Investigation of Shared L2 Cache on Many-Core Processors

Marco A. Z. Alves, Informatics Institute, Universidade Federal do Rio Grande do Sul, Brazil
Henrique C. Freitas, Informatics Institute, Universidade Federal do Rio Grande do Sul, Brazil
Philippe O. A. Navaux, Informatics Institute, Universidade Federal do Rio Grande do Sul, Brazil

Abstract

Several research works point out to share L2 cache for two or more cores. Their main goals are to improve the data access through shared addresses. However, it is hard to define the best organization and sharing model, since shared cache can improve the performance of sharing data, but it can also increase the number of conflict and misses. In this way, shared cache needs more study and evaluation considering different workloads. Therefore, the objective of this paper is to investigate the impact of multiple processing cores sharing different organizations of L2 cache. Based on these organizations, a set of scientific parallel applications is evaluated in order to tune the best cache organization on a simulated 32-core chip. Results point out to the increase of the final performance varying L2 cache organization with the increase of the L2 line size, which achieves better results than cache size. Moreover, applications with memory random access increase its performance just varying the L2 cache sharing.

1 Introduction

Currently, researchers have changed the focus on processor architectures from traditional parallel techniques [1][2][3][4][5] as pipelining, superscalarity and multithreading to multi-core and many-core approaches. In order to exploit the Thread Level Parallelism (TLP), new architectures based on multi-core processors can improve the response time on desktops and server computers running different threads from one or more applications on several processing cores. For this reason, those traditional parallel techniques have been simplified and coupled with the new generation of multi-core processors.

In a superscalar pipeline [2], besides the instruction execution divided on stages, it is possible to compute more than one instruction per cycle. This behavior is supported by some characteristics, such as: diversity of functional units, reduction of data dependence and others. Hence, superscalar processors are capable of increasing the performance speedup due to the high level of instruction parallelism.

The multithreading support [4][5] is a well-known technique that focuses on coarse-grain parallelism based on instruction stream (thread) level, increasing the thread throughput. There are some different types of TLP techniques, as follows: Simultaneous Multithreading (SMT), Blocked Multithreading (BMT), and Interleaved Multithreading (IMT). All of them work with more than one active thread, varying the number of threads in execution and the switching policy.

However, due to the innovative on the integration technology, which leads to an increase on the number of transistors per chip, the multi-core and many-core chips are alternatives to improve the data parallelism support and throughput. Multi-core architectures [6][7][8][9][10] improve performance of multithreaded applications, and also reduce the total power dissipation. For this reason, they are good solutions for several physics problems as wire size, scalability and others.

On multi-core architecture, the processor has a native parallelism to support several threads based on several cores. This native parallelism is different from traditional techniques (for instance, SMT) that add complexity on a pipeline. For instance, on SMT technique, the processing cores are emulated on a superscalar architecture. Besides, due to the increase on the number of resources on the chip (integration technology), the next generations of many-core chips can also incorporate some superscalar techniques in order to increase not just the Thread Level Parallelism (TLP) but also the Instruction Level Parallelism (ILP).

At this point, and considering a large number of threads executing on several processing cores, what is the memory subsystem organization? What is the performance of workloads? For a specific workload, can its behavior point out to specific cache memory organization?

Nowadays, commercial multi-core processors show different memory organizations from model to model, without a very clear strategy. For instance, one can find processors with private L2 caches or shared L2 cache, and also shared multi-sliced cache. Therefore, the definition of cache organization for next multi-core and many-core generations is an open problem, and it has great impact on the performance of different application types.

Focusing on problems related to cache memory organization on many-core chip processors, the objective of this paper is to investigate the impact of multiple processing cores sharing the L2 cache. Simulation tools and well-known parallel workloads are used in order to define relation between application and shared L2 cache memory.

This paper is organized into the following sections: Section 2 describes related works, Section 3 presents evaluation method, simulation model and workload description,

Section 4 presents the proposed experiments and shows results, and finally Section 5 describes conclusions and future works.

2 Related Works

Due to the next generation of many-core processors, the memory bandwidth is an important issue. Hence, some related works from Borkar [11] and Loh [12] point out to three-dimension stacked memory integration as a potential solution. However, to achieve high performance on chip, it is important to evaluate the shared cache memories, since they are the first alternative and level before the 3D shared memory. For this reason, there are several research works focusing on shared cache memories, the influence of application types, and their impact on performance, as follows:
On the research presented by Marino [13], the main goal is to evaluate the performance of shared L2 caches focusing on sharing versus size. In this way, two approaches were modeled: L1 and L2 private caches, and L1 private with L2 shared caches. The total number of processing cores was 32, and the cache size was modified from 1Mbytes to 4Mbytes. In order to verify the performance of all architecture combinations, the SPLASH benchmark composed of Ocean, Raytrace, Volrend, and others was executed and evaluated. The main conclusion is related to better speedup sharing L2 cache than private L2 caches increasing the speed up to 32 cores.
In the same way, the work from Jaleel et al. [14] describes an evaluation of shared cache memory, but focusing on the last cache level and parallel bioinformatics workloads. According to the paper results, the main characteristic of these workloads is the high data sharing by up to 95%. For this reason, the goal of the paper is to identify the impact of shared caches to increase the performance. The main conclusion is related to the increase of shared cache size proportionally to sharing behavior of applications. Moreover, multiple and private caches reduce the data-sharing behavior, and consequently, the performance.
The research work from Hsu et al. [15] shows many-core processors as a trend, and points out the cache hierarchy as a problem in order to support a large number of cores and threads. The main goal is to identify the effect of L2 and L3 caches, considering the different sizes and instruction addresses prefetch. The evaluation methodology focuses on on-line transaction processing benchmark (TPC-C), MPI in heterogeneous scenarios and private versus shared caches. The main results show that shared cache can provide better space efficiency and code prefetching improves performance, but it is not enough to reduce the gap with shared caches.
The work from Zahran [16] 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 same one used by Marino [13]. In this case, the programs evaluated were the following: Barnes, Water and MP3D. The environment was designed to simulate 1, 2, 4 and 8 processing cores. However, the results point out to better performance of memory system when L1 and L2 private caches are used in accordance with a good coherency protocol for L2 level.
In this paper we show some problems related to shared cache memories depicting the reduction of performance, considering different cache organizations and cache parameters such as: cache size, associativity, line size and some considerations about physical size. Moreover, our proposal takes into account the importance of memory latencies on many-core processors, which are not mentioned in related works. Consequently, our goal is to investigate performance limits, but mainly, to evaluate cache architecture configurations and present when a sharing achieves high or low performance, coupled with a parameter analysis in order to achieve best performance. To achieve these results, we used OpenMP based parallel applications from NAS parallel benchmark (NPB) [17] that represents a large set of scientific application behaviors.

3 Evaluation Method

Performance evaluation is required at every stage in the life cycle of a computer system, including its design and manufacturing [18]. On design of processor architecture some different evaluation techniques can be used, but the appropriate selection of techniques, performance metrics and workloads are very important to compare architectural and organizational changes. In terms of performance evaluation, three models must be considered: analytical, simulation and measurement.
For a general-purpose computer architecture evaluation with complex memory organization, simulations offer good features needing neither prototyping nor analytical formulations, which use to be imprecise and hard to model. Thus, for our study, all the evaluations were made using simulation techniques.

3.1 Simulation Model

The simulation environment used was Simics from Virtutech AB [19], which was chosen because it is a full-system simulator platform at the instruction set level. Thus, the results of execution time are based on executed instructions and execution cycles. Executed instructions and stall cycles generated by each architecture component gives the number of cycles.
According to the goal of this paper, Simics was configured to model some cache organizations in order to evaluate the performance of parallel workloads. The simulation model presented in **Figure 1** is an extension of the built-in g-cached model [20], which instantiates some special Simics components:
•**Id-Splitter**: To separate data between instruction and data cache.
•**Splitter**: Module to define the cache size, line size and other characteristics.

•**Trans-Staller**: Simple device used to simulate all the memory latencies.
Moreover, in order to respect all cache configurations and maintain a consistent state during the simulations some coherence parameters were defined such as:
•**Snooper**: Snooper MESI-based coherence protocol.
•**Higher Level Caches**: Parameter to ensure coherence between multiple levels on cache memory hierarchy.

3.2 NPB Workload

Application benchmarks are composed of representative subset of application functions of well-known algorithms. Thus, benchmarks are generally described in terms of functions to be performed. In this paper, the workload NPB (NAS Parallel Benchmark) [17] was used for all evaluations. This benchmark consists of a set of applications where each one represents different kernel of numerical methods described in details in **Table 1**. The NPB workload is based on version 3.3, implemented with OpenMP with W class size and compiled with SunStudio 11 using -fast performance parameter.

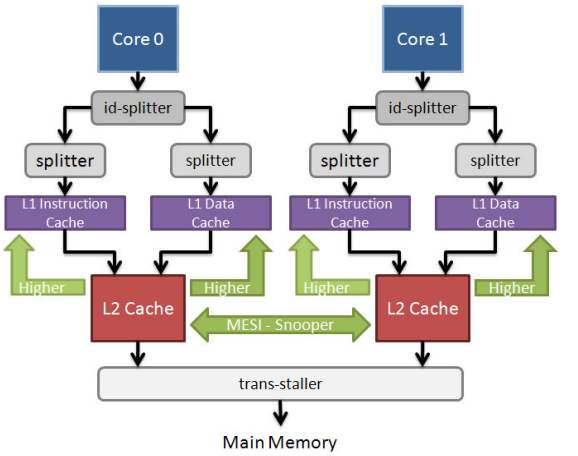

Figure 1 Model for cache memory simulation on the Simics with details of coherence protocol.

4 Cache Sharing Experiments and Results

This section presents the results obtained from NPB workload applications running on Solaris 10 operating system on the modeled experiments. For this reason all the results are from application simulation with operational system influence. The results were based on five executions of each experiment, (approximately two months of simulation time) where each experiment was executed once previously in order to warm-up the cache and reduce transient effects as cold start effects [21]. The executions were made sequentially in order to add non-determinism between the executions on the simulation environment [22].
The modeled parameters on Simics were based on a chip multiprocessor featured in 45 nm integration technology and the main memory built on 65 nm technology. The main memory and the cache memories have multiple ports and it does not represent any contention in order to investigate only the impact of shared cache. The latency values were obtained from CACTI [23] memory modeling tool version 5.3.

Table 1 NAS Parallel Benchmark applications [17].

Name	Description	Memory Usage
BT.W	To solve 3D compressible Navier-Stokes equations with an implicit algorithm. Based on Alternating Direction Implict (ADI) finite differences solver where the resulting system are Block-Tridiagonal, which are solved sequentially along each dimension.	2.7 MB
CG.W	Conjugate Gradient method used to compute the smallest eigenvalue of a large, sparse, unstructured matrix. Exercising unstructured grid computations and communications.	13.7 MB
MG.W	Multigrid V-cycle method used to solve the 3D scalar Poisson equation. The algorithm works between coarse and fine grids. It exercises both short and long distance data movement.	55.7 MB
EP.W	Embarrassingly Parallel benchmark, which generates pairs of Gaussian random. Aiming to establish the reference point for peak performance of a given platform.	1.3 MB
SP.W	Computational Fluid Dynamics (CFD) 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.	8.7 MB
LU.W	Simulated CFD application that uses symmetric successive over-relaxation (SSOR) method to the system resulting from finite-difference discretization of Navier-Stokes equations in 3D by splitting into block Lower and Upper triangular systems.	6.6 MB
IS.W	Test Integer Sort operation that is important in particle method codes. This code exercises both integer speed and communication performance.	3.4 MB
FT.W	Computational kernel of a 3D Fast Fourier Transform (FFT) method. FT performs three 1D FFT, one for each dimension.	20 MB
UA.W	Unstructured Adaptive benchmark, which exercises irregular and continually changes memory accesses measuring its effect.	16.3 MB

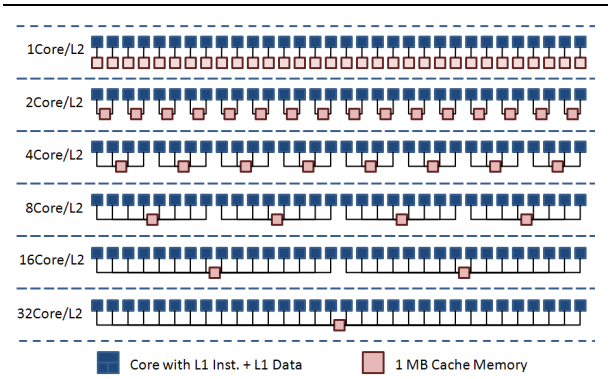

Figure 2 Cache memory organizations modeled on the base experiment.

4.1 Experiment 1: Base Experiment

The proposed experiment is based on a CMP (Chip Multiprocessor) with 32 cores. Each core has its own L1 cache with variations on L2 Cache. On the first model, 1Core/L2, each core has its own L2 cache. The second model, 2Cores/L2, each group of 2 cores shares one L2 Cache. The same occurs to 4Cores/L2, 8Cores/L2 16Cores/L2 and 32Cores/L2, where 4, 8, 16 and 32 cores share the same L2 cache, respectively. **Figure 2** illustrates all the base simulation models used as experiment on this section.

The parameters modeled on the Simics are presented in **Table 2**, where the latencies for all components were obtained by CACTI as explained previously.

The execution speedup from the base experiment is presented in **Figure 3** showing that just two applications (CG and UA) had its performance increased as more cores share the same cache. The other applications had in general its performance degraded regarding the increase on the number of cores sharing the L2 cache. It is important to notice that 2Cores/L2 organization had its performance increased just in 3 of 9 cases (BT, CG and UA). The SUM item in the plot represents the speedup based on the sum of execution time of all applications. The sums show degradations in performance by organization, where 2Cores/L2 had 0.02%, 4Cores/L2 had 1.56%, 8Cores/L2 had 4.66%, 16Cores/L2 had 6.06%, and 32Cores/L2 had 5.99% of degradation in the final performance, related to 1Core/L2 organization.

Considering the applications CG and UA, which had increased performance up to 32Cores/L2 and analyzing its descriptions, it is possible to identify that the characteristic of unaligned access is the main reason for their performance speedup. These applications access data randomly, and so, they decrease the number of address conflicts as the cache memory is shared.

Table 2 Modeled components parameters for the base experiment.

Component	Parameter	Value
Chip Multiprocessor	SO	Solaris 10
	32 Cores	UltraSparcIII+
	Execution	In-Order
	Frequency	2 GHz
	CPI	1.0
Cache L1	Size	32 KB+32 KB
	Line Size	32 B
	Set Associative	2 Ways
	Feature Size	45 nm
	R/W Latency	2 Cycles
	Interconnection Latency	2 Cycles
	Replacement	LRU
	Write Policy	Write-Through
	Write Allocation	No
Cache L2	Slice Size	1 MB / 32MB
	Line Size	64 B
	Set Associative	8 Ways
	Feature Size	45 nm
	R/W Latency	4 Cycles
	Interconnection Latency	4 Cycles
	Replacement	LRU
	Write Policy	Write Back
	Write Allocation	Yes
	Coherence Protocol	MESI Snooper
Main Memory	Size	1 GB
	Feature Size	65 nm
	R/W Latency	78 Cycles

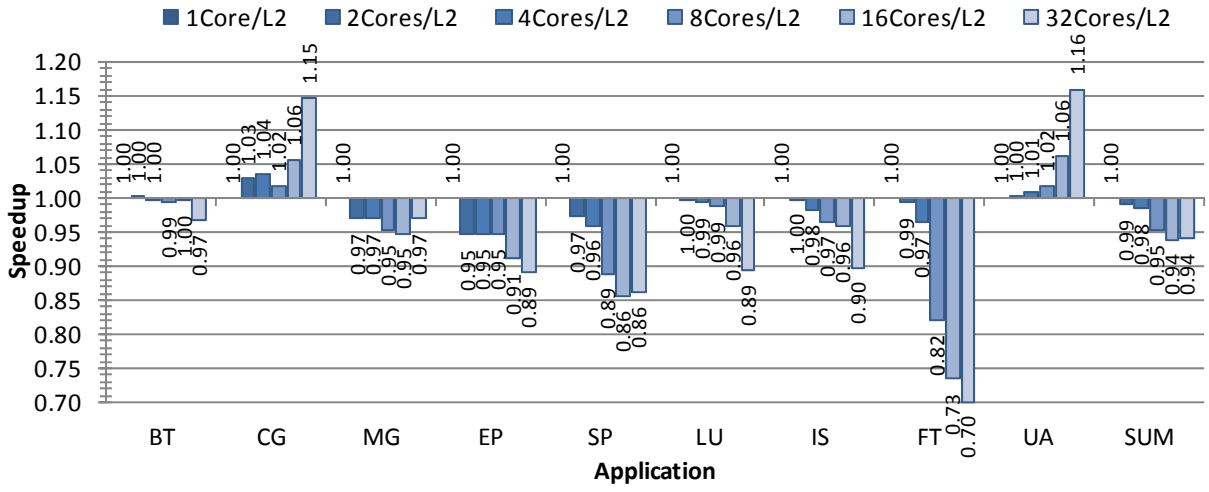

Figure 3 Speedup with base point on 1Core/L2 comparing different L2 cache memory organizations classified by application and SUM which represents sequential execution of all applications.

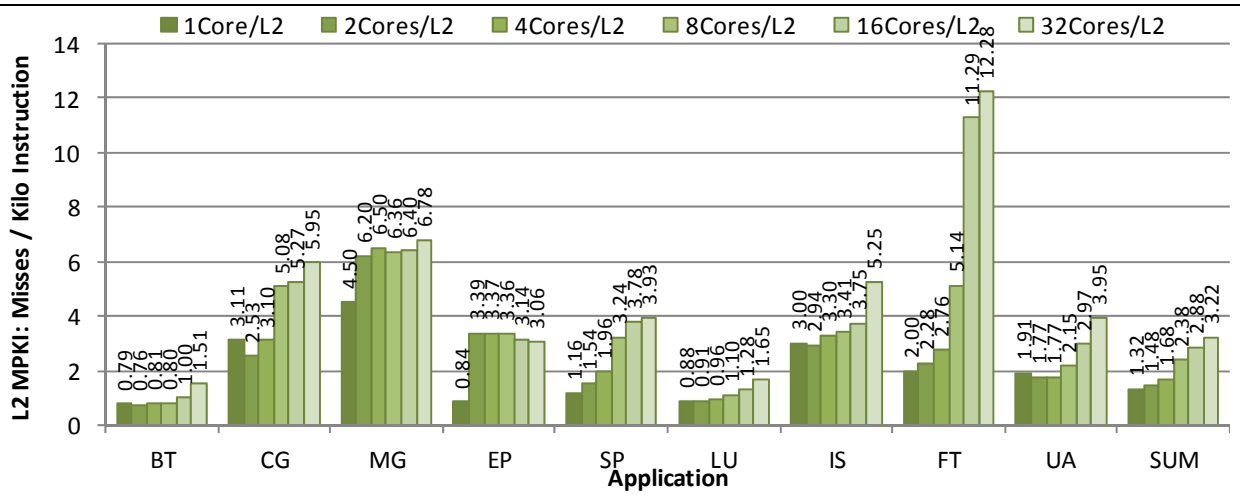

Figure 4 Number of L2 MPKI (cache misses per kilo instructions) for different L2 cache memory organizations per application, from the first experiment.

The L2 cache miss results are verified by MPKI (cache misses per kilo instructions) metric that were used in order to represent the impact of cache misses on the execution by showing the ratio between misses and instructions. **Figure 4** presents the numbers of L2 MPKI which decreases on BT, CG, IS and UA applications. From the first configuration 1Core/L2 to 2Core/L2 the reductions were -3.53%, -18.90%, -1.84% and -7.35%, respectively. From the first configuration 1Core/L2 to 4Core/L2, the reduction where obtained just for CG (-0.30%) and UA (-7.36%).

In order to investigate other possible factors that should have impact on the improvement of performance for CG and UA applications, the last metric analyzed was Executed Instructions, which can decrease as more balanced the application is, since the processor will be less idle executing operating system tasks. **Figure 5** shows the number of executed instructions for each organization per application. From the first organization (1Core/L2) to the last (32 Cores/L2), the applications CG, MG, IS and UA increased the number of executed instructions in 44.93%, 4.93%, 0.08% and 42.85%, respectively. For BT, EP, SP, LU and FT applications, the reduction on execution steps were: -0.13%, -2.75%, -3.25%, -13.42% and -27.40%, respectively.

Considering the L2 cache sharing, it is supposed to lead the system to a decrease on cache misses, as the cores can easily access shared variables. However, as more cores share the same L2 cache, their address conflicts tend to increase. In the same way, applications with random memory access characteristics can still have advantage in the shared L2 cache organizations.

Consequently, one can observe that all applications of NAS benchmark suite have different memory usage characteristics, and because of this, it is important to evaluate and identify the correct behavior of each application to achieve the highest performance.

According to the negative results obtained on this first experiment, some other experiments were planned in order to evaluate the application performance with some different L2 cache parameters, but still considering the shared L2. In this way, the next sections present results for 2Cores/L2 using different parameters as increase on the cache size, improvement on the number of ways set associativity and line size for the modeled L2 cache.

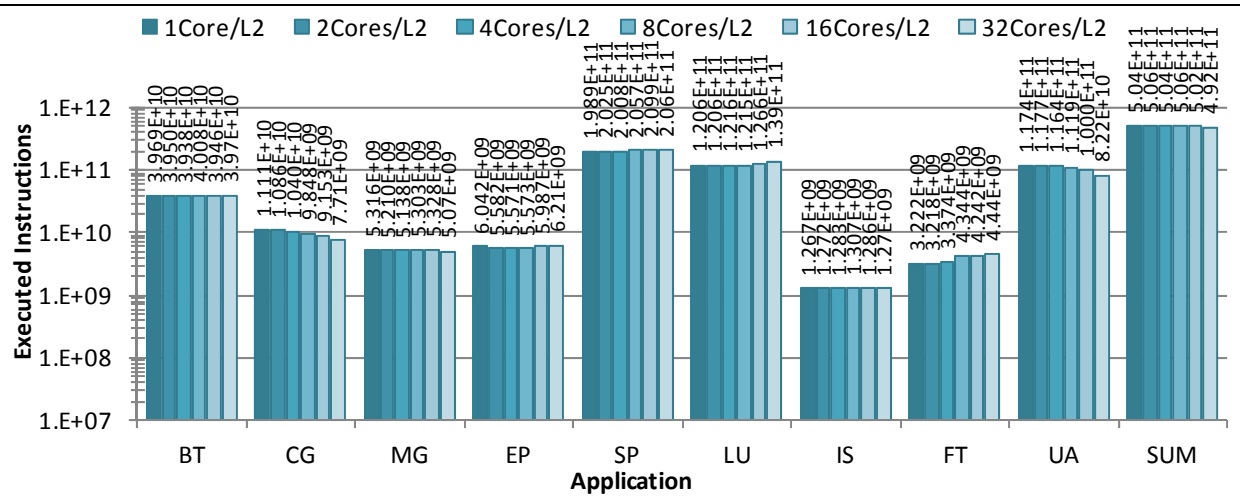

Figure 5 Number of executed instructions (operating system and applications) for evaluated cache memory organizations on the base experiment separated by application. The SUM represents all applications together.

4.2 Experiment 2: Cache Size

This experiment considers an increase on the cache size of each slice of the 2Core/L2 organization, thus, analyzing the impact of this increase on the final performance and cache misses. Notice that this increase on slice size for 2Cores/L2 (2 MB per slice, total of 32 MB) is very useful in order to compare with the first experiment which uses 32MB and 16MB on the total cache size for 1Core/L2 (1MB per slice) and 2Cores/L2 (1MB per slice) respectively. The changes on modeled parameters are presented on **Table 3**, where one can see that read/write latency increased in 1 cycle from the base experiment once cache size increased from 1MB to 2 MB on each cache slice.

Table 3 Modeled components parameters for the second experiment regarding cache size.

Component	Parameter	Value
Cache L2	Slice Size / Total Size	2 MB / 32 MB
	Line Size	64 B
	Set Associative	8 Ways
	R/W Latency	5 Cycles

Considering the change on cache size all workloads were executed for 2Cores/L2 with 2MB each slice of L2 cache. The speedup results are presented in **Figure 6** comparing the base experiment with the cache size experiment; the speedup was calculated with the base point on 1Core/L2 from the first experiment. Considering the increase on the cache size, the performance obtained had a degradation of less than 1% comparing to the 1Core/L2 modeled with 1 MB. This happened, mainly, because the increase on the cache size leads to a reduction on cache misses, which does not pay for the increase on its access time penalty that increased. Therefore, all L1 access to L2 now has this overhead.

The number of L1 and L2 cache misses per kilo instructions and L1 lost stall cycles for this experiment can be observed in **Figure 7**. One can observe the L2 cache MPKI variation from 1Core/L2 with cache slice of 1MB to

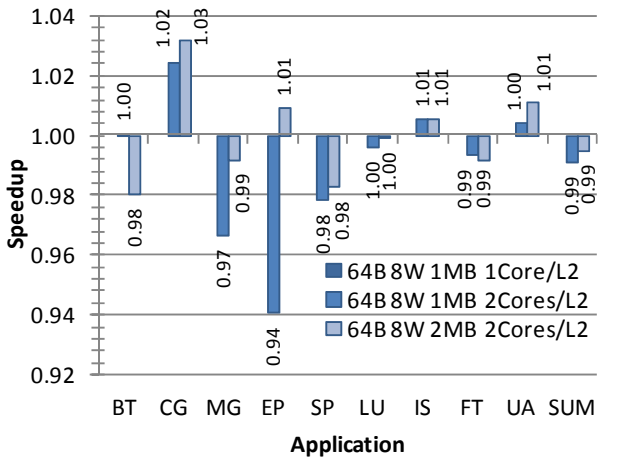

Figure 6 Speedup calculated based on 1Core/L2 from base experiment, comparing different L2 cache memory organizations classified by application and SUM which represents sequential execution of all applications.

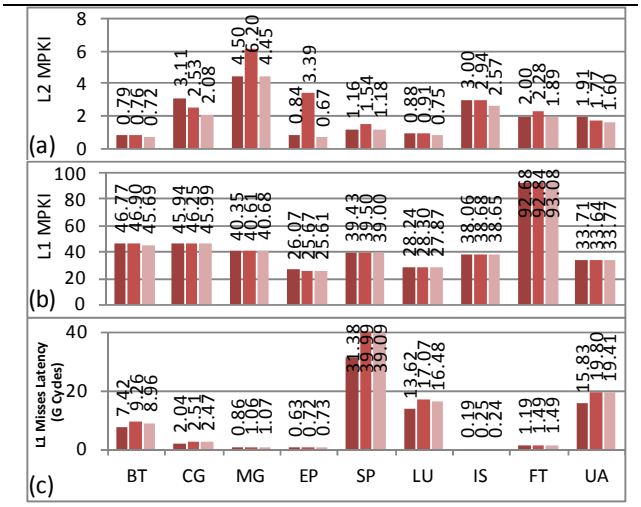

Figure 7 Number of L2 MPKI (a), number of L1 MPKI (b) and L1 lost stall cycles(c), comparing the second and first experiment.

2Core/L2 with cache slice of 2MB was -8.16% for BT, -33.15% for CG, -1.22% for MG, -20.28% for EP, 1.68% for SP, -14.34% for LU, -14.22% for IS, -5.66% for FT, and -16.40% for UA. The increase on the stalled cycles caused by the latency for the data access on the L2 cache increased the stall time from 1Core/L2 with 1MB cache slice to 2Core/L2 with 2MB cache slice of 20.67% for BT, 21.01% for CG, 24.46% for MG, 16.21% for EP, 24.57% for SP, 20.98% for LU, 25.66% for IS, 24.92% for FT, and 22.63% for UA.

For this reason, the number of L2 cache misses should decrease to lead a system to pay for the cost of the increase on the data access latency for L2 cache.

4.3 Experiment 3: Associativity

In this experiment the number of ways on the set associativity of L2 cache was doubled for all cache slices comparing to the first experiment. **Table 4** shows values for L2 cache parameters modeled on this experiment. One can see the increase on the read and write latency modeled by CACTI and caused by the increase on the associativity.

Table 4 Modeled components parameters for the third experiment about associativity.

Component	Parameter	Value
Cache L2	Slice Size / Total Size	1 MB / 16 MB
	Line Size	64 B
	Set Associative	16 Ways
	R/W Latency	6 Cycles

After modeled the parameters, the experiment was executed using 2Cores/L2 organization for all workloads.

The speedup results comparing with the base experiment is shown in **Figure 8**. Using the base point 1Core/L2 for speedup calculus, there is a total reduction of -3.46% on performance occurred as the associativity was doubled, and noticing that when the associativity increases the temporal locality is improved.

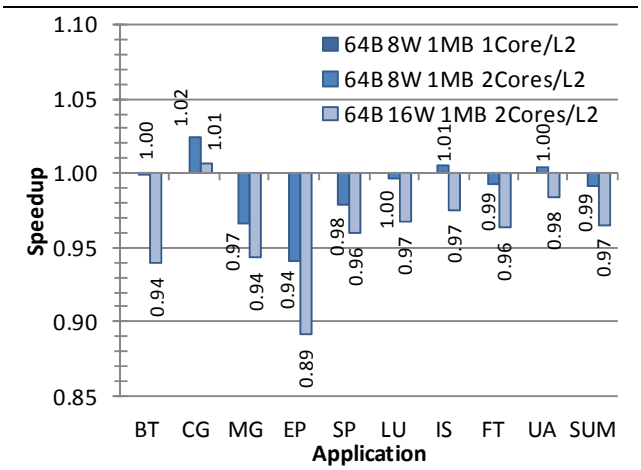

Figure 8 Speedup calculated based on 1Core/L2 from base experiment, comparing different L2 cache memory organizations classified by application and SUM which represents sequential execution of all applications.

Figure 9 shows the L1 and L2 MPKI, and the L1 stalled cycles while accessing the L2 cache. According to L2 MPKI, there was an increase of 0.24% for BT, 36.64% for MG, 281.60% for EP, 25.30% for SP, 3.86% for LU, and 14.27% for FT, and the applications CG, IS and UA obtained an decrease of -18.15%, -1.22% and -6.56%, respectively, comparing 1Core/L2 with 8-way set associativity with 2Core/L2 16-way set associativity. According to L1 lost stall cycles, the increase was higher than the second experiment, achieving 25.86% for BT, 22.42% for CG, 23.08% for MG, 14.93% for EP, 24.17% for SP, 23.14% for LU, 25.29% for IS, 25.73% for FT, and 23.82% for UA. Notice that in this case the data access time on the L2 cache was 6 cycles. Thus, once there was not high reduction on L2 cache misses it is clear that the system should suffer speed degradation.

Related to the increase on associativity, the poor results on L2 cache miss reduction were insufficient to pay for the increase on L2 cache data access latency and it leaded a system to suffer more lost stall cycles and thus the degradation on the final performance.

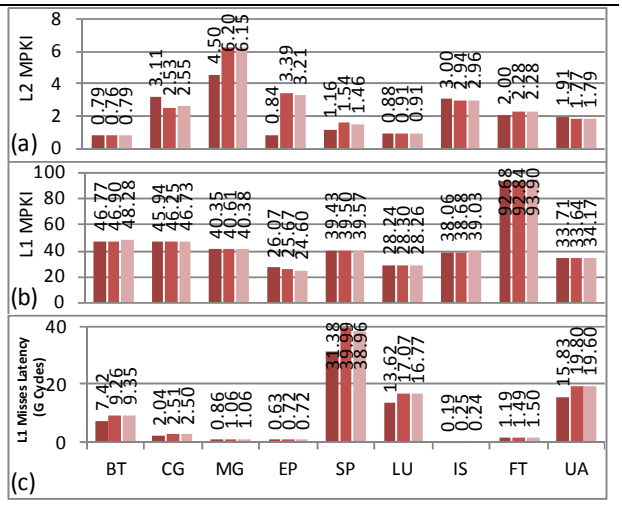

Figure 9 Number of L2 MPKI (a), number of L1 MPKI (b) and L1 lost stall cycles(c), comparing the third and first experiment.

4.4 Experiment 4: Line Size

Finally the impact of line size on the shared cache was evaluated. **Table 5** shows the parameters that were modeled to execute all workloads on 2Cores/L2 organization.

Table 5 Modeled components parameters for the fourth experiment about line size.

Component	Parameter	Value
Cache L2	Slice Size / Total Size	1 MB / 16 MB
	Line Size	128 B
	Set Associative	8 Ways
	R/W Latency	6 Cycles

The speedup results shown in **Figure 10** are based on 1Core/L2 from the first experiment as the base architecture for comparison. This increase on line size, which improves the spatial locality, leaded the system to an improvement on performance up to 1.95% comparing the 2Cores/L2 with 128B line size and 1Core/L2 with 64B line size. Moreover, there is a higher increase of 2.46% comparing 2Cores/L2 with line size of 128B and 2Core/L2 using line size of 64B.

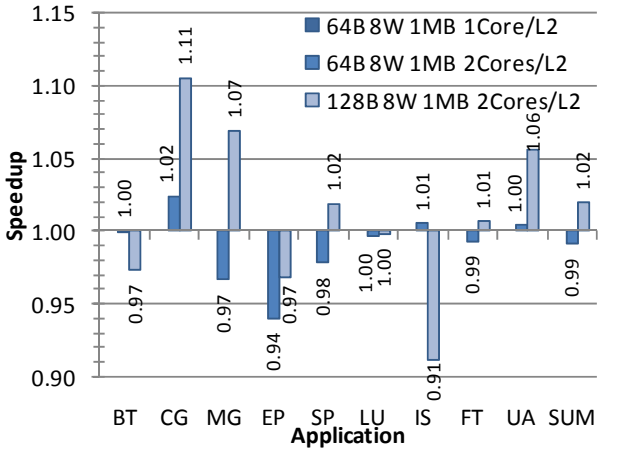

Figure 10 Speedup calculated based on 1Core/L2 from base experiment, comparing different L2 cache memory organizations classified by application and SUM which represents sequential execution of all applications.

L1 MPKI and penalties for accessing the L2 cache and L2 MPKI are presented in **Figure 11.** The variation on the L2 cache misses and L1 misses with latency has leaded to an increase on the final speedup. Comparing the 1Core/L2 from the base experiment with 2Core/L2 of this experiment, the number of L2 MPKI reduced -35.29% for BT, -41.70% for CG, -19.35% for MG, -23.13% for SP, -37.51% for LU, -33.38% for FT, and -24.76% for UA as the line size increased. For EP and IS there were an increase of 106.20% and 87.24% respectively, but this increase has no impact on the final performance.

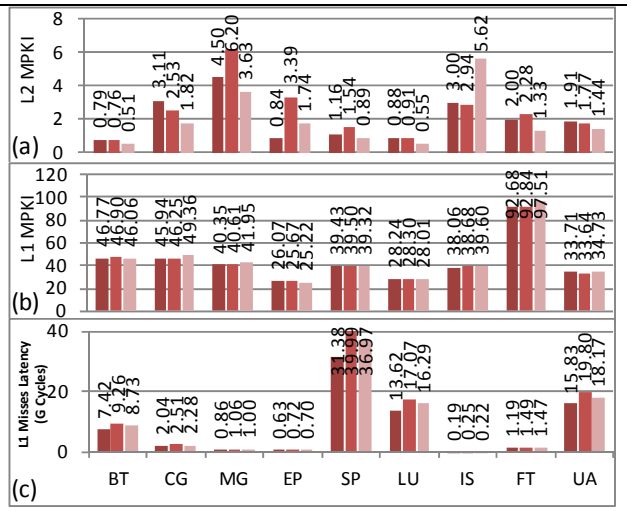

Figure 11 Number of L2 MPKI (a), number of L1 MPKI (b) and L1 lost stall cycles(c), comparing the fourth and first experiment.

The L1 lost stalled cycles increased 17.56% for BT, 11.75% for CG, 16.27% for MG, 11.47% for EP, 17.82% for SP, 19.63% for LU, 16.18% for IS, 23.02% for FT, and 14.79% for UA. Considering the sum of all these results, there is a great reduction on L2 misses and a low increase on L1 stall cycles leaded the system to an increase on the final performance.

4.5 Summary

This section presents general comparisons among all experiments presented on the previous sections. **Figure 12** presents the speedup results for all experiments shown on this paper, where one can see the bad results increasing the cache size and associativity. However, according to line size, an increase on the performance of 1.95% was achieved, showing the great importance of increase the spatial locality, which in our case had greater importance than the increase on capacity of the L2 cache. Related to the increase on the set associativity, which increases the temporal locality, does the system leaded to bad results once the allocation size of the cache had decreased, generating more cache misses. Considering the different experiments, all of them leaded to an increase on access time on L2 cache, in this way, it is very clear the importance of having a good trade off between penalty time and cache misses.

When the comparison comes to the L2 cache misses per kilo instructions, presented in **Figure 13**, the increase on the sharing cache (base experiment) and associativity (third experiment) leads the MG and EP application to bad results, causing a high L2 cache misses. Looking for these specific applications, the cache size (second experiment) showed to be the best solution to decrease the cache misses. For 7 of 9 applications the increase on line size (fourth experiment) leaded the system to a decrease on the L2 cache misses showing the best choice for performance when L2 cache is shared.

4.6 Physical Area Analysis

Since the parameters evaluated on the previous section have an impact on the physic cache, this section considers the physical cache values, comparing the different memory implementations shown in this paper. **Table 6** brings more detailed cache values of all cache parameters modeled on our experiments, showing values of Cache Physical Area, matching these cache values to the modeled organizations. All the cache slices were modeled on CACTI as just one banking cache memory leading to a reduction on the number of stall cycles waiting cache access.

But considering the physical size of the cache, it is clear that the increase on the line size leads the system to a best result of cache misses, decreasing both total physical area and total number of cache misses comparing to the organization 1Core/L2 from base experiment. On the other hand, the increase on slice size, leads to a great reduction of cache misses, but with great cost on physical size. On a many-core context this reduction on cache size is very im-

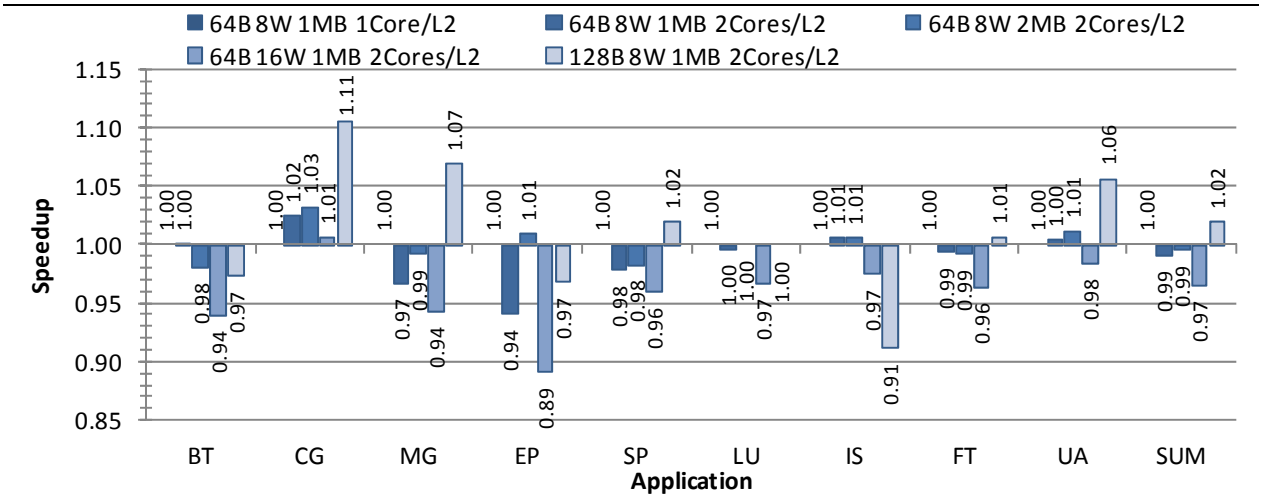

Figure 12 Speedup calculated based on 1Core/L2 from base experiment, comparing different L2 cache memory organizations classified by application and SUM, which represents sequential execution of all applications. Comparing the base experiment with experiments varying cache size, associativity and line size.

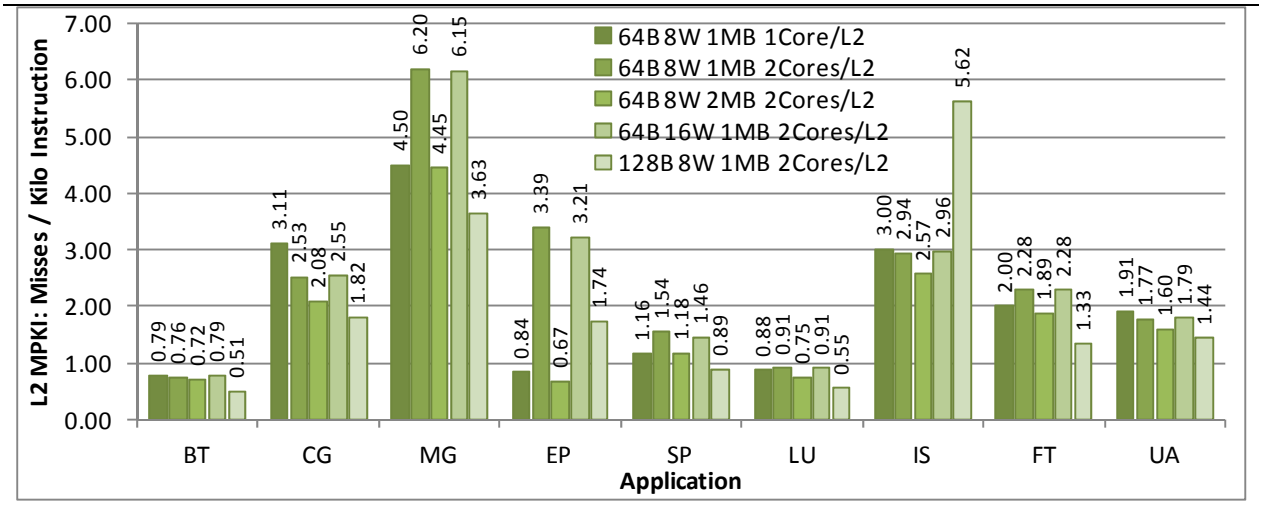

Figure 13 Number of L2 MPKI (cache misses per kilo instructions) for different L2 cache memory organizations per application comparing all experiments.

Table 6 Physical area analysis considering different cache organizations and parameters.

Exp.	Organization	Total Logical Size	Slice Logical Size	Associativity	Line Size	Normalized Physical Area	Total Physical Area	Slice Physical Area	Latency	Penalty	Normalized Cache Misses
1st	1Core/L2	32MB	1MB	8 Ways S.A.	64 Bytes	100%	230.08mm^2	7.19mm^2	1.6 ns	4 Cycles	100%
1st	2Cores/L2	16MB	1MB	8 Ways S.A.	64 Bytes	50%	115.04mm^2	7.19mm^2	1.6 ns	4 Cycles	112%
2nd	2Cores/L2	32MB	2MB	8 Ways S.A.	64 Bytes	78%	180.48mm^2	11.28mm^2	2.1 ns	5 Cycles	89%
3rd	2Cores/L2	16MB	1MB	16 Ways S.A.	64 Bytes	50%	117.44mm^2	7.34mm^2	2.6 ns	6 Cycles	108%
4th	2Cores/L2	16MB	1MB	8 Ways S.A.	128 Bytes	73%	169.92mm^2	10.62mm^2	2.6 ns	6 Cycles	68%

portant, since all the saved space can be used to increase the number of cores and interconnections.

4 Conclusions

Due to the application performance, the cache memory organization for next generation of many-core processors is not very clear. This paper evaluated some cache memory organizations in order to investigate the L2 cache sharing impact on a 32-core processor.

On the base experiment each L2 cache slice had just 1 MB even with more cores sharing the same L2 cache. Once these variations on the organization do not resulted on a good performance, the cache size, associativity and line size were changed and modeled on the point that base experiment showed the first reduction on speedup.

Spotting future architectures, the results show that as more cores share the same L2 cache, the performance may be degraded even for just 2 cores sharing the same L2. However, our evaluations presented that line size helped on reduction of -32% on cache misses, increased the system performance on 2% for NPB workload, even reducing -50% the total amount of memory from 32MB to 16MB. Besides, there is a reduction on the total area of cache in -27%, increasing space for processor cores or on-chip interconnections.

Finally, due to characteristic of random access, the applications achieve a good reduction on cache misses, since the number of address conflicts tends to decrease and, on the other hand, the communication using shared cache tends to be improved.

Our future works focus on the extension of the cache sharing study analyzing the impact of shared cache on Non-Uniform Cache Architectures (NUCA) on scientific parallel applications for many-core architectures.

4 Acknowledgment

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

5 References

[1] J. L. Hennessy and D. A. Patterson, Computer Architecture: A Quantitative Approach. Elsevier, Inc., United States of America, fourth edition, 2007.

[2] J. E. Smith, G. S. Sohi, “The Microarchitecture of Superscalar Processors”, IEEE, v. 83, No. 12, pp.1609-1624, 1995.

[3] Stallings, W., Computer Organization and Architecture, Pearson / Prentice Hall, 2002.

[4] T. Ungerer, et al., “A Survey of Processors with Explicit Multithreading”, ACM Computing Surveys, Volume 35, Issue 1, pp.29-63, March 2003

[5] T. Ungerer, et al., “Multithreaded Processors”, The Computer Journal, British Computer Society, v. 45, n. 3, p. 320-348, 2002.

[6] B. Sinharoy, et al., “POWER5 system microarchitecture”, IBM J. RES. & DEV, Vol.49 No. 4/5 July/September 2005

[7] K. Olukotun, et al., “The Case for a Single-Chip Multiprocessor”, International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), pp.2-11, October 1996.

[8] L. A. Barroso, et al., “Piranha: a scalable architecture based on single-chip multiprocessing” - 27th International Symposium on Computer Architecture (ISCA), pp.282–293, 2000.

[9] P. Kongetira, et al., “Niagara: a 32-way multithreaded Sparc processor”, IEEE MICRO, v. 25, Issue 2, p. 21-29, March-April 2005.

[10] R. Kumar, et al., “Heterogeneous chip multiprocessors”, IEEE Computer, v. 38, Issue 11, p. 32-38, November 2005.

[11] S. Borkar, “Thousand Core Chips: A Technology Perspective”, ACM Annual Conference on Design Automation, pp.746-749, 2007.

[12] G. H. Loh, “3D-Stacked Memory Architectures for Multi-Core Processors”, IEEE International Symposium on Computer Architecture, pp.453-464, 2008.

[13] M. D. Marino, “L2-cache hierarchical organizations for multi-core architectures”, International Symposium on Parallel and Distributed Processing and Applications (ISPA), Sorrento, Italy, pp.74-83, 2006.

[14] A. Jaleel, M. Mattina, B. Jacob, “Last Level Cache (LLC) Performance of Data Mining Workloads on a CMP: A Case Study of Parallel Bioinformatics Workloads”, IEEE international Symposium on High-Performance Computer Architecture, pp.88-98, 2006.

[15] L. Hsu, et al., “Exploring the Cache Design Space for Large Scale CMPs”, ACM SIGARCH Computer Architecture News, Vol. 33, No, 4, pp. 24-33, September 2005.

[16] M. M. Zahran, “On cache memory hierarchy for chip-multiprocessor”, ACM SIGARCH, Vol. 31, Issue 1, pp.39-48, 2003.

[17] H. Jin, M. Frumkin, and J. Yan, “The OpenMP Implementation of NAS Parallel Benchmarks and its Performance”, In NAS Technical Report NAS-99-011, NASA Ames Research Center, 1999.

[18] R. Jain, The art of computer systems performance analysis. J. Wiley, New York, 1991.

[19] P. S. Magnusson, M. Christensson, J. Eskilson, et al., “Simics: A full system simulation platform”, IEEE Computer Micro, 2002.

[20] Virtutech, “Simics 3.0 – User Guide for Unix”, Revision 1376, http://www.simics.net, 2007

[21] A. R. Alameldeen, C. J. Mauer, M. Xu, et al. “Evaluating non-deterministic multi-threaded commercial workloads”. Computer Architecture Evaluation using Comercial Workloads, 2002.

[22] A. R. Alameldeen and D. A. Wood “Variability in architectural simulations of Multi-threaded Workloads”. International Symposium on High Performance Computer Architecture (HPCA), February 2003.

[23] S. Thoziyoor, J. H. Ahn, M. Monchiero, et al. "A Comprehensive Memory Modeling Tool and Its Application to the Design and Analysis of Future Memory Hierarchies," ISCA 35th International Symposium on Computer Architecture vol., no., pp.51-62, 21-25 June 2008