Sampling-based Approaches to Accelerate Network-on-Chip Simulation

Wenbo Dai, Natalie Enright Jerger
Edward S. Rogers Sr. Department of Electrical and Computer Engineering, University of Toronto
{daiwenbo, enright}@ece.utoronto.ca

***Abstract*—Architectural complexity continues to grow as we consider the large design space of multiple cores, cache architectures, networks-on-chip (NoC) and memory controllers. Simulators are growing in complexity to reflect these system components. However, many full-system simulators fail to utilize the underlying hardware resources such as multiple cores; consequently, simulation times have grown significantly. Long turnaround times limit the range and depth of design space exploration.**

Communication has emerged as a first class design consideration and has led to significant research into NoCs. NoC is yet another component of the architecture that must be faithfully modeled in simulation. Here, we focus on accelerating NoC simulation through the use of sampling techniques. We propose NoCLabs and NoCPoint, two sampling methodologies utilizing statistical sampling theory and traffic phase behavior, respectively. Experimental results show that NoCLabs and NoCPoint estimate NoC performance with an average error of 7% while achieving one order of magnitude speedup.

I. Introduction

As the number of cores in contemporary processors continues to scale, the criticality of NoC design to overall performance increases accordingly. NoC designers are relying more heavily on full-system simulation to faithfully evaluate their designs. In full-system simulation, the interaction between applications and the NoC is fully exercised; the performance of new designs is accurately evaluated. Although full-system simulation enjoys the benefit of high fidelity, it suffers from prohibitively long turnaround times. Applications in full-system simulation experience up to 100,000× slow down compared to native execution [17], limiting the range and depth of design space exploration.

Sampled full-system simulation [5][13][14][23][25][28][30] is an effective technique to reduce simulation turnaround times for single-, multi-threaded and multiprogrammed applications. In sampled full-system simulation, only a *small* but *representative* portion of the application is simulated in detail, the un-sampled intervals are fast forwarded. Existing work mainly focuses on evaluating micro-architecture designs, and report metrics such as CPI or application run time. To the best of our knowledge, there is no existing work exploring sampling methodologies for NoC simulation.

Two major challenges exist for sampled NoC simulation[1]. First, NoC simulation focuses on different metrics compared to core simulation, so it requires a new sampling methodology. In processor simulation, CPI [5][13][30] or program basic block information [23] is used to select samples. In contrast, in NoC simulations, the designers are concerned with the traffic behavior and network performance metrics including average packet latency. Therefore, the traffic behavior or statistical characteristics of network metrics should be studied and utilized. Second, a new measure is needed to characterize applications and mark the locations of sample intervals. Such a measure must be network architecture independent, so that its marked samples are consistently representative across different network configurations. In single-threaded and multi-programmed sampled simulations [23][30], instruction count is used to measure the application; however, it is unreliable and architecture-dependent for multi-threaded applications: multi-threaded applications manifest execution divergence as the timing of inter-thread synchronization changes with the hardware configuration, causing instruction count to vary significantly.

In this paper, we address the aforementioned challenges and propose sampling methodologies for NoC simulation. Specifically, we introduce two sampling methodologies: *NoCLabs* and *NoCPoint*. They are based on statistical sampling theory and traffic behavior information, respectively. We make the following primary contributions:

- Demonstrate that using total instruction count (TIC) to sample multi-threaded applications results in an unrepresentative sample; instead, user mode instruction count (UMIC) is a NoC architecture-independent measurement for traffic characterization and sample selection;
- Offer insights on parameter selection by exploring the impact of different parameters including the sample size, unit size and the type of vector used to characterize spatial behavior of traffic;
- Provide concrete implementations of NoCLabs and NoCPoint. These techniques estimate NoC performance with an average error of less than 7% while speeding up simulation by one order of magnitude.

II. NoC architecture independent measurement

When sampling an application, characterizing the application and marking the locations of sample intervals are essential steps. Selected samples must be representative of the application even when it is running with different architectural configurations. Therefore, one must use an architecture-independent measurement to mark the locations of samples. For sampling a single-threaded application [23][30], total instruction count (TIC) has been effectively used as it is architecture-independent. However, the TIC of a multi-threaded application is not an architecture-independent measurement; it varies when the architectural configuration changes. Fig. 1 plots the

This research was funded by a gift from Intel. Additional support was provided by the Canadian Foundation for Innovation and the Ontario Research Fund.

[1]In the rest of our paper, NoC simulation refers to the full-system simulation where NoC performance is the focus of the evaluation.

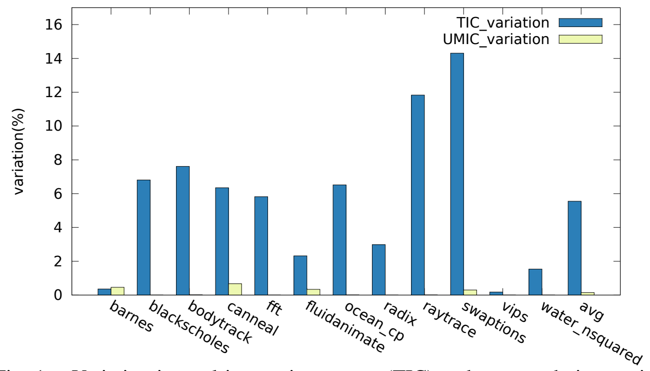


Fig. 1. Variation in total instruction counts (TIC) and user mode instruction counts (UMIC) when running with different network configurations.

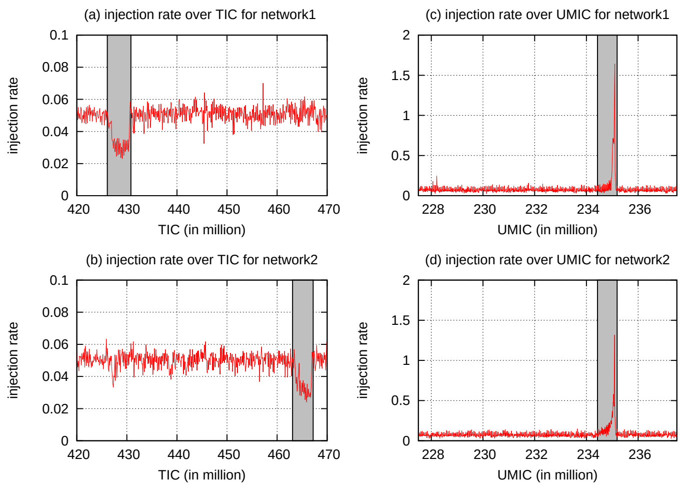


Fig. 2. Injection rates vs total instruction count and user mode instruction count for *canneal* with different network configurations.

variations of TIC for several PARSEC [3] and SPLASH-2 [29] applications when running with two different NoC configurations: $net1$ and $net2$ (described in Sec. IV). The maximal variation is as high as 14.31% ($swaptions$). This raises one major issue for TIC-based sampling: using TIC to specify the start of a sample interval for $net1$ will lead to a different piece of application code being run on $net2$ due to the large variance; as a result, sample intervals with $net2$ are not representative.

Fig. 2(a) and (b) further demonstrate how TIC-based multi-threaded sampling can result in an unrepresentative sample. We plot the traffic injection rates against TIC for *canneal* when running with two different networks. The traffic intervals marked in grey are caused by the same piece of code in the program. However, the TIC corresponding to that traffic is skewed significantly (by 37 million) when a different network is used. Therefore, as a measurement closely tied to architecture, TIC can cause sampled multi-threaded simulation to be inaccurate, and lead to misleading conclusions about performance and design choices. An architecture-independent measurement for multi-threaded applications is needed.

To this end, we propose to use the user mode instruction count (UMIC) instead, which is observed to be stable when different networks are used. Fig. 1 shows a maximal variation of only 0.67% for UMIC (*canneal*). Moreover, looking at the traffic injection rates versus UMIC for *canneal* in Fig. 2(c) and (d), where the grey intervals are the same segment of application code as in Fig. 2(a) and (b), one sample interval will execute the same piece of code across different network architectures.

The rationale behind the stability of UMIC is two-fold. For a multi-threaded application, execution time is spent mainly on two aspects: useful work and scheduling/synchronization [20]. The amount of useful work does not vary a lot with the machine it is running on. Since useful work is largely performed by user mode instructions, it is straightforward that UMIC is stable. However, the timing and duration of scheduling and synchronization can easily change along with architectural factors such as inter-core communication latency. The number of instructions used to perform scheduling/synchronization fluctuates, especially for spin-lock type synchronization. As scheduling and synchronization are handled by the operating system in kernel mode (for instance, pthreads are implemented by Native POSIX Thread Library within linux kernel), its variation will be reflected in the kernel mode instruction count, leaving UMIC unaffected. Given the network architecture independent property of UMIC, we use it in application characterization and sample selection.

User mode instructions have been used to sample multi-processor throughput applications [28]. In throughput applications, user-instruction throughput is proportional to transaction throughput, therefore user-instructions per cycle (U-IPC) is sampled to assess transaction throughput. In contrast, we use user mode instruction count as a network architecture independent measurement, traffic characteristics are examined for intervals of U user-instructions. The fundamental difference is that user mode instruction count is used as a performance metric and the estimation target of the sampling [28], while we use it as a measurement to profile network traffic.

III. Sampling NoC Simulation

Sampled NoC simulation looks at how to select a *minimal* but *representative* portion of the full application traffic to stimulate the network. Network performance metrics measured with the sampled traffic should be an accurate estimate of the true values. We introduce two NoC simulation sampling methodologies: NoCLabs and NoCPoint. They utilize statistical sampling theory and traffic phase behavior information, respectively. Before diving into the details, we introduce several common aspects for both NoCLabs and NoCPoint including the traffic unit size U, population size N, and sample size n.

- Traffic unit size U designates the scale for studying traffic. We divide the full application traffic into non-overlapping units of size U, and then examine network performance metrics such as average packet latency of each unit. U is specified in terms of UMIC.
- Population size N is the total number of U-sized units in the full application traffic ($N = \frac{\text{total UMIC}}{U}$).
- Sample size n is the number of traffic units that will be included in the sample.

Traffic unit size U and sample size n are decided by NoCLabs and NoCPoint users. In the rest of this section, we describe how NoCLabs and NoCPoint select samples and discuss the decision for U and n.

A. NoCLabs: Latency based Statistical Sampling

Inferential statistical sampling [9] estimates a given accumulative property of a population by only measuring a sample. It prescribes a mathematically-sound procedure to select a minimal but representative sample. The minimal sample size n needed is quadratically proportional to the target metric's variation. Since average packet latency is a commonly reported metrics, we base our sample selection on its variation, resulting in Latency based Statistical Sampling for NoC simulation, or NoCLabs. In particular, the minimal sample size n to represent the population depends upon three variables:

- The coefficient of variance of the packet latencies per unit in the population, $\hat{V}$;
- Confidence level $(1 - \alpha)$;
- Confidence interval $\pm\varepsilon$.

$\hat{V} = \frac{\sigma}{\mu}$, where σ and μ are the standard deviation and mean value respectively. Confidence level and interval are to be given by NoCLabs users. Informally, they indicate that one can be $(1 - \alpha)$ confident that the estimated value is within $\pm\varepsilon$ of the true value. The minimal sample size n is defined as:

$$n \geq (\frac{z}{\varepsilon} \cdot \hat{V})^2 \tag{1}$$

where z is the $100[1 - \frac{\alpha}{2}]$ percentile of the standard normal distribution. Such an equation indicates that applications that show larger degrees of variation in their traffic will require a larger sample size. After n is decided, systematic sampling is performed on the population: one traffic unit out of every k units is picked as sample, where $k = \frac{N}{n}$.

If $(1 - \alpha)$, ε and $\hat{V}$ are all known, sample size n is easily calculated from Eqn. 1. However, as it is related to population latencies, $\hat{V}$ can only be obtained after finishing the simulation of the full application. But simulating the full application is exactly the burden that sampling is attempting to avoid. To solve this dilemma, the variation of an initial sample's per unit packet latencies, $\hat{V}_{init}$, is used in place of $\hat{V}$. The steps of performing sampling using NoCLabs are as follows:

1) Specify an accuracy requirement in terms of confidence level $(1 - \alpha)$ and confidence interval $\pm\varepsilon$;
2) Take an initial sample of size n_{init} on the population; simulate and measure the sample;
3) Calculate the initial sample's $\hat{V}_{init}$, and check whether the initial sample's resulting confidence interval ε_{init} meets the specified requirement; ε_{init} is calculated as $\varepsilon_{init} \geq \frac{z \cdot \hat{V}_{init}}{\sqrt{n_{init}}}$.
4) If ε_{init} is within the desired $\pm\varepsilon$, the initial sampling is a success; measured network performance metrics are reported to estimate the true values;
5) If ε_{init} does not meet the requirement, re-sample with a new sample size n_{adjust}. n_{adjust} is calculated by using the initial sample's $\hat{V}_{init}$, the desired $(1 - \alpha)$ and $\pm\varepsilon$ and applying Eqn. (1);
6) Simulate and measure the adjusted sample, and report the network performance.

Among these steps, carefully choosing the initial sample size is vitally important: an n_{init} that is too small can easily result in the first trial failing to meet the accuracy requirement; extra simulation is required. Too large n_{init}, on the other hand, may include more units in the sample than necessary. As a result, potential speedup will be sacrificed. We discuss how to decide n_{init} for PARSEC and SPLASH-2 in Sec. IV.

B. NoCPoint: Exploiting Traffic Phase Behavior

Phase behavior exists in program execution: a set of execution intervals share a greater amount of similarity within themselves compared to other execution intervals. There is a correlation between the program phase and the processor's architectural performance [1][23]. Similarly, application traffic also exhibits phase behavior [2] and there is a correlation between traffic phase and network performance [12]. NoCPoint performs sampling by exploiting an application's traffic phases. There are 3 steps: 1) characterizing and classifying the full application traffic into phases, 2) sampling the full application traffic based on phase information and 3) using the sampled traffic to stimulate the network and measure its performance.

1) Characterizing and classifying traffic: A traffic phase refers to a set of traffic units that behave similarly. To reveal the phases, we first characterize the spatial and temporal traffic behavior. We profile each traffic unit by an injection-ejection rate vector (IERV). In an IERV, each element represents the injection or ejection rate for one node or multiple nodes in the NoC. Two decisions must be made regarding IERVs: 1) how many user mode instructions each IERV covers (traffic unit size U); 2) what is the granularity of injection/ejection rate information in the IERV. In Sec. IV, we discuss the impacts of choosing different Us and types of IERV.

After the behavior of the full application traffic is characterized, one can cluster the traffic units into classes. We use hierarchical clustering [27] to classify traffic: during the clustering, Manhattan distances between IERVs are first calculated, then classes are formed so that intra-class IERVs are closer to each other than inter-class IERVs. That is to say, traffic within the same class manifests similar behavior.

2) Sampling the traffic and measuring the network performance: Once the full application traffic is clustered, we sample the traffic by selecting one unit from each class: each chosen unit has the closest distance to the centroid of that class, and represents the traffic behavior of that class. If there are multiple units with the same minimal distance, the earliest one is chosen. As a result, the sample size n equals to the number of classes formed in the clustering process.

The last step is to measure network performance by simulating the sampled traffic. As the classes formed through clustering may vary in size, the packet latencies of each measured unit are weighted to generate the overall average packet latency. Other network performance metrics, such as latency distribution and network power can also be collected and calculated in a similar manner.

IV. Implementing NoCLabs and NoCPoint

In this section, we describe our implementation of NoCLabs and NocPoint. By carefully selecting the sampling parameters such as traffic unit size U and sample size n, we can perform sampling in a cost-effective way.

A. Simulation infrastructure

We simulate 16 cores using FeS2 [18], a full-system, cycle-accurate x86 simulator. Booksim [11] is used to simulate the NoC. The configurations of FeS2 and Booksim are given in Table I. We run unmodified Linux on top of FeS2 and evaluate PARSEC [3] and SPLASH-2 [29] applications. All applications run with 16 threads, use simsmall input, and are run to completion. Only the regions of interest (ROI) are measured.

$Net0$ is an ideal network with a one-cycle fixed latency between all nodes. It exposes the inherent communication behavior of applications (e.g. spatial behavior). To sample

TABLE I. SIMULATION CONFIGURATIONS

cores	16, P4-like
L1 Cache (I & D)	private, 4-way, 32KB each, 64 Byte Blocks
L2 Cache	private, 8-way, 512KB each, 64 Byte Blocks
Coherence Protocol	MOESI distributed directory
net0	1 cycle fixed latency between all nodes, unlimited network interface buffer, 8 Byte flit
net1	4x4 2D-Mesh, adaptive XY/YX routing, 8 VCs, 8 buffers/VC, 8 Byte flit
net2	4X4 2D-Mesh, dimension-order routing, 4 VCs, 4 buffer/VC, 4 Byte flit
net3	4-ary, 2-fly butterfly, destination tag routing, 4 VCs, 4 buffers/VC, 8 Byte flit
net4	4-ary, 2-fly fat tree, nearest common ancestors routing, 4 VCs, 8 buffer/VC, 8 Byte flit

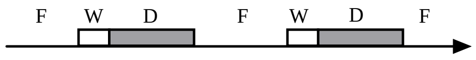


Fig. 3. Simulation state transitions in sampled NoC simulation. Three states are functional simulation (F), detailed timing simulation (D) and warmup (W).

an application, full-system simulation with $net0$ is performed first to obtain the total UMIC and IERVs. This data is used in the steps described in Sec. III. After that, one can run sampled NoC simulation with other network configurations and estimate their performance without needing to simulate the entire application again. $Net1 - 4$ are the candidate networks to be evaluated using NoCLabs and NoCPoint. $Net1$ and $net2$ are direct networks; they are well- and under-provisioned respectively. $Net3$ and $net4$ are indirect networks. These configurations verify the effectiveness of NoCLabs and NoCPoint when applied to different types of networks.

B. Fast forward and warmup

In sampled NoC simulation, only sampled traffic units are simulated in detail by the timing module (D); an unsampled interval between two sampled units is fast forwarded by running faster, functional simulation (F). To keep track of an application's progress, user mode instructions are counted during both D and F. After each D period, the simulator drains all the outstanding packets in the network without measuring them, and then switches to F. As a result, when it switches back from F to D later on, the network is cold, introducing bias into performance measurement. To alleviate the effects of cold start, we warmup the network by adding a warmup (W) period before each D. A warmup period simulates the application in detail but discards all the performance statistics. As a result, the sampled NoC simulation transitions among three simulation states as illustrated in Fig. 3.

A sample of size n contains n timing periods and n corresponding warmup periods; as a result, the number of user mode instructions simulated in detail is:

$$UMIC_{sim} = n \times (W + D) \tag{2}$$

where W and D are the length of warmup and detailed timing period respectively. The value of $UMIC_{sim}$ largely affects how quickly the simulation finishes.

The length of each warmup period W is left to the users to decide. A small W does not warmup the state enough, and an unnecessarily large W hurts the speedup. Dally and Towles [8] provide a heuristic procedure to detect the warmup length, and recognize that event counts on the order of 100 to 1,000 can bring the network to a steady state. We empirically set the length of warmup period to 1000 user mode instructions for both NoCLabs and NoCPoint. This generates ~100 messages

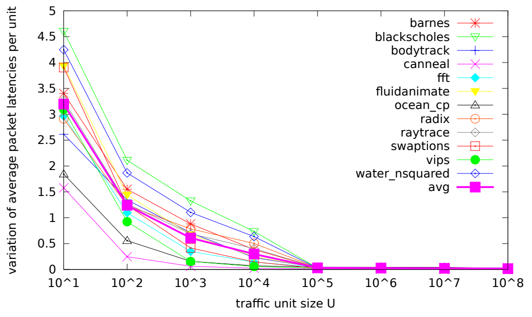


Fig. 4. Coefficient of variance of average packet latency across different traffic unit sizes.

to warmup the network, because the average message per user-instruction is 0.097 across all our applications.

Note that the cache is warmed up by the same 1000 instructions used to warm up the network. Such period of warmup is short for cache and may introduce cold start bias. However, our large traffic unit size U (up to 10 million) mitigates the cold start effect. To completely eliminate the cold start effect, one must keep the cache warmed during fast-forward, but this slows down fast-forward and reduces the simulation speedup.

C. Refining sampling parameters for NoCLabs

NoCLabs users need to determine the initial sample size n_{init}. To optimize n_{init}, we run an analysis on the coefficient of variance of the full application's packet latencies with $net1$. Note that this population information is obtained by running the full-system simulation with $net1$; our sampling method does not require this full-system simulation step. We include the data to illustrate our choice of parameters.

Fig. 4 plots the variations of the full traffic's packet latency, $\hat{V}$ for different traffic unit sizes U. We observe that for each application, its $\hat{V}$ decreases as U grows. As U increases, short term variation is hidden by a larger window. The knee point of the curve exists near U equal to 100,000 for all the applications. Beyond that point, increasing U does not reduce $\hat{V}$ significantly. By considering both Eqn. (1) and (2), this implies using an U larger than the knee point causes $UMIC_{sim}$ to grow proportionally without improving the sampling accuracy.

We use the knee point as a guideline for U and note that the U should not exceed the knee point. However, this does not give a definitive conclusion on what is the best U. We show in Sec. V that a point just to the left of the knee point (e.g. U=10,000) achieves both high accuracy and speedup. Closer examination of the data shows that selecting a U between 10,000 and 100,000 can more accurately locate the knee point. Based on the analysis above, we recommend the traffic unit size U to be 10,000. The average $\hat{V}$ when U=10,000 of all applications is 0.29. If we set an accuracy requirement of 0.99 confidence level with a $\pm 3\%$ confidence interval, an initial sample size n_{init} of 841 can be obtained by using Eqn. (1).

D. Refining sampling parameters for NoCPoint

NoCPoint users also need to answer several questions in regard to parameter selection. They are:

1) What is the traffic unit size U: Since each IERV abstracts the behavior of one traffic unit, using a small U preserves traffic temporal behavior with great detail. However, the benefit comes at a price: 1) because population size $N = \frac{\text{total UMIC}}{U}$, a small U results in a large N for a

TABLE II. INJECTION-EJECTION RATE VECTORS (IERVS)

IERV type	detail level	vector length for an X-node network
per-flow	highest	X^2
per-node	high	$2X$
row-column	medium	$4\sqrt{X}$

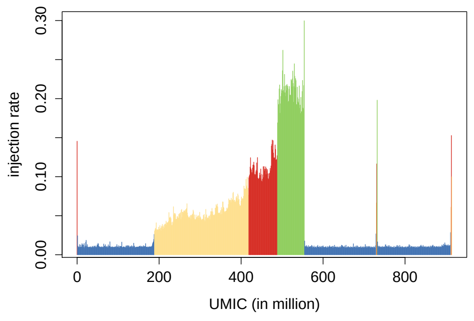
given application, which implies a time-consuming clustering process; 2) after the clustering is finished, a small U may require more than one unit from each class to fully represent the traffic. On the other hand, an extremely large U yields a very small population size. Zhang et al. [32] point out that clustering on a population smaller than 1000 can lead to poor clustering quality. Using these principles as a guideline, we experimentally decide U in Sec. V.

2) How much detail to include in the IERV: To abstract spatial traffic behavior, we consider three different types of IERV. Table II lists their detail levels and the vector lengths needed to characterize an X-node network. In a per-flow IERV, each element represents the injection rate of one source-destination pair, or flow, in the network. It is the finest way to profile traffic spatial behavior. Alternatively, one can use a per-node or row-column IERV. One element in a per-node IERV represents the injection/ejection rate for one node, and each element in a row-column IERV summarizes the injection/ejection rate for all the nodes within the same row/column. As shown in the table, the three candidates require different vector lengths to represent a network of the same size. For our 16-node network, they need 256, 32 and 16 elements respectively; these differences influence the clustering process.

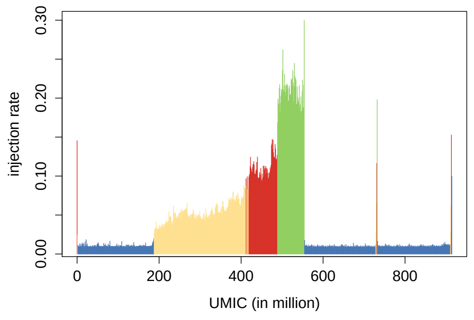
Clustering vectors whose dimension is as high as 256 raises two issues. The first issue is the curse of dimensionality. When dimensionality increases, the data become so sparse that they are no longer statistically significant; consequently, they cannot be clustered effectively. The second is a prohibitively slow clustering process. Clustering the per-flow IERVs is over 100× slower than clustering the row-column IERVs on average. When running on a machine with Intel i5 processor and 4GB memory, the longest clustering time is over 12 hours.

We rule out per-flow IERV since it is too time-consuming, and only consider per-node and row-column IERV. Fig. 5 visually compares the clustering results for $raytrace$ when per-node and row-column IERV are used. In each figure, different colors denote different classes formed. When one compares how the application is divided into different classes (Fig. 5(a) vs Fig. 5(b)), no major difference can be noted; the same results are found for other applications. In Sec. V, we quantitatively compare the performance of per-node and row-column IERV-based sampling.

3) How many classes to form: A stopping rule in hierarchical cluster analysis refers to a procedure for determining the number of classes, K, in a data set. Clustering the data set into more than K classes indicates a solution containing too many clusters; constructing less than K classes, on the other hand, loses information by merging distinct classes. We use the Calinski and Harabasz formal method [4] to guide our traffic clustering process. The basic idea of the Calinski and Harabasz method is to find a number of classes K that can maximize the ratio between the inter-class variance and the intra-class variance: it calculates the ratio of all the Ks ranging from 1 to the population size N, and then finds the maximum ratio and reports its corresponding K. In NoCPoint, after characterizing the traffic by IERVs, we use the Calinski and Harabasz method to analyze the IERVs and calculate the optimal number of classes; then we cluster using the hierarchical clustering algorithm, and sample the traffic following the steps described in Sec. III. Prior work has not used this approach to determine the number of classes in a mathematically-rigorous manner.



(a) per-node IERV



(b) row-column IERV

Fig. 5. Comparison of the clustering result based on per-node and row-column IERV. $raytrace$ is used as an example.

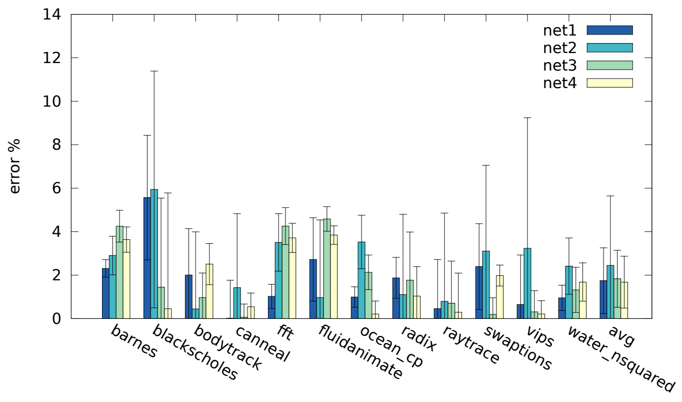


Fig. 6. Error for average packet latency estimation of NoCLabs. The predicted confidence intervals are shown by the error bars.

V. EXPERIMENTAL RESULTS

In this section, we demonstrate how accurately NoCLabs and NoCPoint estimate NoC metrics and compare the two techniques. Insights on parameter selection are provided for both techniques. As network designers are not only interested in average latencies, but also care about the network congestion level and power, we also provide results of latency distribution and power estimation to provide a more comprehensive comparison.

A. NoCLabs

As noted in Sec. IV, we set an accuracy goal of 0.99±3%, use U=10,000 and an initial sample size of 841 to sample NoC

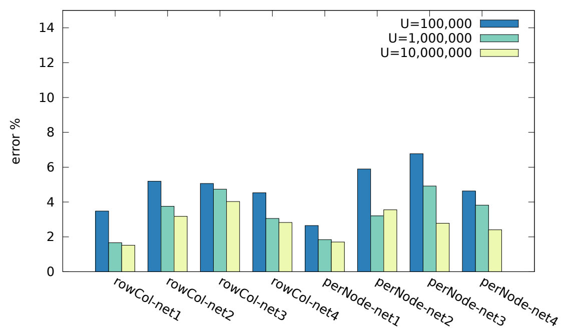


Fig. 7. Error in average packet latency estimation for the four candidate networks when using different traffic unit sizes and IERV types.

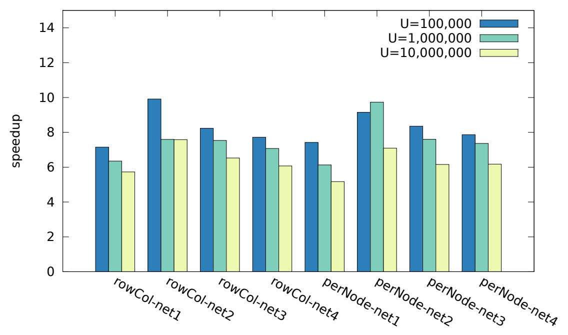


Fig. 8. Simulation speedup achieved by different unit sizes on different sampling configurations.

simulation with all the candidate networks. Fig. 6 illustrates the measured estimation errors for average packet latency. The errors are with respect to the true values measured from full-system simulation. The confidence intervals are also plotted. For all the applications, the results generated by the initial sample provide an accurate estimation without having to perform step 5 described in Sec. III-A. The least accurate estimation comes from $blackscholes$ and $net2$, with an error of 5.94% and a predicted confidence of ±4.45%. From Fig. 4, we see that this is caused by $blackscholes$'s high variation. Our initial sample size n_{init} is selected based on the average $\hat{V}$ of all applications at U=10,000, which is lower than that of $blackscholes$. As a consequence, $blackscholes$'s variation was not fully captured. By sampling using an n_{adjust} of 1878, the error can be reduced to 3.69%. Similarly, resampling other applications using an n_{adjust} can further reduce their estimation errors. On average, we estimate packet latency with an error of 1.75%, 2.45%, 1.83% and 1.68% for the four candidate networks, respectively. Such high accuracy demonstrates that choosing a traffic unit size U just to the left of the knee point in Fig. 4 works well in practice.

B. NoCPoint

The procedure of selecting traffic unit size U for NoCPoint is empirical. As a rule of thumb, we prefer a large U over a small U. The rationale for this is that after the clustering step, each class is represented by only one unit in the sample; a small traffic unit may not stimulate the network long enough to represent the underlying class.

We explore three different unit sizes: 100,000, 1 million and 10 million. They are tested with both row-column and per-node IERV. Fig. 7 shows the errors for different Us on $net1-4$. For a given U, the results vary across different applications; on average, using U of 1 million and 10 million achieves low error percentages: 3.37% for 1 million and 2.75% for 10 million. In contrast, the error increases by $\sim$1.5$\times$ when U=100,000 is used. This validates our preference for large Us.

Although larger U is generally better, an excessively large value reduces the achievable simulation speedup. Fig. 8 shows the speedups when using different unit sizes with different sampling configurations. Wall-clock time is used to compute the speedup with respect to the full-system simulation. The common trend is that, larger unit size results in smaller speedup. For U=10 million, the average speedup is 6.31. If one continues to increase U beyond that, the speedup will keep dropping, as a result, the sampled simulation will not be fast enough to compensate for the accuracy that is sacrificed. Based on our analysis, we recommend U=1 million to achieve a balance between estimation accuracy and simulation speedup.

The other observation is that for the configuration of per-node IERV-$net2$, the highest speedup is achieved by U=1 million, instead of the minimal U of 100,000. By examining the speedups on per-application basis, we find that $vips$ reports a larger speedup (23.1) when U=1 million, compared to 10.1 when U=100,000. Although more instructions are simulated when U=1 million, the last sampled traffic unit occurs significantly earlier than the end of the application. As a result, the simulation can terminate before reaching the end of the application. In contrast, when U=100,000, the sampled traffic includes fewer user mode instructions, but its last sampled unit is near the end of the application; this postpones the termination of the simulation. As a benefit of NoCPoint, if the sampled units do not last until the end of the application, the sample simulation can terminate as soon as the last unit is finished which will increase the speedup opportunity.

Fig. 7 also compares the effects of using row-column and per-node IERV. For estimation accuracy, there is no clear winner. For example, when U=1 million is used, per-node IERV is more accurate than row-column IERV for $net2$ (3.2% error vs 3.75% error), but it is the opposite for $net4$ (3.82% vs 3.06%). Therefore, our quantitative comparison shows that a row-column IERV is as accurate as a per-node IERV in characterizing the traffic behavior. However, clustering on row-column IERVs is 2$\times$ faster than clustering per-node IERVs. We recommend row-column IERV as a cost-effective solution.

C. NoCLabs vs NoCPoint

We compare the estimation accuracy and speedup of NoCLabs and NoCPoint. NoCLabs uses U=10,000 and the initial sample size n=841, NoCPoint uses U=1 million and the row-column IERV.

1) Comparing different networks: NoC designers rely on simulation results to draw conclusions about whether one network outperforms another. Sampled NoC simulation should provide correct conclusions. Fig. 9 plots the absolute packet latency of the four candidate networks collected from full system simulation, NoCLabs and NoCPoint. It verifies that NoCLabs and NoCPoint are drawing correct conclusions when comparing the performance of different networks. For instance, full system simulation suggests that $net1$ has lower packet latency than $net2$ across all applications. Both NoCLabs and NoCPoint agree with full system simulation when comparing the two networks and concludes that $net1$ outperforms $net2$; when comparing the performance of $net3$ and $net4$ based

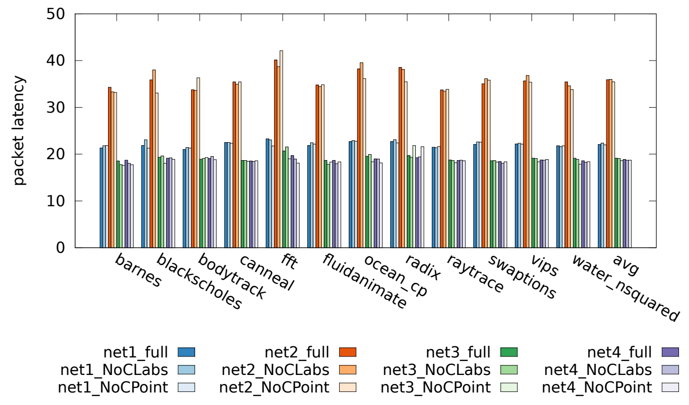


Fig. 9. Packet latency comparison for full system, NoCLabs and NoCPoint.

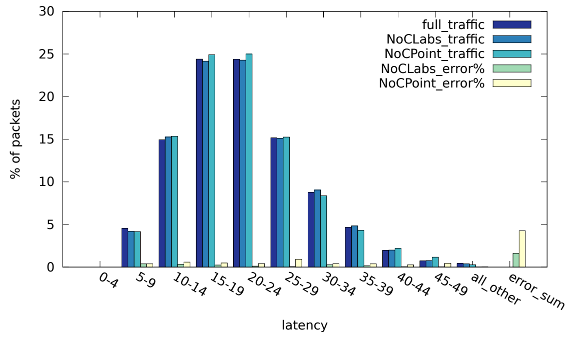


Fig. 10. Packet latency distribution of full vs sample traffic for $raytrace$.

on full system simulation, there is no immediate winner on per-application basis; on average, $net4$ slightly outperforms $net3$. Again, NoCLabs and NoCPoint agree with full system simulation both on per-application basis and average.

2) Estimating NoC congestion: Network designers are also concerned with the network congestion level. To determine if our sampled traffic causes the same level of network congestion as the full traffic does, we quantify the difference in latency distribution between sampled and full traffic. In Fig. 10, we use $raytrace$ as an example. The sampled traffic generated by NoCLabs and NoCPoint in $net1$ are both compared against the original traffic collected for full system simulation. Each group of bars represents the percentage of packets that falls into a range of latency. We compare the two bars representing the same range of latency, and measure the difference. The network congestion estimation error is obtained by summing the differences of all the bars. We find only an 1.56% error for $raytrace$. Fig. 11 compares NoCLabs and NoCPoint's average error of latency distribution estimation for all the applications and networks. Both techniques exhibit low error.

3) Estimating NoC power: Power is another important metric designers should take into consideration. We validate the power estimation accuracy of NoCLabs and NoCPoint by collecting data from both sampled and full-system simulation. We record network activity including buffer read/write, switch and link traverse during the simulation, and fed them into the DSENT [24] power model to calculate dynamic and leakage power. DSENT is configured to use a 45nm process and 1 GHz frequency. The average errors are reported in Fig. 11.

4) Putting all together: In Fig. 11, we plot NoCLabs and NoCPoint's average estimation error of packet latency, latency distribution (i.e. network congestion) and power for $net1-4$. Their speedup is also compared. For NoCLabs, we use error

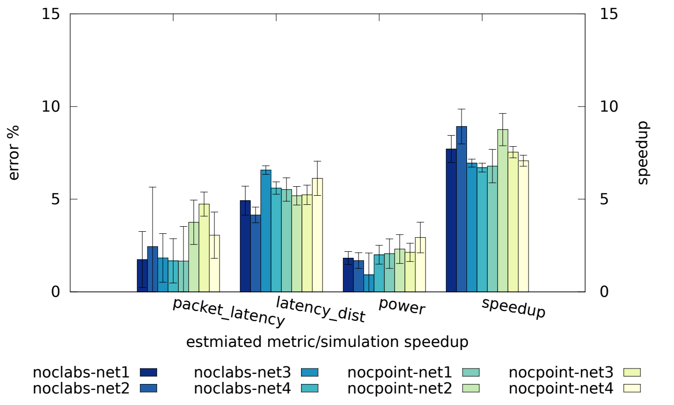


Fig. 11. Comparison between NoCLabs and NoCPoint. Results are averaged across all applications.

bar to demonstrate the predicted confidence interval of latency estimation; for the rest of the data points, error bar implies the coefficient of variance of all the applications. NoCLabs and NoCPoint both show high accuracy in NoC performance estimation. The errors of latency and power estimation are all less than 5%. The latency distribution errors are higher (maximal average error is 6.57%) due to the accumulation of multiple comparison points as explained earlier. For all the performance estimation errors, the coefficients of variance are less than 2 (with the exception of 3.2 for the latency estimation with NoCLabs-$net2$), suggesting a stable accuracy across all the applications. In terms of simulation speed, NoCLabs reports a 8.92× speedup with $net2$; it offers almost one order of magnitude speedup over full-system simulation. Simulations previously lasting for one week now can be finished within a day. All speedups have a variation of less than 1 which implies stable simulation performance for NoCLabs and NoCPoint.

Although NoCLabs and NoCPoint achieve comparable speedup on average (7.57 vs 7.54), they have different strengths and application scenarios. NoCLabs offers lower error than NoCPoint on average across all the candidate networks (1.92% vs 3.3% for latency, 5.31% vs 5.52% for congestion, 1.61% vs 2.36% for power) due to its rigorous mathematical nature. On the other hand, NoCPoint provides a better understanding of application traffic behavior by exploring its phases (e.g. Fig. 5 illustrates how $raytrace$ stresses the network differently over time). Therefore, users can choose between these two approaches according to their needs.

VI. Related Work

SMARTS [30][31] and SimPoint [22][23] apply sampling techniques to the micro-architecture simulation running single-threaded applications. SMARTS utilizes statistical sampling theory to select samples. The sample size is decided by the coefficient of variance of the cycles-per-instruction (CPI) values. In SimPoint, the samples are picked based on the knowledge of program phase behavior, which is detected by analyzing program basic blocks. Both SMARTS and SimPoint sample the dynamic instruction stream, and estimate microarchitectural statistics including IPC, branch prediction and cache miss rate. They were first proposed in the context of single threaded application simulation; they have been extended to multi-programmed workloads [25][28].

Recently, Ardestani and Renau [13] further explore statistical sampling for multi-threaded applications. They propose time-based sampling to handle the problem of execution variation of multi-threaded applications. Fast-forwarding threads using time information rather than instruction count can maintain the correct relative progress among threads. Carlson et al. [5] provide a general-purpose sampling methodology for multi-threaded application. They highlight the importance of maintaining thread interaction and synchronization events during fast forward to achieve correct relative thread progress.

For phase-based multi-threaded application sampling, prior work concentrates on effectively detecting application phases. Instruction counts and traffic count information can be combined to better identify execution phases of multi-threaded applications [32]. Jin et al. [12] examine the characterization and clustering procedure for network traffic. They cluster the traffic by using different types of traffic feature vector. By comparing the intra-class variations of all the cluster results, they also conclude that the row-column vector is efficient. In our paper, we choose row-column IERV over per-node IERV based on the measured results from simulation.

Besides sampling, other approaches can accelerate simulation. FPGA-based acceleration can be used for both full-system [7][15][19] and NoC simulation [10][26]. FPGA-based simulation introduces additional complexity in building hardware models for all the components. Parallelizing simulation is another approach to exploit multi-core hardware for faster simulation [6][16][17][21]. Synchronization can be a bottleneck and some approaches propose to relax synchronization across simulator threads to improve scalability [17][21].

VII. Conclusion

NoC designers often rely on full-system simulation to faithfully evaluate their designs. In this paper, we propose two sampling methodologies to accelerate NoC simulation: NoCLabs and NoCPoint. For NoCLabs, we use statistical sampling to derive a latency-based NoC simulation sampling procedure. It selects a representative sample from the full traffic based on variations in packet latency. By characterizing the traffic using a network architecture-independent measurement and an IERV, NoCPoint captures the intrinsic temporal and spatial behavior of application communication to identify traffic phases. This phase information is used to select simulation points that represent the full traffic. By applying NoCPoint and NoCLabs, only a small portion of the full system simulation is needed to faithfully evaluate various NoC designs. As NoCLabs and NoCPoint users are faced with rich options in terms sampling parameter selection, we provide guidelines for these parameters. We provide concrete implementations for NoCLabs and NoCPoint, and evaluate them against the full system simulation. Evaluation results show that they estimate network performance metrics including average packet latency, latency distribution and power within an error of only 7%. Meanwhile, they speed up simulation by one order of magnitude. With a reduced simulation turnaround time, the range and depth of NoC design space exploration can be enhanced.

Acknowledgment

The authors thank Mike Kishinevsky and Umit Ogras for their invaluable feedback and insight. Additional thanks go to the members of the Enright Jerger research group and the anonymous reviewers for their thoughtful and detailed feedback on this work.

References

[1] M. Annavaram *et al.*, "The fuzzy correlation between code and performance predictability," in *MICRO*, 2004.

[2] M. Badr and N. Enright Jerger, "SynFull: Synthetic traffic models capturing a full range of cache coherent behaviour," in *ISCA*, 2014.

[3] C. Bienia, "Benchmarking modern multiprocessors," Ph.D. dissertation, Princeton University, January 2011.

[4] R. Calinski and J. Harabasz, "A dendrite method for cluster analysis," *Communications in Statistics*, vol. 3, pp. 1–27, 1974.

[5] T. E. Carlson, W. Heirman, and L. Eeckhout, "Sampled simulation of multi-threaded applications," in *ISPASS*, Apr. 2013.

[6] J. Chen, M. Annavaram, and M. Dubois, "SlackSim: a platform for parallel simulations of CMPs on CMPs," *SIGARCH Comput. Archit. News*, vol. 37, no. 2, pp. 20–29, Jul. 2009.

[7] D. Chiou *et al.*, "FPGA-accelerated simulation technologies (FAST): Fast, full-system, cycle-accurate simulators," in *MICRO*, 2007.

[8] W. Dally and B. Towles, *Principles and Practices of Interconnection Networks*. San Francisco: Morgan Kaufmann Publishers Inc., 2003.

[9] J. Devore, *Probability and Statistics for Engineering and the Sciences, Enhanced Review Edition*. Brooks/Cole, Cengage Learning, 2008.

[10] N. Genko *et al.*, "A complete network-on-chip emulation framework," in *DATE*, 2005.

[11] N. Jiang *et al.*, "A detailed and flexible cycle-accurate network-on-chip simulator," in *ISPASS*, 2013.

[12] Y. Jin, E. J. Kim, and T. M. Pinkston, "Communication-aware globally-coordinated on-chip networks," *IEEE TPDS*, vol. 23, no. 2, Feb. 2012.

[13] E. K. Ardestani and J. Renau, "ESESC: A fast multicore simulator using time-based sampling," in *HPCA*, 2013.

[14] J. Kihm and D. Connors, "Statistical simulation of multithreaded architectures," in *MASCOTS*, 2005.

[15] A. Krasnov *et al.*, "Ramp blue: a message-passing manycore system in FPGAs," in *FPL*, 2007.

[16] M. Lis *et al.*, "Scalable, accurate multicore simulation in the 1000-core era," in *ISPASS*, 2011.

[17] J. Miller *et al.*, "Graphite: A distributed parallel simulator for multicores," in *HPCA*, Jan. 2010, pp. 1 –12.

[18] N. Neelakantam *et al.*, "Fes2: A full-system execution-driven simulator for x86," in *ASPLOS*, 2008.

[19] M. Pellauer *et al.*, "HAsim: FPGA-based high-detail multicore simulation using time-division multiplexing," in *HPCA*, 2011.

[20] M. Roth *et al.*, "Deconstructing the overhead in parallel applications," in *IISWC*, 2012, pp. 59–68.

[21] D. Sanchez and C. Kozyrakis, "ZSim: fast and accurate microarchitectural simulation of thousand-core systems," in *ISCA*, 2013.

[22] T. Sherwood, E. Perelman, and B. Calder, "Basic block distribution analysis to find periodic behavior and simulation points in applications," in *PACT*, 2001.

[23] T. Sherwood *et al.*, "Automatically characterizing large scale program behavior," in *ASPLOS*, 2002.

[24] C. Sun *et al.*, "DSENT - a tool connecting emerging photonics with electronics for opto-electronic networks-on-chip modeling," in *NOCS*, 2012.

[25] M. Van Biesbrouck, T. Sherwood, and B. Calder, "A co-phase matrix to guide simultaneous multithreading simulation," in *ISPASS*, 2004.

[26] D. Wang, N. Enright Jerger, and J. Steffan, "DART: A programmable architecture for noc simulation on FPGAs," in *NOCS*, 2011.

[27] J. H. Ward, "Hierarchical grouping to optimize an objective function," *Journal of the American Statistical Association*, vol. 58, no. 301, pp. 236–244, 1963.

[28] T. F. Wenisch *et al.*, "SimFlex: Statistical sampling of computer system simulation," *IEEE Micro*, vol. 26, no. 4, pp. 18–31, Jul. 2006.

[29] S. Woo *et al.*, "The SPLASH-2 programs: characterization and methodological considerations," in *ISCA*, June 1995, pp. 24–36.

[30] R. E. Wunderlich *et al.*, "SMARTS: accelerating microarchitecture simulation via rigorous statistical sampling," in *ISCA*, 2003, pp. 84–97.

[31] ——, "Statistical sampling of microarchitecture simulation," *ACM Trans. Model. Comput. Simul.*, vol. 16, no. 3, pp. 197–224, Jul. 2006.

[32] Y. Zhang *et al.*, "Analyzing the impact of on-chip network traffic on program phases for CMPs," in *ISPASS*, 2009.