Performance and Energy Trade-offs Analysis of L2 on-Chip Cache Architectures for Embedded MPSoCs

Mohamed M. Sabry[†], Martino Ruggiero[†‡] and Pablo G. Del Valle[†§]
[†] Embedded Systems Laboratory, EPFL, EPFL-STI-IEL-ESL, 1015 Lausanne, Switzerland
[‡] University of Bologna, DEIS, Viale Risorgimento, 2 - Bologna 40136, Italy
[§] DACYA, UCM, Avda. Complutense s/n - 28040 Madrid, Spain
mohamed.sabry@epfl.ch, martino.ruggiero@unibo.it, pgarciav@fdi.ucm.es

ABSTRACT

On-chip memory organization is one of the most important aspects that can influence the overall system behavior in multi-processor systems. Following the trend set by high-performance processors, high-end embedded cores are moving from single-level on chip caches to a two-level on-chip cache hierarchy. Whereas in the embedded world there is general consensus on L1 private caches, for L2 there is still not a dominant architectural paradigm. Cache architectures that work for high performance computers turn out to be inefficient for embedded systems (mainly due to power-efficiency issues). This paper presents a virtual platform for design space exploration of L2 cache architectures in low-power Multi-Processor-Systems-on-Chip (MPSoCs). The tool contains several L2 caches templates, and new architectures can be easily added using our flexible plug-in system. Given a set of constrains for a specific system (power, area, performance), our tool will perform extensive exploration to find the cache organization that best suits our needs. Through some practical experiments, we show how it is possible to select the optimal L2 cache, and how this kind of tool can help designers avoid some common misconceptions. Benchmarking results in the experiments section will show that for a case study with multiple processors running communicating tasks allocated on different cores, the private L2 cache organization still performs better than the shared one.

Categories and Subject Descriptors: B.3 Memory Structure: Performance Analysis and Design Aids.

General Terms: Design, Performance.

Keywords: Virtual platform, Multi-core,L2 cache.

1. INTRODUCTION AND RELATED WORK

On-chip memory organization is one of the most important aspects that can influence the overall system behavior in MPSoCs. The memory system must provide the required bandwidth to the software tasks, complying with tight constraints in terms of chip area, power consumption and cost. High-end embedded cores are moving from single-level on chip caches to a two-level on-chip cache hierarchy. Several commercial solutions matching this approach are already available on the market. MPCore [1] and Cortex-A9 [2] from ARM, CELL BE STI [3], or MIPS32 1004K [4] are just a few representative examples.

While there is general consensus on L1 private cache organization, for L2 there is still not a dominant paradigm unlike for the high-performance general-purpose processors. [5] clearly proves that trade-offs analysis of L2 on-chip cache architectures for embedded MPSoCs is a hot topic in the computer architecture community. In this paper, the authors perform a complete study to try to determine the best L2 cache architecture in a multi-core system. However, they use a tool (i.e. Heptane), that simulates only single-core processors, and at a very high level of abstraction (i.e. given the C application, it gives you I-cache misses). Later, they integrate the traces into a multi-core simulation framework. Although interesting, the accuracy of this simulator is very low.

Embedded MPSoC system architecture has always been quite different from the general-purpose one. It usually uses simpler processor micro-architectures, well-matched to the characteristics of the targeted applications.

Cost and time-to-market are important design constraints that dictate the use of conservative technologies and design methodologies (reusability and modularity), implying lower performance targets, instead of customized, high-performance, general-purpose cores. Another main difference between high-end embedded MPSoCs and high-performance general-purpose systems is the speed gap between processor and on-chip memory, in the embedded case, a smaller (but still significant) memory gap has non-negligible architectural implications.

Two other fundamental constraints for embedded designers are area bounds and limited energy budgets. The available on-chip area is always a limited resource that has to be shared by several IPs. As a consequence, dedicating a huge percentage of the chip to L2 cache, as commonly done for general purpose multicores, is not anymore profitable. Application specific accelerators and peripheral blocks, can be included instead, which provide superior power-performance efficiency.

Developing the optimal L2 on-chip cache system for a MPSoC under all these constraints is not trivial, and requires the use of advanced tools. A large number of architectural-level multiprocessor simulators have been developed by the computer architecture community. The SimpleScalar tool is one of them [6]. Using SimpleScalar, users can simulate real programs running on a range of modern processors and systems. Other virtual modeling environments are M5 [7] and

GLSVLSI'10, May 16–18, 2010, Providence, Rhode Island, USA.

UNISIM [16]. Also, companies showed interest in such field: Simics [14] and VaST [15] tools are just a few representative examples. These simulators are very accurate in modeling the processor core insights, but use high level models for describing the other IPs in the system, like communication interconnections, cache controllers or peripherals. Moreover, they do not provide any statistics on power consumption and area occupancy.

The literature also contains several works about architectural and memory hierarchy explorations of L2 cache for MPSoCs. Some authors propose innovative solutions, such as connecting a shared L2 cache to all the cores using a matrix crossbar interconnect (see [22]), or using cooperative caching and custom NoCs, as explained in [17] and [8, 9]. In [18] and [21], authors compare the performance of shared and separate L2 caches. However, the estimation of power consumption and size cost of the proposal is missing. In addition to this, existing simulation environments that perform whole MPSoCs modeling, normally do it at a high-level of abstraction [10] which is not accurate enough to allow detailed thermal studies. Conversely, MPARM simulator is particularly effective, especially in analyzing the communication infrastructure among computing nodes. Finally, several authors have explored the way to optimize the utilization of MPSoC memory hierarchy by exploring software mapping techniques [11, 12, 13], which are complementary approaches to the ideas proposed in this paper.

This paper introduces two main contributions. First, we developed a set of parameterizable models for L2 caches. By specifying the different parameters, our generic and modular L2 cache architecture can be configured as private, shared or hybrid. Furthermore, new architectures (e.g.: coherence protocols, replacement policies) can be easily added using our flexible plug-in system.

The second main contribution is a detailed analysis of how micro-architectural differences in L2 cache architectures impact the overall system behavior, as well as power consumption and area occupancy. We model several L2 cache architecture solutions on a multi-processor system and test their behaviors running various parallel kernels. Our experimental results show that the selection of the optimal L2 cache organization is neither trivial, nor unique, and encourage designers to abandon common misconceptions.

This paper is organized as follows. In Section 2, the proposed virtual platform environment and the examined L2 cache architectures are described and elaborated. Section 3 presents and analyzes the experimental results of our design space exploration. Finally, Section 4 presents conclusions and future work.

2. VIRTUAL PLATFORM ENVIRONMENT

2.1 L2 Cache Architecture Templates

As described in the introduction, in order to help system designers to model and easily explore several L2 cache architectures and variations, we have developed and implemented a flexible L2 cache architecture system. The proposed templates, written in SystemC, can be used either in stand-alone mode or plugged into any virtual platform tool. For completeness and demonstration purposes, we integrated them in an accurate virtual platform environment specifically designed for Embedded MPSoC design space explorations [19]. Similar tools can be found in literature [6, 7], but they model the interaction between MPSoC components at a very high level of abstraction. Our enhanced virtual platform is highly modular and capable of simulating at cycle-accurate level an entire MPSoC, including cores, L1 and L2 caches, L3 memories and system buses. Particular attention has been paid to communication bus modeling. System bus performance is indeed very critical for MPSoCs, since it can limit the overall system performance much more than CPU or memory. Moreover, area and power models for on-chip memories [23] are fully integrated in the platform, conforming an all-in-one tool for MPSoC development.

Our versatile L2 cache module can be reached by cores through a configurable number of ports, and it will request accesses to external memory through a configurable number of bus master ports. All the sub-modules in the L2 cache module are arbitrated and managed by the cache controller logic. It supports simultaneous accesses to internal memory from several CPUs and, in case of not compatible accesses, they are granted in a first-come-first-served fashion. A lot of effort has been paid to modeling such kind of architecture, particularly in the handshaking protocols among sub-modules.

Next figures represent the most representative system architectures that we have modeled: three different L2 cache templates (namely private, shared, or hybrid). In the pictures, the storage modules are colored in gray. Light gray, for these that contain only private data, and dark gray when they include also shared data.

Figure. 1 shows the system with private L2 caches. These caches do not contain any shared data (i.e. the shared data is not cacheable). These caches are identical, i.e. they have the same access time, power consumption, capacity, and number of ways. They are configured with only one port to CPU and one port to system bus.

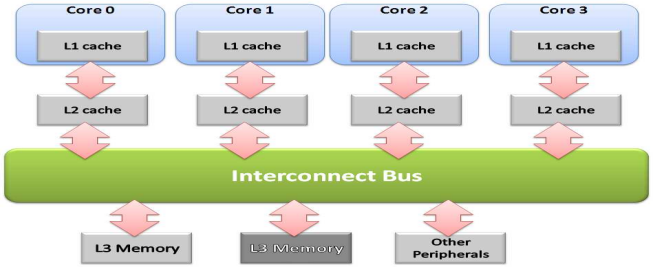


Figure 1: System architecture with separate equivalent L2 caches.

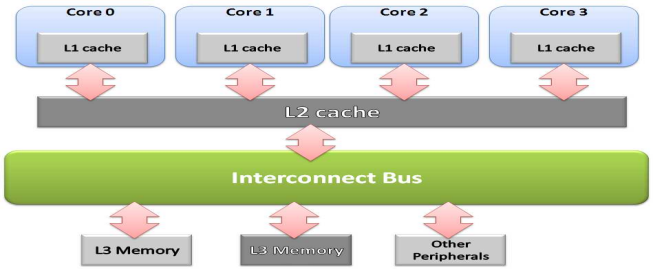


Figure 2: System architecture with multi-port shared L2 cache.

Architectures in Figure 2 and Figure 3 use a unified shared

L2 cache, but with two different configurations. Figure 2 shows a multi-port shared cache where all the cores can read data from the cache simultaneously, while in Figure 3 there is a single-port cache where all the cores get the access to such cache using a multiplexer and a round-robin based arbiter that grants such ports to each core. These caches contain both private and shared data.

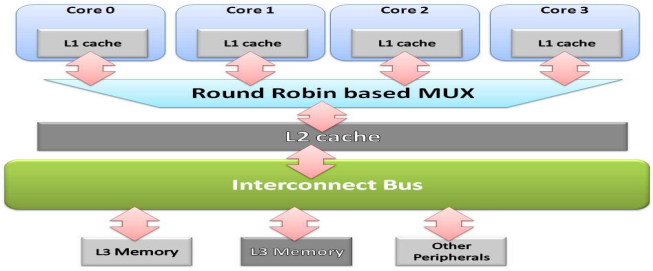


Figure 3: System architecture with single-port shared L2 cache.

Another methodology to make the shared data cacheable is by using the hybrid architecture shown in Figure 4. This cache is composed of several separated private caches that store only private data, and a unified small shared cache that is used to store only shared data. This small shared cache could be either multi-port or single-port cache.

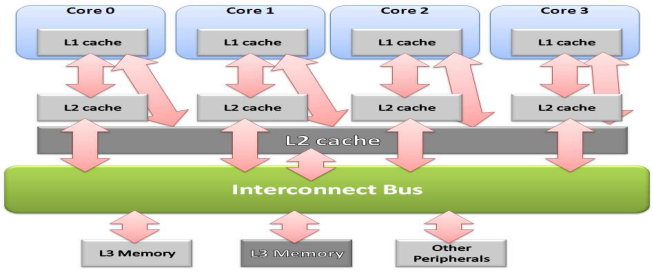


Figure 4: System architecture with hybrid L2 cache, several separate L2 cache with a small shared L2 cache.

2.2 Simulator integration

Our simulation environment is based on the MPARM simulator [19]. Due to the flexibility and accuracy of the MPARM simulator for MPSoC modeling, we decided it was the perfect candidate to incorporate our L2 cache model.

MPARM is a virtual SoC platform, written almost entirely in SystemC, that accurately models both HW and SW parts of a system. A typical MPSoC platform instantiated in such simulator consists of a configurable number of 32-bit ARM processors, with their L1 Data and Instruction caches, L3 memories and the system bus. The L3 memories are virtually divided into private, which contain data strictly related to only one core, and shared memories, containing shared data between different cores. The cores are connected to the L3 memories through an on-chip interconnection bus. Several bus protocols are available (such as AMBA AHB, AMBA AXI, STBus), as well as several bus topologies (shared bus, segmented bus, partial- and full-crossbar).

Regarding the software running on the simulated hardware platform, it is possible to run stand alone (i.e. without the support of an OS), RTEMS OS-based, or uClinux OS-based applications.

In addition to including support for modeling different L2 cache architectures, we integrated an accurate profiling tool that calculates power consumption and area occupancy of memories.

2.3 Support for Power and Area Modeling

When coming to accurately calculate area and power numbers of the memory sub-system, CACTI 6.0 [23] is one of the most used tools for such purposes. It is an integrated cache and memory area and power model. Moreover, it is written in C++ and open-source; thus, being the perfect candidate to embedded its functionalities into our virtual platform. Its main functions have been extracted and integrated into our environment.

Once specified the set of memory architectural parameters that define our experiment (like the size of the caches, the number of their ports, the manufacturing technology, the associativity number, the tag size), our tool will report the area occupied by the memories. At run time, simulation will produce information about every sub-module in L2 cache, such as activity in the input and output ports, control logic, internal interconnect and memory. These information is transparently (without any user participation) fed into the ported CACTI-functions at run-time to automatically calculate the dynamic energy and leakage power consumed by each memory element of the system.

3. DESIGN SPACE EXPLORATION

In section 2 we showed different system configurations using various L2 cache architectures. In this section, we show the effects of these cache configurations on area, speed, and energy of the whole system. The instantiated system contains 4 cores, with L1 caches separated into I-cache and D-cache, L2 cache(s), the interconnection bus, L3 memory, and peripherals for synchronization (i.e. test-and-set HW semaphores, interrupts support). The cache modules employ LRU replacement algorithm, with write back and write allocation techniques.

Several benchmarks from the MiBench suit [20] have been used to evaluate the system performance, with variations of the software environment from stand alone applications, to applications running under uClinux OS.

The workloads applied on various systems were classified into:

- **Multi-Processor Workload Scenario.** Resembles an heterogeneous system, where each core is performing independent tasks which do not require any resources from the other cores.
- **Parallel Workload Scenario.** The application execution is split between the different cores. In such workload, a portion of data is shared between the cores, and stored into shared memory. Each core performs its operations on the shared data and, then, signals the proceeding core to work on the same shared data.

There are two different simulation scenarios have been used in evaluating the cache performance.

- **Fixed Cache Storage Capacity.** This scenario considered a fixed storage capacity of the total L2 cache.

- **Fixed Total Die Area.** This scenario fixes the total die area for the entire chip. the whole L2 cache architectures are designed (including their storage area and internal interconnection) in order to fit in the remaining area of the chip hosting also the other IPs of the system.

Please consider that all graphs in next sections will show results in execution time and energy normalized to the separated architecture ones. It is also important to notice that the shared data is being allocated to a predefined address space that is different from the private data address space.

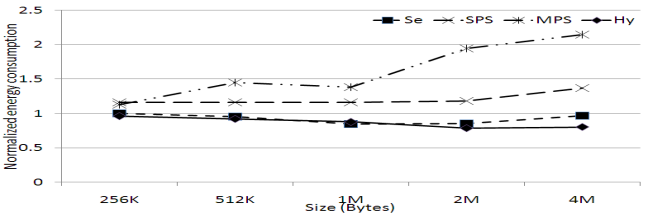
3.1 Fixed Cache Storage Capacity Scenario

We have evaluated the performance of different L2 cache architectures in a system with a total storage capacity of (S). We will vary this parameter, starting from 256KB to 4M. Each architecture is allocated an equivalent storage to S. S is used as a single unified storage and is divided into a number of equivalent separate portions for a system with the mentioned 4 cores. However, in the hybrid cache the shared portion has half of the capacity allocated, while each separate portions is given one quarter of the total allocated space. Each module in each cache architecture is 8-way set-associative. With 32 bits address space used in this system. Performance evaluation of various cache architectures has been done using both parallelized and multi-processor workload. Two parallelized benchmarks have been used: matrix multiplication and Jpeg benchmarks. The parallelized Jpeg decoding code consists of, a sequential part (entropy decoding) and in a parallel region (dequantization and IDCT).

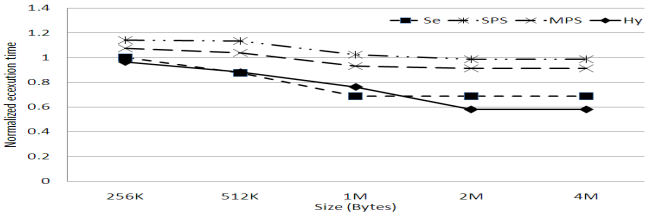
Please note that the following acronyms will be used to represent each of the cache architectures; Separate (Se), Single-port shared (SPS), Multi-port shared (MPS), and Hybrid (Hy). Sub-figures in Figure 5 show execution time, energy consumption, and miss rates of the system executing the previously mentioned benchmarks, having each of the different cache architectures.

Figures 5(a) and 5(b) show that the shared cache with its both configurations has the highest energy consumption and the slowest execution time. Having a larger storage capacity implies that shared caches suffer a larger energy consumption per access, regardless its division into a number of banks, because there will be an additional energy resulting from the interconnection from such banks. Assuming that the total number of access to the cache are equal regardless the architecture, the shared cache will experience higher energy consumption. Despite the availability of large storage capacity that could give an assumption of lower miss rates as shown in Figure 5(c), sharing such resource between multiple cores leads to longer execution time. The usage of write-allocate and write back techniques would lead to higher probability of sending an updated (dirty) data to L3 memory before filling in that dirty location in case of write miss, since any cache location is shared between the various cores. Thus, a relatively long period will be consumed in case of a write miss, in addition to the presence of higher write miss rates in shared caches as shown in Figure 5(d).

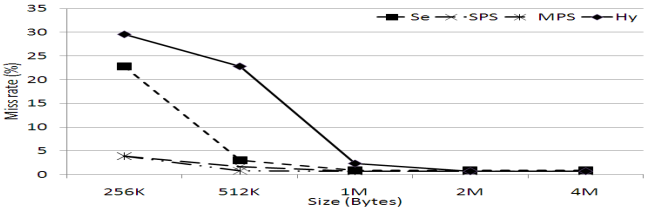
The single-port L2 cache experiences a longer execution time since there is only a single port to be used. Thus, the access to the cache will be round-robin based access. This type of access guarantees fair cache accessibility but, it leads to longer execution time, since the access to the cache from any core is arbitrary and is dependent on the used application. On the other hand, the multi-port shared cache has



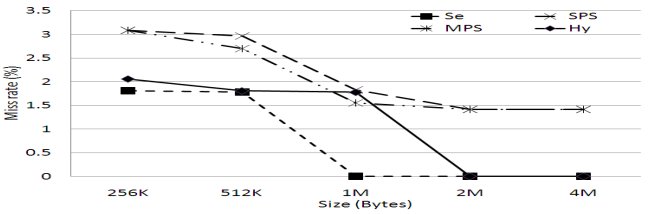
(a) Energy consumption



(b) Execution Time



(c) Read miss rate



(d) Write miss rate

Figure 5: Performance metrics of Jpeg benchmark execution on different cache architectures.

faster execution time than that of the single-port, since any core could perform a read from the cache, simultaneously. However, this enhancement came with the penalty of larger energy consumption since the multi-port shared cache experience larger energy consumption per access.

The only benefit of the shared cache over the private cache is the cacheability of the shared data without the need of a coherency control overhead. But, as it is shown from the graphs, this benefit only worked for the hybrid cache. This is because the amount of shared data is observed to be always a small portion compared to the private data. Thus, the benefit of the cacheable shared data is negated by the increased cache refills and replacements in the shared cache.

Simulation results show that, the hybrid cache based system achieved lower energy consumption and execution time of the separate cache based system in Jpeg benchmark for all storage capacities except for 1MB. These results occurred since hybrid L2 cache combines between the cacheability of shared data and smaller energy consumption per access. However, at 1MB capacity the private cache surpasses the hybrid cache performance. This is related to the ratio between the shared data and private data in such benchmark. While in lower capacities, both private and hybrid cache suffered high private

data miss rates but the hybrid cache had the availability of shared data cacheability, as shown in Figure 5(c). By increasing the capacity, the private data miss rate decreased in both cache sizes but with different rates since the equivalent private data capacity of the hybrid is half of the separate cache. At 1MB separate cache, the miss rate reached the minimum value and by that, the effect of private data cacheability reached its maximum that surpassed the shared data cacheability effect in the hybrid cache. Then, it was at 2MB capacity that the private portion of the hybrid cache reached its minimum miss rate. Thus, the shared data cacheability became into favor of the hybrid cache to obtain lower energy and execution time.

It is also worth to study the die area occupied by each cache configuration. Figure 6 shows the area consumed by each cache configuration. The area of the multi-port cache is the largest amongst the various architectures, since a multi-port cell occupies a larger area than a single-port one due to its wirings. The area of the single-port cache is split into: the area of the storage and the area of buffers and the multiplexer connecting the multiple cores to the caches single port, as shown in Figure 3. The area of such multiplexer and the buffering were estimated by synthesis on $90nm$ technology. Their total area was $0.7mm^2$. Although the figure shows that the private cache, single-port shared cache, and the hybrid cache have similar die areas, the private cache is the smallest cache, followed by variations between the single-port and hybrid caches orders that are related to the total capacity. Despite the overhead in its area, having better execution time,

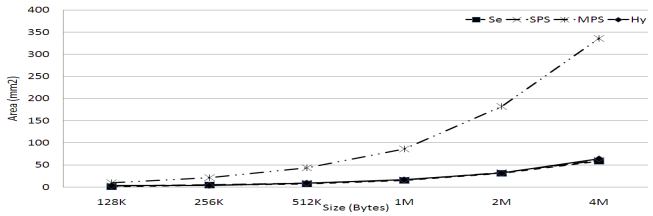
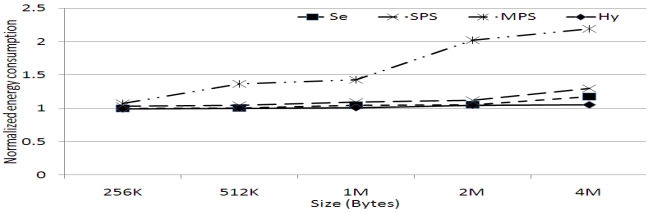
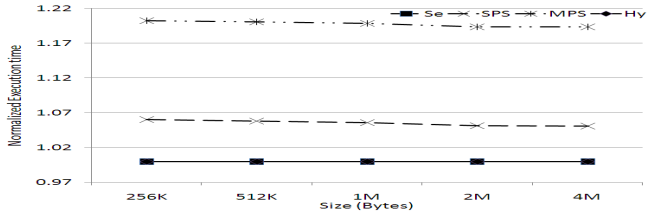


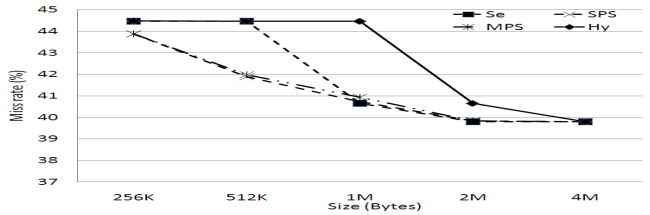
Figure 6: Die area of each cache configuration and storage capacity.



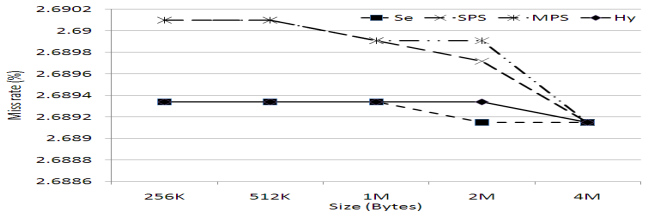
(a) Energy consumption



(b) Execution Time



(c) Read miss rate



(d) Write miss rate

Figure 7: Performance metrics of MiBench benchmark execution on different cache architectures.

energy consumption, the hybrid L2 cache has been shown that it is the optimum choice in the case of parallel workload, where the small shared case could be of effective usage. But, in the case of multi-processor workload, the presence of the small shared cache will be an obstacle due to the absence of shared data between the cores. Thus, there will be a waste of area associated to the cache. We Deployed MiBench benchmark in such workload to elaborate the performance of each cache architecture.

MiBench [20] contains a group of applications used in embedded systems. A runtime environment has been created based on such benchmark where a certain application has been assigned to each core, to assure complete heterogeneity. Figure 7 shows the systems performance with selected applications of MiBench benchmark. The applications are: *qsort, djpeg, fast Fourier transform and sha.* These applications are different in execution times with qsort is the fastest benchmark and djpeg is the longest application. This kind of distribution has been done in order to bias the system to make better use of the shared resources. Even with this kind of biasing, shared caches continue to perform worse than the private cache. Energy consumed in multi-port shared cache is higher than other configurations because it experiences a higher energy consumption per access with respect to other cache configurations, because it experiences a higher energy consumption per access. The different execution times are instead comparable because there is a duration dominant application which biases different architecture performance results.

It is also worth noticing the performance of the hybrid cache against the private cache. Figure 7(a) shows that the hybrid cache consumes similar energy values to the private cache but, with slight increase in the 1MB capacity. The execution time is also similar to each other. This implies that the change of the cache capacity does not have a significant effect on the whole system energy consumption.

3.2 Fixed Total Die Area Scenario

In this scenario, the die area allocated to L2 cache is estimated to be 50% of a chip containing the L2 cache and the core tiles. Based on $90nm$ technology, the estimated die area for the L2 cache in a system with 4 ARM9E cores was $11.067mm^2$. Based on this size criteria, Table 1 shows the best storage capacity allocated to each cache architecture. The only restriction on the storage capacity was that it should be a power of 2, in order to simplify the cache design.

From this table, it is shown that the total capacity of the

Cache Architecture	Size
Separate	128KB per cache
Single-port shared	512KB
Multi-port shared	128KB
Hybrid	128KB per separate 128KB for the shared

Table 1: L2 cache capacities of different cache architectures.

separate and the single-port caches are the same. But, the multi-port cache has much lower capacity since the wiring required for each gate to be accessed concurrently from more than one port causes a significant wiring overhead.

By comparing the allocated capacity to each architecture in order to be best fit in the given area with the capacity allocated in subsection 3.1, the multi-port cache is the only architecture that is affected with the mentioned condition (the fixed die area).

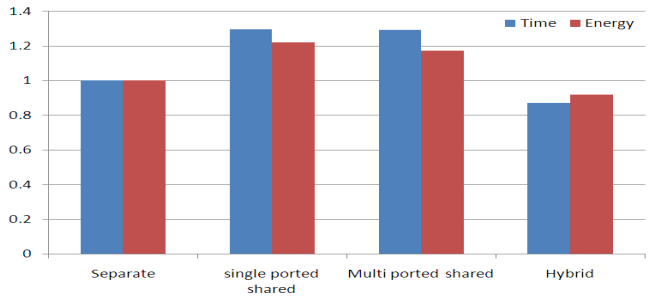


Figure 8: Energy consumption and execution time of Jpeg benchmark applied to different cache architectures.

Figure 8 shows the different caches performance with the Jpeg benchmark. By reducing the cache capacity, the miss rate becomes higher which implies the multi-port cache having the slowest execution time. However, the differences in the energy consumption are not much relevant since the increase in the miss rate is disaffected by a reduced energy consumed per access. Thus the energy consumption of both the multi-port and single-port caches are relatively comparable. The shrinking of multi-port cache capacity has the greatest effect on MiBench, as shown in Figure 9. The execution time of the multi-port cache based system is the highest among the other systems. This is due to the massive augmentation in the miss rates when the multi-port cache capacity is reduced in such scenario (fixed area).

4. CONCLUSIONS

On-chip memory organization is one of the most important aspects that can influence the overall system behavior in multi-processor systems. For embedded MPSoCs L1 cache has been deeply studied, but there is still not a dominant architectural paradigm for the L2 caches. In this paper, we developed a set of parameterizable models for L2 caches and integrated them in an accurate virtual platform environment specifically designed for Embedded MPSoC design space explorations.

In a real scenario we have evaluated private, shared, and hybrid L2 caches according to area, energy consumption, and

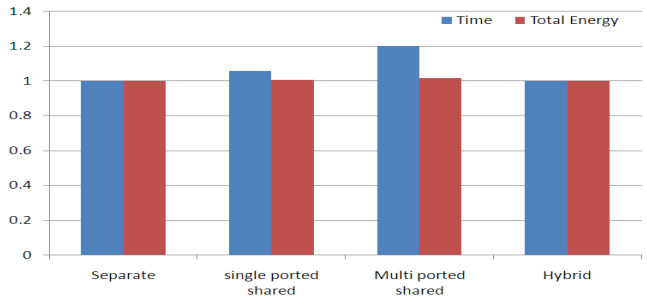


Figure 9: Energy consumption and execution time of MiBench benchmark applied to different cache architectures.

execution time metrics, employing various workloads. Simulation results show that the private L2 cache achieves better performance than the shared one when heterogeneous workload is deployed: 20% faster execution time, 60% less energy consumption, and 400% better area utilization than multi-ported cache, as well as 7% faster execution time, and 10% less energy consumption than single ported cache. Whereas if there is shared data in the runtime environment, the hybrid cache achieves better results than the private one (10% faster execution time and 10% less energy consumption).

5. ACKNOWLEDGMENT

The work described in this publication was partly supported by the PREDATOR Project funded by the European Community 7th Framework Programme, Contract FP7-ICT-216008, EC-FP7 STREP Project nr. 248776-PRO3D STREP, and the Spanish Government Research Grants TIN2008-00508 and CSD00C-07-20811.

6. REFERENCES

[1] ARM11 MPCore. http://www.arm.com/products/DevTools/PB11MPCore.html.
[2] ARM Cortex-A9 MPCore. http://www.arm.com/products/CPUs/ARMCortex-A9_MPCore.html.
[3] Cell Broadband Engine. http://www.ibm.com/developerworks/power/cell/.
[4] MIPS32 1004K. http://www.mips.com/products/processors/32-64-bit-cores/mips32-1004k/.
[5] Abu Asaduzzaman, et al. Impact of level-2 cache sharing on the performance and power requirements of homogeneous multicore embedded systems. In *Microprocess. Microsyst*, Vol 33, 2009.
[6] The SimpleScalar tool set. http://www.simplescalar.com/.
[7] The M5 Simulator System. http://www.m5sim.org/wiki/index.php/Main_Page.
[8] S. Murali, et al. A Method for Routing Packets Across Multiple Paths in NoCs with In-Order Delivery and Fault-Tolerance Guarantees. In *Hindawi VLSI Design Journal, Special Issue on Networks-on-Chip*, Vol.2007, Article ID 37627, April 2007.
[9] D. Atienza, et al. Network-On-Chip Design and Synthesis Outlook. In Integration-The VLSI journal, Elsevier Science, ISSN: 0167-9260, Vol.41, Nr. 3, pp. 340-359, March 2008.
[10] G. Braun, et al. Processor/memory co-exploration on multiple abstraction levels. In *Proceedings of DATE'03*, 2003.
[11] M. Leeman, et al. Automated dynamic memory data type implementation exploration and optimization. In *Proceedings of Annual Symposium on VLSI*, Tampa, USA, pp. 222-224, February 2003.
[12] F. Catthoor, et al. Data access and storage management for embedded programmable processors. Kluwer Academic Publishers, Boston, USA, 2002.
[13] D. Atienza, et al. Systematic Dynamic Memory Management Design Methodology for Reduced Memory Footprint. In Proceeding of TODAES'06, pp.465-489, April 2006.
[14] The Simics tool set. http://www.virtutech.com/.
[15] The VaST Systems. http://www.vastsystems.com/.
[16] UNISIM: UNIted SIMulation environment. http://unisim.org/site/.
[17] J. Chang, et al. Cooperative Caching for Chip Multiprocessors. In *Proceedings ISCA'06*, June 2006.
[18] Z. Chishti, et al. Optimizing Replication, Communication, and Capacity Allocation in CMPs. In *Proceedings of ISCA'05*, Jun 2005.
[19] The MPARM virtual platform. http://www-micrel.deis.unibo.it/sitonew/research/mparm.html.
[20] M. Guthaus, et al. MiBench: A free, commercially representative embedded benchmark suite. In *Proceedings of WWC'01*.
[21] R. Kumar, et al. Core Optimization for Heterogeneous Chip Multiprocessors. In *Proceedings of PACT'06*.
[22] R. Kumar, et al. Single-ISA Heterogeneous Multi-Core Architectures for Multithreaded Workload Performance. In *Proceedings of ISCA'04*.
[23] CACTI 6.0. http://www.cs.utah.edu/~rajeev/cacti6/.