

Performance Effects of a Cache Miss Handling Architecture in a Multi-core Processor

Magnus Jahre

Lasse Natvig

Department of Computer and Information Science (IDI), NTNU

{jahre, lasse}@idi.ntnu.no

Abstract

Multi-core processors, also called Chip multiprocessors (CMPs), have recently been proposed to counter several of the problems associated with modern superscalar microprocessors: limited instruction level parallelism (ILP), high power consumption and large design complexity. However, the performance gap between a processor core and main memory is large and growing. Consequently, multi-core architectures must invest in techniques to hide the large memory latency. One way of doing this is to use *non-blocking* or *lockup-free* caches. The key idea is that a cache can continue to service requests while one or more misses are being processed at a lower memory hierarchy level. This technique was first proposed by Kroft [16].

The main contribution of this paper is the observation that a non-blocking cache must fulfill two functions. Firstly, it should provide sufficient miss parallelism to speed up the applications. Secondly, the number of parallel misses should not be so large that it creates congestion in the on-chip interconnect or off-chip memory bus. While the first function is well known, the other is a result of the possibility for contention when multiple processors are placed on a single chip. A compromise *miss handling architecture (MHA)* evaluated in this work which handles 16 parallel misses in the L1 cache, has an average speed-up of 47 % compared to a blocking cache and has a hardware cost of 9 % of the L1 cache area.

1 Introduction

The performance gap between the processor and memory has been steadily increasing since the beginning of the 80's [14]. The result is a large performance gap which is sometimes referred to as the *Memory Wall*. To achieve high performance, there is a need to hide the memory latency. Because programs exhibit both *spatial* and *temporal* locality, caches are a widely adopted way of dealing with this problem. In this way, the illusion of a large and fast memory is created.

Chip Multiprocessors (CMPs) have recently become popular in both the commercial and research communities. Intel, AMD, IBM and Sun have all implemented CMP

This paper was presented at the NIK-2007 conference; see <http://www.nik.no/>.

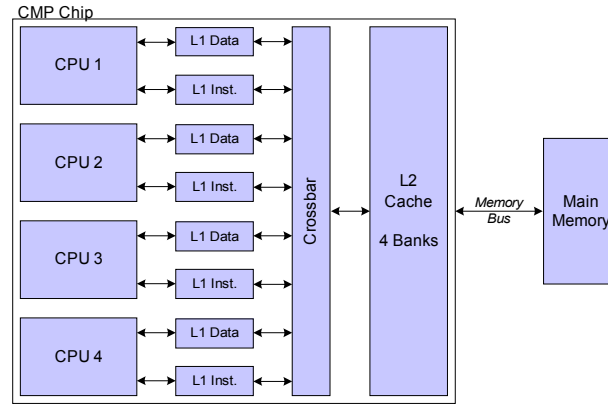


Figure 1: A 4-core, Shared Cache Chip Multiprocessor (CMP)

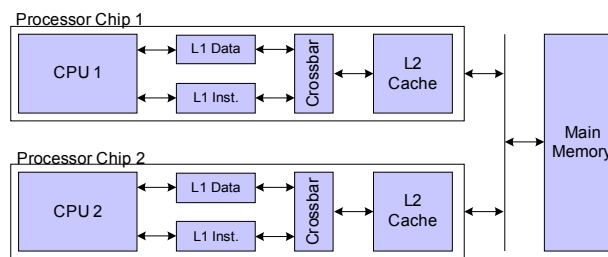


Figure 2: A Traditional 2-processor, Symmetric Multiprocessor (SMP)

processors with from 2 to 8 processor cores [12, 3, 10, 2]. This popularity is primarily due to three factors. Firstly, it is becoming difficult to increase performance with techniques based on *Instruction Level Parallelism (ILP)*. Secondly, complex out-of-order cores have a high power consumption. Thirdly, the large design complexity of speculative, out-of-order processors has become difficult to manage.

CMPs address these issues by providing multiple, less complex processing cores which make true parallel execution of threads possible. Although *Thread Level Parallelism (TLP)* is possible through time-multiplexing on a single-core processor, a CMP devotes much of its resources to TLP at the cost of sacrificing ILP. Consequently, single threaded applications may experience a slow-down when moved to a multi-core processor. The performance might also decrease because of sharing effects in the memory system. For these reasons, the memory hierarchy should have sufficient *Memory Level Parallelism (MLP)* to make these effects as small as possible.

Blocking caches do not accept new requests while a miss is being serviced. Consequently, one way of introducing MLP is to support multiple outstanding misses in each cache. This design option was first introduced by Kroft [16]. A cache that can service multiple misses in parallel is called *non-blocking* or *lockup-free*. The number of requests the cache can handle in parallel is known as the *cache bandwidth* [18].

To handle more than one cache miss at a time, a hardware structure is needed for bookkeeping. Using the terminology of Tuck et al. [8], this structure is called the *Miss Handling Architecture (MHA)*. The key hardware structure in a MHA is the *Miss Status Holding Register (MSHR)*. The number of MSHRs determine the number of cache misses that can be handled simultaneously. If a cache has 4 MSHRs, the cache blocks if a new miss occur when 4 misses are pending at one time. As an example, the Intel Pentium 4 processor can support 8 outstanding misses at a time in the L1 data cache [4].

The *Miss Handling Architecture (MHA)* increases performance in two ways compared to a blocking cache:

- The cache does not block as long as there is an unused *Miss Status Holding Register (MSHR)*.
- Only one request is sent to the next memory hierarchy level if the processor generates more than one request for the same cache block. This is handled by storing information about the additional misses in the already allocated MSHR. This way of combining miss operations reduces memory traffic.

In the terminology of Kroft [16], the misses that lead to the allocation of a new MSHR are called *primary misses*. *Secondary misses* are misses that are added to an existing MSHR entry and do not cause the cache to block. Farkas and Jouppi coined the term *structural-stall miss* for a subsequent miss that makes the cache block. This can happen if there is no more space to store targets in the MSHR.

Earlier works have focused on finding the best number of MSHRs for in-order processors, superscalar processors, multithreaded processors and processors with a large number of in-flight instructions [16, 18, 8]. The aim of this paper is to investigate the performance impact of cache miss bandwidth in shared cache CMPs. As far as we know, this is the first investigation into MHAs for CMPs.

This paper will focus on the CMP architecture shown in figure 1. Here, four processor cores have private L1 caches that communicate with a shared L2 cache through a crossbar interconnect. This crossbar is based on the interconnect used in IBM's Power 5 processor [11]. Figure 2 shows a traditional symmetric multiprocessor. Here, each processor core is on its own physical chip. Therefore, the available off-chip bandwidth will scale with the number of processors. Since the number of pins that can be placed on a chip is limited by manufacturing and economic constraints, this is not the case for CMPs. Consequently, off-chip bandwidth is a more constrained resource in a CMP than in a SMP.

The paper is organised as follows: Section 2 presents previous research on Miss Handling Architectures. Section 3 discusses the simulated processor and memory hierarchy. The results from the experiments are presented in section 4 and discussed in section 5. Finally, section 6 concludes the article and indicates further work.

2 Background

A *Miss Handling Architecture (MHA)* consists of a number of *Miss Status Holding Registers (MSHRs)* [8], and the number of MSHRs determine the number of outstanding misses a cache can have before it blocks. This number is closely related to the notion of *cache bandwidth* which is the number of parallel requests the cache can handle [18]. The difference is that the cache bandwidth term also incorporates the number of simultaneous hits. Multiple cache banks or cache ports enable this.

Figure 3 shows a conventional MHA for a banked cache. In this case, each bank has its own MSHR file. The MHA term refers to the MSHR files of all banks collectively. If the cache only has one bank, there is only one MSHR file in the cache. Consequently, the MHA consists of this MSHR file only.

Figure 4 illustrates the internal structure in each MSHR file. The MSHR file is a small fully associative cache that can handle up to n outstanding misses. To enable this, each MSHR has its own comparator. Consequently, there are n MSHRs and n comparators in the MSHR file. Each MSHR stores the following information:

- The cache block address of the miss.

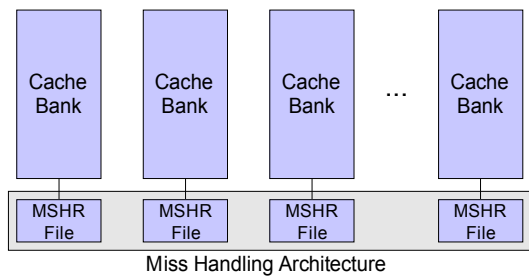


Figure 3: Conventional Miss Handling Architecture

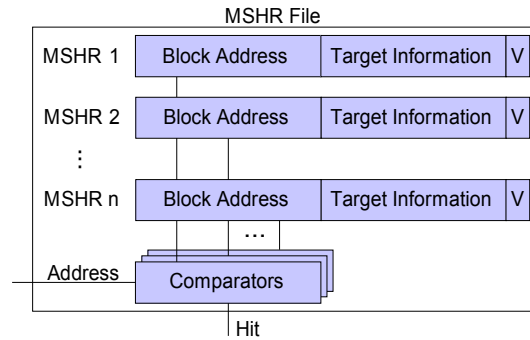


Figure 4: Miss Holding Status Register File

- Some target information is also needed. For instance, if the requested data width is less than a whole cache block, the data offset within the cache block must be stored. In a shared cache, the ID of the requesting CPU must be stored.
- A valid bit is needed to note which MSHRs are in use. If all valid bits are 1, the cache should block immediately as a new miss can not be handled.
- If the cache is write-back, a write buffer is needed. Here, write-back signifies that write hits are updated directly in the L1 cache and only written to the L2 cache when the block is replaced. In the write miss case, the request is sent to the L2 cache and space is only allocated when this request is answered. We do not want to send the data to the L2 cache only for it to be returned to the L1 cache, so it must be stored temporarily in the write buffer.

MSHR Target Storage

One of the advantages of using a MHA is that only one request is sent to the next level of the memory hierarchy when there are multiple requests for the same cache block. The details of how to do this have been hidden in the *Target Information* field in figure 4.

Write-Through Cache Target Storage

Farkas and Jouppi evaluated four different ways of storing target information in a write-through L1 cache [13]. They examined among others the *implicitly addressed* and *explicitly addressed* MSHR organisations. The *implicitly addressed* MSHR fields technique is the design option used by Kroft in the original MSHR paper [16]. Here, combining two requests to the same cache line is possible as long as they request a different processor word. For instance, if the cache block size is 32 bytes and a processor word is 32 bits, there are 8 processor words in a cache block. Each of these words have their own valid bit, destination field and format field. Consequently, if the valid bit for the requested word is 1 on a secondary miss, the cache must block. The destination field is the register the word will be stored in, and the format field contains additional information as for example the width of the load, byte address for byte addressed loads and if the data should be sign extended.

An improvement over the implicitly addressed method is the *explicitly addressed* MSHR field design. Here, the address within the block is explicitly stored in the MSHR, and multiple misses to the exact same address can be handled without stalling at the cost

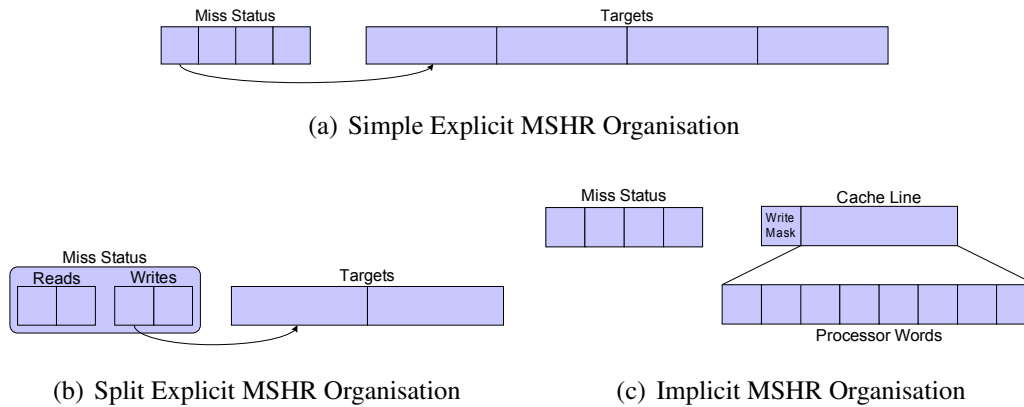


Figure 5: MSHR Target Storage with Write-back Caches

of storing more information per target. Multiple misses to the same processor word can happen in out-of-order processors. The MSHR implementation used in the M5 simulator [9] use explicitly addressed MSHR fields. The reader is referred to the paper of Farkas and Jouppi [13] for details about their last two methods, one called *inverted MSHR organisation* and one based on using the cache data storage to store MSHR information.

Write-Back Cache Target Storage

Farkas and Jouppi’s work is based on write-through and no-allocate caches. As most caches currently are write-back, Tuck et al. [8] extended Farkas and Jouppi’s implicit and explicit schemes. This extension is necessary since the cache must buffer the data that will be written to the cache line until the miss is serviced and space has been allocated for the received line. In the write-through case the data was sent directly to the next memory hierarchy level. Consequently, there was no need to buffer data.

The modified MSHR organisation is shown in figure 5. In the simplest case, all MSHR targets can be reads and writes. Consequently, enough buffer space must be provided to handle the worst-case situation where all entries are writes (figure 5(a)). To reduce the size of the write buffer, Tuck et al. propose to split the MSHR entries into read and write entries. In figure 5(b), the MSHR consists of two read entries and two write entries. Consequently, only two writes can occur at the same time and the write buffer size is decreased. However, the cache must block after two reads or two writes to the same cache line. In other words, there is a trade-off between area and performance.

The write buffer size can be reduced even more by using an implicit organisation as shown in figure 5(c). In this case, the write buffer is the size of one cache line and a write mask is added. The write mask stores which processor words have been written to. Since the cache must block on any write to the same processor word, the performance of this organisation is likely to be worse than the other alternatives. However, its low area cost makes it attractive. Another drawback is that it can only be applied to L2 caches when the cache line size of the L1 cache is less than the cache line size of the L2 cache.

In summary, there are many possible ways of designing a MSHR. Typically, the different solutions signify a different area/performance trade-off.

A Miss Handling Architecture for Superscalar Processors

Sohi and Franklin proposed to use “multi-port, non-blocking” caches to provide sufficient cache bandwidth for superscalar processors [18]. However, their use of the word multi-

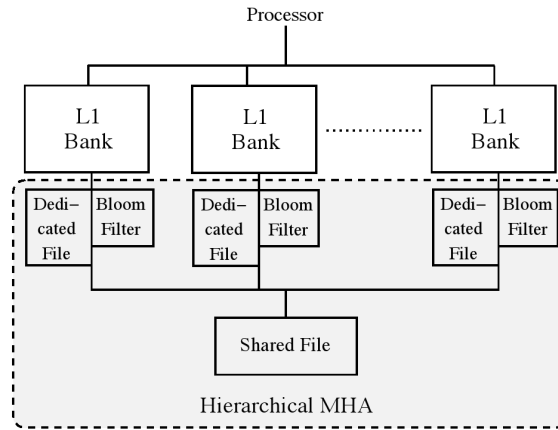


Figure 6: Hierarchical MHA (Reproduced from [8])

port is somewhat misleading. In today's terminology, a multi-port cache can handle simultaneous reads to all cache lines and simultaneous writes to different cache lines. This organisation is expensive both in terms of area and power [17]. The architecture proposed by Sohi and Franklin has multiple banks. Here, independent caches are responsible for a part of the total address space. Consequently, requests can be handled in parallel as long as they go to different banks.

Introducing multiple banks creates the possibility of servicing cache hits in parallel. However, Sohi and Franklin found that the blocking cache banks was limiting performance. Their evaluation showed that a L1 cache with 8 banks and 4 MSHRs per bank performed considerably better than a blocking L1 cache. However, they did not report results for other cache configurations. Therefore, it is unclear whether 4 MSHRs per bank is the best choice. In addition, Farkas and Jouppi [13] note that Sohi and Franklin does not take into account the hardware complexity of this cache design.

Hierarchical Miss Handling Architecture

Tuck et al. [8] have recently proposed a hierarchical MHA primarily aimed at providing high *Memory Level Parallelism (MLP)* to architectures that support a large number of in-flight instructions. High MLP is achieved by providing high cache bandwidth and low lock-up time [8]. An additional constraint is that this parallelism should be provided in an area efficient manner.

Their solution is shown in figure 6 and has one small dedicated MSHR file per L1 bank and a larger MSHR file shared by all banks. This is more area efficient than a large MSHR file per bank. The key idea is that the larger MSHR file can be used by a bank when the local MSHR file is full. This assumes that the requests will not distribute uniformly over the different banks as this would lead to competition for space in the shared MSHR file. A Bloom filter (a space-efficient probabilistic data structure) is used to reduce the number of accesses to the shared MSHR file. According to Tuck et al., this removes a potential performance bottleneck.

Since the shared MSHR file can be accessed from all banks, race conditions are possible. Handling these add to the implementation complexity of the technique.

As mentioned earlier, misses should not distribute evenly across banks. If they do, the dedicated MSHR files become full at the same time and the banks compete for space in the shared MSHR file. In this situation, larger dedicated MSHR files might yield better

performance. In the experiments carried out by Tuck et al., this situation does not happen sufficiently often to have a performance impact as the hierarchical scheme performs close to an unlimited MHA.

Tuck et al. evaluated their MHA with 8 banks in the L1 cache. This is a relatively large number of banks. The aim is to provide a large request bandwidth. However, it is unclear whether their results extend to an L1 cache with fewer banks. In addition, other techniques might be suitable for a large L2 cache. Finally, they assume a very powerful memory bus which removes the downside of supporting high memory level parallelism.

3 Methodology

The Chip Multiprocessor (CMP) architecture used in the simulations in this paper is shown in figure 1 on page 2. All simulations are carried out using the system call emulation mode of the cycle accurate M5 simulator [9]. The model of the crossbar interconnect was developed as a part of Jahre's master thesis [15].

The workloads used in this article was created by randomly choosing applications from the SPEC2000 benchmark suite [19]. Each SPEC2000 benchmark is represented in at least one workload. These workloads (each consisting of 4 SPEC benchmarks) are then run on the simulated hardware. By running one workload on different hardware configurations (e.g. one with 1 MSHR and one with 4 MSHRs), the performance difference between the hardware configurations can be quantified. In a simulation run, each benchmark is fast-forwarded a random number of clock cycles between 1 and 1.1 billion. Then, detailed simulation is carried out for 100 million clock cycles measured from when the last benchmark finished its warm-up. This methodology is based on the methodology used by Dybdahl et al. [7].

The modelled processor core configuration is shown in table 1. This configuration creates a powerful superscalar processor. An alternative would be to use a simpler core that uses less chip area. However, an out-of-order core will create a larger strain on the memory system. As this work focuses on the memory system, this core type is a natural choice. It is expected that the performance gains of adding miss bandwidth would be lower when a less powerful core is used. Table 2 shows the memory system configuration. The cache sizes are comparable to state-of-the-art commercial processors [12], and all caches are write-back and use the LRU replacement policy. Further details regarding the simulator parameters can be found in Jahre's master thesis [15].

The M5 Miss Handling Architecture (MHA) uses explicitly addressed target fields. Furthermore, the simple explicit organisation from figure 5(a) is used. Consequently, a secondary miss to a cache block can be handled as long as there are free target slots. The number of target slots are 4 for all experiments in this report. In this work we focus on performance, and therefore the other explicit organisations from figure 5 with lower performance and better hardware efficiency are not simulated. Investigating the performance impact of varying the target slots and MSHR organisation is left as further work. In addition, each cache bank has a dedicated MSHR file. Exploration of hierarchical MHAs is also left as further work.

In this work, MHAs with 1, 2, 4, 8, 16, 32, 64 and 1024 MSHRs have been considered. 1 MSHR is a *blocking cache* and the unreasonably large 1024 MSHR MHA is used to estimate the performance potential of a very powerful but expensive MHA solution. There are two reasons for increasing the number of MSHRs in powers of two. Firstly, increasing by a power of two means adding one address bit. Using an MSHR count that is not a power of two would therefore imply unused address bits. Secondly, detailed

Parameter	Value
Processor Cores	4
Clock frequency	3.2 GHz
Reorder Buffer Size	128 entries
Store Buffer Size	32 entries
Instruction Queue Size	64 instructions
Instruction Fetch Queue Size	32 entries
Load/Store Queue Size	32 instructions
Issue Width	8 instructions/cycle
Functional units	4 Integer ALUs, 2 Integer Multiply/Divide, 4 FP ALUs, 2 FP Multiply/Divide, 4 Memory Read/Write Ports, 1 Internal Register Access Port
Branch predictor	Hybrid, 2048 local history registers, 2-way 2048 entry Branch Target Buffer (BTB)

Table 1: Processor model parameters

Parameter	Value
Level 1 Data Cache	32 KB 8-way set associative, 64KB blocks, LRU replacement policy, Write-Back, 3 cycles latency, 1 bank
Level 1 Instruction Cache	32 KB 8-way set associative, 64KB blocks, LRU replacement policy, Write-Back, 1 cycle latency, 1 bank
Level 2 Unified Shared Cache	4 MB 8-way set associative, 64KB blocks, LRU replacement policy, Write-Back, 14 cycles latency, 4 banks
L1 to L2 Interconnection Network	Crossbar Topology, 4 Clock Cycle Transfer Latency, 5 Clock Cycle Arbitration Latency, 64 byte Wide Transmission Channel
Main memory	112 cycles access time, 8 byte wide

Table 2: Memory system parameters

simulation is compute intensive. Therefore, there is a need to prune the search space.

To fully explore the merits of a MHA, the power and area overhead must be taken into account. In this work, we use the CACTI tool [6, 1] to estimate area and power consumption. The MSHR file is modelled as a fully associative cache. Each MSHR entry is assumed to take 2 bytes which is in accordance with the assumption made by Tuck et al. [8]. Since the number of targets are 4 for all configurations used in this work, the size of a MSHR line is $2 \cdot 4 = 8$ bytes. Some caches used in this report are too small to be handled by CACTI. In this case, linear interpolation is used to get values for these configurations.

A write buffer is also needed since the caches used in this work are write-back. Furthermore, for each MSHR there is a need to accommodate 4 cache lines. The reason is that all MSHR entries might be writes when the simple explicit MSHR organisation is used. This buffer is modelled as a SRAM memory.

4 Results

This section reports the results from investigating a number of different *Miss Handling Architectures* (MHAs). First, the performance of different MHAs is presented. Then, the area and power overhead of the different configurations is quantified.

Performance Results

As explained in section 3, 40 multiprogrammed workloads were generated by randomly choosing applications from the SPEC2000 benchmark suite. The performance of each workload is then measured by taking the harmonic mean of the number of instructions

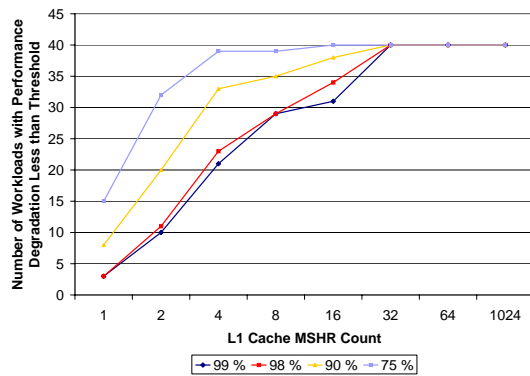


Figure 7: Number of MSHRs in the L1 Cache and Performance

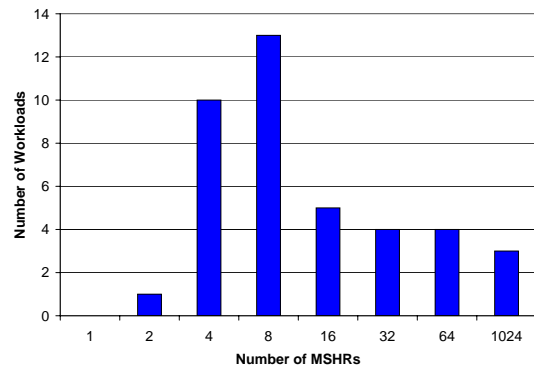


Figure 8: The Best Number of L1 MSHRs for Each Workload

committed per cycle on each core. Then, the number of benchmarks that perform better than a threshold compared to the 1024 MSHR case are counted. This number is used to quantify performance for a given MSHR count. The thresholds used in this article are 75%, 90%, 98% and 99%.

The performance results from varying the number of MSHRs in the L1 cache are shown in figure 7. In this case, the number of MSHRs in the L2 cache was kept constant at 128. The reason was to make sure that the L2 MSHR count did not limit L1 cache miss performance too much. Figure 7 shows that 8 MSHRs are sufficient if the goal is to perform within 75% percent of the performance with 1024 MSHRs. When the number of MSHRs is increased to 32, all workloads perform within 99% of the large MSHR file. 16 MSHRs is a compromise where 31 workloads perform within 99%.

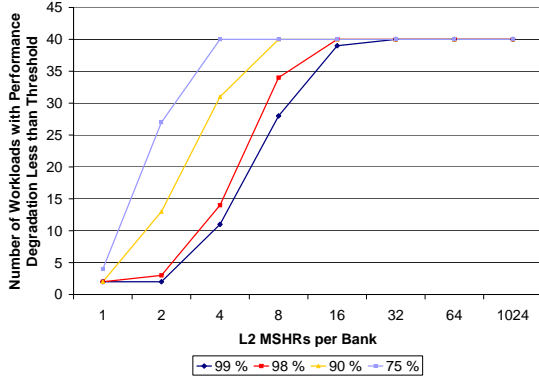
The number of workloads that have their best performance with a given number of MSHRs is shown in figure 8. This figure is necessary because the performance of many workloads is maximised with fewer than 1024 MSHRs. This fact is hidden in figure 7. If more than one MHA have exactly the same performance, the configuration with the lowest number of MSHRs is chosen.

Figure 8 indicates that some benchmarks benefit from a large number of MSHRs and that some do not. In fact, 24 workloads have their best configuration with 8 or fewer MSHRs. The reason is that a high number of parallel misses create a large strain on the L1 to L2 crossbar and the off-chip memory bus. Congestion in the crossbar makes all L2 accesses take more time and congestion on the memory bus makes all memory accesses take longer. The result is that the miss latency becomes large and overall system performance is maximised by having fewer MSHRs.

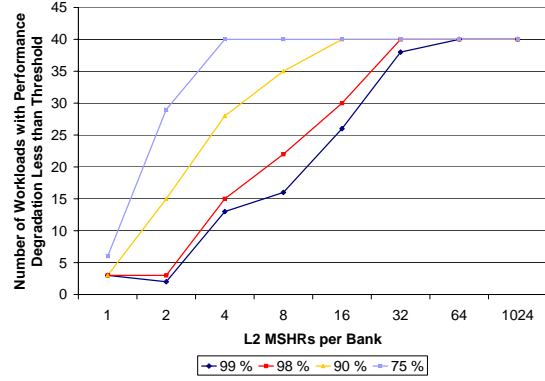
Figure 9 shows the performance impact of the number of MSHRs in the L2 cache. Note that the numbers in the figure show the number of MSHRs *per L2 bank* in this case. Consequently, the total number of MSHRs is four times the number used in the figure.

As shown in figure 9, 16 MSHRs in the L2 cache removes the performance degradation when there are 8 MSHRs in the L1 cache. Similarly, 32 L1 MSHRs require 64 L2 MSHRs to avoid performance degradation. However, only one workload has a performance degradation in the 32 L2 MSHRs case. Consequently, 32 L2 MSHRs is probably the best choice in this case.

With 8 L1 MSHRs it is advantageous to support 16 MSHRs in the L2 cache. The reason is that this significantly reduces the time the L2 cache is blocked. When the L2 cache is blocked, requests can queue up in the L1 to L2 interconnect. When the cache then



(a) 8 MSHRs in the L1 Cache



(b) 32 MSHRs in the L1 Cache

Figure 9: Number of MSHRs in the L2 Cache and Performance

Number of MSHRs	Relative Energy per Read Access	Relative Area
1 [†]	7 %	4 %
2 [†]	8 %	4 %
4 [†]	10 %	5 %
8	14 %	6 %
16	19 %	9 %
32	31 %	14 %
64	68 %	26 %

Table 3: L1 MHA Energy and Area

Number of MSHRs per Bank	Relative Energy per Read Access	Relative Area
1 [†]	0.99 %	0.25 %
2 [†]	1.04 %	0.25 %
4 [†]	1.13 %	0.26 %
8	1.32 %	0.28 %
16	1.68 %	0.31 %
32	2.39 %	0.36 %
64	3.90 %	0.73 %

Table 4: L2 MHA Energy and Area

unblocks, these requests are released. If the L2 cache then blocks right away, this queue will never have a chance to clear. This situation is avoided by having a few more MSHRs in the L2 cache than in the L1 cache. When there are more MSHRs available in the L1 and L2 caches, this situation becomes less likely. The reason is that few applications benefit such large miss level parallelism. Consequently, the trend is nearly invisible in figure 9(b).

Area and Power Overhead

The area and power overhead of the different MSHR counts were estimated with the CACTI tool [1]. Table 3 shows the results for the L1 MHA. The figures are given as percentages of the total L1 cache area and the total L1 cache energy consumption per access to aid understanding.

Supporting 16 MSHRs can be done with an area cost that is less than 10% of the L1 cache area. However, supporting 32 MSHRs or 64 MSHRs in the L1 cache has a large area overhead. Furthermore, the energy consumed in one read access to a MSHR file with 32 registers is 31% of a L1 read access. The impact of this is not large, as the MSHR file will only be accessed on a cache miss. However, these results significantly decrease the attractiveness of large MSHR files.

Table 4 shows the CACTI results for the L2 MHA. Again, the MSHR counts are the number of MSHRs *per L2 bank*. Since the L2 cache is large, the overhead of the MHA

[†]Since CACTI can not analyse caches that are smaller than 64 bytes, these values are found by linear interpolation over the values from the larger caches.

is small even for the largest MSHR files. Consequently, performance issues will be more important than area and energy in this case.

5 Discussion

The experiments have identified two factors related to Miss Level Parallelism that influence overall system performance in the simulated 4-core CMP:

- Less cache blocking increase performance up to a point
- Too many parallel misses can create congestion in the on- and off-chip interconnects

Consequently, the best number of MSHRs is a trade-off between two competing design constraints. Firstly, the MHA should support as many parallel misses as possible. Secondly, the MHA should fulfill a filtering function. Here, it reduces the number of requests to the interconnect and through this avoids congestion. This fits well Burger et al.'s [5] statement that latency hiding techniques increase the bandwidth demand. Furthermore, it makes it unlikely that MHAs with very high memory level parallelism like Tuck et al.'s Hierarchical MHA will perform well in a CMP. On the other hand, more work is needed to verify that these observations extend to other CMP architectures.

All experiments in this article have assumed 4 targets per MSHR. The simulation results shows that the caches have been blocked for this reason a number of times. Consequently, analysing the impact of varying this parameter is interesting future work.

Due to the lack of address translation in the M5 simulator's system call emulation mode, each application has been given a part of the physical address space. Incidentally, this partitioning makes each application use one L2 bank. Since there are four L2 banks and four applications in a workload, each application has its own L2 bank. Consequently, the accesses are perfectly balanced between the MSHR files in the different banks. Carrying out experiments with a different address to bank mapping is left as further work.

In the current crossbar interconnect implementation, the crossbar blocks when one of the L2 cache banks blocks. Consequently, the overall performance impact of the number of MSHRs in the L2 cache might be somewhat exaggerated in this work. Removing this implementation restriction from the crossbar and investigating the impact of this is also left as further work.

The latency of a large cache is higher than the latency of a small cache. Therefore, the access latency of a large MSHR file will be higher than the access latency of a small MSHR file. Furthermore, the L1 hit latency is very important as it often is on the processors critical path. Consequently, the MHA should be designed such that it does not increase the hit latency.

6 Conclusion

All workloads used in this article benefit from non-blocking caches. However, finding the best size of an MHA is slightly more complicated.

In the L1 cache, the best observed performance is a trade-off between two factors. Firstly, some workloads benefit from having a large number of parallel misses. Other workloads benefit from increased miss parallelism to a point. After this point, increasing the number of MSHRs actually reduces overall system performance. This is due to congestion in on- and off-chip interconnects. Furthermore, the area overhead of MSHR files with 32 or more registers is high.

The best performing L2 MHAs have slightly larger MHAs per bank than the MHAs in the L1 cache. By having a few more MSHRs than in the L1 cache, the L2 cache lock-up

time is reduced. Increasing the number of MSHRs further has no performance effect. In addition, the area overhead is small compared to the L2 cache size.

An interesting piece of further work is to look into adaptive techniques that balance the number of parallel misses with the congestion in the on- and off-chip interconnects. This can avoid the performance degradation observed for many workloads. More importantly, one application with many misses should not reduce the performance of applications running on other cores by using a disproportionately large part of the available bandwidth.

References

- [1] CACTI Web Page. <http://quid.hp1.hp.com:9081/cacti/>.
- [2] AMD. AMD Athlon 64 X2 Dual-Core Processor for Desktop. http://www.amd.com/us-en/Processors/ProductInformation/0,,30_118_9485_13041,00.html.
- [3] B. Sinharoy et al. POWER5 System Microarchitecture. *IBM J. Res. Dev.*, 49(4/5):505–521, 2005.
- [4] D. Boggs et al. The Microarchitecture of the Intel Pentium 4 Processor on 90nm Technology. *Intel Technology Journal*, 8(1), 2004.
- [5] D. Burger et al. Memory Bandwidth Limitations of Future Microprocessors. In *ISCA '96*, pages 78–89, New York, NY, USA, 1996. ACM Press.
- [6] D. Tarjan et al. CACTI 4.0. Technical report, HP Laboratories Palo Alto, 2006.
- [7] H. Dybdahl et al. An LRU-based Replacement Algorithm Augmented with Frequency of Access in Shared Chip-Multiprocessor Caches. In *MEDEA '06*, pages 45–52, New York, NY, USA, 2006. ACM Press.
- [8] J. Tuck et al. Scalable Cache Miss Handling for High Memory-Level Parallelism. In *MICRO 39*, pages 409–422, Washington, DC, USA, 2006. IEEE Computer Society.
- [9] N. L. Binkert et al. The M5 Simulator: Modeling Networked Systems. *IEEE Micro*, 26(4):52–60, 2006.
- [10] P. Kongetira et al. Niagara: A 32-Way Multithreaded SPARC Processor. *IEEE Micro*, 25(2):21–29, 2005.
- [11] R. Kumar et al. Interconnections in Multi-Core Architectures: Understanding Mechanisms, Overheads and Scaling. *ISCA'05*, 2005.
- [12] S. Gochman et al. Introduction to Intel Core Duo Processor Architecture. *Intel Technology Journal*, 10(2), 2006.
- [13] K. I. Farkas and N. P. Jouppi. Complexity/Performance Tradeoffs with Non-Blocking Loads. In *ISCA '94*, pages 211–222, Los Alamitos, CA, USA, 1994. IEEE Computer Society Press.
- [14] J. L. Hennessy and D. A. Patterson. *Computer Architecture - A Quantitative Approach, Fourth Edition*. Morgan Kaufmann Publishers, 2007.
- [15] M. Jahre. Improving the Performance of Communicating Workloads in Chip Multiprocessors with Architectural Techniques. Master's thesis, NTNU, 2007.
- [16] D. Kroft. Lockup-free Instruction Fetch/Prefetch Cache Organization. In *ISCA '81*, pages 81–87, Los Alamitos, CA, USA, 1981. IEEE Computer Society Press.
- [17] G. Reinman and N. P. Jouppi. CACTI 2.0: An Integrated Cache Timing and Power Model. Technical report, Compaq Western Research Laboratory, 2001.
- [18] G. S. Sohi and M. Franklin. High-Bandwidth Data Memory Systems for Superscalar Processors. In *ASPLOS-IV*, pages 53–62, New York, NY, USA, 1991. ACM Press.
- [19] Standard Performance Evaluation Corporation. SPEC CPU 2000 Web Page. <http://www.spec.org/cpu2000/>.