2016 IEEE 10th International Symposium on Embedded Multicore/Many-core Systems-on-Chip

Communication-Based Power Modelling for Heterogeneous Multiprocessor Architectures

Baptiste Roux*, Matthieu Gautier†, Olivier Sentieys*, Steven Derrien†
*Inria, Irisa, University of Rennes 1, France
†University of Rennes 1, Irisa, Inria, France
Email: baptiste.roux@inria.fr

***Abstract*—Programming heterogeneous multiprocessor architectures is a real challenge dealing with a huge design space. Computer-aided design and development tools try to circumvent this issue by simplifying instantiation mechanisms. However, energy consumption is not well supported in most of these tools due to the difficulty to obtain fast and accurate power estimation. To this aim, this paper proposes and validates a power model for such platforms. The methodology is based on micro-benchmarking to estimate the model parameters. The energy model mainly relies on the energy overheads induced by communications between processors in a parallel application. Power modelling and micro-benchmarks are validated using a Zynq-based heterogeneous architecture showing the accuracy of the model for several tested synthetic applications.**

I. Introduction

The design of embedded systems is facing two conflicting challenges. Applications (e.g., telecommunications, multimedia) require increasingly computation power to follow consumer requirements, while the same have, by definition, limited power budget and their autonomy is a commercial stake. In recent years, the advent of Multiprocessor System-on-Chip (MpSoC) architectures allowed for a gain at both levels. The simplification of processor cores brought improvement in power consumption per operation, while the multiplication of cores brought improvement in performance. Furthermore, advances in silicon technologies came with an increase of integration densities, while the advent of the dark silicon issue [1] led to the integration of specialized hardware accelerators, such as Field-Programmable Gate Array (FPGA), and to the rise of Heterogeneous MpSoC (HMPSoC).

HMPSoCs are a new class of architecture which introduces complex and hard-to-solve issues on software/hardware partitioning and task mapping, therefore leading to a huge design space. Consequently, when energy consumption is a key requirement of the application, this solution space must be explored, early in the design phase, with a fast and accurate power estimation tool.

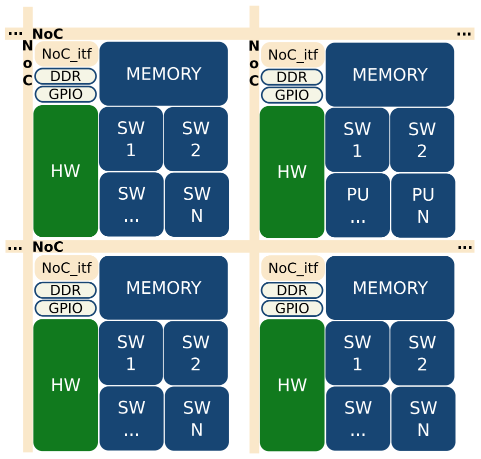
Power modelling is not a new topic. The need for power estimation tools for complex processor architecture design led to the development of Wattch [2] and its successor McPAT [3]. These tools are modelling frameworks, which take micro-architectural and technology information into account to build an architecture power model. Timing and power consumption parameters can be extracted and used as inputs of architecture simulators, such as [4][5][6]. This combination of tools requires the cycle accurate simulation of application execution to compute their power consumption. In [7] Schürmans *et al.* propose an electronic system level power modeling approach that raises the required simulation accuracy to transaction level modeling. The time spent for those power estimations is directly linked to the application size, and remain prohibitively high. Other power modelling tools were introduced with the aim of raising the abstraction level to lower the computational complexity of the estimation. In this perspective, [8] and [9] proposed to model architectures at *instruction-level* or at *functional-level* respectively. These approaches reduce the computation time with a low penalty on accuracy. However they can not be directly applied to multiprocessor architectures. Kahng *et al.* [10] introduce a Network-on-Chip (NoC) model that could be combined with the previous tools to fulfill this gap. Rethinagiri *et al.* [11] proposed a fast virtual platform emulation to address power estimation in multiprocessor systems. It combines a functional-level power analysis with a fast platform simulator to compute application power consumption.

All theses approaches require one simulation per version of the application, which still leads to long estimation time, and they can therefore not be applied to a fast task mapping process when HMpSoC architectures are targeted. The aim of this paper is to propose a fast and accurate power modelling framework able to compute, for each mapping solution, the power consumption with no extra development delay. To answer the above challenge, this paper introduces a communication-based power model that addresses a wide range of heterogeneous architectures. Indeed, when comparing two different parallel application mappings over a HMpSoC architecture, their energy consumption mainly differs from the communication cost and the static power. The contributions of the paper are the followings:

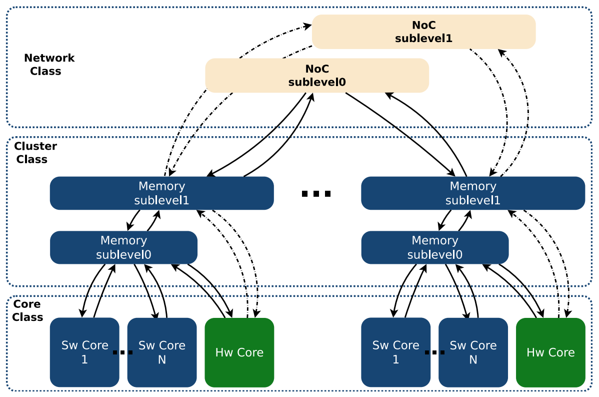
- a new power model of HMpSoC architectures mainly based on the communication cost between processors,
- a methodology based on micro-benchmarking for estimating the model parameters,
- a validation of the proposed power modelling framework on a real and representative hardware platform: the Zynq architecture from Xilinx.

This paper is organized as follows. Section II introduces HMpSoC architectures and defines a generic model based on

978-1-5090-3531-1/16 $31.00 © 2016 IEEE
DOI 10.1109/MCSoC.2016.27



(a) HMpSoC architecture



(b) Memory hierarchy as an oriented tree.

Fig. 1: Generic representation of a HMpSoC.

inter-task communications and memory hierarchy structures. Section III presents the core of our power model. Section IV shows how to determine the parameters of our power model using micro-benchmarks running on HMpSoC architectures. Section V presents the validation of the model on a real HMpSoC architecture using the Zynq platform from Xilinx. Finally, conclusions are given in Section VI.

II. Heterogeneous multiprocessor architectures

A. Generic architecture

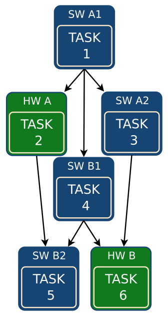
MpSoCs are generally composed of a set of memories, processors, interconnecting elements and I/O peripherals. When associated with specialized hardware accelerators, MpSoC are referred to as heterogeneous, in the sense that they combine software (SW) processors with hardware (HW) accelerators. A generic representation of a HMpSoC is formalized and shown in Fig. 1a. This HMpSoC architecture is built around clusters linked together through NoCs. Each cluster is composed of up to N SW cores coupled with HW accelerators of size S. At the cluster level, the communications occur through shared memory banks. From this description, different families of HMpSoC can be build depending on the mapping of the integrated HW area in the architecture. When it is placed at the processor level in each cluster ($S \neq 0$), *Distributed HMpSoC* are obtained. They enable fast communication between the SW and HW parts. Consequently, the maximum area of a hardware accelerator is reduced. When the HW area is placed at the cluster level, two cluster types are involved: the SW one with $N \neq 0$ and $S = 0$ and the HW one with $N = 0$ and $S \neq 0$. HW area is therefore shared between SW clusters. These *Shared HMpSoCs* induce an increase in communication time and latency between the SW and HW parts. On the other hand, the total HW size could increased and sharing the accelerators between SW clusters becomes possible.

This generic architecture has been extracted from the study of three representative and commercially available architectures with multiprocessor and/or heterogeneous features as described below. As communications mainly rely on the memory hierarchy of the architecture, the memory hierarchy will be particularly reported.

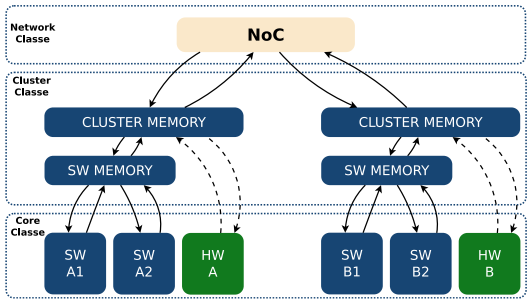
a) Kalray MPPA: Kalray's Massively Parallel Processor Array (MPPA) architecture [12] is a homogeneous MpSoC which is mainly composed of 256 VLIW (Very Long Instruction Word) processors gathered into 16 clusters. All clusters are linked together through two NoCs, one with low bandwidth for control information and one with high-bandwidth for data communications. External communications are managed through four I/O clusters. Each cluster contains 16 5-way VLIW processors and NoC interface. On each side of the cluster array, an I/O cluster provides access to external Double Data Rate (DDR) memory banks and to PCIe and Ethernet interfaces.

b) Tilera TileGx: The TileGx architecture [13] includes from 36 to 72 tiles, each tile being composed of a 3-way 64-bit processor with 3-level cache memory. Tiles are connected together through an *iMesh* NoC. This architecture can be extended by connecting multiple chips through a shared DDR memory or Ethernet interfaces. Including hardware accelerators, such as some Field-Programmable Gate Array (FPGA), establishes a shared HMpSoC architecture.

c) Xilinx Zynq: The Zynq architecture from Xilinx [14] is representative of heterogeneous targets as it combines two ARM cortex A9 processors with an FPGA fabric. The Zynq computation cores communicate through different memory levels: L2 cache and DDR. Each level can be accessed from two channels: one for SW and one for HW. Furthermore, synchronization and configuration communications can occur from a dedicated channel without memory bank access. A distributed HMpSoC can be built using Zynq chip as a target cluster, Zynq chips being connected together through Ethernet interface.



(a) Task graph.



(b) Target HMpSoC memory hierarchy.

Producer	Consumer	Memory Bank Used
1	2	Cluster Memory A
1	3	SW Memory A
1	4	NoC
2	5	NoC
3	6	NoC
4	5	SW Memory B
4	6	Cluster Memory B

(c) Upper communication channels needed.

Fig. 2: Example of communication channel extraction for a 6-task application.

B. *Memory tree abstraction*

The memory hierarchy of a HMpSoC can be classified in three main levels: network, cluster, and core. A representation of this memory hierarchy as an oriented tree, equivalent to Fig. 1a, is shown in Fig. 1b. Each level contains sublevels, with the following properties:

- At the network level, sublevels are separated such that each sublevel can be used independently.
- At the cluster level, sublevels are mixed such that sublevels are chained but can be accessed at each sublevel.
- At the core level, sublevels are chained. Accessing sublevel at depth L crosses the $L-1$ upper sublevels.

To describe precisely a target architecture, the characteristics of each memory class have to be defined. Important parameters that need to be measured are the energy and the time required to transmit a given number of bytes through each edge of the memory hierarchy oriented tree. A method that provides the identification of those parameters on real hardware architectures is described in Section IV. This generic architecture and memory abstraction will be used to derive the proposed communication-based power model.

C. *The need for a communication-based power model*

A fast power model for task mapping exploration is proposed in this paper based on a memory-centric view of heterogeneous architectures. Considering an application which can be perfectly parallelized, its execution on multiple threads distributed on multiple processors reduces the execution time but not the total amount of computation. Computations are equally distributed between the processors and therefore the amount of computation can be considered as independent of the parallelism degree. However, when multiple threads are computed in parallel, the amount of communications and synchronizations is directly linked to the number of execution threads.

Assuming that each thread is executed on the same type of target processor, the energy consumed for executing the application in its sequential and parallel versions solely differs from the communication cost (and slightly from the static power). Therefore, a power model which is able to evaluate the communication cost and execution time to quickly derive the power consumption of a parallel application, is essential for a fast design exploration of the mapping space. In this paper, we propose and validate a communication-based power model, suitable for HMpSoC, with a fine-grained resolution and a low computational cost.

III. Communication-Based Power Model

The energy consumption of an application executed on a heterogeneous multicore architecture depends on three main sources: the dynamic energy consumption used for computations, the static energy dissipated during execution time, and the energy used for communications between processing cores.

Any parallel applications can be divided into N_{Tk} concurrent computational tasks that could be executed in parallel. Those tasks, represented as nodes, are linked together into a graph, where edges depict communications between tasks. Execution of tasks is atomic: synchronization mechanisms and IO accesses are held on communication edges. For each pair of tasks, the amount of required communications is considered as known. As the content of these tasks is sequentially executed, each computational task Tk_k contains the same amount of computations regardless of its mapping in the task graph. Thereby the energy used for computation in each task could be calculated as well as its execution time. After calculation of these values, the task mapping space begins to be explored over the N_{Tk} tasks. For each graph corresponding to a mapping solution, the total energy E_t is

$$\begin{aligned} E_t = E_{stat} + \sum_{k \in N_{Tk}} E_{comp}(Tk_k) \\ + \sum_{(i,j) \in N_{Tk}^2} E_{com}(Tk_i, Tk_j), \end{aligned} \tag{1}$$

where E_{stat} is the static energy consumption, $E_{comp}(Tk_k)$ is the energy of computations in task Tk_k and $E_{com}(Tk_i, Tk_j)$ is the energy used for communication between tasks Tk_i and Tk_j.

A. *Computation energy cost*

The energy consumption $E_{comp}(Tk_k)$ of each computation task is supposed to be known. In the case of heterogeneous architectures, $E_{comp}(Tk_k)$ is computed for each kind of available computational cores. Since this step is executed only once, the estimation time required by the power evaluation tools is not an issue. In the experiments, theses values will be directly measured on the real SW/HW execution.

B. Communication energy cost

Depending on the task mapping, the communications between tasks depend on their allocation to a specific memory hierarchy level. For example, the task graph represented on Fig. 2a, composed of six tasks with two of them being hardware (HW) compatible, is considered. This task graph is mapped over the target HMpSoC shown on Fig. 2b, which is composed of two clusters connected through a one-channel NoC. Each cluster contains two software (SW) cores and one hardware core, associated with a two-level memory hierarchy. Following the task mapping, the memory bank used for each communication flow can be inferred. Fig. 2c shows the communications needed for our example.

When communications are mapped into memory, a solving function can be used to compute the communication cost. Let $C(Tk_i, Tk_j)$ be the set of communication channels crossed from task Tk_i to task Tk_j via the required memory bank. The communication energy can be expressed as

$$E_{com}(Tk_i, Tk_j) = \sum_{c \in C(Tk_i, Tk_j)} e_{0_c} + e_{1_c} \times bytes(Tk_i, Tk_j), \tag{2}$$

where e_{0_c} and e_{1_c} are energy parameters of the c^{th} crossed channel. $bytes(Tk_i, Tk_j)$ returns the number of bytes communicated from Tk_i to Tk_j.

In the same way, the communication time T_{com} can be computed as

$$T_{com}(Tk_i, Tk_j) = \sum_{c \in C(Tk_i, Tk_j)} t_{0_c} + t_{1_c} \times bytes(Tk_i, Tk_j), \tag{3}$$

where t_{0_c} and t_{1_c} are the crossed channel time parameters. Section IV introduces a method to determine those parameters for a target architecture.

C. Static energy cost

Once the execution time of each computation task and each communication is determined, the overall execution time T_{exec} can be computed as the critical path in the mapping graph weighted with execution time of computation and communication. Then, the static energy consumption can be deduced as

$$E_{stat} = T_{exec} \times P_{stat}, \tag{4}$$

where P_{stat} is the static power consumption that could be measured with the micro-benchmark based method presented in the next section.

IV. Model parameter estimation using micro-benchmarks

The proposed communication-based power model relies on HW parameters of the target architecture that are not necessarily provided by chip manufacturers. This section presents a method that enables the extraction of each of those parameters on real hardware multicore architectures.

A. Micro-benchmarking

The proposed methodology relies on the use of micro-benchmarks (μbench). A micro-benchmark is a simple and synthetic application that aims at stressing a specific part of the execution architecture. Equations (2) and (3) show that the energy and time used by a communication could be represented as a multi-parameter function in which each parameter represents a crossed channel. To obtain the value of those parameters, a solution is to compute the partial derivation following each parameter. This is the aim of micro-benchmark applications.

To fulfill this purpose, each μbench is designed to focus on a specific communication channel or a specific memory bank following the following properties:

- Selectivity: μbenchs only stress a specific communication channel, as much as possible.
- Intensity variability: μbenchs stress a communication channel with different intensity.

B. General structure of a μbench set

Using the architecture shown in Fig. 2b, each cluster is composed of three kinds of communication channels: SW Core to SW Memory; SW Memory to Cluster Memory; HW core to Cluster Memory. There is another channel between clusters and NoC. To determine the model parameters of this architecture, a set of micro-benchmarks composed of four subsets must be built as follow:

- SwChannel: this subset focuses on communication cost between processing core and SW memory level. It generates read or write accesses in an array allocated in SW memory.
- HwChannel: this subset focuses on communication cost between HW accelerators and cluster memory. It generates read or write accesses in an array allocated in cluster memory.
- IntraCluster: this subset focuses on communication cost between SW memory and cluster memory. Theses parameters can not be measured directly. Instead the μbench generates read or write accesses in an array allocated in cluster memory from the processing core and then deduces Intracluster parameters by subtracting the SwChannel value.
- InterCluster: this subset focuses on communication between clusters and generates data transfers through the NoC.

All these micro-benchmark executions are parameterized with the size of the data to communicate.

Algorithm 1: Generic micro-benchmark structure.

```
Data: $scaleFactor$, $size$
initBenchmarkEnv()
startPowerMeasure()
for $iteration$ in $scaleFactor$ do
    openCommunicationChannel()
    producer = spawnProducerThread($size$)
    consummer = spawnConsumerThread($size$)
    waitThread(producer, consumer)
    closeCommunicationChannel()
end
stopPowerMeasure()
writePowerMeasureToFile()
```

Communication time is an order of magnitude smaller than the usual power measurement time resolution. To overcome this issue and limit measurement noise, it is necessary to build the μbenchs over large number of communications. For this reason, μbenchs are composed of three parts: opening, kernel, and closing. The opening part is responsible of SW and HW initialization, and then micro-benchmark iterates $scaleFactor$ time on the kernel. The kernel part generates a communication of $size$ bytes over the target channel. Then, the closing part retrieves power measures and logs them into a file. Algorithm 1 presents a generic micro-benchmark structure.

V. Power Modelling of the Zynq Architecture

A. Experimentation infrastructure

For our experiments, the Zc702 Zynq board provided by Xilinx is used with Linux Operating System. Power measurements are done through the *Power Management Bus (PMBus)* embedded on the board. PMBus is an open standard power management protocol that enables the communication with the power converter and other devices on the board. This protocol could be used to set the output voltage of the power converter or to retrieve the output voltage and current. On the Zc702 board, PMBus enables to read eight input power rails of the Zynq SoC. PMBus values can be accessed through two methods: using ARM peripheral with Linux interruption, or using full HW mechanisms implemented in the programmable logic. In our experiments, the second approach is used since HW power consumption is more predictable that SW one.

B. Estimation of communication parameters

A micro-benchmark set is used on the Zynq platform to extract the communication parameters of such architecture. The Zynq architecture has been introduced in Section II. Fig. 3 shows the memory hierarchy tree for a distributed HMp-SoC network composed of several Zynq chips interconnected through an Ethernet network. The Zynq embedded processors can access to different memory hierarchy levels with various channels, as listed below:

- Access to L2 cache memory:
  - SW channels use standard memory access.
  - HW channels use *Advanced Coherency Port (ACP)*.
- Access to DDR memory:
  - SW channels use standard memory access.
  - HW channels use *High Performance Port (HP)*.

Unmapped memory channels can also be used. Heterogeneous communications occur without memory bank access through *General Purpose Port (GP)* using two synchronization modes: pooling and ARM *Interrupt ReQuest (IRQ)*.

In order to extract the communication cost of these channels, a set of six micro-benchmarks is used, as detailed below:

- CL1: for channel between SW cores and the first memory level. It generates read or write operations from SW core in an array located in the L1 cache.
- CL2: for channel between SW cores and the second memory level. It generates read or write operations from SW core in an array located in the L2 cache.
- DDRM: for channel between SW cores and the external memory. It generates read or write operations from SW core in an array located in the DDR.
- HPx: for channel between HW core and the external memory. It generates read and write operations from HW core in an array located in the DDR.
- ACP: for channel between HW core and the second memory level. It generates read and write operations from HW core in an array located in the L2 cache.

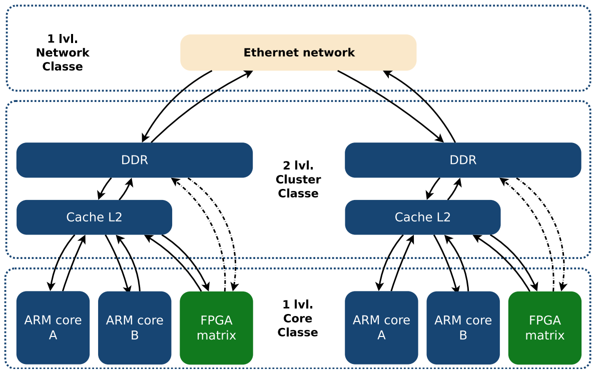
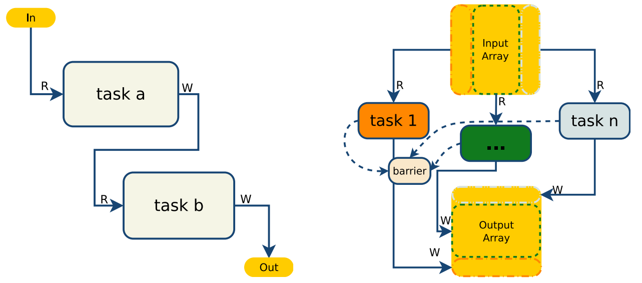


Fig. 3: Memory hierarchy of a network of Zynq clusters.



(a) Pipeline manner.   (b) Shared manner.

Fig. 4: Communication pattern.

- GPx: for unmapped communications on GP. It generates ping-pong control flow between SW and HW with and without IRQ enabled.

Each μbench family listed above, except GPx, is executed on two configurations to illustrate the memory line phenomenon. The first configuration generates cache misses on each memory access, while the second one retrieves a full memory line between two misses.

Two extra micro-benchmarks are written to measure the communication cost at a coarser grain, since the previous set cannot consider communication side effects such as synchronizations. These micro-benchmarks expose synchronization impact on consumption through two communications patterns, named *pipeline* and *shared*. They are illustrated in Fig. 4, and explained thereafter.

a) Pipeline pattern: When a set of tasks requires to process the same input sequentially, the *pipeline* communication pattern is involved. The μbench uses two tasks to illustrate this pattern (Fig. 4a). The first task consumes data from the input channel and produces output for the second one. The second task reads this data and produces results that are sent on the output channel.

b) Shared pattern: This pattern appears in data-parallel applications. The μbench is composed of two tasks working on the same input data block (Fig. 4b). Theses two tasks read part of input data and produce their own data chunk that will be updated in the input block. Before updating the input block, tasks are synchronized with a barrier.

Fig. 5 and 6 show execution time and power for access (read and write) on the CL2 memory bank. The results are given for different data size (in bytes). The values are extracted after 20 independent executions of the micro-benchmark. Results show that the variance of the measurements is quite small. The power consumption is nearly constant during the execution and the execution time could therefore be approximated with a linear function. In the following, the execution time and the energy cost of communication are approximated as a linear function $f(bytes) = a \times bytes + b$, where the values a and b are respectively the dynamic and static parts.

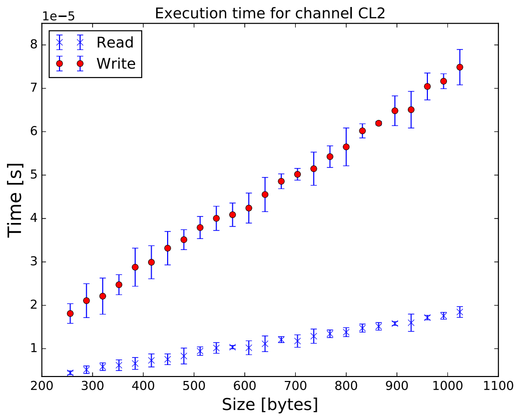


Fig. 5: Execution time on channel CL2.

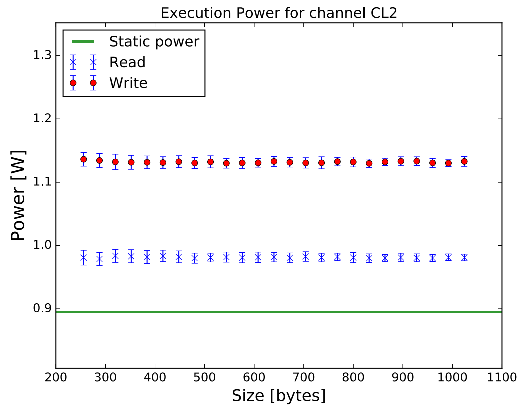


Fig. 6: Power consumption on channel CL2.

Benchmark	Time [s] $f : x \rightarrow ax + b$		Energy [J] $f : x \rightarrow ax + b$	
	a	b	a	b
HPx read	6.71e-09	7.82e-07	5.56e-11	6.49e-09
HPx write	1.34e-08	1.06e-06	1.18e-10	9.37e-09
ACP read	1.14e-08	6.07e-07	9.97e-11	5.30e-09
ACP write	3.59e-08	8.97e-07	2.78e-10	6.95e-09
GPx polling	5.41e-07	0	8.27e-09	0
GPx irq	2.85e-06	0	9.47e-08	0
DDR read	1.86e-08	7.48e-06	1.54e-09	6.16e-07
DDR read burst	7.06e-09	1.54e-06	6.07e-10	1.32e-07
DDR write	8.76e-08	-3.84e-06	2.37e-08	-1.04e-06
DDR write burst	4.40e-08	-1.34e-05	3.24e-09	-9.85e-07
CL1 read	1.82e-08	-2.95e-08	1.52e-09	-2.45e-09
CL1 read burst	1.02e-08	6.68e-09	8.40e-10	5.50e-10
CL1 write	6.03e-08	3.12e-07	4.72e-09	2.44e-08
CL1 write burst	5.05e-08	-3.73e-07	3.73e-09	-2.75e-08
CL2 read	1.76e-08	-2.69e-08	1.51e-09	-2.30e-09
CL2 read burst	9.62e-09	3.19e-07	8.01e-10	2.66e-08
CL2 write	7.19e-08	-5.46e-09	1.70e-08	-1.29e-09
CL2 write burst	5.08e-08	-4.14e-07	3.71e-09	-3.02e-08
Shared Pattern CL1	4.15e-08	1.54e-05	6.66e-09	2.46e-06
Shared Pattern CL2	4.95e-08	-4.79e-04	1.54e-08	-1.49e-04
Shared Pattern DDR	6.08e-08	-7.41e-03	2.87e-08	-3.49e-03
Pipeline pattern	2.98e-07	-1.03e-05	3.32e-08	-1.15e-06
Static Power	n.v.	n.v.	1.20e+00	n.v.

TABLE I: Extracted power model parameters for the Zynq architecture.

Table I gives the execution time and energy cost parameters extracted on the Zynq architecture with the micro-benchmark set detailed before. The validity domains of these functions ranging from [128 bytes, 1 Kbytes] for the L1 cache memory channel to [4 Kbytes, 128 Kbytes] for the DDR memory channel.

C. Power estimation validation on mutant applications

In this subsection, *mutant applications* are generated to validate the parameter values extracted from previous micro-benchmark characterization. A mutant application is an abstract application automatically generated from pattern functions. It randomly generates communication traffic over different communication channels of the target architecture. To this

mutantRank	Total bytes	Channel name											
		Cache L1		Cache L2		DDR		HPx		ACP		GPx	
mutant 1	4.56e+07	read	6.6%	read	1.3%	read	1.3%	read	1.2%	read	0.6%	polling	3.5%
		read burst	0.6%	read burst	5.5%	read burst	18.0%						
		write	6.8%	write	2.5%	write	1.1%	write	2.0%	write	0.4%	irq	0.5%
		write burst	6.6%	write burst	0.0%	write burst	41.3%						
mutant 2	5.37e+07	read	6.7%	read	2.1%	read	4.7%	read	2.9%	read	4.8%	polling	2.0%
		read burst	4.7%	read burst	0.2%	read burst	10.0%						
		write	5.0%	write	0.6%	write	0.0%	write	7.4%	write	2.0%	irq	0.5%
		write burst	4.7%	write burst	6.8%	write burst	35.0%						
mutant 3	4.10e+07	read	5.3%	read	0.0%	read	5.0%	read	3.3%	read	0.0%	polling	3.0%
		read burst	1.9%	read burst	0.0%	read burst	11.7%						
		write	7.6%	write	6.1%	write	0.6%	write	1.9%	write	8.7%	irq	1.2%
		write burst	7.2%	write burst	3.9%	write burst	32.6%						
mutant 4	4.21e+07	read	4.1%	read	2.2%	read	2.9%	read	5.0%	read	0.1%	polling	1.1%
		read burst	6.1%	read burst	1.8%	read burst	13.9%						
		write	16.3%	write	0.4%	write	7.2%	write	5.5%	write	3.4%	irq	1.4%
		write burst	1.6%	write burst	2.1%	write burst	25.2%						

TABLE II: Communications involved in mutant applications.

mutantRank	Time [s]		Energy [J]		Error	
	measured	estimated	measured	estimated	time	energy
mutant 1	2.308	2.311	2.949	2.943	0.1%	0.2%
mutant 2	2.340	2.336	3.031	2.964	0.2%	2.2%
mutant 3	2.775	2.780	3.621	3.540	0.2%	2.3%
mutant 4	2.828	2.833	3.739	3.624	0.2%	3.1%
average on 80 mutants	2.974	2.975	3.855	3.861	0.5 %	1.0 %

TABLE III: Power estimation results of mutant executions.

purpose, 12 SW and 6 HW functions were written for the Zynq architecture. Each of them generates communication traffic on a communication channel following a specific mode. A mutant generator framework then combines these functions randomly to obtain an application that stress the overall communication channels at the same time.

Table II shows the configuration of four generated mutants applications. Their configuration is summarized by the overall generated communications. The percentages of communications over the different channel are also displayed.

Table III details the results of the power consumption estimation on the four described mutant applications. The last line shows average results obtained over 80 mutants. The estimated values of Table III were computed with parameters shown in Table I and the mutant configuration. These inputs were used in the communication-based power model presented in Section III, each mutant iterates multiple time on three randomly chosen tasks. These tasks contain no computation operation, so the computation energy cost of each task can be neglected. The mutant configuration contains the amount of communication in each pair of task and was used alongside the architecture parameters to compute the communication energy cost and time. Then the mutant critical path was computed and associated with static power parameter to compute the static power cost. Table III also gives the error between measured and estimated values of the mutant application power consumption. The error obtained for the previous set of *mutant application* are lower than 3.1%. The three first lines shows that the communication channel division has no incidence on the power estimation error. These experiments enable to validate the parameter values obtain through micro-benchmarking method and the communication-based power model.

The power estimation computed with a mono-threaded python script on *Intel i5 Haswell-ult* processor takes 0.55 second on the 80 previous mutants. The obtained precision and computation time of the introduced communication-based power model shows that our model seems to be the right approach to target task mapping issues.

VI. Conclusion

This paper introduced a generic model of heterogeneous multicore architectures and a new power modeling approach focused on communication channels. The proposed power model is mainly communication-centric and aims at simplifying the task mapping step under energy or power constraints. A micro-benchmarking approach was introduced to enable the identification of the target architecture parameters defined in the power model. This identification method was experimented and validated on the Xilinx Zynq architecture.

The combination of communication-based power model and parameter estimation method based on micro-benchmarking has shown its efficiency on a large number of synthetic applications. The achieved estimation accuracy is largely enough for being used in the task mapping step, while the estimation time also fits design space exploration constraints. These results open new opportunity for future computer-aided design tools. Such fast power estimation model and methodology could be easily integrated into automatic parallelization tools (e.g. [15] [16]) to fasten and improve the development of parallel applications targeting heterogeneous multicore architectures.

REFERENCES

[1] H. Esmaeilzadeh, E. Blem, R. St. Amant, K. Sankaralingam, and D. Burger, "Dark Silicon and the End of Multicore Scaling," in *38th Annual International Symposium on Computer Architecture, (ISCA)*, June 2011, pp. 365–376.

[2] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: a framework for architectural-level power analysis and optimizations," in *27th International Symposium on Computer Architecture (ISCA)*, June 2000, pp. 83–94.

[3] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures," in *42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Dec. 2009, pp. 469–480.

[4] N. L. Binkert, R. G. Dreslinski, L. R. Hsu, K. T. Lim, A. G. Saidi, and S. K. Reinhardt, "The M5 Simulator: Modeling Networked Systems," *IEEE Micro*, vol. 26, no. 4, pp. 52–60, 2006.

[5] M. M. K. Martin, D. J. Sorin, B. M. Beckmann, M. R. Marty, M. Xu, A. R. Alameldeen, K. E. Moore, M. D. Hill, and D. A. Wood, "Multifacet's General Execution-driven Multiprocessor Simulator (GEMS) Toolset," *SIGARCH Computer Architecture*, vol. 33, no. 4, pp. 92–99, 2005.

[6] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood, "The Gem5 Simulator," *SIGARCH Computer Architecture*, vol. 39, no. 2, pp. 1–7, 2011.

[7] S. Schürmans, D. Zhang, D. Auras, R. Leupers, G. Ascheid, X. Chen, and L. Wang, "Creation of ESL Power Models for Communication Architectures Using Automatic Calibration," in *Proceedings of the 50th Annual Design Automation Conference*, 2013, pp. 58:1–58:58.

[8] V. Tiwari, S. Malik, A. Wolfe, and M. T. C. Lee, "Instruction level power analysis and optimization of software," in *9th International Conference on VLSI Design (VLSI)*, Jan. 1996, pp. 326–328.

[9] J. Laurent, N. Julien, E. Senn, and E. Martin, "Functional Level Power Analysis: An Efficient Approach for Modeling the Power Consumption of Complex Processors," in *Design, Automation Test in Europe Conference and Exhibition (DATE)*, Feb. 2004, pp. 1–6.

[10] A. B. Kahng, B. Li, L. S. Peh, and K. Samadi, "Orion 2.0: A fast and accurate noc power and area model for early-stage design space exploration," in *Design, Automation Test in Europe Conference Exhibition (DATE)*, April 2009, pp. 423–428.

[11] S. K. Rethinagiri, O. Palomar, J. A. Moreno, O. Unsal, and A. Cristal, "Vppet: Virtual platform power and energy estimation tool for heterogeneous mpsoc based fpga platforms," in *24th International Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS)*, Sept 2014, pp. 1–8.

[12] B. D. de Dinechin, R. Ayrignac, P. E. Beaucamps, P. Couvert, B. Ganne, P. G. de Massas, F. Jacquet, S. Jones, N. M. Chaisemartin, F. Riss, and T. Strudel, "A clustered manycore processor architecture for embedded and accelerated applications," in *IEEE High Performance Extreme Computing Conference (HPEC)*, Sept 2013, pp. 1–6.

[13] C. Ramey, "Tile-gx100 manycore processor: Acceleration interfaces and architecture," in *Proceedings of the 23th Hot Chips Symposium*, 2011.

[14] L. H. Crockett, R. A. Elliot, M. A. Enderwitz, and R. W. Stewart, *The Zynq Book: Embedded Processing with the Arm Cortex-A9 on the Xilinx Zynq-7000 All Programmable Soc*. Strathclyde Academic Media, 2014.

[15] A. Floch, T. Yuki, A. El-Moussawi, A. Morvan, K. Martin, M. Naullet, M. Alle, L. L'Hours, N. Simon, S. Derrien, F. Charot, C. Wolinski, and O. Sentieys, "GeCoS: A framework for prototyping custom hardware design flows," in *13th IEEE International Working Conference on Source Code Analysis and Manipulation (SCAM)*, Sept. 2013, pp. 100–105.

[16] J. Ceng, J. Castrillon, W. Sheng, H. Scharwächter, R. Leupers, G. Ascheid, H. Meyr, T. Isshiki, and H. Kunieda, "MAPS: An Integrated Framework for MPSoC Application Parallelization," in *Proceedings of the 45th Annual Design Automation Conference*, 2008, pp. 754–759.