Performance and Energy Evaluation of Memory Organizations in NoC-based MPSoCs under Latency and Task Migration

Gustavo Girão, Daniel Barcelos, Flávio Rech Wagner

Federal University of Rio Grande do Sul
Institute of Informatics
Porto Alegre, RS, Brazil
{ggbsilva, danielb, flavio}@inf.ufrgs.br

Abstract. This chapter presents a study on the performance and energy consumption arising from distinct memory organizations in an NoC-based MPSoC environment. This evaluation considers three sets of experiments. The first one evaluates the performance and energy efficiency of four different memory organizations in a situation where a single application is executed. In the second experiment, a traffic generator is responsible for the injection of synthetic traffic into the system, simulating the impact of the parallel execution of additional applications and increasing the latency of the NoC. Results show that, with a low NoC latency, the distributed memory presents better results for applications with low amount of data to be transferred. On the other hand, results suggest that shared and distributed shared memories present the best results for applications with high data transferring needs. In the second set of experiments, with higher NoC latency, for applications with low communication bandwidth requirements, a memory organization that is physically centralized and logically shared (called nDMA) is shown to have a smooth performance degradation when additional traffic rises up to 20% of the network capacity (22% degradation for an application demanding high communication, and 34% degradation for a low communication one). In contrast, a distributed memory model presents 2% of degradation in an application with high communication requirements, when traffic rises up to 20% of the network capacity, and reaches 19% of degradation in low communication ones. Shared and distributed shared memory models are shown to present lower tolerance to high latencies. A third set of experiments evaluates the performance of the four memory organization models in a situation of task migration, when a new application is launched and its tasks must be distributed among several nodes. Results show that the shared memory and distributed shared memory models have a better performance and energy savings than the distributed memory model in this situation. In addition, the nDMA memory model presents a smaller overhead when compared to the shared memory models and tends to reduce the traffic in the migration process due to the concentration of all memory modules in a single node of the network.

Keywords: Multiprocessor-System-on-Chip, Network-on-chip, Memory Organization, Cache Coherence, Task Migration, Performance and Energy Evaluation.

1 Introduction

Nowadays, embedded systems have become very complex. This complexity has many reasons, but the most evident one is the use of such devices for general purpose computing, leading to the execution of many different and complex applications. However, even with higher performance requirements, low power design is still a very desirable goal in portable devices [1].

To support processing requirements and also meet stringent constraints in terms of area and memory, as well as low energy consumption and low power dissipation, a solution using several cores in a single chip is widely adopted. This architecture is known as Multiprocessor System-on-Chip (MPSoC). This scenario usually implies a communication bandwidth between cores that demands a more efficient communication mechanism than a single bus [2]. With this concern in mind, the concept of Network-on-Chip (NoC) has been created.

Considering an MPSoC scenario, memory organization plays a key role since it is not only a major performance bottleneck but also represents a significant component in terms of energy consumption. In addition, memory organization is closely related to the communication model adopted in the application development. For instance, when using a shared memory organization, the communication mechanism usually adopted is the memory itself and, therefore, the memory organization becomes even more important.

Realizing that NoCs are communication structures with high scalability, it is not hard to imagine a situation with dozens or hundreds of processing elements and memory nodes, running a large number of applications concurrently. In this scenario, it is of great interest the evaluation of the behavior of different memory organizations when the network latency increases due to the large number of components and applications in the system. In addition, the memory model also impacts the system performance when a new application is dynamically launched and a task migration mechanism is applied such that a new task allocation is found which better meets system requirements, especially real-time and energy constraints.

This chapter presents a study on the performance and energy consumption arising from distinct memory organizations in an NoC-based MPSoC environment. This evaluation considers three sets of experiments, running on a virtual platform. The first one evaluates the performance and energy efficiency of four different memory organizations in a situation where a single application is executed. In the second set of experiments, a traffic generator is responsible for the injection of synthetic traffic into the system, simulating the impact of the parallel execution of additional applications and increasing the latency of the NoC.

The following memory organizations have been implemented in the virtual platform and evaluated in the experiments: (i) distributed memory, where processors have their local private memories; (ii) shared memory, with a single memory component in a dedicated node on the NoC that is accessed by all processors; (iii) distributed shared memory, composed by several physically distributed memory nodes that share the same address space; and, finally, (iv) a physically shared but logically distributed memory, whose communication model resembles a DMA communication protocol and is thus called nDMA.

Experiments show that, considering the communication requirements of an application, the results of performance and energy consumption may widely vary. For applications with high communication demands, the distributed memory model presents the highest tolerance to communication latency in most situations. However, for applications with low communication workload, the distributed memory model seems to present a larger degradation when NoC latency increases. Also, the nDMA model presents better results as the communication workload decreases. Experiments with the distributed memory model present a variation from 2% to 19% of performance reduction when traffic load rises from 10% to 20%, when using applications with high communication workload and low communication workload, respectively. On the other hand, the nDMA model shows a variation of performance reduction from 22% to 33% in the same situations, which represents a lower relative degradation if compared to the distributed memory model. On the other hand, the shared and distributed shared memory models present low tolerance to high latencies due to the use of a remote memory for communication among tasks.

A third set of experiments show that the shared memory and distributed shared memory models have a better performance and energy savings than the distributed memory model in a task migration situation. In addition, the nDMA memory model presents a smaller overhead when compared to the shared memory models and tends to reduce the traffic in the migration process due to the concentration of all memory modules in a single node of the network.

The remaining of this chapter is organized as follows. Section 2 discusses related work. Section 3 presents the virtual platform used to implement the experiments. Section 4 presents the experimental setup. In Sections 5 and 6, results for experiments with and without additional synthetic traffic, respectively, are presented. Finally, Section 7 draws conclusions and introduces future work.

2 Related Work

Several works regarding memory hierarchy in multiprocessor systems have been developed. However, the majority of these works only consider the use of busses instead of NoCs as communication mechanisms. In such systems, the massive communication parallelism may lead to different side effects due to the memory hierarchy.

Marescaux et al. [3] show a comparison between caches and scratchpads in an NoC-based MPSoC using a distributed shared memory model. In this environment, the use of six DSPs with local L1 caches and a shared L2 cache is considered. Experiments with two NoCs with different QoS methods are presented. The results show that scratchpads have a better performance than caches. However, these experiments do not consider a cache coherence mechanism in hardware. In this case, the adoption of a software coherence conservative approach that invalidates shared data on every access might have led to unnecessary invalidations.

Monchinero et al. [4] explore the use of a distributed shared memory in an NoC-based MPSoC. The platform presents private L1 caches for each core and a shared L2 cache. Each processor has its own address space but there are also several banks of a

distributed shared memory. A hardware MMU manages the shared data allocated to each processor. It is shown that by increasing the number of distributed shared banks the performance also increases but only up to some point where the NoC size leads to a greater latency. It is also shown that the energy consumption drops as the number of distributed shared banks increases.

Enright-Jerger et al. [5] propose a new cache coherence solution for multi-core architectures. The Virtual Tree Coherence (VTC) relies on a virtually ordered interconnection that keeps track of sharers of a coarse grain region. The protocol works in such a way that a virtual tree of nodes that share some region is established and each access to this shared region by any of these nodes leads to a message request to the root node of this virtual tree. The root node requests the data to the node that currently owns them. This request is performed as a multicast message, similarly to conventional snoop implementations. However, those multicast message requests are performed in a tree-like fashion in order to decrease the latency on the network, when compared to a multicast based on sequential unicast messages. The VTC solution is compared to a directory-based protocol and to a greedy-order protocol extended onto an unordered interconnect. VTC presented results 25% and 11% better, respectively, concerning performance. Nonetheless, this work does not present results about energy consumption of those cache coherence solutions.

Although the works presented in this section deal with memory organizations in an NoC-based MPSoC environment, none of them considers a high latency situation.

3 The SIMPLE Virtual Platform

Aiming at an accurate evaluation of the tolerance of memory models to a high latency scenario, four distinct memory models were implemented in the SIMPLE (Simple Multiprocessor Platform Environment) virtual platform. SIMPLE is a SystemC, cycle-accurate virtual platform that emulates an NoC-based MPSoC.

In SIMPLE, each Processing Element (PE) is a multi-cycle Java processor that is a hardware implementation of the Java Virtual Machine [6]. Each instruction takes from 3 to 14 cycles (not including a possible cache miss). To generate the Java bytecodes, a compiler that follows the JVM specification is used. This compiler generates the contents of both the instruction memory and data memory customized for the application. These memories are used as inputs for the simulation in SIMPLE.

The NoC used in SIMPLE [7] implements a wormhole packet switching to reduce energy consumption. It also uses XY routing to avoid deadlock situations and a handshake control flow. Additionally, each router has five bi-directional ports with input buffer size of four phits. The phit size is four bytes.

As already mentioned, the simulator supports distinct memory organizations (distributed memory, shared memory, distributed shared memory, and nDMA memory) and cache configurations, regarding size, replacement policy, associativity, and block size.

For the distributed memory model, each router of the NoC is attached to a PE that, in turn, is attached to private memories (instruction and data). In this configuration,

there are no caches and the communication mechanism uses message passing. Figure 1a depicts an example of this configuration.

The second model is a shared memory. Here, each router is attached to a PE or to a global Data Memory (the placement of each resource in the network is also configurable). Each PE has its own private data cache, while instruction memories are still local. In such environment there is a cache coherence problem. To solve this problem, SIMPLE adopts a directory-based cache coherence solution [8]. This solution centralizes the memory access requests on an entity (the directory) that, based on the state of the block (if it is dirty or clean), makes decisions that could lead to invalidation or write-back requests if the block is dirty. The flowcharts for read and write operations are presented in Figures 2 and 3, respectively. The shared memory in SIMPLE has a hardware-implemented directory, and each request from a cache to the memory is responded by it. This shared memory environment is shown in Figure 1b.

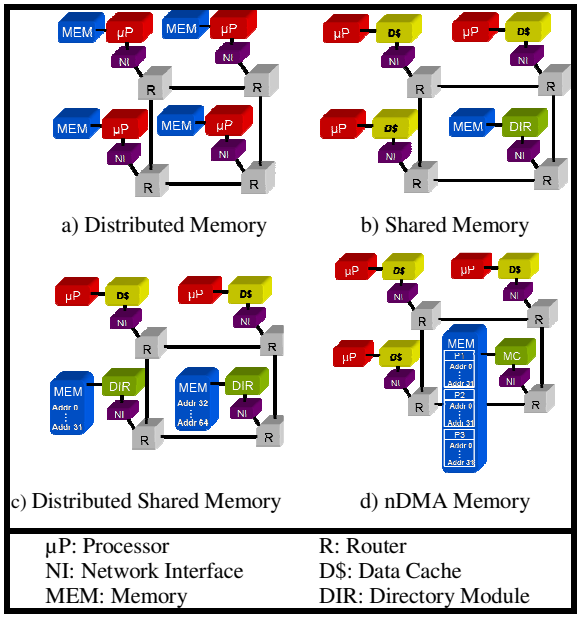

Fig 1. Memory models available in SIMPLE.

The third memory configuration is a distributed shared memory, represented in Figure 1c. In this situation, there can be more than one shared data memory module. However, all of these modules share the same address space. This means that, if a memory module ends with the address N, then some other module in the system begins with the address N + 1. In this scenario, there are also private data caches for each PE and all global memory modules have their own directory module to maintain coherence.

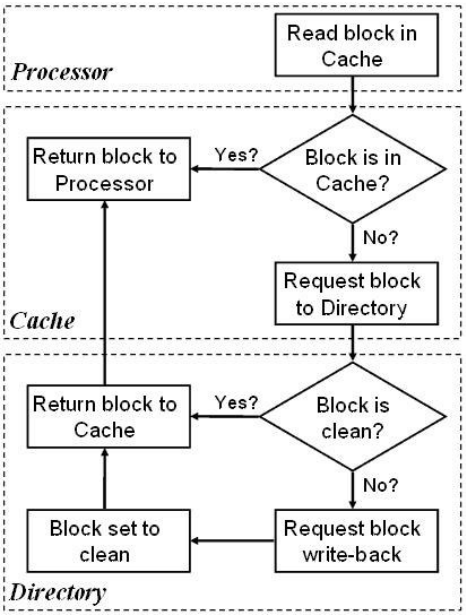

Fig. 2. Read operation using directory.

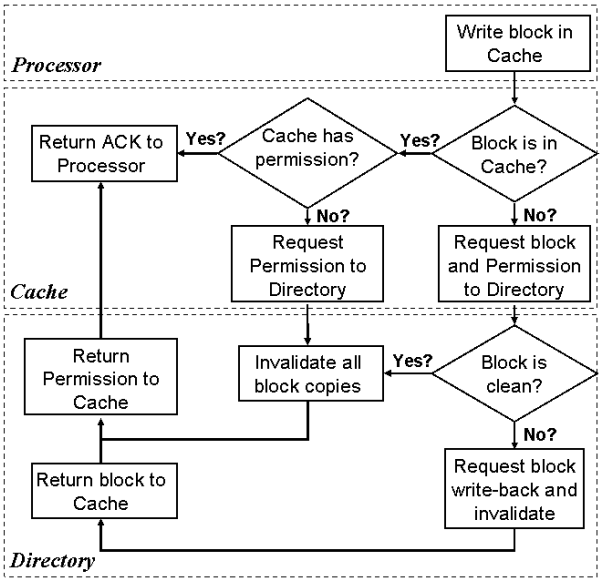

Fig. 3. Write operation using directory.

For each router on the NoC there is a Network Interface. In routers with a PE there is also a component called Memory Access Handler (MAH) that receives the signal from the processor when it needs data. The MAH module acts as an interface between the cache and the network interface when there is a cache miss. It creates a message to be sent to the directory module requesting a block. When the data are ready they are sent to the PE by the MAH module.

In both shared and distributed shared memory organizations the communication between the PEs is performed by means of shared variables protected by mutexes. The operations down and up in a mutex are made through test-and-set and test-and-reset operations, respectively, supported by the hardware (caches and directory).

The fourth memory model available in SIMPLE is the nDMA organization, which implements a physically shared and logically distributed memory model, as depicted in Figure 1d. In this model the data memory is placed in an exclusive node, as in the shared memory model. This memory node, however, has N banks, each of them storing the memory data of each of the N PEs. Each memory bank has its own address space, and no PE can access the data of another PE. In addition, each PE has private data caches. When there is a miss, the cache sends a request to the memory node, where a Memory Controller (MC) receives the message, identifies the PEs that is requesting data, and, based on that, accesses the memory bank that hold its memory data.

A direct consequence of having different address spaces is that the most intuitive communication method to be used is message passing. Message passing is essentially implemented sending data from a processor (from its own memory, to be more precise) to another one through the physical communication mechanism. However, in the nDMA model, all memories are centralized in the same node, and, therefore, there is no need to send data through the network. Hence, the message passing method must be modified to work in such environment.

Basically, the processor that wishes to send a message must send a copy request message to the memory (instead of sending it to the target processor) informing an address source and the amount of bytes to be copied. In the memory node, the memory controller identifies this message and sends a message to the target processor informing the intention of the source processor to send data to it. The target processor replies with the address destination for that specific communication. When this reply message arrives at the memory node, the memory controller starts copying from one memory bank (containing the data memory of the source processor) to another one (containing the data memory of the target processor). Figure 4 illustrates this message passing mechanism.

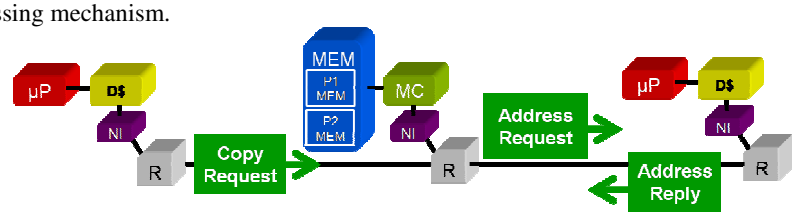

Fig. 4. nDMA message passing mechanism.

Since the memory node needs to be, at some level, programmed by the source processor to copy bytes from one address to another, the memory controller works similarly to a DMA, and, therefore, this memory organization is called nDMA (standing for NoC DMA).

Considering that each processor has its own private data cache, at the moment of sending the copy request the source processor must perform a forced write-back operation of the blocks inside the range of data that it is trying to send. However, the cache performs this write-back operation only on the blocks that have been modified. In a similar way, the target processor must also perform forced write-back operations on the blocks inside the range of the target addresses for that message, before the data are exchanged. Furthermore, the cache of the target processor must invalidate the blocks modified by the data exchange. These are the only moments during the entire communication process when data are exchanged through the network.

4 Experimental setup

The experiments consider four applications: a matrix multiplication, a motion estimation algorithm, a Mergesort algorithm, and a JPEG encoder.

The Matrix Multiplication was parallelized in such a way that each processor multiplies a subset of lines of matrix A by a subset of lines of matrix B. Each matrix used in simulations has 32 x 32 elements.

In the Motion Estimation, every PE searches a macroblock (a subset of an image) in a different part of the reference image. In the simulations, a macroblock of 8x8 pixels and an image in QCIF format (176x144 pixels) have been used.

For the Mergesort, the parallelism took advantage of its divide-and-conquer nature. Initially, each PE performs the Mergesort on a subset of the vector. Afterwards, one PE is responsible for assembling the whole vector, using the subsets already ordered by the various PEs.

A JPEG encoder can be seen as a three step algorithm. The first two steps (2-D DCT and Quantization) can be performed in parallel for different parts of the image. However, the third step (Entropy coding) can only be correctly performed with the whole image. Based on that, this parallel approach divides an image of 32x16 pixels in eight 8x8 blocks, and each PE is responsible for executing the first two steps on an equal amount of those eight blocks. At the end of those steps, each PE sends the resulting blocks to a master PE, which performs the final step with the complete image.

Considering the data inputs for the applications described above, Figure 5 represents their communication workload regarding different numbers of processors. Based on this chart, it is expected that the Motion Estimation algorithm will generate a larger amount of data exchanges.

Three kinds of experiments were performed. The first one evaluates the different memory organizations on an environment executing only a single parallelized application, with no addition of synthetic traffic. The second set of experiments investigates the behavior of the same memory organizations with the addition of

different synthetic traffics, thus simulating the execution of several concurrent applications. Experiments have been performed for different numbers of processors and for different cache sizes. The third set evaluates the overhead in the system caused by a task migration. The task migration mechanism adopted in the experiments is quite simple. It is performed in three distinct steps that are performed sequentially by the processor were the task is located. The three steps correspond to the transmission (and proper receiving) of program code and data memory contents. Of course, the transmission of the data memory is not required when the shared data model is used. Also, because of the fact that the experiments assume a scenario where the tasks are recently created, there is no meaningful stack contents and therefore no need to migrate it.

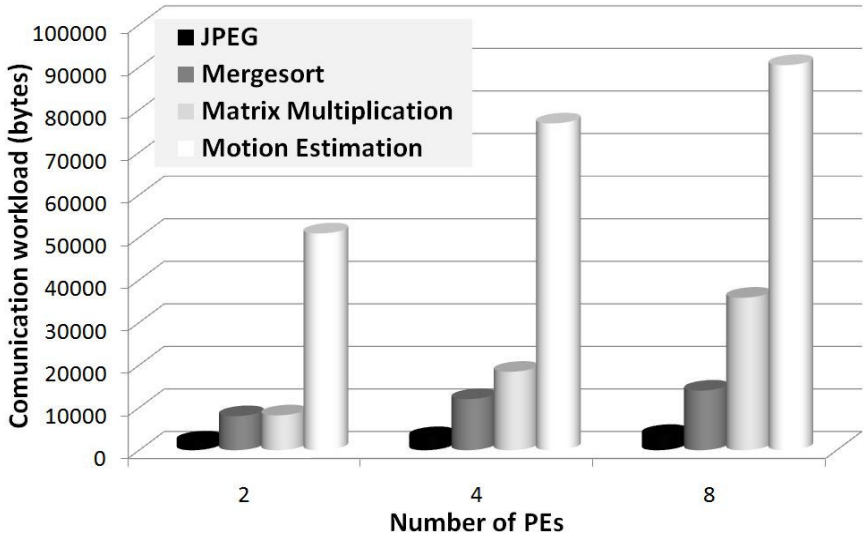

Fig. 5. Communication workload.

Experiments evaluate two characteristics: performance, measured by the total execution time of each application, and the overall dynamic energy spent, including processors, network, and memories. For the energy of the processors, a cycle-accurate power simulator [9] is used. For the network (including buffers, arbiter, crossbar, and links), the Orion library [10] is applied, and for the memory and caches the Cacti tool [11] is used. In all simulations, the processors and the NoC operate at 100 Mhz and the technology considered was 0.18µm. At this technology node, the static energy consumption is negligible.

5 Experiments without Synthetic Traffic

5.1 Performance

Concerning performance, Figures 6 thru 9 show the overall execution time of each application, measured in millions of cycles. For each memory organization, there are three columns representing different cache sizes – 256, 512, and 1024 bytes in the Motion Estimation, Mergesort, and JPEG simulations, and 1024, 2048, and 4096 bytes for the Matrix Multiplication.

For applications with low communication workload (such as Matrix Multiplication, Mergesort, and JPEG), the distributed memory model presents better results than all other models. As for the Motion Estimation algorithm, the chart shown in Figure 6 indicates that the distributed memory does not present an overall result better than the other organizations. This is due to the fact that this algorithm requires a large number of data exchanges between processors, as depicted in Figure 5.

The nDMA memory presents very similar results if compared to the other two shared memory models. For the Mergesort and JPEG application (Figures 8 and 9, respectively), the nDMA presents even better results, especially considering more realistic MPSoC sizes, as with eight PEs. Considering the other applications (like the Matrix Multiplication depicted in Figure 7), the distributed shared memory shows a slightly superior performance, especially with eight processors. This enhanced performance is due to the parallel memory accesses performed by the several PEs. However, this advantage does not increase proportionally to the number of memory modules because not all of the modules have the same access profile. This means that some module may concentrate more accesses than others, and, hence, the performance speed up of this solution is not so high.

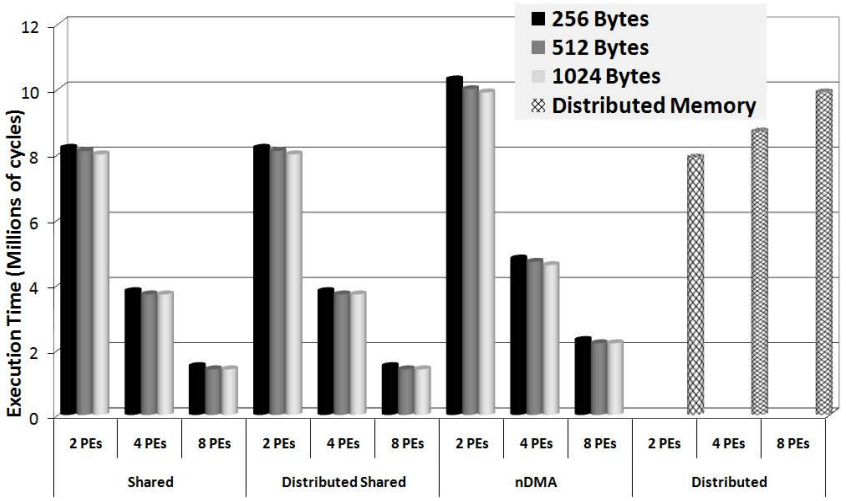

Fig. 6. Performance for Motion Estimation.

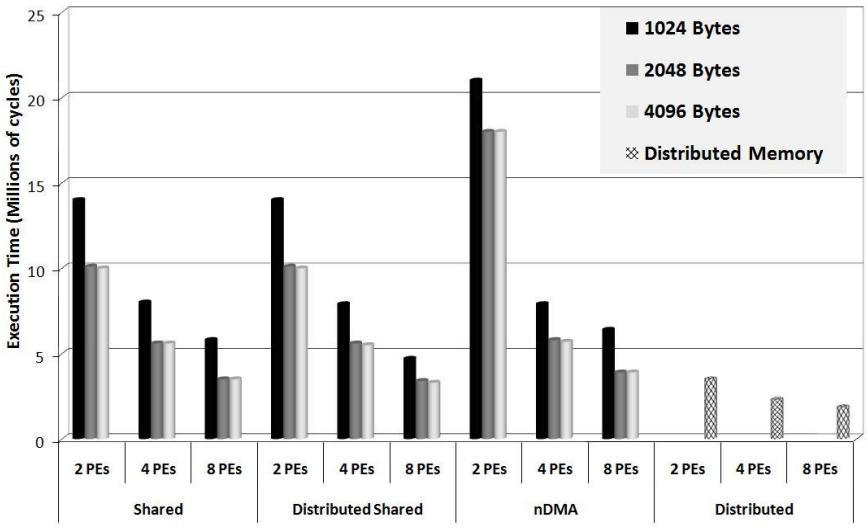

Fig. 7. Performance for Matrix Multiplication.

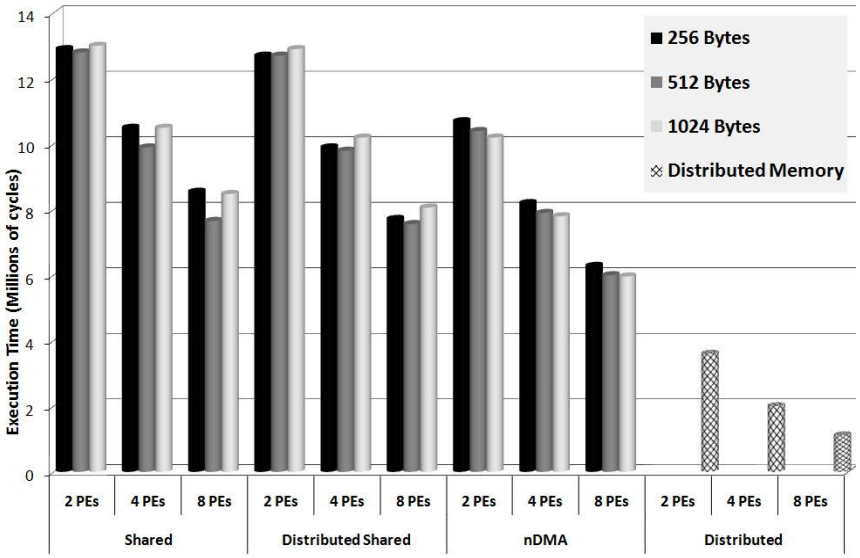

Fig. 8. Performance for Mergesort.

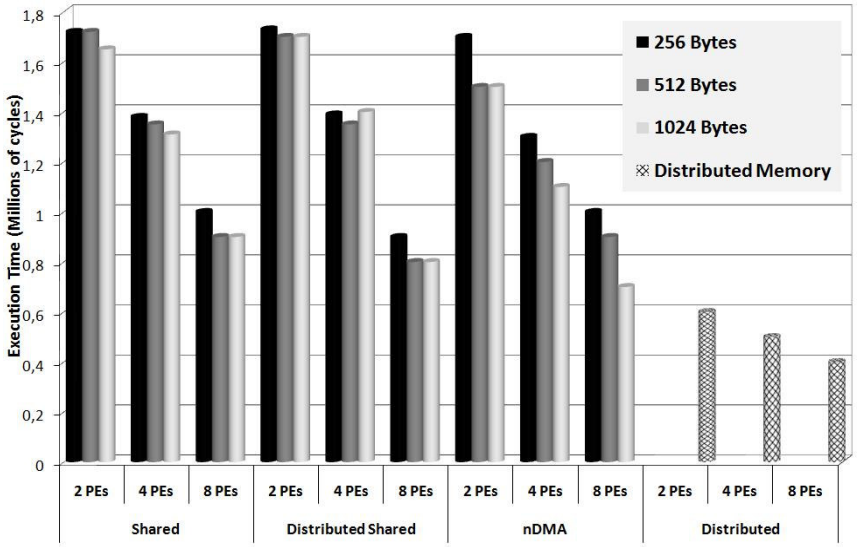

Fig. 9. Performance for JPEG.

5.2 Energy Consumption

Concerning dynamic energy, Tables 1 thru 4 show the consumption for the four applications. For each application, there are four columns representing the energy consumption (considering an average between the cache sizes) for the Processors, Caches, Memories, and NoC for eight PEs.

Analyzing the energy consumption difference between the nDMA solution and the other two shared memory organizations, it is possible to see that the NoC consumption is much different. This behavior is a consequence of the fact that the nDMA memory uses small control messages for communication and does not need cache coherence messages. The use of a cache coherence solution in the shared and distributed shared memories also leads to invalidations that increase the memory accesses (in order to retrieve the block again) and the cache accesses. Therefore, the energy consumption of those components is also higher.

As depicted, in the distributed memory model the processor consumption is the major responsible for the energy consumption in the system. The NoC and sometimes also the memory energy are negligible in this environment. The memory energy is more significant in the Motion Estimation simulation due to the amount of data to be manipulated.

Table 1. Energy consumption (mJ) without synthetic traffic for Motion Estimation.

Memory Model	Motion Estimation				
	µP	***D$***	***Mem***	***NoC***	***Total***
Distributed Memory	3180 87%	--	452 12%	4 0.1%	3634 100%
Shared Memory	668 79%	137 16%	13 2%	25 3%	843 100%
Distr. Shared Memory	667 80%	136 16%	10 1%	24 3%	837 100%
nDMA	866 75%	183 16%	96 8%	12 1%	1157 100%

Table 2. Energy consumption (mJ) without synthetic traffic for Motion Estimation.

Memory Model	Matrix Multiplication				
	µP	***D$***	***Mem***	***NoC***	***Total***
Distributed Memory	288 54%	--	243 45%	2 0.5%	533 100%
Shared Memory	919 41%	1214 54%	21 1%	103 4%	2257 100%
Distr. Shared Memory	919 41%	1214 54%	14 0.6%	83 4%	2230 100%
nDMA	1342 44%	1628 54%	38 1%	24 1%	3032 100%

Table 3. Energy consumption (mJ) without synthetic traffic for Mergesort.

Memory Model	Mergesort				
	µP	***D$***	***Mem***	***NoC***	***Total***
Distributed Memory	408 70%	--	173 29%	2 0.5%	583 100%
Shared Memory	1260 54%	479 20%	122 5%	472 21%	2333 100%
Distr. Shared Memory	1264 56%	484 21%	60 3%	440 20%	2244 100%
nDMA	866 73%	238 20%	45 4%	42 3%	1191 100%

Table 4. Energy consumption (mJ) without synthetic traffic for JPEG.

Memory Model	JPEG				
	µP	***D$***	***Mem***	***NoC***	***Total***
Distributed Memory	48 63%	--	27 36%	1 1%	76 100%
Shared Memory	98 54%	27 15%	10 6%	45 25%	180 100%
Distr. Shared Memory	99 56%	28 16%	8 4%	43 24%	178 100%
nDMA	129 77%	31 18%	5 3%	3 2%	168 100%

For the two shared memory organizations and the nDMA model, the processor also plays a key role in the overall energy consumption. However, in the case of Matrix Multiplication, because of the larger caches, these components present a very significant energy consumption as well. In fact, for the shared memory organizations, the cache energy increases in a non-linear fashion as the cache size increases. As opposed to the increase of the cache sizes, the NoC energy decreases, and, except for the smallest cache size, one can say that the NoC is not a key factor in this case. The same can be said about the memory.

As an overall analysis, the energy consumption results tend to follow the performance results, and, therefore, the distributed memory presents better results for the low communication workload applications (Matrix Multiplication, Mergesort, and JPEG) and worst results for the Motion Estimation.

6 Experiments with Synthetic Traffic

The physical communication mechanism of a multiprocessor system directly affects the memory model adopted. This is due to the fact that the memory model leads to a certain communication model, which, in turn, is influenced by the physical communication mechanism. Based on that, this study tries to emulate an environment with high latency and quantify its impact on the system for different memory models.

In order to emulate such environment, a traffic generator was developed. The general idea is to create a synthetic traffic to increase the latency of the communication mechanism, thus emulating the parallel execution of several applications that generate communications among the various processors.

The Traffic Generator is placed inside the network interface (which is present in every node of the system), as depicted in Figure 10, and a state machine coordinates the inclusion of a traffic package in the Send Buffer. In addition, making use of an identifier in the header of the package, the traffic generator analyzes the incoming packages in the Receive Buffer to exclude any synthetic traffic package. This avoids the resource associated to the Network Interface (PE or a memory) from reading and processing it.

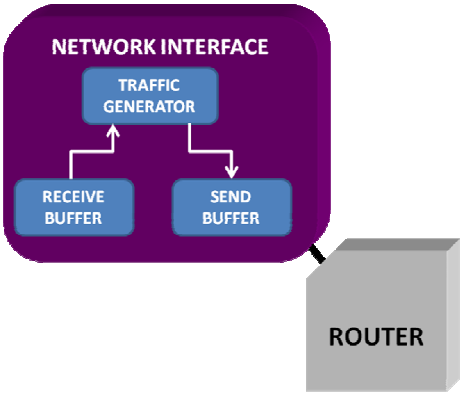

Fig. 10. Traffic Generator.

The traffic generator woks in a very simple fashion. At every period (previously configured), it creates a 10-byte package to be sent to a destination node determined through a round-robin sequence of all routers in the system, one at a time. Each traffic package has a 10-byte size in order to guarantee that the network interface is able to send a package in only one cycle.

With this round-robin system, at every 10 cycles router 0 in the system sends a synthetic traffic packet to router 1, router 1 sends a packet to router 2, and so on. In the next period, router 0 sends a packet to router 2, router 1 sends a packet to router 3, and so on. Therefore, at a certain time, each router sends a synthetic traffic packet to a different destination. On the other hand, when these packets arrive at their final destinations, the Traffic Generator removes them from the Receive Buffer.

The experiments use 10% and 20% traffic loads, meaning that at every 10 cycles or 5 cycles, respectively, each router in the system sends a 10-byte traffic packet to a different router. Results indicate by how much the performance decreases (or the energy consumption increases), when the traffic load increases from 10% to 20%.

This Traffic Generator creates a traffic that is uniformly distributed both in time and in space. As future work, more complex generators (as in [12, 13]) will be used, to assess the possible influence of the traffic model on the experimental results.

6.1 Performance

Considering the summarized results of reduction in performance depicted in Table 5, it is possible to see how the increase in the latency impacts the performance.

Although the distributed memory presents better results when compared to the other memory models, it is very clear how much the higher latency impacts the performance as the communication workload of an application increases. In the Motion Estimation application, a higher latency only affects the distributed memory results in 2%. However, when an application with less communication workload (the JPEG) is executed, the impact of a higher latency reaches up to 19%.

The shared memory and distributed shared memory models seem to suffer less from this communication workload variation, although, in absolute numbers, the

overall decrease of performance is higher than in any other memory model. On the other hand, the nDMA model suffers less as the latency increases, and its absolute results are in the middle between the shared memory models and the distributed one. This is mainly due to the necessity of accessing a remote memory even though small control messages are used.

These results suggest that a higher latency in the NoC affects much more the distributed memory model than the other models, as the communication workload of an application decreases. Furthermore, the nDMA model seems to have results with less variation as the communication workload of an application decreases.

Table 5. Impact of NoC latency on performance.

Application	Motion Estimation	Matrix Multiplication	Mergesort	JPEG
Distributed Memory	2%	4%	6%	19%
Shared Memory	29%	40%	40%	44%
Dist. Shared Memory	33%	48%	49%	55%
nDMA	22%	33%	33%	33%

Note: values in the table indicate how much the performance is reduced when traffic load increases from 10% to 20%.

6.2 Energy Consumption

Again, the pattern present on the performance results also appears in the dynamic energy consumption. These results suggest that applications with high communication requirements seem to have a smaller reduction in the energy consumption as the latency increases.

In the same way as for the performance results, it is possible to see that a higher latency impacts the energy consumption of the memory models differently, depending on the communication workload characteristics of the various applications. Results in Table 6 show that the distributed memory model seems to suffer more in a high latency situation if the communication workload of the application is low. Again, the shared memory and distributed memory organizations seem to suffer more than other models, whereas the nDMA model presents a graceful degradation if the communication workload of the application is low.

Table 6. Impact of NoC latency on energy consumption.

Application	Motion Estimation	Matrix Multiplication	Mergesort	JPEG
Distributed Memory	1%	10%	14%	32%
Shared Memory	25%	50%	65%	67%
Dist. Shared Memory	44%	70%	77%	79%
nDMA	15%	24%	24%	24%

Note: values in the table indicate how much the energy consumption increases when traffic load increases from 10% to 20%.

7 Experiments with Task Migration

This section presents experimental results regarding the impact of the distinct memory models on task migration, considering the performance on the migration execution and the dynamic energy spent on it. Experiments quantify the task migration overhead in each model. Since all memory models consider that each PE has its own private instruction memory, all of them demand the transmission of program code during task migration, as well as of data memory, depending on the model. The experiments consider a worst case scenario, corresponding to the creation of a new application with N tasks in a single node and the following migration of these tasks to other nodes. Considering that N represents the number of processors, each processor has exactly one new task assigned to it after the migration. In this case, the new task has no data on the stack and therefore, it is not necessary to migrate its contents.

7.1 Performance results

According to the performance results presented in Figures 11 thru 14, the distributed memory model presents worst results in all cases. This was already expected due to the fact that the distributed memory model demands the migration of the whole data memory. This situation does not happen in the case of shared and distributed shared memories. In the case of the nDMA model, due to the fact that it is essentially a shared memory model, in a sense that all data (from all processors) is located in a single node, the copy can be made simultaneously to the program code migration. In addition, in the nDMA memory model there is no traffic overhead for the data memory transfer during migration.

However, these are not the only factors responsible for the worse performance of the distributed memory model. Another factor is the size of the code, which, in the case of distributed memory, is usually higher than in other models. This is due to the fact that communication needs to be completely explicit in the code. The programmer has to describe the copying of data to sending buffers and the reading of data from receiving buffers, as well as calls for these functions. This procedure is less intense in the case of the nDMA memory model, which, although not requiring the transmission of the contents of the data memory, works by sending explicit messages to control communication between the processing elements. On the other hand, the shared memory and distributed shared memory models present the best performance in all cases. The fact that only the application code contents is copied is not the single reason for that, but also the simplicity of the code in the shared memory models.

Despite the better results of the shared and distributed shared memory models, it is important to note that the migration of the contents of the caches was not taken into account in these experiments. Therefore, a loss of performance may be expected during the initial stages of task execution after its migration in these cases, due to a considerable amount of compulsory cache misses. This case has not been considered in the experiments since the migration occurs at the time of the creation of the task and so the caches contain virtually no information relevant to that task. However, if one takes into account a migration during the execution of this task, the compulsory cache misses will occur. This is a situation whose outcome depends on the point in

time within the execution of the task in which the decision is taken to perform the migration. This choice is a function of the task allocation algorithm and was not considered in the experiments, which evaluate only the mechanism of task migration itself.

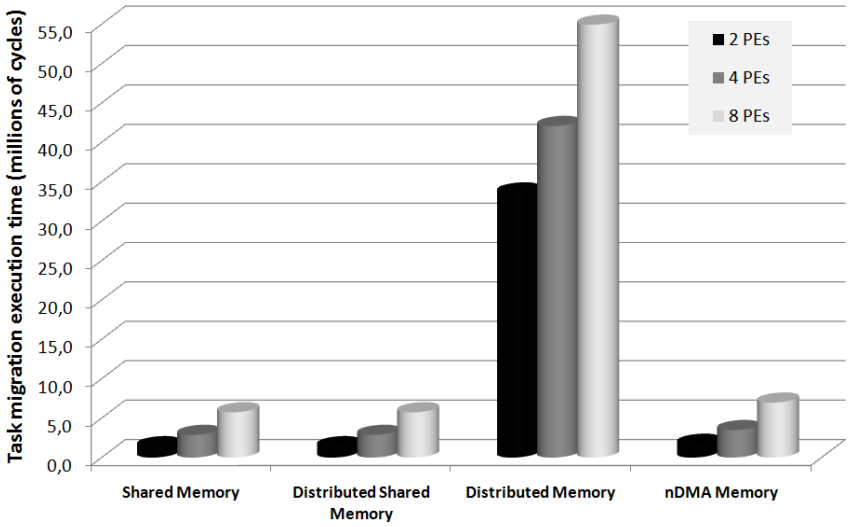

Fig. 11. Task migration performance for the Motion Estimation.

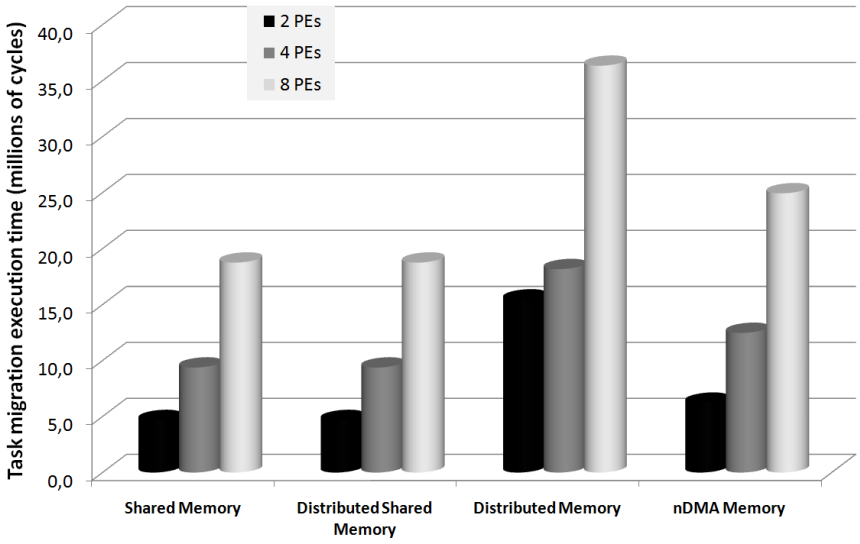

Fig. 12. Task migration performance for the JPEG.

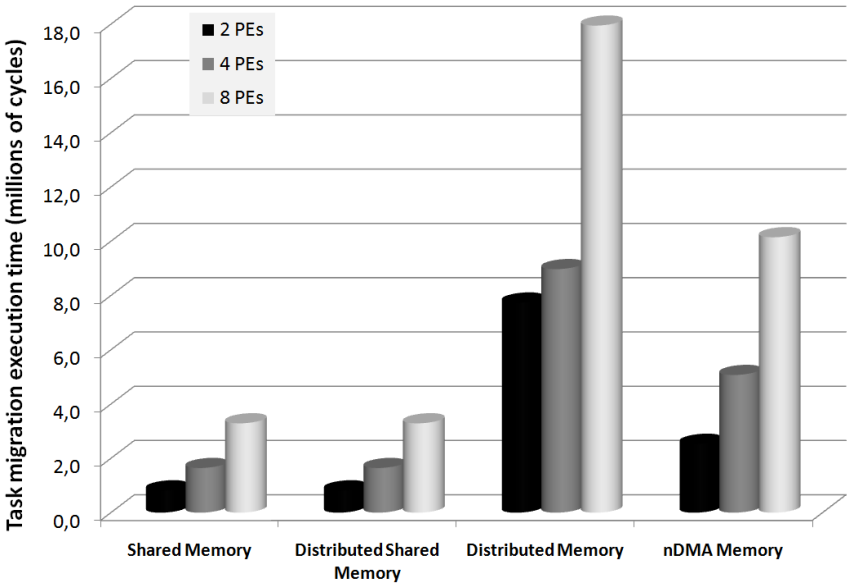

Fig. 13. Task migration performance for the Matrix Multiplication.

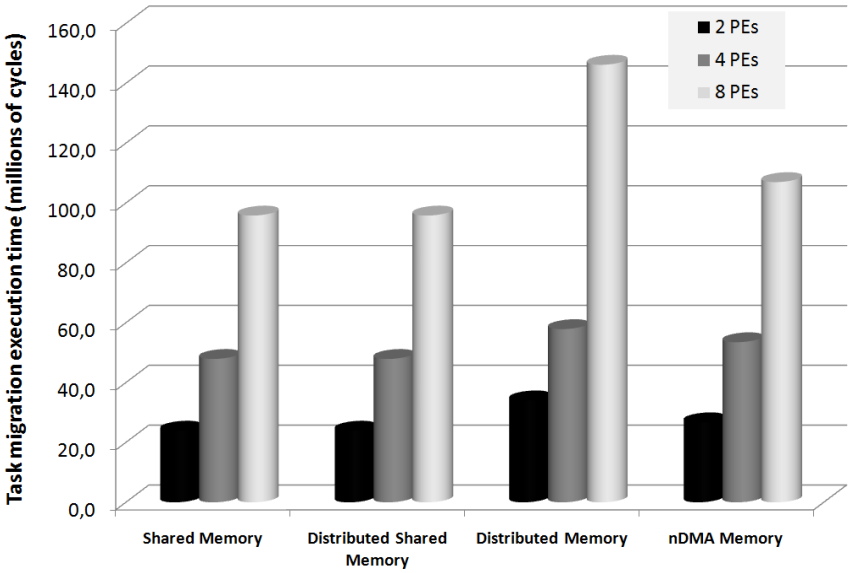

Fig. 14. Task migration performance for the Mergesort.

7.2 Energy consumption results

Regarding the dynamic energy consumption, results shown in Figures 15 thru 18 demonstrate the same pattern presented in the previous performance results.

Among the components of the system, it is noticeable that the NoC has an extremely low energy consumption during the migration process when compared to the energy consumption of processors and memory. In particular, the migration process appears to be very expensive for the processor that needs to execute the whole process of sending and receiving messages and perform the routines of reading and writing on program and data memories (in the case of distributed memory).

Again, the results point to a worse result for the distributed memory, due to the reasons already discussed in the previous section. The shared and distributed shared memories are the most efficient ones, given the smaller size of code to be read and written in memory (thus reducing the number of memory accesses). This also reduces the execution time of the processor performing this task, which leads to a lower power consumption.

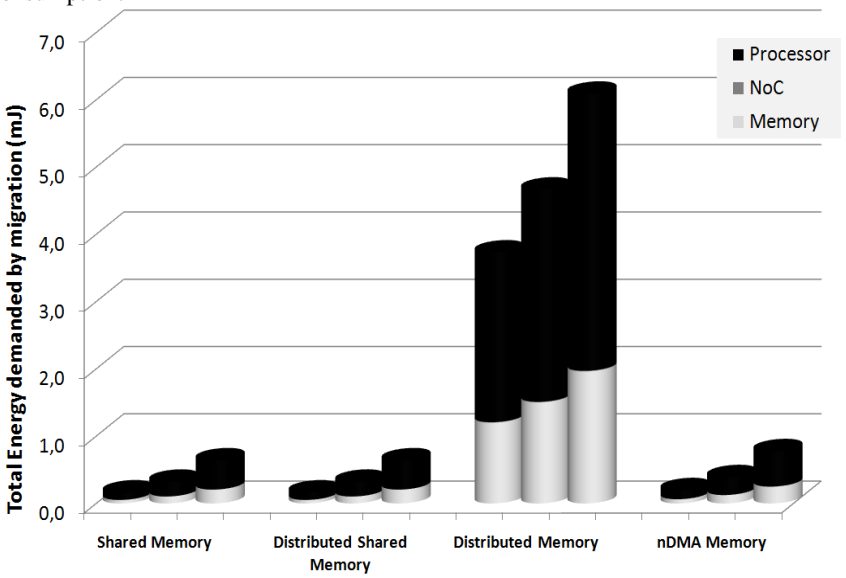

Fig. 15. Task migration energy consumption of the Motion Estimation.

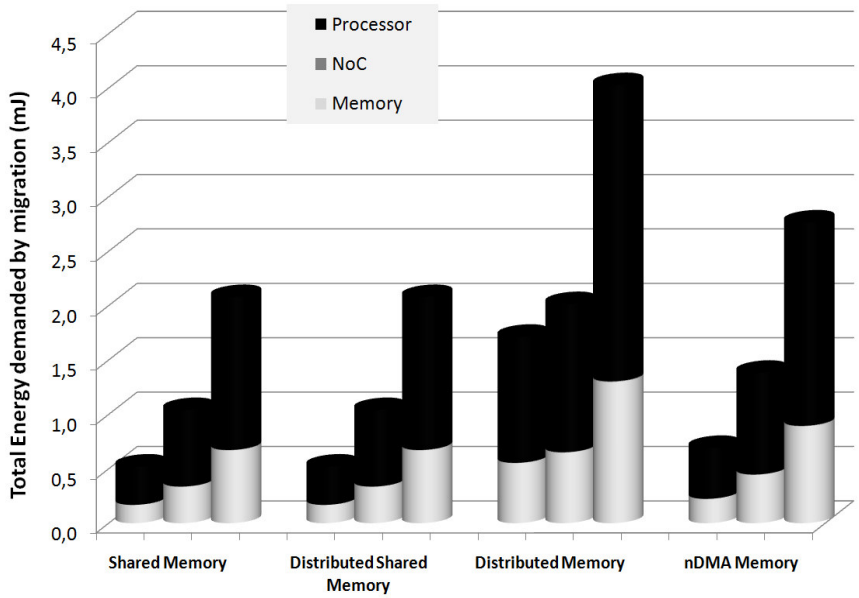

Fig. 16. Task migration energy consumption of the JPEG.

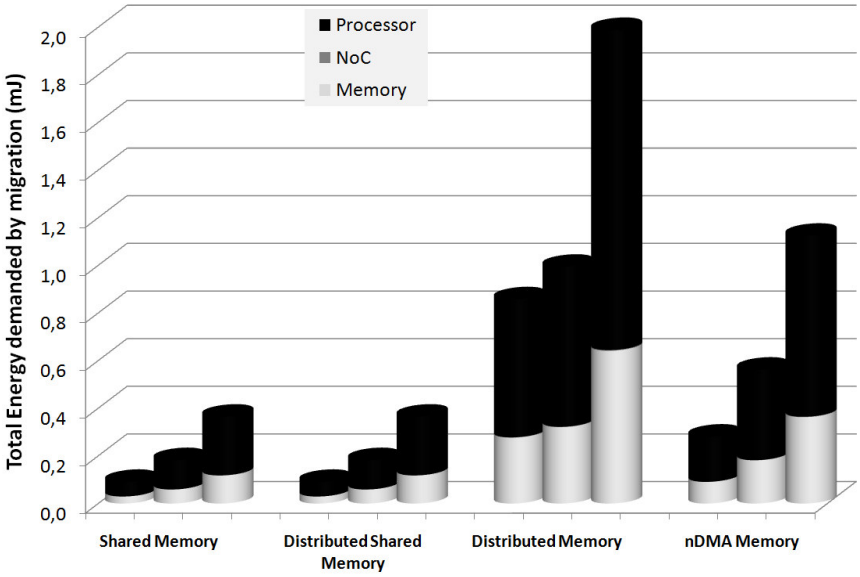

Fig. 17. Task migration energy consumption of the Matrix Multiplication.

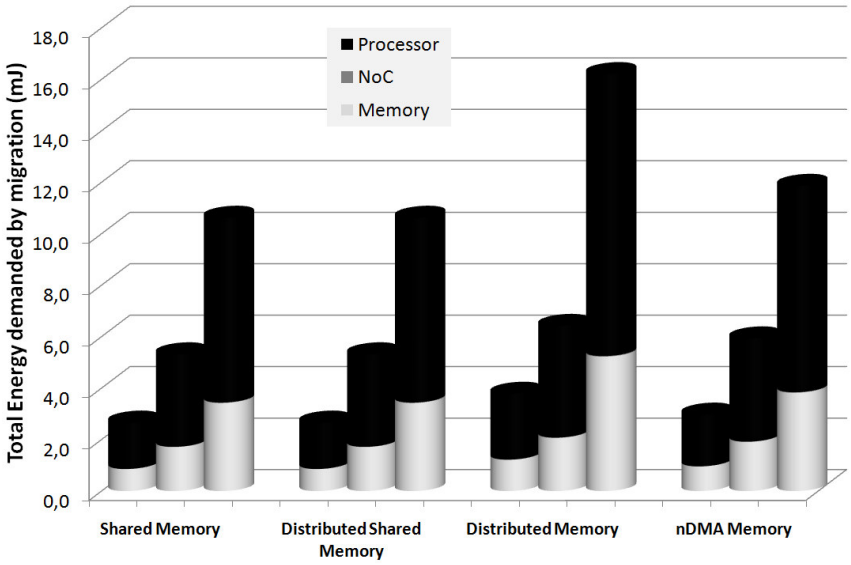

Fig. 18. Task migration energy consumption of the Mergesort.

8 Conclusions and Future Work

This chapter presented a study on the tolerance of different memory organization models under high latency NoC scenarios and also the impact of these memory models in a task migration situation. The experiments considered not only different memory models and traffic loads but also different numbers of processors and cache sizes and various applications with different bandwidth requirements.

The experiments suggest that the distributed memory model presents a higher tolerance to communication latency in most situations. The nDMA model also presents some tolerance and presents better results as the communication workload increases. The high level of communication requirements on shared and distributed shared memory leads to the worst results as the synthetic traffic increases. This is due to the fact that not only network communication is required to retrieve application data but also the communication among tasks is performed through a remote memory.

Regarding dynamic energy consumption, it is possible to conclude that the most affected component of an MPSoC based on a high latency NoC is the NoC itself, while other components such as memory and cache are not affected, due to the fact that the number of accesses does not change and thus neither the dynamic energy consumption.

Although the distributed memory model presents better results in most situations, there is a pattern that shows that, for applications with low communication workload, it suffers more with higher latencies on the NoC. On the other hand, the nDMA

memory model presents results with lower degradation as the communication workload of an application is low.

The experimental results also suggest that the distributed memory model is the one in which the task migration is more expensive. These experimental results were expected due to the simple fact that the distributed memory needs to send task data explicitly through the network, which is not true for the other models. In the case of shared and distributed shared memories the fact of having a global memory limits the amount of information to be transferred in a migration to the program memory. In the case of the model nDMA the situation is very similar. The difference occurs because the data memory copy is performed locally on the node where the memory banks are located, making it a quicker process, independent of the network. Another factor that explains why the shared and distributed shared memories obtain better results is the fact that the application code in such cases is simpler than in the distributed memory model, where all communication must be specified explicitly.

Future work includes more experiments considering other applications and synthetic traffics with more complex distributions, as well as the proposal and evaluation of an optimized task migration model for each memory organization, also considering high latency situations. The evaluation of the static energy consumption for more recent technology nodes must also be considered.

An ongoing work will propose a cluster scenario for an MPSoC with dozens or even hundreds of processors. According to the results presented in this chapter, it was possible to see that different applications have distinct memory hierarchy demands, and, therefore, an MPSoC with different memory organization solutions in the same chip, in order to address the needs of different applications running concurrently, makes sense. The ongoing work relies on these observations to build a system with multiple clusters on the NoC, where each cluster may have a distinct memory organization. Thus, it is possible to take advantage of parallelism at application and task level. Figure 19 gives an insight on how this MPSoC could be. In the figure, there are three applications allocated in the MPSoC and the processors running tasks from the same application build a single cluster. As the parallelism of an application changes during the execution [14], the task allocation can be dynamic, making the cluster size also dynamically modifiable, thus favoring another application that may need more processing or memory resources.

Each of these clusters makes use of a memory organization that provides a better performance for the task (or set of tasks) being performed on it. To accomplish the communication between clusters it is necessary to provide a component that builds a bridge between the memory models. This component could be centralized, so that all inter-cluster communication would be sent to a single component in the system. Another solution would provide components for inter-cluster communication within each cluster, in a manner similar to a router, but at a higher level of abstraction. These components would have to perform communication in accordance to the memory organization adopted by the cluster (message passing, shared variable, etc).

In addition, as future MPSoCs tend to provide adaptability, the memory hierarchy can be also dynamic in a sense that each memory node (i.e. a node that holds a general memory structure) can be configurable in terms of a distinct memory organization. Hence, the memory node can become a cache or a main memory,

leading to a shared memory or to a distributed memory model, respectively. In order to do that, it is also required a software layer to identify the application needs.

Another objective of this future work is the support of consistency of data between different models of memory organization, as proposed in [15]. This problem can be seen as a more expanded version of the problem of consistent caches. However, in this case it is intended to maintain consistency between the memory hierarchies of several clusters. Again the solution to this problem is likely to be centralized, creating a template directory that keeps a record of the location of all data in the system. An alternative solution is the adoption of a hierarchical system of directories. In this case, each cluster contains a directory that centralizes data consistency for the cluster itself and responds directly to a central directory that maintains the consistency of data at a macro level.

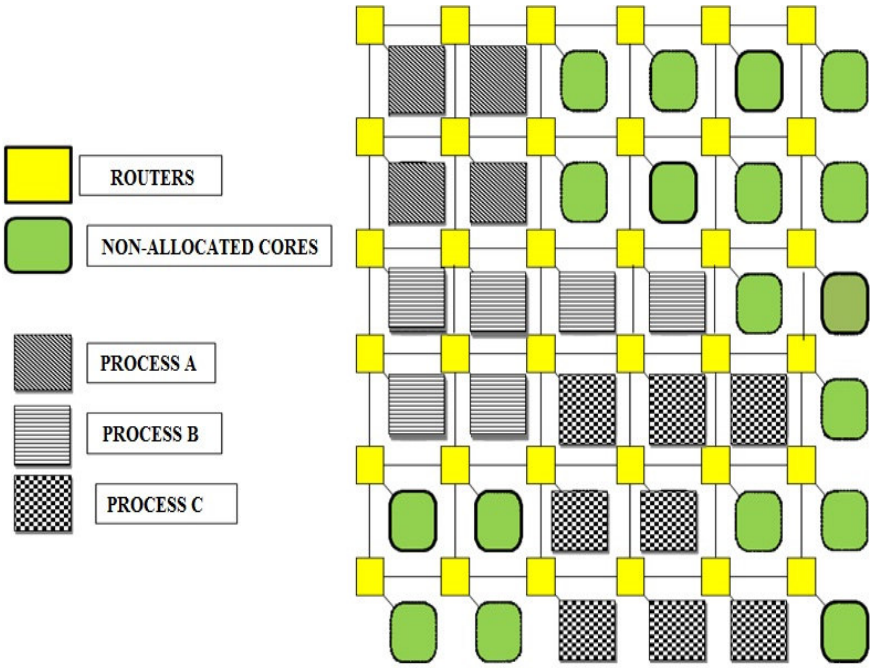

Fig. 19. Cluster allocation in a large MPSoC.

References

1. P. Marwedel, Embedded System Design, Kluwer Academic Publishers, 2003.

2. H.G. Lee, N. Chang, U.Y. Ogras, and R. Marculescu, “On-Chip Communication Architechture Exploration: a Quantitative Exploration of Point-to-Point, Bus and Network-on-chip Architectures”, ACM Transactions on Design Automation of Eletronic Systems, Vol. 12, pp. 21-40, 2007.
3. T.Marescaux, E. Brockmeyer, and H. Corporaal, “The Impact of Higher Communication Layers on NoC Supported MPSoCs”, Proceedings of the First International Symposium on Networks-on-Chip, pp. 107-116, May 2007.
4. M. Monchiero, G. Palermo, C. Silvano, and O. Villa, “Exploration of Distributed Shared Memory Architectures for NoC-based Multiprocessors”, Proceedings of the International Conference on Embedded Computer Systems: Architectures, pp. 144-151, July 2006.
5. N. Enright-Jerger, L.-S. Peh, and M. Lipasti, "Virtual Tree Coherence: Leveraging Regions and In-Network Multicast Trees for Scalable Cache Coherence”, In Proceedings of 41st International Symposium on Microarchitecture (MICRO), Lake Como, Italy, November 2008.
6. S.A. Ito, L. Carro, and R.P. Jacobi, “Making Java Work for Microcontroller Applications”, IEEE Design & Test of Computers, Vol. 18, pp. 100-110, September 2001.
7. C.A. Zeferino, M.E. Kreutz and A.A. Susin. “RASoC: a Router Soft-core for Networks-on-Chip”, Proceedings of Design, Automation and Test in Europe Conference and Exhibition, 2004. Washington, DC: IEEE Computer Society, 2004. pp. 198-203.
8. G. Girão, B.C. de Oliveira, R. Soares, and I.S. Silva, “Cache Coherency Communication Cost in a NoC-based MPSoC Platform”, Proceedings of 20th Symposium on Integrated Circuits and Systems Design, 2007, Rio de Janeiro. New York, NY: ACM, 2007. pp. 288-293.
9. A.C.S. Beck Filho, J.C.B. Mattos, F.R. Wagner, and L. Carro”, CACO-PS: a General purpose Cycle-accurate Configurable Power Simulator”, Proceedings of 16th Symposium on Integrated Circuits and Systems Design, 2003, São Paulo. Los Alamitos, CA: IEEE Computer Society, 2003. pp. 349–354.
10. H.-S. Wang, X. Zhu, L.-S. Peh, and S.Malik, “Orion: a Power-Performance Simulator for Interconnection Networks”, Proceedings of 35th International Symposium on Microarchitecture (MICRO), pp. 294-305, November 2002.
11. S. Wilton and N. Jouppi, “Cacti: An Enhanced Cache Access and Cycle Time Model”, IEEE Journal of Solid State Circuits, Vol. 31, No. 5, pp. 677-688, May 1996.
12. E. Carara, A. Mello, and F. Moraes, “Communication Models in Networks-on-Chip”, Proceedings of 18h International Workshop on Rapid System Prototyping, pp. 57-60, June 2007.
13. S. Mahadevan, F. Angiolini, M. Storgaard, R.G. Olsen, J. Sparso, and J. Madsen, “A Network Traffic Generator Model for Fast Network-on-Chip Simulation”, Proceedings of the Design, Automation and Test in Europe Conference, pp. 780-785, June 2005.
14. R. Kumar, K. I. Farkas, N. P. Jouppi, P. Ranganathan, and D. M. Tullsen, “Single-ISA Heterogeneous Multi-Core Architectures: The Potential for Processor Power Reduction”, Proceedings of 36th International Symposium on Microarchitecture (MICRO), San Diego, USA, December 2003.
15. N. Dutt. “Memory-aware NoC Exploration and Design”, Proceedings of the Design, Automation and Test in Europe, Munich: IEEE, 2008. v. 1, pp. 1128-1129