HAL
archives-ouvertes.fr

Contention on 2nd Level Cache May Limit the Effectiveness of Simultaneous Multithreading

Sébastien Hily, André Seznec

▶ **To cite this version:**

Sébastien Hily, André Seznec. Contention on 2nd Level Cache May Limit the Effectiveness of Simultaneous Multithreading. [Research Report] RR-3115, INRIA. 1997. <inria-00073575>

HAL Id: inria-00073575
https://hal.inria.fr/inria-00073575

Submitted on 24 May 2006

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Contention on 2nd Level Cache May Limit the Effectiveness of Simultaneous Multithreading

Sébastien Hily & André Seznec

N° 3115

février 1997

THÈME 1

ISSN 0249-6399

Contention on 2nd Level Cache May Limit the Effectiveness of Simultaneous Multithreading

Sébastien Hily & André Seznec

Thème 1 — Réseaux et systèmes
Projet CAPS

Rapport de recherche n ° 3115 — février 1997 — 24 pages

Abstract: *Simultaneous multithreading* (*SMT*) is an interesting way of maximizing performance by enhancing processor utilization. We investigate issues involving the behavior of the memory hierarchy with *SMT*. First, we show that ignoring L2 cache contention leads to strongly over-estimate the performance one can expect and may lead to incorrect conclusions.
We then explore the impact of various memory hierarchy parameters. We show that the number of supported threads has to be set-up according to the cache size, that the L1 caches have to be associative and small blocks have to be used. Then, the hardware constraints put on the design of memory hierarchies should limit the interest of *SMT* to a few threads.

Key-words: multithreading, simultaneous multithreading, cache hierarchy

(Résumé : tsvp)

Les travaux de Sébastien Hily sont en partie financés par la région Bretagne

* hily@irisa.fr, seznec@irisa.fr

Unit´e de recherche INRIA Rennes
IRISA, Campus universitaire de Beaulieu, 35042 RENNES Cedex (France)
T´el´ephone : (33) 02 99 84 71 00 – T´el´ecopie : (33) 02 99 84 71 71

La contention sur le second niveau de cache pourrait limiter l'efficacité du *multiflot simultané*

Résumé : Le *multiflot simultané* (*SMT*) est une voie intéressante pour augmenter les performances des microprocesseurs en améliorant leur utilisation. Dans cette étude, nous évaluons le comportement de la hiérarchie mémoire placée dans une architecture supportant le *multiflot simultané*. D'abord, nous montrons qu'ignorer la contention sur le cache de second niveau (ou la mémoire) amène à largement surestimer les performances que l'on peut attendre et peut conduire à des conclusions fausses.

Nous explorons ensuite l'impact de différents paramètres de la hiérarchie mémoire. Nous montrons que le nombre de flots supporté doit être en accord avec la taille des caches, que les caches de premier niveau doivent être associatifs et que les lignes de caches doivent êtres petites.

Alors, les contraintes matérielles impliquées par la mise en œuvre d'une hiérarchie mémoire devraient limiter l'intérêt du *multiflot simultané* à quelques flots.

Mots-clé : multiflot, multiflot simultané, hiérarchie mémoire

1 Introduction

Simultaneous multithreading (*SMT*) appears to be a very promising approach for achieving higher performance through better processor utilization [1, 2]. An *SMT* architecture executes simultaneously several processes (or threads). To do so, the architecture relies on replicated context supports (Figure 1). A context support is implemented generally with a private register bank, a PC, status registers and eventually an instruction window. Thus, the processor can overcome the lack of instruction-level parallelism (ILP) of current programs by allowing parallel execution of instructions from different threads [3]. *SMT* should also allow to hide the latency of long operations, such as instruction or data cache misses, multiplications or divisions. When a thread is stalled, the fetch and issue bandwidths can be utilized by the other threads. Tullsen et al. [1, 2] have shown that an *SMT* architecture supporting 8 threads could achieve up to four times the instruction throughput of a conventional *superscalar* architecture with the same issue bandwidth.

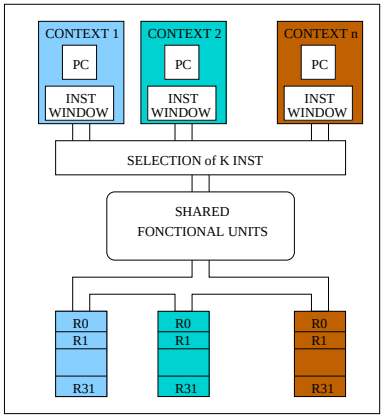

Figure 1: example of a simultaneously multithreaded architecture

However, the simultaneous execution of several threads puts a lot of stress on the memory hierarchy. The bandwidth offered by the second-level cache, either on-chip or off-chip, can be insufficient to allow the parallel execution of more than a few threads. Simulations conducted in [1, 2] were based on the SPEC92 benchmark suite, which is now widely known as being non representative of common *memory pressure* [5].

In this paper, we explore the intrinsic limitations of the memory hierarchy in an *SMT* environment. Our study focuses first on the stress put on the second-level cache. We first show that neglecting to simulate the contention on the second-level cache conducts to very misleading results. This is already true for singlethreaded architectures, however this is far more critical when the number of threads increases.

Then we examine the influence of different memory hierarchy parameters on the performance that an *SMT* microprocessor can achieve. We show that the number of threads supported by the hardware must be carefully chosen according to the cache size: for instance when using 32KB-64KB caches, implementing more than 4 threads will be counter productive. We also establish that on *SMT* processors, the first-level caches have to be set-associative and finally that the block-size has to be small (typically 16 or 32 bytes for a 16-byte bus).

The remainder of this paper is organized as follows. Section 2 presents the experimental framework. It details the simulated architecture and especially the memory hierarchy and the workloads used. Section 3 introduces the methodology. In Section 4, we present results of our simulations. Section 5 summarizes the study.

2 Simulation environment

To conduct this study, we relied on trace-driven simulations. This has the advantage of providing reproducible results. As our work focuses on cache-organization behavior, we therefore first present the simulated memory hierarchy.

2.1 Simulated Memory hierarchy

The design of memory hierarchies for microprocessors has been extensively studied [6, 7, 8]. However, very few studies have discussed the problem of cache bandwidth [9], or L2 cache contention [10]. All these studies were devoted to singlethreaded architectures.
To our knowledge, the only study available on caches and simultaneous multithreading is part of a work done by Tullsen et al. [1]. They showed that associating a private first-level cache with each thread is not interesting, as it would be equivalent (on memory side) to conventional multiprocessors sharing a second-level cache. There would not be any balance of the cache space used by threads according to their needs. With private caches, there is a waste of space when fewer threads are running. Thus, using shared first-level caches is better, except for high number of threads, where private instruction caches and a shared data cache should be preferred.
Tullsen et al. do not study the impact of all cache parameters on performance. Moreover, this study considered only the SPEC92 [17] benchmark suite. This suite is known to be inappropriate for cache evaluation, as the number of requests missing a small L1 instruction cache (and also in fact on the data cache) is particularly low and not realistic [5]. Moreover, the contention on the second-level cache is not evaluated.

We evaluate the impact of the principal cache parameters on the performance of an aggressive architecture supporting *simultaneous multithreading*. We also examine carefully the problem of contention on the second-level (L2) cache. Due to large L2 cache size (often 1 MBytes or more), contention on the memory (or a third-level cache) is far less critical (which however may not be the case for the DEC21164 which has a small 96KB L2 cache

[11]). We built a configurable simulator for a classical two-level memory hierarchy (Figure 2), as described in [6].

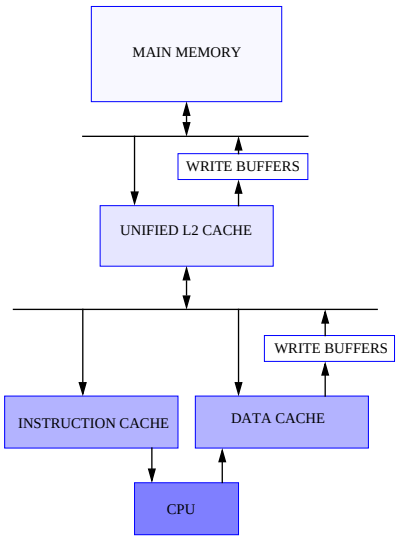

Figure 2: Two-Level memory hierarchy

We simulated a *write-back* scheme for the data cache and the L2 cache [12]. The write policy on miss is *write-allocate*. *Write-back* was preferred to *write-through*, because it induces less traffic on the buses. Moreover here, we do not care about parallelism and memory consistency. However, *write-back* slightly disadvantages long cache lines as a longer line is more likely to be dirty and induce a burst of traffic on the buses. A *write-through* policy should increase the average bus occupancy but could diminish the burst effect, which could result in an interesting tradeoff. *Write-through* scheme supposes a quite different memory hierarchy. This will be part of a future study.

As in recent out-of-order execution microprocessors, L1 and L2 caches are non-blocking [13], which means they do not stall the processor on a miss. Dedicated write ports are used for updating the caches after a miss. The servicing of L1 misses by the L2 cache is pipelined; as on most current out-of-order execution microprocessors (DEC21164, MIPS R10000, UltraSparc, ...) servicing of a L1 miss may begin while another L1 miss is still in progress. To limit the penalty on execution time, the missing word is always sent back first. To improve latency and reduce bandwidth consumption, multiple L1 cache misses that access the same block can be merged [11]. TLB is supposed never to miss. L1 caches are accessed with virtual addresses (i.e. TLB transactions and cache accesses are performed in parallel).
The bus width between the buffers and the L1 caches is 16 bytes. The default bus width

between L1 and L2 caches is set to 16 bytes. This corresponds to a large but realistic width (the size is 8 bytes on the Intel P6 and 16 bytes on the DEC 21164, the MIPS R10000 or the UltraSparc). Larger buses would have two important drawbacks. In case of an on-chip L2 cache, it would result in a higher power consumption and buses would occupy a large silicon area. For off-chip caches, a very high pin-out number would be required, which leads to higher costs.
The L1 caches are multi-ported [9]. In every cycle, up to two concurrent accesses to the instruction cache and 4 concurrent accesses to the data cache are possible. Fully multi-ported cache structure was considered. Such a structure may be considered as unrealistic, however we did not want to explore the spectrum of interleaving degree in caches. The results that are given here may then be considered as upper limits for speed-ups. Simulating a more realistic L1 cache structure would only reinforce our conclusions.
Instruction and data caches are supposed to have always the same size, same blocksize and associativity degree. As in most of current processor designs, we simulated a unified L2 cache.

2.2 Simulated architecture

This study focuses on the memory system behavior. To measure only the impact of this memory system on performance, other parameters were chosen in a very optimistic manner. First, we assumed no resource conflicts on functional units, i.e. one instruction may be executed as soon as its operands have become valid. Second, branche direction and target were supposed to be always correctly predicted. Hily and Seznec [15] showed that when the sizes of the prediction tables are sufficient, the prediction accuracy stays at the same level as for a conventional *superscalar* processor, as high as around 95% for two-level adaptive predictors.
Figure 3 illustrates the simulated architecture. An *SMT* architecture is very similar to a *superscalar* architecture [14] with the exception of the instruction fetch mechanisms which are described below.

In every cycle, a priority is affected to each thread for addressing the instruction cache and issuing instructions. To set up this priority, a simple round-robin algorithm is not effective. With such a simple scheme, the slowest thread has as much opportunities to issue instructions as the other threads. This thread fills the instruction pipeline with instructions that remain for a long time in the instruction window. The latter may become saturated with such instructions. The execution time of all the threads is then stuck to the slowest thread.
To avoid such a catastrophic behavior, the issue priority is determined by the number of instructions present in the static stages of the pipeline (i.e. waiting for execution) [2]. The lower this number, the higher the priority. Intuitively, a low number of instructions present for a thread gives a good opportunity of executing subsequent fetched instructions.
The instructions are fetched from the cache and kept into instruction buffers. Each buffer has a capacity of two cache blocks. In every cycle, up to 8 instructions are read in the

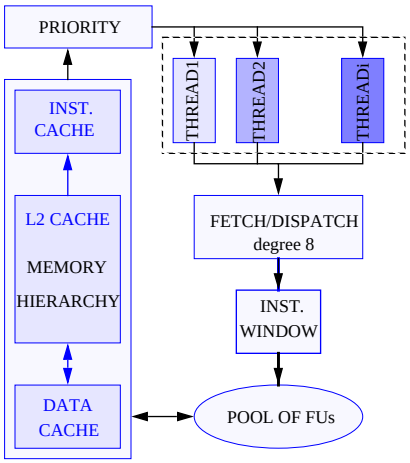

Figure 3: simulated architecture

instruction buffers (Figure 3). Following the calculated priorities, one fireable instruction is selected for each active thread. If all the issue slots are not filled and there are instructions left in the buffers, new selections are carried out. Thus, if only one thread is available, it should be able to fill all slots in the execution pipeline (i.e. 8 instructions may be selected from the same thread). When the number of instructions remaining in a buffer is smaller than the blocksize, a prefetch can be made. Figure 4 illustrates the instruction selection mechanism, for four threads and a 8-degree instruction pipeline. Such a powerful (but complex) fetch mechanism should not be unrealistic. Other studies, such as in [16], have proven that complex instruction fetch mechanisms are feasible.

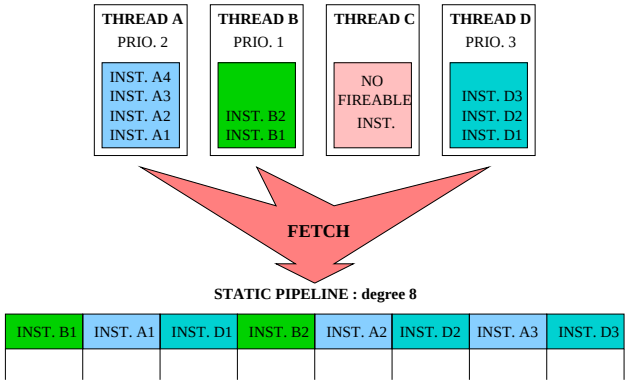

Figure 4: instruction selection

SMT, and especially our model, increases the complexity of certain tasks, such as the selection of the instructions from the buffers or the access to the register banks. To take this higher complexity into account, we assumed a 6-stage static pipeline, illustrated in Figure 5.

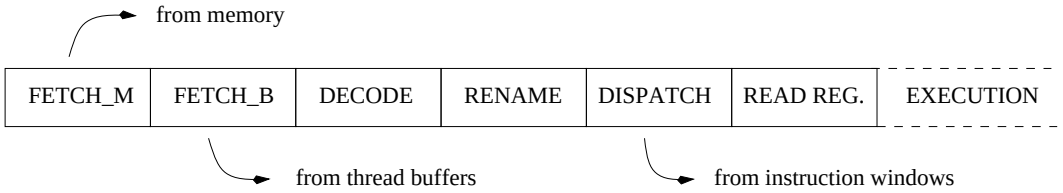

Figure 5: static instruction pipeline

2.3 Workloads

SPEC benchmarks [17] have been widely used for microprocessor simulations. However, different studies have shown that this benchmark set exhibits very low cache miss rates, and so is poorly suited for cache simulation. Limiting the study to user-level instructions usually leads to very optimistic results. The IBS (Instruction Benchmark Suite) is more representative of real workload than the SPEC one [5, 4], and offers good balance between instruction and data misses (as illustrated further in figure 8). The IBS is a set of traces including user-level and kernel-level codes. The eight traces that we used from the IBS are illustrated in table 1.

gs	ghostscript (version 2.4.1). Interprets and displays a single postscript page with text and graphics under X-window.
jpeg_play	xloadimage program displaying two JPEG images.
mpeg_play	mpeg_play program displaying 85 frames from a compressed video file.
nroff	Ultrix text formating program.
real_gcc	the GNU C compiler.
sdet	multiprocess, system performance benchmark from the SPEC SDM benchmark suite.
verilog	verilog-XL simulating the logic design of an experimental processor.
video_play	a modified mpeg_play program displaying 610 frames from an uncompressed video file.

Table 1: IBS workloads used in the simulation

All the benchmarks were compiled with the Ultrix MIPS C 2.1 compiler using -O2 optimizations, with Ultrix version 3.1 from Digital Equipment Corporation. In graphs illustrating the remainder of the paper, the name of the workloads are set as the concatenation of the

first two letters of the executed applications. *gsmp*, for instance, refers to *gs* and *mpeg_play* running simultaneously.
Our study has been limited to multiprogramming environment, i.e. execution of independent programs. First, such a workload put a greater stress on the memory hierarchy than a parallel workload. Second, we did not have access to traces of parallel applications including user-level and kernel-level activities.

3 Methodology

We performed simulations by varying memory-hierarchy parameters while executing 1, 2, 4 or 6 threads simultaneously. Distinct programs are assigned to the threads in the processor. For each simulation, we warm the memory hierarchy with the execution of 1 million instructions per executing thread. Ten million of instructions time the number of threads is then simulated. This is sufficient given the structure of the memory hierarchy. Moreover, we do not look at the absolute performance of *SMT* architectures on an application, but we want to evaluate their relative behavior while varying different parameters of the memory hierarchy. The default parameters used in this study are listed in table 2 (*na* means *not applicable*).

	Inst. cache	Data cache	L2 cache	Memory
Size	32KB	32KB	1MB	na
Line size	32 bytes	32 bytes	64 bytes	na
Associativity	4	4	4	na
Latency (cycles)	1	1	5	30
Transfer time (cycles)	1	1	2	4

Table 2: default parameters of memory hierarchy

4 Simulation Results

The simplest measure for representing the behavior of a memory hierarchy on a program is the miss rate. However, the execution of instructions from independent threads, out-of-order, with non-blocking caches, allows to maintain the instruction pipeline and the functional units busy. The miss rates have now far less significance: a miss on the cache does not imply effective penalty for the execution [10]. For our evaluation, which is mostly oriented towards caches, with a very efficient fetch mechanism, the number of instructions executed per cycle (*IPC*) is a good indicator of program behavior and will be used as performance metric in the remainder of the paper.

4.1 Contention on L2 cache cannot be ignored

4.1.1 Contention on L2 cache with a single thread

We first simulated different conventional configurations of memory hierarchy while executing a single thread, to have clues on the behavior of our benchmark set. Figure 6 illustrates the average obtained IPC, with a 32KB cache and various block sizes and associativities. The other parameters are set up to their default values. The key *32K-lxxx-ay* refers to a cache with blocks of *xxx* bytes and associativity degree *y*.

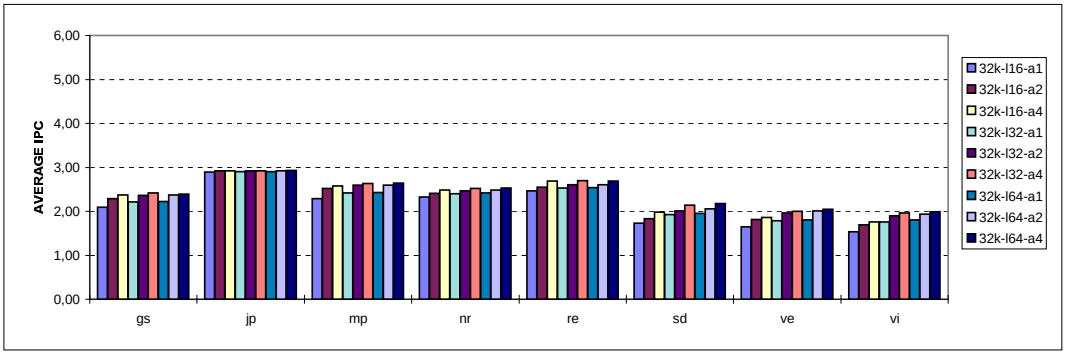

Figure 6: average IPC for 1 thread, L2 contention taken in account

As expected for conventional non-scientific applications, the IPC observed is between 1.5 and 3 [18]. The average number of miss per instruction (MPI) is given in Figure 7. The graph takes into account the instruction as well as the data misses (the rate of instruction misses over the total number of misses is given in Figure 8). The unified L2 cache services L1-cache misses, as well as write transactions and sometimes coherency operations. The average occupancy of the L2 cache and the memory are represented in Figures 9 and 10. The average occupancy can be viewed as the fraction of time the L2 cache (resp. the memory) is busy processing accesses from the L1 cache (resp. the L2 cache). Due to low values, the scale in Figure 10 has been set to one third of the scale used in 9. The occupancy is quite low for the best-performing *jpeg_play* which is essentially constituted by a single loop and fits well in the L1 caches. For applications with larger working set, such as *verilog* or *video_play*, the average L2 cache occupancy can be as high as 64%. One has to keep in mind that these are only average values, and only give an indication of how good or bad things are: contention, i.e. conflicts between accesses, may occur on the L2 cache even when on average, the L2 cache occupancy is low. However, the correlation between the measured performance (Figure 6) and the occupancy (illustrated Figure 9) is quite clear. The higher the occupancy, the higher is the contention and the lower is the performance. There is no clear threshold after which the performance loss becomes dramatic. The L2 cache "could" be busy 100% of the time by ideally spaced out requests without resulting in added

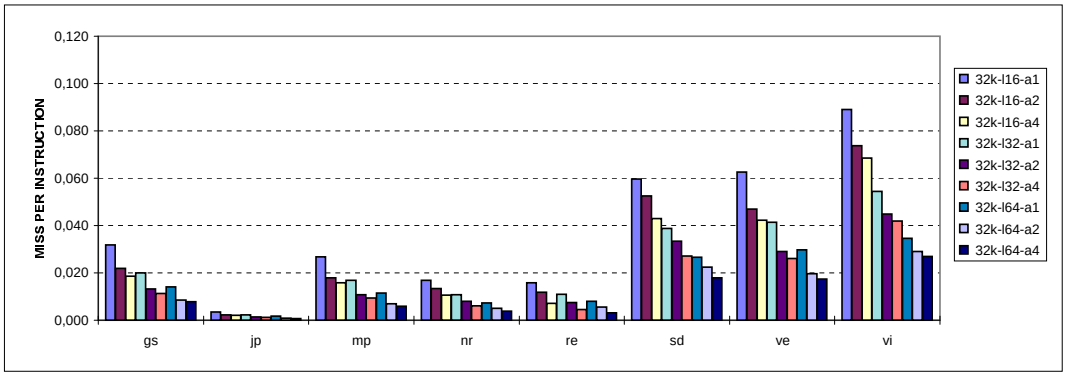

Figure 7: average miss per instruction on L1 caches for 1 thread

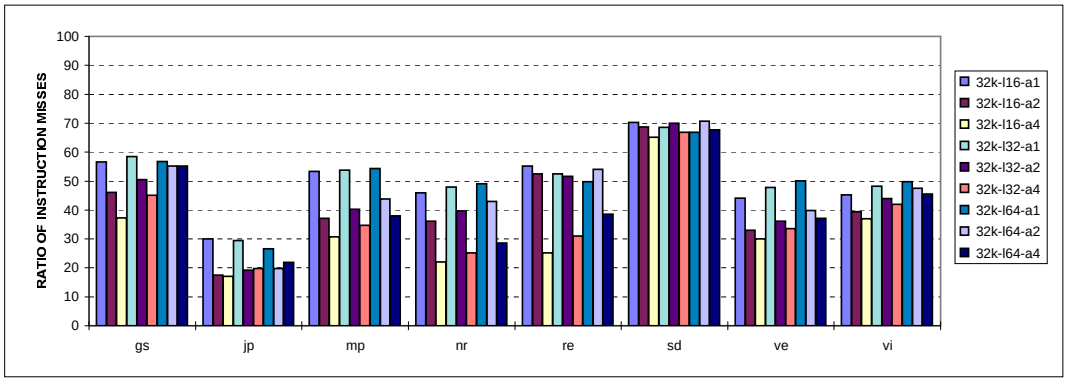

Figure 8: ratio of instruction misses over total number of misses on L1 caches for 1 thread

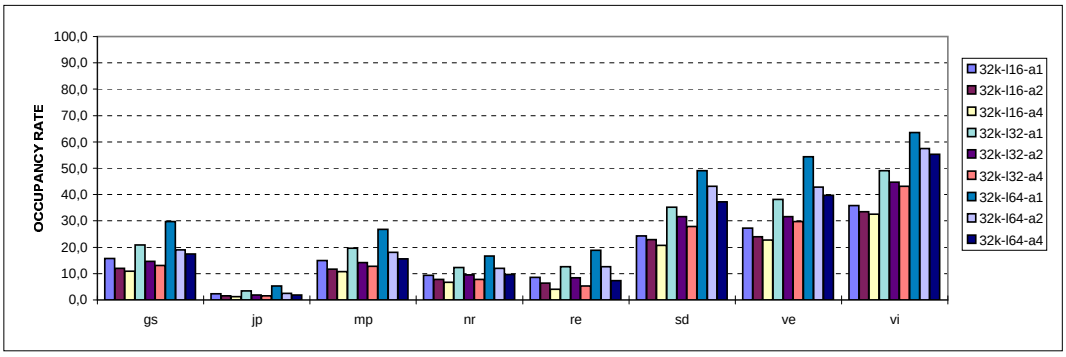

Figure 9: average L2 cache occupancy rate for 1 thread

penalties. On the other hand, the L2 cache could be busy 10% of the time, but by bursts of

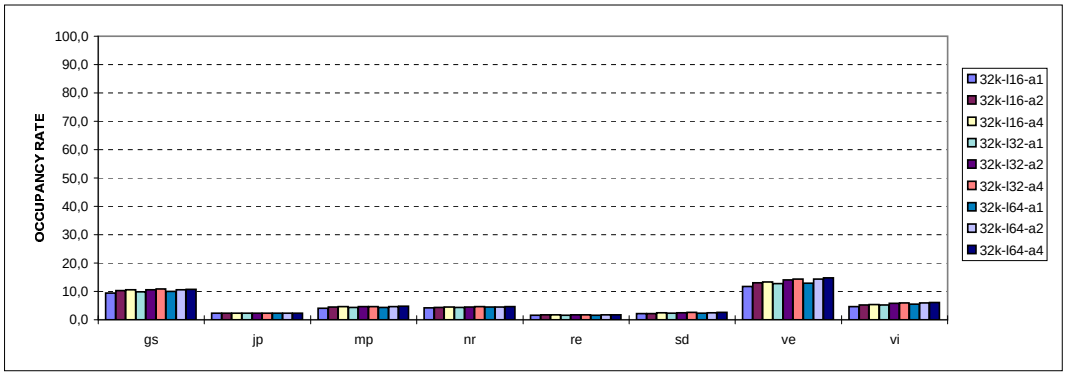

Figure 10: average memory occupancy rate for 1 thread

conflicting requests.
By comparing Figure 6 and 7, one can notice that the IPC is not directly correlated to the MPI. For example, *vi* (32K-l32-a4) has a higher IPC than *vi* (32K-l64-a1), but *vi* (32K-l64-a1) shows a lower MPI. This is true for several other data points. Longer lines result in less misses but, as illustrated by Figure 9, result in higher latencies due to cache occupancy, hence the lower IPC. To evaluate more clearly the impact of contention on single-thread performance, we conducted simulations assuming no contention on the L2 cache (i.e. on every miss, the missing word comes back after 5 cycles) (Figure 11).
For *verilog*, for example, whatever the cache configuration, the IPC is over-estimated by a factor of around 30%. The over-estimation varies between 13% and 28% for *video_play*, and between 14% and 20% for *gs*. These results clearly show the necessity to take L2 cache contention into account when simulating singlethreaded architectures.

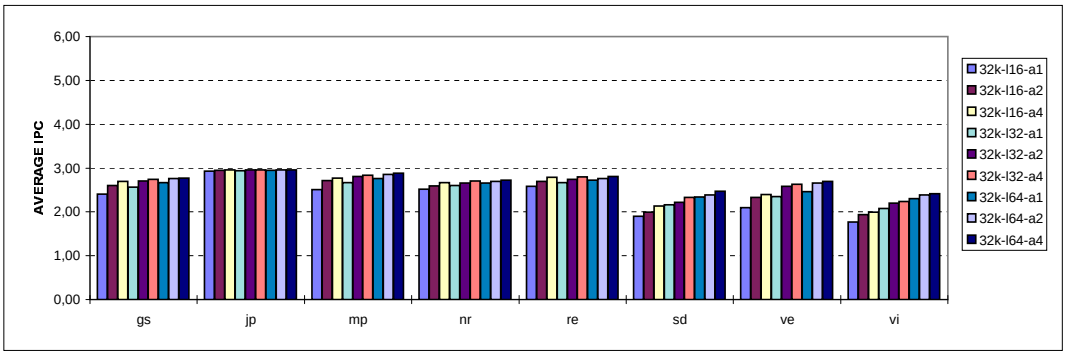

Figure 11: average IPC for 1 thread, assuming no contention on L2 cache

4.1.2 Contention on L2 cache on *simultaneous multithreading*

The impact of contention on performance should be more important as the number of threads executing simultaneously increases. To explore this, we ran simulations with 2, 4, and 6 threads running together. Figure 12 shows the average IPC obtained for eight different groups of benchmarks, for four threads. We used the same default hierarchy as previously described in Table 2 (32KB L1 caches, 1MB L2 cache, ...).
The respective performances of our various groups of benchmarks are very different. The peak instruction parallelism is 8, which corresponds to the static pipeline width (and to the peak fetch bandwidth). The best performing benchmarks hardly exhibit IPC higher than 5. In the worst case (*resdvevi*, for a 32KB direct-mapped cache and a block size of 64 bytes), the IPC is only around 2 (*ve* and *vi* are two of the most L2 bandwidth consuming applications in our benchmark set). This IPC is lower than for *real_gcc*, and just slightly higher than for the three other programs, if they were running alone. This indicates that in some case *simultaneous multithreading* leads to bad (or not interesting) performances.
Figures 13 and 14 illustrate the average L2 cache and memory occupancies during the si-

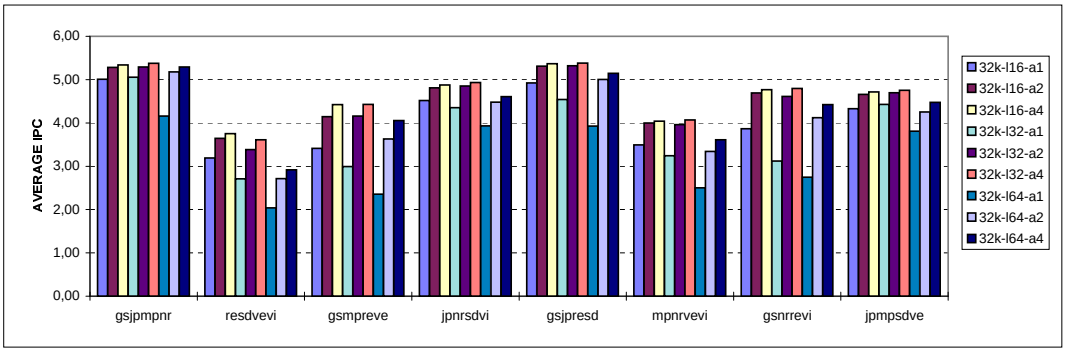

Figure 12: average IPC for 4 threads, L2 cache contention taken in account

mulations. The occupancy rates have to be compared with Figures 9 and 10. The increase is dramatic. For six of the eight groups of benchmarks, the occupancy exceeds 50% whatever be the cache configuration. In several cases, it has nearly reached 100%. This has been possible because a miss will result in only partially stalling a thread, while the other threads may continue the execution. Except for one group of benchmarks for a single hierarchy configuration, the occupancy is always higher than 30%.
Higher occupancy implies higher contention. As previously, we can observe here a strong correlation between the performances of the groups of benchmarks relative to the cache configurations, and the occupancy. Figure 15 shows that the same simulations realized without caring for contention on L2 cache give far better results than in Figure 12. For all the groups of benchmarks, the IPC is between 5 and 6. The over-estimation of performance is larger than for a single thread. This over-estimation varies, but often exceeds 100%, and reaches 174% for *resdvevi*, for a 32KB direct-mapped cache with a block size of 64 bytes.

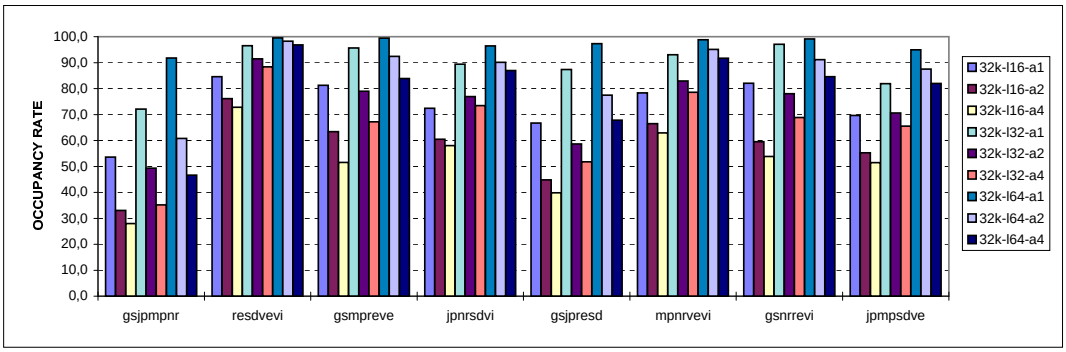

Figure 13: average L2 cache occupancy rate for 4 threads

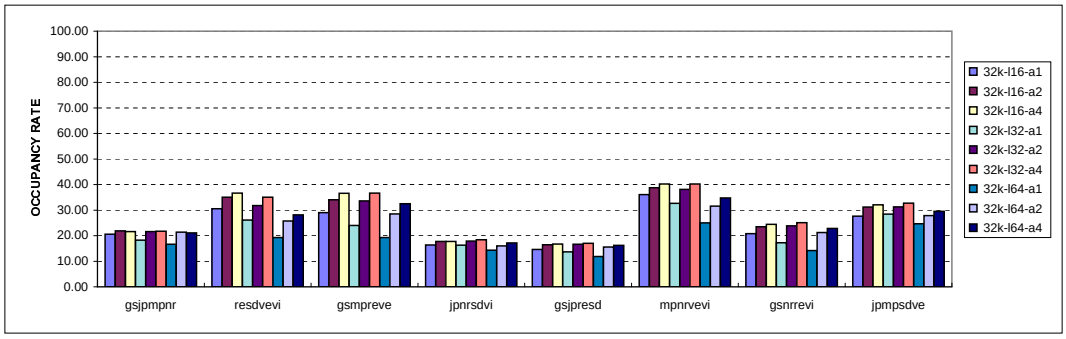

Figure 14: average memory occupancy rate for 4 threads

Moreover, when the contention is ignored, simulation results for different cache configura-

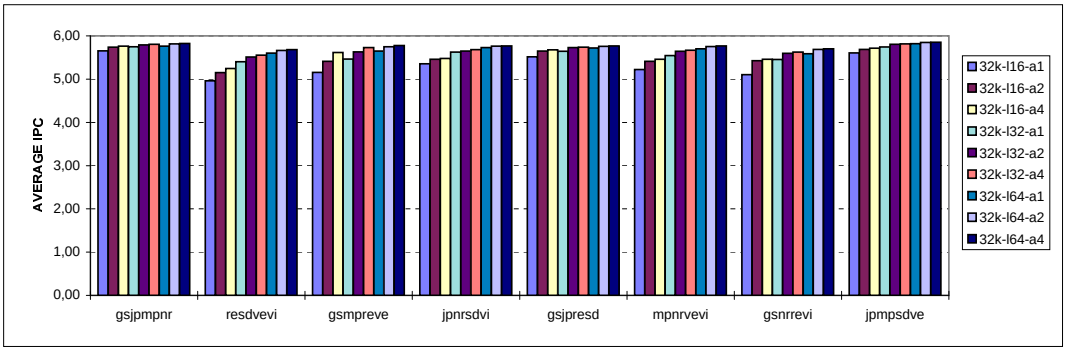

Figure 15: average IPC for 4 threads, assuming no contention on L2 cache

tions can lead to misleading conclusions. Thus when ignoring L2 cache contention for all benchmarks, increasing block size and associativity allows higher performances. A look at Figure 12 proves this to be generally wrong. Finally, Figure 15 shows little variation in IPC between different mixing of threads. In contrast, Figure 12 shows a much wider variation. This suggests that the mixing of threads has little impact on small L1 caches, but has a higher impact on large L2 caches.

The average miss per instruction on the L1 caches (MPI) when 4 threads are executing simultaneously is illustrated in Figure 16. The behaviors of the L1 caches are quite different from the ones with a single thread. Now, for instance a larger cache block does not systematically means a lower miss ratio when the associativity is not sufficient.
To evaluate the impact of the simultaneous execution of four threads on the L1 cache, we have calculated the MPI that is obtained when four applications constituting the different groups of benchmarks run sequentially. Figure 17 shows the average MPI for the sequential and the parallel executions for 32KB 2-way associative L1 caches with varying blocksize. The number of misses on the parallel execution is approximately the double of a sequential execution. This explains the increase in contention on the L2 cache.

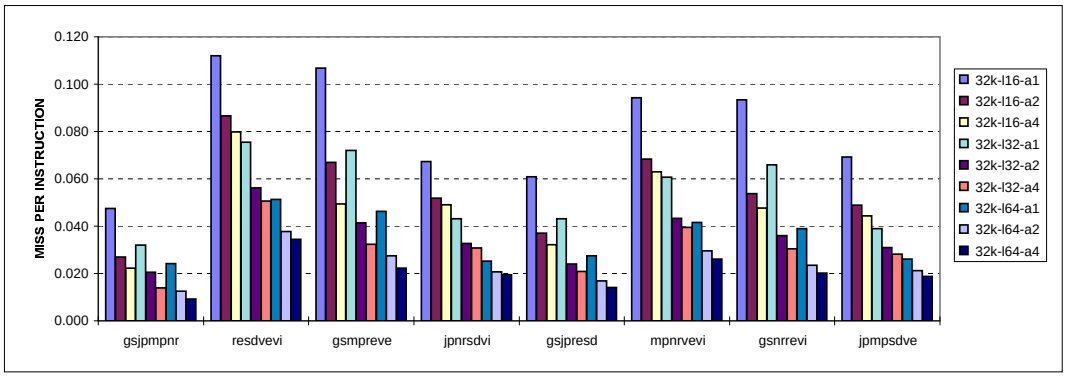

Figure 16: average miss per instruction on L1 caches for 4 threads

4.1.3 Summary

L2 cache contention has a large impact on performance for singlethreaded architecture. We have established that this impact is even larger when several threads are executing simultaneously. Ignoring L2 cache contention and also memory access on *SMT* would conduct to misleading results.
The remainder of this study will explore in depth the impact of the size, the associativity, and the block size of the memory hierarchy on the performance of an *SMT* architecture.

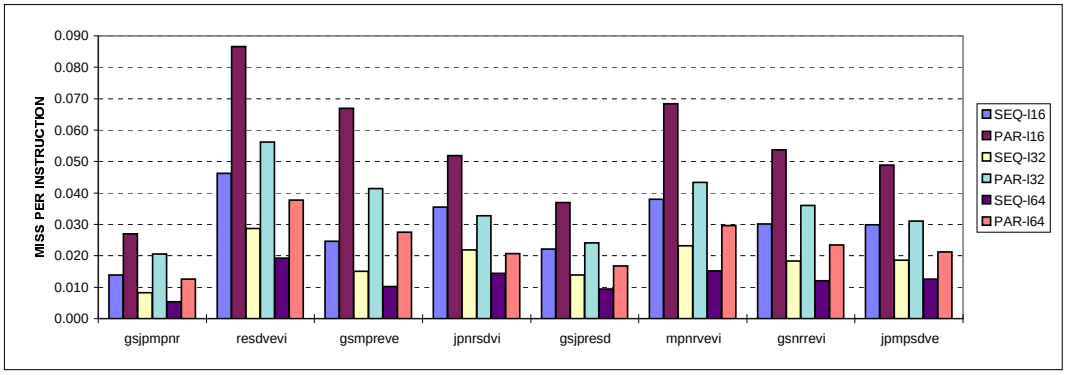

Figure 17: comparison between L1 miss rates for sequential and parallel execution of 4 threads

4.2 Cache size

The access time to the L1 caches has a major impact on the processor cycle time [8]. Large caches, in term of capacity, imply low cycle time. Other factors also have an impact on the cache access time. One of them is the number of ports available for reading, writing or updating the data in the cache. The higher the number of access ports, the higher the cache access time [12].
In this section, we examine the behavior of cache hierarchy when up to six threads are executing simultaneously, by fixing instruction and data cache sizes to successively 16KB, 32KB and 64KB bytes. The other cache parameters are set to the default ones (32-byte blocks, 4-way associativity,...).
As seen in the previous section, applications in the benchmark set have a large range of memory behavior. For instance *jpeg_play* do not exhibit much L2 and main memory traffic. On the other hand, *verilog* and *video_play* run on larger workloads and induce more traffic.

The impact on the performance of the cache size is illustrated in Figure 18, through the resulting average IPC. Each bar (IPC) represents the same number of applications executed either sequentially (for one thread) or in sequences of groups of benchmarks, a group of benchmark consisting in the parallel execution of 2, 4 or 6 programs. Each of our 8 benchmarks have the same weight for the four considered configurations.

For a single thread, 4-way set-associative 32KB caches are sufficient for most of our applications. Caches of this size are not the real limiting factor of the performances. The benefit from shifting from 16KB to 32KB allows a gain of 12% on the IPC. A shift from 32KB to 64KB only enhances the IPC by 4%.
For several threads, this difference in performance is more important. Thus, for 2 threads, shifting to 32KB cache brings a net gain of 13%. A shift to 64KB is still interesting, with a better performance of nearly 10%.
As expected, the cache size is a much more determining parameter for 6 threads. The shift

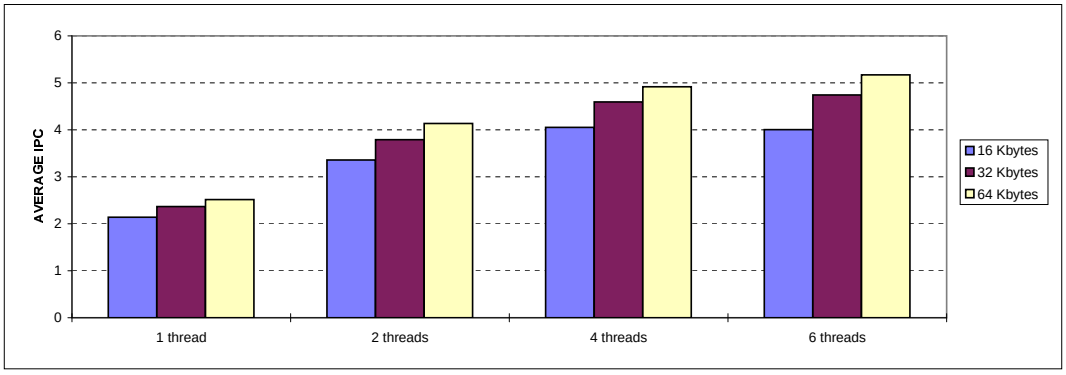

Figure 18: average IPC obtained with 16KB, 32KB and 64KB bytes L1 caches

from 16KB to 64KB allows a gain of nearly 30%, which represents more than one added instruction executed per cycle.
The L1 caches are shared among the running threads, which allows some kind of dynamic adaptation of the available space to the threads' needs (especially because we assumed here associative caches). However, this is true only up to a certain limit; when too many "memory eager" threads are running in parallel, the caches are permanently saturated. The number of requests missing on the L1 cache results in increasing contention. Thus, as illustrated by Figure 19, for a 32KB L1 cache size with 6 threads running together, occupancy is higher than 80% for most of the groups of benchmarks. Conversely, with 64KB caches, occupancy is under 70% for most of the benchmarks.

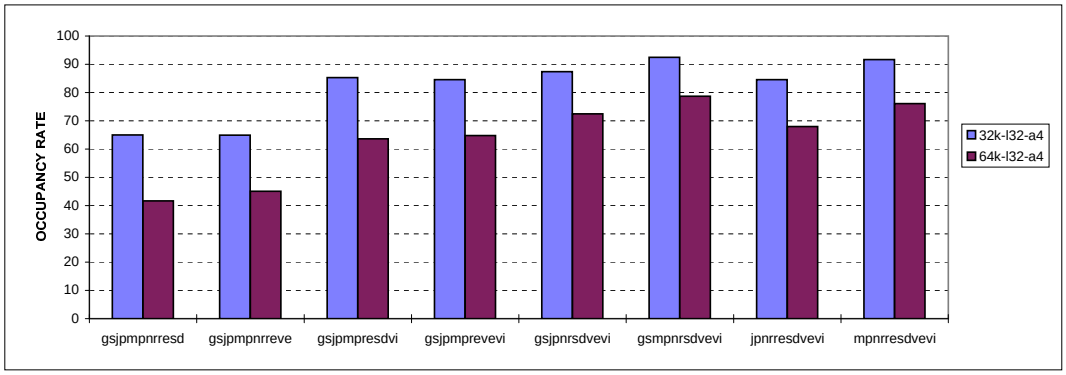

Figure 19: average L2 cache occupancy for 6 threads, with 32KB and 64KB L1 caches

This emphasizes the importance of tuning well the number of threads to the cache size. When the cache is too small, the benefit of executing a larger number of threads is tiny.

An increase in the number of simultaneously supported threads should be accompanied by an increase of cache size. However, in modern microprocessors, the size of the L1 cache is among one of the sensitive parameters which fixes the maximum clock cycle. A larger cache implies a lower clock frequency. Moreover, to get these performances, we allowed respectively two and four accesses per cycle to the instruction and data caches. The added hardware complexity supposes an even more extended cycle time [19].
Within the parameter range that we explored, whatever be the cache size, supporting 6 threads does not bring a real speed-up compared to 4 threads, and is not worth the added hardware complexity.

4.3 Associativity

The associativity degree of L1 cache has been extensively studied for singlethreaded microprocessors. Increasing the associativity allows to reduce the miss ratio, but extends the access time [6]. Figure 20 illustrates the average IPC for 64KB L1 caches for various associativities and block sizes.

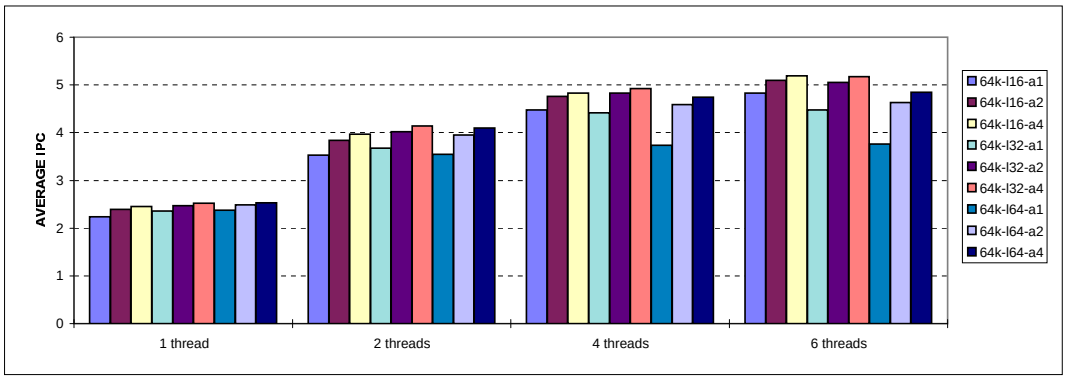

Figure 20: average IPC with varying associativity and block size of 64KB L1 cache

We notice that, as for one thread, a higher associativity degree allows a higher performance when several threads are used. However, the phenomenon is emphasized when the number of threads increases. The main reason for this is that the memory references generated by the simultaneous execution of independent threads exhibit less spatial locality than that of a single thread. Conflict misses have then a higher probability to happen than for a single thread. Increasing the associativity allows to limit conflict misses [6]. As illustrated in Figure 21, this appears far more critical when the L1 cache size is small.

4.4 Block size

Figures 20 and 21 also show that the choice of the block size is even more critical than the choice of associativity degree. For a single thread, using a 64-byte block-size (four times the

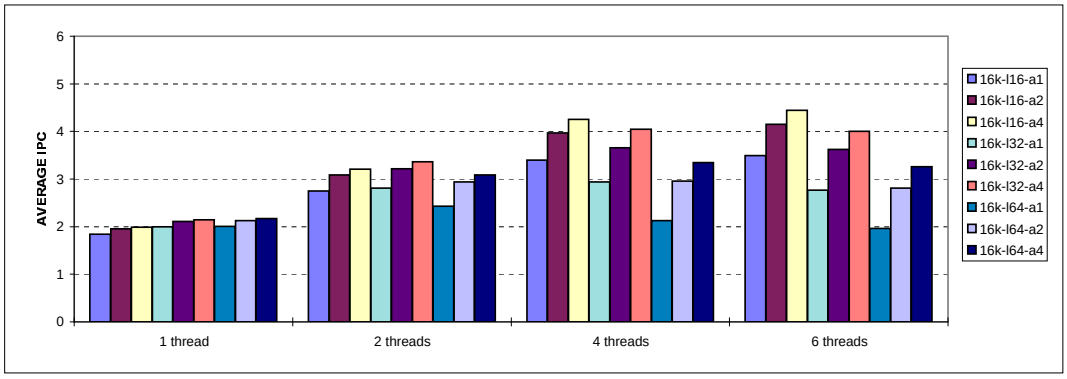

Figure 21: average IPC with varying associativity and block size of 16KB L1 cache

bus width) results in roughly the same performance as a 32-byte blocksize and only slightly higher performance than the 16-byte one. When the number of threads increases, this is no longer true. For two threads, using a 64-byte block already leads to worse performance than using 16-byte or 32-byte blocks. This comes mostly from the contention on the L2 cache, as illustrated in Figure 13.

The block size's impact on L2 cache is different than previously observed for the associativity or for the cache size. When the associativity degree or the cache size are increased, the L1 caches miss ratio decreases. This results in less accesses to the L2 cache. Likewise, when the block size increases, it results in a small decrease of the miss ratio and so less transactions on the L2 cache. However, each transaction keeps busy the L2 cache for a longer period, thus potentially delaying other miss servicing. The induced extra penalty widely exceeds the benefit of a lower miss ratio. Indeed, the time a transaction keeps the L2 cache busy is double if the block size is doubled.

Figure 22 illustrates this problem for L1 cache blocksizes of 16 bytes in (A) and 32 bytes in (B). The L1 cache has two access ports. The bus width between L2 and L1 caches is 16 bytes, and the bus is busy two cycles for each block transfered. The reasoning would be the same for more complex caches, or with the instruction cache.

We consider several loads from 3 different threads (A, B, and C). If we assume that the load

CYCLE	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
L1	Ld A1		Ld B1 Ld A2	Ld C1		Ld B2	Use B2 Use A1		Use B1		Use A2		Use C1							
L2					Ld A1	Ld A1	Ld B1	Ld B1	Ld A2	Ld A2	Ld C1	Ld C1								

(A) 16-byte blocks

CYCLE	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
L1	Ld A1		Ld B1 Ld A2	Ld C1		Ld B2	Use B2 Use A1				Use B1				Use A2				Use C1	
L2					Ld A1	Ld A1	Ld A1	Ld A1	Ld B1	Ld B1	Ld B1	Ld B1	Ld A2	Ld A2	Ld A2	Ld A2	Ld C1	Ld C1	Ld C1	Ld C1

(B) 32-byte blocks

Figure 22: Example of caches transactions for 16-byte and 32-byte blocksizes

B2 is the only load to hit into the L1 cache, in the case (A) (blocks of 16 bytes), data *A1* is available for use in cycle 7, *B1* in cycle 9, *A2* in cycle 11 and *C1* in cycle 13. With 32-byte blocks (B), data *B1* is available in cycle 11, *A2* in 15 and *C1* only in cycle 19. Moreover, if a *write-back* occurred, it would generate a burst of traffic which would result in added occupancy of the L2 cache. This illustrates the stress put on the L2 cache when blocks are large.

As the number of threads increases, this phenomenon is amplified, and the L2 cache becomes a critical resource. The memory requests which miss in the L1 caches are more and more delayed, and this results in added stalls of the instruction pipeline. Adding more threads makes this even worse and can conduct to lower performances. Thus, as illustrated in Figures 20 and 21, average IPC with a cache block of 64 bytes can be lower for 6 threads than for 4 threads. The problem of contention is such for 6 threads that even with 32-bytes cache blocks, the performances are slightly lower than for 16 bytes ones.
As for the size of the cache and the associativity, the block size has to be chosen with care. Small block sizes must be considered because they give better results when the number of threads increases. Moreover, the penalty they induce when fewer threads are available for execution is very small.

In the simulations illustrated in Figure 23, we have set the size of the L1 caches proportionally to the number of threads. Thus, for one thread, we have L1 caches of 16KB, and for 4 threads, the size is 64KB. This shows that the phenomenon described previously involving

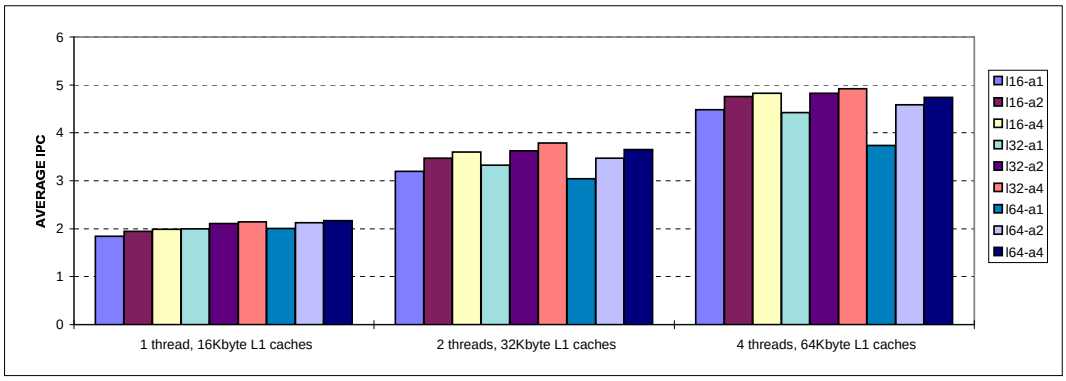

Figure 23: making varying associativity and block size, L1-cache size proportional to the number of threads

the associativity and the block size are mostly dependent of the number of threads, and are amplified when the size of the L1 caches is small.

5 Concluding Remarks

While designing an *SMT* architecture, one has to be very careful with the choice of a memory hierarchy. In this study, we showed that contention on the L2 cache cannot be ignored. First, because even for singlethreaded architectures, it leads to strongly over-estimate the IPC you can expect. For multithreaded architecture, it is even more dramatic. One of our groups of benchmarks executing 4 threads simultaneously exhibited an over-estimation of 174% of the IPC, and for other groups of benchmarks, values over 100% were not rare. Moreover, we showed that for instance on the choice of block size, not simulating L2 cache contention can lead to incorrect conclusion, as several behaviors having negative impact on performance are not shown.

We showed that the maximum number of supported threads has to be set up according to the cache size. When a small L1 cache is used, the traffic to the L2 cache generated by extra threads rapidly degrades performance. Executing more threads can then dramatically pull down the overall system performance. The associativity degree of the L1 cache is also very important: the L1 cache has to be associative. Impact of associativity was shown to be far more important for *SMT* than for singlethreaded architecture. Finally we showed that small L1 cache blocks have to be used. This is already true for singlethreaded architectures, but due to L2 cache contention, this is essential for simultaneously multithreaded architectures. In our simulations, using a 32-byte and especially a 64-byte block size leads to lower performance than a 16-byte block size.
A bad choice of the memory hierarchy parameters can result in very bad behavior when the number of threads increases. Thus for example when using a direct-mapped 32KB cache and a 64-byte block size, performance is significantly lower for 6 threads than for 4 threads. However, if keeping small cache blocks does not induce serious hardware difficulties, the size and the associativity of L1 caches are more sensitive parameters. Their combination generally fixes a microprocessor's cycle time. The larger the cache, the longer is the cache access time. Using associativity may lead to the use of longer cycle or extra load use delay (3 cycles for instance on the DEC21264). The implementation of support for several threads has also a hardware cost which should also be prohibitive compared to the small gain in performance that a shift from 4 to 6 threads can offer.
Thus, for general purpose microprocessors supporting a conventional memory hierarchy and running IBS like workloads, simultaneous multithreading is promising but should not be interesting for the execution of more than a few (let's say 4) threads. Moreover, recent advances in research on superscalar structure have suggested new possibilities for performance on singlethreaded architecture (fetching two basic blocks per cycle [26, 25], predicting or anticipating load addresses or even operation results [21, 22, 23, 24]). As this study has pointed out that the L2 cache is already a major bottleneck when several threads execute simultaneously, multithreaded execution on an *SMT* will not take the same advantage of these techniques as a singlethreaded execution.

References

[1] D. M. Tullsen, S. J. Eggers, and H.M. Levy. Simultaneous Multithreading: Maximizing On-Chip Parallelism. In ACM, editor, *22nd International Symposium on Computer Architecture*, pages 392–403, June 1995.

[2] 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 *23th Annual International Symposium on Computer Architecture*, May 1996.

[3] P. Lenir, R. Govindarajan, and S. S. Nemawarkar. Exploiting Instruction-Level Parallelism: The Multithreaded Approach. In *MICRO 25*, pages 189–192, December 1992.

[4] D. Nagle, R. Uhlig, T. Stanley, S. Sechrest, T. Mudge, and R. Brown. Design Tradeoffs for Software-managed TLBs. In *20th International Symposium on Computer Architecture*, pages 27–38, May 1993.

[5] R. Uhlig, D. Nagle, T. Mudge, S. Sechrest, and J. Emer. Instruction Fetching: Coping With Code Bloat. In *22nd International Symposium on Computer Architecture*, June 1995.

[6] S. A. Przybylski. *Cache and Memory Hierarchy Design : A Performance-Directed Approach.* Morgan Kaufmann, 1990.

[7] N. P. Jouppi. Improving Direct-Mapped Cache Performance by the Addition of a Small Fully-Associative Cache and Prefetch Buffers. In ACM, editor, *17th International Symposium on Computer Architecture*, pages 364–373, June 1990.

[8] N. P. Jouppi and S. J. E. Wilton. Tradeoffs in Two-Level On-Chip Caching. In ACM, editor, *21st International Symposium on Computer Architecture*, pages 34–45, April 1994.

[9] G. S. Sohi and M. Franklin. High-Bandwidth Data Memory Systems for Superscalar Processors. In *ASPLOS*, pages 53–62, April 1991.

[10] A. Seznec and F. Loansi. About Effective Cache Miss Penalty on Out-Of-Order Superscalar Processors. Technical report IRISA-970, November 1995.

[11] Digital Equipment Corporation, editor. *Alpha 21164 Microprocessor Hardware Reference Manual.* September 1994. preliminary edition.

[12] D. Patterson and J. Hennessy. *Computer Architecture : A Quantitative Approach.* Morgan Kaufmann publishers, 1990.

[13] D. Kroft. Lockup-Free Instruction Fetch/Prefetch Cache Organization. In *Proc. of hte 8th International Symposium on Computer Architecture*, pages 81–87, May 1981.

[14] H. Hirata, K. Kimura, S. Nagamine, Y. Mochizuki, A. Nishimura, Y. Nakase, and T. Nishizawa. An Elementary Processor Architecture With Simultaneous Instruction Issuing from Multiple Threads. In *19th International Symposium Computer Architecture*, pages 136–145, May 1992.

[15] S. Hily and A. Seznec. Branch Prediction and Simultaneous Multithreading. In *International Conference on Parallel Architecture and Compilation Techniques*, 1996.

[16] T. M. Conte, K. N. Menezes, P. M. Mills, and B. A. Patel. Optimization of Instruction Fetch Mechanisms for High Issue Rates. In *22nd Annual International Symposium on Computer Architecture*, June 1995.

[17] K.M. Dixit. New Cpu Benchmark Suites from Spec. *COMPCON*, pages 305–310, spring 1992.

[18] N.P. Jouppi and D.W. Wall. Available Instruction-Level Parallelism for Superscalar and Superpipeline Machines. In ACM Press NewYork, editor, *ASPLOS*, pages 272–282, 1989.

[19] S. Jourdan, P. Sainrat, and D. Litaize. Exploring Configurations of Functional Units in an Out-Of-Order Superscalar Processor. In *22nd International Symposium on Computer Architecture*, pages 117–124, June 1995.

[20] A. J. Smith. Line (Block) Size Choice for CPU Cache Memories. *IEEE Transactions on Computers*, C-36(9):1063–1075, September 1987.

[21] T.M. Austin, G.S. Sohi, "Zero-cycle loads: Microarchitecture Support for Reducing Load Latency", *Proceedings of the 28th Annual International Symposium on Microarchitecture*, November 1995.

[22] M.H. Lipasti, C.B. Wilkerson, J.P. Shen "Value Locality and Load Value Prediction", *Proceedings of 7th international conference on Architectural Support for Programming Languages and Operating Systems*, October 1996.

[23] M.H. Lipasti, J.P. Shen, "Exceeding the Dataflow Limit Via Value Prediction",*Proceedings of the 29th Annual International Symposium on Microarchitecture*, December 1996.

[24] Y. Sazeides, S. Vassiliadis, and J.E. Smith, "The Performance Potential of Data Dependence Speculation and Collapsing", *Proceedings of the 29th Annual International Symposium on Microarchitecture*, December 1996.

[25] A. Seznec, S.Jourdan, P. Sainrat, P. Michaud, " Multiple-Block Ahead Branch Predictors", *Proceedings of 7th international conference on Architectural Support for Programming Languages and Operating Systems*, October 1996.

[26] T. Yeh, D. T. Marr, and Y. N. Patt, "Increasing the Instruction Fetch Rate via Multiple Branch Prediction and a Branch Address Cache," *Proceedings of the 7th ACM International Conference on Supercomputing*, July 1993.

Unit´e de recherche INRIA Lorraine, Technopôle de Nancy-Brabois, Campus scientifique,
615 rue du Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY
Unit´e de recherche INRIA Rennes, Irisa, Campus universitaire de Beaulieu, 35042 RENNES Cedex
Unit´e de recherche INRIA Rhône-Alpes, 655, avenue de l'Europe, 38330 MONTBONNOT ST MARTIN
Unit´e de recherche INRIA Rocquencourt, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex
Unit´e de recherche INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex

Éditeur
INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
http://www.inria.fr
ISSN 0249-6399