# Effects of Nondeterminism in Hardware and Software Simulation with Thread Mapping

Giordano Salvador
University of Pennsylvania
Philadelphia, Pennsylvania
gsalv@seas.upenn.edu

Siddharth Nilakantan, Baris Taskin, Mark Hempstead
Drexel University
Philadelphia, Pennsylvania
sn446@drexel.edu, {taskin, mhempstead}@coe.drexel.edu

Ankit More
Intel Corporation
Portland, Oregon
ankitmore@gmail.com

***Abstract*—In this paper, we explore the simulation performance trade-off under the lens of Monte Carlo design space exploration for multi-threaded programs and thread mapping. The vehicle used for this exploration will be a recent study, whose novel Google PageRank-based thread mapping approach is compared to hundreds of random mappings, as well as a Round-Robin-based thread mapping approach proposed in this paper used in similar comparisons. The modern simulator landscape presents a choice between cycle-accurate but slow, and fast but inaccurate program simulation. We find that the use of a fast, inaccurate multi-threaded simulator, such as Sniper 5.3, suffers from large nondeterminism in the reported performance of the program. We perform cycle-accurate simulation which demonstrates that the static thread mapping approach does provide benefits in reaching near-optimal design points. Furthermore, the runtime of static thread mapping is significantly reduced using a cycle-accurate simulator compared to the full Monte Carlo exploration of mapping design points.**

## I. Introduction

As more processing cores become available in chip-multiprocessors (CMP), software programs need to utilize the parallelism inherent in algorithms in order to increase runtime performance. The cores on these CMPs are connected with an interconnect fabric called a network-on-chip (NoC). Tilera's TILE64 [1], a 64-core 2D mesh NoC, and Intel's 80-core Polaris [10] are examples of many-core NoC CMPs available today. On the software end, algorithms are split into individual sequences of instructions which may operate in parallel, called threads. These threads may be independent, or may synchronize and communicate data between one another. If these threads are allowed to run on their own hardware resource, core/processing element (PE), then the parallelism provided by the CMP hardware may be exploited.

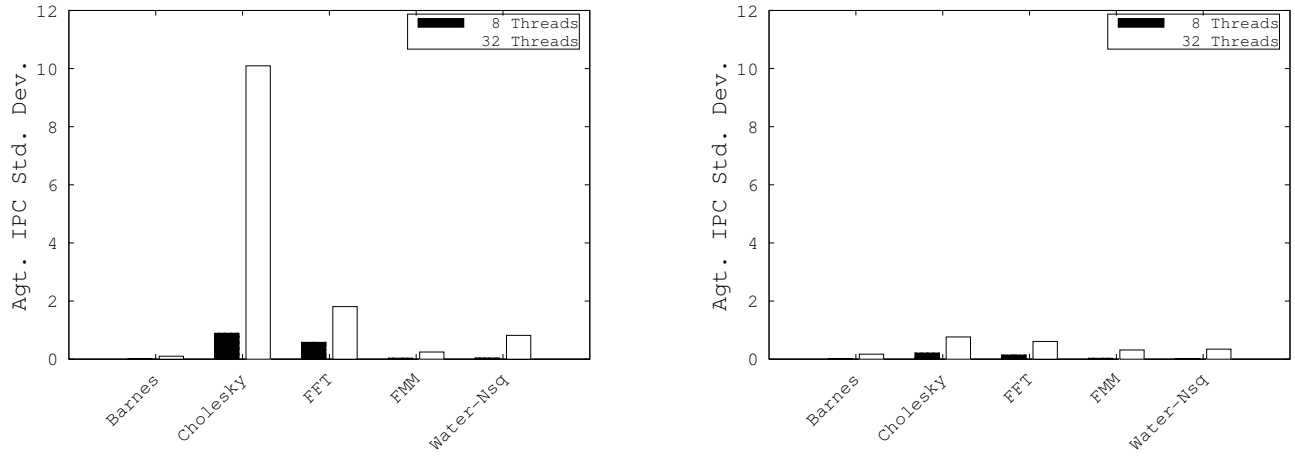
One question raised by introducing software thread and hardware PE resources is how to best allocate these resources to one another. As implicated before, a thread may be assigned a hardware core, or otherwise share the resource with other threads. The act of assigning threads to cores is thread mapping, a primary component of study in this paper. Specifically, the focus is static mapping, where the thread-to-core assignments are fixed at the beginning of the execution of a program. In this paper, we study the effects of thread mapping on program performance, the trade-off of speed over accuracy in hardware simulators, and the interface between both of these subjects where the object of simulation exhibits extreme sensitivity in performance to nondeterminism. In addition to exploring nondeterminism in simulation, this paper's second novelty is the augmentation of a recent mapping bag of algorithms approach [7] with the use of the Round-Robin technique inspired by OS scheduling, for efficient search space exploration over random searching. The salient components of our analysis are: *A*.) The process of thread mapping and *B*.) The hardware simulators. These two components, which are not novelties of this paper but vehicles of analysis, are introduced next.

### A. Relevance of Thread Mapping

First, we introduce techniques in thread mapping. Examples of thread mapping in academia attempt to exploit various hardware features for determining program behavior. The work in [9] minimizes the critical path of parallel programs by using hardware performance counters to determine local memory access. In [5], the authors use memory access traces from the Simics simulator to produce thread affinity matrices, which are then used for a bipartite set matching algorithm, also known as Edmond's matching algorithm. Lastly, [4] shows how a helper software thread which analyzes the frequency of memory access of other threads can assist in dynamically migrating threads to form a mapping via an Integer Linear Program.

Still to be answered is what an optimal thread-to-core mapping is, and what hardware/software artifacts affect the runtime of a program. Unfortunately, thread mapping is a combinatorial problem which is a function of the number of cores available, the number of threads which the core can support concurrently, and the number of threads the program or programs request for execution. Execution of these programs with a given thread mapping on real CMPs would illustrates what the runtime for that configuration would be, assuming access to a many-core and compatible software for the architecture is available. More practically, the solution is to run programs in a software environment which simulates the effects of a NoC CMP as the execution progresses.

Recent previous work in [7] tackles this issue for the case of static thread mappings. Binary instrumented information from a running program is used to create affinity matrices of bytes written/read and shared byte addresses of the workload, which are then used with an adapted Google PageRank algorithm. The rankings produced are then used to determine placement on a two-dimensional mesh NoC CMP, and simulated in a multi-threaded hardware tool, Sniper. The technique is used for this work as a recent example of a high-performing algorithm, illustrative of the power of static thread mapping. The other thread mapping algorithm studied is random mapping. Specifically, we use random mapping to investigate the fidelity of the resulting performance predictions stemming from hardware simulators in mapping scenarios.




(a) Fast [7]

(b) Cycle-accurate

Fig. 1: Splash-2 Benchmark Performance Opportunity Derived from Monte Carlo Exploration

### B. Relevance of Hardware Simulators

Next, we introduce the state-of-the-art in hardware simulation, since several CMP-enabled simulators exist today. Gem5 is a modular simulator capable of full-system simulation, cycle-accurate core and cache models, and simple NoC architectures [2]. ZSim is a CMP simulator with simple NoC model which uses dynamically produced traces to speed up simulation to scale to up to a thousand cores [8]. Sniper is a multi-threaded simulator which uses simple core models, system-level threads, and region-of-interest tags to significantly speed up simulation time for CMP NoCs [3].

Simulating multi-threaded programs with large datasets on an extremely detailed cycle-accurate simulator, such as Gem5, could take hours for a single simulation. The issue is that the number of potential mappings explodes due to the combinatorial nature. In other words, thread mapping optimization is NP-hard. Another naive but useful practice in the face of NP-hard optimization problems is random Monte Carlo search explorations. Yet, it is undesirable for these simulations to take many hours because there is a large sample space of mappings to explore. Therefore, faster multi-threaded simulators, like Sniper, fit the niche demand for many-core, many-thread simulation at the expense of accuracy.

How does trading accuracy for simulation speed affect the reliability of an empirical hardware study? This paper aims to caution readers of nondeterminism in program simulations, while evaluating and refining existing techniques for static thread mapping which use other heuristics other than random sampling to accomplish the task. We expect this work to be of importance to the userbase of the popular simulator programs such as Sniper, in addition to be of interest in software/hardware co-design community in advancing the performance of static thread mapping.

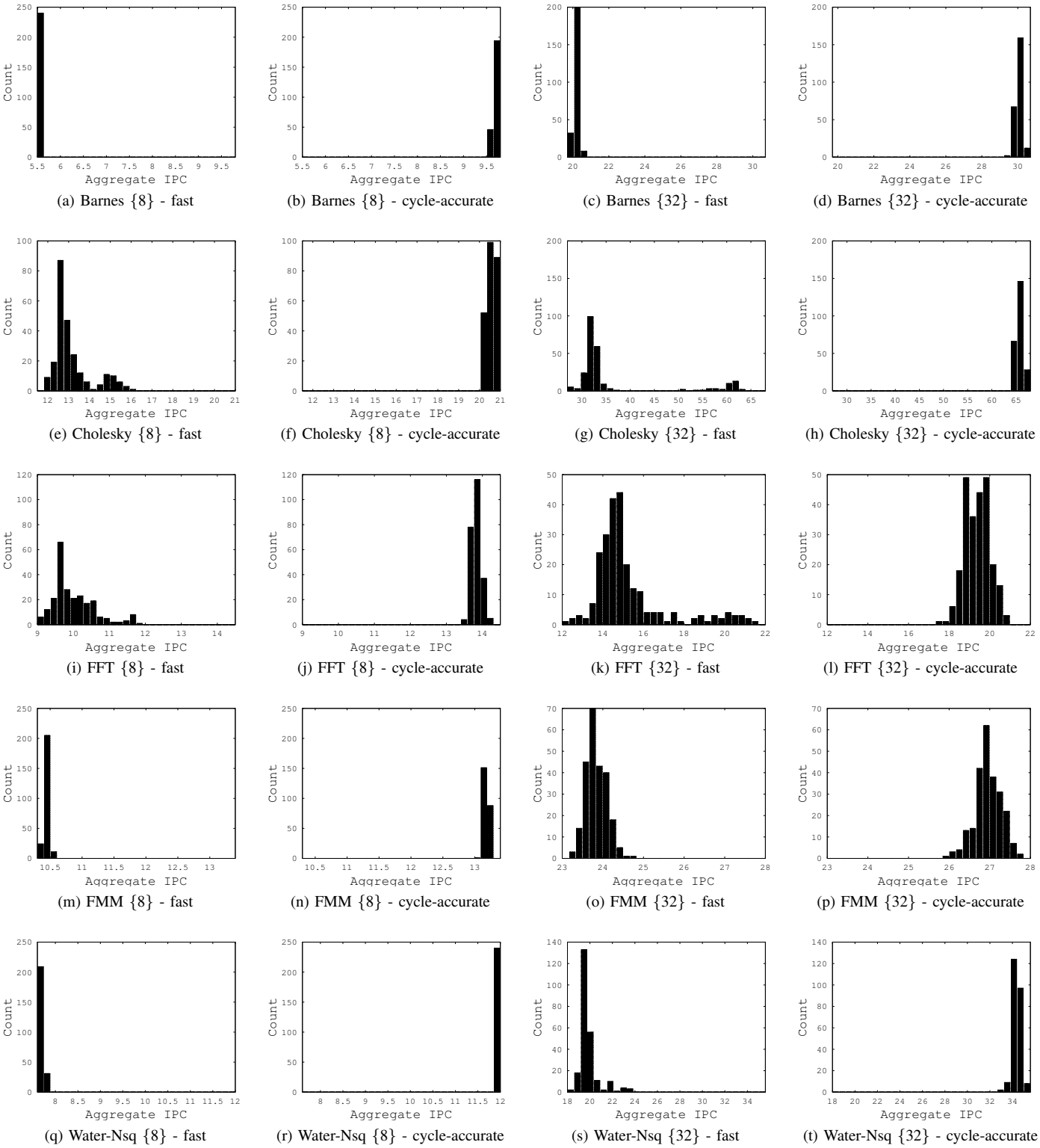
The remaining sections of this paper are organized as follows: Section II introduces a Monte Carlo technique for thread mapping; Section III describes simulation nondeterminism's effect on design space exploration; Section IV presents a reevaluation of a novel technique under a cycle-accurate environment; and Section V concludes the paper.

## II. Monte Carlo Exploration of Thread Mapping Performance

Design space exploration for cache sizes, NoC router buffer sizes, thread mapping, and other architectural resources seeks to find optimal, or near-optimal configurations for best performance under specific constraints. In the case of thread mapping, the exploration of the thread mapping space is limited by the constraint on simulation/exploration time. The increasing need of many-core network-on-chips subsequently increases the combinatorial complexity of choosing the optimal performing thread-to-core mapping.

The relevant work in [7] is summarized here for completeness of this paper. In [7], a Monte Carlo exploration of 8-thread (32) mappings on a 3x3 (6x6) NoC was performed for the multi-threaded Splash2 benchmarks. In the case of the 3x3 NoC for which 8 threads were to be mapped, there are $8! \cdot 9$ permutations of mappings, nearly half a million for just this small configuration. According to [7], merely 300 design points per day were able to be explored using the fast Sniper 5.3 simulation framework. Clearly, it would be extraordinarily impractical to perform an exhaustive simulation of half a million mappings, even at the small end of the many-core NoC spectrum.

A Monte Carlo search thus allows a random sampling to uniformly explore the design space, forming an incomplete, but sufficient image for what can be expected of benchmark performance. For static thread mapping on a NoC whose PEs support one thread at a time, random mappings are generated such that each thread ID is assigned to a PE ID, without allowing overlap - an injective mapping. At simulation time, a thread ID's affinity is set to the core ID defined in the mapping when the thread is created through the thread-management library function. Results in [7] from the "fast" (Sniper 5.3) simulations show a Monte Carlo exploration of the effects on benchmark performance, as measured in aggregate IPC, and have allotted 240 random mapping simulations per benchmark configuration. The histogram distributions of aggregate IPC discovered in their search can be seen in Figure 2, namely, Figures 2a, 2c, 2e, 2g, 2i, 2k, 2m, 2o, 2q, and 2s [7]. Small




(a) Barnes {8} - fast (b) Barnes {8} - cycle-accurate (c) Barnes {32} - fast (d) Barnes {32} - cycle-accurate

(e) Cholesky {8} - fast (f) Cholesky {8} - cycle-accurate (g) Cholesky {32} - fast (h) Cholesky {32} - cycle-accurate

(i) FFT {8} - fast (j) FFT {8} - cycle-accurate (k) FFT {32} - fast (l) FFT {32} - cycle-accurate

(m) FMM {8} - fast (n) FMM {8} - cycle-accurate (o) FMM {32} - fast (p) FMM {32} - cycle-accurate

(q) Water-Nsq {8} - fast (r) Water-Nsq {8} - cycle-accurate (s) Water-Nsq {32} - fast (t) Water-Nsq {32} - cycle-accurate

Fig. 2: Splash-2 Monte Carlo Performance Exploration (8 & 32 Threads) with fast [7] and cycle-accurate simulators

benchmark input sizes as defined by the Sniper 5.3 framework were used in the simulation, and the configurations can be found in Table I.

In the case of Figure 2e, for example, approximately 20 random mappings produced achieved 12.25 to 12.50 aggregate IPC, approximated 90 mappings produced achieved 12.50 to 12.75 aggregate IPC, and so on. The salient features visualized in the IPC distributions are the ranges of IPCs shown. Most of the distributions are roughly Gaussian in shape, with standard deviations falling within tight bands around the means. From

TABLE I: Monte Carlo Simulation Configuration Parameters

| Parameter | 8 threads | 32 threads |
|---|---|---|
| Simulator | Sniper 5.3 | Sniper 5.3 |
| Number of Cores | 9 | 36 |
| NoC Type | 2D mesh | 2D mesh |
| Dimensions | 3x3 | 6x6 |
| Private Cache | L1 & L2 | L1 & L2 |
| Shared Cache | NUCA L3 | NUCA L3 |
| LLC Cache Size | 8MB | 32MB |
| Coherence Protocol | MESI CMP Directory | MESI CMP directory |
| Configuration Templates | gainestown, nuca-cache | gainestown, nuca-cache |
| Benchmark Suite | Splash2 | Splash2 |
| Benchmarks | barnes, cholesky, fft, fmm, water-nsquared | barnes, cholesky, fft, fmm, water-nsquared |
| Thread Mapping Type | static | static |
| Mapping Scheme | random | random |
| Max Threads Per Core | 1 | 1 |
| Number of Iterations | 240 / benchmark | 240 / benchmark |

these aggregate IPC distributions, the opportunity for achieving better benchmark performance purely by changing the thread-to-core mappings is plotted in Figure 1a. These results are telling, since, for some benchmarks, the standard deviation of the aggregate IPCs of the random mappings can be upwards of 10. For these benchmarks, the gains indicated by the high standard deviations derive from the non-Gaussian behavior in distributions. Figures such as Figure 2e have secondary Gaussian modes which achieve higher performance, increasing the aggregate IPC standard deviations. We see that there is much room for improvement over the mean in these cases, indicating that it is worth trying to methodically map threads rather than randomly choosing a mapping if the effect of mapping were to be insignificant.

## III. Nondeterminism in Fast Multi-threaded Simulation

Issues of nondeterminism arise from using fast but inaccurate multi-threaded simulators. In order to achieve the simulation speed boosts, simplifications in the hardware models and software emulation are made. In the case of the Sniper 5.3 framework used in the study [7], the CPU core model is stripped of its branch predictors and instruction cache. Due to these aggressive speed optimizations, the simulation of the core is comparable to that of a one-IPC model, but introduces inaccuracy in the cycle timing of the target program. More importantly, the Sniper 5.3 framework uses a lax model of thread synchronization for multi-threaded programs. Specifically, multi-thread synchronization is not modeled at all [3], causing extreme variability in the reported runtime of program. This nondeterministic behavior introduces significant artifacts in an exploration of benchmark IPC, such as the one shown in Figure 2.

In Figure 3, aggregate IPC histograms for a single mapping design point show the extreme nondeterminism of inaccurate simulation. Both the histograms, "fast" and "cycle-accurate", show the aggregate IPCs reported by simulating the *same* thread mapping for the Cholesky Splash2 benchmark repeatedly. The bulk of the NoC configuration used is identical to those in Table I. The difference between "fast" and "cycle-accurate" is that the latter's configuration adds the "rob.cfg" cycle-accurate parameters offered in Sniper 6.0 to the simulation. Notably, the single thread mapping used for all 480 of these reiterated simulations was chosen because it had reported an aggregate IPC at the high end of the IPC distribution shown in Figure 2e. Also, in order to eliminate the impact of (minor) non-determinism in the BI process on the mapping quality, the

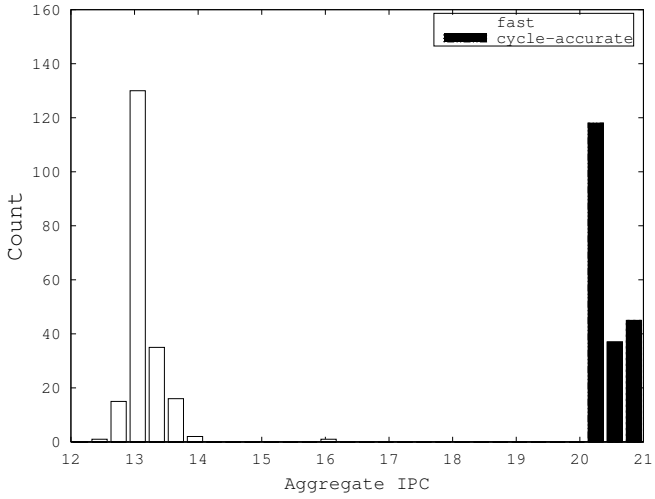



Fig. 3: Splash-2 Benchmark Monte Carlo Exploration of Single Design Point - Cholesky 8-thread 3x3 NoC

*same* BI trace is used for a particular design point, *for all* such design points. However, the repeated simulation for this one mapping using the fast model in Sniper 5.3 shows that the good performance degenerates in the majority of the cases; its high performing iterations are merely outliers. This means that none of the results from the study in [7] are reliable since the fast simulation is nondeterministic *and* inconsistent. In contrast, the "cycle-accurate" histogram shows that by using the cycle accurate model, most of the nondeterminism is tamed, reporting more consistent aggregate IPCs within a small range of less than 1 aggregate IPC. Therefore, any simulations in a framework exhibiting this much sensitivity to nondeterminism due to inaccurate models should be reevaluated under an empirically reproducible environment.

With nondeterminism mitigated, we reevaluate the efficacy of static mapping to see if there is room for improved performance. Sniper 6.0 is used as the simulation framework because of its aforementioned cycle-accurate parameters, which both mitigates the nondeterministic workload performance and provides confidence in properly modeling core hardware components. Sniper is used in favor of other simulator frameworks, such as Gem5, since it still remains relatively agile with respect to its simulation speed, and enables a closer comparison to the Sniper 5.3 framework used in the other study. Similar to the distributions derived from the "fast" configurations shown in Figure 2, configurations in Figures 2b, 2d, 2f, 2h, 2j, 2l, 2n, 2p, 2r, and 2t show the cycle-accurate configured Monte Carlo exploration of each Splash2 benchmark. 240 iterations per configuration were simulated as described in Section II.

From the cycle-accurate distributions in Figure 2, new conclusions can be drawn regarding static mapping. An obvious effect of the cycle-accurate simulation is that the means and shapes of the distributions have changed. More poignantly, the dynamic range of the aggregate IPCs has been drastically reduced. A consequence of this is that there is a smaller difference between the average performing random mapping and the best performing one. If the gains to be made from any possible static mapping are too small, then a random mapping such as those found by this Monte Carlo search will be adequate. Figure 1b highlights this point by showing

TABLE II: Mapping Schemes Simulated on the 2D mesh NoC

| Mapping Scheme | Description |
|---|---|
| Descending Priority Radial Outwards (DPRO) | Map threads to empty core closest to center, highest rankings 1st |
| Ascending Priority Radial Outwards (APRO) | Map threads to empty core closest to center, lowest rankings 1st |
| Descending Priority Radial Inwards (DPRI) | Map threads to empty core farthest from center, highest rankings 1st |
| Ascending Priority Radial Inwards (APRI) | Map threads to empty core farthest from center, lowest rankings 1st |

the standard deviation of the aggregate IPC for each design point under cycle-accurate simulation. However, in certain circumstances, every bit of extra performance matters, since programs can inject trillions of instructions to the processing cores.

## IV. Evaluation of PageRank Static Mapping

Static mapping can be accomplished in more intelligent ways than just randomly choosing which threads are pinned to a core on the NoC. By using communication information between threads during the execution of a multi-threaded program, data flow patterns can detected to inform a more optimal mapping, thus improving performance.

The study done in [7] explored the utility of binary instrumentation to produce virtual memory address traces with thread synchronization information, and applied Google's PageRank to create a novel static mapping approach. The binary instrumentation tool is called Sigil [6], a publicly-available utility built on top of the Valgrind/Callgrind framework. Through running multi-threaded Splash2 benchmarks, virtual address read and writes are captured, abstracting away any architecture specific behavior. Moreover, the traces also contain events signaled by Pthread thread creation and synchronization functions, such as mutex locks. The thread information is used to identify computation (writes) and communication (reads) events to a particular set of virtual addresses. This also allow producer-consumer relationships to be captured across threads as they share data with one another. Then, the read and write data from the traces are aggregated into symmetric byte affinity matrices between all pairs of threads. The OCaml program which analyzes the traces then runs Google's PageRank algorithm on the matrix, outputting an ordered ranking of the threads which communicate the most globally. The idea is then to place heavy communicating nodes close to each other in the center or far apart from each other on the outskirts of the two-dimensional mesh NoC. Since on a NUCA cache system data writes and reads will be stored on the corresponding local cache node corresponding to a thread's assigned core, pinning threads to different cores will indirectly increase or decrease the distance between communicating cache nodes during coherence. Note that while a distributed cache directory at another NoC tile can be the home node of a particular cache line, the cached line data is still stored locally in the LLC per specification of the coherence protocol. Lastly, the threads are pinned to cores according to simple schemes described in Table II as found in the study [7].

To evaluate the novel PageRank static mapping method, the best performing PageRank mapping (among the multiple placement schemes) was plotted versus the worst, mean, and best random mapping found by the Monte Carlo exploration described in Section II. The same configuration found in Table I was used for this comparison. The aggregate IPC number reported for each design point was normalized to the best performing (highest aggregate IPC) random mapping to show the approximate optimality of each mapping. Figure 4a shows the mapping performance comparison of the PageRank method and the Monte Carlo exploration.

TABLE III: PageRank and Round-Robin Mapping Performance Versus Monte Carlo Worst, Average, & Best Case Performance From Agt. IPC

(a) Normalized to MC. max - fast

| Benchmark | MC min. | MC avg. | MC max. | PR max. | RR |
|---|---|---|---|---|---|
| Barnes {8} | 0.99 | 0.99 | 1.00 | 1.00 | 0.99 |
| Cholesky {8} | 0.75 | 0.82 | 1.00 | 1.01 | 0.83 |
| FFT {8} | 0.76 | 0.85 | 1.00 | 0.94 | 0.94 |
| FMM {8} | 0.98 | 0.99 | 1.00 | 1.00 | 0.99 |
| Water-Nsq {8} | 0.98 | 0.99 | 1.00 | 1.00 | 0.99 |
| Barnes {32} | 0.96 | 0.98 | 1.00 | 0.98 | 0.98 |
| Cholesky {32} | 0.44 | 0.58 | 1.00 | 1.00 | 0.91 |
| FFT {32} | 0.57 | 0.71 | 1.00 | 0.98 | 0.90 |
| FMM {32} | 0.94 | 0.96 | 1.00 | 0.98 | 0.96 |
| Water-Nsq {32} | 0.78 | 0.84 | 1.00 | 0.87 | 1.00 |

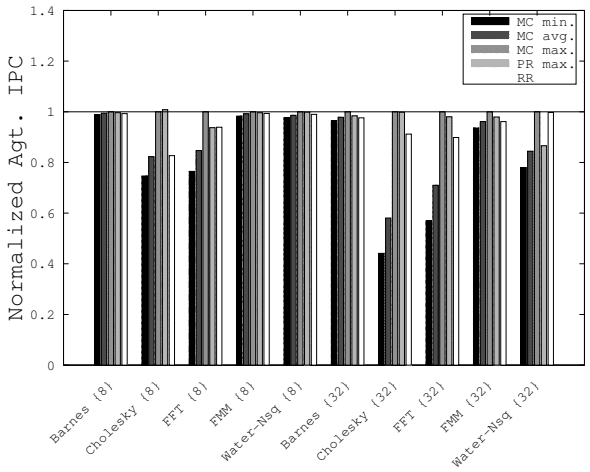
(b) Normalized to MC. max - cycle-accurate

| Benchmark | MC min. | MC avg. | MC max. | PR max. | RR |
|---|---|---|---|---|---|
| Barnes {8} | 0.99 | 0.99 | 1.00 | 1.00 | 1.00 |
| Cholesky {8} | 0.97 | 0.98 | 1.00 | 1.00 | 1.00 |
| FFT {8} | 0.95 | 0.97 | 1.00 | 0.99 | 0.97 |
| FMM {8} | 0.99 | 0.99 | 1.00 | 1.00 | 1.00 |
| Water-Nsq {8} | 0.99 | 1.00 | 1.00 | 1.00 | 1.00 |
| Barnes {32} | 0.97 | 0.99 | 1.00 | 0.99 | 0.99 |
| Cholesky {32} | 0.95 | 0.98 | 1.00 | 0.99 | 0.97 |
| FFT {32} | 0.84 | 0.92 | 1.00 | 0.98 | 0.95 |
| FMM {32} | 0.93 | 0.97 | 1.00 | 0.97 | 0.99 |
| Water-Nsq {32} | 0.93 | 0.97 | 1.00 | 0.98 | 0.97 |

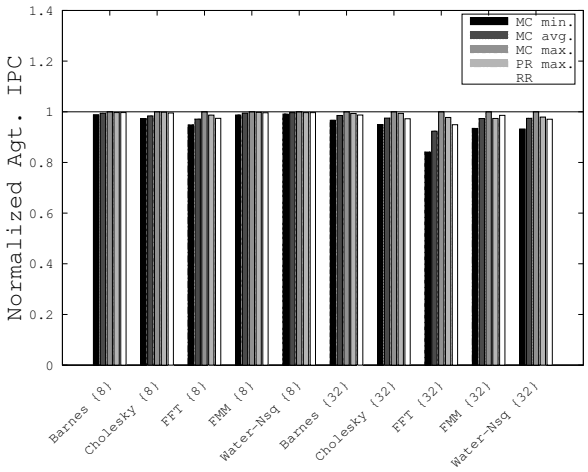
From Figure 4a, we can deduce that the PageRank-based mapping algorithm is quite effective at improving the overall performance of all of the benchmarks shown. As stated in the study, the algorithm achieves 98% of the IPC performance of the best found random mapping, exhibiting near optimal performance as defined by the Monte Carlo search at a fraction of the simulation iterations. The Cholesky 32-thread design point shows an 126% IPC increase over the worst random mapping, indicating that it is definitely worth performing static mapping for some workloads. Lastly, for of all the design points, the best PageRank-derived mapping performs no worse than 87% of the measured optimal IPC, so the algorithm is useful in all situations [7].

Despite the success of the novel PageRank mapping algorithm under the original simulation conditions reported in [7], we are interested in two points of investigation: *1*) The validity of the approach under a reproducible and trustworthy environment *2*) Investigate the efficacy of Round-Robin mappings inspired by OS process-scheduling. Extending the tests done in Section III, we add the "rob" configuration to the simulation configuration in Table I and emulate the PageRank static mapping algorithm. The comparison of the best cycle-accurate test PageRank mapping for each design point versus the cycle-accurate tested Monte Carlo random mappings is found in Figure 4b. Moreover, IPC results for Round-Robin mappings inspired by OS process scheduling are included in the simulated mappings.

The cycle-accurate simulations of PageRank (i.e. previous thread mapping technique in [7] applied in a cycle-accurate simulator) show that the best of the bag of algorithms fall within 99.0% of the optimal found random mapping. We see




(a) Normalized to MC max. - fast




(b) Normalized to MC max. - cycle-accurate

Fig. 4: PageRank and Round-Robin Mapping Performance Versus Monte Carlo Worst, Average, & Best Case Performance

that the improvement in the optimality stems from the smaller range of achievable performance indicated by Figures 2 and 1b. Interestingly, Figure 4b also shows the simplistic Round-Robin mapping, presented in this work, outperforming PageRank in one benchmark by 2% of optimal, as well as doing better than the average random mapping in 5 out of 10 benchmarks. Given this data, we propose that for best performance, the Round-Robin algorithm should be added to the same bag of algorithms for static thread mapping. In essence, we propose to use 9 simulations (8 PageRank algorithms from [7] and Round-Robin) instead of the 8 simulations proposed in previous work in [7]. Both are significantly more efficient than performing 240 Monte Carlo simulations of random mappings. With the 9 simulations, the abridged search space explorations averages an improved 99.2% of the optimal performance. Given the size of the design space for even a small configuration as explored in Section II, we can maximize performance with a significant reduction in offline search time.

## V. Conclusion

From these experiments, we find that cycle-accurate (or close) simulators are necessary for reliable design space explorations. Section III concludes that nondeterminism completely invalidates design points derived through simulation since performance distributions for a single point show modes in performance separated by as much as 8 aggregate IPC. Cycle-accurate simulation shows a much tighter grouping of viable performance results (Figure 3), but concludes that it may still be worth searching for the extra performance in workloads with trillions of instructions, due to simulation/execution time savings. In Section IV, we conclude that we can further refine effective design space exploration for static thread mappings by augmenting the recent PageRank-based approach with a single additional search design point, giving improved performance on average over random mappings (within 99% optimal), while again saving on simulation time against thousands of Monte Carlo runs.

A methodical static mapping technique has been shown to achieve IPC performance close to an approximated upper bound versus a simple technique such as Round-Robin. With this in mind, bolder NoC CMPs can be designed with the quantity of cores in the hundreds and with multi-threaded software.

## References


[1] "Tilera Tile Gx-100," online, http://www.tilera.com.

[2] N. Binkert *et al.*, "The gem5 simulator," in *The ACM SIGARCH Computer Architecture Newsletter*, August 2011, pp. 1–7.

[3] T. E. Carlson, W. Heirman, and L. Eeckhout, "Sniper: Exploring the level of abstraction for scalable and accurate parallel multi-core simulation," in *International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, November 2011.

[4] M. Kandemir, O. Ozturk, and S. P. Muralidhara, "Dynamic thread and data mapping for noc based cmps," in *Proceedings of the 46th Annual Design Automation Conference (DAC)*, 2009, pp. 852–857.

[5] E. Molina da Cruz, M. Zanata Alves, A. Carissimi, P. Navaux, C. Ribeiro, and J. Mehaut, "Using memory access traces to map threads and data on hierarchical multi-core platforms," in *2011 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, May 2011, pp. 551–558.

[6] S. Nilakantan and M. Hempstead, "Platform-independent analysis of function-level communication in workloads," in *Proceedings of the IEEE International Symposium on Workload Characterization (IISWC)*, Sept 2013, pp. 196–206.

[7] G. Salvador, S. Nilakantan, A. More, M. Hempstead, and B. Taskin, "Static thread mapping for nocs via binary instrumentation traces," in *Proceedings of the IEEE International Conference on Computer Design (ICCD)*, October 2014.

[8] D. Sanchez and C. Kozyrakis, "Zsim: Fast and accurate microarchitectural simulation of thousand-core systems," in *Proceedings of the 40th Annual International Symposium on Computer Architecture*, ser. ISCA '13, 2013, pp. 475–486.

[9] C. Su, D. Li, D. S. Nikolopoulos, M. Grove, K. Cameron, and B. R. de Supinski, "Critical path-based thread placement for numa systems," *SIGMETRICS Perform. Eval. Rev.*, vol. 40, no. 2, pp. 106–112, Oct. 2012.

[10] S. Vangal *et al.*, "An 80-tile 1.28tflops network-on-chip in 65nm cmos," in *Proceedings of the IEEE International Solid-State Circuits Conference (ISSCC)*, February 2007, pp. 98–589.