MPSoCBench: A Benchmark Suite for Evaluating Multiprocessor System-on-Chip Tools and Methodologies

Liana Duenha *Rodolfo Azevedo*

Technical Report - IC-13-19 - Relatório Técnico

August - 2013 - Agosto

MPSoCBench: A Benchmark Suite for Evaluating Multiprocessor System-on-Chip Tools and Methodologies

Liana Duenha [*] Rodolfo Azevedo [†]

August 8, 2013

Abstract

The consolidated use of multiprocessors in the embedded systems designs introduces new important challenges for system architects and hardware/software designers. In particular, it becomes necessary to develop tools based on new paradigms, able to deal with MPSoC complexities. Recent design methodologies and tools are focused on enhancing the design productivity by providing a software development platform before the final MPSoC architecture details are fixed. However, the simulation can only be efficiently implemented when using a modeling and simulation engine that supports the system behavior description in a high abstraction level. The lack of multiprocessor platforms integrating both scalable hardware and software in order to design and evaluate new methodologies and tools motivated us to develop MPSoCBench, which is a benchmark composed by a scalable set of MPSoCs including four different ISAs (PowerPC, MIPS, SPARC, and ARM), organized in platforms with 1, 2, 4, 8, 16, 32, or 64 cores, with different interconnections (router or NoCs), capable of running 15 parallel version of software from well known parallel benchmarks. The tool also provide power consumption estimation for MIPS and SPARC processors. The benchmark sums 876 distinct configurations automated through scripts.

1 Introduction

In the last decade, the adoption of Multiprocessor System-on-Chip (MPSoCs) in the embedded systems scenario created several new design challenges, both in software

[*]Both authors are with the Institute of Computing, University of Campinas. Email: liana.duenha@lsc.ic.unicamp.br, rodolfo@ic.unicamp.br

[†]The authors would like to acknowledge CAPES and CNPQ for the partial support for this research.

and hardware implementations.

Considering the increasing device integration, the complexity of such designs and the challenge of improving the performance of such projects establish modeling and simulation of high level abstraction systems as an increasingly important research area. Recent design methodologies and tools focus on enhancing the design productivity by providing a software development platform before the final MPSoC architecture details are fixed. However, the simulation can only be efficiently implemented when using a modeling and simulation engine that supports the system behavior description into high abstraction level [5, 12, 15].

The MPSoCBench is a benchmark composed by a scalable set of MPSoCs to enable the development and evaluation of new tools, methodologies and hardware components. Each of the four supported processors (PowerPC, MIPS, SPARC, and ARM) is used in 3 different MPSoC configurable and scalable platforms with 1, 2, 4, 8, 16, 32, or 64 cores, capable of running 11 parallel version of software from SPLASH-2 [17] and ParMiBench [9]. Additionally, we have also designed 5 multi-software applications, combining different software from ParMiBench and Mibench benchmarks. The total combined size reaches 876 distinct configurations automated through a Python script, which enables easy implementation of part or all of the benchmark locally or for parallel execution on a Hyperthreading Condor (HTCondor) cluster.

1.1 MPSoCBench goals

The configurable and extensible infrastructure of the MPSoCBench suite allows its use in different scenarios, like:

- Implementation and evaluation of new tools or methodologies in MPSoC designs, or comparisons between different tools in a similar environment;
- Development and monitoring design refinements for lower abstraction levels, which we consider that is a important need in the Electronic System Level designs;
- Evaluation and comparison of parallel applications using different techniques for parallelization; and scalability characterization;
- Analysis and optimization of new hardware components, such as routers, buses, NoCs, IPs, and wrappers, in a co-design environment;
- Comparisons among different techniques for power consumption estimation, and dynamic characterization of program power consumption considering different power models;

- Power consumption analysis at the early stages of development and energy efficiency optimization;

This report is organized as follows: Section 2 briefly explores the prior tools or libraries related with our work and exposes how they compare to our main goals; Section 3 describes the components available in MPSoCBench and how they are organized. We present the main results of the MPSoCBench characterization in the Section 4 and we show some specific details about how to use the benchmark in a way to foster future usage and tool development in the Section 5. Finally, Section 6 concludes the work.

2 Related Work

Prior work on MPSoC platforms range from specific processor platforms like [5], where the OpenRISC processor is the central part of the MPSoC that include SystemC communication models (Whishbone bus).

The MPARM [4] is a highly configurable platform for ARM processors, modeled in SystemC. Neither of them provides a set of scalable software like MPSoCBench.

The MultiFlex [14] environment is an application to platform mapping tool that integrates heterogeneous parallel components (HW and SW) into a platform programming environment. MultiFlex can be characterized as a tool, not a benchmark.

SoCLib [13] is an open platform for virtual prototyping of MPSoCs. The core of the platform is a library of SystemC models for virtual components (IP cores). Although there are many components available in SocLib design repositories, modeling platforms containing multiple devices to evaluate MPSoCs demands a great effort of joining them together. Neither of them includes a comprehensive set of scalable software to run on the platforms.

We also cite the benchmarks that we adapted to run on the MPSoCBench simulation infrastructure. The SPLASH-2 [17] and ParMiBench [9] benchmarks have a set of parallel softwares using POSIX Threads, able to use up to 16 threads; however, they are just a software infrastructure, then they do not have hardware components for co-design evaluation.

The lack of multi-processor platforms integrating both scalable hardware and software to design and evaluate new methodologies and tools motivated us to develop MPSoCBench.

3 MPSoCBench Components

The goal of this Section is to describe all tools and components available in MPSoCBench and how they are organized.

3.1 Tools

SystemC [2] is a collection of C++ classes and templates that provides powerful mechanisms to model system architecture with hardware timing, concurrency and reactive behavior, allowing the creation of an executable specification of the system.

A virtual platform is typically constructed by several transaction-level models (TLM) [1], which are a higher level representation of hardware behavior, focusing on discrete events such as register reading and writing. Although TLM is language independent, SystemC fits perfectly in its representation style by allowing adequate abstraction levels and by providing elements to support isolation of computation and communication.

ArchC [3] is an *Architecture Description Language* (ADL) following a SystemC syntax style, which provides enough information in order to allow users to explore and verify a (new or legacy) processor's architecture by automatically generating not only software tools for code generation and inspection (like assemblers, linkers, and debuggers), but also executable processor models for platform representation [3, 15].

PowerSC is a SystemC extension aiming at the gathering of switching activity [10]. It is a complete framework designed to collect information from any SystemC functional description. As ArchC generates a SystemC processor description, it is an eligible candidate to PowerSC workflow, which abstracts from the user a large share of the process. The *acPower Library* is a library of functions incorporated into ArchC, developed to enable the power analysis of ArchC processor modules using the PowerSC tool.

In our benchmark, we use ArchC 2.2 processor models already implemented and validated, and peripherals developed with TLM 2.0 interface, running over SystemC 2.3.0 simulation kernel.

3.2 Description of the Components

We organize all software required to build platforms in a directory structure based on *ArchC Platform Manager* (ACPM) to simplify the ArchC usage, making it easy to reuse the components across distinct platforms [15], and we also provide a front-end Python script based on the ACPM to allow easy selection and execution of single or multiple platform and programs configurations. The directories available are:

- **processors:** Contains all ArchC processor models available: `PowerPC`, `MIPS`, `SPARC`, and `ARM`, which are 32-bit versions of the PowerPC, MIPS-I, V8 version of SPARC architecture, and ARMv5e instruction set, respectively. Section 3.3 describes each one of these processors in details.

- **ip:** Here we store the non-programmable platform components. MPSoCBench provides a configurable `tlm_memory` model, an external memory shared by all processors, connected by a TLM channel, pre-configured to 512MB; and a `tlm_lock`, a TLM component which acts as a hardware lock. Section 3.4 describes these peripherals with further details.

- **is:** Contains the available interconnection structures. MPSoCBench provides two different devices: one simple router module, called `tlm_router`, which essentially consists of TLM ports to connect to the memory and lock devices, followed by communication interfaces with each instantiated processor; and a mesh based NoC using XY routing protocol, which can be used in a approximately timed mode or in a loosely timed mode. More details of these devices are described in the Section 3.5.

- **sw:** This directory stores 15 different applications from MiBench [8], ParMiBench [9], and SPLASH-2 [17] benchmark suites adapted to execute in our scalable parallel environment. Each application in this directory can be compiled for any of the available processor. Tasks are partitioned among processors in runtime. Section 3.6 exposes the main characteristics about these applications.

- **platforms:** Contain the main platform description files which interconnect all the required modules. Each platform connects a set of cores, a shared memory `tlm_memory`, a hardware lock device `tlm_lock`, and one of the available interconnection structure used to connect all platform components. The Figure 1 shows an illustration of a virtual platform that we can build with this benchmark, which is a mesh based NoC as interconnection; this platform is comprising a set of 2 × 3 nodes, each one basically acting as a router, able to connect four processors, memory and a lock device.

  The hereinafter subsections describe each one of these components:

3.3 Processor Models

We make the source code of these models available and we use the appropriate Makefile to create the simulator obeying the user specification, like the kind of channel that must be used to communicate with the interconnection device or power consumption.

A brief description of the ArchC behavioral processor models follows:

PowerPC: is a RISC architecture that became famous for equipping Apple's Macintosh machines. Nowadays, PowerPC has a big family of processors both from IBM and Motorola, that are widely used both on desktop and embedded systems. The ArchC PowerPC model implements the PowerPC 32 bits instruction set, including ABI emulation.

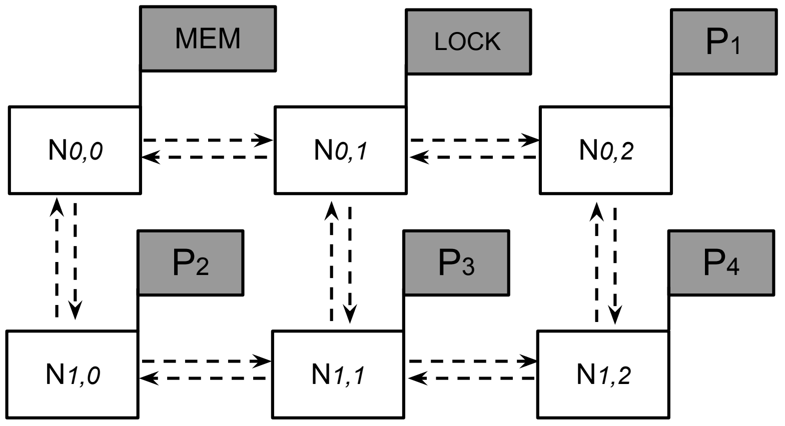


Figure 1: An illustration of a platform designed as a 2×3 mesh

MIPS: is a well known RISC architecture that contains an integer unit plus a 32-bit register bank and two special registers used for multiplication and division. These models implement a MIPS-I ISA description, including delay slots and ABI emulation.

SPARC: is another well known RISC architecture, which has specific features, like register window, and a more complicated ISA if compared with the MIPS-I ISA. This model implements the V8 version of the SPARC architecture, including delay slots, register window and ABI emulation.

ARM: is a RISC architecture, and was inspired by Berkeley's first papers on this simple and fast architecture. The processor succeeded in the embedded market because of its low power consumption characteristics, and today is the most used embedded processor and is still in active development. Our ARM model implements the ARMv5 instruction set, including ABI emulation.

3.4 Memory and Lock

We use a external shared memory that is connected with the master (the processors) using TLM channels, pre-configured to 512MB, which is the minimum necessary to run all the applications available in the suite, considering the inputs provided. We have two versions of the code, depending on which interconnection device is used: the simplest one uses TLM `sc_exports` to receive transactions and to return its answers; the most elaborate uses TLM 2.0 `simple_target_socket`, useful for implementing approximately timed abstraction level simulation.

Since that the main goal of MPSoCBench is to provide a infrastructure to parallel simulation, we include an IP that acts as a hardware lock, which is useful for implementing mutexes, semaphores, conditional variables, and barriers. We use this peripheral to implement a library that emulates the main functionalities of the POSIX Pthreads.

As the memory, this component communicates with the masters using `sc_exports` or `simple_target_socket` channels, according to the parameters provided by the user.

3.5 Interconnection Devices

The interconnection components are responsible to connect the peripherals and the processing units. We make available two different interconnection, which different abstraction level. In both cases, the basic transport unit in this network is a TLM 2.0 generic payload.

Router: This component essentially consists of TLM channels to connect to the memory and lock devices, followed by communication interfaces with each instantiated processor. The transaction is carried through TLM blocking transport with time annotation. At the end of a transaction, we can use the annotated time to update the simulation time, which is useful for a simulation in a loosely timed abstraction level.

NoC: This component is a mesh based NoC using XY routing protocol, configurable in runtime to be able to connect up to 64 cores, memory, lock and wrappers for all components. The transport interface defines the simulation abstraction level. We may choose between two different approaches: the blocking transport characterizes the NoC loosely timed (NoC-LT) and the non-blocking transport characterizes the NoC approximately timed (NoC-AT). This NoC is totally configured in runtime through user parameters. The blocking or non-blocking transport interfaces must be selected before the NoC is built. The main details of these two different NoCs follows:

- NoC-LT: It is composed by a set of nodes, each one connected with a wrapper. Each NoC node has North (N), East (E), South (S) and West (W) `sc_ports` to connect it with other nodes. Each node also has a `sc_port` object to connect it to the wrapper and a `sc_export` channel to receive packages from other nodes and from the wrapper.
- NoC-AT: The NoC-AT uses sockets, forward and backward transport interfaces. Each NoC-AT node has North (N), East (E), South (S) and West (W) `simple_initiator_socket` and `simple_target_socket` objects to connect it with other nodes. Each node also has a local socket to connect

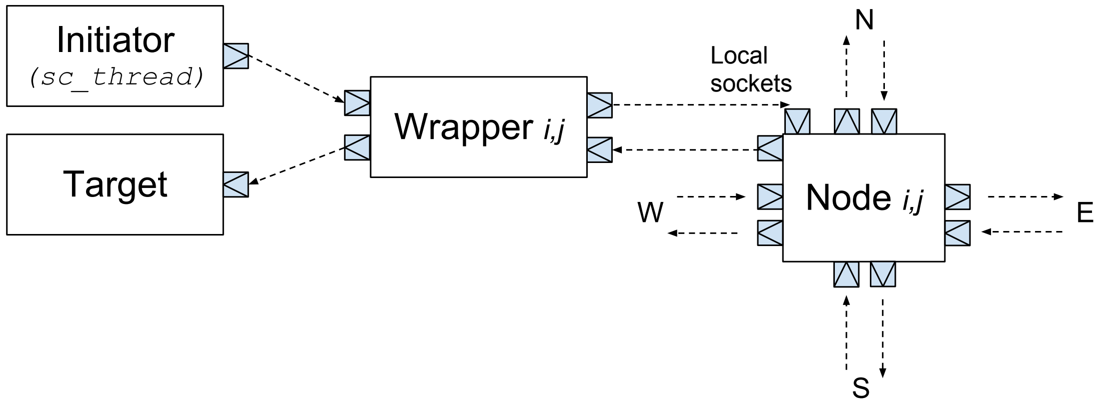


Figure 2: An example of the main components that are associated in order to compose the NoC-AT mesh

it (`simple_initiator_socket` and `simple_target_socket`) to the wrapper. The main difference is that each one of the NoC-AT nodes has a `sc_thread` associated with it, it uses non-blocking transport methods and it has two phases associated with the transaction.

The Figure 2 illustrates the main objects need to implement the NoC-AT. Basically, a initiator is a processing unit and the targets are the IPs in the platform. The communication starts with a request from the initiator `sc_thread`. The initiator creates the generic payload, sends the request using the forward path and calls wait for an answer event. The wrapper receives the request, creates extensions with routing information and puts the package into the NoC. Then, the package passes through several nodes until it reaches the target.

The backward path is similar. The target (in most of cases, the memory) updates the generic payload with the appropriate response, and sends the request using the backward path. The package passes through several nodes until it reaches the initiator. The initiator backward transport method (`nb_transport_bw()`) notifies the answer event and unlocks the `sc_thread` which starts the transaction.

3.5.1 Approximately-timed coding style

The approximately-timed coding style is supported by the non-blocking transport interface, which is appropriate for the use cases of architectural exploration and performance analysis. The non-blocking transport interface provides for timing annotation

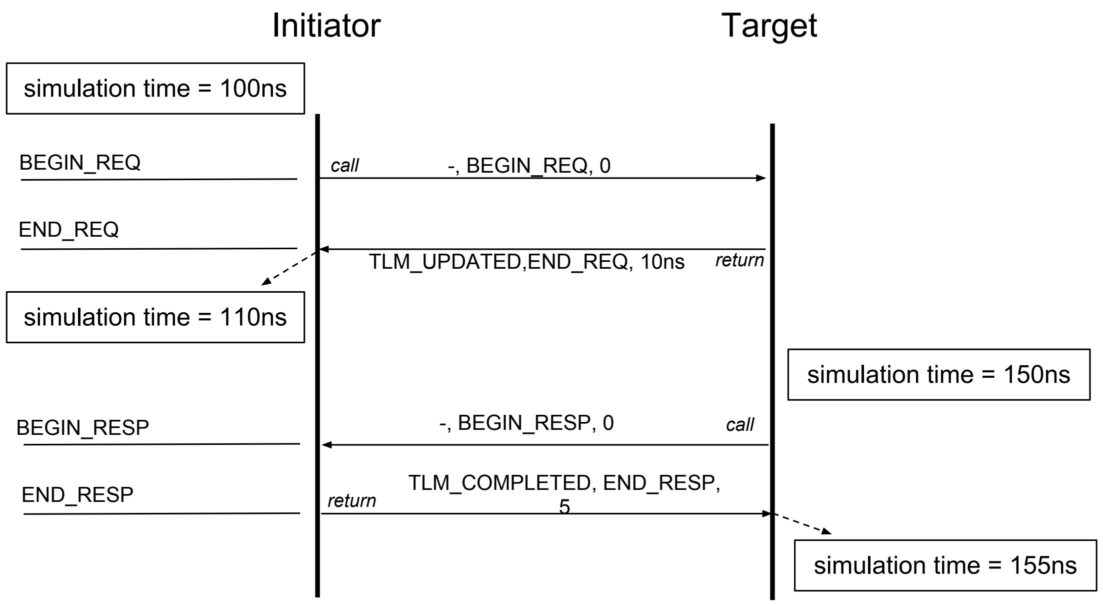


Figure 3: An example of calling sequences to non-blocking transport

and for two distinct phases and timing points during the lifetime of a transaction.

In the protocol adopted in the the NoC-AT based models the transaction is broken down into two distinct phases, with an explicit timing point marking the transition between phases. There are timing points marking the beginning and the end of the request and the beginning and the end of the response (BEGIN_REQ, END_REQ, BEGIN_RESP, END_RESP). Also, each transport method returns one of these two states - TLM_UPDATED, TLM_COMPLETED - which is useful to update the phase properly.

The message sequence chart in Figures 3 illustrates various calling sequences to non-blocking transport (`nb_transport_fw` or `nb_transport_bw`, depending on the transaction direction). The arguments and return value passed to `nb_transport` are shown using the notation *return, phase, delay*, where return is the value returned from the function call, phase is the value of the phase argument, and delay is the value of the `sc_time` argument. The notation '-' indicates that the value is unused. In our infrastructure, a processor is just an initiator, the memory and lock are targets, and each NoC node is both an initiator and a target.

This illustration use the phases of the base protocol as an example, that is, BEGIN_REQ, END_REQ and so on. With the approximately-timed coding style, a transaction is passed back-and-forth twice between initiator and target.

If the recipient of an `nb_transport` call can immediately calculate the next state of the transaction and the delay to the next timing point, it may return the new state on return from `nb_transport`. If the next timing point marks the end of the transaction, the recipient can return either TLM_UPDATED or TLM_COMPLETED. This applies to initiator and target alike.

With TLM_UPDATED, the callee should update the transaction, the phase, and the timing annotation. In Figure 3, the non-zero timing annotation argument passed on return from the function calls indicates to the caller the delay between the phase transition on the hop and the corresponding timing point.

Because processes are regularly yielding control to the scheduler in order to allow simulation time to advance, the approximately-timed coding style is expected to simulate a lot more slowly than the loosely-timed coding style [1].

3.6 Applications

We have adapted 13 different applications from MiBench [8], ParMiBench [9] and SPLASH-2 [17] benchmark suites to be configurable in runtime to execute in our scalable parallel environment. Some of them does not run properly in all available multicore platforms, due to limitations of the original implementation. The available applications, their main characteristics and limitations follow:

- **SHA**: It implements the Secure Hash Algorithm, useful to generate digital signatures used in the secure exchange of cryptographic keys; each processor calculates the digest of a text from a different input file; all processors store the generated data in a unique output file. This is an application from ParMiBench.

- **Dijkstra**: It calculates the all-pairs shortest paths in a graph represented by an adjacency matrix, using a data decomposition strategy in such a way that one processor handles one vertex to get its single-source shortest paths. This is an application from ParMiBench.

- **Stringsearch**: It finds a pattern in a number of given phrases using the by employing case sensitive or insensitive comparisons algorithms. The partitioning strategy is to partition the entire pattern collection into a number of sub-pattern collections according to the number of workers allocated. This is an application from ParMiBench.

- **Susan-corners**: It is part of the original ParMiBench Susan application, which has a set of an image recognition functionalities, useful for image recognition in Magnetic Resonance Images of the brain; this application recognizes corners in images, and is parallelized by using data decomposition. It may be allocation problems running this benchmark in platforms with 64 cores, so we do not consider this platform as a possible configuration to run Susan-corners. The original version of this application was able to run properly in up to 8 cores, but we show that we can obtain correct output in platforms with 16 and 32 cores as well.

- **Susan-edges**: It recognizes edges in images, and it is parallelized by using data decomposition. This is an application from ParMiBench. The same allocation problems may happen in this benchmark. The original version of this application was able to run properly in up to 8 cores, but we show that we can obtain correct output in platforms with 16 and 32 cores as well.

- **Susan-smoothing**: It performs adjustments for threshold, brightness, and image smoothness; it is parallelized by using data decomposition. This is an application from ParMiBench, but unlike the others, its original version has the limitation to running on up 8 cores. We chose do not significantly modify the original variables or methods to guarantee the properly execution in platforms with more cores. In order to not mischaracterize the application, we chose not to modify significantly the original variables or methods to guarantee the properly execution in platforms with more cores.

- **Basicmath**: It is useful for benchmarking mathematical calculation, like cubic function solving, angle conversions from degrees to radians,and integer square root; the parallelization is done by data partitioning, i.e., the input is partitioned into different sets. This is an application from ParMiBench.

- **FFT**: A Fast Fourier Transform (FFT); the data set consists of the complex data points to be transformed, and another complex data points referred to as the roots of unity; communication occurs in three matrix transpose steps, which require all-to-all interprocessor communication. This is an application from SPLASH-2 and its original version has the limitation to running on up 16 cores.

- **LU**: The LU kernel factors a dense matrix into the product of a lower triangular and an upper triangular matrix. The dense $n \times n$ matrix A is divided into an arrays of blocks that are allocated locally to processors that own them. This is an application from SPLASH-2 and its original version has the limitation to running on up 16 cores.

- **Water**: This application evaluates forces and potentials that occur over time in a system of water molecules. A process updates a local copy of the particle accelerations as it computes them, and accumulates it into the shared copy once at the end. This is an application from SPLASH-2 and its original version has the limitation to running on up 16 cores; however, to run this application in a 16-core platform, it is necessary update the input size directly in the source code, as indicated in the comments provided.

- **Water-spatial**: It solves the same problem as Water, but uses a more efficient algorithm. This is an application from SPLASH-2 and its original version has the limitation to running on up 16 cores.

- **Multi_p**: This application is composed by four parallel applications from ParMiBench compiled as an unique application adapted to run in parallel in multi-thread environments (64-core with 16 threads each application; 32-core with 8 threads each application and so on). The applications are: SHA, Stringsearch, Dijkstra, and Basicmath. This multisoftware_p application runs just in platforms with more than 4 cores.

- **Multi_network_automotive**: This application is composed of four different single-core applications from the Network and Automotive categories of the Mibench [8] benchmark, compiled as a single application adapted to run on a platform with four processors. The applications are: Dijkstra, Susan-corners, Basicmath, and Qsort.

- **Multi_office_telecomm**: This application is composed of four different single-core applications from the Office and Telecomm categories of the Mibench [8] benchmark, compiled as a single application adapted to run on a platform with four processors. The applications are: Stringsearch-PBM, Stringsearch-BMH, ADPCM, and FFT.

- **Multi_security**: This application is composed of four different single-core applications from the Security category of Mibench [8] benchmark, compiled as a single application adapted to run on a platform with four processors. The applications are: SHA, Rijndael-encoder, Rijndael-decoder, and Blowfish.

Considering that this simulation environment do not have a operating system to coordinate the threads, this management is implemented through acPthread, a PThread emulation library that can handle thread creation and termination functions, barriers, mutual exclusion, semaphores, and conditional variables. These are the necessary functions to execute the parallel benchmarks.

In order to enable the execution of the mentioned applications in the MPSoCBench infrastructure, we put an extra code in each of them to include, initialize and use some acPthread variables and functions. So, we do this tasks in a specific `main()` function that must be executed before the original main function. This main function is basically the same for all applications; The only change required in the application is mechanism for passing its input/output arguments.

3.7 Power Consumption Estimation

Energy efficiency evaluation is one of the main research topics in the embedded systems designs, so we include a power consumption model for SPARC and MIPS processors, base on the Tiwari method [11, 16].

ArchC SPARC and MIPS processor models were implemented based on the LEON and PLASMA RTL models, which are a SPARCv8 and MIPS I compatible architectures, respectively. Based on these implementations, we included, in MPSoCBench, the PowerSC library and the acPower tool [6, 7, 11] to support power consumption evaluation and energy profiling for multiprocessor platforms when using these two processor models. This integration allows to address some issues, for example: to characterize power consumption of platform processors during execution of a program (or a set of programs), to detect power consumption bottlenecks or to compare different architectures with respect to energy efficiency. After the simulation, the MPSoCBench generates a report with power consumption information by each *core* and also stores details of the estimates in files.

4 MPSoCBench Characterization

We performed our experiments on an Intel i7 860 2.8GHz, with 8 GB RAM, running Ubuntu Linux 12.10 64 bits operating system, gcc 4.7 for the simulators and gcc 3.3.1 for the cross-compilers. We chose some possible configurations to characterize the applications in the benchmark.

4.1 Simulation Time

The Tables 1 to 6 show the execution time of each application running on all available multicore platforms, varying the processor model and the interconnection; each value in the table is the average of three different simulations and the relationship between standard deviation the values are between 0.4% and 8.6%. The values that were not presented in the table correspond to invalid configurations, due to the limitations of the parallelization technique of the original benchmark.

The simulation time increases as the number of processors in the platforms increases, as expected since SystemC is limited to run on a single core, even if the hardware contains more processing resources. We have divided all configurations into three groups of applications according to their increase in time of simulation in relation to increases the number of processors; The weak scaling group shows the weak scaling, and strong scaling shows strong scaling, and group Multi combines several applications of the two previous groups; then, it is not classified as strong or weak scaling. We can note that the simulation time increases for both application groups, although the increase in the applications in weak scaling group is higher. There is

Table 1: Average simulation time, in seconds, for three distinct execution of each software on each platform using the router interconnection device and PowerPC and MIPS processors

Applications		PowerPC							MIPS						
		01	**02**	**04**	**08**	**16**	**32**	**64**	**01**	**02**	**04**	**08**	**16**	**32**	**64**
Weak scaling	**basicmath**	2	3	5	11	21	56	139	2	3	5	10	21	54	121
	lu	13	18	29	57	116	-	-	11	16	25	21	103	-	-
	sha	1	2	3	7	14	36	127	1	2	4	8	17	45	146
	water	25	38	54	73	303	-	-	32	33	49	67	275	-	-
Strong scaling	**dijkstra**	1	1	2	2	5	13	40	1	1	1	2	5	13	42
	fft	24	32	51	93	173	-	-	21	29	44	80	152	-	-
	stringsearch	19	19	20	21	23	28	45	29	29	30	30	31	35	44
	susancorners	17	18	19	21	29	66	-	19	20	20	21	30	65	-
	susanedges	33	42	48	53	59	114	-	35	44	50	55	58	109	-
	susansmoothing	100	101	107	138	-	-	-	19	20	20	27	-	-	-
	water-spatial	248	250	261	264	-	-	-	217	218	223	234	-	-	-
Multi	**multisoftware_p**	-	-	22	42	113	641	2170	-	-	27	53	110	592	2046
	multi_office_telecomm	-	-	68	-	-	-	-	-	-	202	-	-	-	-
	multi_net_automotive	-	-	115	-	-	-	-	-	-	69	-	-	-	-
	multi_security	-	-	381	-	-	-	-	-	-	355	-	-	-	-
	multi_8	-	-	-	542	-	-	-	-	-	-	512	-	-	-
	multi_16	-	-	-	-	940	-	-	-	-	-	-	729	-	-

one main reason for the significant increase in simulation time of the applications in the weak scaling group: each processor handles large and independent inputs, while in applications in the strong scaling group, the same input is partitioned and assigned to the available processors, reducing the job per core as the number of cores increases.

We have compared the simulation time on multicore platforms using the three different interconnection mechanisms in multicore PowerPC platforms running 11 applications. Since that the Router is the simpler and faster interconnection, we present the relation of the simulation time between the NoC-AT and the Router in Figure 4 and between the NoC-LT and Router in Figure 5. We use the simulation time using Router as the baseline in both cases.

The simulation time on platforms using the approximately timed coding style increases faster than using loosely timed coding style and this is expected because the large number of SystemC threads required to simulate the NoC nodes and wrappers, and consequently, the large number of context exchange between them. The Stringsearch benchmark presents the worst slowdown, due to the large number of

Table 2: Average simulation time, in seconds, for three distinct execution of each software on each platform using the router interconnection device and SPARC and ARM processors

Applications		**SPARC**							**ARM**						
		01	**02**	**04**	**08**	**16**	**32**	**64**	**01**	**02**	**04**	**08**	**16**	**32**	**64**
Weak scaling	**basicmath**	2	6	10	17	36	78	192	2	3	9	10	21	48	133
	lu	23	29	44	87	174	-	-	26	33	49	94	188	-	-
	sha	1	2	3	7	16	42	143	1	2	4	9	20	52	174
	water	40	38	52	72	297	-	-	45	44	62	84	323	-	-
Strong scaling	**dijkstra**	1	1	1	2	6	15	46	1	1	1	2	5	12	45
	fft	32	43	66	119	215	-	-	39	54	86	143	261	-	-
	stringsearch	21	22	22	23	24	28	38	29	29	29	30	31	35	45
	susancorners	28	28	29	30	39	72	-	26	26	27	30	39	72	-
	susanedges	83	104	118	143	136	221	-	57	69	78	84	90	160	-
	susansmoothing	33	33	38	52	-	-	-	20	20	22	30	-	-	-
	water-spatial	351	353	369	380	-	-	-	209	198	211	233	-	-	-
Multi	**multisoftware_p**	-	-	25	54	125	520	1707	-	-	30	57	115	227	491
	multi_office_telecomm	-	-	199	-	-	-	-	-	-	100	-	-	-	-
	multi_net_automotive	-	-	87	-	-	-	-	-	-	89	-	-	-	-
	multi_security	-	-	348	-	-	-	-	-	-	412	-	-	-	-
	multi_8	-	-	-	438	-	-	-	-	-	-	555	-	-	-
	multi_16	-	-	-	-	560	-	-	-	-	-	-	562	-	-

Table 3: Average simulation time, in seconds, for three distinct execution of each software on each platform using the NoC-LT and PowerPC and MIPS processors

	Applications	**PowerPC**							**MIPS**						
		01	**02**	**04**	**08**	**16**	**32**	**64**	**01**	**02**	**04**	**08**	**16**	**32**	**64**
Weak scaling	**basicmath**	2	3	5	11	27	82	224	2	3	6	12	27	66	194
	lu	22	31	54	121	272	-	-	20	26	45	100	224	-	-
	sha	1	3	6	13	28	76	270	1	3	6	12	28	78	269
	water	59	62	94	144	702	-	-	51	52	79	118	623	-	-
Strong scaling	**dijkstra**	1	2	2	5	11	32	81	1	2	2	3	8	25	91
	fft	31	48	84	167	1058	-	-	16	39	67	134	270	-	-
	stringsearch	34	35	36	40	47	64	125	51	50	51	57	65	80	128
	susancorners	27	28	31	36	56	143	-	29	30	36	37	54	129	-
	susanedges	56	71	90	117	139	314	-	59	77	85	111	121	270	-
	susansmoothing	168	168	186	277	-	-	-	33	35	36	53	-	-	-
	water-spatial	398	406	451	480	-	-	-	348	347	361	400	-	-	-
Multi	**multisoftware_p**	-	-	34	79	193	900	4041	-	-	31	86	234	380	3743
	multi_office_telecomm	-	-	97	-	-	-	-	-	-	274	-	-	-	-
	multi_net_automotive	-	-	174	-	-	-	-	-	-	108	-	-	-	-
	multi_security	-	-	558	-	-	-	-	-	-	507	-	-	-	-
	multi_8	-	-	-	828	-	-	-	-	-	-	766	-	-	-
	multi_16	-	-	-	-	953	-	-	-	-	-	-	1611	-	-

Table 4: Average simulation time, in seconds, for three distinct execution of each software on each platform using the router interconnection device and SPARC and ARM processors

Applications		**SPARC**							**ARM**						
		01	**02**	**04**	**08**	**16**	**32**	**64**	**01**	**02**	**04**	**08**	**16**	**32**	**64**
Weak scaling	**basicmath**	2	5	9	20	45	101	264	2	3	6	13	30	69	210
	lu	33	48	74	169	380	-	-	44	56	91	189	402	-	-
	sha	1	2	5	11	26	73	264	1	3	7	15	33	96	336
	water	83	78	119	175	874	-	-	110	118	173	242	1040	-	-
Strong scaling	**dijkstra**	1	2	2	4	9	29	102	1	2	2	3	8	23	97
	fft	38	57	94	185	375	-	-	54	80	132	237	458	-	-
	stringsearch	37	37	39	43	49	67	120	49	48	50	56	61	80	-
	susancorners	46	47	51	58	81	164	-	42	44	49	58	82	166	-
	susanedges	125	160	203	361	376	522	-	91	119	149	185	204	443	-
	susansmoothing	52	53	60	91	-	-	-	35	37	41	59	-	-	-
	water-spatial	502	512	529	591	-	-	-	752	777	835	938	-	-	-
Multi	**multisoftware_p**	-	-	36	91	185	785	2279	-	-	45	90	191	428	1014
	multi_office_telecomm	-	-	250	-	-	-	-	-	-	147	-	-	-	-
	multi_net_automotive	-	-	118	-	-	-	-	-	-	140	-	-	-	-
	multi_security	-	-	471	-	-	-	-	-	-	581	-	-	-	-
	multi_8	-	-	-	994	-	-	-	-	-	-	837	-	-	-
	multi_16	-	-	-	-	866	-	-	-	-	-	-	938	-	-

Table 5: Average simulation time, in seconds, for three distinct execution of each software on each platform using the NoC-AT and PowerPC and MIPS processors

Applications		**PowerPC**							**MIPS**						
		01	**02**	**04**	**08**	**16**	**32**	**64**	**01**	**02**	**04**	**08**	**16**	**32**	**64**
Weak scaling	**basicmath**	6	14	33	78	205	510	1472	5	11	21	54	138	394	1531
	lu	69	105	165	404	884	-	-	50	78	123	276	622	-	-
	sha	1	15	34	96	287	1063	1169	1	15	35	97	296	1090	4905
	water	166	205	319	483	2851	-	-	168	193	299	466	2059	-	-
Strong scaling	**dijkstra**	5	6	11	27	82	244	1022	5	6	12	25	64	247	1019
	fft	96	165	265	569	1176	-	-	73	130	206	415	871	-	-
	stringsearch	34	35	36	40	47	64	125	51	50	51	57	65	80	128
	susancorners	27	28	31	36	56	143	-	29	30	36	37	54	129	-
	susanedges	56	71	90	117	139	314	-	59	77	85	111	121	270	-
	susansmoothing	168	168	186	277	-	-	-	33	35	36	53	-	-	-
	water-spatial	398	406	451	480	-	-	-	348	347	361	400	-	-	-
Multi	**multisoftware_p**	-	-	34	79	193	900	4041	-	-	31	86	234	380	3743
	multi_office_telecomm	-	-	555	-	-	-	-	-	-	1064	-	-	-	-
	multi_net_automotive	-	-	459	-	-	-	-	-	-	436	-	-	-	-
	multi_security	-	-	2837	-	-	-	-	-	-	2634	-	-	-	-
	multi_8	-	-	-	3990	-	-	-	-	-	-	4107	-	-	-
	multi_16	-	-	-	-	10861	-	-	-	-	-	-	7533	-	-

Table 6: Average simulation time, in seconds, for three distinct execution of each software on each platform using the NoC-AT and SPARC and ARM processors

Applications		**SPARC**							**ARM**						
		01	**02**	**04**	**08**	**16**	**32**	**64**	**01**	**02**	**04**	**08**	**16**	**32**	**64**
Weak scaling	**basicmath**	6	15	26	65	133	258	1082	4	10	20	53	129	345	1197
	lu	85	126	197	397	641	-	-	179	262	437	1001	2347	-	-
	sha	1	17	34	93	287	1080	5540	2	15	47	98	293	1070	5462
	water	152	179	265	406	1590	-	-	448	542	852	1352	4867	-	-
Strong scaling	**dijkstra**	5	7	9	21	59	185	805	5	6	11	25	65	207	938
	fft	120	197	320	613	1279	-	-	224	371	643	1338	2183	-	-
	stringsearch	9	13	24	50	114	308	1065	134	163	199	285	430	766	1773
	susancorners	143	152	192	249	458	1676	-	134	152	194	258	444	1289	-
	susanedges	315	482	660	906	1160	2143	-	347	506	825	1821	2129	2743	-
	susansmoothing	162	167	212	375	-	-	-	130	179	188	330	-	-	-
	water-spatial	1055	1147	1301	1582	-	-	-	2328	2902	4280	5465	-	-	-
Multi	**multisoftware_p**	-	-	126	269		1456	3981	-	-	194	448	1060	2694	
	multi_office_telecomm	-	-	984	-	-	-	-	-	-	682	-	-	-	-
	multi_net_automotive	-	-	590	-	-	-	-	-	-	673	-	-	-	-
	multi_security	-	-	1977	-	-	-	-	-	-	2977	-	-	-	-
	multi_8	-	-	-	4441	-	-	-	-	-	-	4048	-	-	-
	multi_16	-	-	-	-	6334	-	-	-	-	-	-	6171	-	-

memory access to read its input strings; the simulation time of this benchmark on 64-core platforms with NoC-AT as interconnection is 34× greater than on 64-core platforms with Router.

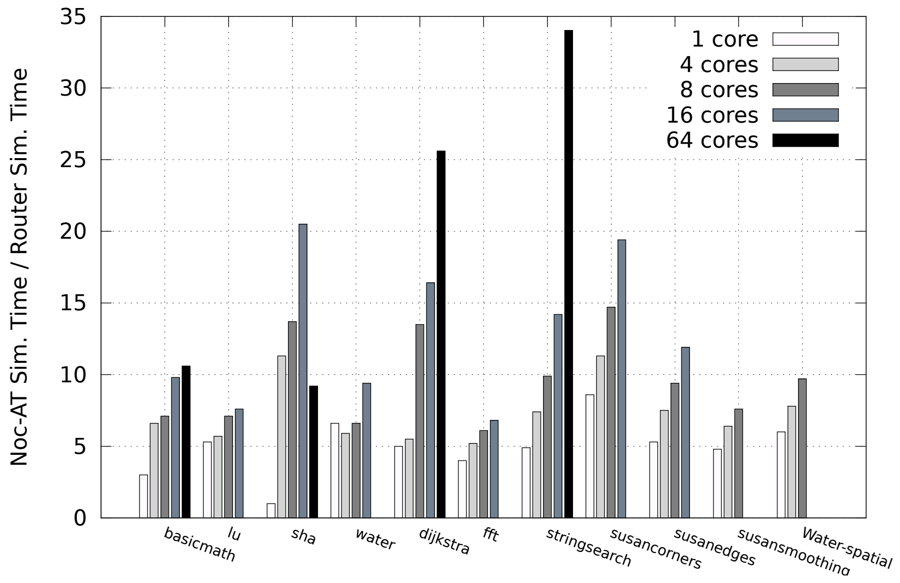


Figure 4: Relation between simulation time using NoC-AT and Router

4.2 Number of Instructions

Figures 6(a) and 6(b) characterize the number of executed instructions on a single-core platform using all four processors, running applications with lower and higher computational load, respectively. The four processor models have fundamental differences and specific optimizations that interfere directly in the number of instructions executed by each of them.

In order to evaluate the traffic between the processor and other peripherals (memory and lock devices), we monitored the `tlm_router` and counted the number of requests from processors, in a multi-core PowerPC environment; the results are in Figure 7(a). We omitted the number of router requests of the `multisoftware_s` application because its value is too high compared to others (4.57 billions). As we expect, the behavior of the router requests are dictated by the application scalability.

4.3 Network Traffic and Simulation Time Advance

In order to evaluate the rate of requests or responses packages made by the applications and the traffic inside the network, we show in Table 4.3 the results of simulating

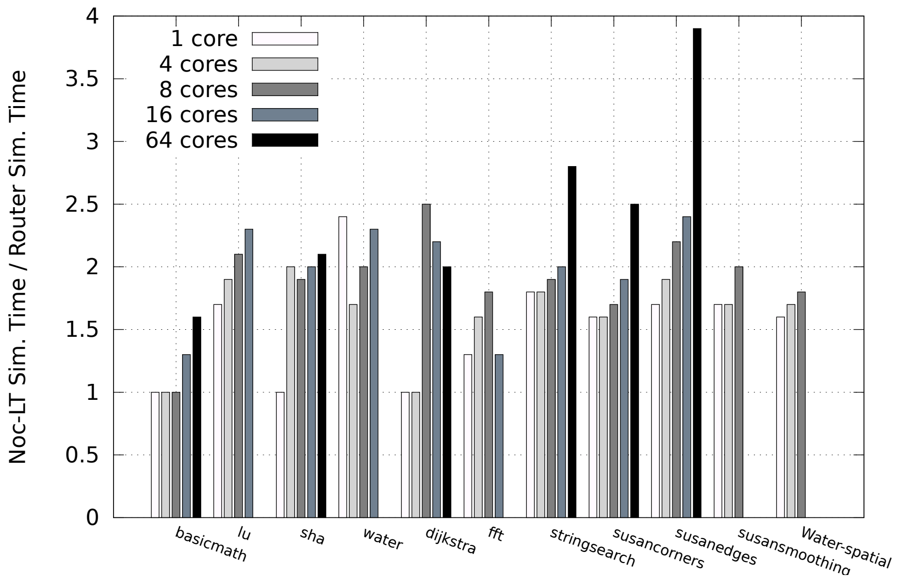


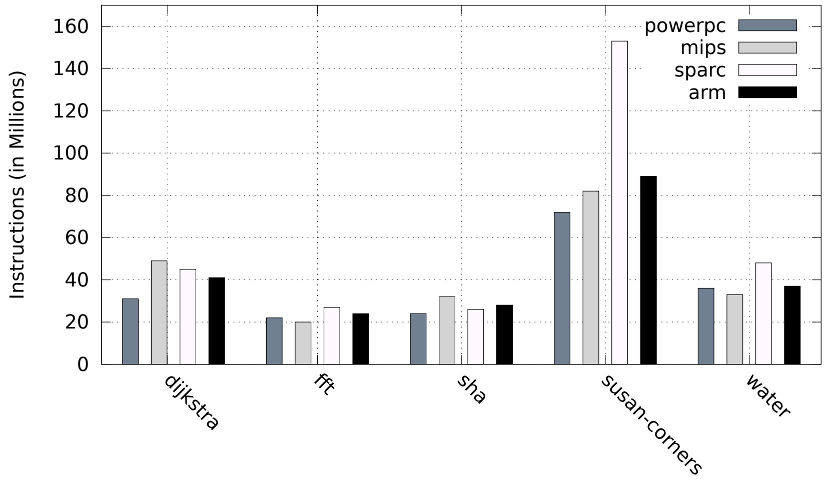
Figure 5: Relation between simulation time using NoC-LT and Router

four applications on multicore platforms with 16, 32, 64 ARM cores. We present the total number of requests made by processors and the total number of hops. In this context, the number of hops represents the number of devices in the request package path (forward path) plus the number of devices in the response package path (backward path). We choose four different applications: SHA and Basicmath are from the weak scaling group and Stringsearch and Dijkstra are from the strong scaling group.

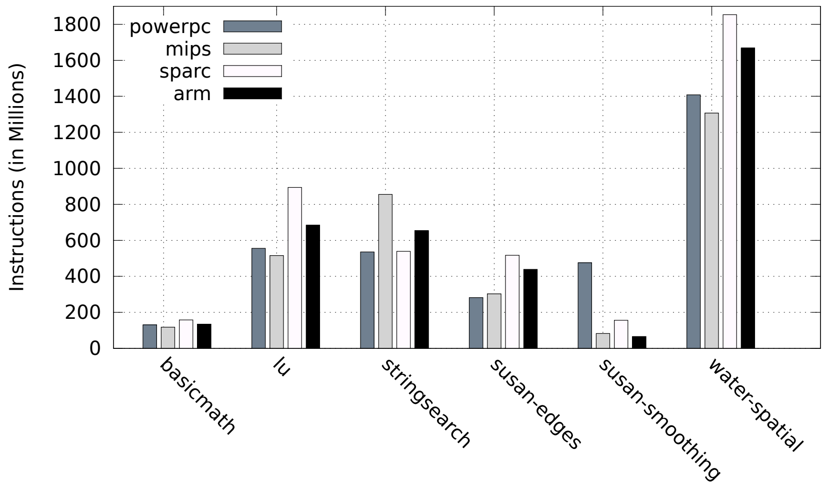
A SystemC-based simulator is driven by discrete events, so the scheduler advances simulation time to the time of the next event in a global event list. Then, it runs any processes due to run at that time or sensitive to that event. The processor models are implemented as SystemC processes and they are pre-configured in such way to advance simulation by 1 nanosecond per instruction, and each routing node is configured to cause the same delay in each package routing. We can access the final simulation time by calling the method `sc_time_stamp()` at the end of simulation. We also show in Table 4.3 the final simulation time advance provided by the MPSoCBench reports related to the same four applications running in ARM multicore platforms.

4.4 Energy Profiling

After the simulation, MPSoCBench generates a report with instruction count per core, simulation time, and power consumption per core, when available. It also outputs power consumption details, which can be used for power profiling as partially plotted in Figures 8(a), 8(b), 9(a), and 9(b).

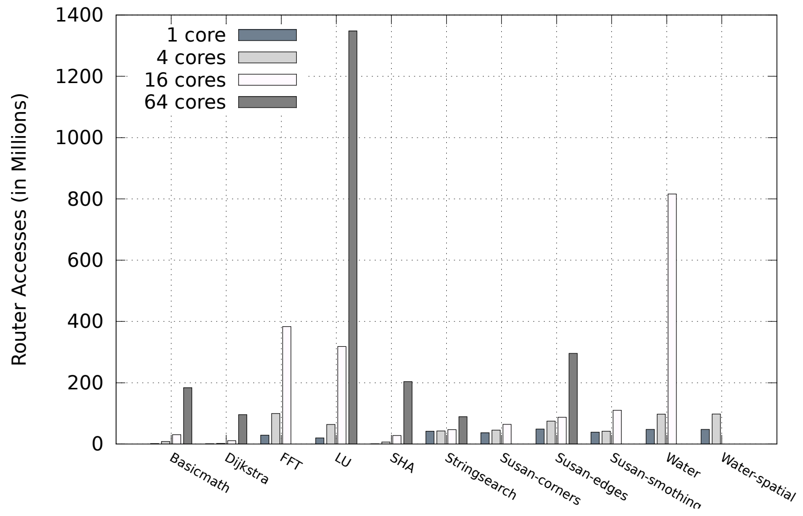


(a) Number of instructions executed in applications with lower computational load

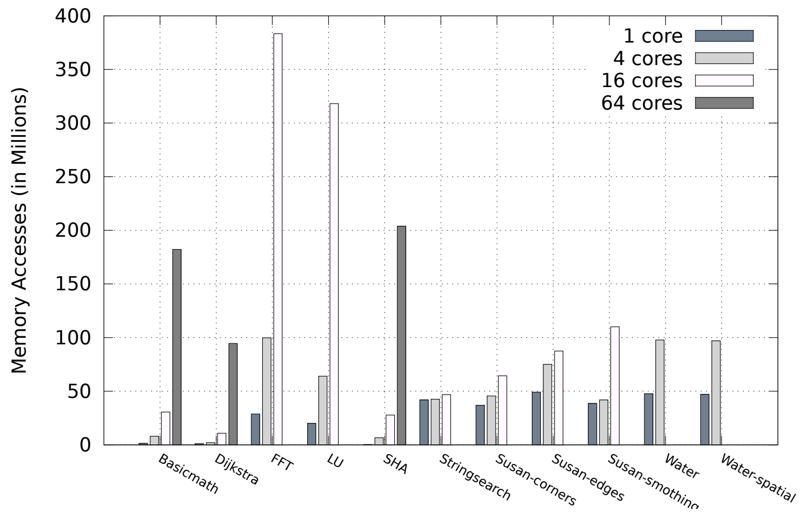


(b) Number of instructions executed in applications with greater computational load

Figure 6: Number of instructions executed in single-PowerPC platforms



(a) Number of router requests



(b) Number of memory access

Figure 7: Number router requests in single-PowerPC platforms

Table 7: Number of requests to the network and hops per request in ARM multicore platforms, running four applications

Applications	**Cores**	**Requests** (in millions)	**Hops** (in millions)	**Average** (Hops/Request)	**Advance** (milliseconds)
SHA	16	40	394	10	81
	32	116	1392	12	232
	64	431	7042	16	862
Basicmath	16	20	195	10	40
	32	45	539	12	90
	64	107	1773	16	214
Stringsearch	16	50	487	10	100
	32	73	884	12	147
	64	129	2144	13	259
Dijkstra	16	7	71	10	15
	32	21	254	12	42
	64	69	1145	16	138

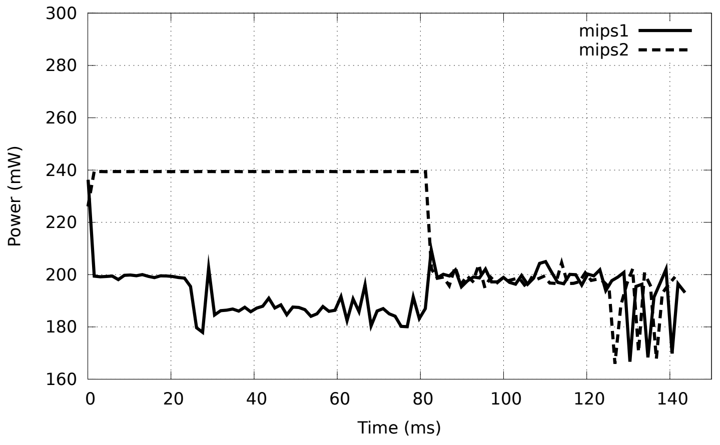
The elementary difference between power consumption of the two cores in the Figures 8(a), 9(a) and 9(b) is caused by the larger number of memory accesses to allocate and initialize the input variables, this setup phase is executed by one of the cores while the other is waiting in a barrier. In this case, we can identify different phases performed by each core.

A further observation is that the power consumption of the cores in Figure 8(b) reflects a large number of mathematical operations carried out over a small statically allocated amount of data. We intend to design power consumption models for PowerPC and ARM in the future.

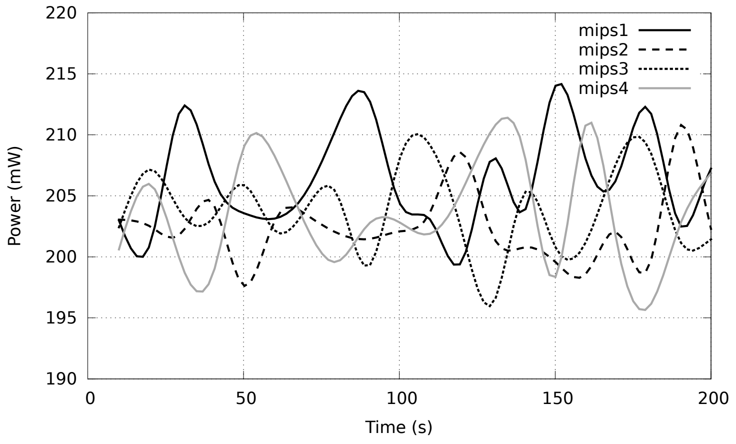
5 How to Use

The MPSoCBench execution can be controlled by the `MPSoCBench` Python script, which has the following control parameters:

- -b or –build: to build simulators
- -r or –run: to run simulators
- -nb or –nobuild: to run without recompiling the models
- -pw or –power: to enable power consumption for SPARC and MIPS platforms

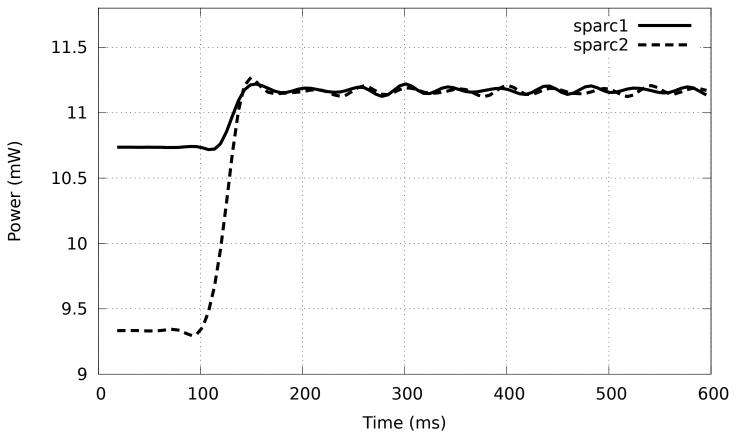


(a) Dual-MIPS platform running FFT

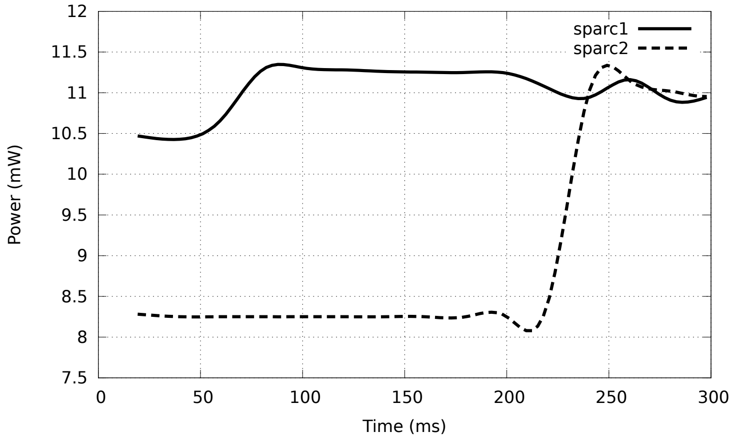


(b) Quad-MIPS platform running Basicmath

Figure 8: Power consumption characterization of MIPS platforms [Power(mW) x Time (s)]



(a) Dual-SPARC platform running Dijkstra



(b) Dual-SPARC platform running FFT

Figure 9: Power consumption characterization of SPARC platforms [Power(mW) x Time (s)]

- -p or –processor: to choose processor models
- -n or –numcores: to choose the number of cores (1,2,4,8,16,32,or 64)
- -s or –software: to choose the application
- -i or –interconnection: to choose the interconnection device
- -c or –condor: to enable execution on HTCondor
- -l or –clean: delete object files and the simulators previosly created
- -h or –help: help

This script creates a run-directory for each different platform and configures all Makefiles required. Some of its configuration parameters accepts the `all` option to make it easy to fully execute the benchmark. Some examples follow:

1. To build and run all programs in the 64-MIPS platform including power consumption, using a NoC-LT as interconnection device:

   ```
   $ ./MPSoCBench -r -s=all -p=mips -n=64 -pw -i=noc-lt
   ```

2. To build (without running) the Dijkstra benchmark in the 16-core platforms, using all available processors and a router as interconnection device:

   ```
   $ ./MPSoCBench -b -s=dijkstra -p=all -n=16 -i=router
   ```

3. To build and run the FFT benchmark in all multicore platforms, using the SPARC processor model including power consumption estimation, and NoC-AT as interconnection, able to run into a HTCondor cluster:

   ```
   $ ./MPSoCBench -r -s=fft -p=sparc -pw -n=all -i=noc-at -c
   ```

4. To run all available configurations without unnecessary recompilation:

   ```
   $ ./MPSoCBench -r -s=all -p=all -pw -n=all -i=all --nobuild
   ```

6 Conclusion

This report introduced the open-source benchmark called MPSoCBench, which is a scalable, configurable and extensible set of MPSoCs, useful to improve development and evaluation of designs in the MPSoC ecosystem, using well known methodologies and tools. MPSoCBench is composed of 3 different platforms, each with a different interconnection device, easily configurable from 1 to 64 cores of 4 different available processor models, capable of running 13 different parallel applications. The total combined size reaches 876 distinct configurations.

We characterized the simulation time, instruction counts, bus traffic and power consumption for different configurations. The benchmark is released as open-source and it is available in two ways: a virtual machine with all infrastructure ready for use, and as source code. The tutorials for installation of all tools are provided in the MPSoCBench website (`http://www.archc.org/benchs/MPSoCBench`).

References

[1] OSCI TLM-2.0 Language Reference Manual. Software version: TLM 2.0.1 Document version: JA32.
http://www.systemc.org.

[2] IEEE Std 1666^{TM} Standard SystemC Language Reference Manual. IEEE Computer Society, January 2012.

[3] R. Azevedo, S. Rigo, M. Bartholomeu, G. Araujo, C. Araujo, and E. Barros. The ArchC Architecture Description Language and Tools. In International Journal of Parallel Programming. Vol. 33, No. 5, pages 453–484. October 2005.

[4] L. Benini, D. Bertozzi, F. Menichelli, and M. Olivieri. MPARM: Exploring the Multi-Processor SoC Design Space with SystemC. In Journal of VLSI Signal Processing, volume 41, pages 169–182. 2005.

[5] S. Boukhechem, E. Bourennane, and A. Samahi. Co-simulation Platform Based on SystemC for Multiprocessor System on Chip Architecture Exploration. In Proceedings of IEEE International Conference on Microelectronics (ICM). 2007.

[6] M. Guedes, R. Auler, E. Borin, and R. Azevedo. An ArchC approach for automatic energy consumption characterization of processors. In Proceedings of IEEE International Symposium of Rapid System Prototyping. October 2012.

[7] M. Guedes, R. Auler, L. Duenha, E. Borin, and R. Azevedo. An automatic energy consumption characterization of processors using ArchC. In Journal of Systems Architecture. June 2013.

[8] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown. MiBench: A free, commercially representative embedded benchmark suite. In Proceedings of IEEE 4th Annual Workshop on Workload Characterization, held in conjunction with The 34th Annual IEEE/ACM, pages 03–14. December 2001.

[9] S. M. Z. Iqbal, Y. Liang, and H. Grahn. ParMiBench: An Open-Source Benchmark of Embedded Multiprocessor Systems. In IEEE Computer Architecture Letters, Vol. 9, No.2, pages 45–48. 2010.

[10] F. Klein, G. Araujo, R. Azevedo, R. Leao, and dos Luiz Santos. An efficient framework for high-level power exploration. In MWSCAS 2007: Proceedings of the 50th Midwest Symposium on Circuits and Systems, pages 1046–1049, Aug 2007.

[11] F. Klein, G. Araujo, R. Azevedo, R. Leao, and L. Santos. An Efficient Framework for High-Level Power Exploration. In Proceedings of the 50th Midwest Symposium on Circuits and Systems - MWSCAS 2007, pages 1046–1049. August 2007.

[12] R. Leupers and O. Temam. Processor and System-on-Chip Simulation. Springer New York, 2010.

[13] A. Mello, I. M. aind A. Greiner, and F. Pecheux. Parallel Simulation of SystemC TLM 2.0 Compliant MPSoC on SMP Workstations. In Proceedings of Design, Automation and Test in Europe - DATE, pages 606–609. March 2010.

[14] P. Paulin, C. Pilkington, M. Langevin, E. Bensoudane, D. Lyonnard, O. Benny, B. Lavigueur, D. Lo, G. Beltrame, V. Gagné, and G. Nicolescu. Parallel Programming Models for a Multiprocessor SoC Platform Applied to Networking and Multimedia. In Proceedings of IEEE Transactions on Very Large Scale Integration (VLSI) Systems, volume 14, no.7. July 2006.

[15] S. Rigo, R. Azevedo, and L. Santos. Eletronic System Level Design. Springer, 2011.

[16] V. Tiwari, S. Malik, and A. Wolfe. Power analysis of embedded software: A first step towards software power minimization. In IEEE Transactions on VLSI Systems, vol. 2, page 437–445. 1994.

[17] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta. The Splash-2 Programs: Characterization and Methodological Considerations. In Proceedings of the 22nd International Symposium on Computer Architecture - ISCA'95, pages 24–36. ACM, 1995.