On-chip memories, the OS perspective

Carlos Villavieja*,Isaac Gelado*,Alex Ramirez*†,Nacho Navarro*
*Departament d'Arquitectura de Computadors. Universitat Politècnica de Catalunya
C/ Jordi Girona 1-3, Campus Nord. 08034 Barcelona, Spain.
e-mail: {cvillavi,igelado,aramirez,nacho@ac.upc.edu}
†Barcelona SuperComputing Center, Spain.

Abstract

This paper is a work in progress study of the operating system services required to manage on-chip memories. We are evaluating different CMP on-chip memories configurations. Chip-MultiProcessors (CMP) architectures integrating multiple computing and memory elements presents different problems (coherency, latency, ...) that must be solved. On-chip local memories are directly addressable and their latency is much shorter than off-chip main memories. Since memory latency is a key factor for application performance, we study how the OS can help.

I. Introduction

The emerge of Chip-MultiProcessors (CMPs) and many-core chips has introduced huge computational power that using traditional programing models, compilers and operating systems is hard to deal with. Chip MultiProcessors are composed by many cores and different distributed memories. The large number of hardware threads these architectures support are not being used at all. In order to use these hardware threads, applications require of higher degrees of parallelism. Therefore, CMP processors require of more adequate programing models. New programming models are already targeting CMP, however, achieving a higher degrees of parallelism is not the only challenge, but to efficiently manage all the hardware resources CMP provide.

As an example, the Cell BE processor already integrates on-chip many processing elements and many different memories. Besides traditional off-chip global memories and cache memories, these processors include on-chip memories in terms of local memories or Scratch-Pad memories. In the Cell, these local memories are named Local Stores. They are included as non-cacheable distributed memories with guarantied latency and high bandwidth replacing traditional cache memories. Another key difference between Local Stores and cache memories is that they are addressable. They require the programmer to explicitly manage data transfers from off-chip main memory to local memory and vice-versa. For some parts of an application, cache memories may perform better than Local Memories(LM) and vice-versa. This is why Scratch-pad memories (reconfigurable cache memory/local store) are also being introduced [1],[2].

CMPs are not only being used in high performance computing but also in embedded computing as home gateways, game consoles, etc. Stream computing [3] is becoming one of the major target for these architectures. One reason is the easy split of the applications in computation and communication kernels. These computational kernels exploit SIMD capabilities of the cores and the guarantied latency of on-chip memories. Stream Computing is a parading example where *block algorithms* are used. *Block algorithms* are parts of code that massively compute a limited data set. Therefore, the efficient usage of on-chip memories might introduce performance improves.

As shown in Figure 1, on-chip memories are considered local memories. Local memories are addressable by any core in the CMP. For this reason, an application uses a computer system as a single addressable memory. All data and code is accessible within this virtual space. With a load/store instructions, any processor can access the whole application virtual space. On-chip data transactions have a much higher bandwidth than off-chip operations. Minimizing off-chip data transactions versus on-chip improves significantly applications performance because of the higher bandwidth between on-chip elements. This is why a new major goal is to provide efficiency in data allocation and placement. A single global name space allows any processor to access any memory location using regular load/store instructions. Having the whole virtual address space accessible makes easier to minimize off-chip traffic.

II. Inside the Chip-MultiProcessor architecture

In this section, we describe a generic scalable CMP architecture. This processor architecture will be our example processor architecture to illustrate all the services required in a CMP architecture. CMP architectures are usually composed by a set of processors and local memories interconnected as shown in Figure 1. Besides these on-chip elements, the off-chip main memory is also shown. Following Figure 1, we describe the different memory elements of the architecture.

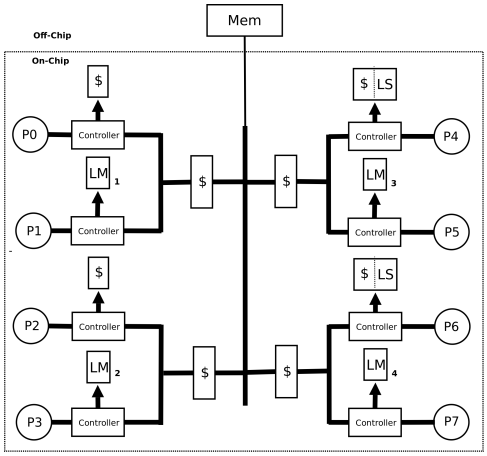

Fig. 1: Generic CMP architecture. The processor architecture is composed by eight processors or cores (Pi), and some local memories. The figure also shows the connection to main memory.

A. Memory

Memory elements in a CMP architecture can be found on-chip and off-chip. The off-chip main memory is usually a DDR chip interconnected through a bus to the processor chip. Other memories can be found on-chip connected to the cores through the on-chip interconnection network. All memory operations are managed by a memory controller. The memory controller is the responsible to locate the data. Moreover, some core configurations may have a second level cache connected to the core or a local addressable memory. The following list is a description of some local on-chip memories:

- Cache memory ($): represents a memory managed by hardware, typically a L2 cache memory. Cache memories are not addressable and they do not affect the OS memory management. Cache memories can be connected in a CMP to private use by one core or to shared use by various cores.
- Scratchpad memory ($ —LS): Simple array of memories directly addressable and explicitly managed by software. They are smaller than hardware cache memories. They also consume less energy per access. Their access time is predictable because as all addressable memories there is no possible miss. They are reconfigurable and they can act as an addressable memory or as a hardware cache memory.
- Local Memory (LM): Type of scratchpad memory, represents a local memory managed by software with a guarantied latency. They are directly addressable.

The operating system manages addressable memories by setting up the protection and translation mechanism when a static or dynamic allocation is requested. Besides, the OS affects the application cache memory contents, by its own use of memory, but mainly by the cache flush at a privilege mode or execution context switch. In order to efficiently manage memory allocation and placement, every possible operation over the on-chip/off-chip memories has to be considered. The associated cost of each memory allocation can be measured using the latency to access this physical memory from the target processor.

For our study, we use the memory operation costs of a Cell BE processor. They are calculated based in a CMP architecture with cores running at 3.2GHz [4],[5].

III. Work in progress status

In this work, we propose an efficient system memory management to handle the access time variation from on-chip to off-chip memories in CMPs, by proposing new mechanisms for hierarchical addressable physical memory allocation and placement. We also take into consideration the coexistence of local storages with (somehow configurable) levels of processor caches. The increasing relative latency to memory, taking hundreds of cycles (off-chip), means that applications must be aware of on-chip memories if they want to perform well. We are currently evaluating how different CMP configurations affect application performance and how they can be managed by the operating system.

References

[1] R. Banakar, S. Steinke, B. Lee, M. Balakrishnan, and P. Marwedel. Scratchpad memory: A design alternative for cache on-chip memory in embedded systems. In *Proc. of the 10th International Workshop on Hardware/Software Codesign, CODES, Estes Park (Colorado)*, May 2002.

[2] J. Gummaraju, M. Erez, J. Coburn, M. Rosenblum, and W. J.Dally. Architectural support fot the stream execution model on general-purpose processors. In *Proceedings of the Sixteenth International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2007.

[3] J. Gummaraju and M. Rosenblum. Stream programming on general-purpose processors. In *Proceedings of IEEE International Symposium on Microarchitecture (MICRO 05)*, 2005.

[4] D. Jimenez-Gonzalez, X. Martorell, and A. Ramirez. Performance analysis of cell broadband engine for high memory bandwidth applications. *Performance Analysis of Systems and Software, 2007. ISPASS 2007. IEEE International Symposium on*, pages 210–219, 25-27 April 2007.

[5] M. Kistler, M. Perrone, and F. Petrini. Cell multiprocessor communication network: Built for speed. *Micro, IEEE*, 26(3):10–23, May-June 2006.