Energy-driven design space exploration of tiling-based accelerators for heterogeneous multiprocessor architectures

Baptiste Roux, Matthieu Gautier, Olivier Sentieys, Jean-Philippe Delahaye

Energy-Driven Design Space Exploration of Tiling-Based Accelerators for Heterogeneous Multiprocessor Architectures

Baptiste Roux*, Matthieu Gautier†, Olivier Sentieys*, Jean-Philippe Delahaye‡
* Univ Rennes, INRIA, CNRS, IRISA, France
† Univ Rennes, CNRS, IRISA, France
‡ DGA MI, French MoD, France
Email: matthieu.gautier@irisa.fr

***Abstract*—Programming heterogeneous multiprocessor architectures combining multiple processor cores and hardware accelerators is a real challenge. Computer-aided design and development tools try to reduce the large design space by simplifying hardware-software mapping mechanisms. However, energy consumption is not well supported in most of design space exploration methodologies due to the difficulty to estimate energy consumption fast and accurately. To this aim, this paper proposes and validates an exploration method for partitioning tilling-based parallel applications on software cores and hardware accelerators under energy-efficiency constraints. The methodology is based on energy and performance measurement of a tiny subset of the design space and an analytical formulation of the performance and energy of an application kernel mapped onto a heterogeneous architecture. This closed-form expression is captured and solved using Mixed Integer Linear Programming, which allows for very fast exploration and results in the best hardware and software partitioning under energy constraint. The approach is validated on two application kernels using a Zynq-based architecture showing more than 12% acceleration speed-up and energy saving compared to standard approaches. Results also show that the most energy-efficient solution is application- and platform-dependent and moreover hardly predictable, which highlights the need for fast exploration tools as in this paper.**

I. Introduction

In the last decade, the design of high-performance embedded systems was pushed to increase computational power while maintaining low energy consumption. With the advent of Multiprocessor System-on-Chip (MpSoC) architectures, simplification of processor cores decreased power consumption per operation, while the multiplication of cores brought performance improvement. However, the *dark silicon* issue [1] led to the benefit of augmenting programmable processors with specialized hardware accelerators and to the rise of Heterogeneous MpSoC (HMpSoC) combining both software (SW) and hardware (HW) computational resources.

For these heterogeneous computing platforms, performance and energy consumption depend on a large set of parameters such as the HW/SW partitioning, the type of HW implementation or the communication cost. Design Space Exploration (DSE) consists in adjusting these parameters while monitoring a set of metrics (execution time, power, energy efficiency) to find the best mapping of the application on the targeted architecture. DSE is a complex task. To motivate this work, we start by illustrating the different possible trade-offs on a computation-intensive matrix multiplication kernel. Fig. 1 plots the execution time and energy needed to compute this kernel according to different HW/SW partitioning on a Zynq platform (for more details on the experimental setup, see Section VII). From the left to the right, HW/SW solutions are explored from full SW to full HW execution by varying the amount of computation allocated onto the HW cores. As shown, the best solution is not the full HW but an intermediate distribution across SW and HW cores, which cannot be trivially discovered without large design space exploration. The size of this design space moreover increases exponentially with design parameters, which highlights the need for fast exploration tools.

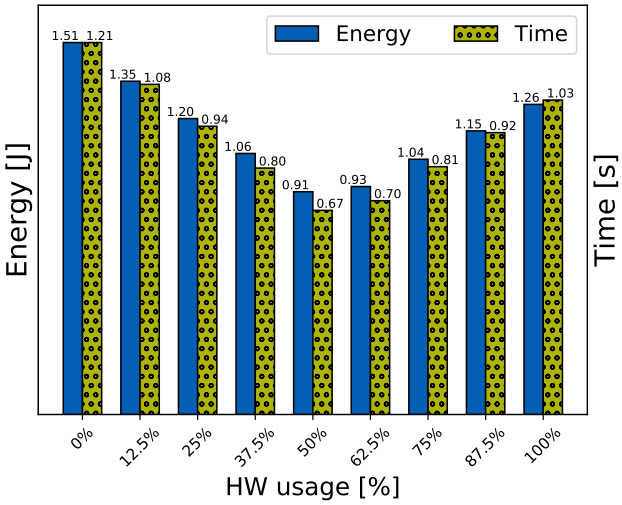


Fig. 1: Energy consumption and execution time of a matrix multiplication kernel running on a heterogenous computing platform for different HW/SW partitioning (HW usage).

DSE was already addressed by a large set of studies. Earlier works have introduced accurate tools based on simulation techniques, e.g., [2], [3], [4], [5]. However with the advent of the many-core era and the explosion of the design space size, the simulation-based approach is no longer an efficient

solution. To tackle this issue, some recent analytical methods were already proposed, as in [6], [7], [8], [9], [10], but they mainly rely on dedicated specification languages or architectures, which narrow their scope and prevent their used with legacy applications. Moreover, with the shift from performance-aware to energy-aware designs, DSE frameworks started to integrate power models, e.g., Wattch [11], SimplePower [12], and Parade [5]. These power modeling tools require simulations of the application, which drastically increases the exploration time.

In this paper, a DSE method based on an analytical power model is proposed to circumvent the computation time bottleneck of state-of-the-art power models solely based on simulation. The execution costs of tasks are directly extracted on the real architecture and inserted in a communication-based power model. The proposed method considers only the acceleration of the critical kernels that generate bottlenecks on a full application. It therefore focuses on the design of computation-intensive accelerators that can be parallelized into multiple homogeneous tasks using tiling techniques. These tasks can be executed using either SW (processors) or HW (accelerators) computational cores. For this purpose, the problem is formulated with Mixed Integer Linear Programming (MILP) and can be solved within a second. The contributions of the paper are:

- a method for quickly estimating the energy consumption of tilling-based parallel application kernels based on task computation-cost extraction and communication-based power modeling,
- an energy-driven DSE suitable for heterogeneous architectures based on an MILP formulation,
- a validation of the proposed DSE and parameter extraction for two application kernels on the Zynq heterogeneous platform from Xilinx.

This paper is organized as follows. Section III introduces an overview of the proposed DSE method. Section IV presents the extraction method of the time and energy parameters. Section V analyzes the design space size and proposes the MILP formulation as a set of constraints. Experimental setup is given in Section VI. Section VII shows the parameter extraction results and evaluates the resulting configuration of the proposed DSE. Finally, conclusions are given in Section VIII.

II. Related Work

In this section, several DSE tools that target MpSoC, HW accelerators or HMpSoC are presented. These DSE aim at evaluating system-level performance and helping the user to find efficient solutions.

The first simulation-based DSE framework targeting heterogenous architectures has been proposed by Erbas *et al.* in [2]. The Sesame framework targets system-level performance evaluation and architecture exploration of heterogeneous multimedia embedded systems. The exploration problem is multi-objective as it focuses on optimizing performance, power, and cost and is solved using both trace-driven co-simulations and a genetic solver. More recently, new frameworks have been proposed [3], [4], [5]. They mainly rely on high-level simulations combined with power model to estimate the consumed energy. For instance, Parade [5] proposes to enhance the network-on-chip simulator gem5 [13] using the power modeling tool McPat [14]. The major drawback of any simulation-based approach is their inability to cover a large design space, since each simulation evaluates only one design point.

On the other hand, analytical-based DSE techniques [6], [7], [8] enable to widely explore the design space of the application. However, some of these models need a description of the targeted architectures and applications within a custom modeling structure. The other methods target a small scope of many-core architectures and can hardly be generalized to heterogeneous architectures. Some tools dedicated to hardware component exploration propose to quickly estimate the achievable performance at a pre-register transfer level [15]. In [16] and [17], the authors use analytic model of accelerator design alongside with a high-level description of the algorithm to quickly sketch the achievable power, performance, and area of the accelerator.

In this paper, a new approach is introduced to reduce the gap between simulation-based DSE methods providing accurate performance estimations and analytical-based DSE techniques that can quickly evaluate performance. The proposed solution targets HMpSoc architectures and relies on an architecture model and a semi-analytic method to quickly extract cost parameters on a real architecture.

III. Overview of Proposed DSE Flow

This section introduces the proposed design flow and briefly describes the content of each design step. The proposed DSE, depicted in Fig. 2, proposes to optimize the HW/SW partitioning and the mapping on the cores under user-defined objectives. An energy objective is especially considered in this approach. The flow targets tiling-based parallel applications and relies on an analytical power model that provides the DSE framework with the execution time and energy of a HW/SW configuration.

A. Tiling-Based Parallel Applications

Tiling is a well-known parallelization technique in the compiler domain [18]. In general, tiling maps an n-deep loop nest into a $2n$-deep one where the most inner n loops include only a fixed number of iterations. Tiling refers to the partitioning of the iteration space into polyhedral blocks. This transformation was initially introduced to increase data locality of applications. Tiling reduces the volume of data accessed between reuse instances of an element, allowing a reusable element to remain in the cache until the next time it is accessed. When dealing with accelerator, this property is useful as it comes with the use of small scratchpad memory. After preliminary transformations on the iteration space (skewing, loop-merging, etc.), tiling exposes the data parallelism of an application. As a consequence, within a dimension of the iteration space, tiles are independent and could be executed in parallel. In short, the tiling transformation has three main advantages in the case of accelerator design: it increases data-reuse with small scratchpad buffer; it exposes

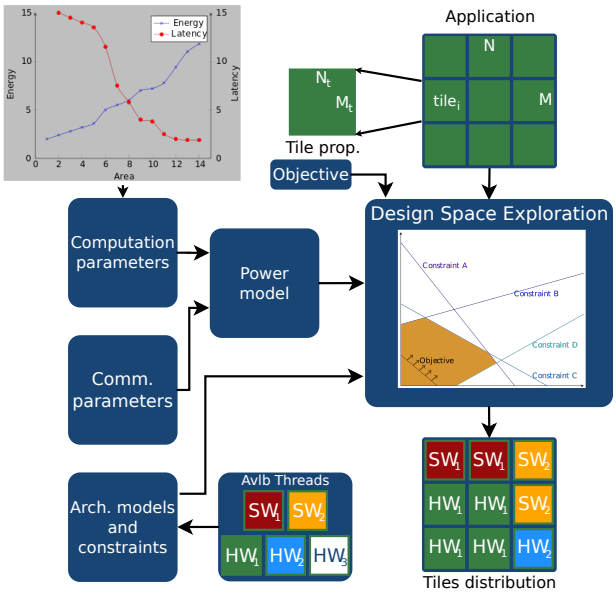


Fig. 2: Overview of the proposed DSE.

coarse-grain parallelism that could be exploited in HW/SW partitioning; within each tile, fine-grain parallelism could be used to draw the full power of the HW architecture. PARSEC benchmarks [19], considered as a representative batch of applications, include 8 applications over the 12 available which are data-parallel. Thus, after small transformations on the iteration space, we can consider that $2/3$ of applications are good candidates for tiling. In the following, we focus on 2D-applications with an N by M iteration space. Tiling is applied with rectangular tiles composed of N_t by M_t elements. Thus, a set of $N_{tiles} = \frac{N}{N_t} \times \frac{M}{M_t}$ tiles are distributed among the execution cores of the target architecture. The tiled application is depicted on top-right of the DSE flow in Fig. 2. Allocation of tiles to HW and SW resources is the output of the DSE flow.

B. Heterogeneous Architectures

In our model, heterogeneous architectures are considered to be composed of N_{SW} software and N_{HW} hardware computation cores. All SW cores could rely on the same processor architecture or could be a set of heterogeneous processor cores. HW cores are implemented in the Programmable Logic (PL) fabric. The PL can be partitioned into at most N_{HW} independent HW cores where N_{HW} is the number of available memory ports within the PL. This restriction, based on the maximum number of simultaneous data transfers to/from the main memory, especially prevents the occurrence of memory congestion that leads to unpredictable memory access-time and energy cost. Moreover, all the resources used by the cores should be less than the overall resources available in the architecture. As an example, the Zynq architecture from Xilinx is a representative case of heterogeneous architectures and is the one that will be used in our experiments. Fig. 3 shows a representation of the memory hierarchy of the Zynq chip. It is composed of two homogeneous SW cores tightly coupled with an FPGA PL fabric. HW cores communicate with SW ones through four dedicated ports (*High Performance Port (HP)*) of the Double-Data-Rate (DDR) memory or using the L2 cache with the *Accelerator Coherency Port (ACP)*. Four other General Purpose (GP) ports are available for FPGA configuration and synchronization, and for communications with standard peripherals. The Zynq architecture will be more detailed in Fig. 5a in the experimental setup section.

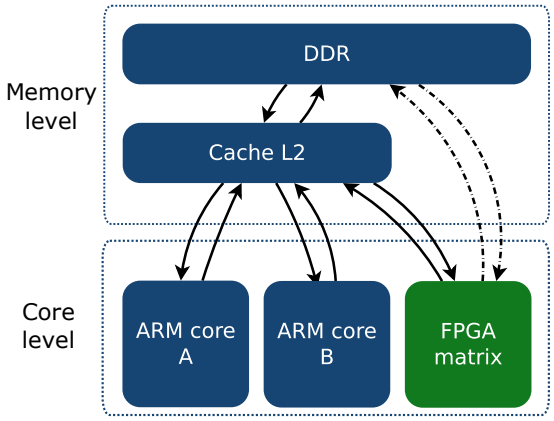


Fig. 3: Memory hierachy of a Zynq architecture.

C. DSE Objectives

The goal of the proposed DSE method is to find the best[1] configuration that minimizes the user objective, which can be either execution time or total energy consumption of the application. A configuration is a vector $\vec{C}$ composed of the tile distribution among the SW and HW cores and of the implementation type used for each of the N_{HW} cores. An example of such distribution is represented in Fig. 2 at the output of the DSE. Let $tiles^i_{SW}$ and $tiles^j_{HW}$ be the number of tiles allocated to the SW core i and to HW core j, respectively. The HW cores could use one computation block within a set of available implementations which have different area-latency-energy trade-offs. As an example, Fig. 2 depicts on its upper left different HW designs, each of them corresponding to an implementation of a computation block with different area, energy, and latency costs. For generality purpose, such implementations could be obtained through hardware exploration tools such as [16]. In our case, hardware exploration was performed manually during the experiments with the help of high-level synthesis. The computation block specification is synthesized under various latency constraints to obtain different implementations varying from 1 to N_{impl}, with $impl_j$ representing the implementation used in the HW core j. Under these assumptions, a configuration $\vec{C}$ is defined as

$$\vec{C} = \Big[tiles^1_{SW}; \ldots; tiles^{N_{SW}}_{SW}; (tiles^1_{HW}, impl_1); \ldots; (tiles^{N_{HW}}_{HW}, impl_{N_{HW}}) \Big], \quad (1)$$

where the tuple $(tiles^j_{HW}, impl_j)$ is the number of tiles allocated to HW core j using a computation block build upon implementation $impl_j$. The aim of the DSE is therefore to find

[1]We use the term "best" as our MILP formulation will guarantee to find the best solution according to the defined model.

the best configuration $\vec{C_{opt}}$ that minimizes the cost (time or energy) objective such as:

$$\vec{C_{opt}} = \min_{\forall \vec{C}} Cost(\vec{C}), \tag{2}$$

where the function $Cost(\bullet)$ relies on the analytical models described in the next section.

D. Energy and execution time models

The execution time and power models are introduced in this section. The energy consumption of an application executed on a heterogeneous multiprocessor architecture depends on three main sources: static energy dissipated during execution time, dynamic energy consumption used for computations, and energy used for communications between processing cores. Applications are composed of N_{tiles} independent tiles that could be executed in parallel. The execution of a tile is atomic and the energy and time needed for its computation only depends on the targeted execution core. The amount of required data for tile computation is assumed to be known at compile time. HW cores are managed with low-level calls and interruptions encapsulated within a SW thread. Spawning the threads among computation cores is performed sequentially and begins with HW cores to minimize the number of context switches within the operating system.

1) Computation time: For a configuration $\vec{C}$, the total computation time is

$$T_t(\vec{C}) = \max\left[T_{HW}(\vec{C}), T_{SW}(\vec{C})\right] \tag{3}$$

where T_{SW} and T_{HW} corresponds to the computation time on the SW and HW cores, respectively, computed as:

$$\begin{aligned} T_{HW}(\vec{C}) &= \max_{j\in(1\ldots N_{HW})}\left[tiles^j_{HW} \times T^{impl_j}_{comp} + j \times T_{spawn}\right] \\ T_{SW}(\vec{C}) &= \max_{i\in(1\ldots N_{SW})}\left[tiles^i_{SW} \times T^{SW_i}_{comp} \right. \\ &\qquad \left. +(N^{used}_{HW}(\vec{C}) + i) \times T_{spawn}\right] \end{aligned} \tag{4}$$

with $T^{impl_j}_{comp}$ and $T^{SW_i}_{comp}$ the computation time of a tile computed on HW implementation j and on SW core i, respectively. T_{spawn} represents the time needed to configure and spawn a computation thread. $N^{used}_{HW}(\vec{C})$ represents the number of HW cores used with configuration $\vec{C}$:

$$N^{used}_{HW}(\vec{C}) = \sum_{j=1}^{N_{HW}} \begin{cases} 1, & \text{if } tiles^j_{HW} > 0 \\ 0, & \text{else} \end{cases}. \tag{5}$$

2) Energy consumption: The total energy $E_t(\vec{C})$ consumed by the execution of the tiled application on configuration $\vec{C}$ comprises the static energy E_{stat}, the energy consumed E_{SW} by SW cores and the energy consumed E_{HW} by the HW cores. Both SW and HW energy consumption values are dynamic energy used for computations and communications. The total energy $E_t(\vec{C})$ is therefore:

$$\begin{aligned} E_t(\vec{C}) &= E_{stat} + \underbrace{\sum_{i=1}^{N_{SW}} (tiles^i_{SW} \times E^{SW_i}_{comp} + E^{SW}_{com})}_{E_{SW}} \\ &\quad + \underbrace{\sum_{j=1}^{N_{HW}} (tiles^j_{HW} \times E^{impl_j}_{comp} + E^{HW}_{com})}_{E_{HW}}, \end{aligned} \tag{6}$$

where E^{SW}_{com} and E^{HW}_{com} are the energy of communications for a tile computed on SW and HW, respectively. These parameters include the energy costs of both the communication channel and the memory access. $E^{SW_i}_{comp}$ and $E^{impl_j}_{comp}$ represent the energy required to compute a tile on SW core i and on HW implementation $impl_j$, respectively.

In the next section, we propose a method to estimate the computational energy of a tile and the energy due to communications between cores.

IV. Estimation of Computation and Communication Parameters

A. Communication Parameter Extraction

In Eq. (6), communication parameters E^{SW}_{com} and E^{HW}_{com} represent the energy required to read and write data for the computation of one tile on a SW core and a HW core, respectively. The extraction of these parameters is performed using the method introduced in [20].

The proposed methodology relies on the use of micro-benchmarks that stress a specific communication path of the architecture. As illustrated by the architecture shown in Fig. 3, a HMpSoC architecture is composed of four kinds of communication channels: SW core to SW memory (L2 cache); SW memory to DDR memory; HW core to SW memory (L2 cache); and HW core to DDR memory. To determine the parameters of this architecture model, a set of micro-benchmarks must be build as follows:

- SwChannel: this micro-benchmark focuses on the communication cost between processing core and SW memory level. It generates read or write accesses in an array allocated in SW memory.
- MemoryChannel: this micro-benchmark focuses on the communication cost between SW memory and DDR memory. Theses parameters can not be measured directly. To this aim, the micro-benchmark generates read or write accesses in an array allocated in the DDR memory from the SW core and then deduces MemoryChannel parameters by subtracting the SwChannel values.
- HwSwChannel: this micro-benchmark focuses on the communication cost between HW accelerators and SW memory level. It generates read or write accesses in an array allocated in SW memory.
- HwChannel: this micro-benchmark focuses on the communication cost between HW accelerators and DDR memory. It generates read or write accesses in an array allocated in DDR memory.

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

When processing cores compute one tile, a solving function can be used to compute the resulting communication cost. Let $\mathcal{C}$ be the set of communication channels used to execute one tile, $\mathcal{C}$ = {SwHwChannel_read, SwHwChannel_write,HwChannel_read, HwChannel_write} for HW core and $\mathcal{C}$ = {SwChannel_read, SwChannel_write, MemoryChannel_read, MemoryChannel_write} for SW core. The communication energy can be expressed as

$$E_{com}^{HW/SW}(N) = \sum_{c \in \mathcal{C}} e_{0_c} N + e_{1_c}, \tag{7}$$

where e_{0_c} and e_{1_c} are energy parameters of the c^{th} crossed channel estimated from the microbenchmarks. Section VII-A will give the extracted energy parameters for the channels used by the target applications.

B. Computation Parameter Extraction

Computation parameters represent the energy and time required to execute a tile for a given configuration (each HW implementation and type of SW core). To extract these parameters, execution traces with time and energy information are needed for each configuration. Characterizing the whole design space would require too many traces and therefore represent a very long time. To reduce this characterization time, we only consider configurations in a subset of the design space defined as the basic architecture with one SW core and one HW core. Therefore, with an application on N_{tiles} tiles, an architecture with N_{impl} HW implementations and N_{core} types of SW processor and assuming that $N_{tiles} >= 0$, $N_{impl} > 0$, and $N_{core} > 0$, the subset size is defined by the following polyhedron:

$$[N_{tiles}, N_{impl}, N_{core}] \rightarrow (1 + N_{tiles}).\max(N_{impl}, N_{core}). \tag{8}$$

In addition, the execution traces are performed on equally distributed samples which are representative points of HW and SW tile distributions. In Section VII-A, we show that only 9 sample configurations are required to achieve good parameter extraction accuracy with an application composed of $N_{tiles} = 256$ tiles. The approach restricting the subset of the design space to the basic architecture, combined with the sampling of the traces, enables to extract the computation parameters with a very small subset of execution traces in the total design space. For instance, if we consider a tiled application with $N_{tiles} = 256$, four available implementations and two kinds of SW processor, the execution traces needed for parameters extraction represent less than 3.6 % of the design space. The execution traces are then processed with a Least Squared Root (LSR) algorithm to extract parameters. LSR computes the parameter values that minimize the squared error between the measurements and the cost function.

The execution time parameters are extracted with the time cost function defined in Eq. (4). For each execution trace, the sample configuration involving implementation $impl_j$ and SW type SW_i is known as well as the overall execution time. The LSR algorithm extracts the values of parameters $T_{comp}^{impl_j}$, $T_{comp}^{SW_i}$ and T_{spawn}. The energy parameters are extracted with the energy cost function introduced in Eq. (6). For an execution trace involving $impl_j$ and SW type SW_i, the overall energy consumption, the communication energy, E_{com}^{SW} and E_{com}^{HW}, and the static energy E_{stat}, are computed. Then, the LSR algorithm extracts the values of parameters $E_{comp}^{impl_j}$ and $E_{comp}^{SW_i}$. The generation of the execution traces is detailed in Section VI and the parameter extractions are given in Section VII-A

V. Design Space Exploration

The second main part of the methodology is the design space exploration. To demonstrate that exploring the design space would lead to large variation in the cost function and to a prohibitive number of solutions, we first describe the case of an exhaustive search. Then, a Mixed Integer Linear Programming (MILP) formulation is proposed to solve the DSE problem and find the best solution.

A. Exhaustive search

Let the target architecture be similar to the Zynq architecture with two SW cores of the same type and four HW cores. The design space is described as a polyhedron and its size can be calculated with the help of the `isl` library [21]. As a result, the design space size is defined with the following polynomial with N_{impl} and N_{tiles} as parameters:

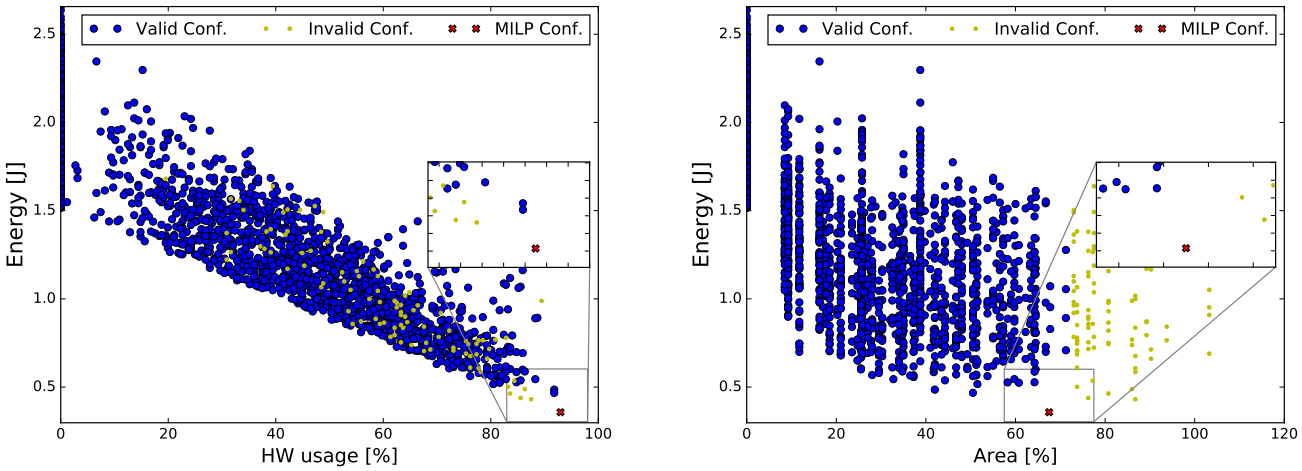
$$\begin{aligned}[N_{impl}, N_{tiles}] \rightarrow \quad & N_{impl}^4 \times (1 && + \tfrac{137}{60} \times N_{tiles} \\ & + \tfrac{15}{8} \times N_{tiles}^2 && + \tfrac{17}{24} \times N_{tiles}^3 \\ & + \tfrac{1}{8} \times N_{tiles}^4 && + \tfrac{1}{120} \times N_{tiles}^5). \end{aligned} \tag{9}$$

For instance with $N_{impl} = 3$ and $N_{tiles} = 256$, the design space is composed of $7,866 \times 10^{11}$ points. If the energy and time consumption can be obtained in about 1ms and using 1kb of memory, the exhaustive DSE would take more than 9 days and a prohibitive amount of memory, which is therefore not a valid approach. Fig. 4a shows a randomly chosen subset of the design space configurations (blue points) for the matrix multiplication kernel. The figure plots the energy consumption as a function of the percentage of used HW equivalent to the proportion of tiles computed on the HW accelerator fabric. Fig. 4b shows, for the same explored configurations, energy consumption as a function of the percentage of resources used in the HW. These plots demonstrate that the design space is very large and the exploration does not guarantee the implementation validity of the obtained configurations. An non-valid configuration in Fig. 4 is a configuration that requires more than the available on-chip resources.

To tackle this issue, the following section introduces an optimization method based on MILP formulation, which solves the problem defined by (2) and finds the best solution in the design space. Fig. 4a and Fig. 4b also plot the solution obtained thanks to MILP optimization (configuration plotted with a red cross) presented in the next section.

B. MILP formulation

Mixed Integer Linear Programming (MILP) is a well-known general framework for capturing partitioning problems. With this approach, constraints are defined as a set



(a) Energy vs percentage of tiles computed on the HW accelerator

(b) Energy vs. percentage of resources used in the PL

Fig. 4: Random subset of the design space configurations for the matrix multiplication kernel (blue points). The configuration plotted with a red cross is the best solution found by the MILP optimization presented in Section V-B.

of inequalities with Boolean, integer (discrete) and non-integer (continuous) variables. Then, solutions can be efficiently determined using commercial or open source solvers. The optimization is defined with a linear objective function. The intersection of the inequality constraints represents a polyhedron of the feasible solution. The objective function defines a direction into the solution space and the optimal solution is found at the intersection between the objective function and the feasible solutions. This section introduces the set of inequalities that captures the constraints of our problem. Then the cost and objective functions are defined.

1) Model constraints:

a) Coverage constraints: To ensure that each tile is computed only once, the coverage constraint is:

$$N_{tiles} = \sum_{i=1}^{N_{SW}} tiles_{SW}^{i} + \sum_{j=1}^{N_{HW}} tiles_{HW}^{j}. \tag{10}$$

b) Unicity constraints: To ensure that only one hardware implementation is used for each HW thread, the variable $usedImpl_{type}^{i}$ is defined as:

$$usedImpl_{type}^{j} = \begin{cases} 1, & \text{if } impl_j = type \\ 0, & \text{else} \end{cases} \tag{11}$$

This variable has binary values and can be multiplied by other variable without introducing unsolvable non-linearity in the model. The unicity constraint, ensuring that only one implementation is used in each HW core, is:

$$\forall j \in (1 \dots N_{HW}) : \sum_{type=1}^{N_{impl}} usedImpl_{type}^{j} \leq 1. \tag{12}$$

c) Resource constraints: The HW part (PL fabric) of the target architecture is constrained by the quantity of available resources. The resources are not shared between the N_{impl} implementations, each HW core uses therefore its own set of resources. These resources are separated into four distinct types: Blocks RAM (BRAM), Arithmetic blocks (DSP), Flip-flops (FF), and Look-Up Tables (LUT). The resource constraint, which guarantees the availability of resources, is defined as:

$$\forall r \in \{\text{BRAM, DSP, FF, LUT}\} : \sum_{j=1}^{N_{HW}} \sum_{type=1}^{N_{impl}} usedImpl_{type}^{j} \times RscCost_{r}^{type} \leq avlbRsc_{r}, \tag{13}$$

where $RscCost_{r}^{type}$ is the cost in resource r of implementation $type$ and $avlbRsc_r$ the available resource of type r in the architecture. These constraints prevent the occurrence of invalid configurations.

2) Cost functions:

a) Time cost function: By including Eq. (11) in the cost function of Eq. (4), two distinct cost functions can be defined. Eqs. (14) and (15) compute the execution time of a HW thread and SW thread, respectively.

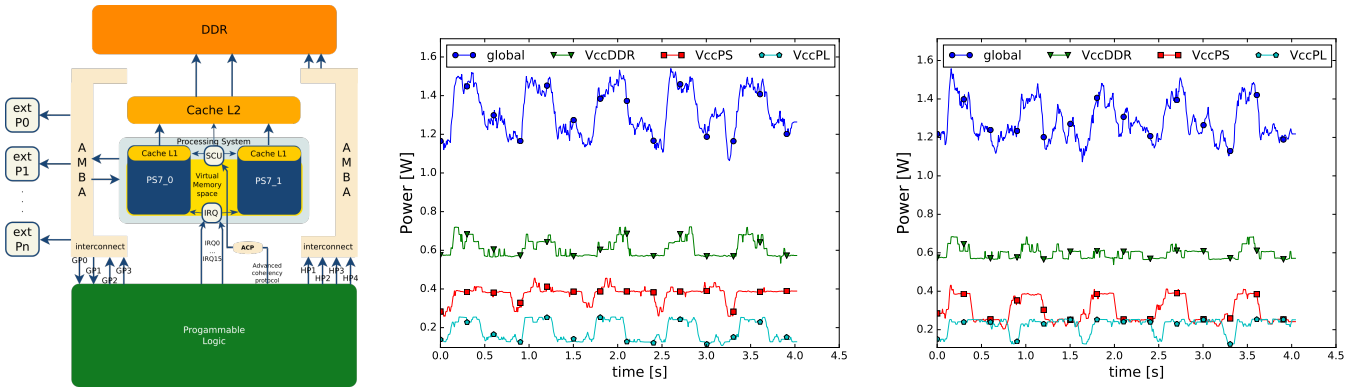
$$\begin{aligned} \forall j \in & \quad (1 \dots N_{HW}) : \\ T_{HW}^{j} & = tiles_{HW}^{j} \times \sum_{type=1}^{N_{impl}} (T_{comp}^{type} \times usedImpl_{type}^{j}) \\ & \quad + j \times T_{spawn} \end{aligned} \tag{14}$$

$$\begin{aligned} \forall i \in & \quad (1 \dots N_{SW}) : \\ T_{SW}^{i} & = tiles_{SW}^{i} \times T_{comp}^{SW_i} + (N_{HW}^{used} + i) \times T_{spawn} \end{aligned} \tag{15}$$

These functions are built upon the computation time parameters, the number of allocated tiles, and the time needed for thread spawning. Finally, the total execution time of the application T_t is estimated by

$$T_t = \max \left[\max_{j \in (1 \dots N_{HW})} (T_{HW}^{j}), \max_{i \in (1 \dots N_{SW})} (T_{SW}^{i}) \right]. \tag{16}$$

b) Static Energy: By denoting P_{base} the static power of the base architecture (architecture with no accelerator built in the PL fabric) and Δp_{stat}^{type} the extra static power introduced



(a) Structure of Xilinx Zynq (b) Matmult 37.5 % performed by HW (c) Matmult 75 % performed by HW

Fig. 5: Experimental infrastructure and power measurement traces.

by the extra resources used by HW implementation $type$ in the PL fabric, the static energy is computed by

$$E_{stat} = T_t \left(P_{base} + \sum_{j=1}^{N_{HW}} \sum_{type=1}^{N_{impl}} \Delta P_{stat}^{type} . usedImpl_{type}^{j} \right). \quad (17)$$

This expression includes non-linear terms, which makes optimization with MILP approach impossible. In order to solve this issue, a variable $T_{HW}^{type,j}$ is introduced. The constraints expressed in Eq. (18) set $T_{HW}^{type,j}$ equal to T_t when implementation $type$ is used on HW core j, and to 0 in the other cases. For this purpose, a large integer K is used to set the constraints to 0 when necessary.

$$\begin{aligned} T_t - T_{HW}^{type,j} &\leq K \times (1 - usedImpl_{type}^{j}), \\ T_{HW}^{type,j} - T_t &\leq K \times (1 - usedImpl_{type}^{j}), \\ T_{HW}^{type,j} &\leq K \times usedImpl_{type}^{j}. \end{aligned} \quad (18)$$

By including $T_{HW}^{type,j}$ in Eq. (17), the static energy can be expressed without non-linear terms as

$$E_{stat} = T_t \times P_{base} + \sum_{j=1}^{N_{HW}} \sum_{type=1}^{N_{impl}} (\Delta P_{stat}^{type} . T_{HW}^{type,j}). \quad (19)$$

c) Energy cost function: Finally, the total energy cost E_t from (6) can be computed as

$$\begin{aligned} E_t \quad &= E_{stat} \\ &+ \sum_{i=1}^{N_{SW}} (tiles_{SW}^{i} \times E_{comp}^{SW_i} + E_{com}^{SW}) \\ &+ \sum_{j=1}^{N_{HW}} (tiles_{HW}^{j} \times \sum_{type=1}^{N_{impl}} E_{comp}^{type} \times usedImpl_{type}^{j} \\ &\quad + E_{com}^{HW}). \end{aligned} \quad (20)$$

3) Objective functions: The solver first builds the configuration vector $\vec{C}$ using the constraints and then selects the best solution using the cost functions according to the optimization objective. Two objectives are defined: minimizing the overall execution energy or minimizing the overall execution time. These objective functions are defined as:

$$\vec{C}_{energy} = \min_{\forall \vec{C}} \left[E_t(\vec{C}) \right] \quad (21)$$

$$\vec{C}_{time} = \min_{\forall \vec{C}} \left[T_t(\vec{C}) \right] \quad (22)$$

The output of the MILP optimization is the best configuration vector to achieve the objective and an estimation of its characteristics (energy consumption and execution time). Fig. 4 plots the solution obtained with the MILP-based method for the matrix multiplication kernel (configuration with a red cross). It can be noticed that this solution is always cost-optimal, computed in a limited amount of time and not covered through brute-force exploration. These results also show that the gain in cost for the optimal configuration is significant.

VI. Experimental Setup

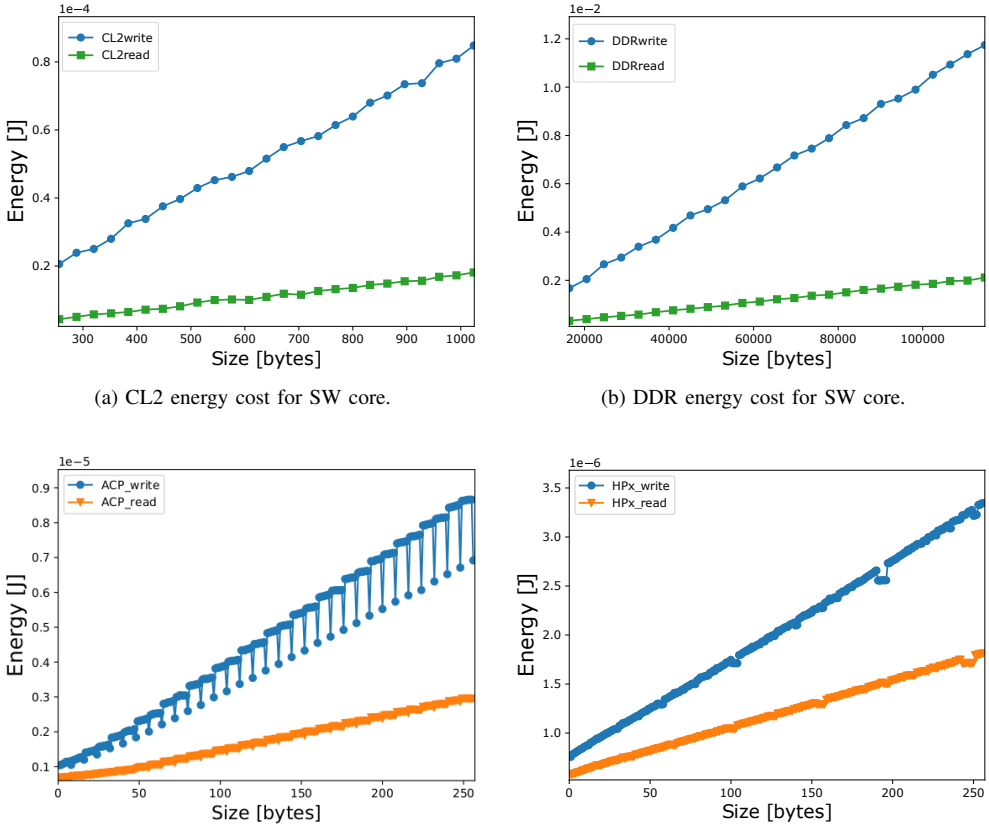
This section describes the two application kernels used, the target architecture, and the power measurement setup. Then the HW implementations used in each test case are described.

A. Application kernels

The first test case is a matrix multiplication (`Matmult`) application. Input and output matrices are of size 512×512. Matrix computation is subdivided in 256 tiles of size 32×32. Each tile reads full rows or columns in the input matrices and write tile results in the output one. The second test case is a `Stencil` computation based on a filter of size 4×4, which is successively applied 10 times. The input and output matrices are of size 542×542 and 512×512, respectively. The application is tiled with overlapping approach to prevent inter-tile dependency, which results in 256 independent tasks to be computed.

B. Measurement infrastructure

Experiments are based on the Zynq architecture from Xilinx [22]. Fig. 5a shows a simplified version of the internal structure of the Zynq, which is composed of two ARM cortex A9 processors tightly coupled with an FPGA fabric. The Zynq Zc702 evaluation board is used with the ARM cores running a Linux operating system *(Linux/arm 4.0.0)*. Power consumption is measured through the *Power Management Bus (PMBus)* embedded on the board. In our experiments, a HW component samples at regular time intervals 7 input power rails of the Zynq SoC. A new sample could be retrieved every 1 ms with 16 bits dedicated for current value and 16 bits dedicated to voltage. A sample frequency of 142 Hz could be achieved, when the 7 rails are monitored at the same time. *PMBus* values can be accessed through two methods: using



(a) CL2 energy cost for SW core.

(b) DDR energy cost for SW core.

(c) CL2 energy cost for HW core.

(d) DDR energy cost for HW core.

Fig. 6: Average energy consumed on each communication channel as a function of the communication size N in Bytes.

ARM peripheral with Linux interruption, or using full HW mechanisms implemented in the PL. In our experiments, the second approach is used since the HW power consumption is more predictable than the SW one.

These measures are used to compute the instantaneous power and the energy variation over the application execution. Fig. 5 displays the power curves obtained for two test examples of the `Matmult` application with different HW usage. The curves show the three more representative power rails: *VccPS* powers the dual cortex A9 processors, *VccPL* the computational core in the PL fabric, and *VccDDR* the external DDR memory. The last curve *(global)* displays the total power consumption over the 7 rails. In the two plots, the computation kernel of the `Matmult` application is executed 5 times with two distinct workload balances between SW and HW resources (37.5 % done by the HW in Fig. 5b and 75 % in Fig. 5c). These curves show that the timing resolution of the power measurement enables to precisely display each computation step involved in the test. In Fig. 5b, the HW computation finishes before the SW one for each execution. The SW computation is therefore the bottleneck of this configuration. On the other hand, the HW part is the computation bottleneck in Fig. 5c.

C. Hardware implementations

Hardware exploration was performed to obtain several implementations with different characteristics in terms of resources used and computation latency [23], [24]. As shown in [25], tiling-based kernels are suitable to be optimized using High-Level Synthesis (HLS) tools. Hardware implementations were thus generated using Vivado HLS tool *(2015.4 HLX Edition)* [26]. Only representative implementations were selected among all the HW design space.

The HW implementations use BRAM scratchpad to maximize the sequential access in memory and make profit of the DMA and burst features. The implementation are done in a manner to maximize the overlapping between the communications and the computations. Table I gives the latency and the hardware resources for each implementation of the `Matmult` and `Stencil` tiled computation. All these designs are pipelined, but the parallelism degree of the inner loop-nest is different. The implementation name in the table expresses the parallelism degree of the three inner loop-nest. LnP stands for *Loop-nest Parallelism*. For instance, LnP 248 means that 2 iterations of the third inner loop was computed in parallel, 4 iterations for the second loop, and 8 for the first loop. To have a fair comparison between the HW implementations, the loop-nest flattening has been stopped when the use of *DSP* block resources explodes without a real incidence on latency.

D. Software implementations

The software implementation is written in order to expose instruction parallelism and to maximize the cache reuse. Compiled with gcc and the *-Ofast* flag, the obtained binary code uses the dedicated NEON instructions to draw the full power of the ARM architecture. The obtained SW takes approximately 8.5×10^{-3} s to compute one tile for the mamult application and 2×10^{-3} s for the stencil application on one ARM core running at 800 MHz.

VII. Exploration Results

This section presents the parameter extraction results as well as the MILP optimization results and accuracy.

A. Parameter extraction

1) Communication parameters: Introduced in [20], a set of micro-benchmarks has been designed to extract the communication parameters of the Zynq platform. For the two application kernels, both SW and HW cores access to data located in both the L2 cache and the DDR memory. The SW cores access to both L2 cache and DDR memories using standard memory access, whereas HW cores access to L2 cache memory using ACP port and to DDR memory using the four HP ports.

In order to extract the communication cost of these channels, a set of four micro-benchmarks is used:

- The CL2 micro-benchmark generates read or write operations from SW core in an array located in the L2 cache to extract the communication cost of the channel between SW cores and the second memory level.
- DDR generates read or write operations from SW core in an array located in the DDR, for the channel between SW cores and the external memory.
- ACP generates read and write operations from HW core in an array located in the L2 cache, for the channel between HW core and the second memory level.
- HPx generates read and write operations from HW core in an array located in the DDR, for the channel between HW core and the external memory.

Fig. 6 shows the average energy consumption for access (read and write) to both L2 cache and DDR memory banks. The results are given for different data size (in bytes).

Apps.	Impl.	Lat. [Mcycle]	Resources [%]			
			BRAM	DSP	FF	LUT
Matmult	LnP 114	140	25%	2%	2%	5%
	LnP 118	73	25%	4%	2%	6%
	LnP 128	40	25%	9%	3%	10%
	LnP 148	23	25%	18%	6%	16%
	LnP 248	14	26%	36%	11%	30%
	LnP 448	12	30%	59%	19%	47%
Stencil	LnP 114	12	24%	39%	10%	42%
	LnP 244	7	21%	75%	20%	72%
	LnP 384	6	24%	100%	28%	96%

TABLE I: Execution latency in millions of cycle and percentage of resources used for various HW implementations of the two considered kernels. Configuration LnP xyz refers to a given Loop-nest Parallelism.

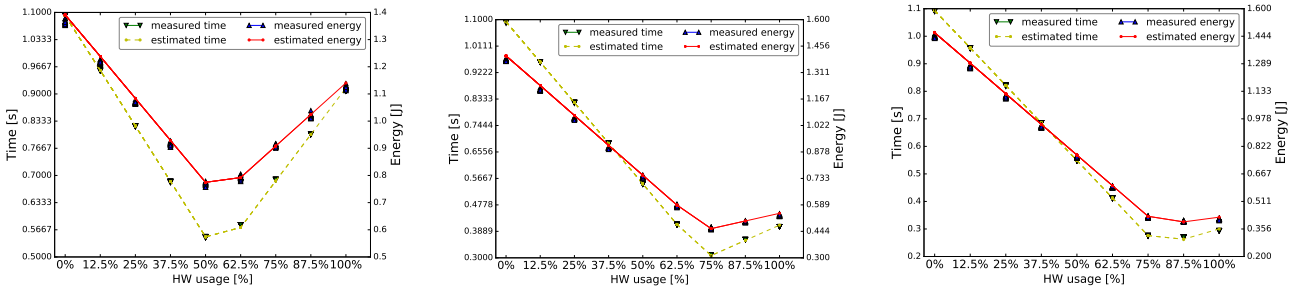
Benchmark	Energy [J] $f : N \rightarrow e_{0_c} N + e_{1_c}$	
	e_{0_c}	e_{1_c}
CL2 write	1.70e-08	-1.29e-09
CL2 read	1.51e-09	-2.30e-09
DDR read	1.54e-09	6.16e-07
DDR write	2.37e-08	-1.04e-06
ACP read	9.97e-11	5.30e-09
ACP write	2.78e-10	6.95e-09
HPx read	5.56e-11	6.49e-09
HPx write	1.18e-10	9.37e-09
Static Power	1.20e+00	no value

TABLE II: Extracted energy model parameters for the Zynq architecture.

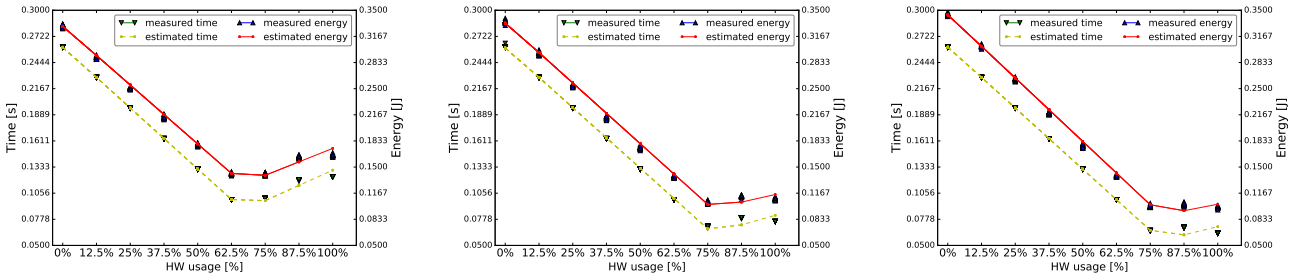
The values are extracted after 20 executions of the micro-benchmarks. Results show that the variance of the measurements is quite small. The energy consumption cost of communication can be approximated as a linear function $f(N) = e_{0_c} \times N + e_{1_c}$, where the values e_{0_c} and e_{1_c} are respectively the dynamic and static parts. Table II gives these energy cost parameters (e_{0_c} and e_{1_c}) extracted on the Zynq architecture with the micro-benchmark set detailed before. The validity domains of these functions are [128 bytes, 1 Kbytes] for the L2 cache memory channel and [4 Kbytes, 128 Kbytes] for the DDR memory channel.

2) Computation parameters: The set of sample configurations was launched 20 times for each implementation. The execution power measurement traces were then parsed and processed to obtain a set of test points composed of the sample configuration, the average energy consumption and the execution time. First, an estimation of the communication cost is computed for each sample configuration. Then, the cost functions described in Eqs. (4) and (6) are used with an LSR algorithm to extract time cost parameters and energy cost parameters. These functions will be used to explore the design space.

Measured values are compared to the estimated one computed using the previous extracted parameters. The extraction and estimation were performed for each HW implementation of the `Matmult` and `Stencil` applications. The measurements are fully automatized and take 336 s for the `Matmult` and 122 s for the `Stencil`. This measurement time is really acceptable since the parameter extraction is executed only once. Fig. 7 shows, for each implementation, the measured and estimated values for energy consumption and execution time for different HW/SW tiles distribution. These results show that the error between measured and estimated values is low, showing the good accuracy of the extracted parameters. SW cost parameters (i.e. HW usage = 0%) are independent of the associated HW implementation. Moreover, another accuracy indicator of the proposed extraction method is that the error on SW parameters has a standard deviation of 7.2×10^{-6} for `Matmult` application and of 1.2×10^{-6} for `Stencil`. In addition, these curves show that the HW/SW tile distribution that minimizes the energy consumption depends on the used implementation.



(a) Matmult implementation LnP 118 (b) Matmult implementation LnP 148 (c) Matmult implementation LnP 448



(d) Stencil implementation LnP 114 (e) Stencil implementation LnP 244 (f) Stencil implementation LnP 384

Fig. 7: Energy consumption and execution time versus HW usage: measured and estimated values.

Conf.		Thread ID					
		SW0	SW1	HW0	HW1	HW2	HW3
Mat.	Impl.	SW	SW	LnP 248	LnP 248	LnP 148	none
	N_{tiles}	9	9	87	85	66	0
Stenc.	Impl.	SW	SW	LnP 114	LnP 114	none	none
	N_{tiles}	25	24	104	103	0	0

TABLE III: Configuration obtained after MILP optimization: best implementations and repartition of the tiles on the SW and HW cores.

B. MILP Optimization

In this section, the application parameters and the previously extracted cost parameters feed the MILP model and the optimization problem is solved based on the energy cost function of Eq. (21) using *Gurobi* MILP solver [27] on an *Intel i7 Haswell-ult* processor running at 2.10 GHz. The MILP solution is obtained after 18980 simplex iterations in $\simeq 0.73$ s for the `Matmult` application kernel and after 348 simplex iterations in $\simeq 0.03$ s for `Stencil`. This difference is mainly due to the higher number of HW implementations available for `Matmult`, which enlarges the design space size. However, even for quite complex kernels and such large design space, these results demonstrate the scalability of the MILP approach.

Table III shows the MILP result configuration (tile distribution across the cores and implementations used) for the two considered kernels obtained after MILP optimization. For `Matmult`, 18 tiles are equally distributed on the SW core, the remaining tiles are distributed on three HW cores. Implementation *LnP 248* is used on two HW cores, the third one uses a smaller implementation to fulfill the remaining space in the FPGA PL fabric. In this case, the number of HW cores used is limited by the available FPGA resources and not by the upper-bound due to the memory bandwidth. As the `Stencil` implementations use more resources than for `Matmult`, only two HW cores are used.

C. Precision and gain factors

In this section, the performance of the MILP configuration is compared with the best sample configuration among the 1028 implementations obtained during computation of the parameters, and with two basic configurations: Full SW and Full HW. The best sample configuration is obtained in around 1 s for both application kernels, in the same order-of-magnitude as for MILP configuration.

Table IV shows the resources used by each configuration, the estimated execution time and estimated energy consumption for the two applications. Since the MILP configuration can use more than one HW core, the total amount of resources used could be greater than that of the sample configuration. This is the case with the `Matmult` application. On the other hand, the MILP configuration for `Stencil` uses two smaller implementations than the one used in the sample configuration, which results in more BRAM used but less DSP Blocks.

The acceleration gain (speedup) and energy reduction are the ratio between a given configuration and the most energy

	Conf.	Resources [%]				Time [s]	Speedup	Energy [J]	Energy reduction
		BRAM	DSP	FF	LUT				
Matmult	Full SW	0%	0%	0%	0%	1.21	-307.7%	1.59	-199.4%
	Full HW	30%	59%	19%	47%	4.1×10^{-1}	-5.1%	5.6×10^{-1}	-3.7%
	Sample Conf.	30%	59%	19%	47%	3.9×10^{-1}	0%	5.4×10^{-1}	0%
	MILP	76%	90%	28%	76%	2.290×10^{-1}	**41.3%**	3.558×10^{-1}	**34.1%**
Stencil	Full SW	0%	0%	0%	0%	3.2×10^{-1}	-150%	4.2×10^{-1}	-135.9%
	Full HW	24%	100%	28%	96%	1.28×10^{-1}	0%	1.78×10^{-1}	0%
	Sample Conf.	24%	100%	28%	96%	1.28×10^{-1}	0%	1.78×10^{-1}	0%
	MILP	48%	78%	20%	84%	1.121×10^{-1}	**12.4%**	1.566×10^{-1}	**12%**

TABLE IV: Comparison of resources used, estimated execution time and estimated energy consumption on the two applications for four configurations: Full SW, Full HW, best sample, and MILP solution. Speedup and energy reduction w.r.t. the best sample is also provided.

Application		Time [s]	err. [%]	Energy [J]	err. [%]
Mat.	Est.	2.29×10^{-1}	3.35%	3.558×10^{-1}	5.03%
	Meas.	2.369×10^{-1}		3.746×10^{-1}	
Sten.	Est.	1.121×10^{-1}	9.58%	1.566×10^{-1}	10.25%
	Meas.	1.24×10^{-1}		1.745×10^{-1}	

TABLE V: Comparison of estimated and measured execution time and energy of the MILP solution for the two applications.

efficient sample configuration taken as the reference. It is noteworthy that the MILP optimization leads to an energy saving of 34.1 % for `Matmult` and 12 % for `Stencil` and to a speedup of 41.3 % for `Matmult` and 12.4 % for `Stencil`. The impact of static energy on the total energy is not negligible, therefore, energy reduction is clearly linked with the speedup factor.

To validate the proposed approach, estimated time and energy of the MILP configuration are compared with the real measurements of the same configuration over the Zynq board. The measured values are obtained over 40 independent executions of the MILP configuration. Table V shows both estimated and measured values for the two applications. The average error for `Matmult` is around 5 %, which is quite low for power measurement on real architecture. For `Stencil`, the average error is higher, which is due to the execution time difference. Indeed, as the matrix multiplication execution time is longer than for the stencil kernel, power measurement inaccuracy is less mitigated in the case of `Stencil`.

VIII. Conclusion

This paper introduced a new exploration method for building energy-efficient accelerators on heterogeneous architecture. The method targets computation kernel parallelized using tiling and is based on the measurements of a tiny subset of the design space. These measurements are then injected into two parameter extraction functions to obtain analytical formulations of the execution time and energy consumption of the computation kernel. Then, the tile distribution constraints are captured using an MILP formulation and a solver is used to obtain the best solution. This methodology was tested on two application kernels: a matrix multiplication and a stencil computation on a Zynq-based heterogeneous architecture. These experiments show that the speedup and energy saving obtained are between 12 % and more than 30 % compared to the best sample configuration, respectively. Furthermore, the estimation accuracy is within 5 % to 10 %, which is quite acceptable for real hardware measurements. These results open new opportunity for future computer-aided design tools. Such method could be included in a complete framework (e.g. [28]) with a multi-step exploration to accelerate every computation kernel within an application and to obtain an energy-efficient mapping of a full application on heterogeneous multiprocessor 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] A. D. Pimentel, C. Erbas, and S. Polstra, "A systematic approach to exploring embedded system architectures at multiple abstraction levels," *IEEE Transactions on Computers*, Feb. 2006.

[3] R. B. Atitallah, E. Senn, D. Chillet, M. Lanoe, and D. Blouin, "An efficient framework for power-aware design of heterogeneous mpsoc," *IEEE Transactions on Industrial Informatics*, Feb. 2013.

[4] S. K. Rethinagiri, O. Palomar, A. Cristal, O. Unsal, and M. M. Swift, "DESSERT: DESign Space ExploRation Tool based on power and energy at System-Level," in *27th IEEE International System-on-Chip Conference (SOCC)*, Sept 2014, pp. 48–53.

[5] J. Cong, Z. Fang, M. Gill, and G. Reinman, "Parade: A cycle-accurate full-system simulation platform for accelerator-rich architectural design and exploration," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, Nov 2015, pp. 380–387.

[6] S. Goossens, B. Akesson, M. Koedam, A. B. Nejad, A. Nelson, and K. Goossens, "The compsoc design flow for virtual execution platforms," in *10th FPGAworld Conference*, Sep. 2013.

[7] M. Ammar, M. Baklouti, M. Pelcat, K. Desnos, and M. Abid, "On exploiting energy-aware scheduling algorithms for mde-based design space exploration of mp2soc," in *24th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP)*, Feb. 2016.

[8] A. Khecharem, C. Gomez, J. Deantoni, F. Mallet, and R. D. Simone, "Execution of heterogeneous models for thermal analysis with a multi-view approach," in *Proceedings of the 2014 Forum on Specification and Design Languages (FDL)*, Oct. 2014.

[9] G. Zhong, A. Prakash, Y. Liang, T. Mitra, and S. Niar, "Lin-Analyzer: A high-level performance analysis tool for FPGA-based accelerators," in *ACM/IEEE Design Automation Conference (DAC)*, June, pp. 1–6.

[10] D. Wijesundera, A. Prakash, T. Srikanthan, and A. Ihalage, "Framework for Rapid Performance Estimation of Embedded Soft Core Processors," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 11, no. 2, Jul. 2018.

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

[12] W. Ye, N. Vijaykrishnan, M. Kandemir, and M. J. Irwin, "The design and use of simplepower: a cycle-accurate energy estimation tool," in *37th Design Automation Conference (DAC)*, June 2000, pp. 340–345.

[13] 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.

[14] 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.

[15] L. Wang and K. Skadron, "Lumos+: Rapid, pre-rtl design space exploration on accelerator-rich heterogeneous architectures with reconfigurable logic," in *IEEE 34th International Conference on Computer Design (ICCD)*, Oct. 2016.

[16] Y. S. Shao, B. Reagen, G.-Y. Wei, and D. Brooks, "The Aladdin approach to accelerator design and modeling," *IEEE Micro*, vol. 35, no. 3, pp. 58–70, May 2015.

[17] L. Wang and K. Skadron, "Implications of the power wall: Dim cores and reconfigurable logic," *IEEE Micro*, Sep. 2013.

[18] M. E. Wolf and M. S. Lam, "A Loop Transformation Theory and an Algorithm to Maximize Parallelism," *IEEE Trans. Parallel Distrib. Syst.*, pp. 452–471, Oct. 1991.

[19] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The PARSEC Benchmark Suite: Characterization and Architectural Implications," in *International Conference on Parallel Architectures and Compilation Techniques (PACT)*, Oct. 2008.

[20] B. Roux, M. Gautier, O. Sentieys, and S. Derrien, "Communication-based power modelling for heterogeneous multiprocessor architectures," in *IEEE 10th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (mcSoC)*, Sept 2016, pp. 209–216.

[21] S. Verdoolaege, *isl: An Integer Set Library for the Polyhedral Model*. Springer Berlin Heidelberg, 2010, pp. 299–302.

[22] 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.

[23] V. Bhatnagar, G. Ouedraogo, M. Gautier, A. Carer, and O. Sentieys, "An FPGA Software Defined Radio Platform with a High-Level Synthesis Design Flow," in *IEEE Vehicular Technology Conference (VTC-Spring 13)*, Dresden, Germany, June 2013, pp. 1–5.

[24] M. Gautier, G. Ouedraogo, and O. Sentieys, "Design Space Exploration in an FPGA-Based Software Defined Radio," in *Euromicro Conference on Digital System Design (DSD 14)*, Verona, Italy, August 2014, pp. 22–27.

[25] G. Zhong, A. Prakash, S. Wang, Y. Liang, T. Mitra, and S. Niar, "Design Space exploration of FPGA-based accelerators with multi-level parallelism," in *IEEE/ACM Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2017, pp. 1141–1146.

[26] Xilinx, "Vivado Design Suite - HLX edition," 2015. [Online]. Available: http://www.xilinx.com/products/design-tools/vivado.html

[27] G. Optimization, "Gurobi optimizer reference manual," 2016. [Online]. Available: http://www.gurobi.com

[28] A. Floch *et al.*, "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.