Static Thread Mapping for NoCs via Binary Instrumentation Traces

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*—A novel methodology is proposed for thread mapping on a chip-multiprocessor (CMP) system with a network-on-chip (NoC). This novel mapping leverages multi-threaded traces produced by a binary instrumentation tool, which classifies the communication and computation events for each thread of a multi-threaded program application. Processing these binary instrumentation traces after profiling, a static thread mapping is computed to improve the NoC performance.**

I. Introduction

The science and practice of distributing workloads on a parallel server farm is well studied [4], [7]. In the same vein, the distribution of threads on a CMP must be studied, respecting all the distinctive, salient features that characterize many-core processors including: cache coherency protocols, memory hierarchies, and NoC topologies. This paper proposes a methodology to address this need.

Thread mapping has been addressed through a variety of approaches for several types of CMPs. Su, Li, et al. present the use of hardware performance counters to collect local memory accesses to determine a thread placement via minimizing the critical path of parallel regions of code [11]. Kandemir, Ozturk, et al. propose using a helper thread to carry out dynamically migrating threads by analyzing the frequency of data accesses between threads in run-time intervals and solving the minimal Manhattan distance for threads affinity via an integer linear program [6]. Liu, Park, et al. address dynamically mapping threads in heterogeneous multi-core CMPs by initializing a maximum throughput mapping and iteratively performing a virtual swap threads on adjacent types of cores until the power constraint of the CMP is satisfied [8]. Molina da Cruz, Zanata Alvez, et al. propose using memory access traces from the Simics simulator to build thread affinity graphs and uses Edmond's matching algorithm to find the optimal mapping [9].

In this paper, we explore algorithms for determining producer-consumer relationships among threads and static thread mapping via multi-threaded binary instrumented traces for 2-dimensional mesh NoCs, optimizing for non-uniform access time by moving data indirectly by pinning program threads to cores. Our results show that the novel application of the PageRank algorithm never performs worse than the worst performing static mapping found during a Monte Carlo (MC) exploration, is in average 28% better, and is up to 126% better. For the all of the best mappings found using our algorithm, we are within at least 87% of the performance of the best mapping found by the exhaustive and impractical Monte Carlo technique. On average, our mappings are within 98% (as good as 101%) of the best solution found by in the exhaustive MC search. All of these performance gains are achieved using only 8 simulation iterations, one for each mapping scheme.

The remaining sections are organized as follows: Section II presents the foundational analysis of workloads via Binary Instrumentation; Section III explains the intermediate PageRank algorithm used to extract thread communication patterns; Section IV describes the process used to map threads to PEs on NoCs; Section V presents the system evaluation setup and results; and Section VI concludes the paper.

II. Binary Instrumentation

Through binary instrumentation of a running application, we can discover communication patterns between the threads of the program which are injected to the NoC for cache coherence. Binary Instrumentation (BI) is the act of inserting a monitor program into the executable of an application, and running it to record statistics of the program's dynamic behavior instruction by instruction. To do analyses of multi-threaded program communication, the BI program must be thread-aware. The BI tool we use in this work, Sigil, accomplishes this by intercepting Pthread function calls via the Valgrind framework as discussed in the work of Nilakantan et al. [10].

The multi-threaded traces output by the Sigil software delimit the execution of each thread's execution with the accesses to memory structures dedicated to Pthread locking and synchronization. Additionally, each thread's memory footprint is recorded as reads and writes to a virtual address space shared by all the threads of the application. The key takeaways are that the multi-threaded traces allow us to: *a*) capture the temporal synchronization of the parallel program *b*) capture the producer-consumer patterns via reads and writes to memory addresses in an architecture agnostic manner.

The communication patterns between the threads are captured at the core-memory interface, rather than the memory-NoC interface, enabling precise analysis of the threaded workload. Other techniques of instrumenting the dynamic communication efforts of an application running on a NoC involve capturing the packet information injected from DRAM onto the network using tools such as Netrace [5]. However, using our approach, refined information about the threads running can be obtained, especially in cases there the cores in the PEs support simultaneous multi-threading (SMT).

III. Ranking Thread Communication

At the most primitive level, thread communication patterns can be captured from multi-threaded traces post-processed in search for aggregate data traffic for use in static thread

mapping. Doing so naturally throws away some of the useful information contained in the multi-threaded traces mentioned in Section II, including synchronization data. However, we still retain the information gained from Pthread create and join calls. In other words, we maintain the identities of the threads in the program regardless of the architectural limits of how many threads can exist on a core.

The aggregate communication data culled from the threaded traces are leveraged for static mapping by creating thread affinity matrices. The matrices are symmetric and square, where each cell $(i, j) | \, i, j \in \{1, \ldots, N\}$ contains the total number of bytes communicated between threads i and j, and N is the total number of threads in the workload. Note that this means that the total bytes recorded is bidirectional. Unlike the work of Barrow-Williams et al. [1] which focuses on just the communication patterns, our analyses are used to inform static thread mapping.

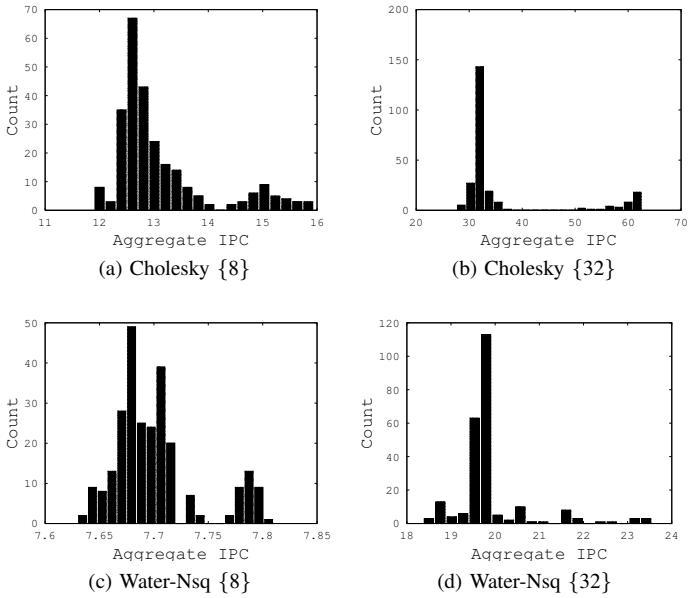
Once we have the thread affinity matrix, we can characterize the producer-consumer relationships by extrapolating patterns from the pairwise communication between threads. Our novel approach to this task is to use an adaptation of Google's PageRank algorithm to determine a global ranking for the producing-consuming centrality of all application threads. The PageRank algorithm was incepted as a solution to computing the relevance of search engine query results by modeling websites and their links as a social network [2]. Naturally, the importance and centrality of an entity in a social network is determined by their connectivity to other entities as well as the importance of those very entities. Given the reflexivity of the centrality definition and the algorithm which computes the rank, the output rank is a global indication of relevance produced from pairwise relationships [2].

By applying PageRank to a thread affinity matrix, we determine an ordering for which threads generate the most traffic on the NoC. For the most central thread, the traffic occurs either because it reads, writes, or reads and writes the largest number of bytes. Given that the PEs in the NoC contain a NUCA last-level cache, once cache requests are resolved by the cache directories, we want to minimize the latency between the bytes injected between the cohering PEs while considering network congestion as an added constraint. Due to the coherence protocol, moving threads to cores causes data to be cached as a line in the locally associated LLC, despite the location of home node of the line being elsewhere.

IV. Static Thread Mapping

Utilizing the thread rankings from our adapted PageRank algorithm, static thread mapping can be accomplished for a representative 2D mesh NoC with NUCA. A 2D mesh NoC is simple to model due its two-axis traffic routing and tractability in the number of network links. With additional engineering, our proposed framework is capable of statically mapping the threads to more complicated NoC topologies, and to CMPs where multiple threads are allowed per core. We do not pursue these extensions in this work, instead choosing to emphasize the effectiveness of our simple thread mapping scheme.

As described in Section III, we want to minimize the latency of data transfer between two threads, and therefore between two cores. However, we must also consider the effect of placing threads which participate in heavy traffic together, since this may cause heavy congestion over the network.



(a) Cholesky {8} (b) Cholesky {32}

(c) Water-Nsq {8} (d) Water-Nsq {32}

Fig. 1: Sampling of Splash-2 Monte Carlo Performance Exploration (8 & 32 Threads)

TABLE I: 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 multi-threaded traces from the BI tool, Sigil, contain the total bytes communicated between pairs of threads, and also the total number of unique byte-addressable, virtual locations in DRAM shared between threads. With these two sets of communication representations, we have proxies for total network bandwidth utilization and shared cache line between threads, respectively.

To explore the performance effect of the thread-to-core pinnings in a NUCA NoC environment using our trace aggregates, we position the threads in order of communication ranking in four configurations. Descriptions of each mapping scheme can be found in Table I. By placing heavily ranked nodes close to each other and in the center of the mesh or far apart and on the outskirts, we discover the tendencies for network congestion or hop latency to drive performance for our selected benchmarks. The distances between the threads/cores on the NoC are determined by using Manhattan (X,Y) distance, and the thread IDs to be mapped are chosen in priority order using either a max-heap or min-heap, determined by the mapping schemes description.

V. Mapping Evaluation

The efficacy of the mapping schemes described in Section IV and the room for performance gains are determined in comparison to an exhaustive Monte Carlo exploration of the sample space for all possible static mappings. For

TABLE II: 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

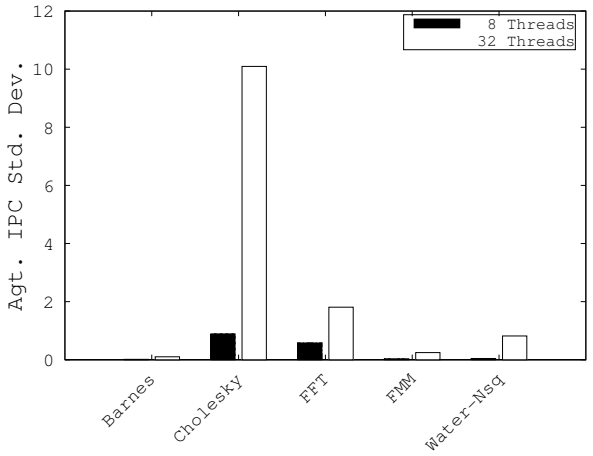


Fig. 2: Splash-2 Benchmark Performance Opportunity Derived from Monte Carlo Exploration

each of the selected benchmarks, we run each of our static mapping schemes using both total bytes communicated or unique byte addresses shared as weights for the rankings. All configurations are compared to the best-case, worst-case, and averages of 240 randomized Monte Carlo iterations for each configuration, for a total of 2400 iterations at a rate of approximately 300 per day on our 100 core cluster. Thus, the Monte Carlo is technique impractical for design exploration, but helps exhaustively search the solution space to establish the baseline for comparison of our work. In comparison, we show that for a given workload, 8 iterations, given our 8 mapping configurations, are sufficient to achieve near-optimal performance for the workload.

With the architecture agnostic Sigil multi-threaded traces and mapping algorithms that utilize general information about the communication between threads, we can execute our mappings on real-world hardware or simulate them under any hardware simulator supporting NoC topologies. We choose to simulate and evaluate our algorithms on the Sniper framework [3] due to its multi-threaded execution and simple 2D NoC mesh configurability. All simulation configuration details can be found in Table II.

The evaluation of our PageRank static mapping algorithm begins by first analyzing the range of performance achievable

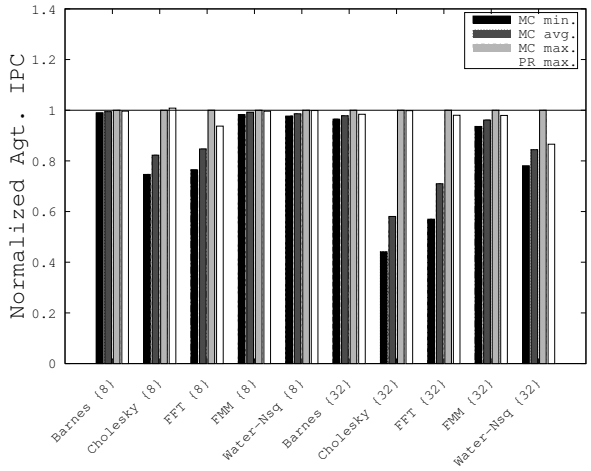


Fig. 3: PageRank Mapping Performance Versus Monte Carlo Worst, Average, & Best Case Performance

TABLE III: Monte Carlo & PageRank Aggregate IPC Results

(a) Aggregate IPC

Benchmark	MC min.	MC avg.	MC max.	PR max.
Barnes {8}	5.53	5.56	5.59	5.57
Cholesky {8}	11.90	13.11	15.94	16.07
FFT {8}	9.03	10.00	11.81	11.07
FMM {8}	10.36	10.45	10.54	10.50
Water-Nsq {8}	7.63	7.70	7.81	7.80
Barnes {32}	19.87	20.15	20.60	20.27
Cholesky {32}	27.63	36.37	62.63	62.53
FFT {32}	12.24	15.25	21.48	21.05
FMM {32}	23.21	23.84	24.80	24.29
Water-Nsq {32}	18.37	19.88	23.55	20.39

(b) Normalized ot MC max.

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

for the selected benchmarks. For the NoC configurations in Table II, Figure 1 shows the full spectrum of instructions per cycle (IPC) values for all static mappings discovered by the Monte Carlo search. Note that the Sniper framework reports aggregate IPC values for the cores across the entire NoC fabric, so we plot these IPCs. Some workloads, such as the Water-Nsquared 8 thread workload on a 3x3 NoC shown in Figure 1c, have little performance variability between thread mappings. For this benchmark the entire spectrum only spans 0.25 IPC (also seen in Table III). For others, such as the 32 thread Cholesky workload on a 6x6 NoC shown in Figure 1b, there is a lot of room for improvement over the average performing thread mapping. These trends are reinforced by Figure 2, which shows the standard deviation of the IPC for all the histograms shown in Figure 1. Moreover, as the NoC size increases to accommodate the higher thread count, we see that there is more variability possible, due to the larger number of

TABLE IV: Best and Worst PageRank Static Mappings (8 & 32 Threads; Aggregate IPC)

(a) 8 Threads

Benchmark	Worst	IPC	Best	IPC
Barnes	APRO {total bytes}	5.55	DPRI {shared bytes}	5.57
Cholesky	APRI {total bytes}	12.17	DPRO {total bytes}	16.07
FFT	DPRI {total bytes}	9.24	APRI {shared bytes}	11.07
FMM	APRI {total bytes}	10.45	DPRI {total bytes}	10.50
Water-Nsq	APRO {shared bytes}	7.67	DPRO {shared bytes}	7.80

(b) 32 Threads

Benchmark	Worst	IPC	Best	IPC
Barnes	APRI {shared bytes}	19.97	DPRI {total bytes}	20.27
Cholesky	APRO {shared bytes}	31.75	APRI {shared bytes}	62.53
FFT	APRI {shared bytes}	13.19	APRI {total bytes}	21.05
FMM	DPRO {shared bytes}	23.56	DPRI {shared bytes}	24.29
Water-Nsq	DPRI {shared bytes}	19.61	APRI {shared bytes}	20.39

thread-to-core assignments and to the increase in possible communication latency described in Section IV.

Trends for improved performance by increasing the parallelism exploited in each workload are also shown by Figures 1 and 2. There is a clear increase in IPC for all workloads when increasing the size of the NoC and number of threads which share the dataset, even by using the smallest available dataset size.

The effectiveness of our novel static mapping approach can be seen in Table IIIa. The Table shows the IPC numbers for the best mappings derived using our approach. We can also see the IPCs for the worst, average, and best case mappings found by the Monte Carlo search, reflected in the results in Figure 1. We see that for benchmarks with the most room for improvement, the application of PageRank via BI achieves the peak performance found by Monte Carlo. For the Cholesky 32 thread workload, the static mapping achieves a performance of nearly 126% over the worst static mapping of the MC exploration, and 72% better than the average-case MC mapping. The benchmarks which benefit the most from a smart static mapping are, namely, Cholesky 32 threads and FFT 32 threads.

In Table IV, we can find the PageRank mapping schemes which ranked best and worst. From this table, we determine that it is difficult to predict which mappings will provide the best IPC for any given workload. No one mapping scheme dominates the best found rankings, and all schemes appear as the worst of the lot for at least one workload. It cannot be known apriori which benchmarks have the most room for performance gains from a static thread mapping purely by inspection of the communication patterns. However, from Table IV, we see that an analysis of the shared bytes between pairs of threads characterizes the most variability in communication, and thus performance, for a benchmark. Shared bytes derived mappings claim 12 out of the 20 best/worst algorithmically derived mappings. Note that our methodology lets allows for the best algorithmic mapping for each benchmark to be chosen (the best mappings in Table IV are the same as the mapping corresponding to the PR max. values in Table IIIa), but if we were to choose the worst algorithm, the IPC performance would still be better than the worst known mapping.

The true merit of the PageRank driven approach is revealed by Figure 3 and Table IIIb. Here the IPC performance values are normalized to the best known mapping found by the Monte Carlo search for each workload and configuration. It is clear that the best mappings derived from the PageRank mappings achieve near optimal performance, within at least 87% of the IPC of best known mapping for every benchmark, and on average within 98% of the best MC mapping. Furthermore, the all PageRank derived mappings do better than the worst found mapping, so it never hurts to the apply our technique.

VI. Conclusion

Through the exploration of novel applications for static thread mapping on NoCs, the efficacy of our applied PageRank approach using only 8 iterations is validated when contrasted to the sample space of all possible mappings on a mesh topology. Figure 2 shows that there is not much room for performance gains in many benchmarks, but we are able to achieve near optimal performance with a simple approach, within at least 87% IPC of the best known solution, and on average within 98% IPC. Moreover, our approach has no performance degenerative cases; it always performs better than the worst found mapping, with nearly a 126% IPC improvement for exploitable workloads.

References

[1] N. Barrow-Williams, C. Fensch, and S. Moore, "A communication characterisation of splash-2 and parsec," in *Proceedings of the IEEE International Symposium on Workload Characterization (IISWC)*, 2009, pp. 86–97.

[2] S. Brin and L. Page, "The anatomy of a large-scale hypertextual web search engine," in *Proceedings of the Seventh International World Wide Web Conference (WWW)*, vol. 30, no. 1–7, 1998, pp. 107–117.

[3] T. E. Carlson, W. Heirman, and L. Eeckhout, "Sniper: Exploring the level of abstraction for scalable and accurate parallel multi-core simulation," in *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, 2011, pp. 52:1–52:12.

[4] S. P. Dandamudi, *Hierarchical scheduling in parallel and cluster systems*, ser. Series in computer science. New York: Kluwer Academic / Plenum Publishers, 2003.

[5] J. Hestness, B. Grot, and S. W. Keckler, "Netrace: Dependency-driven trace-based network-on-chip simulation," in *Proceedings of the Third International Workshop on Network on Chip Architectures (NoCArc)*, 2010, pp. 31–36.

[6] 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.

[7] Y.-K. Kwok and I. Ahmad, "Static scheduling algorithms for allocating directed task graphs to multiprocessors," *ACM Comput. Surv.*, vol. 31, no. 4, pp. 406–471, Dec. 1999.

[8] G. Liu, J. Park, and D. Marculescu, "Dynamic thread mapping for high-performance, power-efficient heterogeneous many-core systems," in *IEEE International Conference on Computer Design (ICCD)*, 2013, pp. 54–61.

[9] 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.

[10] 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.

[11] 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.