Appendix A
Project management

This project was developed from March to October 2010. The results of the project are collected in 3 documents:

- Report for PARSALab (EPFL)
- Report for CPS-Unizar
- Presentation for CPS-Unizar

In this Appendix the initial plan and effort estimation are presented, as well as the effective schedule and effort, and the reasons for the differences between them.

A.1 Initial schedule and effort estimation

The project was initially thought to be developed between March and June 2010. Figure A.1 shows the initial schedule (created in March 2010 and updated in April 2010). The project was divided in 4 main tasks (table A.1 presents those main tasks and the effort estimated for them) which have to do with the initial proposal and division of the project. Figure A.2 shows the detailed schedule (calendar) for those tasks.

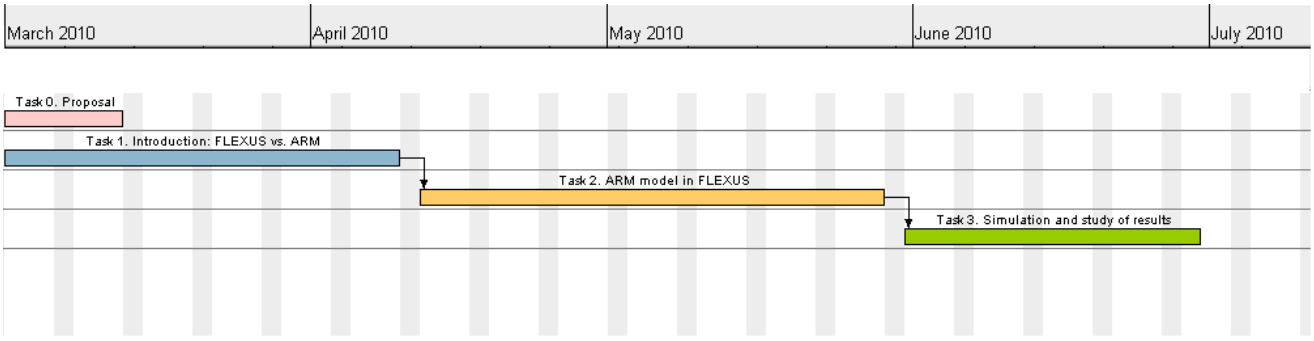
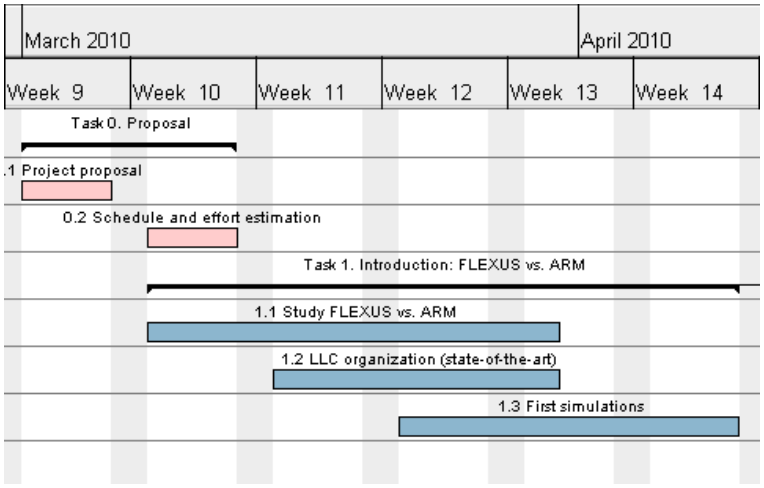


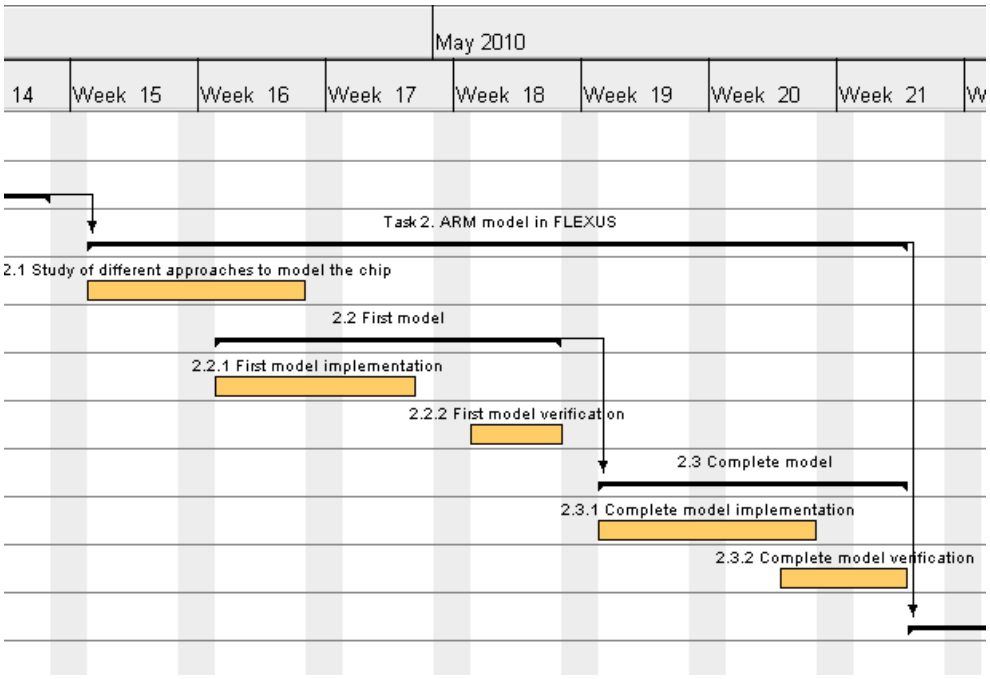
Figure A.1: Initial schedule (calendar and tasks).

TASK	SCHEDULE/DESCRIPTION	ESTIMATED HOURS
Task 0. Proposal	**01/03/2010 - 15/03/2010**	**25 hours**
0.1 Project proposal	Writing the official PFC proposal for the University of Zaragoza.	10 h.
0.2 Schedule and effort estimation	Initial plan for the project including schedule and effort estimation.	15 h.
Task 1. Intro: *FLEXUS* vs. ARM	**01/03/2010 - 11/04/2010**	**80 - 100 hours**
1.1 Study *FLEXUS* vs. ARM	Extended study of the simulation environment (*FLEXUS*) and the differences between the architecture that the simulator models (SPARC) and ARM.	40 h.
1.2 LLC organization (state-of-the-art)	Extended research on LLC organization papers.	40 h.
1.3 First simulations	First simulations in *FLEXUS* and the real hardware.	20 h.
Task 2. ARM model in *FLEXUS*	**12/04/2010 - 31/05/2010**	**100 - 150 hours**
2.1 Study of the different approaches to simulate the chip	Study of the different possibilities to model the chip in the simulator.	30 h.
2.2 First approach	First approach to model the chip. Verification of the model.	50 h.
2.3 Complete model	Complete model of the chip. Verification of the model.	60 h.
Task 3. Simulation and study of results	**01/06/2010 - 31/06/2010**	**200 - 250 hours**
3.1 Simulation	Simulation of commercial workloads.	75 h.
3.2 Study of results	Study and verification of the results obtained.	75 h.
3.3. Report	Report for EPFL and CPS-Unizar.	100 h.
	Total estimated hours (aprox.)	**500 hours**

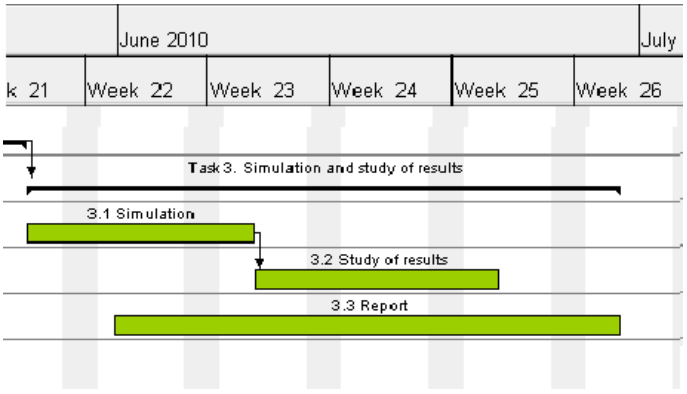
Table A.1: Description of the main tasks and estimated effort (updated in April 2010).



(a) Schedule for Tasks 0 and 1.



(b) Schedule for Task 2.



(c) Schedule for Task 3.

Figure A.2: Detailed schedule for the different tasks.

A.2 Project Development

A.2.1 Schedule

Different issues (see section A.2.3) made our schedule shift in June, which led us to re-schedule a part of the project to September and October. Besides the re-schedule, we also realized that more effort would be needed in the final report, as that report would be delivered in two languages.

Figure A.3 shows the schedule for September and October. The new task created (Task 4) is composed of basically the same sub-tasks than Task 3. This does not mean that Task 3 was not partially completed: the sub-tasks 3.1 and 3.2 (see table A.1) were developed. The study of results showed that in other to improve the reliability of our conclusions, some adjustment should be made in the parameters of the simulator. This, and a simulator bug encountered in July, forced us to add a new round of simulations and study of results.

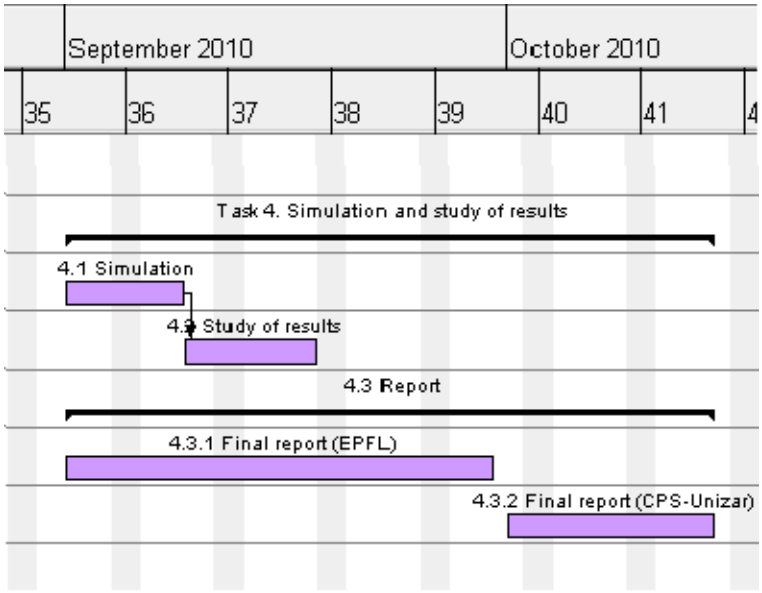


Figure A.3: Schedule for September and October (calendar and tasks).

A.2.2 Effort

Table A.2 collects the amount of hours invested in the project (October 2010). It also includes the comparison with the hours estimated in March 2010 (and September 2010 in the case of Task 4).

The main differences between the estimation and the real effort can be found in Tasks 2 and 3. In the case of Task 2 (ARM model in *FLEXUS*) we overestimated the amount of hours needed, predicting an effort between 100 and 150 hours. In practice, we just explored two models of the three organizations proposed as we concluded theoretically that a third model would not improve our results (see Chapters 3 about the alternatives proposed and 5 the main results). On the other hand, not taking into account the type of project and the fact that the report had to be delivered in two languages made us underestimate the effort needed for the final report.

TASK	ESTIMATED HOURS	REAL HOURS
Task 0. Proposal	25 hours	30 hours
Task 1. Intro: *FLEXUS* vs. ARM	80 - 100 hours	80 hours
Task 2. ARM model in *FLEXUS*	100 - 150 hours	70 hours
Tasks 3.1, 3.2 Simulation and study of results	100 - 150 hours	120 hours
Task 3.3 Report	100 hours	150 hours
Task 4. Simulation and study of results	50 - 100 hours	80 hours
Total hours	**550 hours**	**525 hours**

Table A.2: Comparison of the real and estimated effort. Task 4 only takes into account the effort needed for the new simulation and study of results (not the effort estimated for the report).

A.2.3 Problems ocurred

During the development of the project we faced several problems. Some of them made us change partially our approach:

- **Simulator**: we used *FLEXUS* as simulation environment [33, 35]. Although a new version of the simulator has recently released, the simulator is continuously under development. Because of that reason, our work stuck several times due to new bugs found (which were fixed). The fact that the simulator and the methodology were new for us also influenced the time needed for understanding and fixing the different errors occurred.
- **Workloads**: Due to the lack of appropriate 2-core benchmarks, we could not simulate the exact chip design. One possible solution was creating these benchmarks from scratch. However, we estimated that the effort needed to build new benchmarks would exceed the effort estimated for a project of these characteristics. Instead of that, we decided to contrast the original system with a 16 cores system and extrapolate the results.
- **Methodology**: the methodology we used for simulating in this project was based on statistical sampling. Although the main advantage of using this technique is the speedup in simulation speed, the main disadvantage is that the error introduced in the results may be significant. After the first simulations we discovered our results had an error between 5 and 15 percent (and in some cases above 20%). Our objective was reducing the error to less than 5%, so we invested some time on studying and improving the results.

Finally, we knew that when running commercial applications the cache size plays a fundamental role in the performance of the system [16]. For that reason we added as a variable of our study the cache size, studying not only different organizations, but also the impact of the last level cache size on those organizations.

A.3 Personal evaluation

In general I had a positive experience with this project. The main difficulties I found had to do with the language (the whole project was developed in English and in a foreign country) and the tools we used (both simulation platform and methodology were completely new). I sometimes stuck in different points and it was hard to continue.

I learned that when making research it is important to maintain a global view of what we want, and if we come to a standstill, we have to search for alternatives. I also learned that

everything has to be justified, starting with formulating an hypothesis, and then verifying it.

Maintaining this global view and being open to all kind of ideas and suggestions were two main points that helped me to go on with all this project.

Finally, I want to point out that I found the knowledge acquired during all these years of studies really helpful during all these months of work.

Appendix B
Statistical Sampling Simulation Methodology

This appendix summarizes the SMARTS (Sampling Microarchitectural Simulation) methodology. It is based on [35] and [33]. We present first the main ideas of statistical sampling as described in [35] and then how we applied them to this project. Applying rigorous statistical sampling the simulation time is reduced by a factor of 10000 with an error lower than 5%.

B.1 Introduction

Current software-based microarchitecture simulators are many orders of magnitude slower that the hardware they simulate. Hence, most microarchitecture design studies draw their conclusions from drastically truncated benchmark simulations that are often inaccurate and misleading.

The Computer Architecture Laboratory at Carnegie Mellon University (CALCM) presented the Sampling Microarchitecture Simulation (SMARTS) framework [35] as an approach to enable fast and accurate performance measurements of full-length benchmarks. SMARTS accelerates simulation by selectively measuring in detail only an appropriate benchmark subset.

By measuring only chosen sections (called sampling units) from a benchmark's full execution stream we can reduce simulation time by roughly a factor of 10000 [33]. Sampling provides such drastic reductions by exploiting the homogeneity of application's performance (applications behaviors that repeat millions of times). By applying rigorous statistical methods, we can identify the minimal sample that asses application performance with a desired confidence level.

Using *FLEXUS* [33] and its multiprocessor checkpoint implementation, *flex points*, a multiprocessor simulation can turnaround of only 10 to 100 CPU hours rather than 10 to 20 CPU years required without sampling.

B.2 The SMARTS Methodology

In this section we present the basics behind statistical sampling and how they apply to simulation.

B.2.1 Statistical Sampling

Table B.1 summarizes the standard statistical sampling variables and relevant terminology used in this appendix. We select a sample of n elements (sampling units) at random from a population of N elements. Measurements are taken on the selected sampling units, and for a sufficiently large sample size (i.e., $n > 30$) the sampled results can be meaningfully extrapolated to provide an estimate for the whole population. In particular, the true population mean $\bar{X}$ is estimated by the sample mean $\bar{x}$. The coefficient of variation can be calulated as $V_x = \sigma_x/\bar{X}$. The likelihood that $\bar{x}$ is a good estimate of $\bar{X}$ improves with sample size and decreases with V_x.

Population variables		**Sample variables**	
N	size	n	size
$\bar{X}$	mean	$\bar{x}$	mean
σ_x	standard deviation	$\hat{V}_x$	coeff. of variation
V_x	coeff. of variation	$(1-\alpha)$	confidence level
		$\pm\epsilon\bar{X}$	confidence interval
		k	systematic-sampling interval

Table B.1: Sampling variables.

Formally, the confidence in a mean estimated is jointly quantified by two independent terms: confidence level $(1-\alpha)$ and confidence interval $\epsilon \cdot \bar{X}$. An acceptable interpretation of confidence level is that, for a given sample there is a $(1-\alpha)$ probability that $\bar{x}$ is within $\epsilon \cdot \bar{X}$ of $\bar{X}$.

The confidence interval achieved by a sample (assuming $N \gg n \gg 1$ to simplify) is

$$\pm([(z * V_x)/\sqrt{n}] * \bar{X} \tag{B.1}$$

where z is the $100[1-\alpha/2]$ percentile of the standard normal distribution.

B.2.2 SMARTS Technique

Although statistics provide us with probabilistic guarantees that estimated results are representative, these guarantees do not ensure that estimated results are error-free. One of the most common cause of *bias* (errors introduced into the individual measurements of a sample) is the cold-start effect of unwarmed microarchitectural structures, for example empty caches that result in incorrectly low performance estimates and empty networks that produce overly optimistic performance.

To solve this problem SMARTS has two strategies: detailed warming and functional warming (figure B.1). Before each measurement, SMARTS warms microarchitectural structures for which current state reflects the history of a small, bounded set of recent instructions: reorder buffer, issue queue, etc. This takes place through detailed warming: brief simulation (a few thousand instructions) with the complete detailed performance model sufficient to warm such small structures.

The second component of the warming strategy is the functional warming. It addresses state updates between two measurements. SMARTS functionally simulates each instruction to update architectural state. In addition, it continuously updates structures with microarchitectural state

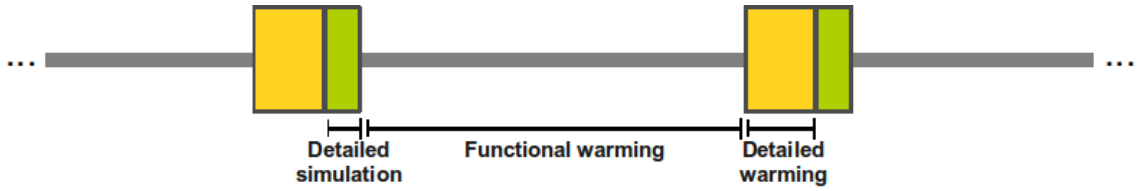
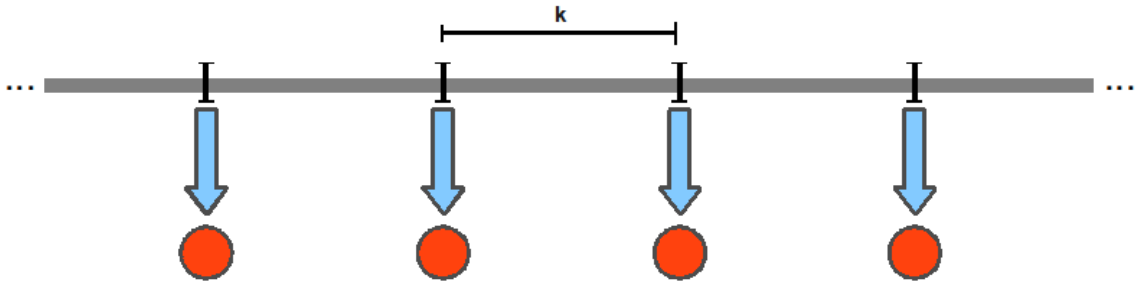


Figure B.1: Warming approaches for simulation sampling. Detailed simulation warms state before each sampling unit, which will be detailed simulated. Large structures (like caches, TLB) are functionally simulated, reducing the *bias* introduced by the cold-start effect.

that have long or unpredictable warming requirements: caches, translation look-aside buffers (TLBs), branch predictors. These structures sometimes require million of instructions to warm, and cannot be warmed sufficiently by a brief detailed warming period. Caches in particular have unpredictable requirements; functional warming eliminates the need to determine these requirements.

Although functional warming enables accurate performance estimation, it limits the simulation speed occupying more than 99 percent of simulation runtime. Live points (called flex points in the multiprocessor sampling and so that in this project) provide an alternative to functional warming that reduces simulation turnaround without sacrificing accuracy. A flex point stores the neccesary data to reconstruct warm state for a simulation sampling execution window (figure B.2).



(a) Functional simulation for flex points creation.

(b) Simulation from flex points.

Figure B.2: Simulating with flex points. We simulate in two steps: first (figure B.2a), we run the complete functional simulation storing the necessary data to reconstruct warm state for structures like caches, TLBs and branch predictors. This step will create a library of flex points. In the second step (figure B.2b), we run detailed simulation from those flex points, including prior detailed warming. Flex points are independent (self-contained) so we can launch simulations from flex points in parallel, reducing even more the simulation time. Finally, flex points can be also reused. If the functional parameters are the same, we just need one flex points library. The size of a flex point varies between 10 and 200MBytes in average.

From [33] we know that multiprocessor commercial applications need a detailed warming of 100,000 cycles and a sampling unit size of 50,000 cycles for a target confidence interval 95 $\pm$ 5.

B.3 SMARTS for our problem

To design a sampling simulation to meet a certain confidence one begins by determining an appropriate n based on the required confidence and V_x using the equation B.1. The true coefficient of variation of a population is rarely available in practice (unless the entire population is examined). Instead, we use $\hat{V}_x$, the coefficient of variation of a sufficiently large initial sample. If the initial sample n_i does not achieve the desired confidence, then the required size of a subsequent sample can be computed using $\hat{V}_x$, where

$$n \geq ((z * \hat{V}_x)/\epsilon)^2 \tag{B.2}$$

For microarchitecture simulation we select systematic sampling as an approximation of random sampling. Thus, we select sampling units from an ordered population at a fixed sampling interval k such that $n = N/k$.

We simulate the workloads described in table 4.2 with the initial number of samples described in table B.2. We assume a confidence level of 95% ($\alpha = 0.05$) and we want to achieve an error $\epsilon \leq 5\%$. With the first simulations we calculated the coefficient of variation ($\hat{V}_x$) of each workload for the specific configuration and the confidence interval (table B.3).

Workload	**Sample size (n)**	**Cycles between sample units (k)**
Web Apache	96	25,000,000.00
Web Zeus	96	25,000,000.00
OLTP DB2	192	25,000,000.00
OLTP Oracle	192	25,000,000.00
DSS Query1	10	100,000,000.00
DSS Query2	67	5,000,000.00
DSS Query17	165	2,000,000.00

Table B.2: Workloads, sample sizes and cycles between sample units for each workload. The numbers correspond to the 16-core workloads utilized in this project.

Workload	V_x	**Error ϵ (%)**
Web Apache	0.36	11 %
Web Zeus	0.34	11 %
OLTP DB2	0.26	8 %
OLTP Oracle	0.48	15 %
DSS Query1	0.27	17 %
DSS Query2	0.64	15 %
DSS Query17	1.08	17 %

Table B.3: Workloads, coefficient of variation and error (%). The numbers correspond to the results obtained for 16-core workloads, initial sample.

As the results do not achieve the desired confidence, we calculate the new sample size using the equation B.2. The use of systematic sampling and the characteristics of our benchmarks do not allow a linear mapping between the sample size theoretically required (the minimum sample size) and the sample size we actually use. Being sure the sample size we use is equal or greater than the minimal sample size, our results will achieve the desired confidence, yet optimal simulation speed is given up in exchange of a lower error. Table B.4 collects the adjusted parameters

as an example for 16-core benchmarks.

Workload	Minimum sample size (n)	Systematic-sampling interval (k)
Web (Apache)	197	8,300,000.00
Web (Zeus)	184	12,500,000.00
OLTP (DB2)	106	25,000,000.00
OLTP (Oracle)	352	12,500,000.00
DSS Query1	112	8,300,000.00
DSS Query2	627	500,000.00
DSS Query17	1799	180,000.00

Table B.4: Workloads, minimum sample size and systematic-sampling interval calculated for 16-core workloads.

With the new sampling size we can launch again simulations which achieve the desired confidence. Figure B.3 shows an example of the error obtained between the first simulation and the second for SPECWeb99 over Zeus. Given the first sampling size, the results may not be conclusive, as the error of the data is very big.

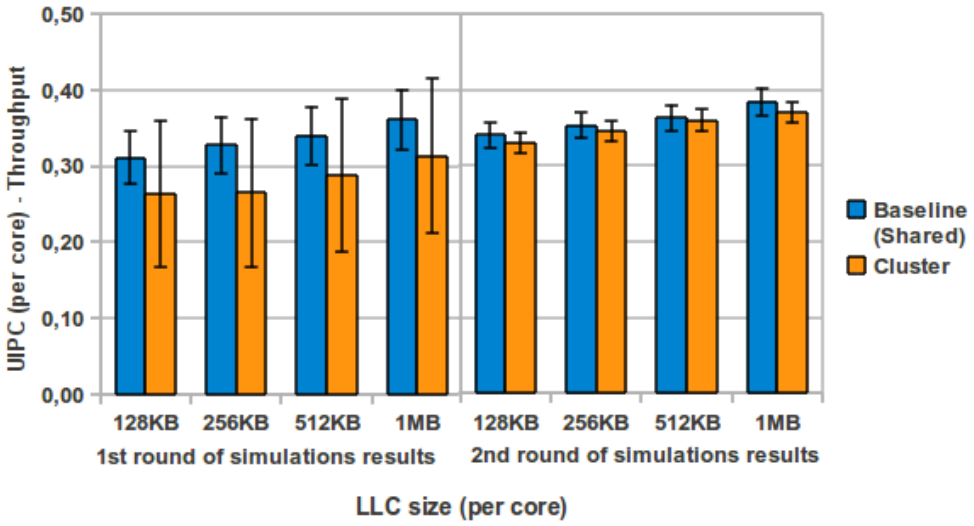


Figure B.3: Example of error reduction by increasing the sample size. The figure corresponds to the benchmark SPECWeb99 over Zeus. On the left part of the figure it can be observed how the insufficient number of samples leads to a huge error. Applying the method described above, we are able to reduce the error from 10% (in the case of the Baseline) and 30% (in the case of the Cluster) to an error between 4 and 5%.

Appendix C
Complementary (additional) results

This appendix extends the results collected in chapter 5. First, we summarize the conclusions of [33] about performance metrics explaining why we used them in our project and why conclusions can be drawn from them. Then, we explored all the results obtained and make an exhaustive study based on the characteristics of our system and the different benchmarks.

C.1 Performance metrics

Instruction interleaving across processors in multiprocessor benchmarks varies over multiple benchmarks runs and, therefore, it can cause changes in the dynamic instruction stream as races resolve differently on different runs. On real hardware, we can launch the benchmark repeatedly to exercise the possible interleavings. However, using statistical sampling is not clear how to construct a population with our simplified simulation model (flex points creation, see appendix B).

Fortunately, the commercial applications we are running (Web serving, online transaction processing and decision-support queries) are throughput applications, allowing us to construct the population of interleavings efficiently. In throughput applications, a server process satisfies a sequence of arriving transactions, queries, or requests (we call them transactions in general). Throughput application benchmarks consist of long sequences of randomly arriving transactions. Because those transactions arrive at random, a single run will cover the range of possible transaction interleavings. We draw our sample by selecting measurement locations over a time window that has proven reliable on real hardware [1].

We typically report the performance of throughput applications in terms of transactions per second. However, transactions are too long for a simulator to execute. That implies that sampling transaction throughput would require simulating seconds to minutes of real CPU time (which means years of full-system simulation time) to obtain high confidence results.

Although the time to complete a particular number of transactions varies greatly, the amount of work the database or Web server process must perform to complete a certain transaction type does not vary. As a result, the rate at which user-mode instructions complete is linearly proportional to transaction throughput [14, 33]. The linear relationship between user-instruction throughput and transaction throughput lets us sample user-instructions per cycle (UIPC) to assess transaction throughput. We define UIPC as the number of user-mode instructions that commit divided by all measured cycles. The commonly used metric of instructions per cycle

cannot be used because it is not proportional to transaction throughput, as it includes many system instructions that do not contribute to forward progress.

We report then UIPC as a measure of the application throughput with a confidence level of 95% ($\alpha = 0.05$) and error around 5%. Since cache miss rate (in this case LLC miss rate) could be deceptive in some cases (such as very high L1 hit rates) we prefer LLC misses per 1000 instructions (MPKI). We report the LLC misses per 1000 instructions (MPKI) for the different designs explored and different LLC sizes. LLC MPKI is architecture and block size dependent, but these parameters are the same among our designs.

C.2 Benchmarks

In this section we make an overview about the benchmarks utilized; detailed aspects are collected in table 4.2.

C.2.1 Web server: SPECWeb99 benchmark

SPECweb99 [29] is a benchmark used to measure the performance of HTTP servers. It uses one or more client systems to create the HTTP workload for the server. Each client sends HTTP requests to the server and then validates the response received.

We run SPECWeb99 benchmark over Apache and Zeus with 16k connections in both cases.

C.2.2 Online transaction processing: TPC-C benchmark

As an OLTP system benchmark, TPC-C [31] simulates a complete environment where a population of terminal operators executes transactions against a database. The benchmark is centered around the principal activities (transactions) of an order-entry environment. These transactions include entering and delivering orders, recording payments, checking the status of orders, and monitoring the level of stock at the warehouses.

In the TPC-C business model, a wholesale parts supplier (the Company) operates out of a number of warehouses and their associated sales districts. An operator from a sales district can select, at any time, one of the five operations or transactions offered by the Company's order-entry system. The most frequent transaction consists of entering a new order which, on average, is comprised of ten different items. Another frequent transaction consists in recording a payment received from a customer. Less frequently, operators will request the status of a previously placed order, process a batch of ten orders for delivery, or query the system for potential supply shortages by examining the level of stock at the local warehouse. A total of five types of transactions, then, are used to model this business activity.

We run OLTP TPC benchmark C over IMB DB2 server (v8) and Oracle 10g Enterprise Database Server. In both cases the database size is 10GB (100 warehouses).

C.2.3 Decision support systems: TPC-H DSS benchmark

The TPC Benchmark H (TPC-H) [32] is a decision support benchmark. It consists of a suite of business oriented ad-hoc queries and concurrent data modifications. The queries and the data populating the database have been chosen to have broad industry-wide relevance while maintaining a sufficient degree of ease of implementation. This benchmark illustrates decision support systems that:

- Examine large volumes of data
- Execute queries with a high degree of complexity
- Give answers to critical business questions

TPC-H evaluates the performance of various decision support systems by the execution of sets of queries against a standard database under controlled conditions. The minimum database required to run the benchmark holds business data from 10,000 suppliers. It contains almost ten million rows representing a raw storage capacity of about 1 gigabyte. Our implementation has a database population of 1 gigabyte.

We run the DSS TPC benchmark H over IBM DB2 server (v8). For our study we select between the 22 queries three representative ones: queries 1, 2 and 17.

Query 1: Pricing Summary Report (Q1)

This query reports the amount of business that was billed, shipped, and returned. The Pricing Summary Report Query provides a summary pricing report for all line items shipped as of a given date.

Query 1 scans one table of the database (full scan) and makes an aggregate operation on rows of the given table, so it is a representative scan-based query of the benchmark.

Query 2: Minimum Cost Supplier (Q2)

This query finds which supplier should be selected to place an order for a given part in a given region.

Query 2 accesses 5 tables of the database, with four hash joins. It is a representative join-dominated query of the benchmark.

Query 17: Small-Quantity-Order Revenue (Q17)

This query determines how much average yearly revenue would be lost if orders were no longer filled for small quantities of certain parts.

This query accesses two tables, makes one hash join, and several aggregate operations. We consider this query in the mixed-behavior category.

C.3 Results

Figure C.1 shows the best result for each workload for Shared and Cluster and the harmonic mean. The LLC sizes that provides the best performance are collected in table C.1. In all the cases, except for SPECWeb99 over Zeus and DSS TPC-H Query 17, the Cluster design works better than the Shared design, and so does it in average. The results in general follow this tendency for all the benchmarks and cache sizes.

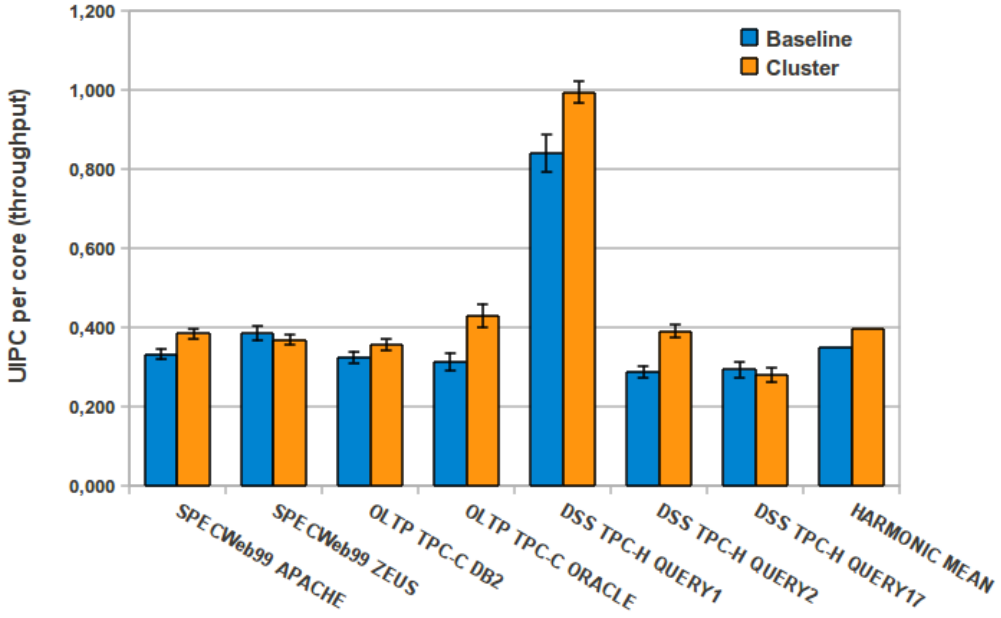


Figure C.1: Best UIPC per core for server workloads.

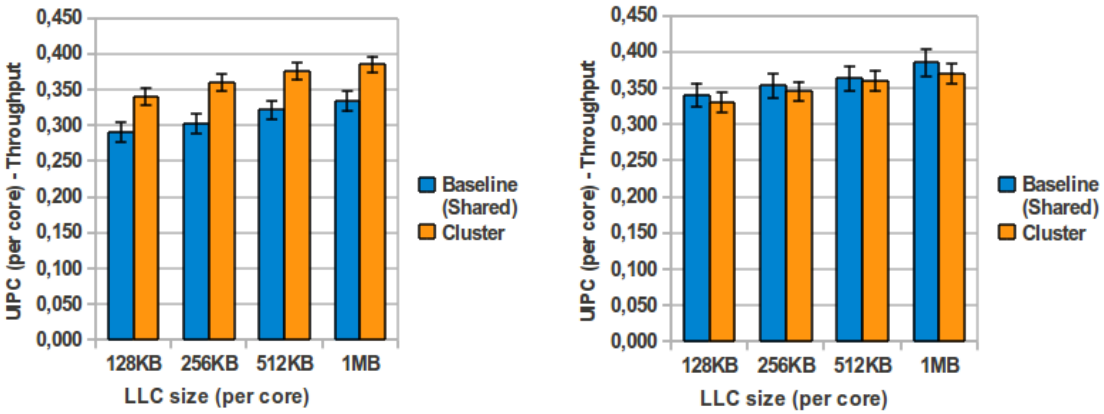
Workload	Best cache size	
	Shared design	Cluster design
SPECWeb99 APACHE	16 MB	4 MB
SPECWeb99 ZEUS	16 MB	4 MB
OLTP TPC-C DB2	16 MB	4 MB
OLTP TPC-C ORACLE	16 MB	4 MB
DSS TPC-H QUERY 1	8 MB	512 KB
DSS TPC-H QUERY 2	8 MB	4 MB
DSS TPC-H QUERY 17	4 MB	2 MB

Table C.1: Best cache sizes for the different workloads/designs. The cache size corresponds to the cache size that is visible for each processors (i.e., in a Cluster design a cache size of 4MB corresponds to a total cache size of 16MB on-chip).

In the remaining of the section we will go into the results for the different benchmarks categories in depth.

C.3.1 Web server benchmarks

Our Web server benchmarks category includes SPECWeb99 over Apache and Zeus. Figure C.2 shows the UIPC (throughput) for both workloads. Both present a similar tendency regarding throughput (UIPC increases as the last level cache size does), yet Apache prefers a Cluster design for all the configurations explored and Zeus gets benefit from a Shared design. We also see that the difference between designs (for Zeus) is very small.

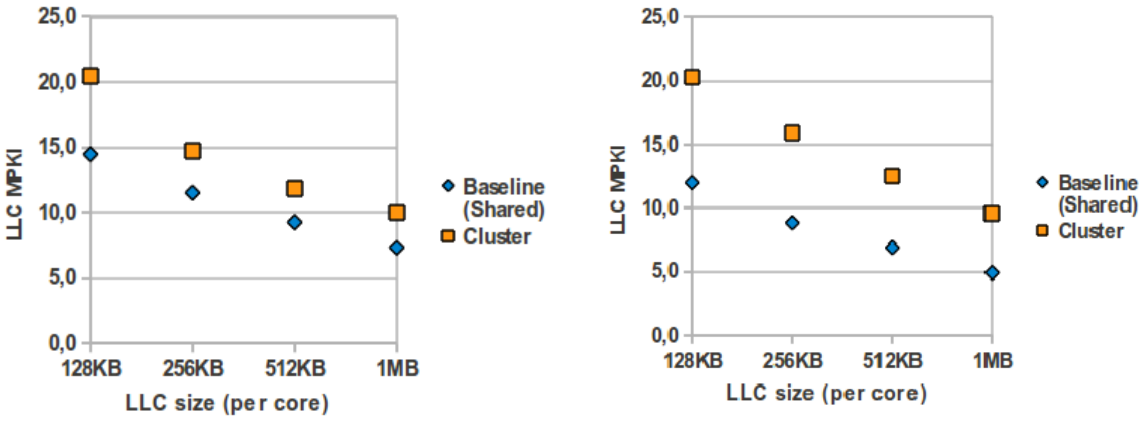


(a) UIPC for Web Apache.

(b) UIPC for Web Zeus.

Figure C.2: UIPC for Web Apache and Zeus.

The LLC MPKI in figure C.3 shows a decrease in the LLC MPKI with larger caches. Observing both graphs together (figures C.2 and C.3), we can conclude that the reduction in the miss rate compensates the higher latency of a large cache.



(a) LLC MPKI for Web Apache.

(b) LLC MPKI for Web Zeus.

Figure C.3: LLC MPKI for Web Apache and Zeus.

These results indicate that Web server benchmarks benefit from large last level caches. For Apache it is clear that the Cluster design outperforms the Shared design; Zeus presents better results for Shared design, yet the difference is very small. Zeus seems to get advantage of sharing

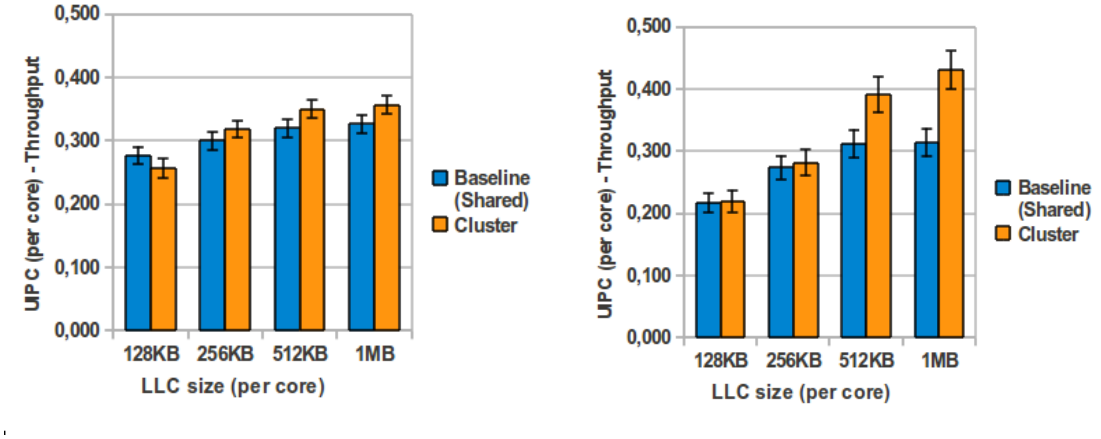
data, and size is more critical than latency, which is why the Shared design performs better. For both benchmarks the best performance corresponds to the largest LLC (1MB per core).

C.3.2 OLTP benchmarks

Our OLTP benchmarks include TPC-C over DB2 and Oracle.

Ailamaki *et al.* [17] pointed out in their study about databases workloads the requirements of this kind of applications. As long as the primary working set fits in the LLC, the performance increases. From there, larger caches may degrade performance.

We can see this tendency in figure C.4 (UIPC per core). Here it is important to understand the difference between our Baseline and Cluster designs (see Chapter 3 for a more detailed explanation and figures). Both designs have the same amount of LLC on-chip. Nevertheless, while in the Shared design each core is able to access the whole LLC (all the on-chip capacity is visible), in the Cluster design each core can only access a fourth of the total LLC (i.e., considering 1MB per core of LLC, in the case of Shared (Baseline) design each core sees a LLC of 16MB; in the case of the Cluster design each core sees 4MB of LLC, as the cache is private for each cluster).



(a) UIPC for OLTP DB2.

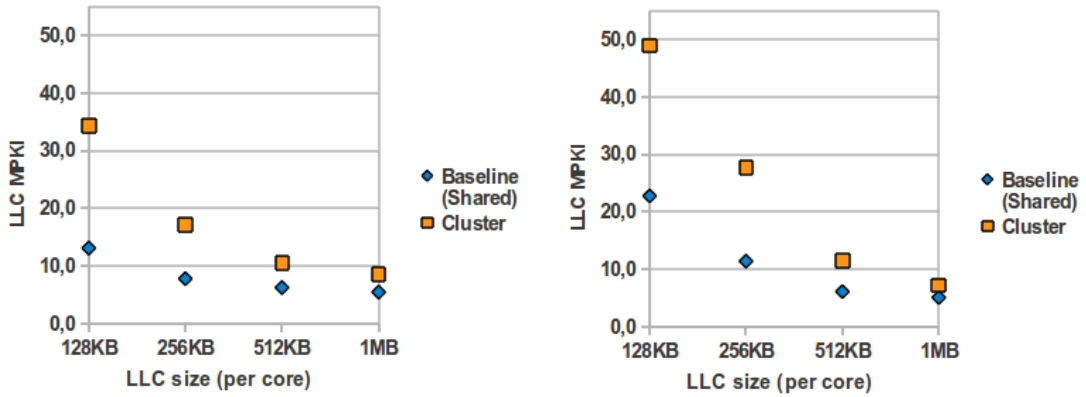
(b) UIPC for OLTP Oracle.

Figure C.4: UIPC for OLTP DB2 and Oracle.

Both benchmarks prefer a Shared design with small last level caches. With the sufficient size of LLC (observing figure C.4, 512KB per core), the Cluster design obtains a better performance. In fact, Oracle presents a difference of 0,1 UIPC (per core) when we consider 1MB LLC (also per core).

LLC MPKI (see figure C.5) corroborates the same. We see a dastric decrease in the number of misses per 1000 instructions when we double the cache from 128KB to 256KB per core (Cluster in both benchmarks), but almost no variation from 512KB to 1MB of cache per core (Shared and Cluster for both workloads).

Exemplifying all the mentioned above, figure C.6 shows the execution time (user) breakdown for OLTP Oracle (Cluster design). We can see that a private cache of 512KB per cluster (128KB



(a) LLC MPKI for OLTP DB2. **(b)** LLC MPKI for OLTP Oracle.

Figure C.5: LLC MPKI for OLTP DB2 and Oracle.

per core) is not able to capture the primary working set and the amount of time spent off-chip is huge. With 1MB of last level cache per core the off-chip spent time is reduced from 50% (128KB per core) to less than 10% of the total execution time (user). That decrease translates into twice the performance per core.

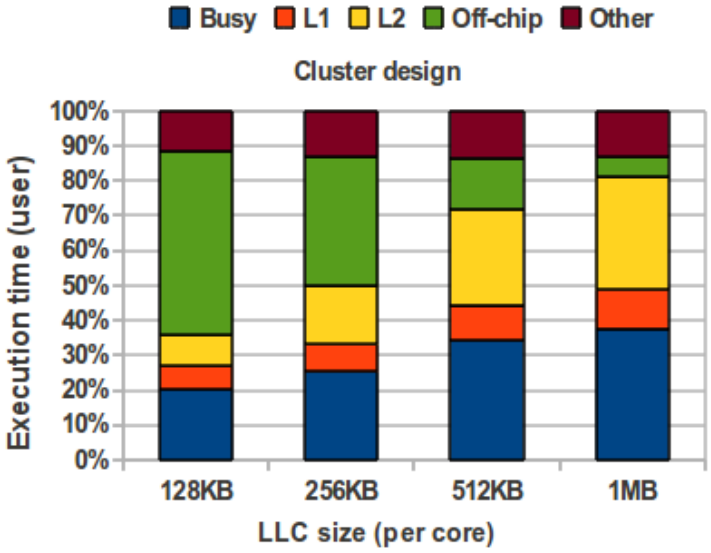


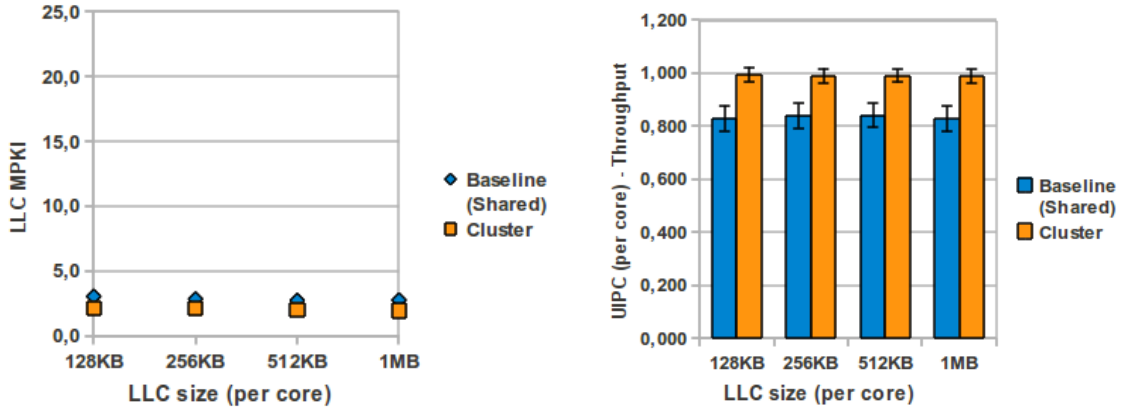
Figure C.6: Execution time breakdown (user) for OLTP Oracle (Cluster).

C.3.3 DSS benchmarks

As DSS benchmarks we study TPC-H queries 1, 2 and 17 on DB2, which are, respectively, scan-dominated, join-dominated and mixed-behavior.

Query 1 presents a very small LLC MPKI (figure C.7a). The first level cache is able to cap-

ture most of the accesses (the data cache has a miss rate of 5% in average, and the instruction cache has 0,2%). That means that increasing the size of the last level cache does not influence performance, as the number of accesses is very small. Our Cluster design works better (figure C.7b) in all the cases, and the performance does not vary.

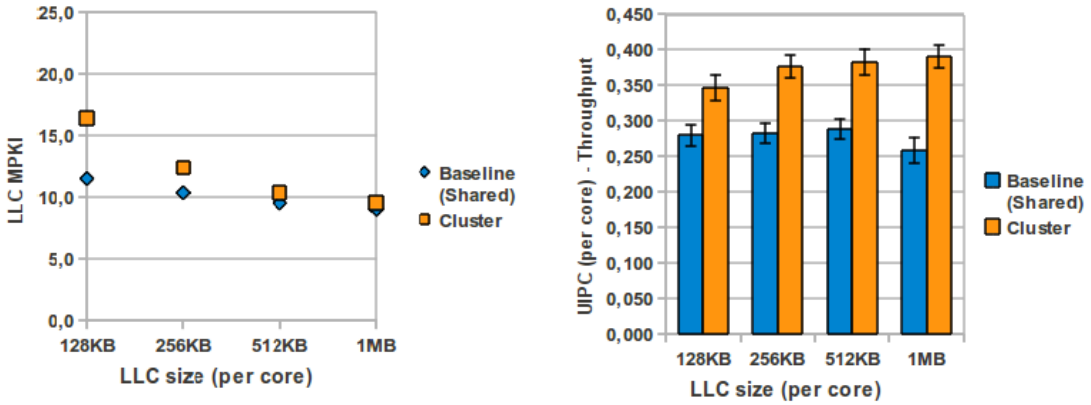


(a) LLC MPKI for DSS Query 1.

(b) UIPC for DSS Query 1.

Figure C.7: LLC MPKI and UIPC for DSS Query 1.

For query 2 the Cluster design also outperforms the Shared design (figure C.8b). In the case of the Cluster design, UIPC increases from 128KB to 256KB per core, but it is practically the same from 256KB to 1MB per core. In the case of the Shared design, it does not change from 128KB to 512KB and it does decrease from 512KB to 1MB. The number of last level cache misses per 1000 instructions (figure C.8a) is practically constant for the Shared design among different sizes.



(a) LLC MPKI for DSS Query 2.

(b) UIPC for DSS Query 2.

Figure C.8: LLC MPKI and UIPC for DSS Query 2.

The execution time breakdown (user) for this workload (figure C.9 for Cluster) shows how the time we spend off-chip decreases when we move from 128KB to 256KB per core. As we continue increasing the size of the last level cache, that decrease is lower than 5%. Besides, the

amount of time spent off-chip is significant with a 20% of the execution time (user). Our Cluster design works better in any case with a clear difference.

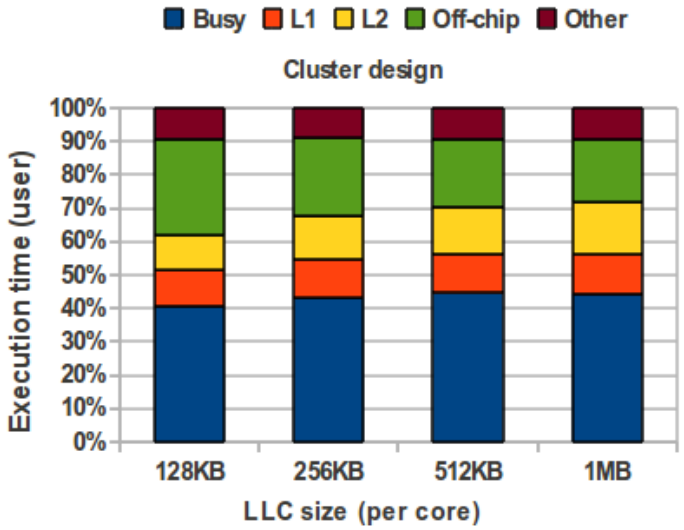
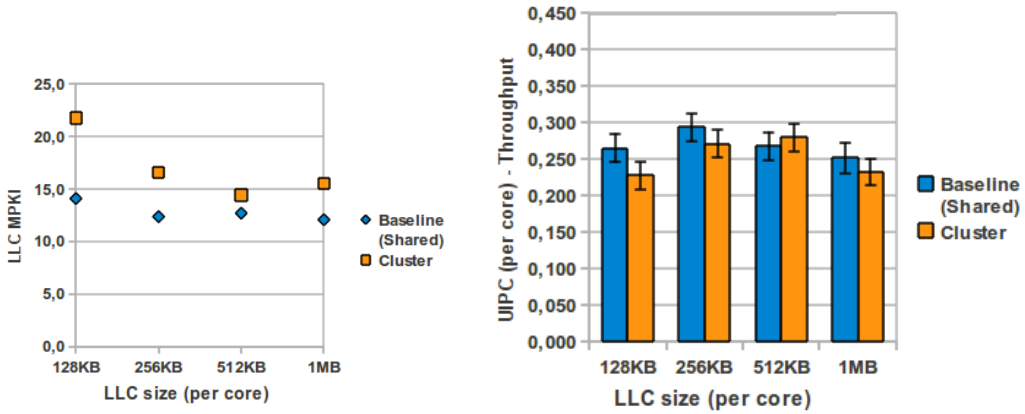


Figure C.9: Execution time breakdown (user) for DSS Query 2 (Cluster).

Finally, figure C.10 collects the results for query 17 (figure C.10b shows throughput and figure C.10a shows the LLC MPKI).



(a) LLC MPKI for DSS Query 17.

(b) UIPC for DSS Query 17.

Figure C.10: LLC MPKI and UIPC for DSS Query 17.

For this query, a shared last level cache is preferable in all cases except for a size of 512KB per core. If we observe the UIPC graph (figure C.10b), we see how our Shared design presents an increase in performance when we move from 128KB to 256KB of LLC per core, yet the performance decreases if we continue adding capacity to the last level cache. For this design, 128KB of LLC per core provides better performance than 1MB per core. The Cluster design presents the same tendency but the decrease comes when we move from 512KB to 1MB.

Both UIPC and LLC MPKI graphs suggest that this query has a low locality and, therefore, large caches decrease performance. The better performance of the Shared design also suggests that this query benefits from sharing data between processors. The execution time breakdown (user) in figure C.11 (Shared) shows that the amount of time we spend off-chip is around 30%.

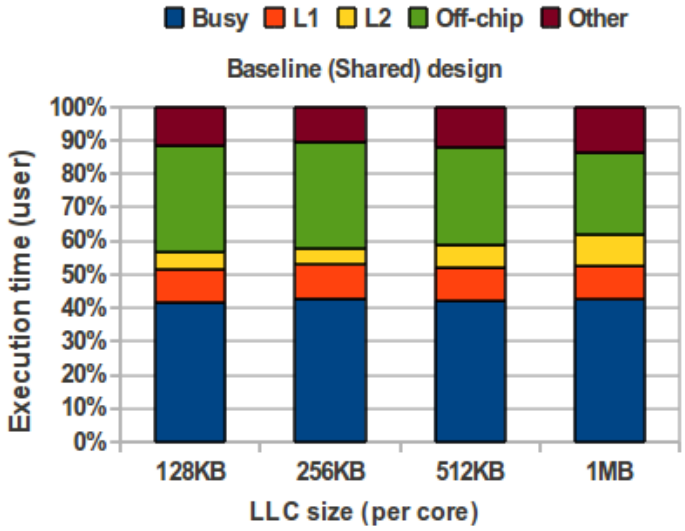


Figure C.11: Execution time breakdown (user) for DSS Query 17 (Baseline).

C.3.4 Summary

We have seen how different types of workloads present different characteristics concerning sharing and last level cache size and organization. In general, a private organization is preferable for most of the benchmarks and in average.

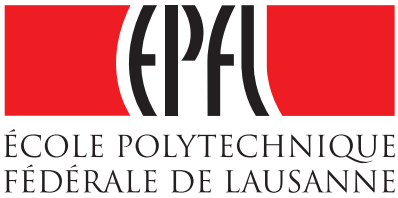
Web server benchmarks present an increase in performance with larger last level caches.

OLTP benchmarks have a minimum requirements of LLC. Once this requirements are satisfied, the design which allows a faster access (in our case, the Cluster design) performs better.

DSS benchmarks have different behavior depending on the category. Our scan-based query's performance does not change when increasing size, as the first level cache is able to capture most of the accesses. Out join-dominated query clearly benefits from private designs. Finally, the query with mixed-behavior has low locality, which translates into better performance with smaller last level caches (lower access time).

Appendix D
English Report

Este apéndice contiene la versión en Inglés de la memoria tal y como fue entregada en la EPFL (Suiza). Los apéndices no se incluyen porque forman parte de la memoria en español. Tampoco se incluyen los contenidos en los índices de contenido, tablas y figuras.



Proyecto Fin de Carrera
Ingeniería en Informática

Coherent vs. non-coherent last level on-chip caches: an evaluation of the latency and capacity trade-offs

Alexandra Ferrerón Labari

Director: Babak Falsafi
Ponente: Darío Suárez Gracia

Parallel Systems Architecture Lab.
Faculté Informatique et Communications
École Polytechnique Fédérale de Lausanne

Departamento de Informática e Ingeniería de Sistemas
Centro Politécnico Superior
Universidad de Zaragoza

Curso 2010/2011
Noviembre 2010

Abstract

The exorbitant energy consumption in current data centers and the growing concern for the environment, make Information Technology consider how to reduce costs for future data centers, while preserving the environment.

ARM in a consortium with Nokia, IMEC, EPFL (Ecole Polytechnique Federale de Lausanne) and UCY (University of Cyprus) leads the project EuroCloud, aiming to develop a new generation of energy-conscious 3D server-on-chip for Green Cloud Services. The EuroCloud project proposes a very low power 3D server-on-chip using many ARM cores, hardware accelerators, and integrated 3D DRAM.

In this project we studied one of the key components of the EuroCloud project ARM's chip from a performance and power-aware perspective. In particular, we focused on the on-chip memory hierarchy, making a comparison of different options for its organization. The configuration of the cache hierarchy affects the average memory access time and consequently influences performance.

The chip we studied consists of two clusters. Each cluster contains two processors with their own private level one caches, and a portion of the second level of the cache hierarchy (in this case the last level of the cache hierarchy). This last level of cache is, therefore, physically distributed among the clusters, and it can be configured in different ways. It specifically supports two organizations: shared and private organization.

In this project we analyzed two content management organizations: a Shared organization in which the two clusters share the last level of cache memory, and which is trying to make the most of the cache capacity, and a Clustered organization, where the last level cache is private to each cluster. In the latter case we give priority to faster access to this level of the hierarchy. Within the Cluster organization, we studied the possibility of introducing a coherence mechanism.

After extensive research on the state of the art and the organization of the chip, as well as its architecture, we modelled the two designs in our simulation platform and simulated representative commercial workloads from three different categories: Web server, online transaction processing (OLTP), and decision support systems (DSS).

We analyzed in detail the results for different cache sizes and concluded that, in general, a Cluster organization performs better. Large caches benefit from the Cluster organization due to the lower latency. For smaller cache sizes and / or workloads with low locality, our Shared design provides better results. We also concluded that coherence mechanisms for this level of the cache hierarchy when running this type of commercial applications are unnecessary and should be avoided. Additionally, we extended the simulation environment as well as studied the simulation methodology to obtain more accurate results.

Contents

List of Tables

List of Figures

Chapter 1
Introduction

Information technology starts to consider energy costs and environmental problems as a first-class constraint in chip designs. Power consumption is one of the main contributing factors in the total cost of current Data Centers [7], where microprocessors and the memory system are the most costly and power-consuming components in a server [13]. In the past decades, microprocessor designers have relied on voltage scaling (reduce supply voltage) for cutting down power consumption. However, this technique comes with an increase in the leakage current, which limits how far it can be taken. Integrating many low-power processor cores on-chip or move to a new class of "embedded-based" server architectures seem to be a way to tackle the power consumption problem of future Data Centers.

ARM in a consortium with Nokia, IMEC, EPFL (Ecole Polytechnique Federale de Lausanne) and UCY (University of Cyprus) leads the project EuroCloud, aiming to develop a new generation of energy-conscious 3D server-on-chip for Green Cloud Services [26]. Cloud Computing appears as a new paradigm shift, proposing internet-based computing, whereby shared resources, software, and information are provided to computers and other devices on demand. The EuroCloud project proposes a very low power 3D server-on-chip using many ARM cores, hardware accelerators, and integrated 3D DRAM.

Commercial applications (e.g., database and web) and emerging Cloud Computing applications (e.g., video/music streaming, song recognition, and data mining) present high thread– and memory-level parallelism, and can take little advantage of complex out-of-order cores, which are focused on extracting instruction level parallelism. This kind of applications could benefit of many simple processor cores in which the individual thread latency is less important than the aggregate thread throughput.

In this project we study a key component of the EuroCloud project ARM's chip from a performance and power-aware perspective. We focus on the on-chip memory hierarchy, comparing different on-chip memory configurations for the chip multiprocessor. We explore the desirability of having a shared or private last level cache (LLC), and in the case of a private cache, whether the caches should be coherent or non-coherent.

1.1 Scope of the Project

This project has been developed in the Parallel Systems Architecture Lab (PARSALab) at the Ecole Polytechnique Federale de Lausanne (Switzerland) and the Computer Architecture Group

(Grupo de Arquitectura de Computadores) at the University of Zaragoza.

The multiprocessor we studied consists of two dual-core clusters where each processor has private L1 caches. The last level cache (LLC [1]) is physically distributed among the clusters and it can be configured in different ways. Depending on that configuration, the average memory access time (which depends on three main parameters: latency, bandwidth and hit rate) will be influenced by one way or another: a private cache configuration will decrease the latency, while shared caches will increase the hit rate. Besides, in case of a private cache configuration, we have to determine if a explicitly management of the cache coherence should be made.

In this project we analyze the different configurations of these last level caches searching for the best option based on the simulation of representative workloads. As work environment we use *FLEXUS* [33]. *FLEXUS* is a simulator developed by the Computer Architecture Lab at Carnegie Mellon University (CALCM) which models the SPARC v9 ISA and can execute unmodified commercial applications and operating systems.

1.2 Objectives

The objective of the poject is to analyze the best memory hierarchy for the EuroCloud project's chip. The main tasks the project can be divided in are:

1. Study of the ARM architecture, its chip organization and related work to the area
2. Extension of the research simulation platform to model the ARM multiprocessor
3. Model of the different organizations for the cache memory hierarchy on-chip
    - shared / statically-interleaved LLC's
    - private / coherent LLC's
    - private / non-coherent LLC's
4. Study of the simulation methodology and simulation of representative workloads
5. Analysis of results

We achieved the main objectives making a comparison among the different configurations proposed by simulating representative commercial workloads. Due to the lack of support for modeling heterogeneous chip multiprocessors from our infrastructure, we modelled a 16-core symmetric multiprocessor from which we can extrapolate the results.

Besides, we extended the simulation environment for the simulation of multiprocessors with two cores.

[1] In this work, all the references to the multiprocessor's LLC refer to the L2 cache

1.3 Report organization

The remaining of the report is organized as follows: Chapter 2 presents the related work; Chapter 3 explores the alternatives proposed; in Chapter 4 the methodology we followed is explained; Chapter 5 collects the main results and Chapter 6 concludes.

Also found as appendix:

A. Project management: it includes schedules and effort.

B. Statistical sampling simulation methodology: the approach we used to reduce the simulation time.

C. Complementary (additional) results: detailed study of the results obtained.

1.4 Acknowledgements

I would like to thank my project director Babak Falsafi for giving me the opportunity of having some contact with the research world. Also all the guys at PARSALab, especially Mutaz, Mike, Pejman and Mammad, who were always available to answer my questions about the simulator environment and about this project in general. Thank you also Mehdi, a good interconnects-guy to share an office with.

Special thanks to Dario, who has been always there for solving my problems and make my work easier. Thank you very much for all your help.

For all the friends and all the coffees, both here in Zaragoza and in Lausanne. To Markus, for all the moments of listening my thoughts, helping me to make some order there.

Finally, I want to express my sincere gratitude to my family, especially to my parents. Without their support, nothing of this would be possible. If I am where I am, it is because of you.

Chapter 2
Related work

CMOS scaling trend is leading to an increase in the number of transistors available on a chip, but also to a relative increase in wire delays [18]. Chip multiprocessors (CMPs) are the common architecture for utilizing the numerous available transistors to achieve high performance. However, as the number of processors increases, so does the capacity pressure on the on-chip memory hierarchy. At the same time, CMP also requires its processors to have fast access to data. The last level on-chip cache (LLC) becomes a new bottleneck in the memory hierarchy, which not only needs to utilize its limited capacity effectively, but also has to mitigate the increasing latencies due to wire delays.

Large caches seemed to be a technique to exploit the available transistors on-chip to improve performance by increasing the cache capacity. Some examples of mega-caches on chip are the Dual-Core Intel Xeon 7100 with 16MB [28] or the Dual-Core Intel Itanium 2 with 24MB [36]. Nevertheless, large caches come at the cost of high access latency. Rising cache latencies penalizes each accesses and increases the number of stalls caused by L2 hits without changing the number of access to other parts of the memory hierarchy.

Ailamiki *et al.* [17] showed how database server applications did not get any improvement in performance or even deteriorated the throughput by up to 30% when increasing the LLC cache size from 4MB to 26MB. If the caches are able to capture the primary working set of this kind of workloads [1], then performance increases. If we continue increasing the size of the caches then the higher latency slows the common case (cache hits) introducing stalls in the execution, while the additional capacity is not able to lower the miss rate enough to compensate.

Facing these circumstances, we focused on the LLC organization and found two options for the LLC in CMPs: shared or private caches. A shared cache has a single copy of each cache line and allows the cores to share the cache capacity. However, shared caches are slow because of the wire delays associated with large caches and interconnections. Private caches are faster because they are smaller and can be located closer to each core, but they provide limited capacity to each core. Thus, shared or private caches can provide either capacity or fast access, but not both.

Recently, many effort has been made in trying to combine the advantages of shared and private cache organizations, proposing hybrid designs. In general, hybrid proposals use selective replication to balance latency and capacity [37, 6, 10]. Chang *et al.* in [9] present a unified framework to manage a CMP's aggregate on-chip cache resources by forming an aggregate shared

[1] This kind of workloads typically have a small primary working set which can be captured on chip, and a large secondary working set which is beyond the reach of on-chip caches for modern processors

cache through cooperation among private caches.

In the same research line, Reactive-NUCA (R-NUCA) [16] proposes to cluster the cores in groups of sharing to minimize off-chip misses. R-NUCA goes further and makes a study about the LLC references in commercial workloads, resulting in a classification of blocks that are likely to be private, shared or clustered. Based on that observation, they propose a cache design which reacts to the class of each reference and places blocks at the appropriate location in the cache cooperating with the operating system.

On the other hand, complex out-of-order cores are focused on extracting instruction level parallelism (ILP), while commercial applications exhibit a different pattern. For these kinds of applications the individual thread latency is less important than the aggregate thread throughput. We can increase the aggregate thread throughput with multithreaded processors, so events that normally stall the processor (cache misses) are hidden, thus increasing their utilization. In addition, using simple scalar processor cores reduces the design complexity and bug rates, as well as the power consumption.

A work by the University of Michigan and ARM proposes a new architecture called PicoServer [20], which claims that by moving to 3D integration the L2 cache can be removed. Performance can be maintained by using wide buses, a small on-chip DRAM memory, and increasing the number of cores, while saving a considerable amount of power. This hypothesis is supported by the idea that future server applications will run on CMPs with a large number of small in-order cores [12]. Nevertheless, increasing the number of cores comes at the cost of higher demand of data to feed those cores, so performance could be degraded if there is not enough bandwidth to provide data to the processors. The situation in which off-chip bandwidth is becoming a performance and throughput bottleneck is referred to as the bandwidth wall problem [27]. Besides, buses are a significant source of power loss, especially inter-chip buses, which are often very long and wide.

The EuroCloud project follows the idea of including many low-power processor cores on-chip and the use of 3D DRAM integration. 3D stacking technologies have received much attention in computer architecture in the last couple of years, because the dense vertical interconnects allow for systems with lower latencies and greater bandwidths [23]. These aspects are very attractive for servers and high throughput processors, because of their large memory footprint and significant levels of thread level parallelism. Standard off-chip DRAM is bandwidth-constrained due to limited pin count, slow due to chip crossing, and power-hungry due to I/O pads and driving circuitry [26]. 3D stacking DRAM on top of the logic in combination with hardware accelerators can eliminate those inefficiencies and address those issues.

Our work focuses on a key component (memory hierarchy on-chip) of a new embedded-based server-on-chip. From our knowledge, this is the first time that the performance of commercial workloads in embedded systems is studied.

Chapter 3
Cache alternatives for embedded multiprocessors

This chapter presents the different alternatives for the cache memory hierarchy on-chip proposed and studied in this project. We expound on the main characteristics and trade-offs of each alternative.

3.1 Background

The objective of a cache hierarchy is to minimize the latency of the frequently accessed data and maximize the performance. In an uniprocessor cache hierarchy, we achieve this objective by exploiting both spatial and temporal locality; this means that we move blocks through the hierarchy based on the access frequency, so the most referenced data gets closer to the processor and, therefore, we obtain faster access to that data. The same principle can be applied to multi-core cache hierarchies. The difference now is that we have to take into account whether cores have to share a given level in the cache hierarchy or whether a level is implemented as a physical block with uniform cache access (UCA) latency or as multiple physically distributed banks, where the latency varies between the cores and the cache slices spread across the die, leading to a Non-Uniform Cache Access (NUCA) latency to each bank [21].

As the number of cores and cache banks increase, physically distributed caches become more attractive from a design, manufacturing and scalability perspective [3]. But also collocating a portion of the cache near a subset of cores can reduce the access latency to that portion of the cache for those cores, instead of offering an equally high latency (UCA designs).

Tiled architectures have emerged as a solution to mitigate the rising access latency of the last level cache [21]. In this kind of architecture the die is divided into a large number of identical (or close-to-identical) tiles that are interconnected using a scalable and energy-efficient interconnect. Each tile contains a slice of the LLC (which may consist of multiple banks) so the cache is physically distributed on the die. In this way, cores can access very fast to the LLC slices which are close to them, paying the latency of travelling through an interconnection network if the data is in further slices (NUCA).

If the LLC is distributed across multiple tiles, private and shared caches introduce distinct trade-offs. In general, shared LLC is preferable if reducing the collective number of LLC misses is important, whereas an organization with private caches is preferable if reducing the LLC access latency and design complexity is important.

A shared cache design increases the effective capacity because only a single copy of each block resides in the cache. Nevertheless, assuming a statically interleaved distribution of the blocks, the blocks can be arbitrarily far away from the core(s) that request them, penalizing the average access time. On the other hand, a private design will allocate blocks near the core(s) that request those blocks, so we guarantee fast access. The downside is that since read-shared blocks will be replicated in multiple tiles, the effective cache capacity is reduced and, therefore, the off-chip traffic increases.

The coherency protocol may play also an important role. While in a shared design the coherence is implicit by construction, when we move to a private design we have again the possibility of choosing between two options: a coherent or a non-coherent cache. In the case that the LLC is coherent, the traffic on-chip (due to coherence messages) grows, with the consequent problems in energy dissipation and cooling [3]. Besides, coherence mechanisms reduce the available cache area and penalize data sharing. In the other case, non-coherent caches will demand extra traffic from outside the chip, with the consequent problems in off-chip bandwidth [27]. In case of choosing a private-coherent approach, a directory based protocol seems to be the best option to keep track of the location of the different copies on-chip [8, 30].

3.2 Server-on-chip basic organization and cache hierarchy options

In this project we evaluate two different LLC organizations for the EuroCloud project ARM's CMP (figure 3.1). The multiprocessor we studied consists of two dual-core clusters where each processor has private L1 caches. The LLC (in this case L2) is physically distributed among clusters. The communication between clusters is through an interconnection network. The Snoop Control Unit (SCU) is responsible for managing the interconnect, arbitration, communication, cache-to-cache and system memory transfers, as well as the coherence.

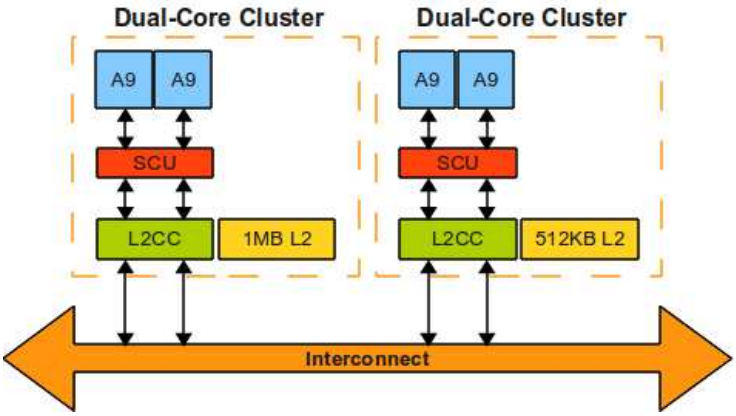


Figure 3.1: Design (approximated) of the ARMs chip. Only included the on-chip cache memory hierarchy. L2CC stands for L2 cache controller.

Our objective is to explore the different alternatives for the on-chip cache memory hierarchy organization, proposing two organizations for the LLC:

1. Shared LLC: the two clusters share the LLC on-chip. In this case a statically interleaved block distribution is supposed. Note that a shared design is coherent by construction.

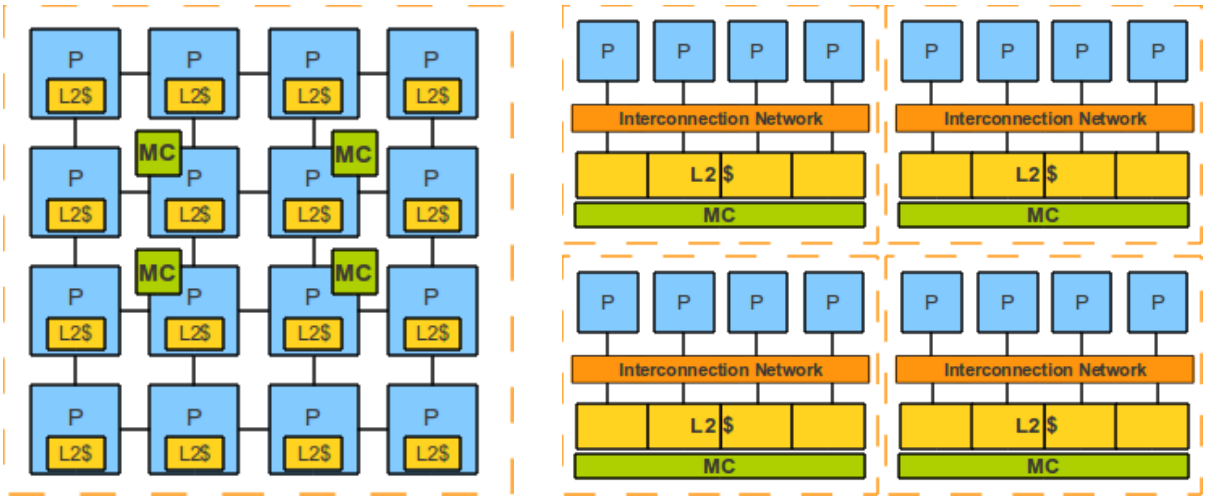
2. Private LLC: the LLC is private for each cluster. In this case we can consider either a coherent or a non-coherent design.

3.2.1 Alternatives explored

Due to the restrictions imposed by the simulation platform, we decided to extend our analysis to a 16 core CMP, exploring two different designs:

1. Shared LLC design (Baseline): it consists of a 16 nodes system, with NUCA, and a shared LLC (L2), interconnected by a 4 by 4 mesh (figure 3.2a). There are in total four memory controllers. This design pretends to exploit the cache capacity.

2. Cluster design (Cluster): it is also a 16 nodes system, but now the cores are divided in 4 clusters of 4 cores each one, where the LLC (L2) is shared inside the cluster, but they are private between the clusters (figure 3.2b). The LLC is a multibanked cache accessed via an interconnection network. There is one memory controller for each cluster. With this design we prioritize access latency.



(a) Shared LLC design (Baseline). **(b)** Cluster LLC design (Cluster).

Figure 3.2: Alternative designs explored.

We can see that the results obtained with this designs can be extrapolated to the original CMP (in any case our results my be degraded due to more cores on-chip and the consequent higher pressure).

3.2.2 Sizing the last level cache

Different workloads and different types of parallelism within these workloads present unique architectural and design challenges for the cache hierarchy to meet performance, scalability, and energy-efficient goals. In our study we focus on commercial applications as they are the preferable kind of applications that nowadays servers run.

Hardavellas *et al.* in [16] made a complete study about the requirements of commercial applications in terms of LLC capacity and organization, proposing a new schema of blocks distribution and cache organization. From that study we can extract several conclusions:

- L2 accesses naturally form three clusters with different characteristics: (1) instructions are shared by all cores and are read-only; (2) shared data are shared by all cores and are mostly read-write; (3) private data exhibit a varying degree of read-write blocks.
- The instruction working set for commercial workloads approximates the size of a single L2 slice (in their study, 1MB per slice).
- Data sets can be captured by the L1D cache.
- L2 hardware coherence mechanisms in a tiled CMP running server workloads are unnecessary and should be avoided.

Based on these conclusions we can affirm:

- If the LLC cache is able to capture the primary working set of the workload [17], then the design which guarantees the smaller latency will perform better.

We can formulate then our hypothesis declaring that as long as the LLC size is bigger than a determined size, we expect our Cluster design to perform better.

If the Cluster design is not able to capture the primary working set of the workload, then the data request will go to the off-chip memory, increasing the average access time. On the other hand, large shared caches with statically interleaved distribution of blocks may imply accesses to far-off slices of the last level cache and, consequently, a high latency because of the interconnection network.

Finally, implementing coherence mechanisms we could get benefit from the cache-to-cache on-chip transfers, yet we pay a high price in terms of on-chip traffic and hardware [9]. Cache-to-cache transfers are used to reduce the off-chip requests for local LLC misses, but these operations require three-way communication between the requester tile, the directory tile and the owner tile. This operation is significantly more costly than remote hits in the shared last level cache case, where a cache-to-cache transfer happens only if the line is held exclusive. Besides, the hardware required to implement the coherency reduces the available area for the LLC (and consequently limits the capacity).

Following our hypothesis of a better performance of the Cluster design, the addition of a coherency mechanism may degrade performance as well as complicate the hardware and increase the power consumption. For these reasons we decided not to explore a private coherent design.

To estimate the L2 cache size for our designs we assume a die size of 180 mm^2 in 45 nm technology. We account for the area of the system-on-chip components, allocating 65% of the die are for the tiles. We assume an area per Cortex A9-core of 3,1 mm^2, so the size left for each LLC slice is 4,2125 mm^2, which allows us to allocate a LLC slice up to 1MB without problems.

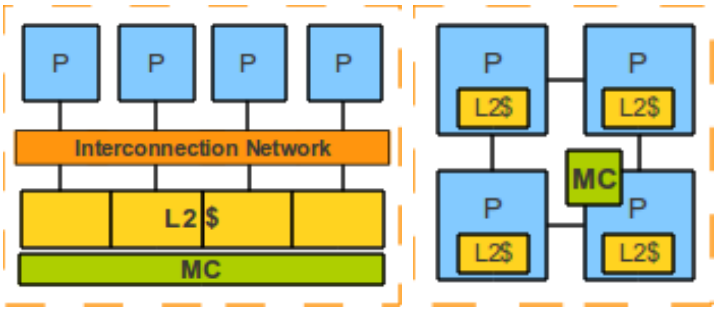
We fix the associativity of the last level cache to 16, since it is a normal value for this parameter in this level of the cache hierarchy [28].

To summarize, we explore cache sizes from 128KB per core to 1MB, which makes a total cache capacity from 4MB to 16MB on-chip, expecting the Shared design to perform better till a determined size of the L2 cache. From that size the Cluster design will get advantage of the smaller latency and present a better performance. We agree with the idea that coherency mechanisms in commercial applications should be avoided, and we do not expect that introducing a

coherency mechanism will increase performance.

3.2.3 Implementation details

In practice we did not implement the Cluster design as it was proposed in figure 3.2b. Instead we decide to simulate a tiled architecture interconnected by a 2 by 2 mesh. Figure 3.3 shows the proposed design (3.3a) and the implemented design (3.3b).



(a) Proposed design for Cluster. **(b)** Implemented design for Cluster.

Figure 3.3: Proposed and implemented designs for Cluster.

Note that low port crossbars have a lower average latency [22, 11], so we consider our implemented design (2 by 2 mesh) as a lower bound of performance.

Chapter 4
Methodology

This chapter presents the methodology followed in this project. We briefly present the simulation methodology and the benchmarks used for the study. More information can be found in appendixes B and C.

4.1 Simulator

We use *FLEXUS* [33] to provide accurate simulations of chip multiprocessors and symmetric multiprocessors running unmodified commercial workloads. *FLEXUS* is a cycle-accurate full-system simulator that simulates both user-level and operating system code. We use the *SimFlex* statistical sampling methodology [33]. *FLEXUS* extends the Virtutech Simics functional simulator [24] with models of processor tiles with cores, NUCA cache, on-chip coherence protocol controllers, and on-chip interconnect.

We launch measurements from checkpoints with warmed caches and branch predictors, then run for 100,000 cycles to warm issue queue and interconnect state prior to collecting measurements of 50,000 cycles. We use the aggregate number of user instructions committed per cycle (i.e., committed user instructions summed over all cores divided by total elapsed cycles) as our performance metric, which is proportional to overall system throughput [33].

We estimate cache access latencies using Cacti 6.5 [34, 25]. Cacti is an integrated cache access time, cycle time, area, leakage, and dynamic power model. By integrating all these models together, cache trade-offs are all based on the same assumptions and, hence, are mutually consistent. Besides, the latencies provided by Cacti are typically lower than the ones achieved in commercial products, thus our assumptions are conservative.

4.2 Design

For our experiments, *FLEXUS* models the system summarized in table 4.1. Although *FLEXUS* models the UltraSPARC III ISA, the parameters are adjust to make the processor as similar as possible to the ARM Cortex A9 [2].

Table 4.1 collects the parameters that do not change among designs. As we explained before, we study two different designs, where the main difference is the organization of the LLC (L2) cache. For the Shared model (Baseline design) we assume a 16-nodes system where each core

has a slice of a shared L2 cache (tiled architecture). For the Cluster design we group 16 cores in groups of 4. Each cluster shares the L2 cache, and these L2 are independent between the clusters. We explore L2 cache sizes from 128KB to 1MB per core (i.e., from 4MB to 16MB in total per chip), which corresponds, from our point of view, to acceptable cache sizes for the technology and type of processor studied (see Chapter 3 for more details, including assumptions about area and cache size).

Our on-chip coherence protocol is a four-state MESI protocol modeled after Piranha [5]. We simulate one memory controller per four cores, each controller co-located with one tile, assuming communication with off-chip memory through flip-chip technology. Tiles communicate through the on-chip network.

CMP Parameters	
CMP Size	16-core // 4x4-core clusters
Processing Cores	UltraSPARC III ISA; 2GHz, OoO cores 8-stage pipeline, 2-wide dispatch/retirement 8-entry ROB and LSQ, 4-entry store buffer
Branch Predictor	8K GShare + 16K bi-modal + 16K selector 2K entry, 16-way, 2 branches per cycle
L1 Caches	Split I/D, 4-way 32KB 2-cycle load-to-use, 2 ports, LRU replacement 64-byte blocks, 32 MSHRs, 8-entry victim cache
L2 Cache	Variable size among desings 16-way, NUCA 64-byte blocks, 32 MSHRs, 16-entry victim cache
Main Memory	3GB, 8KB pages, 75 cycles access time
Memory Controllers	One per 4 cores, round robin page interleaving
Interconnect	Mesh (4x4 for Baseline, 2x2 for Cluster) 8-byte links, 3-cycles link latency

Table 4.1: System parameters.

4.3 Workloads

We simulate systems running the Solaris 8 operating system and executing the workloads listed in table 4.2. We focus on commercial workloads including a wide range of server workloads from competing vendors, including online transaction processing, decision support systems, and web server benchmarks [4].

The following list comprises the three commercial workload categories used in this project:

- **Web server workloads:** SPECWeb99 [29] over Apache and Zeus in this study. These workloads can be characterized for high levels of thread parallelism (TLP), high compulsory cache miss rates that regularly stall the machine due to recently DMAed data to the memory, and a large percentage of time in kernel code due to interrupt handling, packet transmission, and network stack processing. [20]

- **Online Transaction Processing:** OTLP applications have one of the largest shares in the server market. They are used in day-to-day business operations (e.g., airline reservations) and are characterized by a large number of clients who continually access and update small portions of the database through short running transactions. The OTLP workloads used in this project are TPC-C [31] over Oracle and DB2, which both rely on a database system to perform many short transactions.

- **Decision Support System:** DSS systems are primarily used for business analysis purposes, whereby information from the OLTP side of a business is periodically fed into the DSS database and analyzed. In contrast to OLTP, DSS is characterized by long running queries that are primarily read-only and may each span a large fraction of the database. In this project, DSS workloads are based on the TPC-H Benchmark [32]. They consist of queries and concurrent data modifications to large databases in order to give answers to critical business questions, examining large volumes of data and executing queries of higher complexity than most OTLP transactions. We select scan-dominated queries (query 1), join-dominated (query 2) and queries with mixed behavior (query 17).

Web Server (SPECWeb99)	
Apache	Apache HTTP Server v2.0. 16K connections, fastCGI, worker threading model
Zeus	16K connections, fastCGI
OLTP - Online Transaction Processing (TPC-C v3.0)	
DB2	IBM DB2 v8 ESE. 100 warehouses (10 GB), 64 clients, 2 GB buffer pool
Oracle	Oracle 10g Enterprise Database Server. 100 warehouses (10 GB), 16 clients, 1.4 GB SGA
DSS - Decision Support Systems (TPC-H)	
Queries 1, 2, 17	IBM DB2 v8 ESE, 480 MB buffer pool, 1 GB database

Table 4.2: Application parameters.

Chapter 5
Main results

This chapter presents the main results obtained in this project. Appendix C collects all the results and extends them, including more detailed explanations about the benchmarks and the results, as well as information about the performance metrics used.

5.1 Results overview

We simulate the workloads collected in table 4.2 and report the aggregate number of user instructions committed per cycle (i.e., committed user instructions summed over all cores divided by total elapsed cycles) as our performance metric, which is proportional to overall system throughput [33]. We explore the two designs presented in Chapter 3 and we vary the cache size from 128KB to 1MB per core (from 4MB to 16MB of total cache size on-chip).

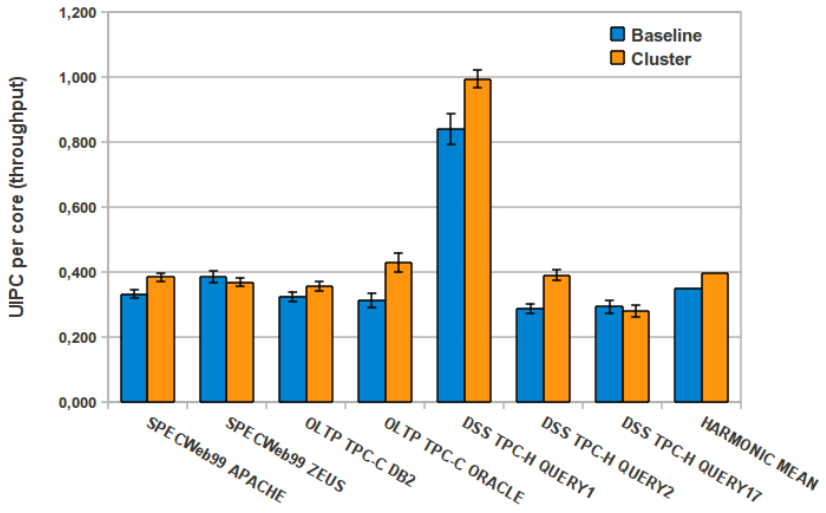


Figure 5.1: Best UIPC per core for server workloads.

Figure 5.1 shows the best result for each workload for Shared and Cluster designs and the harmonic mean. The LLC sizes that provides the best performance are collected in table 5.1. In all the cases, except for SPECWeb99 over Zeus and DSS TPC-H Query 17, the Cluster design works better than the Shared design, and so does it in average. The results in general follow this tendency for all the benchmarks and cache sizes.

Workload	Best cache size	
	Shared design	Cluster design
SPECWeb99 APACHE	16 MB	4 MB
SPECWeb99 ZEUS	16 MB	4 MB
OLTP TPC-C DB2	16 MB	4 MB
OLTP TPC-C ORACLE	16 MB	4 MB
DSS TPC-H QUERY 1	8 MB	512 KB
DSS TPC-H QUERY 2	8 MB	4 MB
DSS TPC-H QUERY 17	4 MB	2 MB

Table 5.1: Best cache sizes for the different workloads/designs. The cache size corresponds to the cache size that is visible for each processors (i.e., in a Cluster design a cache size of 4MB corresponds to a total cache size of 16MB on-chip).

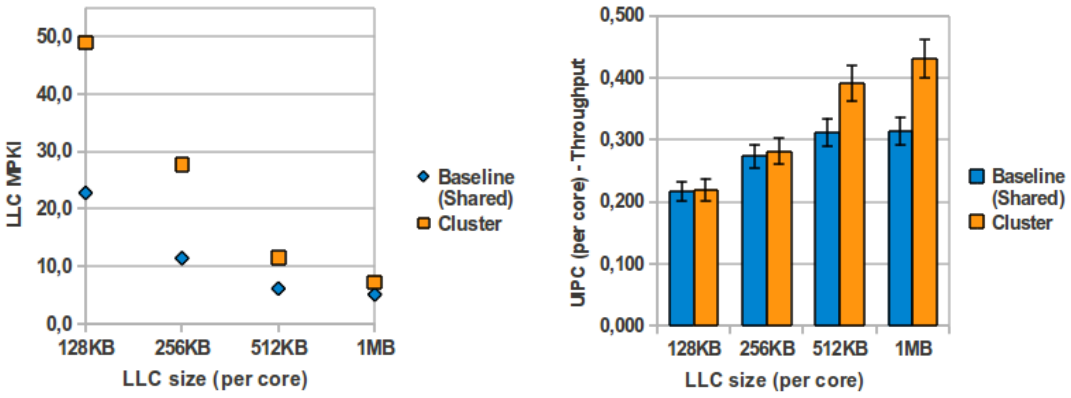
5.2 Representative results

In this section we explore some representative results for each workload, specifically OLTP TPC-C Oracle, Web server Zeus, and DSS TPC-H Query 17. OLTP TPC-C Oracle is a representative benchmark of the set, as well as it corroborates our hypothesis. Web server Zeus and DSS TPC-H Query 17 are the only two workloads for that our Cluster design is not better than a shared last level cache. In the remaining of the section we will explain why.

5.2.1 OLTP TPC-C: Oracle

Our OLTP benchmarks include TPC-C over DB2 and Oracle. In this section we focus on OLTP TPC-C over Oracle as a clear example of benchmark that corroborates our hypothesis.

Ailamaki *et al.* [17] pointed out in their study about databases workloads the requirements of this kind of applications. As long as the primary working set fits in the LLC, the performance increases. From there, larger caches may degrade performance. We can see this tendency in figure 5.2 (UIPC per core).



(a) LLC MPKI for OLTP Oracle.

(b) UIPC for OLTP Oracle.

Figure 5.2: LLC MPKI and UIPC for OLTP Oracle.

It is important to understand the difference between our Baseline and Cluster designs. Although both designs have the same amount of LLC on-chip, the amount of cache each core can access is different (i.e., considering 1MB per core of LLC, in the case of Shared (Baseline) design each core sees the complete LLC – 16MB; in the case of the Cluster design each core sees 4MB of LLC, as this level of cache is private for each cluster).

The benchmark prefers a Shared design with small last level caches. With the sufficient size of LLC (observing figure 5.2b, 512KB per core), the Cluster design obtains a better performance. In fact, we can see how Oracle presents a difference of 0,1 UIPC (per core) when we consider 1MB LLC per core.

LLC MPKI (see figure 5.2a) shows the same tendency. We see a drastic decrease in the number of misses per 1000 instructions when we double the cache from 128KB to 256KB per core (Cluster), but almost no variation from 512KB to 1MB of cache per core (Shared and Cluster).

Exemplifying all the mentioned above, figure 5.3 shows the execution time (user) breakdown for Oracle (Cluster design). We can see that a private cache of 512KB is not able to capture the primary working set and the amount of time spent off-chip is huge. With 1MB of last level cache per core the time spent off-chip is reduced from 50% (128KB per core) to less than 10% of the total execution time (user). That decrease translates into twice the performance (figure 5.2b).

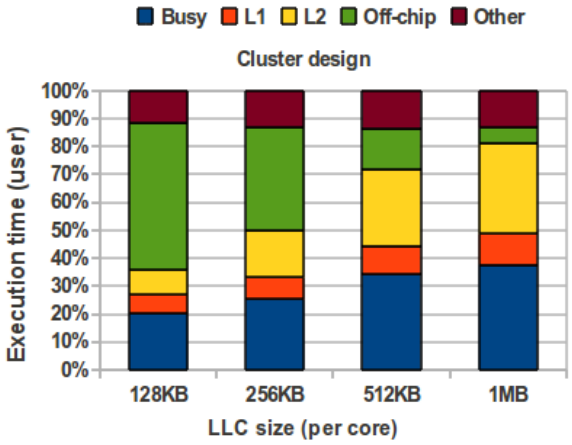


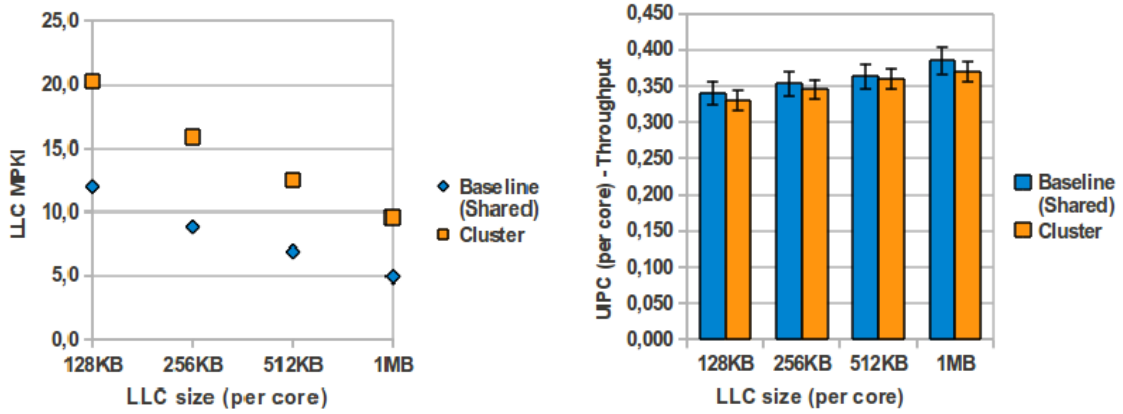
Figure 5.3: Execution time breakdown (user) for OLTP Oracle (Cluster).

5.2.2 Web server: Zeus

Our Web server benchmarks category includes SPECWeb99 over Apache and Zeus. Both present a similar tendency regarding throughput (UIPC increases as the last level cache size does), yet Apache prefers a Cluster design for all the configurations explored and Zeus gets benefit from a Shared design. Figure 5.4b shows the UIPC (throughput) for Zeus. We also see that the difference between designs is very small.

The LLC MPKI in figure 5.4a shows a decrease in the LLC MPKI with larger caches. Observing both graphs together (figure 5.4), we can conclude that the reduction in the miss rate

compensates the higher latency of a large cache. These results indicate that Web server benchmarks benefit from large last level caches. Nevertheless, Zeus seems to get advantage of sharing data, and that size is more critical than latency, which is why the Shared design performs better. For both benchmarks the best performance corresponds to the largest LLC (1MB per core).



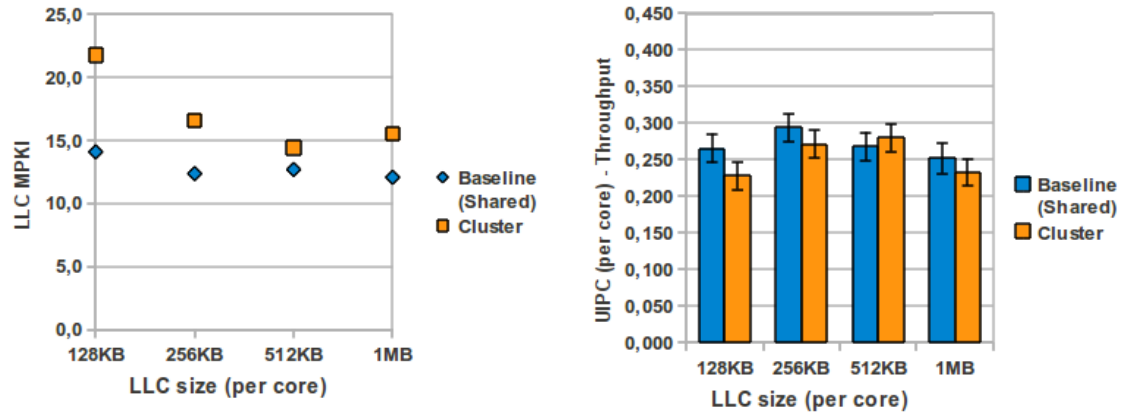
(a) LLC MPKI for Web Zeus.

(b) UIPC for Web Zeus.

Figure 5.4: LLC MPKI and UIPC for Web Zeus.

5.2.3 DSS TPC-H: Query 17

Our DSS benchmarks are TPC-H queries 1, 2 and 17 over DB2. These benchmarks correspond to three categories: scan-based (query 1), join-dominated (query 2), and mixed-behavior (query 17). In this section we present results for query 17 (shown in figure 5.5: figure 5.5b shows throughput and figure 5.5a shows the LLC misses per 1000 instructions).



(a) LLC MPKI for DSS Query 17.

(b) UIPC for DSS Query 17.

Figure 5.5: LLC MPKI and UIPC for DSS Query 17.

For this query, a shared last level cache is preferable in all cases except for a size of 512KB per core. If we observe the UIPC graph (figure 5.5b), we see how our Shared design presents an

increase in performance when we move from 128KB to 256KB of LLC per core, yet the performance decreases if we continue adding last level cache. For this design, 128KB of LLC per core provides better performance than 1MB per core. The Cluster design presents the same tendency but the decrease comes when we move from 512KB to 1MB.

Both UIPC and LLC MPKI graphs suggest that this query has a low locality and, therefore, large caches decrease performance (higher access latencies). The better performance of the Shared design also suggests that this query benefits from sharing data between processors.

The execution time breakdown (user) in figure 5.6 (Shared) shows that the amount of time we spend off-chip is around 30%.

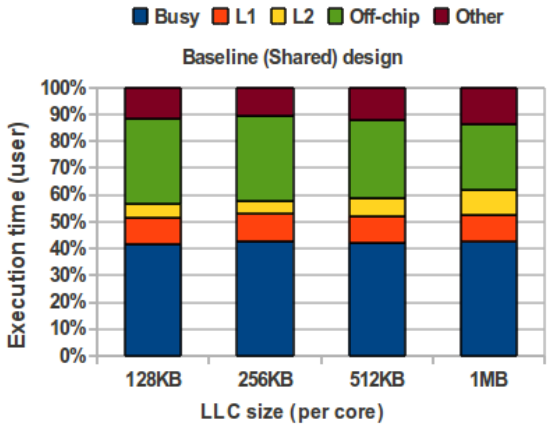


Figure 5.6: Execution time breakdown (user) for DSS Query 17 (Baseline).

For queries 1 and 2, Cluster outperforms Shared design (figure 5.1). In these cases, increasing the cache size does not translate in better performance. In the case of query 1, the first level cache is able to capture most of the accesses, so performance does not vary. Query 2 does not get benefit of larger caches, and even the performance is degraded (see appendix C for more details).

5.3 Summary

We have seen how different types of workloads present different characteristics concerning sharing, last level cache size, and organization. In general, a private organization is preferable for most of the benchmarks and in average.

Web server benchmarks present an increase in performance with larger last level caches. OLTP benchmarks have minimum requirements of LLC. Once these requirements are satisfied, the design which allows faster access (in our case, the Cluster design) performs better. DSS benchmarks have different behavior depending on the category. Our scan-based query's (query 1) performance does not change when increasing size, as the first level cache is able to capture most of the accesses. The join-dominated query (query 2) benefits from private designs. Finally, the query with mixed-behavior (query 17) has low locality, which translates into better performance with smaller last level caches (lower access time).

Chapter 6
Conclusion and future work

Memory hierarchy is one of the performance factors in chip multiprocessors as well as one of the most power-consuming components. Therefore, the organization of the memory hierarchy and especially the cache hierarchy on-chip plays a fundamental role in the CMP performance.

We proposed two organizations for the last level cache of the EuroCloud ARM's chip: a Shared design, where we pretend to make the most of the cache capacity, and a Cluster design, where we prioritize allocating data near the requester core for achieving low access latency.

We modelled both designs in *FLEXUS* [33], our simulation environment. We simulated representative commercial workloads from three categories: web, online transaction processing and decision support systems. From the results obtained we can conclude that our Cluster design worked better in average, corroborating our initial hypothesis: our Cluster design (in general) works better for large last level caches; our Shared design performs better if the benchmark presents low locality or if the last level cache is small.

In great detail, different kinds of workloads throw different conclusions:

- Web workloads present a better performance with larger last level caches.
- OLTP workloads have minimum requirements of last level cache size. Above this size, private organizations and their lower latencies provide better performance.
- DSS workloads have different characteristics depending on the category. In general the Cluster design outperforms the Shared design and the requirements of last level cache size are small.

We corroborated the conclusions in [15] about minimum requirements in database workloads, as we observed that this kind of workloads need a minimum LLC capacity to perform. From a determined LLC size, increasing the capacity of the cache leads to similar results or even a decrease in the performance (due to the higher latency of the cache).

Private caches offer other advantages over shared designs that should be also consider. Besides the lower latency, the on-chip bandwidth requirements are smaller, and the design is more scalable. Private caches are more self-contained and it is easier to implement priority mechanisms and quality of service (QoS) [37, 19].

Although we initially proposed two alternatives for the Cluster design (coherent and non-coherent caches), we just explored the non-coherent approach. We think that LLC (L2) hardware

coherence mechanisms in a tiled CMP running server workloads should be avoided. In addition to a more complicated hardware design, moving blocks on-chip due to the coherence mechanism penalizes the average access latency to the cache.

Nevertheless, private caches present a fundamental drawback compared to shared LLC designs, which is the larger number of off-chip requests. As part of the EuroCloud project, the impact of 3D DRAM integration on-chip will be explored. This 3D integration will allow lower latencies to the memory as it will allocate a portion of several MBs of DRAM near the processor cores. We believe that a minimum amount of LLC should be maintained on-chip, as the latency of the 3D stacked DRAM is a way too high for a second level cache. We let further study on this topic for Future Work.

We also think that future studies should include power consumption results for the CMP, as this is one of the main motivations for the EuroCloud project. We believe that private caches (our Cluster design) will present less consumption than our Shared design. We could also explore the possibility of switch off parts of the memory hierarchy to save energy, and the impact in performance. Hybrid designs seem a good way to make the most of private and shared designs, but we should consider whether, from a power perspective, these designs are a good alternative for embedded-based server-on-chip.

Finally, we should consider that in addition to the workloads studied, Cloud Computing will include applications such as video/music streaming, song recognition, or data mining. Each kind of application presents unique characteristics and behavior, so benchmarks that include these kinds of applications are likely to be implemented and included in the study of Cloud computing servers.

Bibliography

[1] Alaa R. Alameldeen and David A. Wood. Variability in architectural simulations of multi-threaded workloads. In *HPCA '03: Proceedings of the 9th International Symposium on High-Performance Computer Architecture*, page 7, Washington, DC, USA, 2003. IEEE Computer Society.

[2] ARM. The ARM Cortex-A9 Processors, 2009. `www.arm.com/files/pdf/ARMCortexA-9Processors.pdf` (last access October 2010).

[3] M. Azimi, N. Cherukuri, D.N. Jayasimha, A. Kumar, P. Kundu, S. Park, I. Schoinas, and A. Vaidya. Integration Challenges and Tradeoffs for Tera-scale Architectures. *Intel Technology Journal*, 11(3):173–184, August 2007.

[4] Luiz André Barroso, Kourosh Gharachorloo, and Edouard Bugnion. Memory system characterization of commercial workloads. *SIGARCH Comput. Archit. News*, 26(3):3–14, 1998.

[5] Luiz André Barroso, Kourosh Gharachorloo, Robert McNamara, Andreas Nowatzyk, Shaz Qadeer, Barton Sano, Scott Smith, Robert Stets, and Ben Verghese. Piranha: a scalable architecture based on single-chip multiprocessing. In *ISCA '00: Proceedings of the 27th annual international symposium on Computer architecture*, pages 282–293, New York, NY, USA, 2000. ACM.

[6] Bradford M. Beckmann, Michael R. Marty, and David A. Wood. ASR: Adaptive Selective Replication for CMP Caches. In *MICRO 39: Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 443–454, Washington, DC, USA, 2006. IEEE Computer Society.

[7] K.G. Brill. The Invisible Crisis in the Data Center: The Economic Meltdown of Moore's Law. White Paper, 2007. Uptime Institute.

[8] David Chaiken, Craig Fields, Kiyoshi Kurihara, and Anant Agarwal. Directory-Based Cache Coherence in Large-Scale Multiprocessors. *Computer*, 23(6):49–58, 1990.

[9] Jichuan Chang and Gurindar S. Sohi. Cooperative Caching for Chip Multiprocessors. In *ISCA '06: Proceedings of the 33rd annual international symposium on Computer Architecture*, pages 264–276, Washington, DC, USA, 2006. IEEE Computer Society.

[10] Zeshan Chishti, Michael D. Powell, and T. N. Vijaykumar. Optimizing Replication, Communication, and Capacity Allocation in CMPs. In *ISCA '05: Proceedings of the 32nd annual international symposium on Computer Architecture*, pages 357–368, Washington, DC, USA, 2005. IEEE Computer Society.

[11] William Dally and Brian Towles. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003.

[12] John D. Davis, James Laudon, and Kunle Olukotun. Maximizing CMP Throughput with Mediocre Cores. In *PACT '05: Proceedings of the 14th International Conference on Parallel Architectures and Compilation Techniques*, pages 51–62, Washington, DC, USA, 2005. IEEE Computer Society.

[13] Xiaobo Fan, Wolf-Dietrich Weber, and Luiz Andre Barroso. Power Provisioning for a Warehouse-sized Computer. In *ISCA '07: Proceedings of the 34th annual international symposium on Computer architecture*, pages 13–23, New York, NY, USA, 2007. ACM.

[14] Richard A. Hankins, Trung Diep, Murali Annavaram, Brian Hirano, Harald Eri, Hubert Nueckel, and John P. Shen. Scaling and characterizing database workloads: Bridging the gap between research and practice. In *MICRO 36: Proceedings of the 36th annual IEEE/ACM International Symposium on Microarchitecture*, page 151, Washington, DC, USA, 2003. IEEE Computer Society.

[15] Nikolaos Hardavellas. *Chip Multiprocessors for Server Workloads.* PhD thesis, School of Computer Science, Carnegie Mellon University, July 2009.

[16] Nikos Hardavellas, Michael Ferdman, Babak Falsafi, and Anastasia Ailamaki. Reactive NUCA: near-optimal block placement and replication in distributed caches. In *ISCA '09: Proceedings of the 36th annual international symposium on Computer architecture*, pages 184–195, New York, NY, USA, 2009. ACM.

[17] Nikos Hardavellas, Ippokratis Pandis, Ryan Johnson, Naju Mancheril, Anastassia Ailamaki, and Babak Falsafi. Database Servers on Chip Multiprocessors: Limitations and Opportunities. In *Proceedings of the Biennial Conference on Innovative Data Systems Research*, 2007.

[18] Ron Ho, Kenneth W. Mai, Student Member, and Mark A. Horowitz. The future of wires. In *Proceedings of the IEEE*, pages 490–504, 2001.

[19] Ravi Iyer. CQoS: a framework for enabling QoS in shared caches of CMP platforms. In *ICS '04: Proceedings of the 18th annual international conference on Supercomputing*, pages 257–266, New York, NY, USA, 2004. ACM.

[20] Taeho Kgil, Shaun D'Souza, Ali Saidi, Nathan Binkert, Ronald Dreslinski, Trevor Mudge, Steven Reinhardt, and Krisztian Flautner. PicoServer: using 3D stacking technology to enable a compact energy efficient chip multiprocessor. *SIGARCH Comput. Archit. News*, 34(5):117–128, 2006.

[21] Changkyu Kim, Doug Burger, and Stephen W. Keckler. An adaptive, non-uniform cache structure for wire-delay dominated on-chip caches. In *ASPLOS-X: Proceedings of the 10th international conference on Architectural support for programming languages and operating systems*, pages 211–222, New York, NY, USA, 2002. ACM.

[22] Poonacha Kongetira, Kathirgamar Aingaran, and Kunle Olukotun. Niagara: A 32-way multithreaded sparc processor. *IEEE Micro*, 25(2):21–29, 2005.

[23] Gabriel H. Loh. 3D-Stacked Memory Architectures for Multi-core Processors. In *ISCA '08: Proceedings of the 35th Annual International Symposium on Computer Architecture*, pages 453–464, Washington, DC, USA, 2008. IEEE Computer Society.

[24] Peter S. Magnusson, Magnus Christensson, Jesper Eskilson, Daniel Forsgren, Gustav Hallberg, Johan Hogberg, Fredrik Larsson, Andreas Moestedt, and Bengt Werner. Simics: A Full System Simulation Platform. *Computer*, 35:50–58, 2002.

[25] Naveen Muralimanohar, Rajeev Balasubramonian, and Norm Jouppi. Optimizing NUCA Organizations and Wiring Alternatives for Large Caches with CACTI 6.0. In *MICRO 40: Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 3–14, Washington, DC, USA, 2007. IEEE Computer Society.

[26] Emre Ozer, Krisztian Flautner, Sachin Idgunji, Ali Saidi, Yiannakis Sazeides, Bushra Ahsan, Nikolas Ladas, Chrysostomos Nicopoulos, Isidoros Sideris, Babak Falsafi, Almutaz Adileh, Michael Ferdman, Pejman Lotfi-Kamran, Mika Kuulusa, Pol Marchal, and Nikolas Minas. EuroCloud: Energy-conscious 3D Server-on-Chip for Green Cloud Services. In *2nd Workshop on Architectural Concerns in Large Datacenters in conjunction with ISCA-2010*, June 2010.

[27] Brian M. Rogers, Anil Krishna, Gordon B. Bell, Ken Vu, Xiaowei Jiang, and Yan Solihin. Scaling the bandwidth wall: challenges in and avenues for CMP scaling. In *ISCA '09: Proceedings of the 36th annual international symposium on Computer architecture*, pages 371–382, New York, NY, USA, 2009. ACM.

[28] S. Rusu, S. Tam, H. Muljono, D. Ayers, and J. Chang. A Dual-Core Multi-Threaded Xeon Processor with 16MB L3 Cache. In *IEEE International Solid-State Circuits Conference Digest of Technical Papers*, pages 315 –324, feb. 2006.

[29] Standard Performance Evaluation Corporation (SPEC). Specweb99 benchmark, 2000. `http://www.spec.org/web99/` (last access October 2010).

[30] Per Stenstrom. A Survey of Cache Coherence Schemes for Multiprocessors. *Computer*, 23(6):12–24, 1990.

[31] Transaction Processing Performance Council (TPC). TPC Benchmark C Standard Specification, 2010. `http://www.tpc.org/tpcc/` (last access October 2010).

[32] Transaction Processing Performance Council (TPC). TPC Benchmark H (Decision Support) Standard Specification, 2010. `http://www.tpc.org/tpch/` (last access October 2010).

[33] Thomas F. Wenisch, Roland E. Wunderlich, Michael Ferdman, Anastassia Ailamaki, Babak Falsafi, and James C. Hoe. SimFlex: Statistical Sampling of Computer System Simulation. *IEEE Micro*, 26(4):18–31, 2006.

[34] Steven J. E. Wilton and Norman P. Jouppi. CACTI: An Enhanced Cache Access and Cycle Time Model. *IEEE Journal of Solid-State Circuits*, 31:677–688, 1996.

[35] Roland E. Wunderlich, Thomas F. Wenisch, Babak Falsafi, and James C. Hoe. SMARTS: accelerating microarchitecture simulation via rigorous statistical sampling. *SIGARCH Comput. Archit. News*, 31(2):84–97, 2003.

[36] J. Wuu, D. Weiss, C. Morganti, and M. Dreesen. The Asynchronous 24MB on-chip Level-3 cache for a dual-core Itanium®-Family Processor. In *Solid-State Circuits Conference, 2005. Digest of Technical Papers. ISSCC. 2005 IEEE International*, pages 488 –612 Vol. 1, feb. 2005.

[37] Michael Zhang and Krste Asanovic. Victim Replication: Maximizing Capacity while Hiding Wire Delay in Tiled Chip Multiprocessors. In *ISCA '05: Proceedings of the 32nd annual international symposium on Computer Architecture*, pages 336–345, Washington, DC, USA, 2005. IEEE Computer Society.