Core to Memory Interconnection Implications for Forthcoming On-Chip Multiprocessors

Carmelo Acosta †, Francisco J. Cazorla ⋆, Alex Ramirez †⋆, Mateo Valero †⋆

† Universitat Politecnica de Catalunya
HiPEAC European Network of Excellence
Barcelona, Spain
{cacosta,aramirez,mateo}@ac.upc.edu

⋆ Barcelona Supercomputing Center
Barcelona, Spain
{francisco.cazorla,alex.ramirez,mateo.valero}@bsc.es

Abstract

Nowadays, there is a clear trend in industry towards employing the growing amount of transistors on chip in replicating execution cores, where each core is Simultaneous Multithreading (SMT). In order to appropriately connect such a growing number of on-chip execution cores to a shared cache subsystem, some traditional considerations regarding SMT should be revisited.

In this paper we study the effects of long latency instructions and their interaction with the processor to cache interconnection network in new scenarios comprising multiple on-chip SMT cores. We show results that clearly indicate the important role of the on-chip interconnection networks, used to communicate the execution cores with the shared L2 Cache, and its interaction with the specific architecture of each core.

1 Introduction

As process technology advances, and we have at our disposal more transistors on chip, the issue of how to effectively employ so many resources gets more importance. This quest of effectiveness led to Simultaneous Multithreading (SMT) [4, 8, 11] and On-Chip Multiprocessors (CMP) [5]. Nowadays, there is a clear trend in industry towards CMP or even CMP+SMT processors, like the Intel Core 2 Duo [10], IBM Power5 [2], and Sun's T1 [1] Niagara processors. Morever, this trend seems to head towards high-degree CMPs [1], with lots of on-chip cores.

On the one hand, the CMP conventional designs share the second level (L2) cache among all the on-chip cores by means of an interconnection switch. As the number of on-chip cores increases, the pressure on this L2 cache and interconnection network is also augmented. As a result, the L2 access time turns more unpredictable.

On the other hand, in SMT processors, L2 access time is used to determine when a thread has missed in the L2 cache. As shown by some authors the L2 cache misses are of a key importance in SMTs [7], since a long latency instruction may stall the whole machine. A way to deal with long-latency operations, like L2 cache misses, is the Instruction Fetch Policy. The IFetch Policy determines from which thread(s) instructions are fetched every cycle. Several authors have shown that long latency operations have to be taken into account by the IFetch Policy in order to boost SMT performance [3, 7, 9]. Some of these IFetch policies track the delay of loads when accessing the outer cache level (the L2 cache in our processor setup) to determine whether they miss and stop/flush the corresponding thread.

In this paper we shed some light on the implications of having multiple SMT cores sharing a single L2. We focus on the forthcoming high-degree CMP processor generations, with multiple SMT cores sharing an L2 Cache. As we augment the number of replicated

[1]In this paper the term *degree of a CMP* refers to number of cores of the CMP. Analogously, the term *degree of an SMT* refers to the number of contexts of that SMT. For example, IBM Power5 is a 2-degree CMP where each core is a 2-degree SMT.

SMT cores sharing an L2 cache the competition for the interconnection network that communicates each core with the shared L2 increases. Our results show that the FLUSH IFetch policy, which has probed to be better than the ICOUNT fetch policy for single-core SMTs, reduces its improvements over ICOUNT as we increase the number of SMT cores and even provides worse results than ICOUNT for quad-core configurations.

2 Methodology

We use a trace driven SMT simulator derived from SMTsim [8]. The simulator consists of our own trace driven front-end and an improved version of the SMT-sim's back-end that provides multicore support. Our simulator also permits simulating the impact of executing along wrong paths on the branch predictor and the instruction cache by having a separate basic block dictionary in which information of all static instructions is contained.

Our workloads [2] use the SPEC2000 benchmark suite. From them, we have collected traces of the most representative 300 million instruction segment of each benchmark, following the idea presented in [6]. Each program is compiled with the *–O2 –non_shared* options using DEC Alpha AXP-21264 C/C++ compiler and executed using the reference input set. Since a complete study of all benchmarks is not feasible due to excessive simulation time, we have chosen some of them, making groups according to their characteristics. We classified benchmarks into INT/FP according to the spec suite, and into high-ILP (ILP) or memory-bounded (MEM) according to their memory behavior [3]. Figure 1 shows the main simulation parameters so as the workloads chosed.

Regarding the configurations used in this paper, we denote them as CXTY, where X stands for the number of cores and Y stands for the number of threads on each core (i.e. C4T2 stands for a configuration with 4 cores with 2 threads in each core). The specific configuration of each core is the one shown in Figure 1.

Running multi-threaded simulations requires a commitment regarding the finalization methology, that is, when do we consider a simulation to be finished. For such a simulation to yield reliable results, we must assure that all threads run simultaneously during a representative amount of time. This may suppose prohibitive computational costs with a wide range of workloads, since not all benchmarks require the same amount of time to finish their executions. In this sense we considerer a simulation finishes after executing 120 million of cycles[3].

3 Interaction between the IFetch Policy and the intercontection network

In our research we focus on CMPs comprised of SMT cores. Each core allows two threads running simultaneously and has its private Instruction Cache and Data Cache (see details in Figure 1). The first level cache is connected through an on-chip bus-based interconnection network to a shared L2 Cache; each core‘s ICache and DCache is connected to each of the shared L2 banks.

In order to evaluate the traffic effects on the on-chip core-to-cache interconnection network and the interaction with the intra-core IFetch policy implemented we use two well-known IFetch policies: ICOUNT and FLUSH.

The ICOUNT policy [9] prioritizes threads with fewer instructions in the pre-issue stages, and presents good results for threads with high ILP. However, SMT has difficulties with threads that experience many loads that miss in L2. When this situation happens, then ICOUNT does not realize that a thread can be blocked on an L2 miss and will not make forward progress for many cycles. Depending on the amount of instructions dependent of the blocked load, many processor resources may be blocked and the total throughput suffers from a serious slowdown.

The performance of fetch policies dealing with load miss latency depends on the following two factors: the detection moment (DM) and the response action (RA). The DM indicates the moment in which the policy detects a load that fails or is predicted to fail in cache. Possible values range from the fetch of the load until the moment that the load finally fails in the L2 cache. Two characteristics associated with the DM are the

[2]Whenever a thread is used more than once in a workload each additional instance is forwarded 1 million instruccions more than the prior instance.

[3]We refer here to simulated cycles.

Simulation Parameters			
Pipeline depth	11 stages	L1 I-Cache	64KB, 4-way, 8 banks
Queues Entries	64 int, 64 fp, 64 ld/st	L1 D-Cache	32KB, 4-way, 8 banks
Execution Units	4 int, 3 fp, 2 ld/st	L1 lat./miss	3/22 cycs.
Physical Registers	320 regs.	I-TLB ,D-TLB	512 ent. Full-assoc.
ROB Size*	256 entries	TLB miss	300 cycs.
Branch Predictor	perceptron	L2 Cache	4MB, 12-way, 4 banks
	(4K local, 256 perceps)	L2 latency	15 cycs.
BTB	256 entries,	Main Memory lat.	250 cycs.
	4-way associative		
RAS*	100 entries		

gzip	a	swim	n
vpr	b	apsi	o
gcc	c	wupwise	p
mcf	d	equake	q
crafty	e	lucas	r
perlbmk	f	mesa	s
parser	g	fma3d	t
eon	h	sixtrack	u
gap	i	facerec	v
vortex	j	applu	w
bzip2	k	galgel	x
twolf	l	ammp	y
art	m	mgrid	z

	Number of Threads			
Type	2	4	6	8
ILP1	e, j	e, t, u, j	h, e, t, w, u, j	i, j, c, f, k, e, a, h
ILP2	i, f	h, j, k, f	a, h, j, k, e, f	k, e, a, h, o, p, s, t
ILP3	o, p	w, p, s, t	u, w, p, z, s, t	j, f, e, h, p, t, v, y
ILP4	a, p	e, a, w, p	k, j, h, s, y, t	w, u, v, z, u+1, p, s, t
ILP5	k, t	k, h, s, t	h, u, y, j, a, w	c, w, f, x, e, z, a, y
MEM1	l, d	m, l, q, d	b, m, l, q, g, d	d, l, b, g, m, n, r, q
MEM2	m, n	m, n, r, q	m, r, n, r+1, q, m+1	l, b, l+1, g, b+1, l+2, b+2, g+1
MEM3	b, g	d, l, b, g	b, d, l, b+1, g, l+1	m, n, m+1, r, n+1, m+2, n+2, r+1
MEM4	g, r	b, g, m, r	l, b, g, m, q, r	b, g, r, q, b+1, g+1, r+1, q+1
MEM5	n, q	l, g, n, q	l, l+1, g, n, q, n+1	d, l, m, n, d+1, l+1, m+1, n+1
MIX1	b, j	b, q, t, j	l, b, q, f, t, j	d, l, b, g, i, j, c, f
MIX2	n, e	l, n, p, e	g, l, n, p, e, a	b, g, m, n, a, h, o, p
MIX3	d, a	d, s, r, a	d, l, s, w, r, a	m, n, r, q, i, j, e, h
MIX4	g, f	g, b, m, f	r, g, b, m, h, f	l, b, g, m, n, r, f, s
MIX5	r, p	r, j, f, p	h, l, e, r, m, d	q, b, c, k, e, a, o, t

Figure 1. Simulation parameters and Workloads. (resources marked with * are replicated per thread)

reliability and the speed. The higher the speed of a method to detect a delinquent load, the lower its reliability. On the one hand, if we wait until the load misses in L2, we know for certain that it is a delinquent load: totally reliable but too late. On the other hand, we can predict which loads are going to miss by adding a load miss predictor to the front-end. In this case, the speed is highest, but the reliability is low due to predictor mispredictions. The RA indicates the behavior of the policy once a load is detected or predicted to miss in cache, that is, it defines the measures that the fetch policy takes for delinquent threads. With these two parameters, we will classify all current policies related to long latency loads.

In [7] several RA are proposed. We focus on the mechanism leading to the best performance, what we call FLUSH. As a result of applying FLUSH, the offending thread temporarily does not compete for resources and, more importantly, the resources used by this thread are freed, giving the other threads full access to them. Several DM are proposed for the FLUSH response action.

- *Delay after issue DM:* When this DM is used, a load is declared to miss in the L2 cache when it spends more cycles in the cache hierarchy than needed to access the L2 cache, including possible resource conflicts. We will refer to this FLUSH's DM as Speculative (FL-SX) - X stands for the delay after which the mechanism is triggered.

- *Trigger on miss DM:* In this case we wait the load miss in the L2 to start the corresponding RA. We will refer to this FLUSH's DM as Non Speculative (FL-NS).

3.1 Single-core analysis

According to our simulation parameters (see Figure 1) we chose 30 cycles (FL-S30) as the delay after issuing a load from the load issue queue to activate FLUSH mechanism.

Our results are consistent with [7]: the delay-after-issue DM is better than trigger on miss, both improving ICOUNT. For this experiment, we simulated a single-core SMT configuration. In this uniprocessor, with two hardware contexts, we ran all two-threaded workloads in Figure 1. From the results shown in Figure 2 it can be asserted that the FLUSH mechanism reduces throughput losses in workloads containing threads with bad memory behaviors (MEM and MIX). Thus, the FLUSH mechanism yields speedups of up to 93% in MIX workloads, with an average speedup of 22%. However, as we will see following, these asserts are highly dependent on the specific configuration. In particular with the amount of replicated SMT cores.

3.2 Multiple-core analysis

Next, we simulated the whole workloads in Figure 1 replicating SMT cores with two threads per core. The average results per each workload type and machine configuration are shown in Figure 4. These results reveals that in multicore configurations with an increasing number of SMT cores per chip the prior affirmations can not be asserted. In fact, as we increase the amount of replicated SMT cores the 22% average speedup obtained with the FLUSH mechanism, as compared to ICOUNT, experiences a progressive reduction. Hence, with a four-core configuration (C4T2) the FLUSH mechanism yields an average slowdown close to 9%.

In order to shed some light into the rationale behind these results, we deeply analyzed the influence of the access time to L2. For this experiment, we measure the number of cycles it takes a load to access the L2 cache since it is issued from the load/store issue queue, see Figure 3. For this experiment we use the ICOUNT policy given that FLUSH work on top of it. We observe that augmenting the number of cores critically increases the probability of suffering from higher latencies in L2 accesses. Increasing the number of SMT cores has a deep impact on the traffic of the interconnection network that communicates each core with each of the shared L2 cache banks, so as L2 banks contention. Since each core has a limited number of Load/Store (LDST) units, shared by all threads running in the core, increasing the number of cores puts more pressure on the interconnection network and the L2 cache. As we increase the amount of cores connected to this shared resource, the possibility of suffering from contention increases. We also observe that as we increase the number of cores the dispersion also increases. This indicates us that there is no a single threshold that we can choose for the trigger mechanism that provides good results.

This high variability in the L2 access time causes the following effects.

- On the one hand, if we establish a low threshold the number of *false misses* increases. That is, the number of times we flush a load predicting that it is going to miss and finally it turns out to be a non-miss L2 cache access. As a result, the performance of the FLUSH policy is heavily affected.

- On the other hand, if we establish a high threshold the number of cycles a thread can clog resources increases, leading to performance lost. We comment on this issue in the next section.

To sum up, the throughput of the FLUSH mechanism configurations reflects a clear trend to get diminishing returns as we increase the number of cores. In fact, the FLUSH mechanism turns ineffective just by passing from a dual core to a quadruple core as depicted in Figure 4.

3.3 Detection Moment Analysis

The results presented so far indicates a clear trend to higher levels of dispersion as augment the number of on-chip replicated SMT cores. To deeply analyze this effect we ran some additional simulations covering a wider DM spectrum. For an explanatory analysis, in this experiment we chosed two representative four-cored configuration examples. The 8-threaded MIX3 (see Figure 1) and an 8-threaded workload comprised of bzip2 and twolf, where instances of the two applications never share a single core, are presented in left

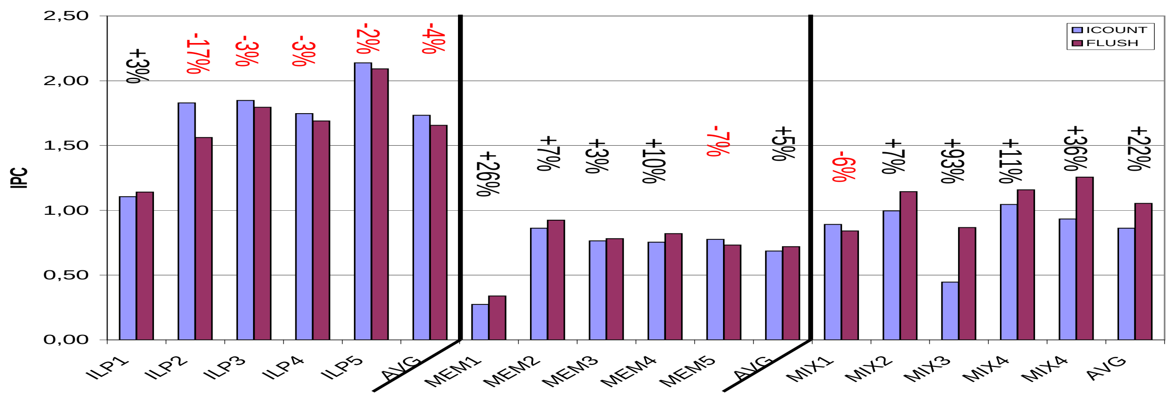
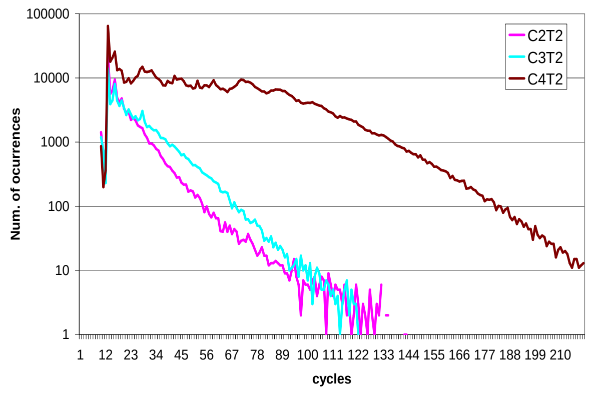


Figure 2. Throughput in SMT.



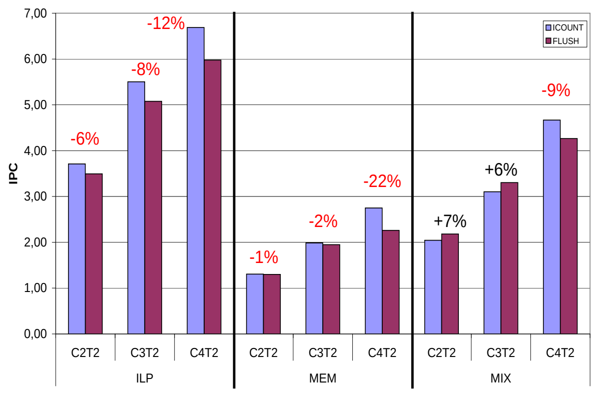
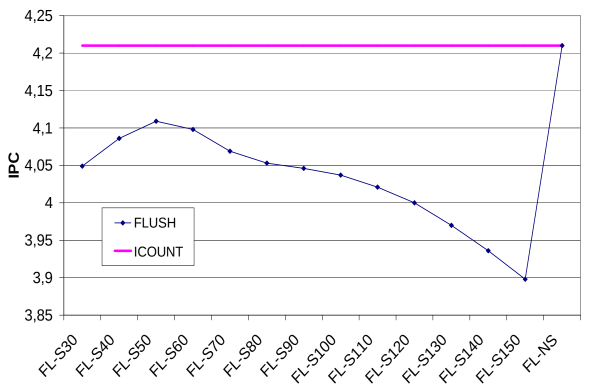


Figure 3. Average access to L2: cycles between a load is issued until if access L2.

Figure 4. Average throughput in multicore configurations.

and right side of Figure 5, respectively. In the first, the trigger value that yields the highest overall throughput is 50 cycles. However, compared to speculative instances, the non-speculative FLUSH implementation yields the highest overall throughput. In the second, the best trigger value is 90 cycles. These examples indicates that there may be trigger values which best balance the amount of false misses and the clog resources, yielding the highest overall throughput.

As shown in the left side of Figure 5, such a best-balanced trigger may be improved by a non-speculative FLUSH implementation, where no harmful false misses occurs. However, this latter situation varies depending on the specific workload analyzed, with examples in both senses that either favor the non-speculative or some speculative instance, as occurs in the right side of Figure 5. This dependency of the specific workload run together with the dependency to the number of cores definitely imposes hard constraints to stablish a good-for-all trigger value.



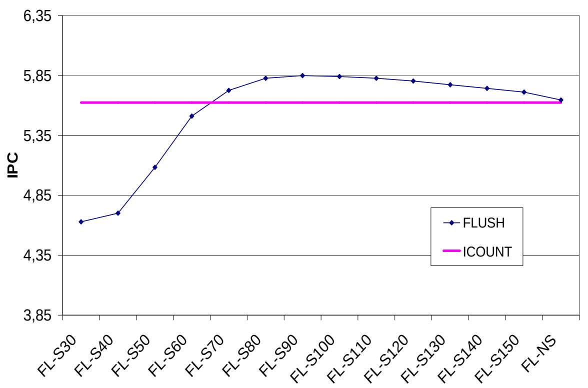


Figure 5. Detection Moment Analysis.

4 Conclusions

In this paper we show results which clearly indicate that future high-degree Multithreaded CMPs, with multiple SMT execution cores sharing an L2 cache, may not simply rely on SMT IFetch policies to boost overall throughput. Neglecting the important role of both the on-chip processor to memory interconnection network and the outer on-chip cache level design may have disastrous consequences. Thus, conventional SMT mechanisms to obtain high performance are directly affected by possible interconnection traffic and heavy cache banks contention, not considered in their original design, turning them into not effective. A bus-based interconnection network between each SMT execution core and a shared L2 cache may not be enough in this new scenario; requiring a bigger on-chip resource budget to implement latency hiding interconnection networks.

Acknowledgements

This work has been supported by the Ministry of Education of Spain under contract TIN2004–07739–C02–01, the Barcelona SuperComputing Center(BSC) and the HiPEAC European Network of Excellence. The authors wish to thank Daniel Ortega, Ayose Falcon, Jeroen Vermoulen, and Oliverio J. Santana for their support and help with the simulation tools.

References

[1] UltraSPARC T1 Supplement. *Draft D2.0, 17 Mar*, 2006.

[2] J. M. Tendler R. J. Eickemeyer . Sinharoy, R. N. Kalla and J. B. Joyner. POWER5 system microarchitecture. *IBM Journal of Research and Development. 49(4/5):505–52*, 2005.

[3] F. J. Cazorla, E. Fernández, A. Ramirez, and M. Valero. Dynamically Controlled Resource Allocation in SMT Processors. In *Proc. of MICRO-37*, 2004.

[4] M. J. Serrano and R. Wood and M. Nemirovsky. A Study on Multistreamed Superscalar Processors. Technical Report 93-05, University of California Santa Barbara, 1993.

[5] K. Olukotun, B. A. Nayfeh, L. Hammond, K. Wilson, and K. Chang. The case for a single-chip multiprocessor. In *Proc. of ASPLOS-7*, 1996.

[6] T. Sherwood, E. Perelman, and B. Calder. Basic Block Distribution Analysis to Find Periodic Behavior and Simulation Points in Applications. In *Proc. of PACT-10*, 2001.

[7] D. M. Tullsen and J. A. Brown. Handling Long-latency loads in a Simultaneous Multithreaded Processor. In *Proc. of MICRO-34*, 2001.

[8] D. M. Tullsen, S. Eggers, and H. M. Levy. Simultaneous multithreading: Maximizing on-chip parallelism. In *Proc. of ISCA-22*, 1995.

[9] D. M. Tullsen, S. J. Eggers, J. S. Emer, H. M. Levy, J. L. Lo, and R. L. Stamm. Exploiting choice: Instruction fetch and issue on an implementable simultaneous multithreading processor. In *Proc. of ISCA-23*, 1996.

[10] Ofri Wechsler. Inside Intel Core Microarchitecture. *White Paper*.

[11] W. Yamamoto and M. Nemirovsky. Increasing superscalar performance through multistreaming. In *Proc. of PACT*, 1995.