

Exploration of Distributed Shared Memory Architectures for NoC-based Multiprocessors

Matteo Monchiero Gianluca Palermo Cristina Silvano Oreste Villa
Dipartimento di Elettronica e Informazione
Politecnico di Milano, Milano, Italy
Email:silvano@elet.polimi.it

Abstract— Multiprocessor system-on-chip (MP-SoC) platforms represent an emerging trend for embedded multimedia applications. To enable MP-SoC platforms, scalable communication-centric interconnect fabrics, such as networks-on-chip (NoC), have been recently proposed. The shared memory represents one of the key elements in designing MP-SoCs, since its function is to provide data exchange and synchronization support. In this paper, a distributed shared memory architecture has been explored, that is suitable for low-power on-chip multiprocessors based on NoC. In particular, the paper focuses on the energy/delay exploration of on-chip physically distributed and logically shared memory address space for MP-SoCs based on a parameterizable NoC. The data allocation on the physically distributed shared memory space is dynamically managed by an on-chip Hardware Memory Management Unit. Experimental results show the impact of different NoC topologies and distributed shared memory configurations for a selected set of parallel benchmark applications from the power/performance perspective.

I. INTRODUCTION

Multiprocessor system-on-chip (MP-SoC) architectures are emerging as appealing solutions for embedded multimedia applications. In general, MP-SoCs are composed of core processors, memories and some application-specific coprocessors. Communication is provided by advanced interconnect fabrics, such as high performance and efficient networks-on-chip (NoCs) [1]. To meet the system-level design constraints in terms of performance, cost and power consumption, a careful design of the memory subsystem and the communication layer is mandatory.

The main focus of this paper is to explore distributed shared memory architectures suitable for low-power on-chip multiprocessors based on the NoC approach. The three main characteristics of our target MP-SoC architecture are: the NoC communication, the memory subsystem, and the memory management unit.

- The PIRATE NoC [2] is a packet-based network-on-chip IP core composed of a set of parameterizable interconnection elements that can form different on-chip micro-network topologies. The router architecture is based on a crossbar switch, the switching policy is wormhole, the routing is distributed and the algorithm is static and table-based.
- The memory subsystem is based on a Non Uniform Memory Access (NUMA) paradigm. The latency of memory

accesses is non-uniform, because the system is built around a network interconnect and the memory space is composed on physically distributed and logically shared memory modules. Besides the shared memory space, the memory model is composed of private memory spaces for each processing element (PE). The shared portion of the addressing space represents one of the key elements of the MP-SoC to support synchronization and data exchange between PEs.

- The data allocation/deallocation on the physically distributed shared memory space is dynamically managed by an on-chip hardware memory management unit (HwMMU). The memory management features are implemented in an hardware module so as to reduce the overhead due to the operating system memory management in terms of energy and performance [3]. The on-chip HwMMU makes the memory utilization more efficient especially for embedded applications whose memory requirements can change significantly during run-time.

For the target multiprocessor architecture, the paper explores different on-chip network topologies and a variable number of distributed shared memory modules. Although the origins of the considered interconnect topologies and memory architectures can be tracked back to the field of multiprocessing systems, a different set of design constraints must be considered when adopting these architectures to the system-on-chip design context. Energy consumption and area constraints become mandatory as well as high throughput and low latency.

The design space exploration has been carried out by using *GRAPES* [4], a network-centric MP-SoC modelling, simulation and power estimation platform. *GRAPES* features a cycle-accurate simulation environment for power/performance evaluation of MP-SoC architectures based on a configurable NoC.

Our analysis enables us to identify power/performance trade-offs that represent critical issues for the design of network-based MP-SoCs. In particular, the experimental results show how an optimized design of the on-chip memory subsystem can save both system energy and execution time depending on the intrinsic nature of the target application (communication- or computation-oriented). The exploration of NoC topologies can identify energy/delay trade-offs and communication bottlenecks.

The paper is organized as follows. In Section II, some

previous works are reported, while in Section III the target system-on-chip multiprocessor architecture is presented focusing on the interconnection, the memory subsystem and the memory management. The experimental results are described in Section IV. Finally, some concluding remarks are pointed out in Section V.

II. RELATED WORK

Most previous research in embedded systems has focused on static allocation and how to synthesize memory hierarchies for an SoC [5]. For applications whose memory requirements change significantly during run-time, static allocation of memory makes the on-chip memory utilization inefficient and system modification very difficult [6].

Scratchpad memories have been proposed as an alternative to large L2 caches [7], especially for embedded systems. When adopting scratchpad memories, a software controlled local storage is provided instead of L2 cache. The shared memory design targeted in this paper can be seen as a set of scratchpad memories, managed as a typical multiprocessor shared memory.

Several works focus on scratchpad/L2 design, especially for single-processor systems. The authors of [8] present an algorithm to optimally solve the memory partitioning problem by dynamic programming. Regarding to multiprocessor systems, Kandemir *et al.* [9] present a compiler strategy to optimize data accesses in regular array-intensive applications for a system based on scratchpads. In [10], the authors propose compositional memory systems based on the partitioning and exclusive assignment of the L2 lines to running threads.

These approaches are based on static or, for more complex approaches, dynamic profiling of the application, but they are not so effective when the memory utilization changes dynamically with the system load, especially in shared memory multiprocessors [11]. On the other hand, the dynamic memory management can consume a great amount of a program execution time, especially for object-oriented applications, without providing deterministic timing behavior. Many researchers have proposed hardware accelerators for dynamic memory management. The literature shows hardware implementations proposed in [12]–[14]. Chang *et al.* [15] have implemented the *malloc()*, *realloc()*, and *free()* C-Language functions in hardware. They also proposed an hardware extension integrated in the future microprocessors to accelerate the dynamic memory management [16].

In [17] and [3], the authors propose a memory allocator/deallocator (namely, SoCDMMU) suitable for real-time systems. The main characteristic is that, by using the SoCDMMU, the memory management is fast and deterministic. The main limitation of the SoCDMMU architecture is that all memory accesses need to pass through the SoCDMMU, becoming a bottleneck for multiprocessors.

None of the existing works in literature on MP-SoC architectures has yet compared contextually the design parameters related to NoC, memory sub-system and memory management with regard to throughput, latency and energy.

III. TARGET ON-CHIP MULTIPROCESSOR ARCHITECTURE

In this section, we present the architecture of the target multiprocessor system-on-chip, focusing on the central points of our analysis: the interconnection network, the organization of the memory subsystem and the memory management unit.

Figure 1 shows the architecture of the on-chip multiprocessor composed of several processing elements (PEs), memory modules and the HwMMU. The core of the system is the scalable communication-centric interconnect fabrics (namely, PIRATE NoC [2]). The PIRATE NoC is connected to each module through the corresponding network interface (NI), that implements the translation of communication protocols and the data packeting. The memory space is composed of a shared memory space and separate private memory spaces for each processing element (PE). Private data are cached to improve performance without requiring the need of cache coherency support.

Each processing node (PE) is composed of the core processor, the L1 private data cache, the L1 instruction cache, the shared data translation look-aside buffer (SD-TLB), the TLB for private data and instructions, and the communication coprocessor interfacing the NI. Two different TLBs have been introduced to support different virtual-to-physical addresses translation schemes (such as different page sizes). The L2 cache (unified for private data of each processor and instructions) is centralized on a dedicated module interfacing the network. The network also interfaces the off-chip main memory through the on-chip main memory controller.

The memory modules for shared data are distributed across the NOC topology providing non-uniform memory accesses. For each shared memory module, the corresponding memory controller interfaces the network. The proposed NoC-based architecture is scalable in terms of number of processing elements and distributed shared memory modules. The on-chip centralized MMU module supports memory allocation/deallocation dynamically. Finally, global asynchronous events are managed by the interrupt controller module.

A. Interconnection

The interconnect is the key element of the multiprocessor system, since it provides low latency communication layer, capable of minimizing the overhead due to thread spawning and synchronization. This approach, when compared to bus-based solutions, solves physical limitations due to wire latency providing higher bandwidth and parallelism in the communication.

PIRATE [2] is a Network-on-Chip composed of a set of parameterizable interconnection elements (routers) connected by using different topologies (such as ring, mesh, torus, etc.). The router internal architecture is based on a crossbar topology, where the $N \times N$ network connects the N input FIFO queues with N output FIFO queues. The number of each input (output) FIFO queue is configurable as well as queue length (at least one). The incoming packets are stored in the corresponding input queue, then, the packets are forwarded to

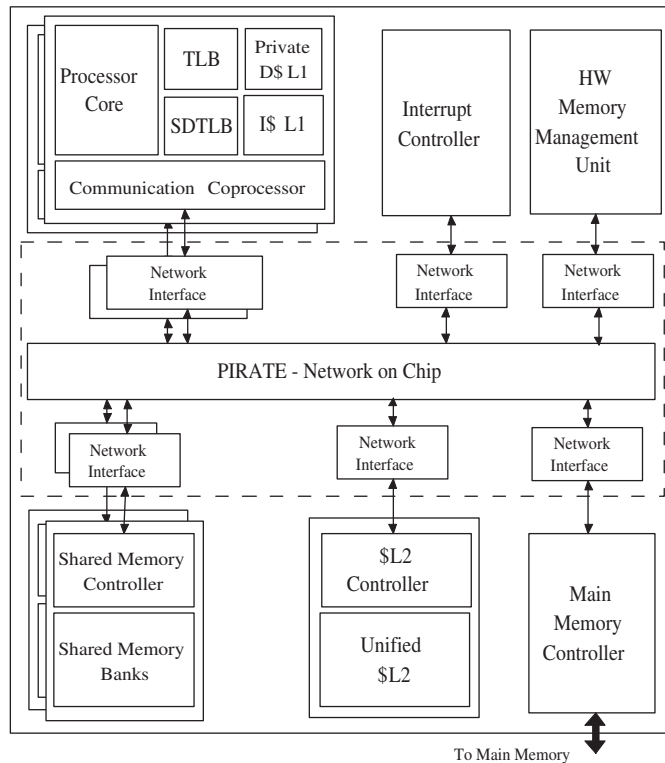


Fig. 1. Target on-chip multiprocessor architecture

the destination output queue. The I/O queues and the crossbar are controlled by the router controller, that includes the logic to implement routing, arbitration and virtual-channels. The PIRATE router is 3-stage pipelined architecture that implements table-based routing and wormhole switching. The PIRATE network interface (NI) implements the adaptation between the asynchronous protocol of the network and the specific protocol adopted by each IP-node composing the system. The NI is also responsible for the packeting of the service requests coming from the IP nodes. The service level is best effort and there is a round robin priority-based arbitration mechanism.

B. Memory Subsystem

The memory space is composed of separate private memory spaces and shared memory space. A private memory space exists for each PE and it cannot be seen by any other PE in the system except for the owner. Shared portions of the addressing space are mainly used for synchronization and data exchange. While private memories map to L1/L2 caches and off-chip main memory, shared memory is only implemented by on-chip RAM modules distributed on the NoC (without caching). Shared memory modules can be exposed to the programmer, which can partition and place shared data.

Private data are cached to improve performance without requiring the need of cache coherency support. Two levels of caching are provided: the L1-caches for private data and instructions are placed on each PE, while one L2 unified (private data and instructions) cache is centralized on a dedicated

module interfacing the network.

The memory modules for shared data are distributed on the NoC topology providing non-uniform memory accesses (*NUMA approach*). Shared data are not cached to avoid the overhead of cache coherency support. Globally, data locality is exploited for private data by L1 and L2 (where no coherency support is required), while shared data access locality is assured by distributing the shared memory modules on-chip.

C. Memory Management Unit

In the proposed target architecture, shared data are managed dynamically and shared data allocation and deallocation mechanisms are provided by the on-chip centralized HwMMU module. A virtual memory mechanism decouples logical addressing space, as seen by each processors, from its physical addressing implementation.

The memory management features have been implemented in a hardware module to reduce the overhead due to the operating system in terms of energy and performance [18]. The HwMMU makes the memory management fast and deterministic. The dynamic management makes memory utilization more efficient especially for embedded applications whose memory requirements can change significantly during run-time.

The HwMMU holds the table of the allocated shared pages (Shared Page Table) and the virtual to physical addresses correspondence for each shared page. The SD-TLB in each PE is used to cache this information and in our target architecture it has a fully associative structure. When a SD-TLB miss occurs,

TABLE I
COMPARISON OF EXECUTION TIME REQUIRED BY MMU COMMANDS IMPLEMENTED BY ON-CHIP HWMMU OR SW ROUTINE (WHERE *cycle* CORRESPONDS TO 200 MHZ PE CLOCK FREQUENCY AND *page* REPRESENTS THE NUMBER OF 4KB-PAGES TO BE MOVED/COPIED)

Command	On-chip HwMMU Exec. Time [cycle]	SW Routine Exec. Time [cycle]
<i>MMU_MALLOC(MEM, size)</i>	61	≥ 220
<i>MMU_FREE(addr)</i>	58	≥ 96
<i>MMU_COPY(MEM, add, number)</i>	$66 + 54886 \times \text{page}$	$\geq 220 + 94208 \times \text{page}$
<i>MMU_MOVE(MEM, add, number)</i>	$69 + 54886 \times \text{page}$	$\geq 316 + 94208 \times \text{page}$

physical page information are fetched from the HwMMU and the corresponding SD-TLB entry is updated.

There are two types of commands that the on-chip HwMMU can execute: *Allocation/Deallocation* and *Move/Copy*. These commands are described hereafter, while Table I compares, for each command, the execution time required by the HwMMU and by the corresponding software routine to completely execute the command from the user application. The execution times have been calculated (for the target architecture described in Section IV) by assuming one-hop network distance between each module. For the computation of execution times related to the software case, data structures used for the memory management have been assumed as already stored locally on private data caches. This assumption represents a lower bound for the execution time of software routines since, typically, data structures to support memory management in shared-memory multiprocessors are stored in shared memory. Globally, dynamic memory management provided by the HwMMU requires at least half execution time with regard to software routine. Furthermore, the HwMMU offers a predictable execution time, so the hardware solution is also suitable for real-time applications.

The centralized HwMMU does not represent a bottleneck for memory accesses, since, in the target architecture, SD-TLBs on each PE are used for locally caching address translations. The HwMMU is used for allocation/deallocation/copy/move for shared data and in case of SD-TLB misses to access the shared page table.

a) Allocation/Deallocation: The basic allocation and deallocation primitives for the shared data have the following structures:

MMU_MALLOC(MEM, size)

MMU_FREE(address)

where *size* is the number of data bytes to allocate, *MEM* is the number of the memory module where data must be allocated, and *address* is the pointer of the space to release. By setting to -1 the value of *MEM*, the memory allocation is assigned to HwMMU. In all cases (when *MEM* is equal to -1 or not) the allocation request is sent to the HwMMU.

b) Move/Copy: The HwMMU can be used not only to allocate/deallocate space, but also to *move* and *copy* shared data. These two operations are similar but, while move operation needs to free the original space of the data to be moved, the copy operation does not. The basic copy and move primitives have the following structures:

MMU_COPY(MEM, address, number)

MMU_MOVE(MEM, address, number)

where *address* is the page virtual address, *number* is the number of pages to be copied/moved (starting from *address*), while *MEM* is the number of the memory module where the pages must be placed. When *MEM* is equal to -1 , the decision on the destination memory module is left to the HwMMU.

The *MMU_MOVE* operation starts allocating space in the destination shared memory module indicated by *MEM*. Once performed the allocation, the HwMMU broadcasts an invalidate request to SD-TLB locations referring to the pages to be moved in order to force each memory request to those pages to pass through the HwMMU. Then, the copy of the pages can be performed and, if a request occurs during the copy, the HwMMU delays the response until the move operation ends. The source pages can be deallocated and the new page address is updated in the HwMMU table.

The *MMU_COPY* operation starts allocating space in the destination memory module indicated by *MEM*. The copy of the pages can then be done and, at the end of the operation, the new page address is sent back to the PE that required the copy. Performing a move primitive, the virtual address of the page is the same before and after the operation, while the copy primitive returns the new page address.

The *MMU_COPY* and *MMU_MOVE* operations can be useful to exploit memory locality (by data migration and duplication) and space compaction.

IV. EXPERIMENTAL RESULTS

In this section, we discuss the experimental results obtained by using the GRAPES platform [4] to model, to simulate and to estimate the power consumption of the target MPSoC by varying the memory architecture (in Section IV-B) and the NoC topology (in Section IV-C). The simulations have been carried out at the cycle-accurate level. Power model parameters have been estimated by gate-level simulation (with 90 nm STMicroelectronics CMOS technology) and extracted by experimental data (from ARM core [19] and from CACTI for memories [20]). Accuracy of power models can be directly derived from the accuracy of the models used for each module ([2], [19], [20]).

A. Experimental Setup

The parameters of the target MP-SoC architecture chosen for the experiments are reported in Table II. The system is

TABLE II
PARAMETERS OF THE TARGET ARCHITECTURE

Parameter	Value
Number of PEs	8
PE clock frequency	200 MHz
NoC clock frequency	400 MHz
NoC data-path size	1 Byte
NoC topology	Mesh
Memory interface and control	
logic clock frequency	200 MHz
Shared memory space	512 KB
SD-TLB size	8 entry
SD-TLB replacement policy	LRU
Shared memory page size	4 KB

composed of 8 PEs and each PE is an ARM7 core with a clock frequency of 200MHz. The 200 MHz clock frequency has also been used for shared memories interfaces and control logic. The *PIRATE-NoC* has a 400MHz clock to minimize the latency, while not dramatically impacting the power consumption. The network data-path size is equal to one byte to reduce the hardware complexity of the on-chip network, while the NoC topology has been configured as a mesh. The mesh topology includes $(12 + n)$ nodes, where n is the number of distributed shared memory modules (varying from 1 to 5 in the experiments reported in Section IV-B). Three different NoC topologies have been explored in Section IV-C.

The mapping of the IPs and memory modules to the nodes of the network topology has been optimized for each benchmark in order to minimize the average network latency by using an *in house* developed tool [4].

All the reported results have been obtained by using the dynamic features of the on-chip HwMMU by setting to -1 the MEM field in the MMU primitives. The algorithm used by HwMMU for data allocation is round-robin.

As application benchmarks, a set of kernels have been extracted from the SPLASH-2 suite [21] of parallel benchmark applications. The SPLASH-2 suite has been released to facilitate the study of centralized and distributed shared address space for multiprocessing. Two other applications have been written by using PARMACS macros [22]: a matrix multiplication algorithm (*Matrix*) and a vector normalization (*Norm*). PARMACS macros offer basic primitives for synchronization, creation of parallel processes and shared memory allocation. Table III summarizes the selected set of benchmarks with a short description of the functionality and the origin of each benchmark (SPLASH-2 suite or PARMACS-based).

The main characteristics of the selected benchmarks have been reported in Figure 2 for the target architecture with a single shared memory module. For each benchmark, the breakdown of the normalized execution time spent by each processor in computation, shared memory accesses and synchronization has been reported. Based on the results shown in Figure 2, the selected benchmarks can be grouped in computation-intensive (*FFT* and *Norm*), memory-intensive (*Matrix*) and synchronization-intensive (*LU-1*, *LU-2* and *Radix*).

TABLE III
THE SELECTED SET OF BENCHMARKS

	Description	From
<i>FFT</i>	Complex 1D radix- \sqrt{n} 6-step FFT	SPLASH-2
<i>LU-1</i>	Lower/upper triangular matrix factorization	SPLASH-2
<i>LU-2</i>	<i>idem</i> , but optimized for spatial locality (<i>contiguous blocks</i>)	SPLASH-2
<i>Radix</i>	Integer radix sort	SPLASH-2
<i>Matrix</i>	Matrix multiplication	PARMACS-based
<i>Norm</i>	Parallel vector normalization	PARMACS-based

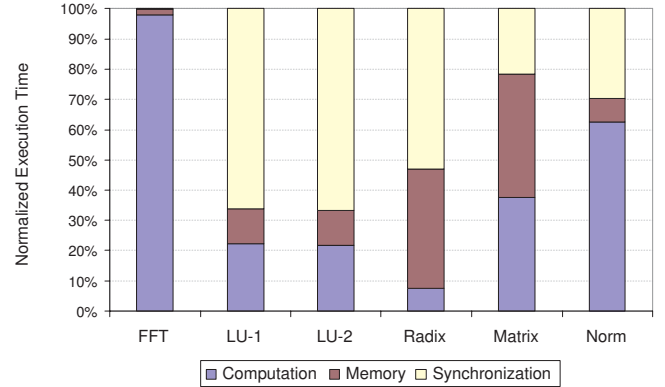


Fig. 2. Breakdown of normalized execution time spent in computation, shared memory accesses and synchronization for the selected set of benchmarks

B. Exploration of Distributed Shared Memory Architectures

In this section, we show the results of the exploration of the distributed shared memory subsystem.

Figure 3 shows the shared memory request latency for the Matrix application by varying the number of distributed shared memory modules from 1 to 5. Each point has been computed by considering the average value of shared memory request latency in $10\mu\text{s}$ time interval. Matrix benchmark has been chosen since is memory-intensive, so the effects of memory exploration are more evident.

The shared memory request latency represents the latency necessary to receive the response of a memory request. It includes the network request latency (from source to memory module), the memory latency, the network response latency (from memory back to the source of request) and the possible latencies due to contentions.

Except for the initial behavior (in the time slot between 0 ns and 1.5×10^6 ns), where the high values of request latency are mainly due to cold start SD-TLB misses, the shared memory request latency is almost stationary and its average value mainly depends on the number of physical memories. In Figure 3, it can be observed that using a single physical memory the request latency, on average, reaches the higher value (712 ns approximately). Increasing the number of physical memories, up to 4, we can observe a reduction in the average request latency (approximately 512 ns, 360 ns, 348 ns respectively for 2, 3 and 4 physical memory). The reduction is due to two main factors: first, the reduction in memory contentions, since with a higher number of physical

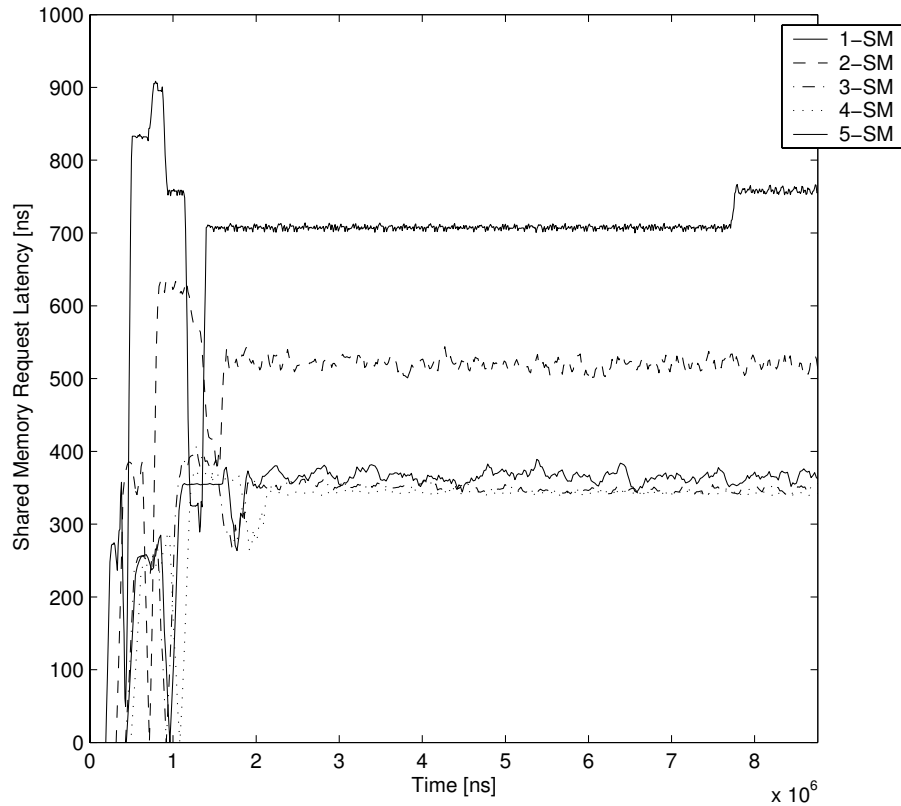


Fig. 3. Shared memory request latency for Matrix benchmark by varying the number of distributed shared memory modules from 1 to 5.

memories is less probable to have two accesses at the same time to the same memory, second, the reduction of the network latency, since with a higher number of physical memory distributed in the network is more probable to find the data at a reduced network distance with respect to a single shared memory. For 5 distributed shared memories, we can see how the average latency (approximately 376 ns) slightly increases with respect to the configurations with 3 and 4 physical memories. This is due to the fact that increasing the number of memories, the network size increases consequently as well as the diameter of the network. In the case of 5 memory modules, the advantages in memory contention and data locality are not enough to balance the disadvantage due to a greater network size.

Figures 4 and 5 show performance and energy results obtained by exploring the number of distributed shared memory modules from 1 to 3 across the network. The analysis for 4 and 5 distributed shared memories are not shown because a negligible performance speedup has been obtained for the selected set of benchmarks (as confirmed by Figure 3 for the *Matrix* benchmark).

For the selected benchmarks, Figure 4 reports the execution times for 2 and 3 distributed shared memory modules, where the reported values have been normalized with respect to a single shared memory. For all the benchmarks, the distribution of shared data implies shorter execution time. This is more evident for *LU-1*, *LU-2*, *Radix* and *Matrix* (for which the

gain is up to 53%), even if some small improvements can be observed also for the remaining benchmarks (up to 3%).

Distributing shared data in physical memories is more effective for *LU-1*, *LU-2*, *Radix* and *Matrix*, since these benchmarks are communication-based (as seen in Figure 2) due to large synchronization overhead (for *LU-1*, *LU-2* and *Radix*) or due to many accesses to shared data (for *Matrix*). In particular, *Matrix* obtains the highest performance advantage with respect to *LU-1*, *LU-2* and *Radix* because it manages a large amount of shared data. In fact, the other three benchmarks show a high communication rate, but limited to synchronization structures, that have small size. This behavior is more evident by increasing the number of physical memories. In fact, passing from 2 to 3 physical memories, *Matrix* continues to reduce the execution time (approximately 14%), *LU-1*, *LU-2* and *Radix* gains only approximately 2%. For *FFT* and *Norm* benchmarks, the advantage in distributing shared data is not evident due to the computation-based nature of these applications. The time spent in communication is negligible and so the advantage of a reduced shared memory request latency does not affect significantly the system performance.

For what concerns energy, Figure 5 shows energy results for systems including 2 and 3 shared memory modules normalized with respect to the system with a single shared PEs. The corresponding energy breakdown between PEs, shared memory subsystem and NoC is also shown. Energy contributions due to HwMMU, L2 cache, interrupt controller and

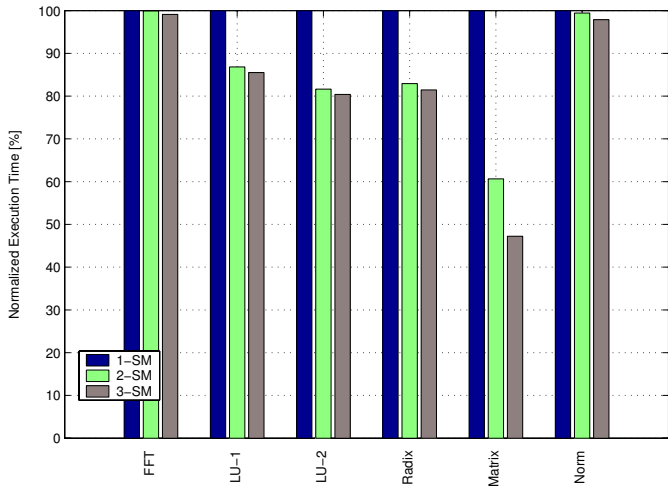


Fig. 4. Execution time for 2 and 3 distributed shared memory modules normalized with respect to a single shared memory for the selected set of benchmarks

main memory controller are not shown since their values are constant by varying the number of shared memories. Energy saving in Figure 5 is mainly due to two aspects: the first one is due to execution time reduction; the second one is related to the reduced energy cost of memory accesses for configurations of the shared memory distributed on 2 and 3 memory modules. The NoC energy is reduced, despite its complexity increases. In fact, more routers are needed to serve more memories, but the locality of the communication (in the system with distributed memories) trade-offs the energy consumption due to the additional routers. *FFT* and *Norm* do not show significant energy saving (up to 3%), while *LU-1*, *LU-2*, *Radix* and *Matrix* show up to 42% energy saving. The motivation of the small energy savings for *FFT* and *Norm* is the same shown for performance and proved from energy breakdown: these two benchmarks are characterized by high computation and a few memory accesses.

C. Exploration of NoC Topologies

In this section, three different NoC topologies have been considered from the energy/delay perspective: ring, spidergon [23] and mesh. For the exploration, the memory-intensive Matrix benchmark has been executed by the target architecture by varying the number of distributed memory modules from 1 to 3 (corresponding to 13- to 15-node topologies). In Figure 6, for each memory configuration, the reported values have been normalized with respect to the ring-based multiprocessor architecture.

The exploration results reported in Figure 6.a show how, due to the reduced average number of hops, mesh offers the shortest execution time, followed by spidergon and ring. This behavior is more evident for a single memory module, while, increasing the number of shared memory modules, the execution time is less sensitive with respect to network topologies.

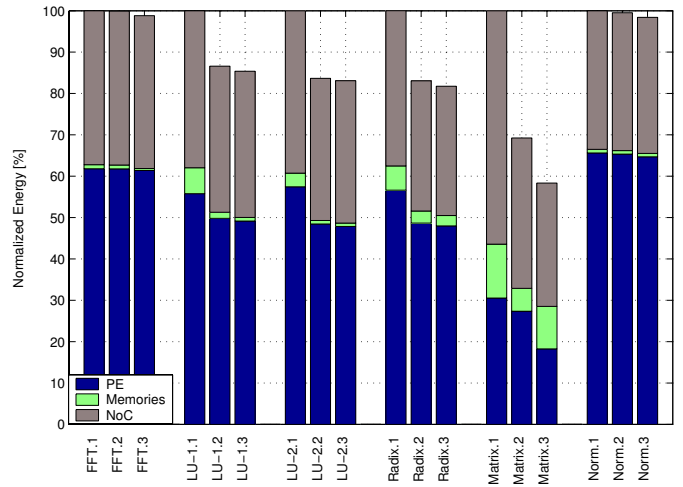


Fig. 5. Energy breakdown for the selected set of benchmarks ($\langle bench \rangle.x$, where x is the number of distributed shared memory modules)

A similar trend has been reported in Figure 6.b for energy in the cases of 1 and 2 memory modules, where the energy saved by the reduced execution time is more than the energy spent for a more complex network (passing from 3-port router required by ring, through 4-port router for spidergon up to 5-port router for mesh). For 3 memory modules, ring topology offers the lowest energy, followed by spidergon and mesh. This is due to the fact that, in the case of 3 memory modules, the energy saved by the reduced execution time is less than the energy spent for a more complex network.

V. CONCLUSIONS

This paper focuses on the power/performance exploration of the distributed shared memory approach in MP-SoC architectures based on network-on-chip. For this purpose, a distributed shared memory architecture has been proposed, that is suitable for low-power on-chip multiprocessors and supported by an on-chip hardware MMU. The scalability of the proposed architecture with respect to the number of distributed memory modules has been evaluated in terms of power and performance. Exploration results have also shown how, increasing the number of shared memory modules, the system-level latency and energy are less sensitive with respect to network topologies.

As future work, we are considering the energy/delay comparison for a distributed shared memory multiprocessor including hardware cache coherency support for L1 shared data, not considered in this paper.

REFERENCES

- [1] L. Benini and G. D. Micheli, "Networks on Chip: A New SoC Paradigm," *IEEE Computer*, vol. 35, no. 1, pp. 70–78, January 2002.
- [2] G. Palermo and C. Silvano, "PIRATE: A Framework for Power/Performance Exploration of Network-On-Chip Architectures," in *Lecture Notes in Computer Science*, vol. 3254, 2004, pp. 521–531.
- [3] M. Shalan and V. Mooney, "Hardware Support for Real-Time Embedded Multiprocessor System-on-a-Chip Memory Management," in *Proceedings of CODES*, May 2002, pp. 79–84.

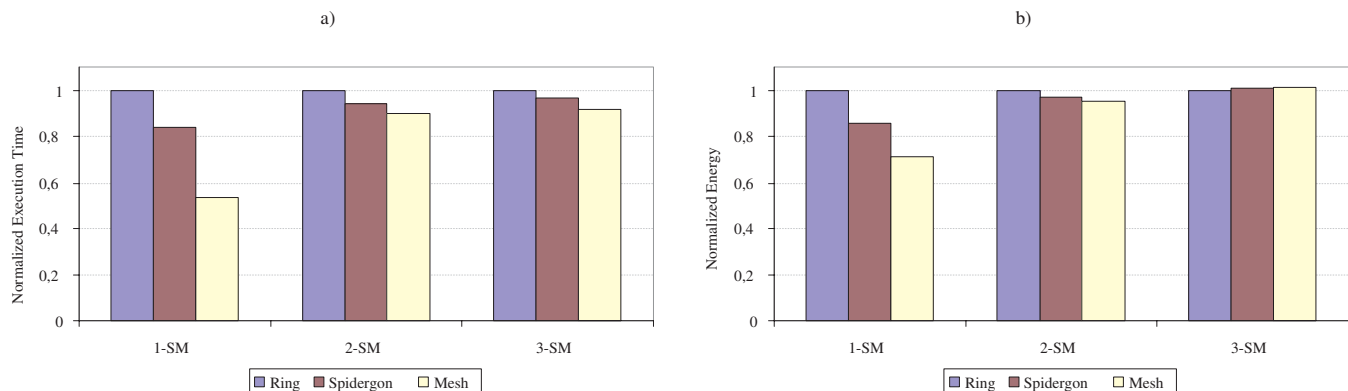


Fig. 6. Normalized execution time and energy for Matrix benchmark by varying NoC topologies and distributed shared memory modules from 1 to 3.

- [4] V. Catalano, M. Monchiero, G. Palermo, C. Silvano, and O. Villa, "GRAPES: A Cycle-Based Design and Simulation Framework for Heterogeneous MPSoC," Politecnico di Milano, Tech. Rep. 45, 2006.
- [5] T. Li and W. H. Wolf, "Hardware/Software Co-Synthesis with Memory Hierarchies," *IEEE Transactions on Computer*, vol. 18, no. 10, pp. 1405–1417, November 1999.
- [6] S. Wuytack, J. da Silva, F. Catthoor, G. de Jong, and C. Ykman-Couvreur, "Memory Management for Embedded Network Applications," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 18, no. 5, pp. 533–544, May 1999.
- [7] R. Banakar, S. Steinke, B. Lee, M. Balakrishnan, and P. Marwedel, "Scratchpad Memory: Design Alternative for Cache on-Chip Memory in Embedded Systems," in *Proc. of CODES*, 2002.
- [8] F. Angiolini, L. Benini, and A. Caprara, "Polynomial-Time Algorithm for on-Chip Scratchpad Memory Partitioning," in *Proc. of CASES: International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*, 2003.
- [9] M. Kandemir and J. R. A. Choudhary, "Exploiting Shared Scratch Pad Memory Space in Embedded Multiprocessor Systems," in *Proc. of DAC*, 2002.
- [10] A. Molnos, M. Heijligers, S. Cotofana, and J. van Eijndhoven, "Compositional Memory Systems for Multimedia Communicating Tasks," in *Proc. of DATE*, 2005.
- [11] "Special Issue in Distributed Shared Memory Systems," *Proceedings of IEEE*, vol. 87, no. 3, pp. 397–532, March 1999.
- [12] E. V. Puttkamer, "A simple hardware buddy system memory allocator," *IEEE Transaction on Computers*, vol. 24, no. 10, pp. 953–957, October 1975.
- [13] P. R. Wilson and et al., "Dynamic Storage Allocation: A Survey and Critical Review," in *International Workshop on Memory Management*, September, 1995, pp. 1–78.
- [14] J. M. Chang and E. F. Gehringer, "A high-performance memory allocator for object-oriented systems," vol. 45, no. 3, March 1996, pp. 106–111.
- [15] W. Srisa-an, C. D. Lo, and J. M. Chang, "A Hardware Implementation of Realloc Function," in *Proceedings of WVLSI'99 IEEE Annual Workshop on VLSI*, April, 1999, pp. 106–111.
- [16] J. M. Chang, W. Srisa-an, and C. D. Lo, "Introduction to DMMX (Dynamic Memory Management Extension)," in *ICCD Workshop on Hardware Support for Objects and Micro architectures for Java*, October, 1999, pp. 11–14.
- [17] M. Shalan and V. Mooney, "A Dynamic Memory Management Unit for Embedded Real-Time System-on-a-Chip," in *Proceedings of CASES: International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*, November 2000, pp. 180–186.
- [18] M. Loghi and M. Poncino, "Exploring Energy/Performance Tradeoffs in Shared Memory MPSoC: Snoop-based Cache Coherence vs. Software Solutions," in *Proceedings of DATE-05: Design Automation and Test in Europe Conference and Exhibition*, 2005.
- [19] A. Sinha, N. Ickes, and A. Chandrakasan, "Instruction level and operating system profiling for energy exposed software," *IEEE Transaction on VLSI*, vol. 11, no. 6, pp. 1044–1057, December 2003.
- [20] P. Shivakumar and N. P. Jouppi, "CACTI 3.0: An Integrated Cache Timing, Power, and Area Model," HP Labs, Tech. Rep., 2001.
- [21] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta, "The SPLASH-2 Programs: Characterization and Methodological Considerations," in *Proc. of ISCA*, 1995. [Online]. Available: citeseer.csail.mit.edu/woo95splash.html
- [22] E. Artiaga, N. Navarro, X. Martorell, and Y. Becerra, "Implementing PARMACS Macros for Shared-Memory Multiprocessor Environments," UPC-DAC, Tech. Rep. UPC-DAC-1997-07, 1997. [Online]. Available: citeseer.csail.mit.edu/artiaga97implementing.html
- [23] M. Coppola, R. Locatelli, G. Maruccia, L. Pieralisi, and A. Scandurra, "Spidergon: A Novel On-Chip Communication Network," in *SOC 2004: International Symposium on System-on-Chip*, November 2004.