Design Space Exploration of CMPs with Caches and Local Memories

Lluc Alvarez*†, Ramon Bertran*†, Marc Gonzàlez*†,
Xavier Martorell*†, Nacho Navarro*†, Eduard Ayguadé*†

*Departament d'Arquitectura de Computadors
Universitat Politècnica de Catalunya
Barcelona, Spain.

†Barcelona Supercomputing Center
Barcelona, Spain.
name.surname@bsc.es

ABSTRACT

Chip multiprocessors (CMPs) are the dominating architectures nowadays. There is a big variety of designs in current CMPs, with different number of cores and memory subsystems, because they are used in a wide spectrum of domains and so its best configuration highly depends on several design goals such as performance, energy consumption, scalability, area and programmability. This paper studies different chip configurations in terms of number of cores, size of the shared L3 cache and off-chip bandwidth requirements in order to find what is the most efficient design for High Performance Computing applications. In addition, it analyzes two types of CMPs: CMPs with a traditional cache hierarchy, or cache-based CMPs, and CMPs with both cache hierarchy and local memories, or hybrid memory CMPs.

Results show that, for HPC workloads, cache-based cores perform better when the shared L3 cache is reduced in order to make room for additional cores and the bus is able to provide a high bandwidth, reaching a speedup of 3.31x against a baseline architecture. The best chip configurations of hybrid memory CMPs are the ones with a moderate number of cores and not so mall L3 caches and, furthermore, they don't need a bus with high bandwidth. They achieve a maximum speedup of 3.06x against the baseline architecture. In the direct comparison between the chip configurations of the two types of CMPs, hybrid memory CMPs outperform cache-based CMPs in almost all configurations, achieving a maximum speedup of 1.74x.

1. INTRODUCTION

Chip multiprocessors (*CMPs*) are the dominating architectures nowadays and, probably, in the future. CMPs are used in a wide spectrum of domains: high performance computing (*HPC*), servers, commodity desktops, gaming, embedded systems, etc. The majority of CMP designs that exist nowadays are based on the replication of several cores inside the same chip sharing up to some degree a complex memory subsystem. It is the relation between the number of cores and the memory subsystem design that has become crucial for a CMP design, given that this relationship has an immediate impact on the overall performance and scalability of the architecture [14, 17, 22].

Current CMP architectures show different organizations in their number of cores and their memory hierarchy. The IBM POWER7 [24] and the most powerful Intel processors based on the Nehalem microarchitecture [3] have 8 cores a large shared L3 cache. Others like the AMD Phenom, based on the AMD K10 architecture [1], and the Sun Microsystems UltraSPARC T3 [23] have a very small shared last level cache and different number of cores: 4 and 16, respectively. Beyond that, some newer designs like the Intel SCC [4] completely eliminate the shared cache in order to make room for more cores, reaching 48 simple P54C-based cores on chip. Besides this variety, some processors have asymmetrical or heterogeneous cores and memory models. For example, the IBM Cell [13] includes 9 cores (1 general-purpose core and 8 accelerators) and a memory organization that combines the use of local memories and caches. These examples demonstrate that the optimal chip configuration has not been found yet, since it highly depends on several design goals such as performance, energy consumption, scalability, area and programmability. However, all these CMP schemes have some common characteristics: a number of cores as big as possible, given the area and technology limitations, and a memory hierarchy with a low-latency, high-bandwidth size-constrained stack of intermediate memories, plus a final huge high-latency, low-bandwidth off-chip main memory.

The trend in the HPC domain is to use CMPs but it is not clear what is the best CMP design for this environment, since all kind of general purpose CMPs have been used to build supercomputers in the last years. In order to find what is the CMP configuration that better fits in HPC domains it is important to know what are the characteristics of HPC applications. In general, HPC applications are dominated by parallelizable computational loops that operate on huge data sets. The data sets are normally implemented in the form of matrices and vectors, and they are sequentially traversed during the execution of the computational loops, which implies having predictable access patterns with a high degree of spacial locality and a uniform distribution of accesses to the shared cache and to the main memory. Depending on how the computational loops are distributed in the application it is also possible to find temporal locality in memory accesses. These characteristics seem to fit better in architectures with a high number of cores because they are able to take more profit of the high degree of parallelism existent in HPC applications, as proven by the fact that GPGPUs are becoming a very important component of the most powerful supercomputers that currently exist [5]. Following this trend and the one opened by the design of the Intel SCC processor it is interesting to study the behaviour of CMP configurations derived from reducing the size of the shared cache in order to add more cores to the chip.

This paper studies different chip configurations in terms of number of cores, size of the shared L3 cache and off-chip bandwidth requirements. Starting from a multicore with a very large L3 cache, in this paper it is proposed to reduce the amount of area dedicated to the shared L3 cache and use this area to put more cores in the chip. Decreasing the capacity of the L3 cache frees an amount of area in the chip, but this amount of area can vary significantly depending on the technology and the design of the cache. The area of a core is also very dependent on the technology, the design of its private caches and the complexity of its pipeline. Due to these variabilities it is possible to have very different chip configurations. Adding cores to the chip has the immediate consequence of being able to execute the parallel loops faster, in general. The problem is that reducing the capacity of the L3 cache in order to make room for these extra cores has two important implications. Firstly, current cache coherence protocols limit the number of cores in a CMP because of their lack of scalability. Secondly, reducing the size of the L3 cache increases its miss ratio, in the general case, and this increase in the miss ratio generates traffic between the chip and the main memory, so the bus has to be able to deliver the required bandwidth in order to maintain an acceptable performance. These two problems can be alleviated by adding local memories to the cores [7], since local memories do not generate coherence traffic and make a better utilization of the bus bandwidth. This is the reason why this paper takes into account two types of cores: cores with a traditional cache hierarchy, or *cache-based cores*, and cores with a local memory side to the cache hierarchy, or *hybrid memory cores*.

The main contribution of this paper is the study of the relation between the number of cores, the size of the L3 cache and the off-chip bandwidth for CMPs with two kinds of memory models: a hardware cache hierarchy and a hybrid memory model that combines the cache hierarchies with on-chip local memories. The study results in several chip configurations that include a number of cores that range from 16 to 28, L3 cache sizes that vary between 4.5MB and 21MB and bus bandwidths from 20GB/s to 65GB/s. All of these chip configurations are realistic and report speedup numbers of 2.13x to 3.31x for a representative set of loops coming form the HPC domain.

The rest of this paper is organized as follows: Section 2 presents some related work. Section 3 describes the architecture of a CMP and its design parameters as well as the microarchitecture of the cache-based core and the hybrid memory core. Section 4 explains the experimental framework and the methodology. Section 5 evaluates the performance of the different chip configurations. Finally, Section 6 concludes this paper with a summary of its main ideas and results.

2. RELATED WORK

The design space exploration of CMPs has been a very relevant topic since these architectures appeared, so many works have been done on it from the point of view of different metrics.

From the pure performance perspective, J. Huh et al. [12] study the performance of CMPs under several design decisions concerning core complexities, cache hierarchies and available off-chip bandwidths, using several single-threaded applications to evaluate their proposals. A similar study is done by M. Ekman et al. [9] using parallel HPC workloads.

Many other works include power consumption in their explorations. J. Li and J. F. Martinez [18, 19] explore parameters such as the number of cores and the voltage/frequency scaling with analytical and experimental models to study power efficiency. R. Kumar et al. [15] focus on the characteristics of the cores, proposing to use fully custom heterogeneous cores to improve the performance of CMPs given power and area constraints. Another work [16] by the same group analyse the most commonly used on-chip interconnections in CMPs, adding also area and scalability considerations to the study.

Some other works also include temperature in their studies. Y. Li et al. [20] do a multidimensional design space exploration of CMPs given thermal constraints, concluding that these constraints dominate over other physical constraints. Similar conclusions are extracted from the work of M. Monchiero et al. [21], where they evaluate performance, power and temperature of chip configurations with different number of cores, issue width of the cores and size of the L2 caches, that can be private or shared.

The main difference between this work and the ones just reviewed is that they do not include hybrid memory cores in their studies. Although the introduction of local memories (also called scratchpad or streaming memories) in a core is not novel, nobody has done a design space exploration of CMPs with such memory model. In addition, most of the previous work regarding local memories propose to replace caches with local memories, not to combine them like this work proposes to.

A work by J. Levich et al. [17] compares the two memory models in a CMP. The authors design a CMP with caches and another one with local memories, fixing the number of cores and the size of the shared cache, and compare the two designs in terms of performance, bandwidth utilization and energy consumption. The study the authors do is very different from this one because, firstly, they don't do a design space exploration work varying the design parameters of the chip and, secondly, they use caches or either local memories, but don't propose to build a hybrid memory model.

R. Bertran et al. [8] propose to have a core with both the classical cache hierarchy and a local memory. In their work the authors describe how the local memory fits in the core, how it interacts with the cache hierarchy and how this hybrid memory model is managed by the hardware and the compiler. This work assumes almost all the design decisions taken in the work by R. Bertran but, at the same time, differs a lot from that work, since they evaluate the performance and power consumption of a single hybrid memory core while this work constructs a CMP with such cores and analyzes its performance, discussing the effect of varying different design parameters of the chip.

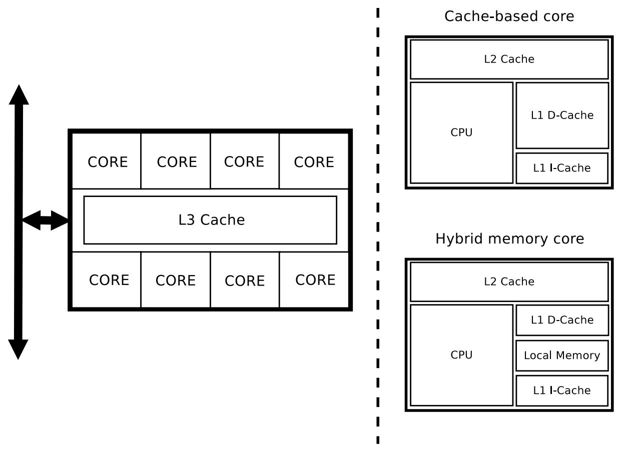


Figure 1: Scheme of the CMP architecture.

3. ARCHITECTURE AND DESIGN SPACE

This section describes the architecture of the CMP this work assumes and enumerates its main design parameters. After that the microarchitecture of both cache-based cores and hybrid memory cores is presented. At the end it is explained what values have been studied for each design parameter of the CMP in order to do the design space exploration.

3.1 CMP architecture

Figure 1 shows an scheme of the architecture this work assumes. The CMP is composed of *num_cores* cores, a shared L3 cache of size *L3_size* and an interconnection network that connects these components. The chip is connected to a bus with a bandwidth of BW that is used to access main memory. Cores can be either cache-based cores of hybrid memory cores. The former cores have a 32KB L1 I-cache, a 64KB L1 D-cache and a unified 256KB L2 cache. The latter cores have a 32KB L1 I-cache, a private 32KB L1 D-cache, a 32KB local memory and a unified 256KB L2 cache. Depending on the kind of cores a CMP is composed of, it can be either a *cache-based CMP* or a *hybrid memory CMP*.

Since the aim of this work is to study what is the behaviour of the two CMP models with different chip configurations of *num_cores*, *L3_size* and BW, a set of values has been fixed for each parameter. In order to obtain realistic chip configurations, it has been defined two parameters α and β. The parameter α corresponds to the relation between the area of 1MB of L3 cache and the area of a core. The parameter β is defined as the minimum amount of memory in L3 cache per core. This means that in any chip configuration the size of the L3 cache is forced to be equal to $\beta \times$ *num_cores*. Given a baseline chip configuration and a pair of values for α and β one can derive new chip configurations by reducing *L3_size* and increasing *num_cores* according to the given parameters. Using this methodology it is possible to avoid undesired chip configurations, since α controls that the area of the chip configurations never exceeds the area of the original chip configuration and β ensures a balance between the number of cores and the resulting L3 cache. Section 3.3 justifies the values for α and β and describes all explored chip configurations.

3.2 Core microarchitecture

The cache-based core used is an Intel Nehalem-like core, with a private 64KB L1 D-cache, a private 32KB L1 I-cache and a private 256KB L2 cache. The CPU has a 4 instructions-width pipeline so, each cycle, up to 4 instructions are fetched, decoded, renamed, issued and committed. The reorder buffer has 96 entries, the issue queue 64 entries and the load/store queue 80 entries. The CPU has also 3 ALUs and 256 registers for integer operations and, symmetrically, it also has 3 ALUs and 256 registers for floating point operations. Memory operations are executed by 2 load/store units.

The hybrid memory core has exactly the same CPU as the cache-based core. The memory hierarchy, though, is slightly different. The L1 I-cache and the L2 cache are identical, but the L1 D-cache is reduced to 32KB in order to make room for a 32KB local memory. The way the local memory is integrated in the core is very similar to the one proposed by R. Bertran et al. [8]. The authors of that work extend the architecture of general-purpose processors by adding a software managed local memory and a very simple *programmable DMA controller (PDC)*. The local memory is used to store only data. The local memory is accessed reserving a range of physical addresses for it. This address range is direct-mapped to logical addresses, so when a memory operation is executed a range check is performed for each logical address generated. This check is done in parallel with the segmentation mechanism, prior to any MMU action. In order to move data between the local memory and the main memory a simple PDC is also added to the core. The PDC only supports asynchronous get (transfer from the main memory to the local memory) and put (transfer from the local memory to the main memory) operations to and from aligned addresses. The issue of a DMA command is notified by the CPU to the PDC via non-cacheable store instructions to memory-mapped PDC registers. DMA commands are enqueued in the DMA command queue and are executed in order. When a DMA command is executed it is split in bus requests, that are kept in a bus request queue and are also issued in order. For a get operation, each bus request snoops the L3 cache. If the data is present in the L3 cache it is moved from there to the local memory, otherwise it is moved from the main memory to the local memory. For a put operation, the line is transferred to the main memory and, in case the line is present in the L3 cache, it is invalidated. This mechanism guarantees memory coherence for DMA transfers but leaves memory consistency as a responsibility for the software.

3.3 Architectural parameters exploration

A set of values has been fixed for each CMP design parameter. For α, floorplans of modern architectures such as the IBM POWER7, the Intel Nehalem and the AMD K10 have been examined and, from there, observed that the ratios between the area of 1MB of L3 cache and the area of a core range from 0.25 to 0.5. Following this observation, the values used for α are 0.25, 0.33 and 0.5. For β a wide range of values has been set. The highest value observed in existing architectures is 4MB of L3 cache per core in the IBM POWER7, followed by 2MB of L3 cache per core in the Intel Nehalem. On the other side, the AMD K10 has only 512KB of L3 cache per core. In order to cover a representative spectrum of possibilities, the set of values fixed for β

Table 1: Chip configurations.

		α 0.25	α 0.33	α 0.5
β	6MB	8 cores 48 MB L3	8 cores 48 MB L3	8 cores 48 MB L3
	4MB	10 cores 40 MB L3	10 cores 40 MB L3	10 cores 40 MB L3
	2MB	13 cores 26 MB L3	14 cores 28 MB L3	16 cores 32 MB L3
	1MB	16 cores 16 MB L3	18 cores 18 MB L3	21 cores 21 MB L3
	512KB	17 cores 8.5 MB L3	20 cores 10 MB L3	25 cores 12.5 MB L3
	256KB	18 cores 4.5 MB L3	22 cores 5.5 MB L3	28 cores 7 MB L3

is 6MB, 4MB, 2MB, 1MB, 512KB and 256KB of L3 cache per core. The baseline chip configuration used in this paper is a chip with *num_cores* = 8 and *L3_size* = 48MB. This baseline chip configuration is used as the starting point to derive new chip configurations combining values of α and β. The resulting chip configurations are shown in Table 1.

The values for BW have also been derived from modern chips. The trend in current CMPs is to use HyperTransport [2] for off-chip communication, which has bandwidths that go from 12.8GB/s in HyperTransport 1.0 to 51.2GB/s in HyperTransport 3.10. The set of values used for BW are 20GB/s, 35GB/s, 50GB/s and 65GB/s.

4. EXPERIMENTAL FRAMEWORK AND METHODOLOGY

This section explains the experimental framework used to gather the results. In Figure 2 it can be observed that the experimental framework has two main components: PTLsim and CMP performance estimator. PTLsim is a single core simulator that is used to do detailed simulations of the execution of different benchmarks on different core configurations. The single core results of the simulations are fed to the CMP performance estimator along with the corresponding CMP configurations in order to derive what would be the performance of the chip. To do that the CMP performance estimator uses performance and bandwidth models. The next two subsections explain the main characteristics of these two components. The first one describes the simulation environment and the benchmarks that have been used. The second subsection discusses how the numbers extracted from the simulations have been used to study the behaviour of the whole set of chip configurations.

4.1 Simulation environment

For the performance evaluation, PTLsim [25] has been used to simulate 28 computational loops from four different NAS benchmarks [6]. The benchmarks are typical HPC workloads: CG is a conjugate gradient algorithm, FT computes a Fourier transformation, IS does an integer sort and MG realizes 3-dimensional multigrid relaxation with periodic boundary conditions. The main reason for presenting per loop results is that they allow the behaviour of the proposal to be studied for different memory access patterns (regular and regular/irregular). The overall application results follow the same trends as the ones presented because the evaluated applications are dominated by these computational loops. The simulator has been extended to include a local memory and a PDC in the hybrid memory core. In addition, the main architectural parameters in the simulator have been tuned so that it models the core microarchitecture explained in Section 3.2, as can be observed in Table 2. The 28 computational loops have been compiled using GCC 4.0 with the -O3 optimization flag on. 150 millions of x86 instructions have been simulated for each loop, except in those cases where the loop finishes before reaching that threshold.

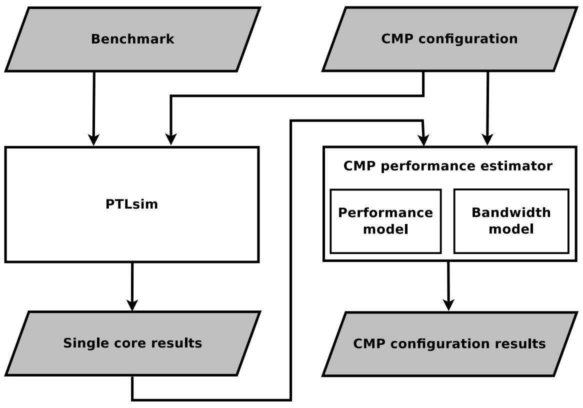


Figure 2: Scheme of the experimental framework.

4.2 CMP performance estimation

PTLSim simulates a single core, so from the simulation results the behaviour of a multicore has to be estimated. To do that, the CMP performance estimator splits the work to be done for each loop in N units of work. For each β and BW, this module receives the single core simulation results of the execution of the N units of work of every loop on both a cache-based core and a hybrid memory core. The CMP results are obtained by applying a performance model based on the Amdahl's law for multicores [11] to the simulation results. Three main reasons support this methodology. First, the studied loops are completely parallel and so the whole execution time can be divided by the number of cores. Secondly, the N units of work done in every loop are uniformly distributed among its iterations, causing no work imbalance between cores. Besides, the only core synchronization point in the whole loop execution happens at its end. Due to the small amount of cores in this study, the overheads related to fork/join operations can be considered negligible. Thirdly, the data set used by each unit of work is disjoint so it can be assumed that, after the work distribution, the cores do not interfere in the shared L3 cache.

For cache-based CMPs, the CMP performance estimator uses equation 1 to calculate their times. A chip configuration is defined by three values: *num_cores*, *L3_size* and BW. For simplicity, and since β = *L3_size* / *num_cores*, a chip configuration can also be defined using its β instead of its *L3_size*. Following this nomenclature, let $T\_multicore_{num\_cores,\beta,BW}$ be the time it would take to execute the N units of work of a loop in a chip configuration defined by *num_cores*, β and BW, and let $T\_core_{\beta,BW}$ be the time it takes a single cache-based core with an L3 cache of β and a bandwidth of BW to execute the N units of work of that loop. The execution time in the chip configuration will be the time of the single core divided by the number of cores.

Table 2: PTLsim configuration parameters.

PARAMETER	CONFIGURATION
Fetch width	4 instructions
Decode width	4 instructions
Rename width	4 instructions
Issue width	4 instructions
Commit width	4 instructions
Branch predictor	Hybrid 4K selector 4K G-share 4K Bimodal 4K BTB 4-way RAS 32 entries
Functional units	3 integer ALUs 3 floating point ALUs 2 load/store units
Register file	256 integer registers 256 floating point registers
L1 I-cache	32 KB 8-way set-associative 64-byte lines 2 cycles latency
L1 D-cache	Size depends on the configuration studied 8-way set-associative 64-byte lines 2 cycles latency
L2 cache	256KB 24-way set-associative 64-byte lines 15 cycles latency
L3 cache	Size depends on the configuration studied 32-way set-associative 64-byte lines 40 cycles latency
Local memory	Size depends on the configuration studied 2 cycles latency

$$T\_multicore_{num\_cores, \beta, BW} = \frac{T\_core_{\beta, BW}}{num\_cores} \qquad (1)$$

Notice that the single cache-based core and the chip configuration have the same bandwidth. The bandwidth required by the loop executions in the different chip configurations has to be also estimated in order to ensure that the bandwidth provided by the bus is enough to execute the loops without any performance penalty. The CMP performance estimator uses equation 2 to calculate the required bandwidth. The average bandwidth required by the single core execution is obtained dividing the number of requests to main memory $Requests\_MM\_core_{\beta, BW}$ by the execution time $T\_core_{\beta, BW}$. The average bandwidth required by each core in the chip configuration is the same as the one in the single core execution, since every unit of work has the same bandwidth requirements, and units of work are distributed among the cores in the multicore chip configuration. Therefore, the bandwidth required by the whole chip configuration is *num_cores* times the bandwidth required by one core.

$$Required\_BW\_multicore_{num\_cores, \beta, BW} =$$

$$= num\_cores \times \frac{Requests\_MM\_core_{\beta, BW}}{T\_core_{\beta, BW}} \qquad (2)$$

In the case of hybrid memory CMPs the CMP performance estimator calculates their execution times using equation 3. The equation is similar to the one used for cache-based CMPs, adding an overhead of $DMA\_time_{\beta, BW}$ times

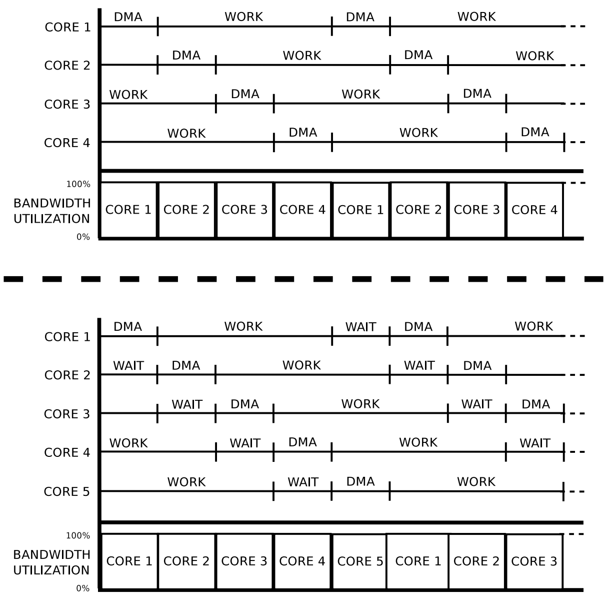


Figure 3: Bus utilization in hybrid memory CMPs.

$num\_cores\_overhead_{num\_cores, \beta, BW}$. This overhead is added in case the execution of a loop in the chip configuration has higher bandwidth requirements than the bandwidth provided by the bus, as illustrated in Figure 3. In this figure the typical execution model induced by local memories is assumed [10]. Under this model the execution is organized in two parts: control and work. During the control part data is transferred to/from the local memory and main memory, and the execution is blocked until all DMA transfers are completed. Then, some work is performed. This behaviour is repeated until all assigned units of work are computed. In this execution model the control part is dominated by the time it takes to move the data from/to the local memory using DMAs, which is *DMA_time*. After this part, the work part starts, executing the body of the loop, which lasts *work_time*. The sum of these two times is *total_time*. Notice that a core is using 100% of the provided bandwidth during *DMA_time* and then, during *work_time*, the bandwidth it requires is negligible. Because of this behaviour, it is possible to interleave the usage of the provided bandwidth between the cores. If the number of cores is smaller than *total_time* divided by *DMA_time* no problem arises: a given core has its *DMA_time* of dedicated bus and, by the time it needs the bus again, the other cores have already finished their DMA operations. This situation is illustrated in the upper part of Figure 3. However, as it is shown in the lower part of the figure, if the number of cores is greater than *total_time* divided by *DMA_time*, the second time a given core needs the bus for its DMA operations, some other core is using it. This forces the core to wait for its turn again. In particular, it has to wait *wait_time*, which is *DMA_time*, times the number of cores that have not executed its work part yet. Because of this possible behaviour, the overhead is added to the multicore time.

$$T\_multicore_{num\_cores, \beta, BW} = \frac{T\_core_{\beta, BW}}{num\_cores} +$$

$$+ DMA\_time_{\beta, BW} \times num\_cores\_overhead_{num\_cores, \beta, BW} \qquad (3)$$

5. EVALUATION

This section first shows the behaviour of the different configurations of cache-based CMPs, analyzing their performance with a perfect bus and discussing its main architectural requirements. Hybrid memory CMPs are later studied, showing their performance and how the addition of local memories relax the architectural requirements imposed by cache-based CMPs. The comparison between the two types of CMPs is finally presented and discussed.

In order to conclude with an optimal CMP configuration, the evaluation of the design space exploration has been organized in three steps. Section 5.1 studies the cache-based CMP configurations and its outcome is an optimal configuration in terms of the number of cores, the L3 size and the required bandwidth. Section 5.2 studies the hybrid memory CMP configurations looking for optimal configurations in the same form as for cache-based configurations, but with the aim of determining whether it is possible or not to reduce the bandwidth requirements. Finally, Section 5.3 compares both cache-based and hybrid memory CMP configurations and, for the optimal configurations obtained in the first two sections, it is determined which one exposes better performance and at what bandwidth cost.

5.1 Cache-based CMPs

This section evaluates the potential speedup and bandwidth requirements for the different cache-based CMP configurations with different memory latencies. The outcome of this evaluation is a cache-based CMP configuration with a number of cores and a size of L3 cache connected to a bus that maximizes performance.

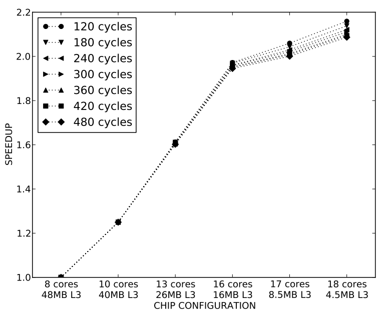
In Figure 4 it can be observed the average speedups of the 28 loops executed on the cache-based CMP configurations with respect to the baseline architecture. The chip configurations are the ones derived from the baseline architecture setting α to 0.25 in Figure 4a, 0.33 in Figure 4b and 0.5 in Figure 4c. In all figures seven lines are printed, each one showing the behaviour of the chip configurations with a given main memory latency.

From Figure 4 it can be studied the scalability of the cache-based CMPs. The three figures show two clear tendencies: for chip configurations with $\beta \geq$ 1MB the speedups present a perfect scalability and, after that point, reducing the L3 cache to add more cores provide under-linear speedups. This situation can be observed in each plot comparing the speedups of the first four chip configurations with the speedups of the last two chip configurations. On average, the L3 cache miss ratios in the first four chip configurations are almost identical, so having a smaller L3 cache has a negligible performance impact. When $\beta <$ 1MB the average L3 cache miss ratio starts to increase, so the speedups do not present a perfect scalability with the addition of extra cores.

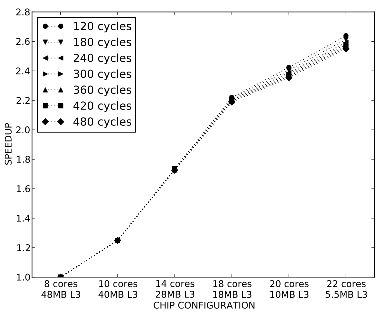
Figure 4 shows also the importance of the main memory latency. The effect of having a fast or a slow main memory is very important for a cache-based CMP, though it may not seem that if one looks at the figure and realizes that the lines of the main memory latencies are very close one from each other, specially in chip configurations with $\beta \geq$ 1MB. What happens is that in these chip configurations the L3 cache miss ratios are very similar to the L3 cache miss ratio of the baseline architecture, so they all are equally affected by the latency of the main memory. This fact makes that, when the speedup is calculated, the effect of the main memory latency disappears due to the division of execution times. When $\beta <$ 1MB the effect of the latency of the main memory can be observed in the figure, since these chip configurations have higher L3 cache miss ratios than the baseline and so their speedups vary depending on the latency. In Figure 4a the chip configuration of 18 cores with 4.5MB of L3 cache reaches a 2.15x speedup with a main memory latency of 120 cycles, decreasing to a 2.08x speedup with a main memory latency of 480 cycles. Similarly, the chip configuration with 22 cores and 5.5MB of L3 cache in Figure 4b has a maximum speedup of 2.63x with a latency of 120 cycles and 2.54x with the slowest main memory, and the 28 core chip configuration with 7MB of L3 cache in Figure 4c goes from 3.35x to 3.22x.

The aim of Figure 5 is to show the average bandwidth requirements of cache-based CMPs. To ensure representative bandwidth requirements several main memory latencies have been taken into account in the study. This figure shows that for chip configurations with $\beta \geq$ 1MB the average bandwidth required grows linearly due to the addition of cores that have the same L3 cache miss ratios. When the L3 cache gets smaller, the miss ratios increase and so the bandwidth requirements do, showing a super-linear growth in the three plots. In any case, the figure shows that all chip configurations require average bandwidths that range between acceptable limits, with a maximum average required bandwidth of 51.2GB/s. Although the range of average bandwidths is acceptable it has been detected that not all the loops have the same bandwidth requirements: some of them have very good hit ratios in all cache levels so they almost require no bandwidth, while some others have huge data sets and irregular accesses, so they need a much higher bandwidth. For instance, the loop of the function *compute_indexmap* in the FT benchmark computes a sum and a product of three values and stores the result to memory using a non-regular access pattern. A lot of stores are issued at a fast pace and the stores almost always miss in the L3 cache so the bandwidth requirements are huge, more than 115GB/s in any chip configuration. Similar behaviours can be observed in other loops. On the other hand, some loops have minimal bandwidth requirements, because of the regular access patterns that do not provoke cache misses (many loops of the CG and FT) or because the data set fits in the L3 cache (any loop of the MG). This observation highlights the need to quantify how many loops can be executed without any performance penalty given a fixed and realistic bandwidth for the different chip configurations.

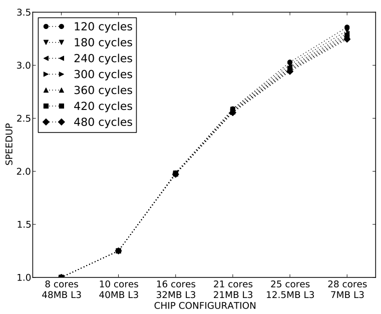
Following that direction, Figure 6 shows what is the percentage of loop executions that can be handled without any performance penalty when the provided bandwidth is limited. For each chip configuration, a stacked bar with aggregated percentages is plotted. Each bar has four stacks which show, from lower and darker to higher and lighter, the percentage of executions that do not have bandwidth requirements of more than 20GB/s, 35GB/s, 50GB/s and 65GB/s, respectively. The percentage is calculated as follows: for each chip configuration, the 28 loops have been executed with the seven main memory latencies. That gives 196 executions for each chip configuration. It has been calculated the bandwidth requirements of these 196 executions and checked if they are lesser than thresholds of 20GB/s, 35GB/s, 50GB/s and 65GB/s, obtaining as a result the four percentages shown in each stacked bar of the figure. Notice that one of the FT loops, the one previously mentioned, has



(a) $\alpha = 0.25$

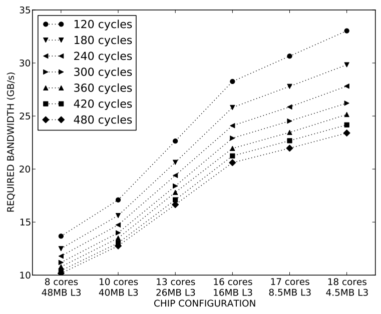


(b) $\alpha = 0.33$

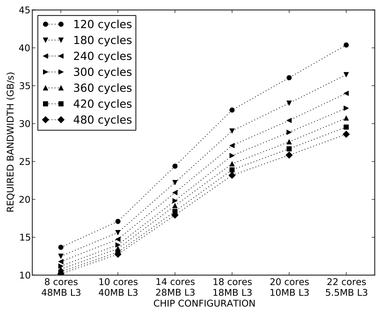


(c) $\alpha = 0.5$

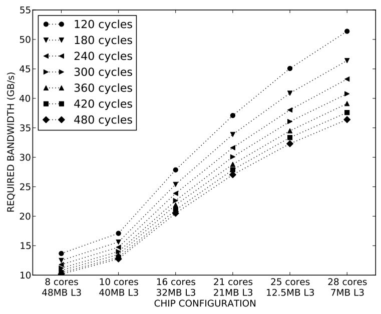
Figure 4: Speedup of cache-based CMP configurations.



(a) $\alpha = 0.25$



(b) $\alpha = 0.33$



(c) $\alpha = 0.5$

Figure 5: Bandwidth requirements of cache-based CMP configurations.

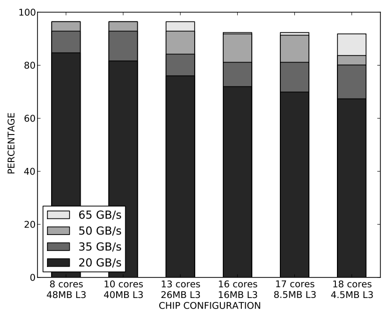
such a big required bandwidth that no chip configuration can handle it with any of the provided bandwidths, so the bars never reach the 100%. For chip configurations with up to 10 cores, a 20GB/s bandwidth is capable of handling between 84.6% and 81.5% of the loops. This percentage drops when more cores are added to the chip, to the point of only being able to handle the execution of 56.5% of the loops in the chip configuration with 28 cores and 7MB of L3 cache, in Figure 6c. Having a bus that can deliver 35GB/s increases this percentages in 9 to 15 per cent quite uniformly across all chip configurations. Increasing the bandwidth to 50GB/s slightly enlarges the set of covered loop executions for chip configurations with 8 and 10 cores, with an increase of 4%, it has an important impact in chip configurations with 13 to 17 cores, with increases around 11%, and it has a minor impact in chip configurations with 18 cores or more, with a maximum increase of 6.5%. Finally, a 65GB/s bus has almost no benefits over a 50GB/s bus on chip configurations with less than 17 cores but, in the range of 18 cores to 22 cores, it succeeds in executing an important number of loops, increasing the percentage to a 91.4%. For the most extreme chip configurations, the ones in the Figure 6c with 25 and 28 cores, this percentage is reduced to 83.5% and 81.5%, respectively.

Figure 7 shows the average speedup of the different chip configurations against the baseline architecture when the bandwidth is limited. Since it is very difficult to make an analytical model of the performance penalty due to insufficient provided bandwidth, the results of the loops that cannot be executed with the fixed bandwidth have been discarded. Notice that the results of the figure don't differ from the ones without bandwidth constraints. This means that the non-fitting executions that have been discarded don't change the general trends and that the remaining experiments are representative in order to take conclusions on the overall behaviour.

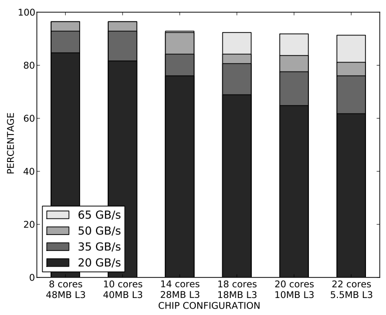
Table 3: Best cache-based CMP configurations.

α	0.25	0.33	0.5
CHIP CONFIGURATION	18 cores 4.5MB L3	22 cores 5.5MB L3	28 cores 7MB L3
BUS BANDWIDTH	65GB/s	65GB/s	65GB/s
SPEEDUP	2.13x	2.58x	3.31x
LOOP EXECUTIONS	91.4%	90.8%	81.5%

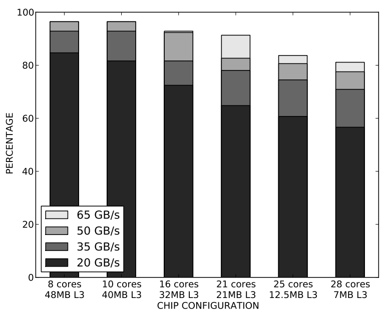
Table 3 shows the best cache-based CMP configurations for the three values of α. These chip configurations are the ones with higher speedups that can execute more than 80% of the loops without penalizing their performance. The three resulting chip configurations are 18 cores with 4.5MB of L3 cache for $\alpha = 0.25$, 22 cores with 5.5MB of L3 cache for $\alpha = 0.33$ and 28 cores with 7MB of L3 cache for $\alpha = 0.5$, all of them with a bus of 65GB/s. The speedups obtained with these three configurations are, respectively, 2.13x, 2.58x and 3.31x. Notice that the three best chip configurations are obtained with β = 256KB. Also notice that, in general,



(a) $\alpha = 0.25$

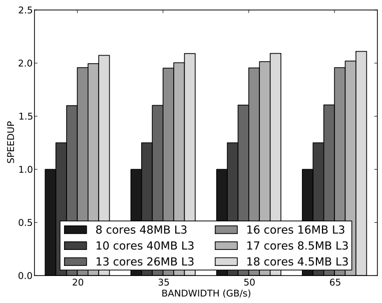


(b) $\alpha = 0.33$

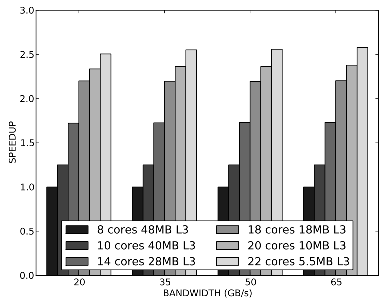


(c) $\alpha = 0.5$

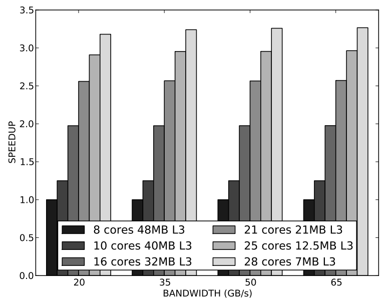
Figure 6: Percentage of executable loops in cache-based CMP configurations with bandwidth constraints.



(a) $\alpha = 0.25$



(b) $\alpha = 0.33$



(c) $\alpha = 0.5$

Figure 7: Speedup of cache-based CMP configurations with bandwidth constraints.

aggressive cache-based CMP designs are very sensible to how the chip is connected to the memory subsystem, since their performance varies significantly when changing the latency of the main memory and they require a powerful bus of 65GB/s to be able to satisfy the bandwidth requirements in many of the loops.

5.2 Hybrid memory CMPs

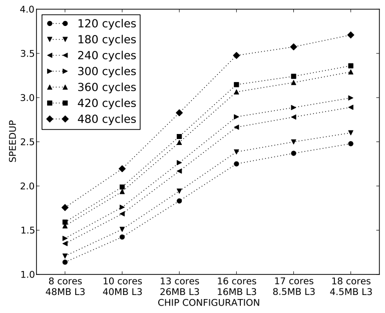
In the previous section it has been concluded that aggressive configurations of cache-based CMPs are capable of reaching notable speedups at the cost of imposing some architectural requirements such as a fast main memory and a wide bus. In this section hybrid memory CMPs are evaluated from the point of view of performance and bandwidth requirements. It is also shown how local memories alleviate the impact of having modest off-chip components. In this section it has been followed the same steps as in the previous section, computing ideal speedups and their corresponding bandwidth requirements to later fix a provided bandwidth and, from there, chose the best chip configuration for a hybrid memory CMP.

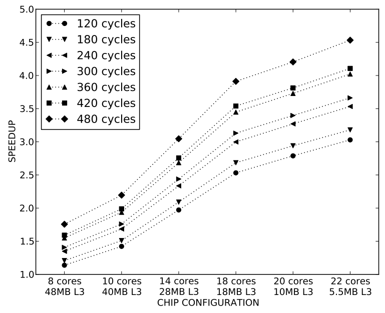
The average speedups obtained with all the hybrid memory CMP configurations against the baseline architecture can be observed in Figure 8. The figure shows the behaviour of the different chip configurations in three plots, one per each value of α, and assuming infinite bandwidth, which means ignoring the overhead explained in Figure 3 and calculated in Equation 3.

The first important observation from Figure 8 is that in the direct comparison between an 8 core cache-based CMP and an 8 core hybrid memory CMP the latter clearly outperforms the former, with speedups that range between 1.11x and 1.75x, depending on the latency of the main memory. This proofs that the introduction of local memories has a good performance, achieving significant speedups even in chip configurations with a big L3 cache.

Figure 8 shows also the scalability of the hybrid memory CMPs. Chip configurations with $\beta \geq$ 1MB show a perfect scalability, while the scalability drops in chip configurations with $\beta <$ 1MB. The same behaviour can be observed in a cache-based CMP, as explained in the previous subsection. In any case, the results show that any chip configuration of a hybrid memory CMP outperforms the baseline architecture, with performance gains that go from 1.11x with 8 cores and a slow main memory to 5.75x with 28 cores and a fast main memory.

The importance of the latency of the main memory can be also withdrawn from Figure 8, taking a look at the separation between the lines. It can be observed that the slower the main memory the higher speedups are achieved in all chip configurations. For instance, in the 8 core chip configurations the speedup goes from 1.11x with 480 cycles to 1.75x with 120 cycles, a 0.64x difference. This difference grows with the addition of cores, reaching a 1.96x difference (from 5.75x to 3.79x) in the 28 core with 7MB of L3 cache chip configuration, in Figure 8c. The reason is that


(a) $\alpha = 0.25$


(b) $\alpha = 0.33$

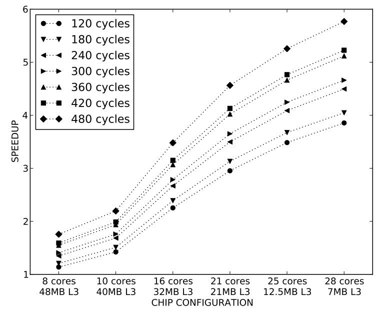
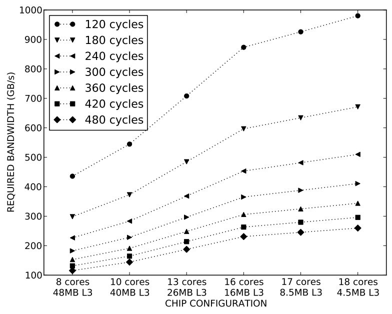
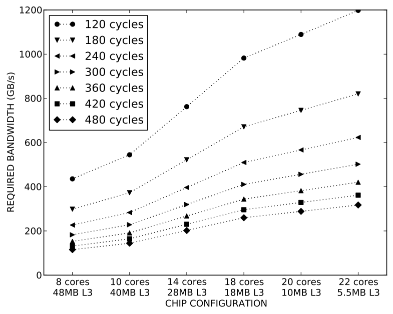

(c) $\alpha = 0.5$

Figure 8: Speedup of hybrid memory CMP configurations.


(a) $\alpha = 0.25$


(b) $\alpha = 0.33$

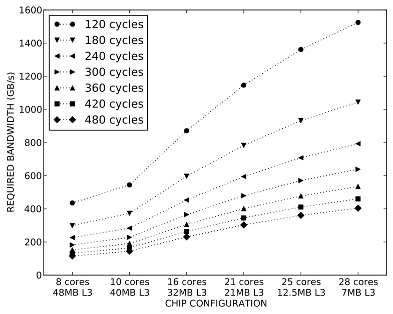
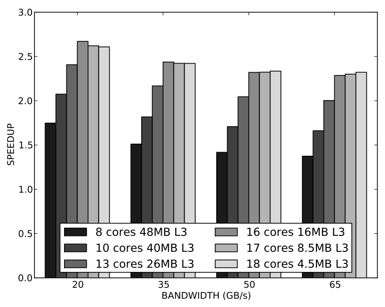

(c) $\alpha = 0.5$

Figure 9: Bandwidth requirements of hybrid memory CMP configurations.

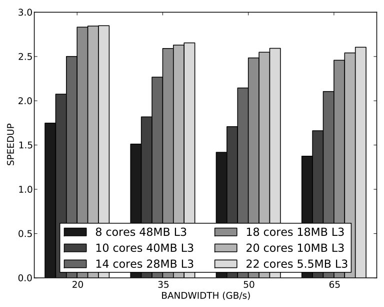
the hybrid memory cores use the bandwidth only during the control part, which is a small period of time compared to the work part. Obviously, the DMA transfers that get data from main memory benefit from having a fast main memory, and this is reflected in a shorter control part, but the work part is not affected by the main memory latency. In the case of a cache-based CMP, like the baseline, the whole execution time benefits from having a faster main memory, making the whole execution time lower than the one obtained with a slow main memory. This variance in the execution time of cache-based CMPs is what causes the differences in the speedup of hybrid memory CMP configurations against the baseline when the latency of the main memory varies. Consequently, the importance of the main memory latency is minimal in hybrid memory CMPs.

Figure 9 shows the average bandwidth required for the execution of the 28 loops for each chip configuration. This figure, like the previous one, ignores the interleaving in the usage of the bus explained in Figure 3. It is clear that any chip configuration with any main memory latency requires an exaggerated amount of bandwidth, from 111GB/s in the most modest chip configurations in terms of number of cores and main memory latency to more than 1500GB/s with 28 cores and a latency of 120 cycles, in Figure 9c. These numbers show what should be the provided bandwidth in order to maintain the performance in all the cores, allowing all of them to execute their DMA phases at the same time and then leaving the bus unutilized during their work parts. This implies that the bandwidth required by a single core during its DMA phase, which is already high due to the nature of the DMA engine, has to be multiplied by the number of cores in the chip configuration in order to allow them all to move the data concurrently without performance penalties, resulting in an unfeasible required bandwidth. These unacceptable required bandwidths motivate the use of some mechanism that guarantees that two control parts will not be executed in parallel, that is why this paper proposes to interleave the usage of the bus.

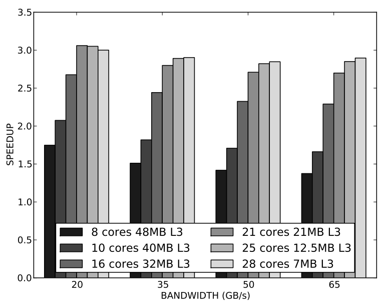
Figure 10 shows the actual speedups obtained when bandwidth restrictions are applied. Having a limited bandwidth means the provided bandwidth is fixed by the bus and its usage is interleaved between the cores, according to Figure 3. The figure shows that fixing the bandwidth limits the scalability: for any bandwidth, the chip configurations with $\beta \geq$ 1MB present increasing but not linear speedups, meaning that the increment of bandwidth requirements due to the addition of cores can be handled by the bus only for some loops, but for some others it cannot, so performance penalties appear as described in Equation 3. For chip configurations that have $\beta <$ 1MB the required bandwidth cannot be handled by the bus in most of the executions and the performance drops due to the wait times imposed by the interleaving mechanism. The magnitude of the drop varies depending on the bandwidth provided by the bus: in Figure 10a, with a 20GB/s bus the 18 core chip configuration performs worst than the 16 core chip configuration, while with a 65GB/s the



(a) $\alpha = 0.25$



(b) $\alpha = 0.33$



(c) $\alpha = 0.5$

Figure 10: Speedup of hybrid memory CMP configurations with bandwidth constraints.

18 core chip configuration performs slightly better than the 16 core chip configuration. The reason behind this is that the more modest the bus bandwidth, the larger the waiting times, having a bigger penalty in performance.

Figure 10 also shows that the speedup obtained with any given chip configuration decreases as the bus bandwidth is increased. Take, for instance, the chip configuration with 14 cores and 28MB of L3 cache in Figure 10b. It can be seen that the speedup with a bus bandwidth of 20GB/s is 2.49x, 2.25x with 35GB/s, 2.14x with 50GB/s and 2.09x with 65GB/s. The effect of the waiting time in this chip configuration is negligible because the number of cores is relatively small. What happens is that the baseline architecture takes a bigger profit of the increasing bus bandwidth than the hybrid memory CMP configuration, so the speedups against the baseline architecture decrease when the provided bandwidth gets higher.

Table 4: Best hybrid memory CMP configurations.

α	0.25	0.33	0.5
CHIP CONFIGURATION	16 cores 16MB L3	22 cores 5.5MB L3	21 cores 21MB L3
BUS BANDWIDTH	20GB/s	20GB/s	20GB/s
SPEEDUP	2.62x	2.83x	3.06x

The summary of the best chip configurations for a hybrid memory CMP is shown in Table 4. The best chip configurations for $\alpha = 0.25$, $\alpha = 0.33$ and $\alpha = 0.5$ are, respectively, 16 cores with 16MB of L3 cache, 22 cores with 5.5MB of L3 cache and 21 cores with 21MB of L3 cache, all of them with a bus of 20GB/s. The speedups reported by these chip configurations are 2.62x, 2.83x and 3.06x, respectively. In this case of hybrid memory CMPs, for $\alpha = 0.25$ and $\alpha = 0.5$ the best chip configurations are obtained with $\beta = 1$MB while, when $\alpha = 0.33$, having $\beta = 256$KB slightly improves βs of 512KB and 1MB. In general, it can be concluded that hybrid memory CMPs achieve a very good performance and, unlike cache-based CMPs, they are able to keep the performance level even with a modest memory subsystem.

5.3 Comparison between cache-based CMPs and hybrid memory CMPs

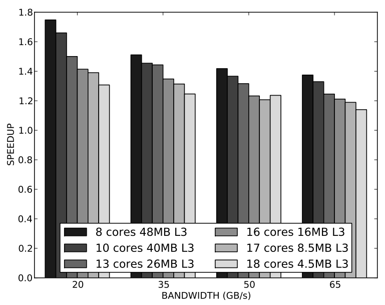
This section compares the performance of hybrid memory CMPs and cache-based CMPs in all their chip configurations, taking into account a fixed provided bandwidth.

The average speedup of the hybrid memory CMP configurations over the cache-based CMP configurations is shown in Figure 11. The loop executions on the cache-based chip configurations that require more bandwidth than what is provided by the bus are discarded. When this happens, the corresponding loop executions on the same chip configuration of a hybrid memory CMP are also discarded, for fairness.

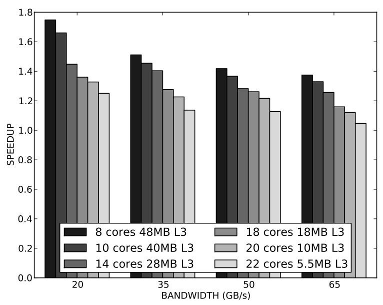
Figure 11 shows that almost all the chip configurations achieve speedups no matter the chip configuration nor the bandwidth provided by the bus. The only exception is the chip configuration with 28 cores and 7MB of L3 cache, in Figure 11c, when the bus has a bandwidth of 50GB/s or more. In this case, the cache-based CMP configuration has a 1% speedup over the same chip configuration in a hybrid memory CMP. It can be observed that the speedups range from 1x (no improvement) to 1.74x, and the factors that determine the speedup are the number of cores and the provided bandwidth.

In any plot of Figure 11, given a particular bandwidth constraint, the speedup factor decreases as the number of cores in the chip configurations increases. The most affected set of chip configurations are the ones with 20GB/s in Figure 11c, where speedups go from 1.74x with 8 cores to 1.08x with 28 cores. The reason is that the scalability of cache-based CMPs is perfect up to 16 cores, while in hybrid memory CMPs the interleaving of the usage of the bus limits the scalability. With more than 16 cores hybrid memory CMPs suffer this problem heavily, much more than what cache-based CMPs suffer from the increase in the L3 cache miss ratio. This unbalance in the performance penalties is what provokes that the speedup decreases as cores are added to the chip.

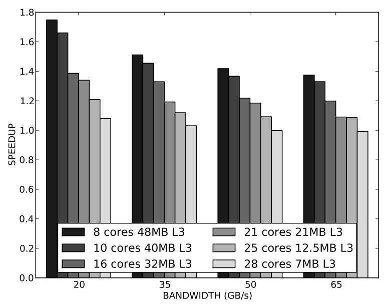
The impact of the provided bandwidth can be clearly seen if one looks at the speedups of a single chip configuration in a single plot. In any chip configuration, its speedup with a bus of 20GB/s is greater than its speedup with a bus of 35GB/s, which at the same is greater than the speedup with 50GB/s, and so on. In the case of the chip configuration with 13 cores and 26MB of L3 cache, for instance, the speedups are 1.49x, 1.42x, 1.3x and 1.22x. This is because the cache-based CMPs take more profit of a higher bus bandwidth, as explained in Section 5.2. This greater benefit is the responsible for the speedup decrement when the provided bandwidth is augmented.



(a) $\alpha = 0.25$



(b) $\alpha = 0.33$



(c) $\alpha = 0.5$

Figure 11: Speedup of hybrid memory CMP configurations against cache-based CMP configurations.

It is also interesting to compare the best chip configurations of a kind of CMP with the corresponding chip configurations of the other kind of CMP. The best chip configurations have been selected in Sections 5.1 and 5.2 and summarized in Tables 3 and 4. The best chip configuration of cache-based CMPs with $\alpha = 0.25$, which is composed of 18 cores, a 4.5MB L3 cache and a 65GB/s bus, is outperformed by the symmetric chip configuration of hybrid memory CMPs by 1.14x. The other two best chip configurations of cache-based CMPs perform very similar to their corresponding chip configurations of hybrid memory CMPs: the hybrid memory CMP is 1.03x faster in the chip configuration of 22 cores and 5.5MB of L3 cache with a bus of 65GB/s, while with the chip configuration of 28 cores, 7MB of L3 cache and a bus of 65GB/s the cache-based CMP is 1.01x faster. Contrariwise, the three best chip configurations of hybrid memory CMPs achieve moderate speedups against the corresponding chip configurations of cache-based CMPs. The speedup of the chip configuration with 16 cores, 16MB of L3 cache and a 20GB/s bus is 1.38x, the speedup of the chip configuration with 22 cores, 5.5MB of L3 cache and a 20GB/s bus is 1.23x and the speedup of the chip configuration with 21 cores, 21MB of L3 cache and a 20GB/s bus is 1.34x.

In conclusion, the hybrid memory CMPs perform better than cache-based CMPs with almost all the configurations, with speedups of up to 1.74x. In two cases, cache-based CMPs outperform hybrid memory CMPs by 1.01x. The best chip configuration in the direct comparison is the one with 8 cores, 48MB of L3 cache and a bus bandwidth of 20GB/s. Adding a lot of cores penalizes the hybrid memory CMPs because of the overhead introduced by the bus usage interleaving and having a bus with a higher bandwidth benefits the cache-based CMPs much more than the hybrid memory CMPs.

6. CONCLUSIONS

This paper explores the design space of CMPs with caches and local memories in order to find an efficient chip configuration for HPC applications. Specifically, two types of CMPs are compared: CMPs composed of cores with a traditional cache hierarchy, or cache-based CMPs, and CMPs composed of cores with a local memory side to the cache hierarchy, or hybrid memory CMPs. The design space exploration contemplates different design parameters such as the number of cores, the size of the shared L3 cache and bus bandwidth. The presented results show the performance of the different chip configurations for 28 computational loops of the NAS benchmark suite. It is presented a discussion of what is the impact of varying each design parameter on the two kinds of CMPs separately, to finally make a direct comparison between them.

Results show that cache-based CMPs are very efficient when the number of cores and the bus bandwidth are very high. Configurations of 18 cores with 4.5MB of L3 cache, 22 cores with 5.5MB of L3 cace and 28 cores with 7MB of L3 cache achieve, respectively, speedups of 2.13x, 2.58x and 3.31x with a bus of 65GB/s when compared to a cache-based CMP with 8 cores and 48MB of L3 cache. The best chip configurations of hybrid memory CMPs, which are 16 cores with 16MB of L3 cache, 22 cores with 5.5MB of L3 cache and 21 cores with 21MB of L3 cache, achieve 2.62x, 2.83x and 3.06x speedups with a 20GB/s bus against the same baseline. It has been also seen that hybrid memory CMPs outperform their symmetric cache-based CMP configurations in the vast majority of the cases, with a maximum speedup of 1.74x. The variation of design parameters affects the two kind of CMPs differently. Adding cores is very beneficial for cache-based CMPs but limits the performance of hybrid memory CMPs. On the other hand, cache-based CMPs need a very high provided bandwidth while hybrid memory cores are much less sensible to this parameter.

7. REFERENCES

[1] AMD 2006 Technology Analyst Day: Official Introduction of K10 Microarchitecture.

[2] http://www.hypertransport.org.

[3] Intel Core i7 Processor Extreme Datasheet. November 1 2008.

[4] The SCC Platform Overview. Revision 0.7. May 24 2010.

[5] Top 500 Supercomputer Sites List. November 2010. http://www.top500.org/lists/2010/11.

[6] D. H. Bailey, E. Barszcz, J. T. Barton, D. S. Browning, R. L. Carter, L. Dagum, R. A. Fatoohi, P. O. Frederickson, T. A. Lasinski, R. S. Schreiber, H. D. Simon, V. Venkatakrishnan, and S. K. Weeratunga. The NAS Parallel Benchmarks. In *SC '91: Proceedings of the 1991 Conference on Supercomputing*, pages 158–165, Albuquerque, New

Mexico, USA, November 18-22 1991. IEEE Computer Society.

[7] R. Banakar, S. Steinke, B.-S. Lee, M. Balakrishnan, and P. Marwedel. Scratchpad Memory: A Design Alternative for Cache On-chip memory in Embedded systems. In *CODES '02: Proceedings of the 10th International Symposium on Hardware/Software Codesign*, pages 73–78, Estes Park, Colorado, USA, May 6-8 2002. ACM.

[8] R. Bertran, M. Gonzàlez, X. Martorell, N. Navarro, and E. Ayguadé. Local Memory Design Space Exploration for High-Performance Computing. *The Computer Journal*, March 2010.

[9] M. Ekman and P. Stenstrom. Performance and Power Impact of Issue-width in Chip-Multiprocessor Cores. In *ICPP '03: Proceedings of the 32nd International Conference on Parallel Processing*, pages 359–368, Kaohsiung, Taiwan, October 6-9 2003. IEEE Computer Society.

[10] M. Gonzàlez, N. Vujic, X. Martorell, E. Ayguadé, A. E. Eichenberger, T. Chen, Z. Sura, T. Zhang, K. O'Brien, and K. O'Brien. Hybrid Access-Specific Software Cache Techniques for the Cell BE Architecture. In *PACT '08: Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*, pages 292–302, Toronto, Ontario, Canada, October 25-29 2008. ACM.

[11] M. D. Hill and M. R. Marty. Amdahl's Law in the Multicore Era. *IEEE Computer*, 41(7):33–38, July 2008.

[12] J. Huh, D. Burger, and S. W. Keckler. Exploring the Design Space of Future CMPs. In *PACT '01: Proceedings of the 10th International Conference on Parallel Architectures and Compilation Techniques*, pages 199–210, Barcelona, Spain, September 8-12 2001. ACM.

[13] J. Kahle. The Cell Processor Architecture. In *MICRO 38: Proceedings of the 38th International Symposium on Microarchitecture*, pages 3–4, Barcelona, Spain, November 12-16 2005. IEEE Computer Society.

[14] M. Kandemir, O. Ozturk, and M. Karakoy. Dynamic On-Chip Memory Management for Chip Multiprocessors. In *CASES '04: Proceedings of the 2004 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*, pages 14–23, Washington, DC, USA, September 22-25 2004. ACM.

[15] R. Kumar, D. M. Tullsen, and N. P. Jouppi. Core architecture optimization for heterogeneous chip multiprocessors. In *PACT '06: Proceedings of the 15th International Conference on Parallel Architectures and Compilation Techniques*, pages 23–32, Seattle, Washington, USA, September 16-20 2006. ACM.

[16] R. Kumar, V. V. Zyuban, and D. M. Tullsen. Interconnections in Multi-Core Architectures: Understanding Mechanisms, Overheads and Scaling. In *ISCA '05: Proceedings of the 32nd International Symposium on Computer Architecture*, pages 408–419, Madison, Wisconsin, USA, June 4-8 2005. IEEE Computer Society.

[17] J. Leverich, H. Arakida, A. Solomatnikov, A. Firoozshahian, M. Horowitz, and C. Kozyrakis. Comparing Memory Systems for Chip Multiprocessors. *SIGARCH Computer Architecture News*, 35(2):358–368, May 2007.

[18] J. Li and J. F. Martínez. Power-Performance Implications of Thread-level Parallelism on Chip Multiprocessors. In *ISPASS '05: Proceedings of the 5th International Symposium on Performance Analysis of Systems and Software*, pages 124–134, Austin, Texas, USA, March 20-22 2005. IEEE Computer Society.

[19] J. Li and J. F. Martínez. Dynamic Power-Performance Adaptation of Parallel Computation on Chip Multiprocessors. In *HPCA '06: Proceedings of the 12th International Symposium on High-Performance Computer Architecture*, pages 77–87, Austin, Texas, USA, February 11-15 2006. IEEE Computer Society.

[20] Y. Li, B. Lee, D. Brooks, Z. Hu, and K. Skadron. CMP Design Space Exploration Subject to Physical Constraints. In *HPCA '06: Proceedings of the 12th International Symposium on High-Performance Computer Architecture*, pages 17–28, Austin, Texas, USA, February 11-15 2006. IEEE Computer Society.

[21] M. Monchiero, R. Canal, and A. González. Power/Performance/Thermal Design-Space Exploration for Multicore Architectures. *IEEE Transaction on Parallel and Distributed Systems*, 19(5):666–681, May 2008.

[22] R. Murphy. On the Effects of Memory Latency and Bandwidth on Supercomputer Application Performance. In *IISWC '07: Proceedings of the 10th International Symposium on Workload Characterization*, pages 35–43, Boston, Massachusetts, USA, September 27-29 2007. IEEE Computer Society.

[23] J. L. Shin, K. Tam, D. Huang, B. Petrick, H. Pham, C. Hwang, H. Li, A. Smith, T. Johnson, F. Schumacher, D. Greenhill, A. S. Leon, and A. Strong. A 40nm 16-core 128-thread CMT SPARC SoC processor. In *ISSCC '10: Proceedings of the 2010 International Symposium on Solid-State Circuits Conference*, pages 98–99, San Francisco, California, USA, February 7-11 2010. IEEE Computer Society.

[24] M. Ware, K. Rajamani, M. Floyd, B. Brock, J. C. Rubio, F. Rawson, and J. B. Carter. Architecting for Power Management: The IBM® POWER7^{TM} Approach. In *HPCA '10: Proceedings of the 16th International Symposium on High-Performance Computer Architecture*, pages 1–11, Bangalore, India, January 9-14 2010. IEEE Computer Society.

[25] M. T. Yourst. PTLsim: A Cycle Accurate Full System x86-64 Microarchitectural Simulator. In *ISPASS '07: Proceedings of the 7th International Symposium on Performance Analysis of Systems and Software*, pages 23–34, San Jose, California, USA, April 25-27 2007. IEEE Computer Society.