DESIGN AND PERFORMANCE EVALUATION OF A CACHE CONSISTENT NOC-BASED MP-SOC

Gustavo Girão[1], Bruno Cruz de Oliveira[1], Rodrigo Soares[2], Ivan Saraiva Silva[1]

[1]Departamento de Informática e Matemática Aplicada – UFRN, Brazil
[2]Escola Politécnica – USP, Brazil

{girao, bcruz}@natalnet.br, rodsoares@gmail.com, ivan@dimap.ufrn.br

ABSTRACT

With the increasing complexity of current applications and the processors reaching its physical speed limit the most feasible solution to increase the processing power of a computational system is parallelism. Recent researches have integrated several processors in a single chip. This new technology is called Multiprocessor System-on-chip (MP-SoC). This paper presents the SystemC implementation of a complete MP-SoC platform called STORM. The platform integrates a Sparc V8 processor, a Cache module, a Directory module for cache coherence maintenance, a Memory module and mesh-based NoC. These modules are presented in details, as also the cache coherence mechanism and the NoC models. The platform can run applications written in C and compiled with Sparc-gcc cross-compiler. To analyses the STORM performance this paper presents the simulation results of two, including a parallel JPEG implementation.

1. INTRODUCTION

The complexity of current systems grows pushed by the crescent demands of applications, especially multimedia. Technology provides an unprecedented integration capacity, culminating with the creation of Systems-on-Chip (SoCs). On the other hand, despite of that, the time-to-market of new products decreases gradually. The combination of these two facts leads system designers to a challenge hard to overcome: the design of complex systems within a limited period of time. A popular approach to face this challenge is the concept of platform-based design, used in conjunction with components reuse. A platform can be generically defined as a high abstraction level model that covers possible low level refinements [1].

Another important trend in current system design is the design space exploration. Due to the complexity of current systems there are so many variables to be taken into account, that it is hard to find a good balance point among performance, on-chip area and power consumption. The decision of a satisfactory result might need several simulations and adjustments in the project. The use of platform-based design can clearly reduce the costs of design space exploration. As a solution for a better and fast design space exploration it is desirable to have a hardware design language that allows a high abstraction level. To attend this need, in 1999 it was announced the OSCI, and released the first SystemC [2] version. SystemC allows the modeling of a RTL system starting from a pure C/C++ code, through a series of refinements in the code, such as insertion of timing and synchronization and separation between hardware and software functionalities [3] (top-down methodology).

This paper presents the SystemC design of a complete MP-SoC platform, using the methodology described above (the combination of top-down and bottom-up methodology). All of its components were implemented in a cycle-accurate SystemC. The platform consists of the SPARC V8 microprocessor, a Cache module (DCache and ICache), a Network-on-Chip model, a Directory module for cache coherence maintenance, and a Memory module. For programming the platform the C language can be directly used through the Sparc-gcc cross-compiler.

This paper is divided as follows: Section 2 presents an overview of current use of SystemC to MP-SoC implementation and simulation; the implemented platform's features are presented in Section 3; in Section 4 there are some results about the platform concerning about cycles per instruction and average buffer occupation; and finally the conclusions are presented in Section 5, followed by the references.

2. RELATED WORKS

Current commercially available MP-SoCs are heterogeneous bus based application-specific on-Chip platforms, containing some couple cores. Most noticeably are: Intel's network specific IXP2850 [4]; ST's Nomadik, for mobile multimedia applications, based in an ARM9 processor [5] and Texas Intrument's OMAP [6], for wireless applications. Those platforms must work under area, energy consumption and real-time performance constraints.

However the combination of hundreds cores integration capability and parallel and concurrent software design, including NoC integration, Operating System support (race condition avoidance) and cache coherence, are not yet a reality. In [7] some of this software issues are included but considering a bus based platform. Some co-simulation (using an Instruction-Set Simulator and a SystemC implementation of a bus) papers have been presented, as in [8] and [9], methodology of high-level refinements is presented in [3] and an ASIC is used as an example. This work aims to provide a NoC based MP-SoC platform with coche consistency support. This platform will be used to MP-SoC design space exploration and programming model research.

3. THE STORM PLATFORM

3.1.Overview

STORM (MP-SoC DirecTory-Based PlatfORM) is a customizable Multi-Processor System-on-Chip (MP-SoC) platform, designed to support up to 256 cores. Its objectives are to produce a real and fully functional MP-SoC platform, actually running binary compiled codes of complete applications written in a high level language (C language), and able to provide a set of results, especially useful in design-space exploration. These results regard microprocessor execution, NoC communication and memory hierarchy overhead. Currently, on-chip area and power consumption results are not given, but we expect to include those by using the proper tools and estimation techniques [10].

Since it is a platform, STORM does not possess a fixed architecture. Its supported cores can be placed anywhere in the NoC, and its architecture is identified during the boot process. Currently, STORM supports the integration of SPARC V8 processor plus a Cache module (DCache and ICache); and Memory module plus its associated Directory module, as seen in figure 1. Any other SystemC module can be placed in STORM, as long as it follows its standard NoC communication protocol. The NoC model, NoCX4 (Mesh/Torus topology) [11] follows this protocol.

All Memory modules instantiated form a single address space, which makes STORM a Shared Memory environment. This way, running processes communicate via shared variables (global variables), and synchronize via mutexes. The mutexes were implemented in a C library using the SPARC's LDSTUB and SWAP instructions to perform Test-and-Set operation. Cache coherence is attained by using the directory cache coherence protocol, implemented in the Directory module.

3.2.Cache Coherence

Currently STORM has a directory based mechanism to

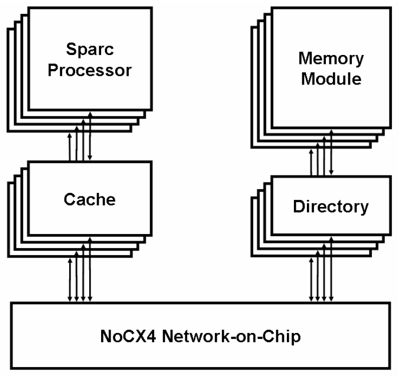

Figure 1. **STORM Modules and their integration.**

maintain coherence among all caches in the system. The cache coherence problem could be easily solved in a bus-based system, by using the snoop protocol [12]. The implementation of snoop protocol in a NoC is not impossible, but it would represent a costly communication overhead, due to the necessary broadcast in every memory access. Therefore, a feasible choice is the directory protocol, which centralizes all information about memory accesses [13]. To facilitate the control over memory's positions and to lessen the memory overhead incurred by the use of a Directory, the memory is divided in blocks. These blocks have configurable size, but it is always the same size of cache's blocks.

Each Memory Module in STORM has a Directory module that manages its NoC communication and cache coherence. It has information on the status of all blocks in that memory (clean or dirty) and the processors that currently have copies of the block.

STORM uses Dirn NB Directories, meaning it requires n+1 bits per block in the memory to store its status, where n is the number of processor cores in the platform [13]. That memory bits overhead is necessary so every processor can have a copy of the block.

To store the block status information, the directory uses two tables: a Status TAble (STA) and a Processor TAble (PTA). The STA contains information concerning all blocks in that memory. The PTA contains the NoC address of all active processors in the system. The STA is composed of a bit matrix. Every row in the table has information concerning one memory block. The Dirty bit indicates if that particular block is dirty (i.e. was modified and not updated to memory) or not. The other bits indicate which processors have a copy of that block. A dirty block will only have one copy. Notice that the STA alone has no information on where in the NoC are the processors with the block copies. For that, it is necessary that the directory has another table with processor's NoC address, the PTA. The PTA and STA were implemented as dedicated memories of the Directory module, no system memory accesses are necessary.

The Cache module has a Cache Communication Manager (CaCoMa), which performs the cache/NoC router communication. The CaCoMa has a virtual address to physical address translation table, ATA

(Address TAble), so the CaCoMa can calculate the correct destination of a memory access. Both Directory's PTA and CaCoMa's ATA are mounted during boot process. The STA varies according to the system's behavior, the PTA is static. The Address Table is also implemented as dedicated memory of the CaCoMa.

The results of cache coherence are presented in the next section.

3.3. Interconnection

STORM uses Network-on-Chip for modules' interconnection. The NoC model implemented was the NoCX4, a mesh/torus topology. NoCx4 uses VCT switching, credit-based flow control, RR (Round-Robin) or FCFS (First-Come First-Served) arbitration and size configurable FIFO (First-In First-Out) buffering and uses dimensional routing.

4. RESULTS

Table 1 shows minimum costs in cycles for Cache operations. These numbers has been obtained using a platform configuration (figure 2) with two processors and three memories. On this configuration, each processor has its own local variables (non-shared) stored in the memory on the right and on the left (for the processor on the right and the one on the left, respectively). Shared data has been stored in the memory located in the center of the configuration so each processor is located at the same distance from this memory. Applications running on the processors in the configuration above explained were written in such way that guarantee minimal costs without any NoC traffic issues.

Following are presented the results about two simulations over the platform. The running applications were the well-known Mergesort algorithm and a JPEG encoder. The platform configuration used in these simulations are presented in figure 3 (a and b, respectively).

The first aspect analyzed is the number of cycles per instruction (CPI). The equation used to calculate this value is shown in figure 4 [14]. Figure 5 presents the CPI for Mergesort and JPEG applications. Has been used simulations for several DCache different sizes varying form 32 bytes to 64 Kbytes.

Both simulations indicate that the number of cycles per instruction decreases with the increase of cache size. This occurs because with a larger size of DCache, the number of cache coherence maintenance packets in the NoC decrease and hence, the number of cycles to complete a read or write operation also decreases. The limit to this behavior is find ins a 512 bytes cache.

The behavior of CPI according with the number of processors has also been analyzed. Figure 6 presents the CPI versus processors number to the Mergesort application considering a cache of 512 bytes. It is possible to realize that the CPI increases with the number of processors. This is due to the fact that with more processor there is more competition for a shared data and

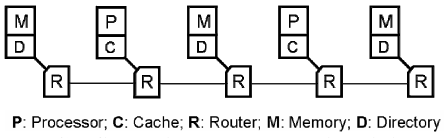

Figure 2. Configuration used to obtain minimum costs.

Table 1. Minimum costs.

Operation	*NocX4*	*Obtree*
Read Hit	1	1
Read Miss Clean	36	30
Read Miss Dirty	61	49
Write Hit with Permission	1	1
Write Hit without Permission	27	21
Write Miss Clean	36	30
Write Miss Dirty	60	48

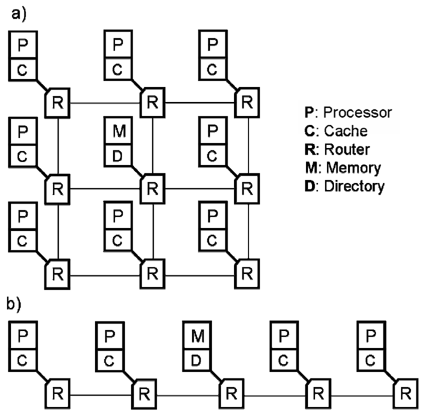

Figure 3. Configurations used in a)Mergesort and b)JPEG

$$\mathrm{CPI} = \frac{c \times p}{\sum_{j=0}^{P-1} i_j}$$

c: total cycles
p: total processors
i_j: total instructions of j processor

Figure 4. Cycles per Instruction equation.

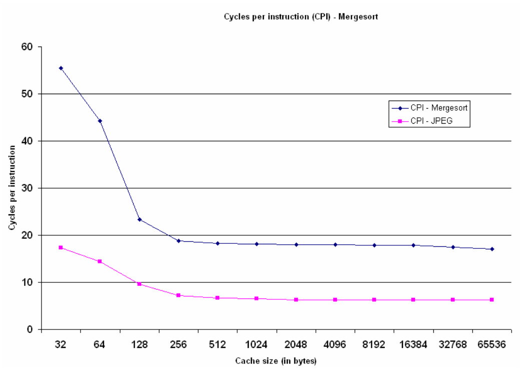

Figure 5. CPI vs. Cache size.

so the number of cache coherence maintenance packets increases.

Another important feature to be analyzed is the average occupation of Directory and Cache receiving buffers. The figures 7 and 8 show the average occupation of Directory and DCache buffers (respectively), using the Mergsort application and several DCache sizes. From these figures is possible to conclude that the average

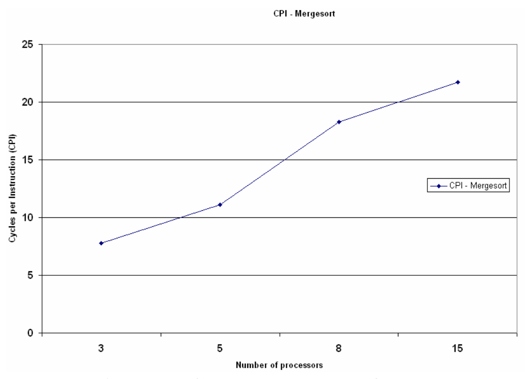

Figure 6. CPI vs. Number of processors.

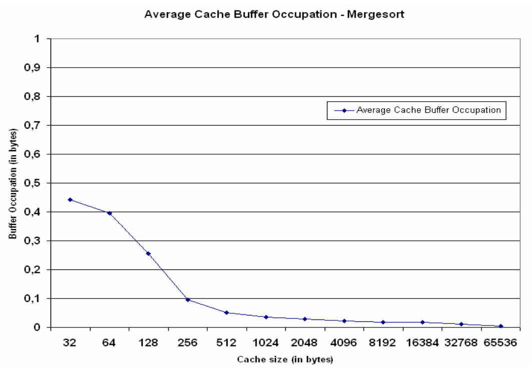

Figure 7. Average Directory buffer occupation (Mergesort).

buffer occupation strongly increases with a larger DCache size. This is due to the decrease of the number of read and writes request sent to the Directory. It is also important to realize that the average buffer occupation on DCache is way smaller than the average buffer occupation on Directory. This observation can be explained by the fact that this cache coherence maintenance solution centralizes all the requests on the Directory module.

5. CONCLUSIONS

This paper presented the design and implementation of a MP-SoC platform that uses a network-on-chip as interconnection mechanism. STORM platform adopts a directory-based solution for the cache coherence problem. The platform was developed using the SystemC hardware description language for a better and faster design space exploration. Also in this paper was presented a set of simulations of well-known algorithms exposing aspects of latency of cache operations, cycles per instruction and average buffer occupation on each simulation.

About the simulations is possible to conclude that the directory solution do not represent a great overhead in terms of buffer occupation and has a regular behavior on the number of cycles per instructions starting in an specific size of cache.

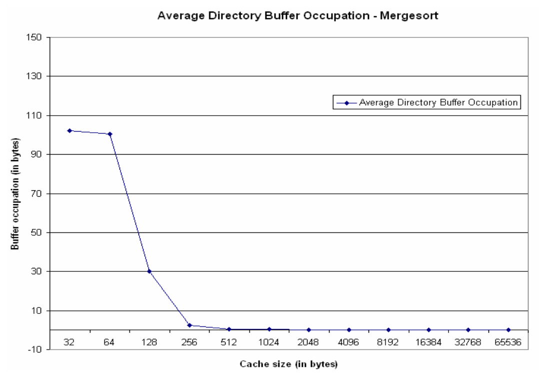

Figure 8. Average Cache buffer occupation (Mergesort).

6. REFERENCES

[1] A. Sangiovanni-Vincentelli, G. Martin; Platform-based design and software design methodology for embedded systems; Design & Test of Computers, IEEE Volume 18, Issue 6, pp. 23 – 33, Nov.-Dec. 2001.
[2] Open SystemC Initiative (OSCI), Functional Specification for SystemC 1.0, 1999.
[3] F. Abbes; E. Casseau; M. Abid; “SoC Design Case Study Using SystemC Specifications”, ICM03, 15th International Conference on Microelectronics, December 2003.
[4] Intel, “Product Brief: Intel IXP2850 Network Processor,” 2002.
[5] A. Artieri; et al; “NomadikTM Open Multimedia Platform for Next-generation Mobile Devices”, STMicroelectronics Technical Article TA305, 2003.
[6] J. Helmig; “Developing core software technologies for TI’s OMAPTM platform” Texas Instruments, 2002.
[7] L. Benini; et al; “MPARM: Exploring the Multi-Processor SoC Design Space with SystemC” The Journal of VLSI Signal Processing, 41(2): 169 – 182, September 2005.
[8] L. Benini; et al; “SystemC Cosimulation and Emulation of Multiprocessor SoC Designs”, IEEE Computer 36, April 2003.
[9] F. Fummi; et al; “Native ISS-SystemC Integration for Co-Simulation of Multi-Processor SoC” Design, Automation and Test in Europe (DATE), Proceedings, 2004.
[10] S. Niar; S. Metfali; J.L. Dekeyser, “Power Consumption Awareness in Cache Memory Design with SystemC”, International Conference on Microelectronics (ICM), 2004.
[11] R. Soares , I.S., Silva, A. Azevedo, “When Reconfigurable Architecture Meets Network-on-Chip”, 17th Symposium on Integrated Circuits and Systems Design, pp. 216 – 221, 2003.
[12] A. S. Tanenbaum; Structured Computer Organization, 4th edition; Prentice Hall, 1999-2000.
[13] K. Hwang “Advanced Computer Architecture: Parallelism, Scalability, Programmability” McGraw-Hill, Inc. 1993.
[14] F. Pétrot, A. Greiner, P. Gomez, “On Cache Coherency and Memory Consistency Issues in NoC Based Shared Memory Multiprocessor SoC Architectures”, 9th EUROMICRO Conference on Digital System Design, pp. 53-60, 2006.