
Average VC Load [3] = 0.029272 flits/cycle
......
Average network latency = 29.4251
......
```

Code 10 Request message.

```
RequestMsg: Address=[0x2c486c0, line
0x2c486c0] Type=GET_INSTR
AccessMode=SupervisorMode Requestor=L1Cache-
14 Destination=[NetDest (3) 0 0 0 0 0 0 0 0 0 0 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0  - 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0  - 0 0 0 0  - ] MessageSize=Control
DataBlk=] Dirty=0 Prefetch=No Time=5
```

Code 11 Request message types.

```
enumeration(CoherenceRequestType, desc="...") {
  GETX,      desc="Get eXclusive";
  UPGRADE,   desc="UPGRADE to exclusive";
  GETS,      desc="Get Shared";
  GET_INSTR, desc="Get Instruction";
  INV,       desc="INValidate";
  PUTX,      desc="replacement message";
}
```

Code 12 Request message types.

```
enumeration(CoherenceResponseType, desc="...") {
  MEMORY_ACK,          desc="Ack from memory controller";
  DATA,                desc="Data";
  DATA_EXCLUSIVE,      desc="Data";
  MEMORY_DATA,         desc="Data";
  ACK,                 desc="Generic invalidate ack";
  WB_ACK,              desc="writeback ack";
  UNBLOCK,             desc="unblock";
  EXCLUSIVE_UNBLOCK,   desc="exclusive unblock";
}
```

Code 13 Example of Node-Node message queue.

```
29655002,4,30,S4
29655013,4,31,R2
29655013,4,30,R1
29655041,30,58,S4
29655051,30,59,R1
29655057,31,4,R7
29655097,4,31,S5
29655256,59,30,S2
29655305,30,4,S2
29655340,4,30,S4
29655344,4,26,R2
29655344,4,4,R1
29655356,4,0,R1
29655379,30,59,S4
29655384,26,4,R7
29655419,4,26,S5
29655544,0,4,S2
```

Code 14 Example of Cache-Memory message queue.

```
62,L2Cache-11,Directory-14,M0,[0x2c48780]
63,L2Cache-11,Directory-3,M0,[0x2c39cc0]
66,L2Cache-11,Directory-12,M0,[0x2c48700]
70,L2Cache-11,Directory-10,M0,[0x2c49a80]
71,L2Cache-11,Directory-1,M0,[0x2c4a040]
71,L2Cache-11,Directory-13,M0,[0x2c48740]
73,L2Cache-11,Directory-13,M0,[0x2c49b40]
73,L2Cache-11,Directory-11,M0,[0x2c486c0]
78,L2Cache-11,Directory-2,M0,[0x2c39c80]
322,L2Cache-3,Directory-3,M0,[0x10d940c0]
340,L2Cache-3,Directory-0,M0,[0x10dae000]
345,L2Cache-11,Directory-9,M0,[0x2c49a40]
374,L2Cache-11,Directory-14,M0,[0x2c49b80]
412,L2Cache-3,Directory-0,M0,[0x10dbe000]
431,L2Cache-11,Directory-11,M0,[0x2c49ac0]
445,L2Cache-12,Directory-2,M0,[0x3000080]
492,L2Cache-3,Directory-0,M0,[0x10dca000]
```

Code 15 Perl source code of generating plot data from traffic trace.

```
#!/usr/bin/perl
$total_nodes = 64;
$array_size = $total_nodes * 100 * 3;
$array_magic = $total_nodes * 3;

$trace_file = $ARGV[0];
$output_file = $ARGV[1];
open(IN, "< $trace_file") || die "Cannot open $trace_file";
open(OUT, "> $output_file") || die "Cannot write $output_file";

my $file_head = qx(head $trace_file -n 1);
$file_head =~ s/(\d+),.*/\1/;
my $file_tail = qx(tail $trace_file -n 1);
$file_tail =~ s/(\d+),.*/\1/;
$total_cycles = int($file_tail)-int($file_head);
$cycle_frac = int($total_cycles / 100)+1;

for ($i=0; $i<100; $i++) {
   for ($j=0; $j<$total_nodes; $j++) {
       push (@xyz, ($i, $j, 0));
   }
}

while(<IN>){
   /(\d+),(\d+),(\d+),/;
   $x = int(($1-int($file_head)) / $cycle_frac);
   $y = $2;
   $location = $x*$array_magic+$y*3+2;
   splice(@xyz,$location,1,$xyz[$location]+1);
}

for ($i=0; $i<100; $i++) {
   for ($j=0; $j<$total_nodes; $j++) {
       print OUT "$xyz[$i*$array_magic+$j*3] ";
       print OUT "$xyz[$i*$array_magic+$j*3+1] ";
       print OUT "$xyz[$i*$array_magic+$j*3+2]\n";
   }
}

close(OUT);
close(IN);
```

Code 16 Bash source code of generating point-to-point message statistics in a 4×4 mesh.

```
# for i in ‘seq 0 15‘;
  do for j in ‘seq 0 15‘;
  do echo -n $i,$j" ";
  grep "^.*,$i,$j," trace|wc -l;
  done; done > p2pmsg
```

Code 17 Bash source code of generating inter-layer message statistics in an 8×8 mesh.

```
# for i in ‘seq 0 63‘;
  do for j in ‘seq 64 127‘;
  do grep "^$i,$j " p2pmsg;
  done; done > p2pmsg.l0-l1
```

Code 18 Bash/sed source code of counting inter-layer messages in an 8×8 mesh.

```
# sed -e "s/\(.*,.*\) \(.*\)/\2/" p2pmsg.l0-l1 | tr ’\n’ ’+’ |
  sed -e "s/\(.*\)\(+\)/\1\n/" | bc
```

Bibliography

[1] S. J. Aarseth, M. Henon, and R. Wielen. A comparison of numerical methods for the study of star cluster dynamics. *Astronomy and Astrophysics*, 37:183–187, 1974.

[2] Dennis Abts, Natalie D. Enright Jerger, John Kim, Dan Gibson, and Mikko H. Lipasti. Achieving predictable performance through better memory controller placement in many-core cmps. In *Proceedings of the 36th annual international symposium on Computer architecture*, pages 451–461, June 2009.

[3] Niket Agarwal. A detailed interconnection network model inside a full-system simulation framework, December 2011. http://www.princeton.edu/ niketa/garnet.

[4] R. Airoldi, F. Garzia, and J. Nurmi. Fft algorithms evaluation on a homogeneous multi-processor system-on-chip. In *Parallel Processing Workshops (ICPPW), 2010 39th International Conference on*, pages 58 –64, sept. 2010.

[5] AMD. Family 10h amd phenom processor product data sheet, November 2008. http://www.amd.com/us-en/assets/content_type/white_papers_and_tech_docs/44109.pdf.

[6] AMD. The amd opteron 6000 series platform, May 2010. http://www.amd.com/us/products/server/processors/6000-series-platform/pages/6000-series-platform.aspx.

[7] Semiconductor Industry Association. The international technology roadmap for semiconductors (itrs), 2011. http://www.itrs.net/Links/2011ITRS/Home2011.htm.

[8] D. Astely, E. Dahlman, A. Furuskar, Y. Jading, M. Lindstrom, and S. Parkvall. Lte: the evolution of mobile broadband. *Communications Magazine, IEEE*, 47(4):44 –51, april 2009.

[9] Manu Awasthi, David W. Nellans, Kshitij Sudan, Rajeev Balasubramonian, and Al Davis. Handling the problems and opportunities posed by multiple on-chip memory controllers. In *Proceedings of the 19th international conference on Parallel architectures and compilation techniques*, pages 319–330, New York, NY, USA, 2010. ACM.

[10] Ahmad R. Bahai and Burton R. Saltzberg. *Multi-Carrier Digital Communications: Theory and Applications of Ofdm.* Plenum Publishing Co., 1999.

[11] Jun Ho Bahn, Jungsook Yang, and Nader Bagherzadeh. Parallel fft algorithms on network-on-chips. In *Proceedings of the Fifth International Conference on Information Technology: New Generations*, pages 1087–1093, Washington, DC, USA, 2008. IEEE Computer Society.

[12] David H. Bailey. Ffts in external or hierarchical memory. *The Journal of Supercomputing*, 4:23–35, 1990. 10.1007/BF00162341.

[13] J. Barnes and P. Hut. A hierarchical o(n log n) force-calculation algorithm. *Nature*, 1988.

[14] Bradford M. Beckmann and David A. Wood. Managing wire delay in large chip-multiprocessor caches. In *Proceedings of the 37th annual IEEE/ACM International Symposium on Microarchitecture*, pages 319–330, December 2004.

[15] E. Beigne, F. Clermidy, H. Lhermet, S. Miermont, Y. Thonnart, Xuan-Tu Tran, A. Valentian, D. Varreau, P. Vivet, X. Popon, and H. Lebreton. An asynchronous power aware and adaptive noc based circuit. *Solid-State Circuits, IEEE Journal of*, 44(4):1167 –1177, april 2009.

[16] Luca Benini and Giovanni De Micheli. Networks on chips: A new soc paradigm. *IEEE Computer*, 35(1):70–78, January 2002.

[17] Christian Bienia, Sanjeev Kumar, and Kai Li. Parsec vs. splash-2: A quantitative comparison of two multithreaded benchmark suites on chip-multiprocessors. In *IEEE International Symposium on Workload Characterization*, pages 47–56, September 2008.

[18] Christian Bienia, Sanjeev Kumar, Jaswinder Pal Singh, and Kai Li. The parsec benchmark suite: characterization and architectural implications. In *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*, pages 72–81, October 2008.

[19] Guy E. Blelloch, Charles E. Leiserson, Bruce M. Maggs, C. Greg Plaxton, Stephen J. Smith, and Marco Zagha. A comparison of sorting algorithms for the connection machine cm-2. In *Proceedings of the 3rd Annual ACM Symposium on Parallelism in Algorithms and Architectures*, pages 3–16, New York, NY, USA, 1991. ACM.

[20] S. Bourduas and Z. Zilic. A hybrid ring/mesh interconnect for network-on-chip using hierarchical rings for global routing. In *Networks-on-Chip, 2007. NOCS 2007. First International Symposium on*, pages 195 –204, may 2007.

[21] M. Butler, L. Barnes, D.D. Sarma, and B. Gelinas. Bulldozer: An approach to multithreaded compute performance. *Micro, IEEE*, 31(2):6 –15, march-april 2011.

[22] Yi-Jung Chen, Chia-Lin Yang, and Yen-Sheng Chang. An architectural co-synthesis algorithm for energy-aware network-on-chip design. *J. Syst. Archit.*, 55(5-6):299–309, 2009.

[23] Chen-Ling Chou and R. Marculescu. Incremental run-time application mapping for homogeneous nocs with multiple voltage levels. In *Hardware/Software Codesign and System Synthesis (CODES+ISSS), 2007 5th IEEE/ACM/IFIP International Conference on*, pages 161 –166, 30 2007-oct. 3 2007.

[24] Chen-Ling Chou and R. Marculescu. User-aware dynamic task allocation in networks-on-chip. In *Design, Automation and Test in Europe, 2008. DATE '08*, pages 1232 –1237, march 2008.

[25] Chen-Ling Chou and Radu Marculescu. Run-time task allocation considering user behavior in embedded multiprocessor networks-on-chip. *Trans. Comp.-Aided Des. Integ. Cir. Sys.*, 29(1):78–91, January 2010.

[26] James Cooley and John Tukey. An algorithm for the machine calculation of complex fourier series. *Mathematics of Computation*, 19(90):297–301, 1965.

[27] Intel Corporation. *Intel 64 and IA-32 Architectures Optimization Reference Manual.* Intel Corporation, 2011.

[28] William J. Dally and Brian Towles. Route packets, not wires: on-chip inteconnection networks. In *Proceedings of the 38th conference on Design automation*, pages 684–689, June 2001.

[29] W.R. Davis, J. Wilson, S. Mick, J. Xu, H. Hua, C. Mineo, A.M. Sule, M. Steer, and P.D. Franzon. Demystifying 3d ics: the pros and cons of going vertical. *Design Test of Computers, IEEE*, 22(6):498 – 510, nov. 2005.

[30] E.L. de Souza Carvalho, N.L.V. Calazans, and F.G. Moraes. Dynamic task mapping for mpsocs. *Design Test of Computers, IEEE*, 27(5):26 –35, sept.-oct. 2010.

[31] H. Dejonghe. A completely analytical family of anisotropic Plummer models. *Royal Astronomical Society, Monthly Notices*, 224:13–39, January 1987.

[32] P. Duhamel and H. Hollmann. 'split radix' fft algorithm. *Electronics Letters*, 20(1):14 –16, 5 1984.

[33] Noel Eisley, Vassos Soteriou, and Li-Shiuan Peh. High-level power analysis for multi-core chips. In *Proceedings of the 2006 international conference on Compilers, architecture and synthesis for embedded systems*, CASES '06, pages 389–400, New York, NY, USA, 2006. ACM.

[34] C. Eklund, R.B. Marks, K.L. Stanwood, and S. Wang. Ieee standard 802.16: a technical overview of the wirelessmantm air interface for broadband wireless access. *Communications Magazine, IEEE*, 40(6):98 –107, jun 2002.

[35] G. Faria, J.A. Henriksson, E. Stare, and P. Talmola. Dvb-h: Digital broadcast services to handheld devices. *Proceedings of the IEEE*, 94(1):194 –209, jan. 2006.

[36] T. Fischer, S. Arekapudi, E. Busta, C. Dietz, M. Golden, S. Hilker, A. Horiuchi, K.A. Hurd, D. Johnson, H. McIntyre, S. Naffziger, J. Vinh, J. White, and K. Wilcox. Design solutions for the bulldozer 32nm soi 2-core processor module in an 8-core cpu. In *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2011 IEEE International*, pages 78 –80, feb. 2011.

[37] Brian R. Gaeke, Parry Husbands, Xiaoye S. Li, Leonid Oliker, Katherine A. Yelick, and Rupak Biswas. Memory-intensive benchmarks: Iram vs. cache-based machines. In *Proceedings of the 16th International Parallel and Distributed Processing Symposium*, page 203, April 2002.

[38] HiTech Global. Ddr 2 memory controller ip core for fpga and asic, June 2010. http://www.hitechglobal.com/ipcores/ddr2controller.htm.

[39] HiTech Global. Ddr 3 sdram memory controller ip core, May 2011. http://www.hitechglobal.com/IPCores/DDR3Controller.htm.

[40] Cristian Grecu, Partha Pratim Pande, Andre Ivanov, and Res Saleh. Structured interconnect architecture: a solution for the non-scalability of bus-based socs. In *Proceedings of the 14th ACM Great Lakes symposium on VLSI*, pages 192–195, April 2004.

[41] Leslie Frederick Greengard. *The rapid evaluation of potential fields in particle systems.* PhD thesis, New Haven, CT, USA, 1987. AAI8727216.

[42] M. Hakem and F. Butelle. Dynamic critical path scheduling parallel programs onto multiprocessors. In *Proceedings of 19th IEEE International Parallel and Distributed Processing Symposium*, page 203b, 2005.

[43] T. Hamada and K. Nitadori. 190 tflops astrophysical n-body simulation on a cluster of gpus. In *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1 –9, nov. 2010.

[44] R.H. Havemann and J.A. Hutchby. High-performance interconnects: an integration overview. *Proceedings of the IEEE*, 89(5):586 –601, may. 2001.

[45] C. Holt and J. P. Singh. Hierarchical n-body methods on shared address space multiprocessors. In *Proceedings of the Seventh Siam Conference on Parallel Processing for Scientific Computing*, 1995.

[46] Jingcao Hu and Radu Marculescu. Energy-aware communication and task scheduling for network-on-chip architectures under real-time constraints. In *DATE '04*, page 10234, Washington, DC, USA, 2004. IEEE Computer Society.

[47] Jaehyuk Huh, Changkyu Kim, Hazim Shafi, Lixin Zhang, Doug Burger, and Stephen W. Keckler. A nuca substrate for flexible cmp cache sharing. In *Proceedings of the 19th international conference on Supercomputing*, pages 31–40, June 2005.

[48] IBM. Ibm power 7 processor. In *Hot chips 2009*, August 2009.

[49] Cisco Systems Inc. *Internetworking Technologies Handbook (4th Edition).* Cisco Press, 2003.

[50] Intel. Intel core i7 processor extreme edition and intel core i7 processor datasheet, volume 1, December 2008. http://download.intel.com/design/processor/datashts/320834.pdf.

[51] Intel. Single-chip cloud computer, May 2010. http://techresearch.intel.com/articles/Tera-Scale/1826.htm.

[52] Intel. Intel xeon processor e7-8870, July 2011. http://ark.intel.com/Product.aspx?id=53580.

[53] Intel. Intel core i7-980x processor extreme edition, January 2012. http://ark.intel.com/Product.aspx?id=47932.

[54] Axel Jantsch and Hannu Tenhunen. *Networks-on-Chip*. Kluwer Academic Publishers, 2003.

[55] P. Jetley, L. Wesolowski, F. Gioachin, L.V. Kale? and, and T.R. Quinn. Scaling hierarchical n-body simulations on gpu clusters. In *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1 –11, nov. 2010.

[56] J.W. Joyner, P. Zarkesh-Ha, and J.D Meindl. A stochastic global net-length distribution for a three-dimensionalsystem-on-a-chip (3d-soc). In *Proceedings of the 14th Annual IEEE International ASIC/SOC Conference*, pages 147–151, September 2001.

[57] A.B. Kahng, Bin Li, Li-Shiuan Peh, and K. Samadi. Orion 2.0: A fast and accurate noc power and area model for early-stage design space exploration. In *Design, Automation Test in Europe Conference Exhibition, 2009. DATE '09.*, pages 423 –428, april 2009.

[58] Changkyu Kim, Doug Burger, and Stephen W. Keckler. An adaptive, non-uniform cache structure for wire-delay dominated on-chip caches. In *ACM SIGPLAN*, pages 211–222, October 2002.

[59] Yoongu Kim, Dongsu Han, O. Mutlu, and M. Harchol-Balter. Atlas: A scalable and high-performance scheduling algorithm for multiple memory controllers. In *2010 IEEE 16th International Symposium on High Performance Computer Architecture (HPCA)*, pages 1 –12, 2010.

[60] P. Kongetira, K. Aingaran, and K. Olukotun. Niagara: a 32-way multithreaded sparc processor. *IEEE Micro*, 25(2):21–29, March 2005.

[61] KTH. Nostrum, December 2011. http://www.ict.kth.se/nostrum.

[62] Man lap Li, Ruchira Sasanka, Sarita V. Adve, Yen kuang Chen, and Eric Debes. The alpbench benchmark suite for complex multimedia applications. In *In Proceedings of the IEEE International Symposium on Workload Characterization*, pages 34–45, 2005.

[63] John H. Lau. Tsv manufacturing yield and hidden costs for 3d ic integration. In *Proceedings of the 60th Electronic Components and Technology Conference*, pages 1031 –1042, 2010.

[64] James Laudon and Daniel Lenoski. The sgi origin: a ccnuma highly scalable server. In *Proc. of the 24th international symposium on Computer architecture*, pages 241–251, June 1997.

[65] Hyung Gyu Lee, U.Y. Ogras, R. Marculescu, and N. Chang. Design space exploration and prototyping for on-chip multimedia applications. In *2006 43rd ACM/IEEE Design Automation Conference*, pages 137–142, September 2006.

[66] Jae W. Lee, Man Cheuk Ng, and Krste Asanovic. Globally-synchronized frames for guaranteed quality-of-service in on-chip networks. In *Proceedings of the 35th International Symposium on Computer Architecture*, pages 89–100, Washington, DC, USA, 2008. IEEE Computer Society.

[67] Tang Lei and S. Kumar. A two-step genetic algorithm for mapping task graphs to a network on chip architecture. In *Proceedings of the 2003 Euromicro Symposium on Digital Systems Design*, pages 180 – 187, sep. 2003.

[68] Ville Leppnen. Studies on the realization of pram, 1996.

[69] Scott T. Leutenegger and Mary K. Vernon. The performance of multiprogrammed multiprocessor scheduling algorithms. In *Proceedings of the 1990 ACM SIGMETRICS Conference*, pages 226–236, April 1990.

[70] Feihui Li, Chrysostomos Nicopoulos, Thomas Richardson, Yuan Xie, Vijaykrishnan Narayanan, and Mahmut Kandemir. Design and management of 3d chip multiprocessors using network-in-memory. In *Proceedings of the 33rd annual international symposium on Computer architecture*, pages 130–141, June 2006.

[71] Gabriel H. Loh. 3d-stacked memory architectures for multi-core processors. In *ISCA '08: Proceedings of the 35th Annual International Symposium on Computer Architecture*, pages 453–464, Washington, DC, USA, 2008. IEEE Computer Society.

[72] Gian Luca Loi, Banit Agrawal, Navin Srivastava, Sheng-Chih Lin, Timothy Sherwood, and Kaustav Banerjee. A thermally-aware performance analysis of vertically integrated (3-d) processor-memory hierarchy. In *DAC '06: Proceedings of the 43rd annual Design Automation Conference*, pages 991–996, New York, NY, USA, 2006. ACM.

[73] P.S. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hallberg, J. Hogberg, F. Larsson, A. Moestedt, and B. Werner. Simics: A full system simulation platform. *Computer*, 35(2):50–58, February 2002.

[74] Marcelo Mandelli, Alexandre Amory, Luciano Ost, and Fernando Gehm Moraes. Multi-task dynamic mapping onto noc-based mpsocs. In *Proceedings of the 24th symposium on Integrated circuits and systems design*, SBCCI '11, pages 191–196, New York, NY, USA, 2011. ACM.

[75] Milo M.K. Martin, Daniel J. Sorin, Bradford M. Beckmann, Michael R. Marty, Min Xu, Alaa R. Alameldeen, Kevin E. Moore, Mark D. Hill, and David A. Wood. Multifacet's general execution-driven multiprocessor simulator (gems) toolset. *Computer Architecture News*, September 2005.

[76] Onur Mutlu and Thomas Moscibroda. Parallelism-aware batch scheduling: Enhancing both performance and fairness of shared dram systems. *SIGARCH Comput. Archit. News*, 36(3):63–74, 2008.

[77] Kyle J. Nesbit, Nidhi Aggarwal, James Laudon, and James E. Smith. Fair queuing memory systems. In *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 208–222, Washington, DC, USA, 2006. IEEE Computer Society.

[78] NVIDIA. Nvidia tegra, December 2011. http://www.nvidia.com/object/tegra.html.

[79] Lars Nyland, Mark Harris, and Jan Prins. Fast N-Body Simulation with CUDA. In Hubert Nguyen, editor, *GPU Gems 3*, chapter 31. Addison Wesley Professional, August 2007.

[80] University of Catania. Noxim, an open network-on-chip simulator, December 2011. http://noxim.sourceforge.net.

[81] Cheolmin Park, R. Badeau, L. Biro, J. Chang, T. Singh, J. Vash, Bo Wang, and T. Wang. A 1.2 tb/s on-chip ring interconnect for 45nm 8-core enterprise xeon processor. In *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2010 IEEE International*, pages 180 –181, feb. 2010.

[82] Dongkook Park, Soumya Eachempati, Reetuparna Das, Asit K. Mishra, Yuan Xie, N. Vijaykrishnan, and Chita R. Das. Mira: A multi-layered on-chip interconnect router architecture. In *Proceedings of the 35th annual international symposium on Computer architecture*, pages 251–261, June 2008.

[83] Avadh Patel and Kanad Ghose. Energy-efficient mesi cache coherence with pro-active snoop filtering for multicore microprocessors. In *Proceeding of the thirteenth international symposium on Low power electronics and design*, pages 247–252, August 2008.

[84] Vasilis Pavlidis and Eby Friedman. *Three-Dimensional Integrated Circuit Design*. Morgan Kaufmann, 2008.

[85] Vasilis F. Pavlidis and Eby G. Friedman. 3-d topologies for networks-on-chip. *IEEE Transaction on Very Large Scale Integration System*, 15(10):1081–1090, 2007.

[86] Fernando C. Pereira and Touradj Ebrahimi. *The MPEG-4 Book*. Prentice Hall, 2002.

[87] L.F. Perrone and D.M. Nicol. Using n-body algorithms for interference computation in wireless cellular simulations. In *Proceedings of the 8th Internations Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, pages 49 –56, 2000.

[88] A. Pullini, F. Angiolini, S. Murali, D. Atienza, G. De Micheli, and L. Benini. Bringing nocs to 65 nm. *Micro, IEEE*, 27(5):75 –85, sept.-oct. 2007.

[89] Kiran Puttaswamy and Gabriel H. Loh. Implementing caches in a 3d technology for high performance processors. In *ICCD '05: Proceedings of the 2005 International Conference on Computer Design*, pages 525–532, Washington, DC, USA, 2005. IEEE Computer Society.

[90] Brian M. Rogers, Anil Krishna, Gordon B. Bell, Ken Vu, Xiaowei Jiang, and Yan Solihin. Scaling the bandwidth wall: challenges in and avenues for cmp scaling. In *Proceedings of the 36th annual international symposium on Computer architecture*, pages 371–382, June 2009.

[91] J. Salmon. Parallel n log n n-body algorithms and applications to astrophysics. In *Compcon Spring '91. Digest of Papers*, pages 73 –78, feb-1 mar 1991.

[92] Patrick Schmid and Achim Roos. Core i7 memory scaling: From ddr3-800 to ddr3-1600, September 2009. Tom's Hardware.

[93] Debendra Das Sharma and Dhiraj K. Pradhan. Processor allocation in hypercube multicomputers: Fast and efficient strategies for cubic and noncubic allocation. *IEEE Transactions on parallel and distributed systems*, 6(10):1108–1123, October 1995.

[94] Thoziyoor Shyamkumar, Muralimanohar Naveen, Ahn Jung Ho, and Jouppi Norman P. Cacti 5.1. Technical Report HPL-2008-20, HP Labs.

[95] F.N. Sibai. Adapting the hyper-ring interconnect for many-core processors. In *Parallel and Distributed Processing with Applications, 2008. ISPA '08. International Symposium on*, pages 649 –654, dec. 2008.

[96] Jaswinder Pal Singh, John L. Hennessy, and Anoop Gupta. Implications of hierarchical n-body methods for multiprocessor architectures. *ACM Tran. Comp. Sys.*, 13:141–202, May 1995.

[97] Kevin Skadron, Mircea R. Stan, Karthik Sankaranarayanan, Wei Huang, Sivakumar Velusamy, and David Tarjan. Temperature-aware microarchitecture: Modeling and implementation. *ACM Transactions on Architecture and Code Optimization*, 1(1):94–125, March 2004.

[98] SPEC. Specjbb 2000. http://www.spec.org/jbb2000/.

[99] Herbert Sullivan and T R Bashkow. A large scale, homogeneous, fully distributed parallel machine. In *Proceedings of the 4th annual symposium on Computer architecture*, pages 105–117, March 1977.

[100] D. Sylvester and K. Keutzer. Getting to the bottom of deep submicron. In *Computer-Aided Design, 1998. ICCAD 98. Digest of Technical Papers. 1998 IEEE/ACM International Conference on*, pages 203–211, Nov 1998.

[101] K. Takahashi and M. Sekiguchi. Through silicon via and 3-d wafer/chip stacking technology. In *VLSI Circuits, 2006. Digest of Technical Papers. 2006 Symposium on*, pages 89 –92, 0-0 2006.

[102] Tilera. Tile-gx processor family, May 2011. http://www.tilera.com/products/processors/TILE-Gx_Family.

[103] T. Tillo, M. Grangetto, and G. Olmo. Redundant slice optimal allocation for h.264 multiple description coding. *IEEE Transactions on Circuits and Systems for Video Technology*, 18(1):59–70, January 2008.

[104] TPC. Tpc-h decision support benchmark. http://www.tpc.org/tpch/.

[105] Marc Tremblay and Shailender Chaudhry. A third-generation 65nm 16-core 32-thread plus 32-scout-thread cmt sparc processor. In *2008*

IEEE International Solid-State Circuits Conference, pages 82–83, February 2008.

[106] Yuh-Fang Tsai, Yuan Xie, N. Vijaykrishnan, and Mary Jane Irwin. Three-dimensional cache design exploration using 3dcacti. In *ICCD '05: Proceedings of the 2005 International Conference on Computer Design*, pages 519–524, Washington, DC, USA, 2005. IEEE Computer Society.

[107] S. Vangal, J. Howard, G. Ruhl, S. Dighe, H. Wilson, J. Tschanz, D. Finan, P. Iyer, A. Singh, T. Jacob, S. Jain, S. Venkataraman, Y. Hoskote, and N. Borkar. An 80-tile 1.28tflops network-on-chip in 65nm cmos. In *Solid-State Circuits Conference, 2007. ISSCC 2007. Digest of Technical Papers. IEEE International*, pages 98–589, Feb. 2007.

[108] Athanasios V. Vasilakos, Christos A. Moschonas, and Constantinos T. Paximadis. Adaptive window flow control and learning algorithms for adaptive routing in data networks. *ACM SIGMETRICS Performance Evaluation Review*, 18(1):265–266, May 1990.

[109] D. Velenis, M. Stucchi, E.J. Marinissen, B. Swinnen, and E. Beyne. Impact of 3d design choices on manufacturing cost. In *3D System Integration, 2009. 3DIC 2009. IEEE International Conference on*, pages 1–5, Sept. 2009.

[110] Jesús Camacho Villanueva, José Flich, José Duato, Hans Eberle, Nils Gura, and Wladek Olesinski. A performance evaluation of 2d-mesh, ring, and crossbar interconnects for chip multi-processors. In *Proceedings of the 2nd International Workshop on Network on Chip Architectures*, NoCArc '09, pages 51–56, New York, NY, USA, 2009. ACM.

[111] Hang-Sheng Wang, Xinping Zhu, Li-Shiuan Peh, and S. Malik. Orion: a power-performance simulator for interconnection networks. In *Proceedings of the 35th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 294–305, November 2002.

[112] Scott Wasson. Intel's xeon 5600 processors, westmere-ep adds two more cores to an already-potent mix, July 2010. http://techreport.com/articles.x/19196/4.

[113] Scott Wasson. A quick look at bulldozer thread scheduling, October 2011. http://techreport.com/articles.x/21865.

[114] A.Y. Weldezion, Zhonghai Lu, R. Weerasekera, and H. Tenhunen. 3-d memory organization and performance analysis for multi-processor network-on-chip architecture. In *3D System Integration, 2009. 3DIC 2009. IEEE International Conference on*, pages 1 –7, 28-30 2009.

[115] D. Wentzlaff, P. Griffin, H. Hoffmann, Liewei Bao, B. Edwards, C. Ramey, M. Mattina, Chyi-Chang Miao, J.F. Brown, and A. Agarwal. On-chip interconnection architecture of the tile processor. *Micro, IEEE*, 27(5):15 –31, sept.-oct. 2007.

[116] Steven Cameron Woo, Moriyoshi Ohara, Evan Torrie, Jaswinder Pal Singh, and Anoop Gupta. The splash-2 programs: Characterization and methodological considerations. In *Proceedings of the 22nd International Symposium on Computer Architecture*, pages 24–36, June 1995.

[117] Steven Cameron Woo, Jaswinder Pal Singh, and John L. Hennessy. The performance advantages of integrating block data transfer in cache-coherent multiprocessors. In *Proceedings of the Sixth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 219–229, New York, NY, USA, 1994. ACM.

[118] Qi Wu, K. Rose, Jian-Qiang Lu, and Tong Zhang. Impacts of though-dram vias in 3d processor-dram integrated systems. In *3D System Integration, 2009. 3DIC 2009. IEEE International Conference on*, pages 1–6, Sept. 2009.

[119] T.C. Xu, Liang Guang, A.W. Yin, Bo Yang, P. Liljeberg, and H. Tenhunen. An analysis of designing 2d/3d chip multiprocessor wit different cache architecture. In *NORCHIP, 2010*, pages 1 –6, nov. 2010.

[120] T.C. Xu, P. Liljeberg, and H. Tenhunen. A study of through silicon via impact to 3d network-on-chip design. In *Electronics and Information Engineering (ICEIE), 2010 International Conference On*, volume 1, pages V1–333 –V1–337, aug. 2010.

[121] T.C. Xu, P. Liljeberg, and H. Tenhunen. Explorations of optimal core and cache placements for chip multiprocessor. In *NORCHIP, 2011*, pages 1 –6, nov. 2011.

[122] T.C. Xu, P. Liljeberg, and H. Tenhunen. Optimal number and placement of through silicon vias in 3d network-on-chip. In *Design and Diagnostics of Electronic Circuits Systems (DDECS), 2011 IEEE 14th International Symposium on*, pages 105 –110, april 2011.

[123] T.C. Xu, Bo Yang, A.W. Yin, P. Liljeberg, and H. Tenhunen. 3d network-on-chip with on-chip dram: an empirical analysis for future chip multiprocessor. *World Academy of Science, Engineering and Technology*, 70:18–24, october 2010.

[124] T.C. Xu, A.W. Yin, P. Liljeberg, and H. Tenhunen. A study of 3d network-on-chip design for data parallel h.264 coding. In *NORCHIP, 2009*, pages 1 –6, nov. 2009.

[125] Thomas Canhao Xu, Pasi Liljeberg, and Hannu Tenhunen. A study of through silicon via impact to 3d network-on-chip design. In *Proceedings of the 2010 International Conference on Electronics and Information Engineering (ICEIE 2010)*, August 2010.

[126] Thomas Canhao Xu, Pasi Liljeberg, and Hannu Tenhunen. A minimal average accessing time scheduler for multicore processors. In *Proceedings of the 11th international conference on Algorithms and architectures for parallel processing - Volume Part II*, ICA3PP'11, pages 287–299, Berlin, Heidelberg, 2011. Springer-Verlag.

[127] Thomas Canhao Xu, Pasi Liljeberg, and Hannu Tenhunen. Optimal memory controller placement for chip multiprocessor. In *Proceedings of the seventh IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, CODES+ISSS '11, pages 217–226, New York, NY, USA, 2011. ACM.

[128] Thomas Canhao Xu, Pasi Liljeberg, and Hannu Tenhunen. Optimal number and placement of through silicon vias in 3d network-on-chip. In *Proceedings of the 14th IEEE International Symposium on Design and Diagnostics of Electronic Circuits and Systems*. IEEE, 2011.

[129] Thomas Canhao Xu, Pasi Liljeberg, and Hannu Tenhunen. Process scheduling for future multicore processors. In *Proceedings of the Fifth International Workshop on Interconnection Network Architecture: On-Chip, Multi-Chip*, INA-OCMC '11, pages 15–18, New York, NY, USA, 2011. ACM.

[130] Thomas Canhao Xu, Alexander Wei Yin, Pasi Liljeberg, and Hannu Tenhunen. Operating system processor scheduler design for future chip multiprocessor. *Architecture of Computing Systems (ARCS), 2010 23rd International Conference on*, pages 1 –7, feb. 2010.

[131] Thomas Canhao Xu, Alexander Wei Yin, Pasi Liljeberg, and Hannu Tenhunen. A study of 3d network-on-chip design for data parallel h.264 coding. *Microprocessors and Microsystems*, 35(7):603 – 612, 2011.

[132] Abdulrahman Yarali and Babak Ahsant. 802.11n: the new wave in wlan technology. In *Proceedings of the 4th international conference on mobile technology, applications, and systems and the 1st international symposium on Computer human interaction in mobile technology*, Mobility '07, pages 310–316, New York, NY, USA, 2007. ACM.

[133] Alexander Wei Yin, Liang Guang, Ethiopia Nigussie, Pasi Liljeberg, Jouni Isoaho, and Hannu Tenhunen. Dvfs architecture exploration of per-core dvfs for energy-constrained on-chip networks. In *Proceeding of 12th EUROMICRO Conference on Digital System Design*, August 2009.

[134] Alexander Wei Yin, Liang Guang, Pekka Rantala, Pasi Liljeberg, Jouni Isoaho, and Hannu Tenhunen. Hierarchical agent based noc with dynamic online services. In *Proceeding of 4th IEEE Conference on Industrial Electronics and Applications (ICIEA2010)*, May 2009.

ISBN 978-952-12-2772-1
ISSN 1239-1883

Thomas Canhao Xu – 许粲昊

Hardware/Software Co-Design for Multicore Architectures

Thomas Canhao Xu – 许粲昊

Hardware/Software Co-Design for Multicore Architectures