Analysing and Modelling the On-chip Traffic of Parallel Applications

Thomas Canhao Xu, Jonne Pohjankukka and Ville Leppänen
Department of Information Technology, University of Turku, Turku, Finland
canxu@utu.fi

***Abstract*—In this paper, we investigate the traffic characteristics of parallel and high performance computing applications. Parallel applications that utilize multiple processing cores are widespread nowadays due to the trend of multicore processors. However the design paradigm of traditional sequential execution and concurrent execution can vary significantly. Therefore the estimation and prediction approaches used in conventional software can be limited for parallel applications. The communication among different nodes in a multicore system should be analysed and categorized in order to improve the accuracy of system simulation. We study several parallel applications running on a full system simulation environment. The communication traces among different nodes are collected and analysed. We discuss the detailed characteristics of these applications. The applications are grouped into different categories depending on several parallel programming paradigms. We apply power-law model with maximum likelihood estimation, Gaussian mixture model, as well as the polynomial model for fitting the trace data. A generic synthetic traffic model is proposed based on the results. Experiments show the proposed model can be used to evaluate the performance of parallel systems more accurately than by other synthetic traffic models.**

I. Introduction and Related Work

Parallel applications are more and more common nowadays due to the widely utilization of multicore processors. In fact the number of cores integrated on a multicore chip is increasing rapidly. It is hard to integrate multicore processors into smartphones a decade ago, however today smart devices such as phones and tablets are equipped with 8-core or even 10-core processors [14]. To utilize the processing power of multiple cores, it is critical to design an efficient parallel software system. According to Amdahl's law [21], theoretically the speedup of executing a parallel program in a multiprocessor environment depends on the portion of code that can be parallelized. In real world, there are still several problems to be addressed: first, the percentage of parallel codes in a program differs depending on the problem to be solved; second, there are other overheads and bottlenecks from the system such as core-core communication. A parallel program can generate huge amount of traffic to exchange and synchronize data, causing performance bottlenecks. Therefore high bandwidth scalable on-chip interconnection networks, such as fat tree, mesh and torus are proposed for massive scalable multicore processors with tens or even hundreds of cores [8] [28].

In a mesh-based multicore processor, the computational resources are connected by a general communication infrastructure [29]. Figure 1 illustrates a typical processor with 4×4 mesh network, where 16 Processing Elements (PEs) are connected by Routers (R) and related links. The processor cores, related L1 and L2 caches, and Network Interfaces (NIs) are integrated in the PE. Communication among different PEs are performed by transmitting data packets. The scalability and throughput of the interconnect are improved compared with conventional bus-based systems, since the interconnect can process multiple transactions simultaneously. Furthermore the modular design provides better expandability. Commercial multicore processors consisting up to 72 cores on a single chip are available with 8×9 mesh network [24].

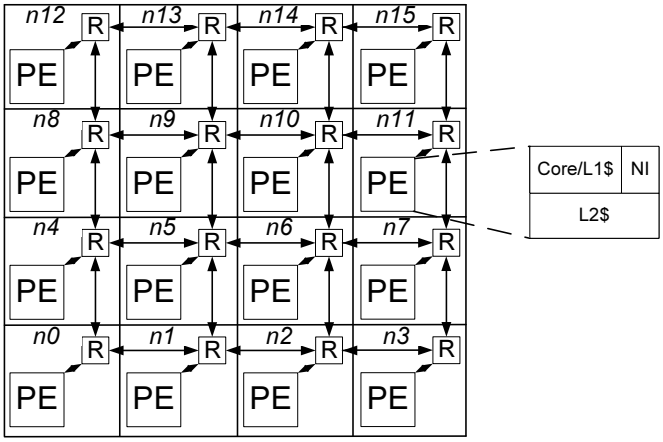


Fig. 1: A multicore processor with 4×4 mesh.

To estimate and predict the performance of parallel applications and on-chip networks, a simulation environment is usually used with different traffic configurations. The traffic pattern can be synthetic which represents an abstract model of transmitted data packets among nodes, or realistic which follows actual applications running on the system. Obviously realistic traffic patterns from parallel applications are more meaningful, however the traffic pattern for different applications can vary significantly [26] [27], making the evaluation process more challenging. Several traffic suites and models are proposed by various research groups [17] [11] [3] [3]. Specific traffic models are extracted from multimedia and signal processing tasks, where tasks are represented by a set of task graphs including nodes, edges and weights [17] [11]. However it can be difficult to reflect the performance of the multicore processor since applications are usually executed

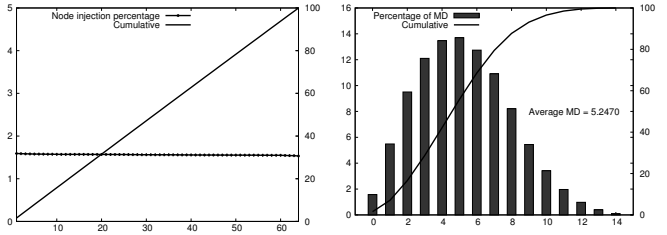
with processes and threads, and thus have different communication pattern compared with task graph. Rent's rule is used to characterise the traffic behaviour of parallel applications. A model based on Rent's rule was proposed to predict NoC traffic locality and the distribution of communication probability [3].

Several papers analysed the traffic model based on empirical application data [30] [23] [2] [6] [3] [1]. For example full system simulation is used to gather traces in [23]. The model considers both spatial and temporal characteristic of the traffic. The paper also proposed a process of generating synthetic traces based on the application traffic. The authors conducted experiments based on several system configurations: 4-core TRIPS processor, 16-core traditional processor and 16-core cache coherent processor (4×4 mesh). The previous research is extended with 7×7 mesh network [2]. However both processors in the two researches were based on the MSI cache coherent protocol which is rarely used currently. In [1], the authors extended the previous research with more advanced MOESI protocol, however only a 16-core processor is simulated. Studies in [30] investigated 6 parallel applications with a 64-core processor, the traffic was categorized into 2 groups and a power-law model was proposed with different parameters. Another paper proposed a statistical model based on quantum-leap [6], the method can account for non-stationarity observed in packet arrival processes. The approach is claimed to have advantages in estimating the probability of missing deadlines in packets. In this paper, we first investigate several parallel applications which are widely used. The traffic patterns of these applications are analysed and discussed. We discover similarities among the application traffic traces. We propose mathematical models based on the analysis of traces.

II. Motivation

Synthetic traffic models include uniform random, transpose, bit-complement, bit-reverse and hotspot etc. [8]. Here we discuss a widely used pattern: the uniform random traffic, in which each node generates packets in equally random possibility with random destinations. Therefore the source and destination nodes in a packet are random and uniform. Figure 2 illustrates the results of uniform random traffic with 1M packets for an 8×8 mesh. It is obvious that the number of packets injected to the network for all 64 nodes are basically the same, which is around 1.5625% ($1/64$). Similarly the cumulative sum of node injection percentage is linear. On the other hand, the distribution of Manhattan Distances (MDs) makes the traffic model far from the real application as well. We calculate the average Manhattan Distance (MD) of all source-destination pairs is 5.2470.

Two aspects must be considered for a traffic pattern: spatial, which represents the location of the source and destination nodes; and temporal, which represents the interval of packets. Theoretically, both spatial and temporal properties can be modelled by using synthetic patterns. However the variations of the two properties must be captured to fit the actual applications. Previous researches have focused on average injection rate of nodes, Gaussian-like distribution for injection rates of nodes, the average hop count of packets, or the burstiness of the traffic [23] [2] [6] [1]. While our research tries to give a generic model based on the average value and distribution of spatial and temporal attributes.



(a) UR Injection Rate (b) UR MDs

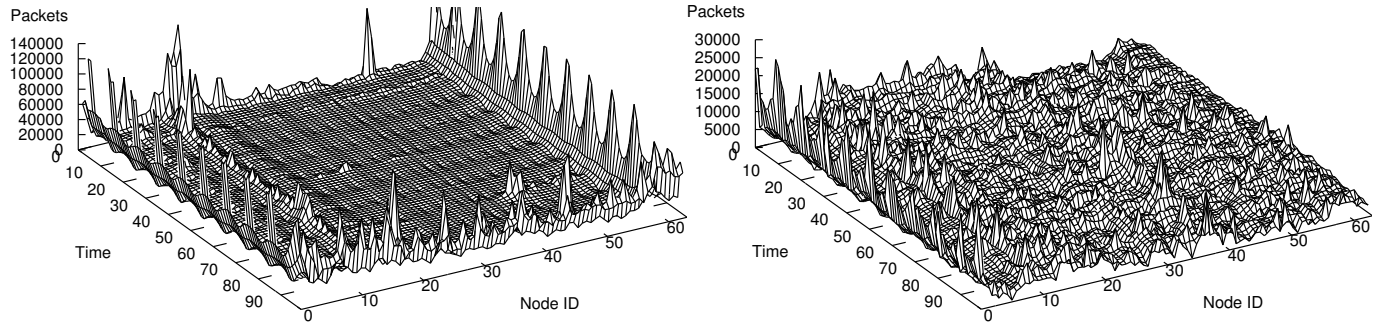
Fig. 2: (a), Traffic injection percentage (left Y-axis) and cumulative percentage (right Y-Axis) for 64 nodes (X-axis) with the uniform random (UR) traffic. (b), Percentage (left Y-axis) and cumulative percentage (right Y-axis) of MDs between sources and destinations of packets (X-axis) with the UR traffic.

III. Data Analysis and Parallel Programming Paradigms

A. Data Analysis

The traffic patterns are collected based on trace data of parallel programs running on a full system simulator [12] [13]. We configure a multicore processor with 64 UltraSPARC III+ cores running at 2GHz (8×8 mesh). Each node in the mesh consists of a processor core and shared caches. The private L1 cache is split into instruction and data cache, each 16KB with 3-cycle access delay. The unified shared L2 cache is split into 64 banks (1 bank per node), each 256KB with 6-cycle access delay. The simulated memory/cache architecture mimics static non-uniform cache architecture, where MOESI cache coherence protocol is implemented. We execute widely-used parallel applications from SPLASH-2 [25] and PARSEC [4] with 64 threads on Solaris 9 with 4GB memory.

Figure 3 illustrates the detailed traffic profiles in terms of transmitted packets from different nodes over the execution time of applications. Obviously the traffic of real-world parallel applications are different from the uniform random traffic. For example, it can be seen that a small portion of nodes generated significant amount of traffic. Besides, regular patterns, as well as traffic spikes can be observed from the traffic result of $Radix\ Sort$ (as well as $Barnes-Hut$, $Raytrace$ and $Water$, not illustrated due to page limitations). On the other hand, applications such as FFT, FMM, LU and $Swaptions$ did not show significant regular nor hot-spot traffic. The executed cycles and transmitted packets of applications and other experimental results are shown in Table I. In terms of PPC, it can be seen that the applications with significant hot-spot and regular traffic have lower packet injection rates (0.1024 for $Radix\ Sort$ to 0.1561 for $Raytrace$, average 0.1309) than the other 4 applications (0.2197 for FFT to 0.5850 for $Swaptions$, average 0.3754).

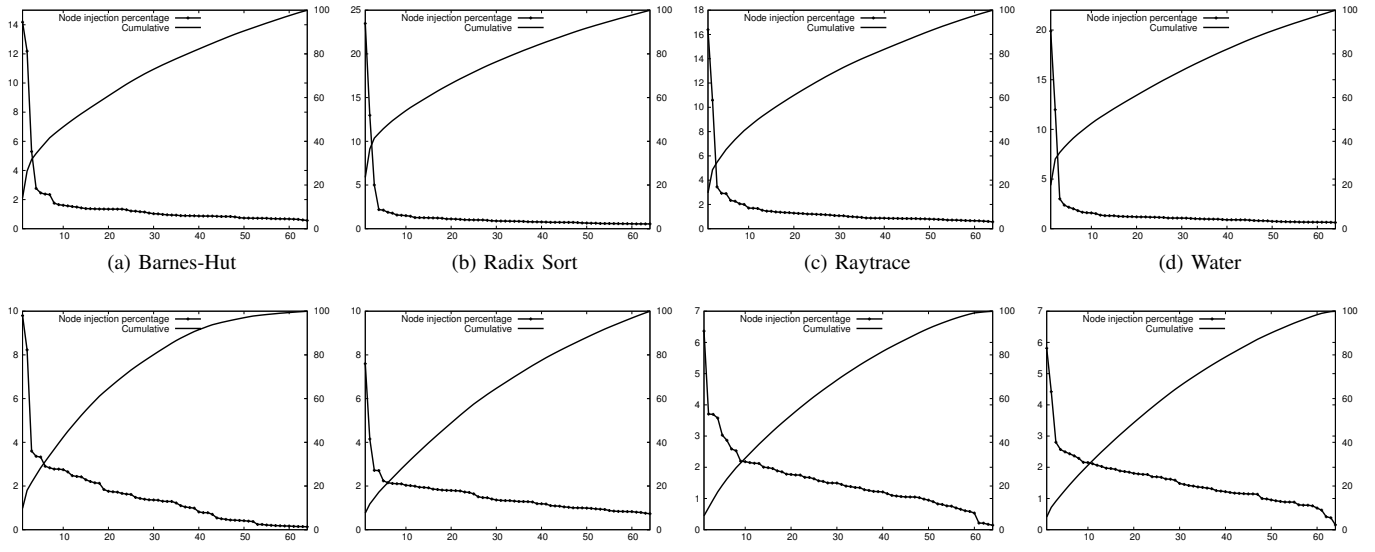


(a) Radix Sort (b) LU Matrix Decomposition (LU)

Fig. 3: Injected packets (Z-axis) for 64 nodes (X-axis) of two applications. The percentage of executed cycles/times is shown in Y-axis. The traffic figures of other six applications are not illustrated due to page limitations.

TABLE I: Profiles of different applications. TI%/4 and TI%/60 mean total injection percentage of top 4 and other 60 nodes respectively. PPC stands for Packet Per Cycle.

Application	Cycles	Packets	PPC	Category	Injection % of Top 4 nodes	TI%/4, TI%/60
Barnes-Hut	1146.7M	160.5M	0.1399	1	14.1%, 12.1%, 5.3%, 2.8%	34.3%, 65.7%
Radix Sort	1064.9M	109.1M	0.1024	1	23.5%, 13.0%, 5.0%, 2.2%	43.7%, 56.3%
Raytrace	399.5M	62.4M	0.1561	1	16.4%, 10.6%, 3.4%, 2.9%	33.3%, 66.7%
Water NSquared	687.9M	86.3M	0.1254	1	19.9%, 12.0%, 3.0%, 2.4%	37.3%, 62.7%
Fast Fourier Transform (FFT)	26.2M	5.7M	0.2197	2	9.8%, 8.2%, 3.6%, 3.4%	25.0%, 75.0%
Fast Multipole Method (FMM)	168.7M	57.4M	0.3402	2	7.6%, 4.2%, 2.7%, 2.7%	17.2%, 82.8%
LU Matrix Decomposition (LU)	98.1M	35.0M	0.3569	2	6.3%, 3.7%, 3.7%, 3.6%	17.3%, 82.7%
Swaptions	184.6M	108.0M	0.5850	2	5.8%, 4.4%, 2.8%, 2.6%	15.6%, 84.4%

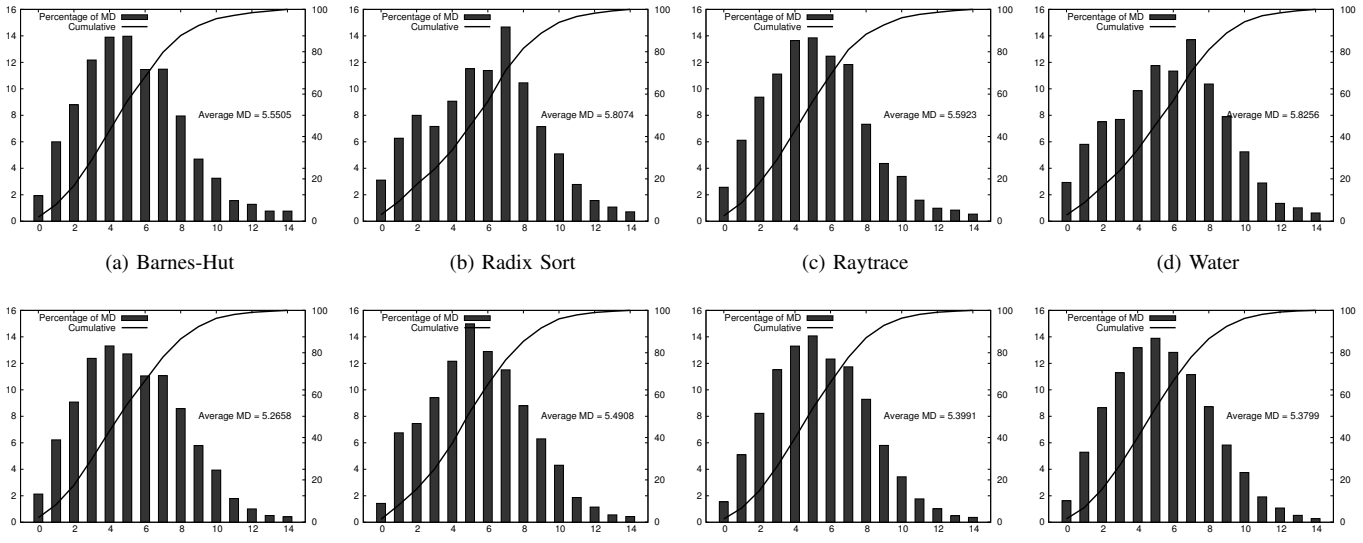


(a) Barnes-Hut (b) Radix Sort (c) Raytrace (d) Water

(e) Fast Fourier Transform (f) Fast Multipole Method (g) LU Matrix Decomposition (h) Swaptions

Fig. 4: Sorted packet injection percentage (left Y-axis) and accumulated percentage (right Y-axis) for 64 nodes (X-axis) of several applications.

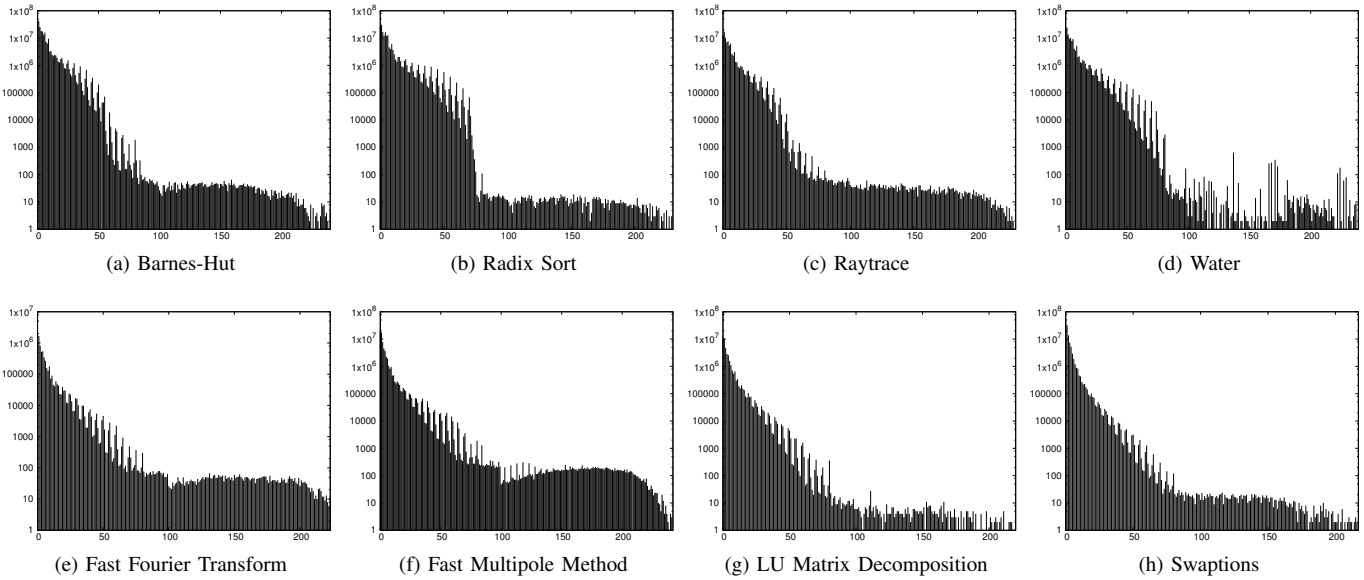
The percentages of injected packets by different nodes are demonstrated in Figure 4 and Table I. For example in $Radix\ Sort$, one node generated 23.5% of all traffic, where the top 4 nodes out of 64 generated 43.7% of all packets. The phenomenon is similar for other three applications as well ($Barnes-Hut$, $Raytrace$ and $Water$): 33.3% to 43.7% of traffic are concentrated in 4 nodes, while the remaining 60 nodes injected relatively small amount of traffic on average. The percentage of traffic for top 4 nodes is lower in other 4 applications: the total traffic from top 4 nodes contributed 15.6% to 25% of all traffic, in which the node with top injection rate generated 9.8% to 5.8% packets. The detailed node



(a) Barnes-Hut (b) Radix Sort (c) Raytrace (d) Water

(e) Fast Fourier Transform (f) Fast Multipole Method (g) LU Matrix Decomposition (h) Swaptions

Fig. 5: Percentage (left Y-axis) and accumulated percentage (right Y-axis) of MDs between source nodes and destination nodes of packets (X-axis).



(a) Barnes-Hut (b) Radix Sort (c) Raytrace (d) Water

(e) Fast Fourier Transform (f) Fast Multipole Method (g) LU Matrix Decomposition (h) Swaptions

Fig. 6: Interval (cycles) between packets (X-axis), and frequency of the interval (Y-axis).

injection rates and accumulated percentages are illustrated in Figure 4. It is noteworthy that the data/curve in the figure is similar to the power laws [16], where most of the effects come from a small portion of the causes. It can also be noticed that some of the curves are steeper than others.

The distributions of MDs between source and destination nodes in all packets are illustrated in Figure 5. The curve of accumulated values and average MD of all node pairs in all packets are also shown in the figure. Overall, compared with UR traffic, there are more packets with low (e.g. 0 to 1 hop) and high (e.g. 7 to 14 hops) MDs, while the packets with medium MDs are less than that in UR. Notice that in terms of MD distribution of packets, UR traffic is the closest to the actual applications, among other synthetic traffic patterns such as transpose, bit-complement, bit-reverse and nearest neighbour [3]. Furthermore, in terms of average MD, all applications have higher values than UR traffic. We notice that the aforementioned four applications with significant hot-spot traffic also have higher average packet MD (5.55 to 5.82), while the metric is lower for other applications (5.26 to 5.49).

Figure 6 shows the distribution of interval of packets. Most packets are injected to the system with a small interval, e.g.

less than 10 cycles between two packets, while long intervals are very rare in practise. We also notice the distribution of packet interval is similar to the power law. The aforementioned characteristics of applications can be explained by different programming models.

B. Parallel Programming Paradigms

The difference of on-chip traffic pattern can be affected by hardware such as cache coherence protocol, cache size and cache/memory architecture. On the other hand the software aspect can play a more important role. For instance, a parallel application usually applies a certain kind of programming paradigm, i.e. a class of methods/algorithms that have similar control structures [20]. Different programming paradigms can produce different traffic profiles. There are several common parallel programming paradigms including: Single Program Multiple Data (SPMD), Master-Slave (Process Farm), Divide and Conquer, Phase Parallel, Data Pipelining and Hybrids.

The choice of paradigm is usually determined by the given problem, as well as the limitations of hardware resources. Multiple paradigms can be used together in an application. For instance, the $Master-Slave$ model consists of a master process and several slave processes [15]. The master process is responsible for splitting the problem into smaller pieces, and allocating these to slave processes. Result or part of the result are collected by the master process. The traffic can exhibit in several $phases$ if the results are collected in an interval [18]. $Divide\ and\ Conquer$ is similar as $Master-Slave$, where the problem is decomposed dynamically. Parallel applications such as image, signal processing and graphic rendering utilize $Master-Slave$, $Divide\ and\ Conquer$ and $Phase\ Parallel$ models. On the other hand, $SPMD$ paradigm achieves data parallelism by using the same code of each process on different data [10]. Certain physical and mathematical problems have regular data structure that can be distributed to the processors uniformly. Obviously in most cases, the problem distributes to the system more evenly in $SPMD$ compared with other paradigms such as $Master-Slave$. We classify the applications into two categories based on the programming paradigms: *(1)* Master-Slave, Divide and Conquer, Phase Parallel paradigms; relatively significant hot-spot and/or phase (bursty) traffic; relatively low packet per cycle; higher average MD than UR traffic and category 2; distance between packets is generally shorter than category 2; *(2)* SPMD paradigm; relatively insignificant hot-spot and/or phase (bursty) traffic; relatively high packet per cycle; higher average MD than UR traffic, but lower than category 1; distance between packets is longer than category 1. Notice that the classification is general and non-specific since the border between two categories can be fuzzy. Moreover the categorization cannot cover all applications.

IV. Generic Synthetic Traffic Model

In this section, we analyse the traffic data of applications by using several mathematical models. A generic traffic generation algorithm based on the models is proposed and evaluated.

A. Power-law Fitting Using Maximum Likelihood Estimation and Gaussian Mixture Model

Many natural phenomena exhibit power-law distributed behavior. In this section our goal is to find the most likely power-law and Gaussian Mixture Model (GMM, [5]) distributions that could give rise to the packet injection percentages (Figure 4) and interval/frequency distribution (Figure 6) of category 1 and 2 applications. In [30] the Least-Squares (LS) fitting is used for finding the power-law model of the data. LS is a common method for analysing data which seems to follow power-law distribution but it is not always the optimal choice. The LS method can produce substantially inaccurate estimates of parameters for power-law distributions. Even though the LS can have a seemingly good fit to the data in many cases the solutions are not normalizable and hence they can not occur in nature [7]. In this paper we will use the methods described in [7] for finding the most likely power-law distribution. In addition, we will also calculate the goodness-of-fit of our power-law and GMM models using the *Kolmogorov-Smirnov* (KS, [7]) statistic.

A discrete random variable X obeys a power-law if it is drawn from a probability distribution:

$$p(x) = \Pr(X = x) \propto x^{-\alpha}, \tag{1}$$

where α is a constant called the *scaling parameter* of the power law distribution. The process of fitting empirical distributions into power-law distribution involves solving the scaling parameter α and some normalization constant. We will use the *maximum-likelihood estimation* (MLE, [5]) for estimating the scaling parameter α of the most likely power-law distribution that generated the data. For a set of n independent and identically distributed data points $\mathcal{X} = \{x_1, x_2, ..., x_n\}$ the MLE is defined as:

$$p(\mathcal{X} \mid \theta) = \prod_{i=1}^{n} p(x_i \mid \theta) \equiv \mathcal{L}(\theta), \tag{2}$$

where $\mathcal{L}(\theta)$ is the *likelihood function* and θ is the parameter(s) we want to solve by maximizing $\mathcal{L}(\theta)$. Usually it is practically more convenient to work with the negative of the logarithm of $\mathcal{L}(\theta)$:

$$E(\theta) = -\ln \mathcal{L}(\theta) = -\sum_{i=1}^{n} \ln p(x_i \mid \theta) \tag{3}$$

and then minimize this function.

For a discrete random variable X the power-law distribution is defined as:

$$p(x) = \Pr(X = x) = \frac{x^{-\alpha}}{\zeta(\alpha, x_{\min})}, \tag{4}$$

where

$$\zeta(\alpha, x_{\min}) = \sum_{m=0}^{\infty} (m + x_{\min})^{-\alpha} \tag{5}$$

is the Hurwitz zeta function. The variable $x_{\min} > 0$ denotes the lower bound on the power-law behavior of the random variable X. In our calculations we assumed that $x_{\min} = 1$. The cumulative distribution function CDF of power-law distributed discrete random variable X, denoted by $P(x)$ is defined as:

$$P(x) = \Pr(X \geq x) = \frac{\zeta(\alpha, x)}{\zeta(\alpha, x_{\min})}. \quad (6)$$

The MLE estimate $\hat{\alpha}$ for the scaling parameter α in Equation 4 is given by the solution to the transcendental equation

$$\frac{\zeta'(\hat{\alpha}, x_{\min})}{\zeta(\hat{\alpha}, x_{\min})} = -\frac{1}{n}\sum_{i=1}^{n} \ln x_i, \quad (7)$$

where the prime denotes differentiation with respect to the first argument and n denotes the number of data points. In practice the solution to Equation 7 is solved using numerical methods. In our calculations we used the bisection method also known as interval halving method. An estimate of the standard error for the scaling parameter α given by solution $\hat{\alpha}$ to Equation 7 is given by:

$$\sigma = \frac{1}{\sqrt{n\left[\frac{\zeta''(\hat{\alpha},x_{\min})}{\zeta(\hat{\alpha},x_{\min})} - \left(\frac{\zeta'(\hat{\alpha},x_{\min})}{\zeta(\hat{\alpha},x_{\min})}\right)^2\right]}}. \quad (8)$$

The GMM for a random variable X is defined as the weighted average of Gaussian probability density functions. Explicitly, we define the GMM model by:

$$p(x) = \sum_{i=1}^{k} P(\theta_i)p(x \mid \theta_i), \quad (9)$$

where $p(x)$ is the density value of the observed value x, $P(\theta_i)$ is the prior probability of ith Gaussian component and $p(x \mid \theta_i)$ is the density value of ith component for value x. For the prior probabilities of the mixture components we have $\sum_{i=1}^{N} P(\theta_i) = 1$ and $0 \leq P(\theta_i) \leq 1$. When X is a one-dimensional variable the Gaussian density functions in Equation 9 have the form:

$$p(x \mid \theta_i) = p(x \mid \mu_i, \sigma_i^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu_i)^2}{2\sigma^2}} \quad (10)$$

The MLE estimate for the parameters of the GMM model is usually solved by using iterative method called the *expectation-maximization* (EM) algorithm [9]. The EM iterates by alternating between performing an expectation (E) step, which creates a function for the expectation of the log-likelihood evaluated using the current estimate $\hat{\boldsymbol{\theta}} = \{\hat{\theta}_1, ..., \hat{\theta}_N\}$ for the parameters $\boldsymbol{\theta} = \{\theta_1, ..., \theta_N\}$, and a maximization (M) step, which computes parameters maximizing the expected log-likelihood found on the E step.

Figure 7 illustrates the MLE power-law and GMM fits for the packet injection percentages (Figure 4) and interval/frequency distribution (Figure 6) of category 1 and 2 applications. The number of mixture components k in the GMM models were selected by using the *Bayesian information criterion* (BIC, [22]). From the results one can notice that Figures 7a and 7d data sets tends follow more power-law distribution than Figure 7b and Figure 7c data sets. Figure 7d data set shows the best fit to the power-law distribution from the four data sets.

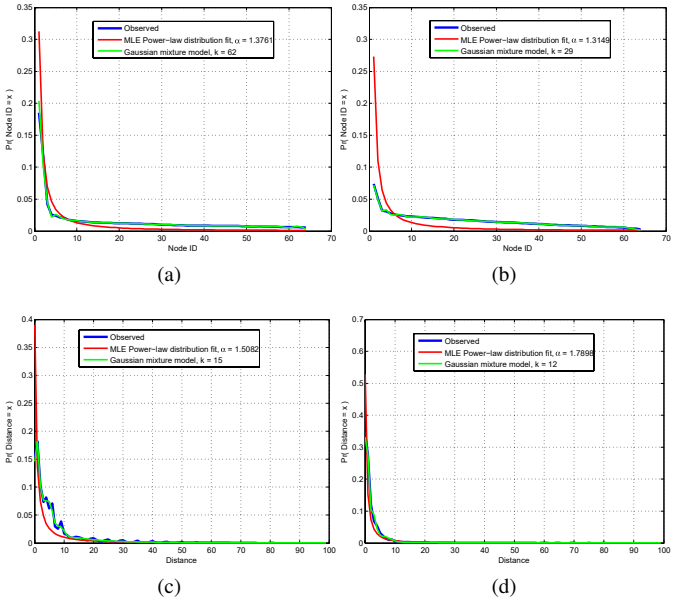


Fig. 7: (a) The MLE power-law distribution and GMM fits for the packet injection percentages of the category 1 applications. (b) The corresponding fits for category 2 applications. (c) interval/frequency distribution fits of the category 1 applications. (d) The corresponding fits for category 2 applications. For (c) and (d) we only model intervals no more than 100 since larger intervals are very rare and thus negligible. The standard error for the MLE estimates of α was $\sigma \approx 10^{-4}$ for all data sets.

We conducted goodness-of-fit tests for the MLE power-law and GMM models by using the KS statistic. The KS statistic is defined as the maximum distance between CDFs of the empirical data and the fitted model:

$$D = \max_{x \geq x_{\min}} |S(x) - P(x)|. \quad (11)$$

This was implemented by the following steps as given in [7]:

1) Calculate the MLE estimate $\hat{\alpha}$ and KS statistic for the empirical data.
2) Sample $\epsilon^{-2}/4$ synthetic data sets, where ϵ is the maximum deviation from the real scaling parameter α.
3) Fit a power-law distribution and calculate the KS statistic for each of the synthetic data sets.
4) Calculate the p-value which is defined as the fraction of the time the KS statistic is larger than for the empirical data.

The power-law hypothesis is rejected if $p \leq 0.1$ [7]. For the data sets we used $\epsilon = 0.01$, which results in 2500

synthetic data sets. For data in Figures 7a, 7b and 7c the p-value was less than 0.1 which means the rejection of power-law distribution under the condition $x_{\min} = 1$. For data in Figure 7d the p-value turned to be very close to the approval of the power-law hypothesis. The GMM models were also evaluated using *two-sample Kolmogorov-Smirnov* tests which also rejected the hypotheses that the data sets are generated by GMM distributions. One can notice in the results that the empirical data has the largest differences with power-law model in the smallest $x_{\min}$ value ranges. This can be one of the main reasons why the hypothesis tests rejected the power-law models. This suggests that the data sets might have power-law behavior with $x_{\min} > 1$. In future work we will continue on trying $x_{\min} > 1$ values for the lower bound parameter.

B. Polynomial Fitting for the Distribution of MD

Polynomial fitting is a form of linear regression in which the relationship between independent variable and dependent variable is modelled as an Mth degree polynomial. Our task is to fit an Mth-order polynomial to a set of N data points by the technique of minimizing an error function. A single variable Mth-order polynomial is defined by:

$$y(x) = w_0 + w_1 x + ... + w_M x^M = \sum_{j=0}^{M} w_j x^j, \qquad (12)$$

where x is the independent variable and the values $\mathbf{w} = (w_0, w_1, ..., w_M)$ are the weights of the polynomial we want to solve.

In practical applications we usually have a data set consisting from N data points, that is $\mathcal{D} = \{(x_1, t_1), (x_2, t_2), ..., (x_N, t_N)\}$ where the t_i-values are the target values. Our goal is to find the set of weights $\mathbf{w}$ such that the error function:

$$E = \frac{1}{2} \sum_{i=1}^{N} \left(y(x_i; \mathbf{w}) - t_i \right)^2, \qquad (13)$$

is minimized. In Equation 13 the term $y(x_i; \mathbf{w})$ denotes the value of the polynomial in Equation 12 with a value x_i and weights $\mathbf{w}$. It is easy to show by differentiation that the Equation 13 is minimized by a solution to a set of linear simultaneous equations [5].

TABLE II: Parameters of fitting results for 3 sets of data.

	UR	Category 1	Category 2
w_0	1.5705e+00	3.0158e+00	1.9009e+00
w_1	3.7587e+00	1.8555e+00	2.9341e+00
w_2	4.4176e-01	6.4167e-01	6.0736e-01
w_3	-2.3091e-01	-1.8430e-01	-2.2837e-01
w_4	2.0786e-02	1.3166e-02	1.8824e-02
w_5	-5.6823e-04	-2.8613e-04	-4.7985e-04

The results are shown in Table II. The real applications demonstrate different fitting output compared with UR traffic. We also notice the quality of the fitting result did not change significantly beyond 5th-order polynomial.

C. Traffic Generation Algorithm and Evaluation

Here we propose a traffic generation algorithm for parallel applications based on the aforementioned models of traces. The algorithm identifies different application categories, and the number of generated packets are determined beforehand. For each packet $P_{<Src,Dst,Cycle>}$, the source node Src and destination node Dst are generated according to the GMM (Figures 7a and 7b) and polynomial fitting (Table II). The distance between two packets, i.e. $Cycle$, is calculated depending on the application category: GMM is applied for category 1 applications (Figure 7c), while MLE power-law model is used for category 2 applications (Figures 7d). The process is repeated until the desired number of packets are fulfilled.

We use the same simulation environment as in Section III-A. Two metrics are evaluated in the experimentation. The average link utilization ALU represents the number of packets transferred between system nodes per cycle. The average network latency ANL is defined as the average number of cycles required for transmitting all packets. We measure the two metrics by using real applications ($Baseline$ in the figure) and synthetic traffic, including the traffic generated by the proposed method ($Proposed$), UR, $Hotspot$, $Rent$ [3] and NED [19]. For $Hotspot$, two random nodes are selected with each 10% (category 2) and 20% (category 1) of total traffic. The Rent's model is configured with $p = 0.75$. For NED the parameter m is set to $1/8$. The number of packets generated for all the synthetic traffic is 50M, where the injection rates of UR, $Hotspot$, $Rent$ and NED are 13% for category 1 applications and 37% for category 2 applications (the average value from Table I, notice the rates are for all the nodes). The results are illustrated in Figure 8.

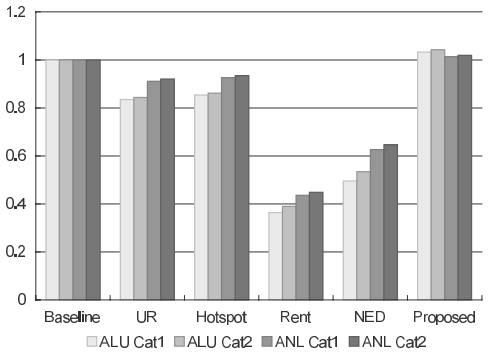


Fig. 8: Normalized ALU and ANL for different traffic models.

The experiments revealed that the proposed algorithm provided the most accurate results in terms of ANL and ALU for both categories of applications (around 1% and 3% differences for ANL and ALU respectively compared with $Baseline$). The differences between $Baseline$ for other algorithms are much larger. The ALUs and ANLs for the four compared algorithms are lower than that in $Baseline$. The main reason is the lower average MD of traffic packets, the average packet MDs of UR and $Hotspot$ are about 10% lower compared with $Baseline$. Even worse, for example in $Rent$, the traffic

is simulated according to the communication probability distribution of the Rent's rule, where the 1-hop communication accounted for over 60%, meaning that over 60% of Src-Dst pairs are adjacent. The 2-hop communication is around 20% in $Rent$. Similar exponential distribution is used in NED, where adjacent and 2-hop communication still has a significant portion. This is hardly the case for real applications. Application traces shown the distribution of distances follows the Gaussian distribution. Higher number of adjacent communication means that the average hop count for the traffic would be lower, in which two consequences can be observed: on the one hand the ANL is reduced due to the shorter communication distance, on the other hand the ALU is reduced due to the fewer active links. Furthermore the metric of packet injection percentage of nodes and packet intervals are mostly missing for the four compared models, where the packets are injected with the same average interval.

V. Conclusion

We analysed the traffic patterns of several parallel applications in this paper. An accurate traffic model for parallel applications was proposed based on the detailed analysis of trace data. The applications were categorized into two groups based on the programming models. Mathematical analysis revealed that the traffic profiles show power-law- and Gaussian-like distributions. We applied power-law model with maximum likelihood estimation, Gaussian mixture model, as well as the polynomial model for fitting the trace data. Experiments were conducted by using a simulator. Results shown that, compared with four other traffic models, the proposed model generates the most accurate network metrics.

References

[1] Badr, M., Jerger, N.: Synfull: Synthetic traffic models capturing cache coherent behaviour. In: Computer Architecture (ISCA), 2014 ACM/IEEE 41st International Symposium on. pp. 109–120 (June 2014)

[2] Bahn, J.H., Bagherzadeh, N.: A generic traffic model for on-chip interconnection networks. Network on Chip Architectures p. 22 (2008)

[3] Bezerra, G.B., Forrest, S., Forrest, M., Davis, A., Zarkesh-Ha, P.: Modeling noc traffic locality and energy consumption with rent's communication probability distribution. In: Proceedings of the 12th ACM/IEEE International Workshop on System Level Interconnect Prediction. pp. 3–8. SLIP '10, ACM, New York, NY, USA (2010)

[4] Bienia, C., Kumar, S., Singh, J.P., Li, K.: The parsec benchmark suite: characterization and architectural implications. In: Proceedings of the 17th international conference on Parallel architectures and compilation techniques. pp. 72–81. PACT '08, ACM, New York, NY, USA (2008)

[5] Bishop, C.M.: Neural Networks for Pattern Recognition. Oxford University Press, Inc., New York, NY, USA (1995)

[6] Bogdan, P., Kas, M., Marculescu, R., Mutlu, O.: Quale: A quantum-leap inspired model for non-stationary analysis of noc traffic in chip multiprocessors. In: Networks-on-Chip (NOCS), 2010 Fourth ACM/IEEE International Symposium on. pp. 241–248 (May 2010)

[7] Clauset, A., Shalizi, C.R., Newman, M.E.J.: Power-law distributions in empirical data. SIAM Rev. 51(4), 661–703 (Nov 2009)

[8] Dally, W.J., Towles, B.: Principles and Practices of Interconnection Networks. Morgan Kaufmann (2003)

[9] Dempster, A.P., Laird, N.M., Rubin, D.B.: Maximum Likelihood from Incomplete Data via the EM Algorithm. Journal of the Royal Statistical Society. Series B (Methodological) 39(1), 1–38 (1977)

[10] Lee, Y., Grover, V., Krashinsky, R., Stephenson, M., Keckler, S., Asanovic, K.: Exploring the design space of spmd divergence management on data-parallel architectures. In: Microarchitecture (MICRO), 2014 47th Annual IEEE/ACM International Symposium on. pp. 101–113 (Dec 2014)

[11] Liu, W., Xu, J., Wu, X., Ye, Y., Wang, X., Zhang, W., Nikdast, M., Wang, Z.: A noc traffic suite based on real applications. In: VLSI (ISVLSI), 2011 IEEE Computer Society Annual Symposium on. pp. 66–71 (July 2011)

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

[13] Martin, M.M., Sorin, D.J., Beckmann, B.M., Marty, M.R., Xu, M., Alameldeen, A.R., Moore, K.E., Hill, M.D., Wood, D.A.: Multifacet's general execution-driven multiprocessor simulator (gems) toolset. Computer Architecture News (September 2005)

[14] Mediatek: Mediatek - true octa-core (January 2015), http://event.mediatek.com/_en_octacore/

[15] Mostaghim, S., Branke, J., Lewis, A., Schmeck, H.: Parallel multi-objective optimization using master-slave model on heterogeneous resources. In: Evolutionary Computation, 2008. CEC 2008. (IEEE World Congress on Computational Intelligence). IEEE Congress on. pp. 1981–1987 (June 2008)

[16] Newman, M.: Power laws, pareto distributions and zipf's law. Contemporary Physics 46(5), 323–351 (2005)

[17] Pekkarinen, E., Lehtonen, L., Salminen, E., Hamalainen, T.: A set of traffic models for network-on-chip benchmarking. In: System on Chip (SoC), 2011 International Symposium on. pp. 78–81 (Oct 2011)

[18] Perelman, E., Polito, M., Bouguet, J.Y., Sampson, J., Calder, B., Dulong, C.: Detecting phases in parallel applications on shared memory architectures. In: Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International. pp. 10 pp.– (April 2006)

[19] Rahmani, A.M., Kamali, I., Lotfi-Kamran, P., Afzali-Kusha, A., Safari, S.: Negative exponential distribution traffic pattern for power/performance analysis of network on chips. In: Proceedings of the 2009 22Nd International Conference on VLSI Design. pp. 157–162. VLSID '09, IEEE Computer Society, Washington, DC, USA (2009)

[20] Rauber, T., Rnger, G.: Parallel Programming - for Multicore and Cluster Systems. Springer (2010)

[21] Rodgers, D.P.: Improvements in multiprocessor system design. SIGARCH Comput. Archit. News 13(3), 225–231 (Jun 1985)

[22] Schwarz, G.: Estimating the dimension of a model. Ann. Statist. 6(2), 461–464 (03 1978)

[23] Soteriou, V., Wang, H., Peh, L.S.: A statistical traffic model for on-chip interconnection networks. In: Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, 2006. MASCOTS 2006. 14th IEEE International Symposium on. pp. 104–116 (Sept 2006)

[24] Tilera: Tile-gx processor family (January 2015), http://www.tilera.com/products/processors/TILE-Gx_Family

[25] Woo, S., Ohara, M., Torrie, E., Singh, J., Gupta, A.: The splash-2 programs: characterization and methodological considerations. In: Computer Architecture, 1995. Proceedings., 22nd Annual International Symposium on. pp. 24–36 (1995)

[26] Xu, T., Liljeberg, P., Plosila, J., Tenhunen, H.: Evaluate and optimize parallel barnes-hut algorithm for emerging many-core architectures. In: High Performance Computing and Simulation (HPCS), 2013 International Conference on. pp. 421–428 (July 2013)

[27] Xu, T., Pahikkala, T., Airola, A., Liljeberg, P., Plosila, J., Salakoski, T., Tenhunen, H.: Implementation and analysis of block dense matrix decomposition on network-on-chips. In: High Performance Computing and Communication 2012 IEEE 9th International Conference on Embedded Software and Systems (HPCC-ICESS), 2012 IEEE 14th International Conference on. pp. 516–523 (June 2012)

[28] Xu, T.C., Liljeberg, P., Plosila, J., Tenhunen, H.: A high-efficiency low-cost heterogeneous 3d network-on-chip design. In: Proceedings of the Fifth International Workshop on Network on Chip Architectures. pp. 37–42. NoCArc '12, ACM, New York, NY, USA (2012), http://doi.acm.org/10.1145/2401716.2401725

[29] Xu, T.C., Liljeberg, P., Tenhunen, H.: An optimized network-on-chip design for data parallel fft. Procedia Engineering 30(0), 311 – 318 (2012), international Conference on Communication Technology and System Design 2011

[30] Xu, T.C., Pohjankukka, J., Nevalainen, P., Leppnen, V., Pahikkala, T.: Parallel applications and on-chip traffic distributions: Observation, implication and modelling. In: Proceedings of the 10th International Conference on Software Engineering and Applications. pp. 443–449 (2015)