A Comparison of Cache Hierarchies for SMT Processors

Darío Suárez Gracia[1], Teresa Monreal Arnal[2], and Víctor Viñals Yúfera[1]

Abstract—**In the multithread and multicore era, programs are forced to share part of the processor structures. On one hand, the state of the art in multithreading describes how efficiently manage and distribute inner resources such as reorder buffer or issue windows. On the other hand, there is a substantial body of works focused on outer resources, mainly on how to effectively share last level caches in multicores. Between these ends, first and second level caches have remained apart even if they are shared in most commercial multithreaded processors.**

This work analyzes multiprogrammed workloads as the worst-case scenario for cache sharing among threads. In order to obtain representative results, we present a sampling-based methodology that for multiple metrics such as STP, ANTT, IPC throughput, or fairness, reduces simulation time up to 4 orders of magnitude when running 8-thread workloads with an error lower than 3% and a confidence level of 97%.

With the above mentioned methodology, we compare several state-of-the-art cache hierarchies, and observe that Light NUCA provides performance benefits in SMT processors regardless the organization of the last level cache. Most importantly, Light NUCA gains are consistent across the entire number of simulated threads, from one to eight.

Keywords— **Cache Hierarcy, Multithreading, Simulation, Sampling, NUCA**

I. Introduction

MULTITHREADING (MT) is supported by an ample spectrum of current processors devoted to uneven computing segments such as: embedded, high throughput, or high performance. Examples of representatives from these segments are Netlogic XLP832 (4-way multithreading, 8-cores), Oracle SPARC T3 (8-way multithreading, 16-cores), or IBM POWER7 (4-way multithreading, 4-6-8 cores), respectively [1], [2], [3].

All previous examples share a powerful multilevel cache hierarchy with large Last Level Caches (LLC), and only the XLP832 departs from the conventional organization including a ring for communicating the private L2 caches, the eight L3 cache banks, and the four DDR ports. While these LLCs seem able to accommodate the multiple working sets of SMT execution, sharing in the levels close to the processors proves to be more complex. On one hand, L1 and L2 caches deal with the latency-power vs. size trade-off. On the other hand, MT architectures add a new trade-off, number of threads in execution vs. miss rate. With many threads, cache misses can be tolerated executing instructions from other threads, but as the number of threads grows, the collective working set becomes larger and more changing, resulting in miss ratios potentially harmful to performance. The larger their number, the larger and size changing the collective working set becomes and the larger the miss rate. So when the miss rate reaches a critical value in which threads execution fails to overlap, the processor stalls and has the same problem that single thread machines.

SMT architectures may be favored by caches designed to support working set awareness such us the L-NUCA [4]. L-NUCAs belong to a family of cache organizations that has received much attention for improving cache performance: Non-Uniform Cache Architecture (NUCA) [5]. The seminal NUCA work targets the wire delay problem[1], and proposes the melting of the L2 and L3 caches into a meshed array of caches banks. Nevertheless, to the best of our knowledge, there is little work on evaluating NUCA with simultaneous multithreading processors (SMT).

Part of the complexity of assessing multiple cache hierarchies lies in the required simulation framework. So to carry out the experiments, we propose a simple yet efficient MT simulation methodology ensuring the accuracy of the results abreast with a sort simulation time. The methodology is based on statistical sampling, and contrary to other alternatives does not require a prior long profiling of the applications.

This work gives two main contributions. The first one is introducing a powerful methodology to evaluate MT architectures. The second one is the comparison and evaluation of several state-of-the-art cache hierarchy organizations driven by the SPEC CPU2006 benchmark suite. From the results, we conclude that regardless the number of threads L-NUCAs outperform conventional multibanked and dynamic NUCA organizations, both in terms of throughput and fairness.

The rest of the paper is organized as follows. Section II elaborates on previous work. Section III presents the proposed evaluation methodology. Section IV describes our experimental framework and the hierarchies under test. Section V comments on the results, and Section VI concludes the paper.

II. Background

Tullsen, Eggers, and Levy in their SMT seminal work compare the performance of private and shared L1 caches (for both instruction and data) and observe that regardless the number of threads (from 1 to 8) shared data caches are the best choice and private

[1]Computer Architecture Group (gaZ). Dpto. de Informática e Ingeniería de Sistemas. Instituto de Investigación en Ingeniería de Aragón. Universidad de Zaragoza. e-mail {dario, victor}@unizar.es

[2]Department of Computer Architecture. Universitat Politécnica de Catalunya (UPC). e-mail: teresa@ac.upc.edu

[1]The wire delay is longer than the bank delay and represents most part of the total cache latency in LLCs.

instruction ones are only preferable for the 8-thread case [6]. Also they point out that shared caches do not require any special hardware for coherence support. Then, Tullsen and Brown observed that in many cases when a thread experiences a very long latency operation, such as a cache miss, it is better to flush the resources of the stalled thread rather than keeping them ready [7].

Hily and Seznec studied how secondary cache bandwidth limited SMT performance in a trace-driven environment [8]. They point out that the larger the number of executed threads the larger the L1 cache size has to be. Besides, when the number of threads increases, the memory references generated by the simultaneous execution of independent threads exhibit less spatial locality than that of a single thread, increasing conflict misses.Block size is more critical than associativity and as the number of threads rise, it is preferable to keep small block sizes (16 to 32 BS).

To improve SMT performance, Settle *et al.* define a cache partitioning scheme based on column caching [9]. Two policies can control the partitioning: (a) synchronous in which each 1 million cycles, the partition is heuristically set for the next interval. (b) asynchronous in which the LRU algorithm is affected by some thread reuse counters.

Nemirovsky and Yamamoto analyzed the effect of varying cache capacity, associativity, and line size on miss rate for multistreamed architectures [10]. They observe that increasing both cache capacity and associativity reduces miss ratio specially for small caches and that large block size increase miss ratio.

The Multithreaded Virtual Processor (MVP) is a coarse-grain multithreaded system with software support that explicitly forces context switching on long latency events such as cache misses, I/O, or synchronization [11]. The evaluation comprised parallel workloads, and they show that when threads share a few data, the increment in miss rate affects performance.

Garcia *et al.* studied several data cache organizations for multithreaded processors using a trace-driven environment [12]. In accordance with other authors [8], [13], they observe that large associativities reduce inter-thread misses and that XOR-based placement reduces inter-thread miss rate in some cases. Besides, they proposed several organizations combining the hash-rehash caches and static cache splitting.

Sarkar and Tullsen proposed two strategies to minimize inter-object data cache misses at compilation time [14]. Lopez *et al.* studied control strategies for reconfigurable caches in SMT GALS processors with a limited set of SPEC CPU2000 benchmarks. They conclude that the best control strategy to maximize performance is the harmonic mean of the per-thread weighted access time [15], [16].

Several authors have proposed SMT methodologies for selecting representative mixes of programs. Raasch and Reinhardt propose to profile individual applications and with the extracted characteristics choose the most representative pairs [17]. By combining the pairs, they also generate 4-thread workloads. Van Biesbrouck *et al.* proposed a methodology considering all starting phases and points but it requires a complex profiling and an advanced work flow to gather the results [18].

On the contrary our approach neither require profiling nor complex output information gathering, but it does not consider multiple starting points by default. Nevertheless, since we employ last simulation policy[2], faster programs execute multiple times until the slower ones finishes, so they run together starting at different points.

Finally, our proposal cares about how to select representative mixes of multiprogrammed workloads and not when to finalize simulations for obtaining accurate metrics. In fact, this work pairs perfectly methods such as FAME [19].

III. A Simple Statistic-based Methodology for Multiprogrammed Workloads

The generation of simulation traces is always a time consuming and costly process. Single program traces extracted with solid approaches such as SimPoints or SimFlex provide accurate simulation results [20], [21]; however, no guidelines are found in the literature about how to obtain a representative set of thread mixes from representative samples of thread individuals. Our goal is to provide a simple method to efficiently simulate multiprogramed workloads for multiple metrics.

The key idea is instead of executing all possible combinations for a given number of threads, to execute a representative sample based on statistical sampling [22], [21] so that simulation time reduces by several orders of magnitude while keeping measurements below a given error within a confidence level large enough. In this work, we focus on mixes composed of different threads (combinations without repetition), but the proposed methodology can be used with repetition as well.

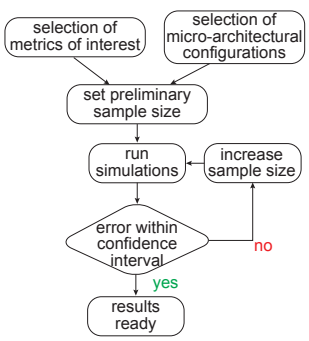


Fig. 1. Flowchart of the proposed methodology

Figure 1 shows the required steps to obtain the sample:

Selection of metrics of interest: For example if we want to compare the impact of different cache

[2]Simulation finishes when all threads have executed at least 100M instructions.

hierarchies on SMT processors we can take adjusted STP and ANTT, IPC throughput[3], and fairness [23], [24].

$$STP = \sum_{i=1}^{n} \frac{CPI_i^{SP}}{CPI_i^{MP}},\ ANTT = \frac{1}{n}\sum_{i=1}^{n} \frac{CPI_i^{MP}}{CPI_i^{SP}}$$

$$IPC\ througput = \sum_{i=1}^{n} IPC_i,\ fairness = \frac{\min_i \left(\frac{CPI_i^{MP}}{CPI_i^{SP}}\right)}{\max_i \left(\frac{CPI_i^{MP}}{CPI_i^{SP}}\right)}$$

MP and SP refer to the multithreaded and singlethreaded execution of a program.

Selection of micro-architectural configurations: Pick some of the configurations to be analyzed. In general those with lower performance are preferable because they tend to experiment the highest variances of the metrics. In our cache hierarchy comparison, we can take a conventional multibanked organization, a dynamic NUCA, and a light NUCA.

Set preliminary sample size: In this step, we have to choose a sample size—the larger the number of threads, the lower this value [25]—, and then, to randomly pick the combinations of programs for their simulation. This value should be big enough [22], larger than 30 at least, but tractable in the desired simulation environment.

Run simulations: Run the selected combinations and compute the sample size for your given confidence interval with the next formula [22]:

$$n = \left(\frac{100 \times z \times s}{r \times \overline{x}}\right)^2 \quad (1)$$

where n, z, s, r, and $\overline{x}$ stand for the sample size, normal variate of the confidence interval, error (in %), sample standard deviation, and population mean, respectively.

Error within confidence level?: Check if the obtained n is lower of equal than your preliminary sample size. If not, pick some different extra combinations, run them, and repeat the last steps until the results are ready.

Once a sample size has been established, all the configurations under test must be run in order to check that their results also fit within the confidence interval. Adding new configurations require the same procedure, so if the initial selection of micro-architectural configurations is carefully done, the sample size will not grow.

IV. Comparing SMT cache Hierarchies

A. Common Baseline Parameters

We have heavily extended Simplescalar 3.0d [26] for Alpha with: Simultaneous Multithreading Support [6], Reorder buffer and three issue windows (IW) for integer, floating point, and memory instructions, speculative wake-up support and selective recovery (as in the Intel Pentium 4 [27]), one-cycle payload and register file stages, accurate timing models for non-blocking caches, write buffers, buses, network contention, flow control, and request arbitration. For the rest of parameters see Table I.

TABLE I
Baseline Processor Configuration

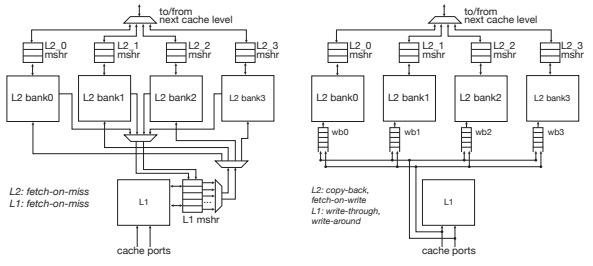
Fetch	ICOUNT.2.8		
Decode / Commit Width			4
Branch Predictor	bimodal + gshare, 16 bit	ROB / LSQ	198 / 96
Issue Width	4 (INT + MEM) + 4 (FP)	Issue Queues	64(INT) / 32 (FP & MEM)
Functional Units	4 INT + 4 FP ALU, 2 INT MULT + 4 FP MULT/DIV		
L1 cache	64KB-4Way-32B BS-2ports-LRU, Lat 2, Init. Rate 1		
L2	128KB:1MB-8Way-32B BS-1port-LRU		
D-NUCA	128KB:512KB-4Way-128B BS-1port-LRU, 8 columns		
L-NUCA	2:5Levels-16KB-4Way-32B BS-1port-LRF		
L3	8MB-16W-128B BS-1port-LRU, 8 banks		
Main Memory	First chunk: 200 cycles, 4-cycle inter chunk, 16B wires		

Thread fetch is prioritized based on the ICOUNT policy [28]. Structures such as ROB, IWs, STB, WB,... are shared among threads. To avoid starvation at commit, a thread can not occupy more that three quarters of L2/L-NUCA/D-NUCA Write Buffers. Threads commit in Round Robin fashion with a maximum of 4 instructions committed by all the threads (RR.4.X), where X corresponds to the number of threads in execution [28].

B. Cache Memory Organizations

Focussing in the first and second cache levels, we have selected three very distinct organizations. The first one is the baseline, corresponding to a conventional 3-level hierarchy with a 4-banked L2 cache, and an 8-banked, 8-MB L3 cache, see Figure 2.

In any cycle, the L2 cache can start servicing up to two L1 load misses from the MSHR, the rest of banks can simultaneously begin the processing of a write buffer entry, and two whole cache blocks can be in transit to the L1 MSHR. For this configuration, we tested L2 sizes from 128KB to 1MB organized in either 1, 2, 4, and 8 banks. The output crossbar has a fixed latency of 2 cycles with an initiation rate of 1 in configurations with 2 or more banks.



(a) Loads (b) Stores

Fig. 2. Baseline L2 cache organization with 4 banks

The second hierarchy replaces the L2 and L3 caches by a dynamic NUCA (D-NUCA) [5], Figure 3, in

[3]IPC throughput is advantageous because it allows an absolute comparison among configurations. We can use IPC throughput whenever mixes are made from independent threads, because then no unpredictable instruction spinning can arise; e.g., before entering a critical section.

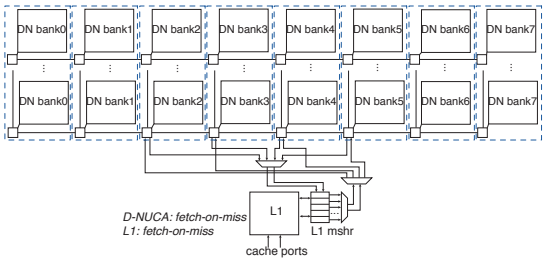
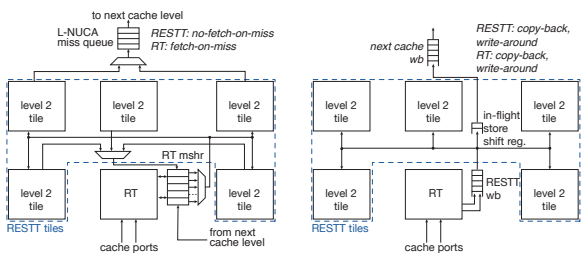


Fig. 3. Multibanked Dynamic NUCA. A block can be mapped to any of the columns surrounded by a dashed line

which its original interface with the L1 cache and shared mapping have been replaced by a crossbar to provide more bandwidth and by a simple mapping[4], respectively. As in the conventional organization, there is a write buffer for each column. In this way we can inject the same number of requests per cycle, so the conventional hierarchy can be considered as a particular case of a NUCA with a single row and without the partial tags [5].

The last one is Light NUCA (L-NUCA) [4], but also modified to provide more bandwidth, see Figure 4. Changes include widening the search links from word to block size (from 8 to 32B), increasing the write buffer size to match the rest of organizations (the buffer in between the r-tile and the rest of tiles). Besides the replacement buffers are managed with an improved back-pressure policy, so that the eviction of blocks from the r-tile does not stall until the replacement buffer of the r-tile becomes full.

We tested 2 organizations based on L-NUCA, one including the same L3 than the baseline organization, and another in which the LLC is a dynamic NUCA, similarly to previous work [4].



(a) Loads (b) Stores

Fig. 4. 2-level L-NUCA load and store logical organization. For the sake of clarity, the Figure includes neither all network nor control flow links

During the setup or the organizations, we observed two interesting design details helping to maximize performance. First, regarding the priority of L1/r-tile misses, loads should have priority over writes except when a miss hits in the write buffer[5] in which case priorities are reversed until the requested data can be served by the cache. Second, a centralized write buffer reduces performance except when the number of entries at each distributed write buffer is very small. As an approximate rule of thumb, each bank requires a coalescing write buffer with as many entries as threads are simultaneously executing. As stated by Hily and Seznec, bandwidth can be the limiting factor of the cache hierarchy in SMT [8], and the larger the number of banks (up to 8), the bigger the performance.

[4]A block can only reside in one column.

[5]Write buffers do not provide data.

C. Workloads

Our multiprogrammed workload comprises combinations of 2, and 4, 6, and 8 programs from SPEC CPU2006 without repetition, namely, all benchmarks but 483.xalanbmk [29]. For this type of study, multiprogramed loads are preferable over parallel ones because they stress more the memory hierarchy since there is no sharing among threads. For each program, we have selected a representative trace consisting of 100 million instruction following the SimPoint guidelines [20]. Simulations terminate when all threads have executed at least 100M instructions, last policy, but program statistics are gathered for the first 100M.

V. Experimental Results

A. Sample Sizes and Simulation Time

Table II shows the number of all combinations without repetition of our traces executing 2, 4, 6, and 8 threads, the average time required for running one combination of programs[6], and the sample size required to obtain accurate results, with an error less than 3% with a 97% confidence level for all configurations under test. This means for example that in 97 of 100 times a randomly chosen sample of 251 combinations will have an error than 3% in all the metric compared to the whole population of 6-threads combinations (37674). For the 2 threads case, we execute all combinations because total execution time is affordable. Nevertheless, as we increase the number of threads, our methodology obtains savings in simulation time of ×93, ×150, and ×9743, for 4, 6, and 8 threads, respectively.

TABLE II
Sample sizes varying the number of threads

	# Threads			
	2	4	6	8
Total combinations	378	20475	37674	3108105
Avrg. sim. time (min)	18.5	45.7	60.4	90.5
Sample size	—	220	251	319

Table II provides the minimum sample size for all metrics of interest with an error less than 3%, in our case: STP, ANTT, IPT throughput, and fairness. Since fairness represents the ratio between the minimum and maximum slowdowns, it has the largest variance, and, hence, determines the required sample size.

If we focus on a single number of threads, for example 4, we can show the sample size of the best configurations from each organization as table III does. L2, NC-IxJ, LNI, LNI-NC-JxK correspond

[6]This value is for the L-NUCA + D-NUCA simulator, the slowest in our simulation framework, in an Intel Nehalem 2.33 GHz.

to the conventional baseline, D-NUCA with I columns and J rows, L-NUCAs with I levels, and L-NUCAs combined with D-NUCAs organizations, respectively. Besides, each configuration includes its total size for the L2 and L-NUCAs and the size of their banks for D-NUCA.

TABLE III
SAMPLE SIZES FOR THE DIFFERENT METRICS AND CONFIGURATIONS IN 4SMT EXECUTION

	Metric			
	STP	ANTT	IPC Throughtput	Fairness
L2-256KB-8Banks	126	111	85	162
L2-1MB-8Banks	126	113	90	160
NC-8x2-512KB	111	132	119	218
NC-8x4-256KB	118	141	128	220
LN3-240KB	103	96	51	194
LN4-448KB	103	96	51	193
LN2-NC-8x2	104	96	48	189
LN2-NC-8x4	105	97	47	188
LN3-NC-8x2	106	96	46	210
LN3-NC-8x4	103	95	47	211

Irrespective of the organization, we observe that the required sample size follows a common trend: computing IPC throughput requires small sizes while fairness, due to its greater variance, requires doubling or even quadrupling the sample size. STP and ANNT lie in a middle ground. Across all organizations, IPC throughput is the metric giving the more uneven sample sizes, meaning that L-NUCA organizations achieve less IPC variability among individual thread combinations. From these results, we conclude that in general adding a different configuration to test does not require a large investment in new simulations for the preexisting configurations. Finally, fairness is the only metric that requires more samples as the number of threads rise. While many threads are able to keep all functional units busy, they tend to bother each other starving some of them. Conclusions are similar for other number of threads, and we do not show them for the sake of brevity.

B. STP, ANTT, IPC throughput, and Fairness

Figure 5 shows the results for the four metrics of interest for the best configuration of each hierarchy organization. STP and ANTT are metrics relative to the performance of a single thread system aimed at programs that may expend time in spin-lock loops, which is not our case [24]. In both metrics, L2 overpasses the rest of hierarchies from 4 threads and beyond, and the L-NUCA ones dominates in the 2 thread case. This results are mostly due to the fact the L2 has the worst IPC in single-thread mode and the maximum IPC for all configurations is 4, so L2 relative slowdowns are smaller as the number of thread rises.

IPC throughput is an absolute metric representing the amount of committed instructions per unit of time. LN+NC achieves the best results regardless the number of threads close followed by LN. The small difference between them is mostly due to the NUCA partial tags not present in the L3 [5]. As the number of threads and bandwidth demand increase, the L2 overpasses the NC.

Fairness is defined as the quotient between the programs that have suffered the lowest and the highest slowdowns. A value of zero means complete starvation of one thread,and a value of one means all threads experience the same slowdown.

VI. CONCLUSIONS

The adoption of thread level parallelism as the mainstream way to continue improving the performance pace of computers requires novel mechanism and the reevaluation of those which are well established such as cache hierarchies. While large Last Level Caches have received a lot of attention in recent years, first and second levels have remained apart.

This work analyzes multiple state-of-the-art cache hierarchies executing multiprogramed workloads from 2 to 8 threads. In order to provide accurate results in a reasonable amount of time, we propose a sampling based methodology reducing simulation time by up 4 orders of magnitude for 8 thread workloads, respectively. These savings do not occur at the cost of fidelity because their error is less than 3% for a 97% confidence level for a high variance metric such as fairness.

From the analysis we observe that regardless of the Last Level Cache and the number of threads, L-NUCA provides the more efficient solution in terms of both throughput and fairness.

ACKNOWLEDGEMENT

This work was partially supported by grants TIN2010-21291-C02-01 (Spanish Government and European ERDF), gaZ: T48 research group (Aragón Government and European ESF), Consolider CSD2007-00050 (Spanish Government), and HiPEAC-2 NoE (European FP7/ICT 217068).

REFERENCES

[1] Tom R. Halfhill, "Netlogic broadens XLP family," *Microprocessor Report*, vol. 24, no. 7, pp. 1–11, 2010.

[2] J. L. Shin, K. Tam, D. Huang, B. Petrick, H. Pham, Changku Hwang, Hongping Li, A. Smith, T. Johnson, F. Schumacher, D. Greenhill, A. S. Leon, and A. Strong, "A 40nm 16-core 128-thread cmt sparc soc processor," in *Proc. IEEE Int. Solid-State Circuits Conf. Digest of Technical Papers (ISSCC)*, 2010, pp. 98–99.

[3] Ron Kalla, Balaram Sinharoy, William J. Starke, and Michael Floyd, "Power7: Ibm's next-generation server processor," *IEEE Micro*, vol. 30, pp. 7–15, 2010.

[4] Darío Suárez, Teresa Monreal, Fernando Vallejo, Ramón Beivide, and Víctor Viñals, "Light NUCA: a proposal for bridging the inter-cache latency gap," in *Prooceedings of the 12th Design, Automation and Test in Europe Conference and Exhibition (DATE'09)*, April 2009.

[5] Changkyu Kim, Doug Burger, and Stephen W. Keckler, "An adaptive, non-uniform cache structure for wire-delay dominated on-chip caches," in *Proceedings of the 10th international conference on architectural support for programming languages and operating systems (ASPLOS-X)*. October 2002, pp. 211–222, ACM Press.

[6] D.M. Tullsen, S.J. Eggers, and H.M. Levy, "Simultaneous multithreading: Maximizing on-chip parallelism," in *Proceedings. 22nd Annual International Symposium on Computer Architecture*, Jun 1995, pp. 392–403.

[7] Dean M. Tullsen and Jeffery A. Brown, "Handling long-latency loads in a simultaneous multithreading processor," in *MICRO 34: Proceedings of the 34th annual*

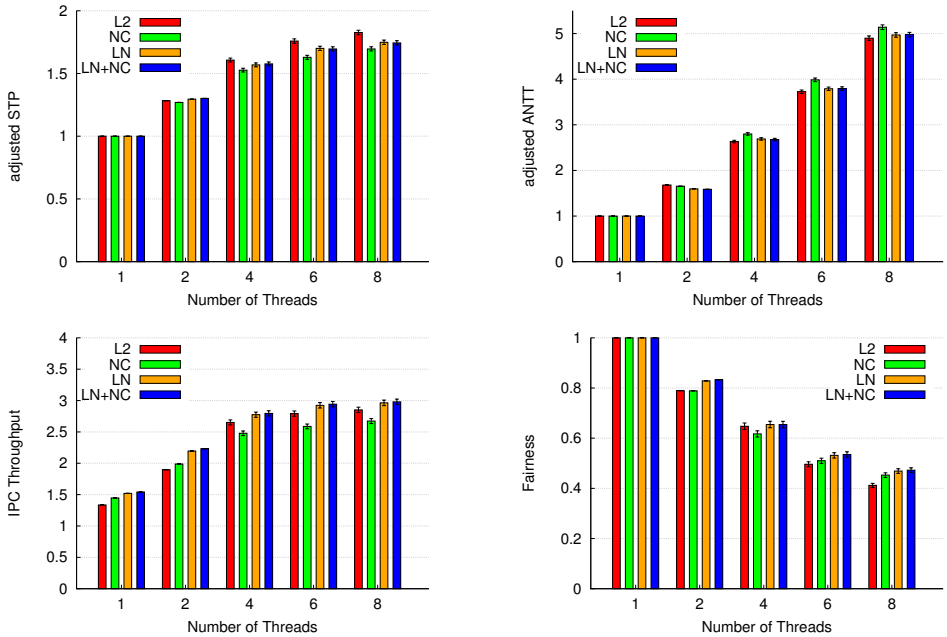


Fig. 5. Results for the best configuration of each organization: L2, NC, LN, and LN+NC correspond to the L2-256KB, NC-8x4-256KB, LN3-240KB, and LN2-NC-8x4, respectively

ACM/IEEE international symposium on Microarchitecture, Washington, DC, USA, 2001, pp. 318–327, IEEE Computer Society.

[8] Sébastien Hily and André Seznec, "Contention on 2^{nd} level cache may limit the effectiveness of simultaneous multithreading," Tech. Rep. 1086, IRISA, février 1997.

[9] Alex Settle, Dan Connors, Enric Gibert, and Antonio González, "A dynamically reconfigurable cache for multithreaded processors," *J. Embedded Comput.*, vol. 2, no. 2, pp. 221–233, 2006.

[10] Mario Nemirovsky and Wayne Yamamoto, "Quantitative study of data caches on a multistreamed architecture," in *In Workshop on Multithreaded Execution, Architecture and Compilation*, 1998.

[11] Hantak Kwak, Ben Lee, Ali R. Hurson, Suk-Han Yoon, and Woo-Jong Hahn, "Effects of multithreading on cache performance," *IEEE Transactions on Computers*, vol. 48, pp. 176–184, 1999.

[12] Montse García, José González, and Antonio González, "Data caches for multithreaded processors," in *Proc. of the Workshop on Multithreaded Execution, Architecture and Compilation*, 2000.

[13] Sébastien Hily and André Seznec, "Standard memory hierarchy does not fit simultaneous multithreading," in *Proceedings of the 2^{nd} Workshop on MULTI-THREADED EXECUTION, ARCHITECTURE and COMPILATION (MTEAC-2)*, 1998.

[14] Subhradyuti Sarkar and Dean M. Tullsen, "Data layout for cache performance on a multithreaded architecture," in *Transactions on high-performance embedded architectures and compilers III*, Per Stenström, Ed., chapter Data layout for cache performance on a multithreaded architecture, pp. 43–68. Springer-Verlag, Berlin, Heidelberg, 2011.

[15] Sonia López, Steve Dropsho, David H. Albonesi, Oscar Garnica, and Juan Lanchares, "Dynamic capacity-speed tradeoffs in smt processor caches," in *Proceedings of the 2nd international conference on High performance embedded architectures and compilers*, Berlin, Heidelberg, 2007, HiPEAC'07, pp. 136–150, Springer-Verlag.

[16] Sonia Lopez, Oscar Garnica, David H. Albonesi, Steven Dropsho, Juan Lanchares, and Jose I. Hidalgo, "Adaptive cache memories for smt processors," *Digital Systems Design, Euromicro Symposium on*, vol. 0, pp. 331–338, 2010.

[17] Steven E. Raasch and Steven K. Reinhardt, "The impact of resource partitioning on smt processors," in *Proceedings of the 12th International Conference on Parallel Architectures and Compilation Techniques*, Washington, DC, USA, 2003, PACT '03, pp. 15–, IEEE Computer Society.

[18] Michael Van Biesbrouck, Lieven Eeckhout, and Brad Calder, "Representative multiprogram workloads for multithreaded processor simulation," in *IEEE Workload Characterization Symposium*. September 2007, pp. 193–203, IEEE Computer Society.

[19] F.J. Cazorla, A. Pajuelo, O.J. Santana, E. Fernandez, and M. Valero, "On the problem of evaluating the performance of multiprogrammed workloads," *IEEE Trans. on Computers*, vol. 59, no. 12, pp. 1722 –1728, 2010.

[20] Greg Hamerly, Erez Perelman, Jeremy Lau, and Brad Calder, "Simpoint 3.0: Faster and more flexible program analysis," in *Proceedings of Workshop on Modeling, Benchmarking and Simulation*, 2005.

[21] Thomas F. Wenisch, Roland E. Wunderlich, Michael Ferdman, Anastassia Ailamaki, Babak Falsafi, and James C. Hoe, "Simflex: Statistical sampling of computer system simulation," *IEEE Micro*, vol. 26, pp. 18–31, July 2006.

[22] Raj Jain, *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*, John Wiley & Sons, Inc., April 1991.

[23] Ron Gabor, Shlomo Weiss, and Avi Mendelson, "Fairness and throughput in switch on event multithreading," in *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, Washington, DC, USA, 2006, MICRO 39, pp. 149–160, IEEE Computer Society.

[24] S. Eyerman and L. Eeckhout, "System-level performance metrics for multiprogram workloads," *Micro, IEEE*, vol. 28, no. 3, pp. 42 –53, may. 2008.

[25] M. Ekman and P. Stenstrom, "Enhancing multiprocessor architecture simulation speed using matched-pair comparison," in *Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software, 2005*, Washington, DC, USA, 2005, pp. 89–99, IEEE Computer Society.

[26] Todd Austin and Doug Burger, *SimpleScalar Tutorial (for tool set release 2.0)*, SimpleScalar LCC, 1997.

[27] Glenn Hinton, Dave Sager, Mike Upton, Darrell Boggs, Doug Carmean, Alan Kyker, and Patrice Roussel, "The microarchitecture of the Pentium® 4 processor," *Intel Technology Journal*, vol. 1st quarter, pp. 1–13, 2001.

[28] Dean M. Tullsen, Susan J. Eggers, Joel S. Emer, Henry M. Levy, Jack L. Lo, and Rebecca L. Stamm, "Exploiting choice: instruction fetch and issue on an implementable simultaneous multithreading processor," in *Proceedings. 23nd Annual International Symposium on Computer Architecture*, New York, NY, USA, 1996, vol. 24, pp. 191–202, ACM.

[29] John L. Henning, "Spec cpu2006 benchmark descriptions," *SIGARCH Comput. Archit. News*, vol. 34, no. 4, pp. 1–17, 2006.