Scalability Evaluation in Many-core Systems due to the Memory Organization

Guilherme Madalozzo[1], Liana Duenha[2], Rodolfo Azevedo[2], Fernando G. Moraes[1]

[1] PUCRS University, Computer Science Department, Porto Alegre, Brazil
[2] Institute of Computing (IC-UNICAMP), Campinas, Brazil
guilherme.madalozzo@acad.pucrs.br, {liana.duenha, rodolfo.azevedo}@ic.unicamp.br, fernando.moraes@pucrs.br

Abstract—**Many-core systems are a common place in the electronic consumer market. Thus, complex benchmark suites have been modeled for shared memory (SM) architectures since it is easier to develop applications (threads) for many-core. Shared memory presents a scalability limitation due to the number of memory accesses. One way to mitigate the SM limitations is to adopt distributed memory system (DSM) architectures. However, DSM presents the same scalability limitation, since the number of processors is clearly superior to the number of SMs in the system. The alternative is to adopt distributed memory organization (DM), which enables the direct communication among the tasks (message passing). The goal of this paper is to highlight the limitations of shared memory connected through a network-on-chip (NoC) in many-core systems, executing a similar workload with both memory organizations. Results evaluate communication volume into NoC and a total number of executed instructions. SM has a worst traffic distribution, with hotspots around the SM, and a much higher communication volume compared to DM. It is worth to highlight such effects since aging effects may appear earlier due to hotspots, reducing the system lifetime.**

Keywords—MPSoC; Scalability; Memory Organization; Thread; Message Passing; Communication API

I. INTRODUCTION AND RELATED WORKS

The increasing usage of many-core systems raises the interest in benchmarks to evaluate different memory organizations. Modern embedded applications are becoming increasingly complex, and the memory organization defines the programming model. For example, multithreading is suited for shared memory and message passing for distributed memory [1]. Memory organization in many-core embedded systems presents a challenge for software and hardware designers. The communication among processors is not only a function of the communication infrastructure but also on the adopted memory organization.

There are multiple available benchmarks able to evaluate the memory organization, such as: SPLASH-2 [2], MiBench [3] and ParMiBench [4]. These benchmarks assess the performance of several software and hardware models in many-core systems. A benchmark is a set of applications, which execute and report a quantitative evaluation of the performance in a given platform.

The interaction between system components in the platform is exposed to the software developer as an Application Programming Interface (API), which implements the communication interface in an abstract level [1]. An API is a set of functions in the program, e.g. *pthread_create* in PThread API, and *send*/*receive* in MPI (Message Passing Interface). In shared memory architecture, the PThread API implements the parallelism using *threads*, which share memory data with other *threads* running in the system. In distributed memory architectures, the MPI API implements the parallelism using message passing among processors in the system.

Shared memory is a common inter-processor communication paradigm for many-core systems [5]. Lukovic et al. [6] present a framework suitable for reconfigurable multiprocessor systems, using the Xilinx EDK tool-chain for NoC-based MPSoC generation. The communication among processors occurs by shared memory, which is implemented using part of the chip memory (BRAM). Benini et al. [7] present the Platform 2012 (P2012) project which uses shared memory banks to interconnect its processing elements. P2012 can run standard OpenCL and OpenMP parallel applications, as well as the platform-specific Native Programming Model (NPM), which provides the highest level of control on applications' mapping.

Distributed memory system exploits communication level parallelism integrating multiple distributed memory interfaces with massive concurrent memory accesses [8]. Busseuil et al. [9] present a scalable distributed memory platform named OpenScale. The platform is described as an open-source RTL NoC-based MPSoC that executes a preemptive RTOS, providing an MPI-like API to communicate applications' task among the processors. The platform reports the memory occupation, performance evaluation and communication volume. Matilainen et al. [1] present an implementation of Multicore Communication API (MCAPI) on a distributed memory system, providing a unified programming message passing API to different OS types and processors. Results show that the implementation is, in the worst case, ten times faster than others approaches.

In this context, *this work evaluates the scalability in many-core systems due to the memory organization*. The *main* contribution of this paper is to evaluate two NoC-based MPSoCs with different memory organizations: distributed and shared memory. The *goal* is to present and evaluate both platforms executing a similar workload. Shared memory applications are modeled using the PThread API, and distributed memory applications are modeled using MPI API.

II. ARCHITECTURAL MODELING

This Section presents the MPSoC architectural model for both memory organizations. The MPSoCs are described as virtual platforms, using high-level abstraction models. Each platform is modeled using an architectural description language (ADL), ArchC [10] for the shared memory platform and Open Virtual Platform (OVP) [11] for the distributed memory platform.

ArchC [10] is an open-source SystemC-based language that supports the implementation of processor architecture descriptions, following the C++/SystemC syntax style. The main goal is to describe the platform in a high-level abstraction, enabling the designer to explore and to evaluate new architectures with automatic generation software tools and simulators. The architecture description is divided as following:

- *Architecture Resources* `<AC_ARCH>`: the designer provides information about storage devices, pipeline structure, memory hierarchy and micro-architectural details available in the processor documentation. ArchC provides a set of data types (e.g. registers and register banks, pipeline and pipeline stages, caches).
- *Instruction Set Architecture* `<AC_ISA>`: the designer details each instruction executed by the processor (e.g. *opcodes* information, instruction behavior).

OVP [11] is an open virtual platform and modeling framework proposed by Imperas. The OVP models and platforms are described in C/C++, aiming to accelerate the development of embedded systems, specifically for SoCs and MPSoCs. There are three components in the framework: (*i*) APIs that enable to model hardware components, in C language; (*ii*) library of free open-source processors (e.g. ARM, MIPS, PowerPC) and peripheral models (e.g. memories, external interrupts); (*iii*) OVPsim simulator. OVPsim is a dynamically linked library, which supports the simulation of bus-based multiprocessor platforms. OVPSim relies on dynamic binary translation that increases the simulation speed.

A. Shared Memory Platform

The shared memory platform is modeled in ArchC, which allows designers to parameterize several architectural features of the MPSoC. The communication between processors occurs through the shared memory, without support for message passing.

The platform contains a POSIX PThread Embedded API. There are functions that handle thread management (e.g. *pthread_create*) and memory accesses (e.g. *pthread_barrier_init*). The API implements the thread queue control (e.g. *enqueue* and *dequeue*) and supports the management of the memory space for each thread (e.g. *pthread_malloc*).

Figure 1 presents a simplified view of a 2x3 shared memory MPSoC instance.

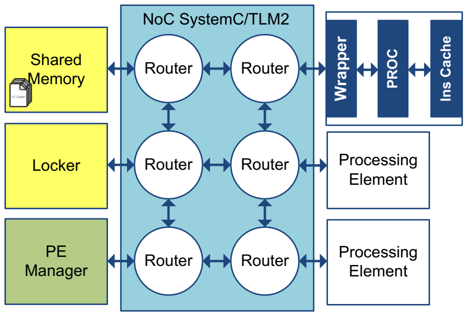


Figure 1 - ArchC shared memory platform.

The NoC is implemented in SystemC-TLM2.0. A set of wrappers implements the interface between IPs and the NoC. The wrapper is an important component of the system, enabling to improve performance and meet requirements of real-time applications [12]. Each processing element (PE) connected to the NoC router contains a wrapper, an ARM ArchC processor, and an instruction cache. One PE is responsible for launching applications (*Manager PE* - PE_M). The platform contains an external shared memory (512MB). A TLM channel connects the memory to the NoC router. The third IP type connected to the NoC is the *locker*. The locker is responsible for implementing semaphores, mutexes, barriers and conditional variables to manage the memory access, ensuring the atomicity of data in the system.

Figure 2 details the sequence of events to execute a given application - app_{exec}. All PEs, except PE_M, start in a *waiting state*. Initially (1), the shared memory receives the object code of app_{exec}. The PE_M receives a set of instructions starting the execution of app_{exec}. When a `pthread_created` is called (2), the PThread API stores the thread ID in a thread queue in the shared memory (3). After updating the thread queue, the API sets a control variable that enables a PE to execute the thread (4). The PE_M sends a message to the PE to execute the thread (5). When the PE enabled to execute a thread receives the message, it leaves the waiting state (6) and requests instructions to the shared memory (7). The thread instructions are stored in the instruction cache, and the thread starts its execution (8). The adoption of instruction caches reduces the number of packets using the NoC.

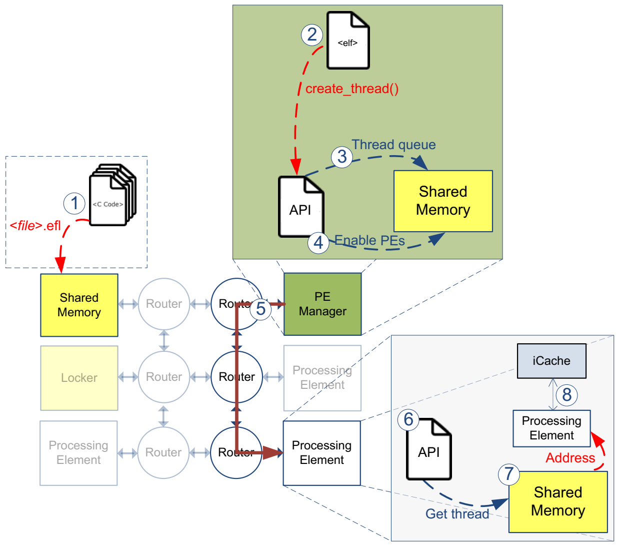


Figure 2 – Execution of shared memory architecture sequence.

B. Distributed Memory Platform

This Section presents the OVP distributed memory platform, which contains a NoC implemented in C language (OVP APIs *ppm* and *bhm*), with the same routing algorithm and arbitration policy of the SystemC-TLM2.0 NoC used in the shared memory platform. The objective of this platform is to allow the designer to evaluate benchmarks using a message passing API to transfer data between processors.

Figure 3 presents a simplified view of the 2x3 distributed memory MPSoC platform. Each PE contains an ARM OVP processor connected to a private memory (in fact a scratchpad memory), a DMA module, a network interface (NI) that connect the NoC router to the OVP processor and a DMA. A script generator, written in Python, generates the platform (hardware and software).

The distributed memory platform contains an MPI-like API that implements the inter-task communication. There are two main functions: *send*() and *receive*(). A non-blocking *send()* insert messages in communication queues, while a blocking *receive()* read from the communication queues.

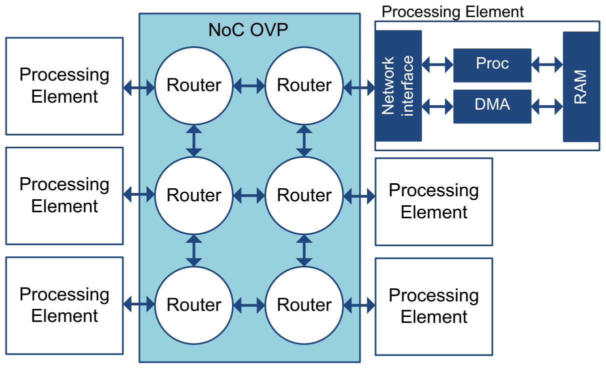
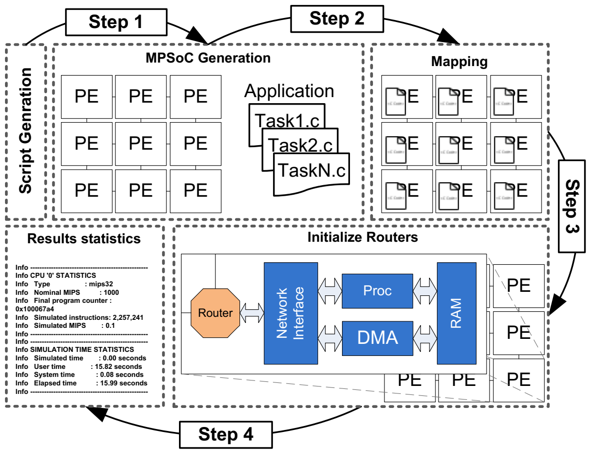


Figure 3 - OVP distributed memory platform.

Figure 4 presents the flow to generate and evaluate the Distributed Memory MPSoC. The first step generates the platform with different applications and different MPSoC configurations, compiling the software (applications code) and the hardware (processors and OVP peripherals). In the second step, the object codes of applications' tasks are stored in the memories. It is important to note that the mapping is static, and each PE receives one task to execute. The third step initializes all routers, starting the communication among processors. The router initialization is a function implemented in the API. In the fourth step, the OVP reports at the end of the simulation details related to the number of instruction executed per processor, and the communication volume at NoC.



```
Info ------------------------------------------------
Info CPU '0' STATISTICS
Info   Type                  : mips32
Info   Nominal MIPS          : 1000
Info   Final program counter :
0x100067a4
Info   Simulated instructions: 2,257,241
Info   Simulated MIPS        : 0.1
Info ------------------------------------------------
Info ------------------------------------------------
Info SIMULATION TIME STATISTICS
Info   Simulated time        : 0.00 seconds
Info   User time             : 15.82 seconds
Info   System time           : 0.08 seconds
Info   Elapsed time          : 15.99 seconds
Info ------------------------------------------------
```

Figure 4 – Framework execution

Both platforms have the same NoC model. There is no operating system, the APIs (*pthread* and *MPI-like*) are responsible for the system management and applications' execution.

III. Results

This Section presents the scalability evaluation of distributed and shared memory organization. Relevant features of the MPSoC models include: 32-bit PE word and 16-bit flit; NoC router: wormhole packet switching, XY routing, input buffering, centralized round-robin arbitration. Both models execute two applications. The first one is the Dijkstra application in three versions: (*i*) one producer and one consumer; (*ii*) one producer and three consumers; (*iii*) one producer and four consumers. The second one is an FFT, with a larger communication volume compared to the Dijkstra application.

Two evaluations are carry-on to discuss the scalability issue. The first evaluation concerns the communication volume transferred into the NoC. The second evaluation concerns the total number of instructions simulated in all processors, showing the traffic distribution into NoC.

A. *Distributed Memory Evaluation*

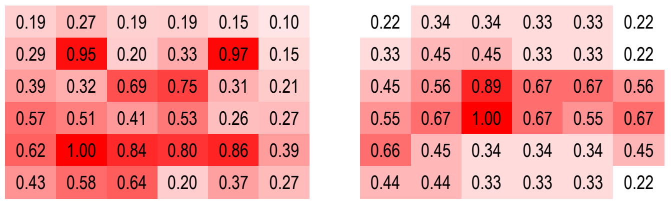
Each PE executes one task in the distributed memory (DM) platform. Thus, the number of tasks is equal to the MPSoC size. Experiments are executed in MPSoCs with 4, 16, 36, 64, 100, 144 and 225 PEs.

Table I presents for each MPSoC size the total number of transferred flits and the total number of executed instructions. The DM platform assumes that the binary code is already loaded at each local memory. Therefore, all transferred flits are data flits. There is no traffic related to instructions. It is possible to note that the growth in both performance parameters is nearly linear with the MPSoC size ($r^2 > 0.98$ for MPSoCs larger than 16 PEs). *This fact demonstrates the scalability of the distributed memory architecture.*

Table I - Number of flits and executed instructions for the distributed memory platform.

MPSoC size	Total number of transferred flits (10^3 *flits*)		Total number of executed instructions (10^6 *instructions*)	
	Dijkstra	FFT	Dijkstra	FFT
4	14.6	17.1	1.7	5.2
16	46.0	112.6	4.5	12.4
36	143.2	339.6	13.8	22.3
64	263.8	1084.1	49.3	40.1
100	446.7	1915.7	117.1	62.9
144	921.9	3294.6	241.0	91.1
225	1425.2	6294.5	588.1	143.0

Another relevant feature to evaluate is the traffic distribution internally to the MPSoC. Figure 5 presents the *relative communication volume* transferred by each router. *Note the absence of hotspots*, with the traffic evenly distributed in the MPSoC. Routers sending a larger volume of data flits correspond to the producer tasks of the applications.



0.19	0.27	0.19	0.19	0.15	0.10
0.29	0.95	0.20	0.33	0.97	0.15
0.39	0.32	0.69	0.75	0.31	0.21
0.57	0.51	0.41	0.53	0.26	0.27
0.62	1.00	0.84	0.80	0.86	0.39
0.43	0.58	0.64	0.20	0.37	0.27

(a) Dijkstra application

0.22	0.34	0.34	0.33	0.33	0.22
0.33	0.45	0.45	0.33	0.33	0.22
0.45	0.56	0.89	0.67	0.67	0.56
0.55	0.67	1.00	0.67	0.55	0.67
0.66	0.45	0.34	0.34	0.34	0.45
0.44	0.44	0.33	0.33	0.33	0.22

(b) FFT application

Figure 5 – Relative communication volume at each router for a 6x6 DM MPSoC.

B. *Shared Memory Evaluation*

Each PE also executes one task in the shared memory (SM) platform. The number of tasks is different to MPSoC size, as showed in Table II, because one router is dedicated to the shared memory and another one to the locker IP. Smaller SM platforms are evaluated, due to the longer simulation time of ArchC compared to OVP, but enough to show the effects of scalability.

TABLE II – SHARED MEMORY PLATFORM SETUP.

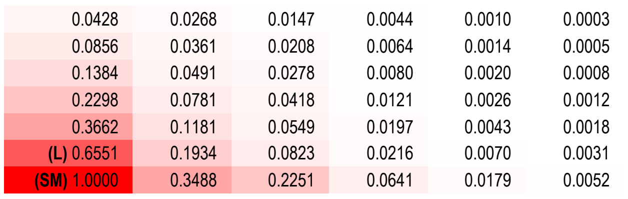
MPSoC Size	4	8	10	12	18	42
Tasks	2	6	8	10	16	40

Table III presents for each MPSoC size the total number of transferred flits and the total number of executed instructions. The SM platform presents a different behavior compared to the DM platform. An exponential growth in both parameters is observed. Two reasons explain such behavior. First, the adoption of a shared memory implies in a large number of accesses to the memory and to the locker, for shared data. With message-passing, this behavior is avoided, with local communications. Second, the use of instruction caches by the architecture increases the traffic in the NoC. The number of executed instructions increases because during a cache miss the operating system executes idle operations, waiting for valid code to execute.

TABLE III – NUMBER OF FLITS AND EXECUTED INSTRUCTIONS FOR THE SHARED MEMORY PLATFORM.

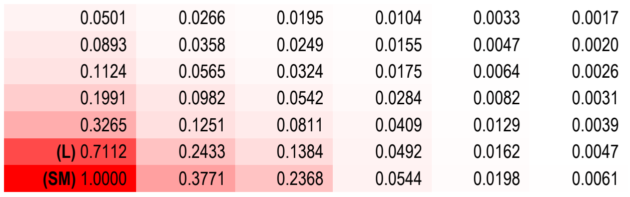
MPSoC size	Total number of transferred flits (10^6 *flits*)		Total number of executed instructions (10^6 *instructions*)	
	Dijkstra	FFT	Dijkstra	FFT
4	31.3	68.7	5.18	10.3
8	45.5	110.7	6.06	13.4
10	76.3	164.7	8.68	17.9
12	111.8	287.6	13.21	32.6
18	188.1	399.8	18.13	38.4
42	4075.7	9278.8	339.99	706.9

Figure 6 presents the *relative communication volume* transferred by each router in the SM architecture (Dijkstra and FFT applications). As expected, hotspots are clearly identified near to the shared memory (SM) and the locker IP (L). The traffic distribution is unevenly distributed internally to the MPSoC. Wear out effects, as electromigration, may appear earlier compared to a DM organization, reducing the system lifetime.



0.0428	0.0268	0.0147	0.0044	0.0010	0.0003
0.0856	0.0361	0.0208	0.0064	0.0014	0.0005
0.1384	0.0491	0.0278	0.0080	0.0020	0.0008
0.2298	0.0781	0.0418	0.0121	0.0026	0.0012
0.3662	0.1181	0.0549	0.0197	0.0043	0.0018
(L) 0.6551	0.1934	0.0823	0.0216	0.0070	0.0031
(SM) 1.0000	0.3488	0.2251	0.0641	0.0179	0.0052

(a) Dijkstra application



0.0501	0.0266	0.0195	0.0104	0.0033	0.0017
0.0893	0.0358	0.0249	0.0155	0.0047	0.0020
0.1124	0.0565	0.0324	0.0175	0.0064	0.0026
0.1991	0.0982	0.0542	0.0284	0.0082	0.0031
0.3265	0.1251	0.0811	0.0409	0.0129	0.0039
(L) 0.7112	0.2433	0.1384	0.0492	0.0162	0.0047
(SM) 1.0000	0.3771	0.2368	0.0544	0.0198	0.0061

(b) FFT application

Figure 6 - Relative communication volume at each router for a 6x7 SM MPSoC. L→Locker, SM→ Shared Memory.

Results presented in Table III and Figure 6 highlights the limitations of the shared memory organization for large MPSoCs (at least 64 PEs). For intermediate MPSoC sizes, DSM may be an alternative for SM. For large MPSoC sizes, DM seems to be a suitable alternative. However, it is not possible to execute applications written with threads. Hybrid Memory Architecture (some shared memories and distributed local memory at each PEs) may be another direction to follow, with the concomitant execution of threads and message passing.

IV. CONCLUSIONS AND FUTURE WORKS

This paper proposed a scalability evaluation in many-core systems, which enables to simulate and validate benchmark suites in shared and distributed memory platforms. Results showed the effects of scalability executing real applications, with a large amount of exchanged data and longer execution times. Limitations of SM organizations are clearly identified as larger communication volume compared do DM and the presence of hotspots in the system.

Works may be extended in two main directions: (*i*) include a dynamic mapping algorithm in the DM platform, enabling an absolute communication volume comparison (the present work made only relative evaluations, since there is no instruction traffic in the DM platform); (*ii*) develop a hybrid platform, using shared and distributed memory.

ACKNOWLEDGMENTS

The Author Fernando Moraes is supported by CNPq - projects 472126/2013-0 and 302625/2012-7, and FAPERGS - project 2242- 2551/14-8.

REFERENCES

[1] Matilainen, L.; Salminen, E.; Hamalainen, T.D.; Hannikainen, M. "Multicore Communications API (MCAPI) Implementation on an FPGA Multiprocessor". In: SAMOS, 2011, pp. 286-293.

[2] Woo, S.C.; Ohara, M.; Torrie, E.; Singh, J.P.; Gupta, A. "The SPLASH-2 programs: characterization and methodological considerations". Computer Architecture, 1995. Proceedings., 22nd Annual International Symposium, 1995, pp. 24-36.

[3] Guthaus, M.R.; Ringenberg, J.S.; Ernst, D.; Austin, T.M.; Mudge, T.; Brown, R.B. "MiBench: A free, commercially representative embedded benchmark suite". In: WWC-4. 2001, pp. 3-14.

[4] Iqbal, S.M.Z.; Yuchen L.; Grahn, H. "ParMiBench - An Open-Source Benchmark for Embedded Multiprocessor Systems". Computer Architecture Letters, v.9(2), 2010, pp. 45-48.

[5] Loghi, M.; Poncino, M.; Benini, L. "Cache coherence tradeoffs in shared-memory MPSoCs". In: TECS, 2006, pp. 383-407.

[6] Lukovic, S.; Fiorin, L. "An Automated Design Flow for NoC-based MPSoCs on FPGA". In: RSP, 2008, pp. 58-64.

[7] Benini, L.; Flamand, E.; Fuin, D.; Melpignano, D. "P2012: Building an ecosystem for a scalable, modular and high-efficiency embedded computing accelerator". In: DATE, 2012, pp. 983-987.

[8] Fu, W.; Chen, T.; Wang, C.; Liu, L. "Optimizing Memory Access Traffic via Runtime Thread Migration for On-chip Distributed Memory Systems". The Journal of Supercomputing, v.69(3), 2014, pp. 1491-1516.

[9] Busseuil, R.; Barthe, L.; Almeida, L. b.; Ost, L.; Bruguier, F.; Sassatelli, G.; Benoit, P.; Robert, M.; Torres, L. "Open-Scale: A Scalable, Open-source NoC-Based MPSoC for Design Space Exploration". In: ReConFig, 2001, pp. 357-362.

[10] Rigo, S.; Araujo, G.; Bartholomeu, M.; Azevedo, R. "ArchC: a systemC-based architecture description language". In: SBAC-PAD, 2004, pp. 66-73.

[11] OVP 2014, Available at: www.ovpworld.org/technology_ovpsim.php

[12] Zhuo, L.; Du, G.; Zhang, D.; Song, Y.; Li, L.; Pan,H. "Design and Implementation of DDR2 Wrapper for Cluster based MPSoC". In: ASID, 2010, pp. 60-62.