Intermediary Cache Memory Level for Data Exchange and Interactions of Parallel Scientific Applications

Marco Antonio Zanata Alves
mazalves@inf.ufrgs.br

Philippe Olivier Alexandre Navaux
navaux@inf.ufrgs.br

Universidade Federal do Rio Grande do Sul (UFRGS), Informatics Institute
Av. Bento Gonçalves, 9500 - Porto Alegre, Rio Grande do Sul, 91501-970, Brazil

Abstract

In current multi-core processors and future many-core processors, the memory project plays a key role due to its demanding of high data throughput, coherence and data exchange between threads. Based on this scenario, we propose to use an intermediary cache memory level between the first private level and second shared level of the memory hierarchy. This intermediate shared cache has a lower capacity and latency than the next cache level, creating and attractive space for a fast data exchange and interaction between different execution threads distributed among the multiple cores. The results evaluating the intermediary cache over traditional cache organizations show a performance improvement of up to 44% running the synthetic workload and up to 55% of speedup with the real workload when the intermediary cache level was used.

1 Introduction

The innovations on integration technology, lead to an increase on the number of transistors per chip. In this scenario, the multi-core and many-core chips became alternatives to support performance gains for the next years, shifting the focus from instruction level parallelism (ILP) and traditional techniques [5] [13] such as pipelining and superscalarity to explore the thread level parallelism (TLP) with multi-threaded [15] and multi-core [11] processors. Moreover, these new architectures allowed designers to improve the performance of parallel applications, maintaining or even reducing the total power dissipation. However, the cache memory hierarchy became more important on these data hungry processors, which requires high data throughput to support all the processing elements.

Nowadays, commercial multi-core processors show different memory organizations. For instance, one can find private L2 caches or shared L2 cache, and also shared multi-sliced cache on the state-of-the-art processors. Therefore, the definition of cache organization for next multi-core and many-core generations do not considers a specific layout to improve the performance of data sharing, which has a great impact on the performance of different application types.

For many years, the design considerations between the primary and secondary cache memories are different [12]. In a given cache hierarchy with two levels, the first level should be designed to minimize the data access penalty, while the second cache level should be focused on reducing the data misses rate, and reducing the long latency penalty for accessing the main memory.

For the current multi-core processors, cache memories have to aggregate thread-to-thread interaction functions, such as synchronization and communication for threads running on different processor cores. That is, besides providing the necessary data for the instruction flows running on a particular processor core, cache memories are now responsible for improving the performance of data exchange and interactions among the different cores. The cache should treat data exchange every time that two or more threads need to share some data or buffer.

In this multi-core context, we propose the inclusion of one intermediate cache in the memory hierarchy, between the first and second cache levels. This new level shall be shared among multiple cores and should be faster than the subsequent level, in order to be responsible for fast communication and synchronization between the various cores. This approach of shared intermediary cache is quite different from the traditional cache architecture, once it is proposed for leverage the data sharing. Thus, in future this approach may vary the cache size dynamically depending on the application data sharing

For evaluation, an ISA level full-system simulator named Simics was used to model and simulate an 8 core processor with different cache organizations in order to analyze and validate the performance of our proposal, comparing two architectures with the intermediary cache memory organi-

zation over two traditional architectures without the middle cache. In addition, it was used a producer-consumer synthetic workload and a real workload, consisting of 4 applications of computational fluid dynamics from NAS parallel benchmark, in order to exercise the simulated models.

This paper is organized as follows: Section 2 presents some related work. Section 3 presents some information about the simulator and the cache memory organizations modeled. Section 4 shows the synthetic and the real parallel benchmarks. Section 5 brings some results and analysis about the performance and cache behavior. Section 6 presents some conclusion and future work.

2 Related Research

In [3], Fide and Jenks present a proposal to include new instructions to control block eviction on private caches. The programmer must use these instructions to decide when a particular block should be removed from the private cache and sent to the cache memory level which is shared among multiple cores. The authors demonstrate performance gains from 8% to 28% by using a shared L3 cache memory together with the new cache control instruction.

Kandemir et al. [8], use a shared scratch-pad memory to increase communication and synchronization performance in embedded systems. In this proposal, the programmer must select and send to the scratch-pad all the data that must be shared among the processor cores. The results show reductions of up to 33.8% (24.3% on average) on the energy consumption.

Chen et al. [2], evaluate a network-on-chip (NoC) interconnection to exchange data between cores, focusing on the influence of the communication latency on final performance. The theoretical results are based on a matrix multiplication, and show that NoC interconnections have great potential to offer performance gains increasing the data throughput. However, the study does not address the problem of programming parallel applications with the NoC message passing.

Jaleel et al. [6], describe an evaluation of shared cache memory. The focus was on the last level of cache memory using parallel workloads in bio-informatics. According to their results, the main characteristic of this workload is the high degree of data sharing, over 95%. This paper affirms that multiple private cache memories can reduce the performance on application with high data sharing behavior. The same conclusion was achieved by Marino [10] using the SPLASH workload and the Simics simulator, but none of these papers considers the high latency of a large shared cache.

Zahran [17] describes the importance of memory hierarchy and coherency protocols. The evaluation methodology is based on a trace-driven simulator called CHESS (Cache Hierarchy Estimator using Scalable Simulator) and the benchmark suite was SPLASH. The environment was designed to simulate 1, 2, 4 and 8 processing cores. The results point out to performance gains when L1 and L2 private caches are used in accordance with a good coherency protocol for L2 cache.

Our previous work [1] presented an investigation about the L2 cache shared by 32 processing cores in a many-core system. In this study, using the NAS parallel applications, it was emphasized that the choice for a memory hierarchy for many-core processors is still unclear. Nevertheless, the results show that performance for many-core processors shall degrade if one shared cache for all cores were adopted. This performance degradation is caused by the high latency of this big shared cache memory, and shows that the shared cache latency is the key point for performance gains.

Concerning the related work, this approach of shared intermediary cache is transparent for the programmer. Thus, the proposed organization does not increase the complexity of parallel programming for future multi-core and many-core processors. Moreover, it is possible to conclude that the use of shared memory for communication is still valid, even when using a NoC interconnection, as presented by Wang [16]. This way, even changing the interconnection model, it is not necessary to change the shared memory parallel programming paradigm, that is considered easier to program than the message passing model [4].

Regarding the high latency of large shared caches, this paper proposes a structure that reduces the impact of shared memory by inserting an intermediate cache level with low latency for data exchange and synchronization.

3 Intermediary Level Cache Simulation

For a general-purpose computer architecture evaluation with complex memory organization, simulations offer good features, once it neither needs the high cost prototyping nor analytical formulations, which use to be imprecise and hard to model. Thus, for this paper, all the evaluations were made using simulation.

The simulation environment used was the Simics version 3.0.30 from Virtutech AB [9], which was chosen because it is a full-system ISA level simulator. It supports the modeling of various machines and processors types. For the experiments, the machine chosen was the SunFire, modeling Sun Enterprise 3500 - 6500 class servers with native support of up to 30 UltraSPARC II processors. To model the cache latencies it was used the memory access time, given by the CACTI memory modeling tool [14], then converted to clock cycles considering the given clock frequency that is used inside the Simics. Moreover, the operation frequency outside the processor is considered the ideal and does not cause great influence in the simulation latency.

The general cache parameters were chosen based on the Intel Clovertown, which is a quad-core processing state of the art and represents a commonly used cache configuration from different companies. Beyond that, to study the next generation of processors and to increase the data throughput requirement, an 8-core processor was modeled. A list of fixed parameters and the values modeled in the experiments is shown in Table 1.

Table 1. Fixed simulation parameters.

OS	Ubuntu 6.06.2 LTS	SMP
Processing Cores	IPC (without cache)	1.0
	Pipeline	Not modeled
	Number of cores	8 Cores
	Cores model	UltraSparc II - V9
	Clock frequency	2 GHz
	Feature size	45 nm
Interconnection Model	Type	Cross-bar switch
	Latency	1 Cycle / cache level
L1 Cache Memory	Data mapping	2 Way set associative
	Replacing policy	LRU
	Write policy	Write-through
	Feature size	45 nm
	Size Inst./Data	16 KB / 16 KB
	Access latency	2 Cycles
	Line size	16 Bytes
Main Memory	Size	1 GB
	Access latency	78 Cycles
	Feature size	65 nm

3.1 Proposed Model - L2-1MB_L3-5MB

This first model brings the cache memory organization with an intermediary level for rapid data exchange between applications. With L2 cache size of 1 MB and a larger L3 cache of 5 MB, this organization has the necessary characteristics for fast data exchange and synchronization between multiple execution flows running on separate cores.

Figure 1 illustrates the proposed organization named L2-1MB_L3-5MB, which has a large difference between L2 and L3 cache access time. However, the L2 cache size is generous enough to keep more than twice data allocated in L1 cache. In the same way, the third cache level can also allocate more data than those available inside the L2 cache.

3.2 Proposed Model - L2-2MB_L3-4MB

This second model shows the cache memory organization also with the proposed intermediary level for fast communication, but this model brings a smaller size difference between the second and third cache levels. The L2 cache memory was modeled with 2 MB and the L3 cache with 4 MB, and this different modeling brings the same amount of total cache size, and it aims to point out the trend when the difference between latencies and capacities of memory caches are reduced.

Figure 2 illustrates this second proposed model with intermediary cache. It can be noted that, although more data can be allocated in the second cache level, this data will have greater access latency. The nomenclature for this model is L2-2MB_L3-4MB.

3.3 Traditional - L2-1MB_L3-0MB

This third model shows a cache memory organization with just two levels of cache memory. The model represents a more traditional cache memory organization, with only one level of shared memory. Moreover, this model aims to generate data for comparison, simulating the current memory system used on nowadays multi-core processors.

The model named L2-1MB_L3-0MB, exhibited in Figure 3, presents the L2 cache equal to the L2 from the first model, but without its third cache level. This model is the only one which does no preserve the total amount of cache memory, therefore some performance loss shall occur caused by the increase on the number of high latency accesses to the main memory.

3.4 Traditional - L2-6MB_L3-0MB

This fourth model presents another cache memory organization with only the first and second levels of cache memory. Thus, as the previous model, the organization is traditional, but the total amount of cache memory is the same from the two first models.

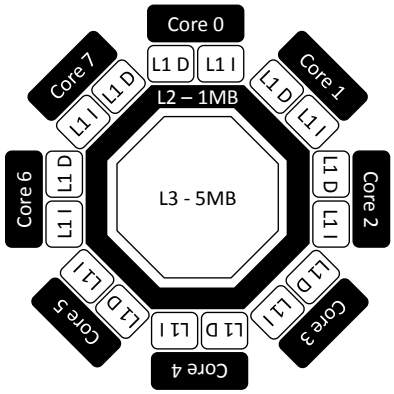
Figure 4 shows the organization for this fourth model L2-6MB_L3-0MB, where it is possible to see the large L2 cache memory, which its size is equal to the sum of L2 and L3 from the first model. With this cache of higher capacity, it should have less need of main memory access with the drawback of more penalty cycles.

4 Workloads

As workload for the simulated models, it was implemented a synthetic producer-consumer algorithm to perform the preliminary tests, then, a subset of NAS-NPB applications was adopted to represent a real workload.

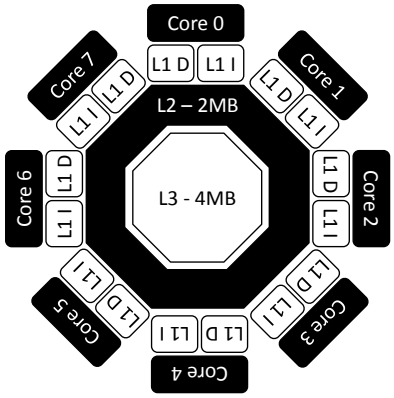
4.1 Producer-Consumer Benchmark

The producer-consumer is a classic problem in computer science, and it is composed by two processes that share a fixed size buffer. The first process is the producer, which



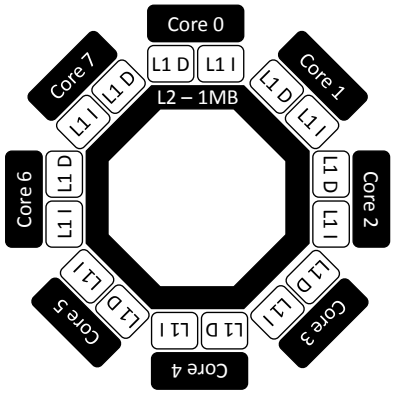
Cache Memory Level	L2 Cache	L3 Cache
Cache Size	1 MB	5 MB
Block Size	64 B	128 B
Associativity	8 Ways	16 Ways
Write Policy	Write-back	Write-back
Access Latency	12 Cycles	22 Cycles
Data mapping	Set associative	Set associative
Replacing policy	LRU	LRU
Feature Size	45 nm	45 nm
R/W ports	8 Ports	8 Ports

Figure 1. First proposed model of the cache memory organization.



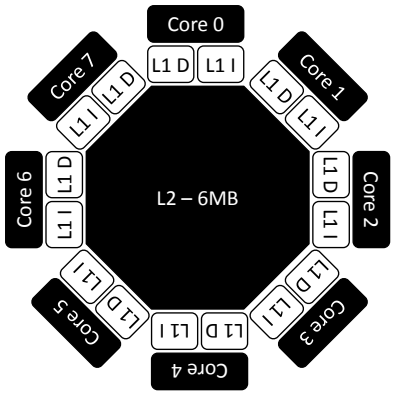
Cache Memory Level	L2 Cache	L3 Cache
Cache Size	2 MB	4 MB
Block Size	64 B	128 B
Associativity	8 Ways	16 Ways
Write Policy	Write-back	Write-back
Access Latency	14 Cycles	21 Cycles
Data mapping	Set associative	Set associative
Replacing policy	LRU	LRU
Feature Size	45 nm	45 nm
R/W ports	8 Ports	8 Ports

Figure 2. Second proposed model of the cache memory organization.



Cache Memory Level	L2 Cache	L3 Cache
Cache Size	1 MB	-
Block Size	64 B	-
Associativity	8 Ways	-
Write Policy	Write-back	-
Access Latency	12 Cycles	-
Data mapping	Set associative	-
Replacing policy	LRU	-
Feature Size	45 nm	-
R/W ports	8 Ports	-

Figure 3. First traditional model of the cache memory organization.



Cache Memory Level	L2 Cache	L3 Cache
Cache Size	6 MB	-
Block Size	128 B	-
Associativity	16 Ways	-
Write Policy	Write-back	-
Access Latency	23 Cycles	-
Data mapping	Set associative	-
Replacing policy	LRU	-
Feature Size	45 nm	-
R/W ports	8 Ports	-

Figure 4. Second traditional model of the cache memory organization.

inserts (produces) information in the buffer, and the second process is the consumer, that removes (consumes) the information from the buffer.

Although it is a simple algorithm, due to its shared buffer, the two processes need to exchange data between the producer and the consumer. This way, the algorithm has all the characteristics to stress the communication and synchronization of different cache organizations.

The implementation of this synthetic workload was made in C with OpenMP. For the complete control of what is happening in the workload, some system calls were used in order to set the affinity of each process to one particular processor core.

The producer consumer algorithm was planned to work with 4 different buffer sizes to analyze the data exchange between processes. The sizes chosen were based on the modeled size of the cache memories in order to reproduce different data sharing loads. The small size (64KB) has just 16,384 integers elements, the medium size (1MB) has 262,144 integers, the big size (2MB) has 524,288 integers and finally, the huge size (4MB) has 1,048,576 integers. Moreover, for each execution, the buffer is produced and consumed 100 times in order to stress the synchronization between the threads.

4.2 NAS Parallel Benchmark

The real workload used in this study was a subset of the NAS Parallel Benchmark (NPB) version 3.3 parallelized with OpenMP. The NPB has many applications related to numerical methods of aerodynamic simulations for scientific computing, designed to compare the performance of parallel computers.

For performance evaluation, it was chosen only the applications that simulate computational fluid dynamics (CFD) solving Navier-Stokes equations, which reproduce many of the data movements and computation found in full CFD codes [7]. Below, there is a description of CFD applications used in the measurements.

- BT - *Block Tridiagonal*: To solve 3D compressible Navier-Stokes equations with an implicit algorithm. Based on Alternating Direction Implicit (ADI) finite difference solver where the resulting system is Block-Tridiagonal, which are solved sequentially along each dimension. It uses approximately 2.6MB.
- MG - *Multigrid*: Multigrid V-cycle method used to solve the 3D scalar Poisson's equation. The algorithm works between coarse and fine grids. It exercises both short and long distance data movement. It uses approximately 55.6MB.
- SP - *Scalar Pentadiagonal*: Computational fluid dynamics application similar to BT. The problem is based on a Beam-Warming approximate factorization that decouples in 3D. The resulting Scalar Pentadiagonal system is solved sequentially along each dimension. It uses approximately 9.8MB.
- LU - *Lower and Upper triangular system*: Simulated CFD application that uses a symmetric successive over-relaxation (SSOR) method to the system resulting from finite-difference discretization of Navier-Stokes equations in 3D by splitting into a block Lower and Upper triangular systems. It uses approximately 6.5MB.

5 Performance Results

This section brings performance and cache results regarding the execution of the producer-consumer workload and the NAS parallel applications.

5.1 Producer-Consumer Workload

The first results obtained present in this section are related to the synthetic producer-consumer workload, planned to validate the environment and the proposal, since this workload implementation has strong synchronization due to occupied wait and by several sharing data volumes. Moreover, once it is a controlled workload, it tends to expose the system response for each architectural change. During the executions, the producer and consumer were binded to different processor cores using the Set-affinity system call.

Figure 5 shows the execution time given in milliseconds and the speedup results for different cache organizations. The speedup is based on the first proposal L2-1MB_L3-5MB. With these results, one can see the best performance of the first proposal L2-1MB_L3-5MB, where the first organization obtained gains of 8.06% over the organization L2-2MB_L3-4MB, obtained gains of 6.01% over the L2-1MB_L3-0MB and gains of 44.53% comparing with the L2-6MB_L3-0MB organization.

The second results for Producer-Consumer workload presented on Figure 6 are about the number of lost cycles due to misses on L1, L2 and L3 cache. For the organizations L2-1MB_L3-5MB and L2-2MB_L3-4MB, it is possible to see that lost cycles are spread over all the caches. However, the L2-1MB_L3-0MB organization had to face a high number of lost cycles for access the main memory as the buffer size increases, and on the other hand, the cycles lost during the L1 cache misses for the L2-6MB_L3-0MB organization impacts more than the cycles lost due to main memory access, which occurs because the L2 memory high latency.

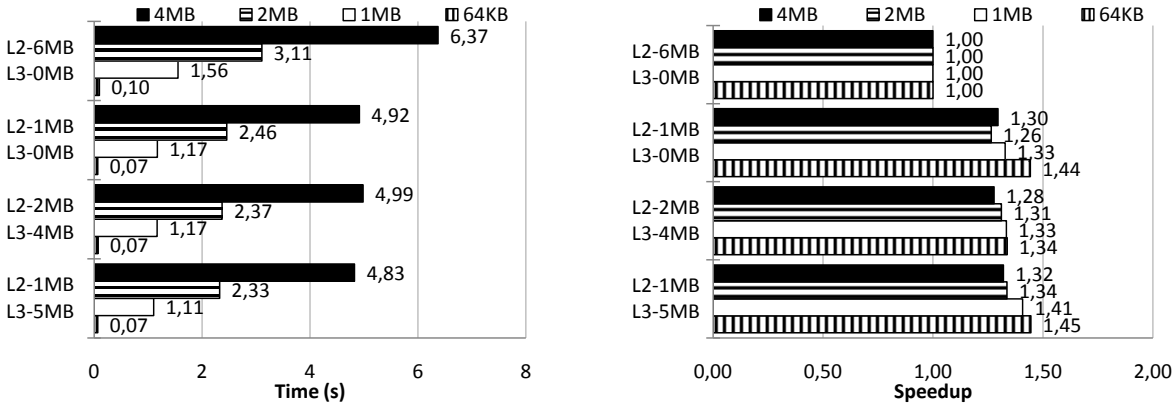


Figure 5. Execution time and speedup for Producer-Consumer workload.

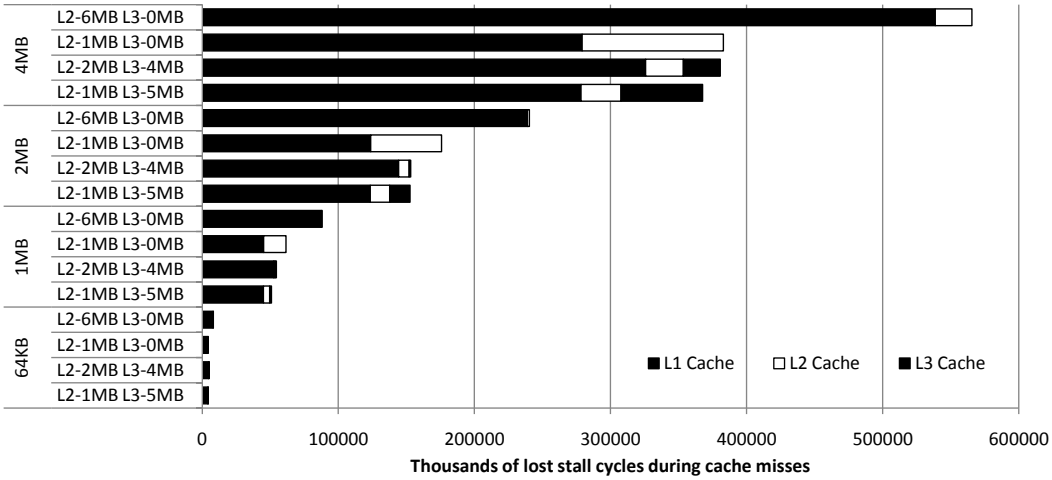


Figure 6. Lost stall cycles during cache misses for Producer-Consumer workload.

Analyzing the results, one can see the good performance using an intermediate cache level for fast communication and synchronization between processors. However, for the second proposal, L2-2MB_L3-4MB, as the difference between the cache sizes and latencies decreases, reduced gains were obtained.

Even with good results for L2-1MB_L3-0MB organization sharing small and medium vectors, as the vector increases, more data must be fetched from main memory, and this had a negative impact on performance.

Considering the large difference on latencies among the memory hierarchy of the models, some great differences on execution time were expected. However, considering that each memory data request brings a big block for the cache memory, and the shared buffer has great spatial locality, even with buffer size changes, the results had smooth variations between organizations that have a different total amount of memory, e.g. L2-1MB_L3-5MB and L2-1MB_L3-0MB organizations.

Concerning the L2-6MB_L3-0MB organization, even with a large block size, it is clear the problem of the 6MB cache on the second level, where every access on L2 cache has high latency, resulting in weak performance results.

5.2 NAS Workload

For this second part of the results, running the NAS workload, all the experiments executed the parallel applications with 8 threads.

The first results from the NAS workload, are about the execution time given in seconds and the speedup, both present in Figure 7. For the workload executions, one can see the gains on the organization L2-1MB_L3-5MB followed by organizations L2-2MB_L3-4MB, L2-1MB_L3-0MB and L2-6MB_L3-0MB respectively.

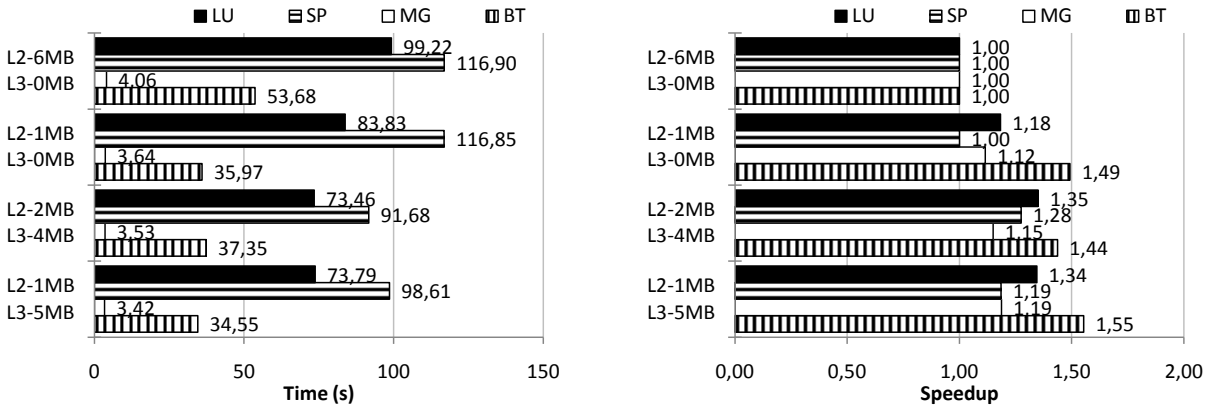


Figure 7. Execution time and speedup for NAS parallel applications.

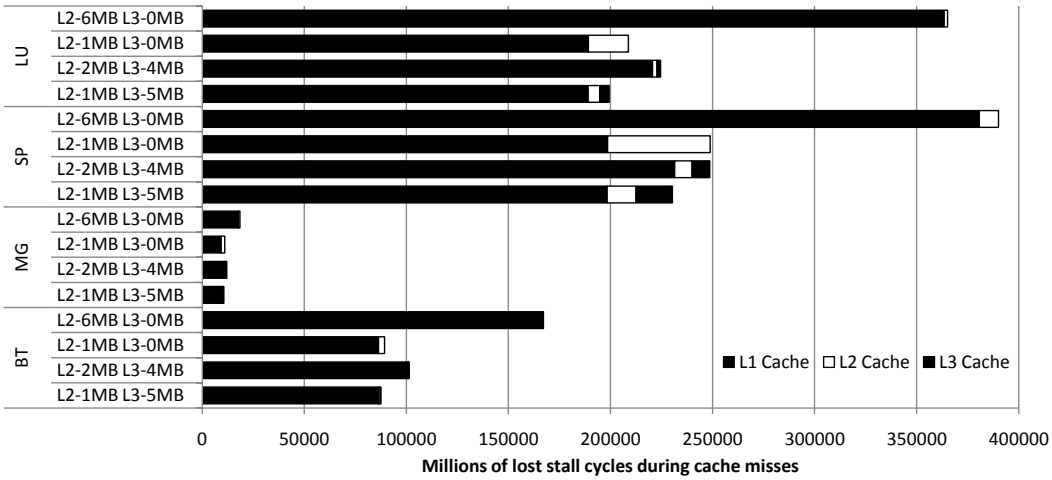


Figure 8. Lost stall cycles during cache misses for NAS parallel applications.

It is also possible to infer that the difference in the volume of memory usage impacts on the final system performance. Remembering that BT implementation occupies 2.6MB, MG occupies 55.6MB, SP occupies 9.8MB and LU 6.5MB. This way, the application BT that uses fewer amounts of memory, presents small execution time variations between the organizations L2-1MB_L3-5MB and L2-1MB_L3-0MB, which were expected, since this application may share data and do synchronization all inside the L2 cache without accessing the next memory level. However, this application demonstrates more clearly the impact of high L2 cache latency.

The second results for this workload, shown in Figure 7 are about speedup. The organization L2-1MB_L3-5MB presents performance improvements of up to 8.10% over the organization L2-2MB_L3-4MB, and gains of up to 18.49% compared with L2-1MB_L3-0MB and up to 55.36% over the last organization L2-6MB_L3-0MB. Note that organization L2-2MB_L3-4MB running the application SP has achieved a speedup of 7.56% over the first proposal L2-1MB_L3-5MB.

The second results about cache memories are shown in Figure 8, presenting lost cycles caused by wait time during cache misses. This plot shows a lot of cycles lost by the organization L2-1MB_L3-0MB, accessing the main memory due to the small L2 cache capacity, while the L2-6MB_L3-0MB organization had a small number o cycles lost during main memory access but loss many cycles accessing the L2 cache during L1 cache misses.

Even considering the wide variation in the cache memory results, it is notable the behavior of each organization. For the proposed organizations L2-1MB_L3-5MB and L2-2MB_L3-4MB, the number of lost cycles was merged between the three cache levels, where most of the misses occurring until the second level, with few misses on the third cache level.

For the L2-1MB_L3-0MB organization that had a small number of data misses on the first cache level, the data

misses in the second level generate a great impact on its final performance. On the other hand, the organization L2-6MB_L3-0MB had few data misses being fetched on the main memory, but the misses in the first level generated many lost cycles during the 6MB cache memory access.

6 Conclusion

With current multi-core processors, studies which show alternative cache organizations and architectures are welcome to next-generation processor designers. In this context, this paper presented a new cache memory organization to improve communication and synchronization between processor cores based on an intermediary level of cache memory. For the evaluation a complete system simulation tool and synthetic and real workloads, were used to compare our proposal with other traditional architectures.

Evaluating the system with the synthetic workload with a high use of synchronization and different volumes of shared data make possible to note performance gains on the proposed cache organization, with maximum gains of 30.81%, and 15.28% on average comparing to traditional organizations.

For the simulations running real workloads, there are performance gains of up to 35.64% with an average of 16.28%. Where the traditional organizations L2-1MB_L3-0MB and L2-6MB_L3-0MB have large numbers of cycles lost access to main memory and L2 cache memory access respectively, the organizations proposed L2-1MB_L3-5MB and L2-2MB_L3-4MB present a more spread distribution of lost cycles among the memory hierarchy during cache misses.

Although nowadays multi-cores processors has an increasing levels of cache memory inside a single chip, the obtained results show the importance of an intermediary shared cache in order to support fast data communication and synchronization.

As future work, the results motivate the research on data exchange for multi-core processors. Thus, the study of ways to increase the performance of parallel applications with the automatic cache size reconfiguration for fast data sharing shall produce performance improvements on multi-core systems.

Acknowledgment

This project was supported in part by CNPq (National Counsel of Technological and Scientific Development - Brazilian Government).

References

[1] M. A. Z. Alves, H. C. Freitas, and P. O. A. Navaux. Investigation of shared l2 cache on many-core processors. In *Proceedings Workshop on Many-Core*, 2009.

[2] X. Chen, Z. Lu, A. Jantsch, and S. Chen. Speedup analysis of data-parallel applications on multi-core nocs. In *ASIC, 2009. ASICON '09.*, 2009.

[3] S. Fide and S. Jenks. Proactive use of shared l3 caches to enhance cache communications in multi-core processors. *Computer Architecture Letters*, 2008.

[4] A. Grama, A. Gupta, G. Karypis, and V. Kumar. *Introduction to Parallel Computing*. Pearson / Prentice Hall, USA, second edition, 2003.

[5] J. L. Hennessy and D. A. Patterson. *Computer Architecture: A Quantitative Approach*. Elsevier, Inc., USA, fourth edition, 2007.

[6] A. Jaleel, M. Mattina, and B. Jacob. Last level cache (llc) performance of data mining workloads on a cmp: A case study of parallel bioinformatics workloads. In *Proceedings Int. Symp. on High-Performance Computer Architecture*, 2005.

[7] H. Jin, M. Frumkin, and J. Yan. The openmp implementation of nas parallel benchmarks and its performance. In *Technical Report: NAS-99-011*, 1999.

[8] M. Kandemir, J. Ramanujam, and A. Choudhary. Exploiting shared scratch pad memory space in embedded multiprocessor systems. In *Design Automation Conference*, 2002.

[9] P. Magnusson et al. Simics: A full system simulation platform. *IEEE Computer Micro*, 2002.

[10] M. D. Marino. 32-core cmp with multi-sliced l2: 2 and 4 cores sharing a l2 slice. In *Proceedings SBAC-PAD: Int. Symp. on Computer Architecture and High Performance Computing*, 2006.

[11] K. Olukotun, B. A. Nayfeh, L. Hammond, K. Wilson, and K. Chang. The case for a single-chip multiprocessor. In *Proceedings ASPLOS: Int. Symp. on Architectural Support for Programming Languages and Operating Systems*, 1996.

[12] D. A. Patterson and J. L. Hennessy. *Computer Organization and Design*. Elsevier, Inc., USA, third edition, 2005.

[13] W. Stallings. *Computer Organization and Architecture: Designing for Performance*. Prentice Hall, USA, fourth edition, 1996.

[14] S. Thoziyoor, J. H. Ahn, M. Monchiero, J. B. Brockman, and N. P. Jouppi. A comprehensive memory modeling tool and its application to the design and analysis of future memory hierarchies. In *Int. Symp. on Computer Architecture. ISCA.*, 2008.

[15] T. Ungerer, B. Robic, and J. Silc. Multithreaded processors. *British Computer Society*, 2002.

[16] X. Wang, G. Gan, J. Manzano, D. Fan, and S. Guo. A quantitative study of the on-chip network and memory hierarchy design for many-core processor. In *Parallel and Distributed Systems, 2008. ICPADS.*, 2008.

[17] M. M. Zahran. On cache memory hierarchy for chip-multiprocessor. *SIGARCH Computer Architecture News*, 2003.