Energy efficient mapping on manycore with dynamic and partial reconfiguration: Application to a smart camera

Robin Bonamy, Sebastien Bilavarn, Fabrice Muller, François Duhem, Simon Heywood, Philippe Millet, Fabrice Lemonnier

Energy Efficient Mapping on Manycore with Dynamic and Partial Reconfiguration: Application to a Smart Camera

ROBIN BONAMY[1], SÉBASTIEN BILAVARN[1], FABRICE MULLER[1], FRANÇOIS DUHEM[2], SIMON HEYWOOD[2], PHILIPPE MILLET[2], AND FABRICE LEMONNIER[2]

[1] *LEAT, CNRS UMR7248, University of Nice Sophia-Antipolis, France*
[2] *High-Performance Computing Lab, Thales Research & Technology, Palaiseau, France*

This paper describes a methodology to improve the energy efficiency of high-performance multiprocessor architectures with Dynamic and Partial Reconfiguration (DPR), based on a thorough application study in the field of smart camera technology. FPGAs are increasingly being used in cameras owing to their suitability for real-time image processing with intensive, high-performance tasks, and to the recent advances in dynamic reconfiguration that further improve energy efficiency. The approach used to best exploit DPR is based on the better coupling of two decisive elements in the problem of heterogeneous deployment: design space exploration and advanced scheduling. We show how a tight integration of exploration, energy-aware scheduling, common power models, and decision support in heterogeneous DPR multiprocessor SoC mapping can be used to improve the energy efficiency of hardware acceleration. Applying this to a mobile vehicle license plate tracking and recognition service results in up to a 19-fold improvement in energy efficiency compared with software multiprocessor execution (in terms of energy–delay product), and up to more than a 3-fold improvement compared with a multiprocessor with static hardware acceleration (i.e. without DPR).

INTRODUCTION

The use of heterogeneous multiprocessor system-on-a-chip devices (SoCs) has grown because of their potential to address energy efficiency, power and heat density problems as we reach the limits of CMOS technology scaling. Although promising, the increasing level of computational power, heterogeneity and energy efficiency requirements in new systems greatly affects their design and deployment complexity. Indeed, it is no longer simply a question of efficiently mapping concurrent processes to the right cores in typical multi/many-core platforms, but also taking into account critical dynamic aspects such as power management (DVFS, clock gating, power gating) and hardware acceleration (eFPGA, DPR, GPU) to deliver the best processing efficiency at each instant.

Hardware acceleration is a relatively well-known player in the energy performance equation which is regaining attention with the advent of Dynamic and Partial Reconfiguration (DPR). Partial reconfiguration is a technique related to FPGAs that can be used to extend their inherent flexibility: it allows specific FPGA regions to be reprogrammed with new functionalities while other regions continue running. A significant reduction in hardware resource utilization results in less static power, and thus better energy efficiency, adding to the inherent benefits of dedicated hardware (i.e. statically reconfigurable accelerators). However, a variety of parameters such as FPGA region partitioning, scheduling, accelerator parallelism and multiprocessor execution oportunities strongly affect the actual processing efficiency, and combine with other techniques such as blanking or DPR-based clock gating that can also be used to further decrease power. As a result, the

quantity and scope of decisions in a dynamically reconfigurable multi/many-core system greatly complexifies the scheduler's job. It is, however, critical to provide good support at this level, since bad decisions can affect energy efficiency to the point of total ineffectiveness. In this paper, we describe a methodology better able to address these considerations by further integrating the definition of advanced energy-aware scheduling and exploration or application mapping analysis. This makes for a good combination of the improvements specific to hardware acceleration and DPR, which are investigated in depth through their real-life application to a mobile vehicle license plate tracking and recognition service.

The outline of the paper is as follows. We first review existing works in the field of heterogeneous multi-core scheduling, pointing out relative novelty in the use of DPR for that purpose. In section 3, we introduce the proposed methodology and its underlying power models. We then consider a dedicated power minimization policy defined for heterogeneous deployment and scheduling puposes in section 4. Detailed results of a license plate recognition application on Xilinx devices are analyzed and discussed in section 5. Finally, we present our principal conclusions from the detailed case study and future directions for the research and its applications.

RELATED WORK

Energy efficiency in heterogeneous multiprocessors

With the rise of multicore heterogeneous architectures, there have been many works addressing the efficient scheduling of application workloads for multiple cores, where the cores can be of different types. Early investigations focused primarily on improving load balancing to achieve better performance (throughput, instructions per cycle, etc.). Recently, the challenge of heterogeneous thread scheduling and global power management has hinged on delivering higher power-performance levels (performance per watt). Many works explored the energy efficiency benefits of heterogeneity; these are extensively discussed for instance in [19]. There is a broad consensus that exploiting heterogenity is essential for better use of energy, the inevitable conterpart being that it greatly complexifies the job of the scheduler.

Of the work adressing this problem, [14] proposed a scheduler for a system of processors based on execution prediction to map future processing needs to the most suitable processor. Their method applies to single-ISA heterogenity supporting differing voltages and frequencies. [10] extended a symbiotic scheduling heuristic [27], originally developed to enhance throughput and lower response time, for chip multiprocessors with simultaneous multithreading cores. They report up to 7.4% savings in energy, 10.3% savings in energy–delay product and 35% savings in power. [30] is another contribution that considers both scheduling and power management to address process variations in CMPs. They conducted a design exploration, proposed a number of schedulers to satisfy different objectives, and developed a linear programming solution for power management. The authors in [34] examined the scalability problem for manycores, comparing basic scheduling heuristics and proposing scheduling and power management algorithms for heterogeneous systems scaling up to 256 cores. Recent studies like [16] started to address the ARM big.LITTLE architecture that combines an energy-efficient processor cluster (Cortex-A7) with a higher performance processor (Cortex-A15). Specifically, this case study reports that different classes of resource allocation heuristics (race-to-idle vs. never-idle) have very different results on different platforms, indicating that the efficiency of a strategy greatly depends on platform characteristics and can go as far as reducing efficiency in some cases.

Aside from the question of core specialization, heterogeneity also extends to the aspects arising at run-time due for example to power management [7]. The dynamic use of different power states (P-states, sleep states) available for each core matches the problem of low-power scheduling which had a long history of research over the last 20 years. Surveys have been described for example in [3] and [26] from the abundant literature. Our own previous experimental studies in this field [4] came to similar conclusions as [16] and [17] concerning the importance of platform characteristics and application knowledge on the efficiency of a strategy. Likewise, results indicate that custom strategies, i.e. more specialised schedulers dedicated to an application or application domain can reach significant levels of energy gains (in the 5% to 50% range for video processing applications on representative platforms) compared to existing OS and platform-based strategies.

In this large body of literature, existing works often consider a specific perspective on the heterogeneous scheduling problem: a type of architecture, one precise objective (manycore scaling, process variability),

limited heterogeneity (similar cores, DVFS), etc. In addition, a type of heterogeneity remains relatively unexplored regarding the recent advancements in graphics processing units and reconfigurable processing units: the possibility of running tasks in hardware (or using hardware accleration). [19] thus envision adding heterogeneity to hardware acceleration by taking advantage of progress in Dynamic and Partial Reconfiguration (DPR) and High-Level Synthesis (HLS). It is however worth noting that Intel is considering, in addition to mixing large and small cores, the use of accelerators, including FPGAs, for heterogeneous architecture research [8]. The following introduces relevant works in this field with an emphasis on their possible use in heterogeneous multicore systems.

Hardware acceleration

Hardware acceleration delivers several times better performance and energy efficiency compared to software execution, and the flexibility introduced with DPR can improve these benefits. The underlying heterogeneous mapping and scheduling problems have started lately to be re-investigated with the possibility of dynamic reconfiguration of hardware tasks adding new dimensions to heterogeneity, resource management, scheduling and energy efficiency issues.

Early works like [2] and [15] addressed run-time management for dynamically reconfigurable systems which were mainly motivated by the search for the best fit/speed-up trade-off. [21] is a first significant work addressing DPR from a practical energy efficiency perspective. They show that using DPR to eliminate the power consumption of the accelerator when it is inactive (blanking) can outperfom clock gating and reduce the energy consumption by half. But the application study on 64-bit division accelerator makes it difficult to extrapolate conclusions to a more complex application level. Notably it does not address DPR ability to share and reuse reconfigurable regions (RR) to further reduce static power, nor does it address whether or not DPR improves energy compared to static acceleration.

Energy models of DPR have been previously investigated in depth in [5] with a real life application study on a H.264/AVC video profile decoder. Three functions are accelerated using HLS and an analysis is conducted to identify a DPR scheduling solution and compute the associated energy cost. Results reported 22.5% energy gains for dynamic over static execution. Both previous works and others pointed out the importance of configuration speed as an essential condition for energy efficiency, but various opportunites lies in dynamic reconfiguration.

Aside from blanking which reduces energy consumption by decreasing the share of static power associated with reconfigurable regions [31], some low-level techniques investigated the use of dynamic reconfiguration to reduce clock-related losses. A low-overhead clock gating implementation based on dynamic reconfiguration has been proposed in [29], achieving a 30% power reduction compared to standard FPGA clock-gating techniques based on LUTs. Another approach has been developed to modify the parameters of clock tree routing during run-time reconfiguration to moderate clock propagation across the whole FPGA and decrease dynamic power [32]. Finally, self-reconfiguration also allows online modification of clock frequency with low resource overhead by acting directly on clock management units from the reconfiguration controller [24].

Although such DPR-based features are fully effective with the potential to improve the already significant efficiency inherent to hardware processing, there are significant practical problems that remain to be addressed. The most important of them is the need for integrated methodologies able to fully explore their benefits before final implementation. For a long time (and even now) the same reason has prevented the widespread adoption of reconfigurable technologies, as too much hardware expertise is needed to quickly produce fully working accelerators from software programming languages such as C and C++. Indeed, two main directions of research stand out in the light of recent works: methodologies and programming models.

There has been a lot of advances since the emergence of HLS in recent years. SDSoC from Xilinx [18] is among the most advanced heterogeneous development tools in this regard. It helps in particular to compensate for the lack of automated system connectivity generation in HLS, the design of which is still critical and very time consuming. However, there are other dimensions to explore at the system level, as far as DPR and power are concerned, that are still not being investigated enough despite a variety of design methodologies available in the literature. ReConOs [1], for example, is one of these representative efforts to develop approaches

targeting heterogeneous CPU/FPGA systems, with a unified programming model, and execution environment for threads running in software and reconfigurable hardware. Among more recent works, PAAS [20] is another design approach for CPU with ASIC- or FPGA-based accelerators in which the focus is on memory hierarchy. Other works like PolyPC [11] addressed an OpenCL-based framework to implement a custom hardware platform using HLS. Despite all these necessary contributions, it is worth noting that (i) all of them examine aspects of the problem from a performance perspective; (ii) very few address the definition of a consistent DPR methodology, although of course some works like [25] do consider DPR, but rather in the scope of a particular application study; and (iii) few works investigate power and energy efficiency as an explicit primary focus.

There is currently a profusion of research investigating the hardware/software code compilation problem, from the programming model perspective to the extraction of abstract parallelism, on-chip communications and platform integration. Indeed a variety of works addresses specifically the use of GPU-based (OpenCL [33][18], CUDA [22]), or domain-specifc languages such as HIPAcc [23] or previously RVC-CAL [35] for video and image processing. Aside from the fact that they do not truly address DPR, the primary objective in these works is to automate the generation of statically reconfigurable code to achieve the best performance. Compared to these works, we aim to develop a methodology that fully supports DPR techniques and is explicitly centered on energy efficiency, covering in particular complete power modeling and energy-aware scheduling to identify very highly energy-efficient mapping solutions.

EXPLORATION AND SCHEDULING METHODOLOGY

FoRTReSS methodology

Overview

FoRTReSS stands for Flow for Reconfigurable archiTectures in Real-time SystemS [1]. It is a framework helping the user to explore dynamic and partially reconfigurable multiprocessor systems. Given an application or a set of applications, it basically searches a set of reconfigurable regions and processor cores and simulates time-accurate mappings of hardware and software application tasks using fully defined scheduling strategies. Each task has a set of possible implementations, corresponding to a resource and performance trade-off. These implementations can be in hardware (to be mapped on a reconfigurable region) or software (to be mapped on a processor core). A hardware implementation is defined by the actual task resources and performance that can be determined from FPGA synthesis or from estimations (both with the help of HLS). There may be several hardware implementations of the same task to reflect different parallelism trade-offs. A software implementation is mainly characterised by the task execution time on a CPU core, possibly at different operating frequencies. The target architecture is given however by a precise description of FPGA resource organization and a set of processor cores. From application tasks and architecture descriptions, a built-in SystemC/TLM simulator engine called RecoSim is used, first to look for the best reconfigurable regions that are able to correctly host the tasks. Then, using additional performance and timing characteristics (deadline, period,...), it simulates the mapping and scheduling of the full application for various combinations of reconfigurable regions and cores. A detailed description of this methodology is given in publications such as [12] and [13]. In this work, we focus on our extensions and an original methodology that have been developed on this base so as to provide support for very highly energy efficient mapping on multi/many-core platforms with DPR.

Extensions

The goal of the FoRTReSS methodology was initially to ensure a given performance level, potentially under real-time constraints, for a given application. Here we aim to exploit the same DPR flow, focusing initially on real-time reconfigurable multiprocessors, but we extend the methodology so as to manage power consumption and significantly improve the energy efficiency (targeting real-time reconfigurable multi/many-core platforms). The founding principles of this approach arise from recognising the very complex heterogeneity implied by software and hardware execution in a modern multiprocessor system (management of cores and accelerators,

[1] http://fortress-toolbox.unice.fr

DVFS, clock gating, power gating, scheduling, etc.). Therefore the methodology is based on better coupling two decisive and tightly dependent factors in heterogeneous deployments in general, especially when DPR is concerned: design space exploration (or application mapping analysis) to find the best architecture settings with deployment estimations (e.g. how many cores, what types of cores, number, FPGA partitioning, size and shape of RRs,...) and scheduling to further improve the actual mapping at run time on the identified architecture. Indeed, the pertinence of mapping analysis depends obviously on the scheduler which role is essential in the fine-grained exploitation of the architecture resources. Defining dedicated advanced schedulers that better exploit application knowledge at run time is also an important opportunity to grasp, as it can bring up to 50% more energy gains [4]. Additionally, exploration and scheduling require fast and reliable decision support to (i) allow a large design space to be analyzed, and (ii) quickly evaluate scheduling choices at run time. Both processes can greatly benefit from using the same estimations to improve the coherence of design and scheduling decisions.

Power modeling

The first step to achieving this is to extend the various FoRTReSS resource and performance models with efficient power characterization. This can be beneficially based on our previous studies which led to define pragmatic, abstract and overall energy modeling of reconfigurable multiprocessor systems [5][6]. The following describes main features of the models (FPGA, CPU, application and mapping) and some modifications made to fit our requirements, notably concerning multiprocessor power characterization.

FPGA

Previous and current work has been carried out on Xilinx devices. To define a consistent formalization hereafter, we employ a more general terminology for FPGA resources: logic cells (Xilinx *Slices* or Altera *Logic Elements*), RAM and DSP blocks.

The power model for reconfigurable regions is divided into three components that reflect their static, idle and run power. Static power P_j^{static} is the power consumed when region j is empty, and which depends on the configurable resources of the region. We approximate P_j^{static} as a proportion of the full FPGA static power in terms of logic cells only, therefore P_j^{static} can be more conveniently derived for any size and shape of partition.

Since the idle and run power associated with region j depends on the actual task i configured on it, these values are characterized by an application and mapping model. Idle power $P_{i,j}^{idle}$ is the extra power required when task i is configured on region j but not running. Run power $P_{i,j}^{run}$ is the additional power consummed by task i being executed on region j. Idle and run power can be determined by using post-synthesis estimations (e.g. Xilinx Power Estimator) but in the following, they are determined by direct measurements on the FPGA (to improve accuracy of results) for each hardware task implemented.

Reconfiguration overheads must also be addressed to ensure if there's an actual benefit in using complex dynamic reconfiguration. The reconfiguration controller is modeled with two parameters P^{reconf} and T^{reconf} to characterize its power and performance. As with static power, the reconfiguration time T_j^{reconf} and energy E_j^{reconf} required to reconfigure region j are derived from the number of logic cells in the region.

CPU system

Figure 1 shows power measurements of the dual Cortex-A9 in various configurations (frequency, idle and running cores). As with the FPGA model, power can be divided into static, idle and run components. Static power P_{cpu}^{static} has a stable value that can be evaluated at 180.78 mW (intersection of the three linear trends at $F = 0$ MHz). Idle and run power depend on the number of cores and frequency. At this stage, we consider a preliminary model at constant nominal frequency (667 MHz). This model will be extended in future work to DVFS (using representative multiprocessor platforms like Exynos) and also to soft cores (e.g. MicroBlaze). Under these conditions, the idle power P_{cpu}^{idle} of the CPU system is a function of the number of cores N_{cores}:

$$P_{cpu}^{idle} = C_{cpu}^{idle} \times N_{cores}$$

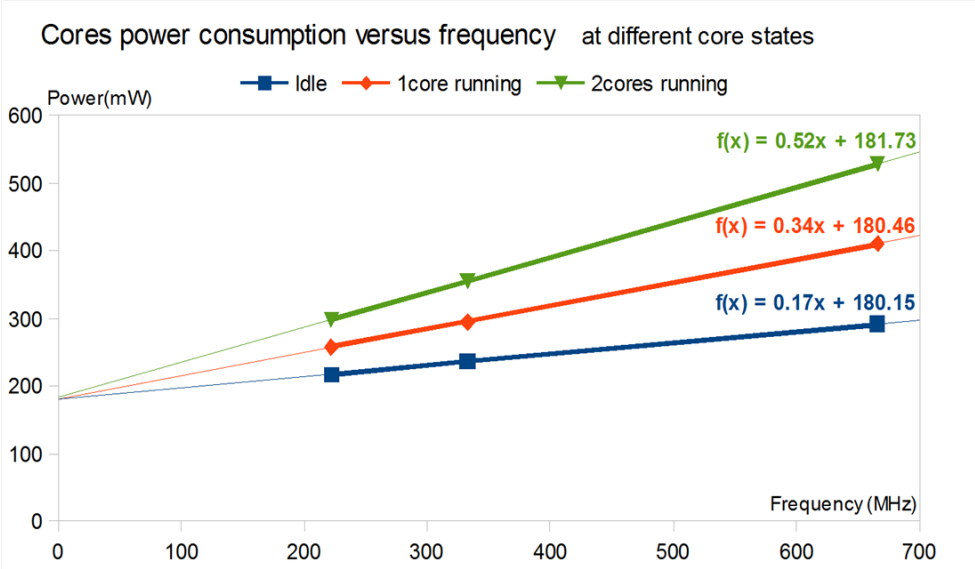

Fig. 1. Power consumption of Zynq processor measured on the ZC702 platform

where C_{cpu}^{idle} is a coefficient expressing the *idle* contribution for a given type of CPU, whose value is 111.66 mW for a Zynq processor (derived from previous measurements). Run power P_{cpu}^{run} of the CPU system is a function of the number of running cores $N_{running\_cores}$:

$$P_{cpu}^{run} = C_{cpu}^{run} \times N_{running\_cores}$$

where C_{cpu}^{run} is a coefficient expressing the *run* contribution for a given type of CPU, whose value is 119.39 mW for a Zynq processor (derived from previous measurements). Therefore, the total CPU power is:

$$P_{cpu} = P_{cpu}^{static} + P_{cpu}^{idle} + P_{cpu}^{run}$$

$$P_{Zynq} = 292.44 + 119.39 \times N_{running\_cores}$$

Application and mapping

An application is characterized by a set of nodes (tasks) with data and execution dependencies, usually modeled by a task flow graph ($\mathcal{G}$). To support the modeling of more complex applications, this specification model has been extended to control structures (which means that communications from a node to another can be conditional), hierarchy (a node can be another task flow graph) and multi-application (several applications can be instantiated).

Each task has one or more implementations available, but every task has at least a software implementation. An implementation is a description of how the task is executed on a software or hardware execution unit. The implementation model reflects the task id i, the execution unit j, an execution time $T_{i,j}$, and – for hardware execution – idle power $P_{i,j}^{idle}$ and run power $P_{i,j}^{run}$. For software execution, idle and run power of a task coincide with idle and run power of a core which can be derived from previous CPU model.

Model evaluation

We apply our model to a real example so as to evaluate the extent of our formalization and show the setting of model parameters from a full measurement process. The input specfication is a License Plate Recognition (LPR) software (C++) to be mapped onto Xilinx platforms (ML605, Zynq-7000). A detailed application description is provided in section A. In the following, we show how the various model parameters were determined from physical measurements on a ZC702 platform (containing a dual-core ARM Cortex-A9 MPCore on a 28nm Xilinx Artix-7 device) to show the applicability and relevance of the models.

In the mapping characterization of section B.3, hardware and software task parameters can be settled by defined implementations and measures (table 1). Classical profiling can be used to set out software execution time and derive the associated energy cost from previous CPU power P_{Zynq} (e.g. $E_{2,1}^{1core\_running} = 412mW * 17.5ms = 7.21mJ$ for the *dilate* function when core 1 is running and core 2 is idle). For the xc7z020 device, *Cores* are the two software execution units ($j = [1;2]$) of the platform and *RRs* are the hardware execution units ($j \geq 3$) that will be automatically defined during the exploration of FPGA partitioning.

Table 1. Hardware / software characterization of LPR application tasks on the ZC702 platform.

Function (i)	Execution unit (j)	$T_{i,j}$(ms)	$P_{i,j}^{idle}/P_{i,j}^{run}$(mW)	$N^{cell}; N^{bram}; N^{dsp}$
Img_load ($i = 1$)	*Core* ($j = [1;2]$)	31	–	–
dilate ($i = 2$)	*Core* ($j = [1;2]$)	17.5	–	–
	RR ($j \geq 3$)	4.3	38/63	2718; 0; 0
erode_fifo ($i = 3$)	*Core* ($j = [1;2]$)	167	–	–
	RR ($j \geq 3$)	12.1	42.6/46	3554; 0; 0
erode ($i = 4$)	*Core* ($j = [1;2]$)	17.3	–	–
	RR ($j \geq 3$)	4.4	35/64	2681; 0; 0
binarize ($i = 5$)	*Core* ($j = [1;2]$)	24	–	–
Img_preproc ($i = 6$)	*Core* ($j = [1;2]$)	24	–	–
Img_write ($i = 7$)	*Core* ($j = [1;2]$)	15	–	–

In the following, hardware tasks are fully generated using an Electronic System Level (ESL) methodology described in [**?**] to provide maximum relevance to our results. Mapping parameters are thus derived from measurements made possible by full accelerator implementation (they could have been defined more easily using estimation tools like Xilinx Power Estimator). $P_{i,j}^{idle}$ is the consumption measured when hardware task i is configured on RR j but not running. This power is supposed to be independent from RRs in our model. $P_{i,j}^{run}$ is the fraction of dynamic power added when hardware task i is running on RR j (also assumed to be independent of the RR used), that can be determined in practice by subtracting the consumption of a configuration where the task is running from the consumption of a configuration where the task is idle. The total power consumption of hardware task i on RR j is therefore the sum of P_j^{empty} of RR j and $P_{i,j}^{idle}$ when the task is idle, plus an additional contribution $P_{i,j}^{run}$ when the task is running. For example, the hardware implementation of the *dilate* function (task 2) on RR 3 (assuming it is made of 4000 slices) in table 1 has a total energy cost $E_{2,3}^{run}$ computed from P_3^{empty} of RR j ($4000 \times P_j^{static} = 4000 \times 0.002180mW/slice = 8.72mW$), $P_{2,3}^{idle}/P_{2,3}^{run}$ of function *dilate* (task 2), and the corresponding execution time $T_{2,3}$:

$$\begin{aligned} E_{2,3}^{run} &= (P_3^{empty} + P_{2,3}^{idle} + P_{2,3}^{run}) \times T_{2,3} \\ &= (8.72mW + 38mW + 63mW) \times 4.3ms \\ &= 0.472mJ \end{aligned}$$

ENERGY-AWARE HETEROGENEOUS SCHEDULER

This section describes an advanced energy-aware heterogeneous scheduler (EAHS) developed within the above framework and which exploits DPR so as to minimize the energy cost, possibly under performance and deadline constraints. The scheduling procedure involves two steps: a waiting task list is first determined and sorted according to specific criteria (e.g. EDF – earliest deadline first), then a task implementation is chosen and mapped to a resource such that the resulting execution cost (including reconfiguration overheads) is minimized. As such it requires that the power impact of resource allocation decisions can be correctly and quickly predicted, which relies on the models of section B.

Task Scheduling

Common scheduling strategies fall under two broad categories: time sharing and time-critical scheduling. Time-sharing algorithms are based on the principle where each process takes an equal share of CPU time in turn, without priority (i.e. round robin). On the other hand, time-critical scheduling is more concerned with classes of real-time systems where processing of jobs must be completed under strict execution constraints.

Deadline scheduling, for example, is often associated with dynamic priorities that increase when a process becomes more critical. However, the strict definition of deadlines may be alleviated for the sake of heterogenous scheduling as power becomes the growing constraint. This provides ways to derive energy-aware strategies under performance constraints. Considering, for example, a scenario where the top task in the waiting list is blocked, waiting for a matching execution unit to be free, then all other tasks in the waiting list are blocked despite the fact that they may be run on other free execution units. This tends to increase the application latency and underutilization of resources. Thus we can devise a *soft* scheduling approach which consists of using a deadline strategy (e.g. EDF) while leaving the possibility for a blocked task to be bypassed when its corresponding mapping is momentarily infeasible (e.g. either the reconfiguration controller is busy or a compatible execution unit is not free).

Task Mapping

When processing the allocation of a task to the resources of a heterogeneous platform, the choice of implementation is the key part of reducing energy. A cost function is thus defined to help identify the most energy-efficient mapping from all possibilities. This cost is computed each time a task at the top of the waiting list is eligible for execution, for all its possible implementations, and this process is based on energy and execution time estimations of the implementations of the task being processed.

The energy estimation is given by equation 1:

$$\forall j \; ; \; E_j^{cost} = \left\{ \begin{array}{ll} E_{i,j}^{run} & \text{when } \rho_{i,j} = 0 \\ E_{i,j}^{run} + E_j^{reconf} & \text{when } \rho_{i,j} = 1. \end{array} \right\} \quad \textbf{(1)}$$

where

$$\begin{aligned} E_{i,j}^{run} &= P_{i,j}^{run} \times T_{i,j} \\ E_j^{reconf} &= T_j^{reconf} \times P^{reconf} \end{aligned}$$

and $\rho_{i,j}$ represents the need to perform a reconfiguration or not. For instance, if the same implementation is already configured, $\rho_{i,j} = 0$ means that region j can be used directly to run task i, otherwise $\rho_{i,j} = 1$ and a reconfiguration is required before execution. For all non-reconfigurable execution units, $\rho_{i,j} = 0$.

To execute a new task, the scheduler has to check first that the execution unit j currently evaluated is free. If not, the task can be implemented later and this delay must be considered for decision. Execution time of a task is thus estimated as described in equation 2:

$$\forall j \; ; \; T_j^{cost} = T_{i,j} + T_j^{reconf} \times \rho_{i,j} + T_j^{busy} \quad \textbf{(2)}$$

where T_j^{busy} represents the time during which execution unit j is busy running another task i', which can be estimated from the execution time $T_{i',j}$, the start time of current task i and the current scheduling time.

The final cost for execution unit j is computed in equation 3, where α is a user parameter (a real value between 0 and 1) to promote either performance (α close to 0) or energy (α close to 1).

$$\begin{aligned} &\forall j \; ; \\ Cost_j \quad &= \alpha \, \frac{E_j^{cost}}{max(E^{cost})} + (1 - \alpha) \, \frac{T_j^{cost}}{max(T^{cost})} \end{aligned} \quad \textbf{(3)}$$

The cost function is evaluated for all available implementations and execution units for the current top task in the waiting list. The lowest $Cost_j$ value is selected and the task is implemented on execution unit j. However, if execution unit j is busy the task is kept waiting and will be implemented later.

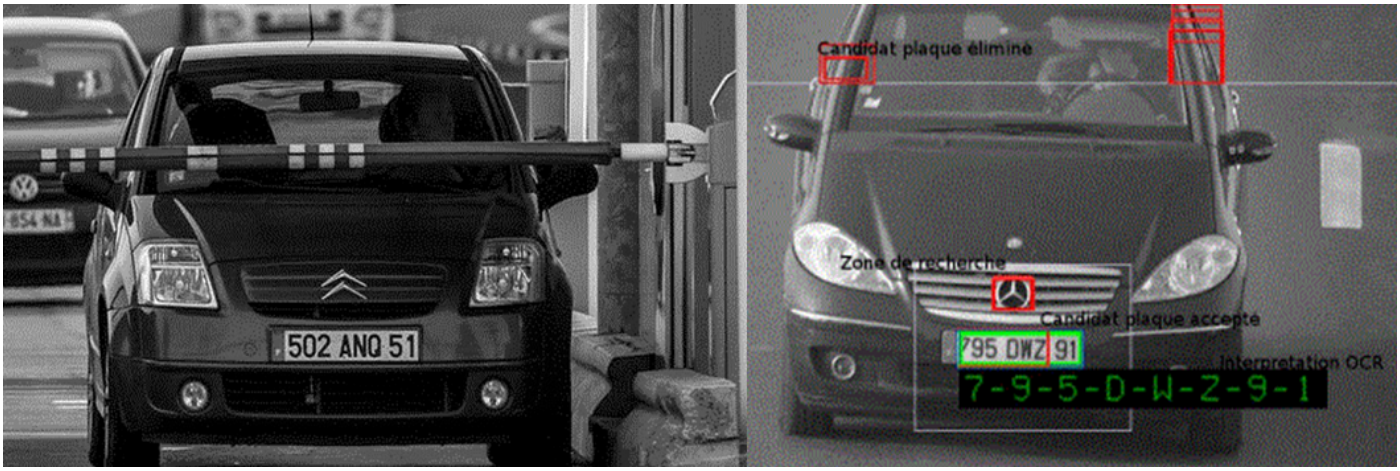

Fig. 2. License plate detection and recognition in a configuration of two monitored toll lanes (2-lane LPR).

APPLICATION TO A SMART CAMERA SYSTEM

This section illustrates the search for significant energy efficiency improvement in a manycore system with DPR accelerators. This real-life case study concerns a mobile vehicle license plate tracking and recognition application developed for the purpose of highway traffic management. Two representative families of devices are used to address realistic DPR implementations with Xilinx-based FPGAs: Virtex-6 with MicroBlaze soft IP cores and Zynq-7000 with ARM Cortex-A9 hard IP cores. All power and performance models used in this application study are therefore based on full prototyping hardware accelerators and real platform measurements (rather than estimations) so as to improve the relevance of the subsequent power analysis. The two platforms used for this characterization are the ML605 and ZC702 evaluation boards.

LPR application overview

License Plate Recognition (LPR) is popular in the transportation industry for its applications in managing traffic congestion and in monitoring, security and access control systems. In our particular scope of application, the LPR system is designed for automatic detection and identification of vehicles passing highway toll barriers. Cameras (one per lane) are set up to detect and follow the front plates of cars in each lane. The system can process different lanes in parallel; this is illustrated for two lanes in figure 2. Here, we will consider an LPR system that can be set up to monitor 1, 2, 4 or 8 lanes simultaneously (refered to as 1-, 2-, 4- and 8-lane LPR).

The lower area of each lane's video feed is used to track the license plate of the current car while the upper area is used to detect the plate of the next car. The two regions (respectively *Main task* and *Follow task*) are visible on the right side of figure 2. Both regions undergo a succession of typical morphological operations and transforms (erosion, dilation, ...) to improve the quality of detection and recognition. A sequence of operations (*dilate*, *erode_fifo*, *erode*, *binarize*, *preprocessing*) is applied seven times for the detection area and ten times for the recognition area, as depicted in the task flow graph of figure 3. This graph, which is composed of 291 nodes, specifies basic LPR process for one lane and can be very easily replicated to process multiple tracks using the multi-application feature of FoRTReSS (section B.3).

In the enhanced detection region, Optical Character Recognition (OCR) is used to identify alphanumeric characters, with a multi-layer artificial neural network (FANN [28]). Aside from the learning phase, application profiling shows that the neural network recognition phase can be neglected in comparison with the very computationally intensive morphological operations, and so we won't address FANN processing acceleration here. More specifically, it turns out that three morphological functions account for 79% of the total execution time: *erode_fifo* (36.68%), *dilate* (25.27%) and *erode* (16.71%). Therefore the maximum theoretical speed-up for the whole LPR process is 4.7× which combines with the energy gains from any hardware implementations of the *erode_fifo*, *dilate* and *erode* functions and points to significant overall gains in energy efficiency. This energy efficiency is reported below as an energy–delay product (EDP) reduction compared with software execution

(or speed-up × energy gain).

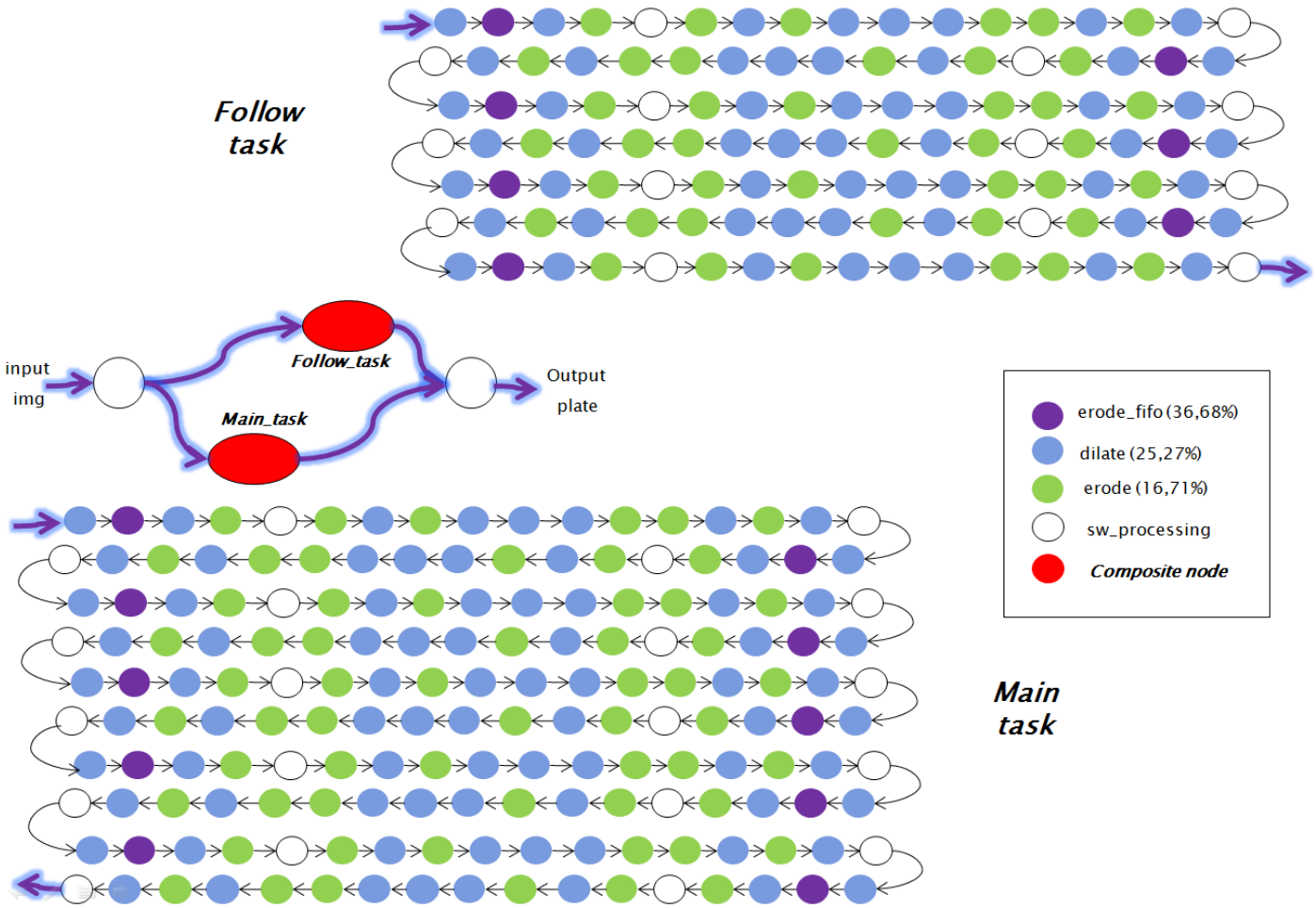

Fig. 3. Task flow graph of basic LPR processing (1-lane LPR)

Deployment on Virtex-6 / MicroBlaze

In this deployment analysis, we report and discuss the performance, energy and efficiency gains of the LPR application in various configurations (1-, 2-, 4- and 8-lane LPR). First, we consider a Xilinx Virtex-6 FPGA with MicroBlaze soft IP cores. All necessary power / performance characterization of hardware and software tasks were derived from implementation, execution and measurements on an ML605 evaluation board, as depicted in section B.4. Each reference software solution (one for each LPR configuration) corresponds to the most energy-efficient multicore execution, i.e. the one able to achieve maximum performance with the parallelism of the configuration (i.e. 2 cores for 1-lane LPR, 4 cores for 2-lane LPR, 8 cores for 4-lane LPR, 16 cores for 8-lane LPR).

The lower part of figure 4 shows stable software performance while scaling application configurations with the number of cores (i.e. 1-lane LPR / 2 cores, 2-lane LPR / 4 cores, etc.). Logically, software energy use doubles with each successive LPR configuration (requiring double the number of cores), while DPR solutions always perform better with half the number of cores. For 8-lane LPR, for example, DPR acceleration (8 cores + 6 RRs) achieves much better results than software (16 cores) with respectively 4× better performance and 4.2× less energy (top part of figure 4). It is worth noting that both the speed-ups and the energy gains of DPR compared with software execution remain very close across all four LPR configurations, and that energy efficiency (i.e. the energy–delay product) improved up to a factor of 19 (4.5× speedup, 4.2× energy gain) for 1-lane LPR.

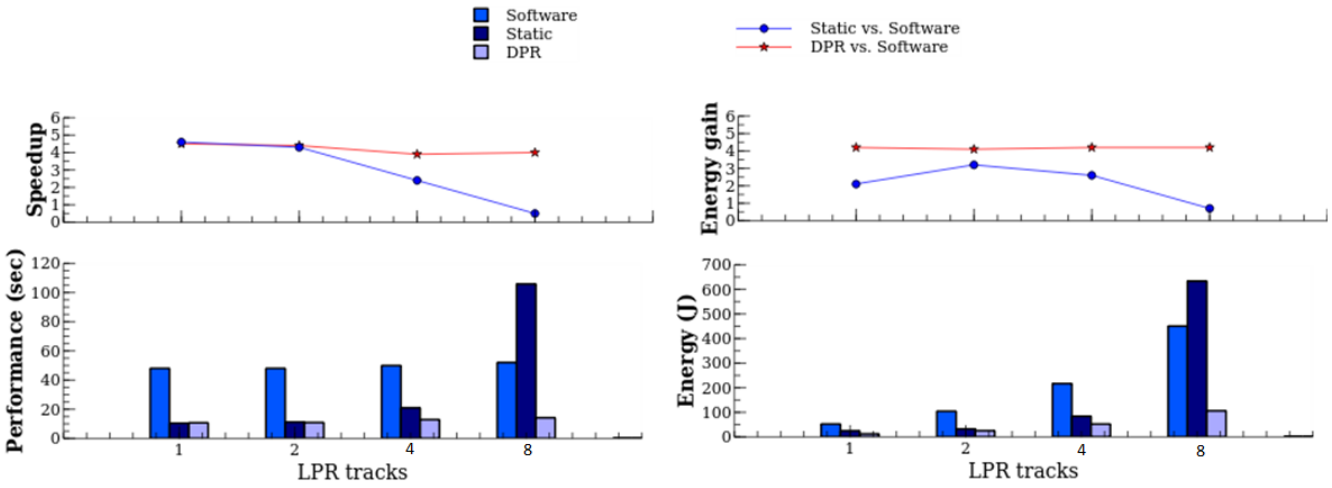

Fig. 4. Speed-up and energy gain of static and DPR acceleration against software execution for XC6VLX240T device

In the various lane configurations we have considered, static solutions combine a minimum number of cores with three accelerators (one for *dilate*, one for *erode_fifo*, one for *erode*). We intentionally limited the number of static accelerators to three – whatever the LPR configuration – because otherwise the FPGA would rapidly run out of space. This reflects in more performance variability and performs generally worse than DPR. More precisely, the gap is widening as LPR configuration (thus parallelism) grows. For 1-lane LPR, for example, DPR is only 2.6% slower than static acceleration because it is affected little by having less hardware parallelism (1 RR vs. 3 static accelerators) and reconfiguration latencies. However, this decrease of logic resources results in 50.3% less energy consumption. For 8-lane LPR, DPR benefits from optimal processing parallelism (8 cores + 6 RRs) to produce a speed-up of 8.1× and a 6× energy gain over static acceleration (8 cores + 3 accelerators). A static solution is inefficient in this case, even less than software execution (16 cores), as it is strongly disavantaged by only three accelerators to process eight LPR lanes in parallel. In the end, static acceleration has improved the energy efficiency from 13.9× (4.3× speedup, 3.2× energy gain) to 0.3× (0.5× speedup, 0.7× energy gain) against reference software executions, while DPR improved from 16.1× (3.9× speedup, 4.2× energy gain) to 19× (4.5× speedup, 4.2× energy gain).

Deployment on Zynq-7000

We address the same LPR benchmark and configurations considering now two Zynq-7000 platforms based on XC7Z020 and XC7Z045 devices (respectively 85K and 350K logic cells) with ARM Cortex-A9 hard IP cores. All necessary power / performance characterization was derived from implementation, execution and measurements on a ZC702 evaluation board. In the subsequent analysis and exploration, we assume the original dual-core capability can be extended up to sixteen Cortex-A9 cores to better cope with the processing requirements of the defined LPR configurations.

XC7Z020

We address first the XC7Z020 device of a ZC702 evaluation board which served for the purpose of hardware / software performance and power characterization. Similar trends can be observed in the results of figure 5 with regards to previous XC6VLX240T FPGA (figure 4). DPR is still more efficient than static solutions, but the difference is smaller. In addition, DPR provides less benefit with this device when parallelism grows, now with 1.3× speed-up and 3× energy gain over software execution (8-lane LPR). The reason relates clearly to the size of the device which does not allow the definition of more than three regions, therefore limiting the maximum gains closer to those of static acceleration (3 accelerators) for high parallelism levels (4- and 8-lane LPR). This limitation in size is also the reason for the decline in speed-up / energy gain in these configurations.

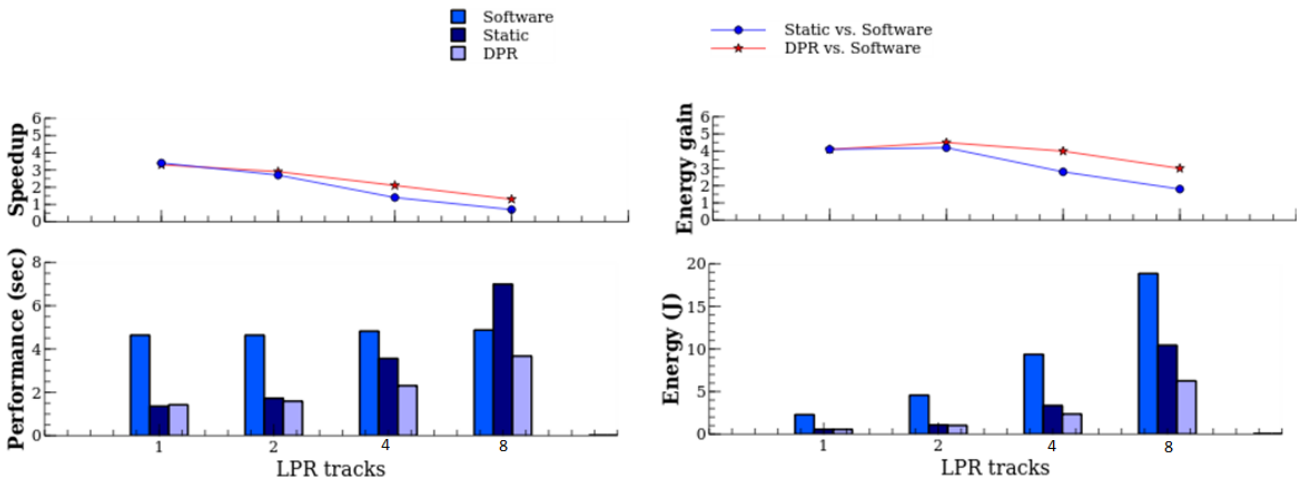

Fig. 5. Speed-up and energy gain of static and DPR acceleration against software execution for the XC7Z020 device

However, energy efficiency improvements over software execution are still significant for lower paralellism levels (1- and 2-lane LPR) with respectively 13.5× (3.3× speed-up, 4.1× energy gain) and 13× (2.9× speed-up, 4.5× energy gain). These gains are below the results of Virtex-6 (19×) mainly because MicroBlaze is much slower compared to Cortex-A9 cores, leading to greater hardware versus software acceleration values. The best DPR performance for 2-lane LPR, for instance, is 1.6 s compared to 11 s for Virtex-6 / MicroBlaze. Besides the undeniable improvement of both DPR and static acceleration, this example shows a moderate advantage in using DPR compared to static acceleration, because the potential for processing several hardware functions in parallel for highly parallel configurations exceeds the device capability (3 RRs max). The next study on a larger device will allow to further investigate this.

XC7Z045

For this device, the energy efficiency of DPR improves from 9.3× (3.0× speed-up, 3.1× energy gain) to 11.7× (3.5× speed-up, 3.4× energy gain) compared to software execution (figure 6). The increase in programmable logic resources improves the parallel hardware processing potential with up to 11 RRs (against 3 for the XC7Z020), which raises the speedup around 3× to 3.5×. Speed-ups and energy gains of DPR over software execution now remain at similar levels accross all four LPR configurations. Overall performance is stable at around 1.5 s whereas it ranged from 1.4 s to 3.7 s for the XC7Z020, meaning an ideal performance scaling up to the 8-lane configuration for the XC7Z045.

In comparison, again there are more variations for static solutions with energy efficiency varying from 9.6× (3.4× speed-up, 2.8× energy gain) to 1.1× (0.7× speed-up, 1.6× energy gain) improvement over software execution. The static speed-up follows the same declining trend coming from the limitation to three static accelerators. In a high-parallelism solutions (4- and 8-lane LPR), DPR is 3× and 10× more efficient than static acceleration, illustrating the excellent job done by the scheduler in exploiting RRs and cores efficiently (section 4).

There is an optimal set of RRs for the application / device which becomes more significant as the parallelism grows. If we focus on 4-lane LPR, for example, table 2 reports the set of RRs explored by RecoSim from the possible function implementations (section B.4) and from FPGA layout knowledge, with the corresponding performance (*Tex*), energy (*En*), speedup (*Spu*), energy gain (*En.gain*) and energy efficiency (*Spu*×*En.gain*) improvement over software execution.

The simulator defines and considers between 1 and 11 regions to map the application on the XC7Z045. A set of 8 RRs provides the best energy efficiency with an overall 11.7× improvement (3.5× speed-up, 3.4×

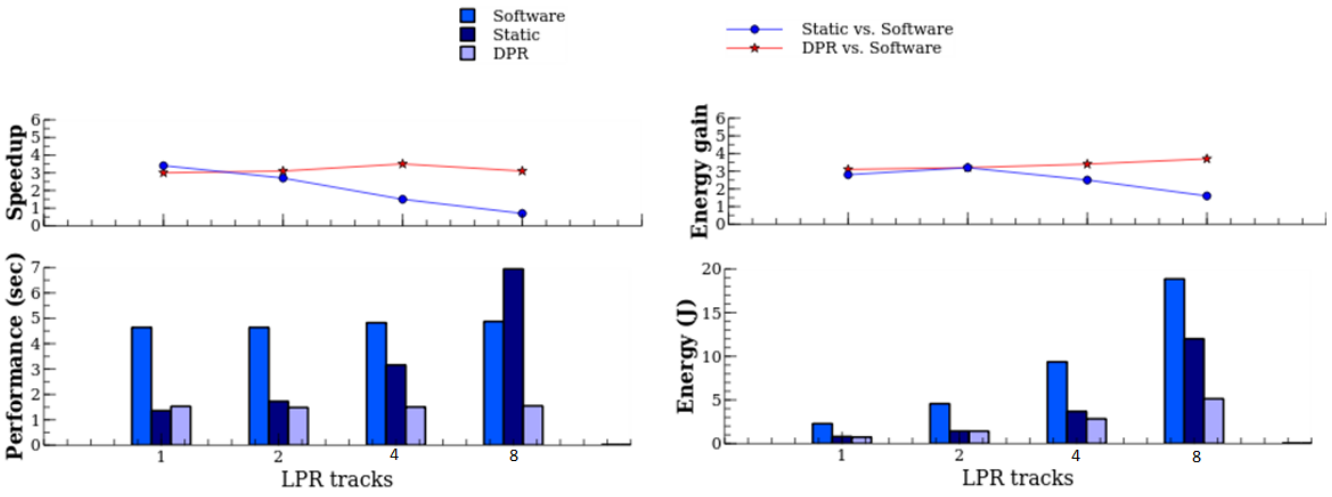

Fig. 6. Speed-up and energy gain of static and DPR acceleration against software execution for the XC7Z045 device

Table 2. Impact of different RR configurations for 4-lane LPR (XC7Z045, XC6VLX240T).

	XC7Z045					XC6VLX240T				
	Tex(sec)	En(J)	Spu	En.gain	Spu×En.gain	Tex(sec)	En(J)	Spu	En.gain	Spu×En.gain
8cores	4.8	9.4	1	1	1	49.9	217	1	1	1
4cores_1RR	5.5	4.6	0.9	2	1.8	28.7	78.6	1.7	2.8	4.8
4cores_2RRs	2.9	3.1	1.6	3	4.9	15.5	52.9	3.2	4.1	13.2
4cores_3RRs	2.3	2.9	2.1	3.2	6.6	12.9	52.2	3.9	4.2	16.1
4cores_4RRs	1.9	2.7	2.5	3.4	8.6	12.4	57.9	4	3.7	15
4cores_5RRs	1.7	2.7	2.8	3.4	9.8	12.3	65.4	4	3.3	13.4
4cores_6RRs	1.5	2.6	3.1	3.6	11.2	11.2	66	4.5	3.3	14.7
4cores_7RRs	1.5	2.8	3.2	3.3	10.6	11.5	75.3	4.3	2.9	12.5
4cores_8RRs	1.4	2.8	3.5	3.4	11.7	11.1	78.7	4.5	2.8	12.4
4cores_9RRs	1.4	2.9	3.5	3.2	11.2	11	84.4	4.5	2.7	11.6
4cores_10RRs	1.4	3.1	3.5	3	10.6	–	–	–	–	–
4cores_11RRs	1.4	3.3	3.5	2.9	9.9	–	–	–	–	–

energy gain). Comparison with the XC6VLX240T, for which the best solution is achieved with only 3 RRs, points to the fact that the best RR configuration is likely to be different from one device to another. Finding an optimal RR partitioning for the application is therefore very important, as scheduling (which largely depends on partitioning) is an essential condition of DPR efficiency, for which we will strive to provide an effective measure in the next section.

Overall efficiency analysis

The main outcomes of these simulations are firstly that a variety of conditions influence proper exploitation of the complex mapping heterogeneity and the amount of DPR benefit. Not all kinds of processing suit DPR acceleration. A significant share of hardware functions is the primary necessary (but not sufficient) condition for significant efficiency (over 75% at full application level), and the parallelism of tasks (i.e. the ability to run concurrently) will greatly influence the quality of the results. If the benefits of DPR and static hardware against software execution are a given in most cases, the existence of a potential gain of DPR over static is less trivial. To really improve processing efficiency, the platform needs to provide adequate resources to fully exploit the parallelism in the application. From the LPR case study, which has been investigated in detail, DPR can reach up to more than 3 times better energy efficiency than static acceleration under fair parallelism conditions (e.g. 4-track LPR).

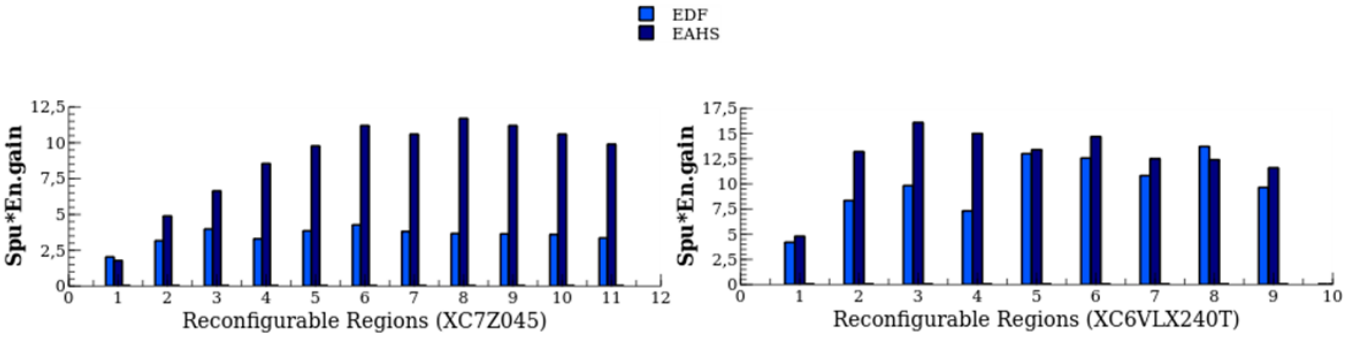

Fig. 7. Scheduler efficiency on 4-lane LPR (left XC7Z045, right XC6VLX240T).

The essential conditions to achieve this are scheduling and FPGA partitioning. To assess the extent of these aspects, we have set up simulations where a standard real-time scheduler (earliest deadline first, EDF) is used in place of the energy-aware heteregeneous scheduler (EAHS) of section 4 (figure 7). With 4-lane LPR, results show a drop of 63.4% in energy efficiency for the XC7Z045 / Cortex-A9 and 14.6% for the XC6VLX240T / MicroBlaze. These comparisons provide an idea of the potential impact of scheduling and partitioning on which energy efficiency strongly depends. This case study therefore points out the importance of a methodic holistic exploration and mapping methodology to properly exploit the potential of DPR. The benefits when compared with software execution are a given in most cases, but the real issue is whether DPR is worth the design effort compared with classical (static) FPGA acceleration. With poor static (system sizing and region partitioning) and / or dynamic (scheduling and mapping) choices, the real benefit is likely to be missed because a great deal of static and dynamic variables come into play as far as energy is concerned. For example, a crucial point is to identify adequate trade-offs in technology (device, IP cores) and partitioning (number, size and resources for the programmable logic) because a small programmable logic area will not allow the application parallelism to be fully exploited, whereas a large area will draw a lot of power due to the relative high FPGA static consumption. For the same application, this can lead to significant variations in results from one platform to another. Considering 1-lane LPR, for example, DPR is twice as energy efficient than static on the XC6VLX240T / MicroBlaze, owing to the reduction in size of programmable logic resources (1 RR versus 3 accelerators) and the high static power coming with this large device. On Zynq-7000, DPR and static have a very similar energy efficiency, which probably makes DPR too complex to implement given the benefit.

CONCLUSIONS

This study makes a number of contributions to power analysis designed to make the problems involved in achieving higher energy efficiency more intelligible. Firstly, on better exploiting heterogenity in complex multiprocessor systems, the methodology it defines brings significant energy efficiency improvements compared with homogeneous multiprocessor execution: 16.1× to 19× (Virtex-6 / MicroBlaze), 13.5× to 4× (XC7Z020 / Cortex-A9), and 11.7× to 9.3× (XC7Z045 / Cortex-A9). These results underline the benefits of a more methodical approach, improving interaction across three key areas: design space exploration, scheduling, and decision support. The relevance of this on a practical level relies on a realistic decision-support scheme common to exploration and scheduling, and an energy-aware heterogeneous scheduler which is effective in getting the best out of heterogeneous resources and techniques.

On the more specific question of exploiting hardware acceleration, the contributions are essentially defined by the determination of the potential of DPR, both in terms of improvement when compared with multiprocessor execution as stated previously, but also compared with static acceleration (1.1× to 2.6× for Virtex-6 / MicroBlaze, 1× to 2.2× for XC7Z020 / Cortex-A9, and 0.9× to 1.7× for XC7Z045 / Cortex-A9). These results are driven by concrete power characterisation efforts of DPR, now making it easier to determine mapping solutions that can be markedly more efficient than static acceleration, and – just as importantly – to find out if and how dynamic reconfiguration can be used correctly.

Finally, regarding the general question of advances in low power research, existing approaches are broadly divided in two categories: those that build on classical approaches (with power as a secondary requirement after performance, or adressing different techniques separately) and those that explore new techniques and technologies (e.g. process nodes, memories, . . .). This work uncovers another direction based on integrated and power-dedicated approaches, which is often overlooked. In light of the results, this smarter integration of techniques brings high hopes to extend energy efficiency beyond a factor of ten, which is a major challenge for many upcoming critically power-constrained application domains such as embedded systems (IoT, security, artificial intelligence), telecommunications (5G) and high-performance computing (Exascale).

ACKNOWLEDGMENTS

This work was carried out under the BENEFIC project (CA505), a project labeled within the framework of CATRENE, the EUREKA cluster for Application and Technology Research in Europe on NanoElectronics.

REFERENCES

1. Agne A., Happe M., Keller A., Lubbers E., Plattner B., Plessl C., Platzner M. Reconos: an operating system approach for reconfigurable computing. IEEE Micro. 2014;34(1):60 –71.
2. Bazargan Kiarash, Kastner Ryan, Sarrafzadeh Majid. Fast template placement for reconfigurable computing systems. IEEE Des. Test. 2000;17(1):68–83.
3. Benini L., Bogliolo A., De Micheli G. A survey of design techniques for system-level dynamic power management. IEEE Transactions on Very Large Scale Integration (VLSI) Systems. 2002;8(3):299–316.
4. Bilavarn S., Khan J. J., Belleudy C., Bhatti M. K. Effectiveness of power strategies for video applications: a practical study. Journal of Real-Time Image Processing. 2014;12(1):123–132.
5. Bonamy R., Bilavarn S., Chillet D., Sentieys O. Power consumption models for the use of dynamic and partial reconfiguration. Microprocessors and Microsystems. 2014;38(8, Part B):860 –872.
6. Bonamy R., Bilavarn S., Chillet D., Sentieys O. Power modeling and exploration of dynamic and partially reconfigurable systems. Journal of Low Power Electronics. 2016;12(3):172–185.
7. Bower F.A., Sorin D.J., Cox L.P. The impact of dynamically heterogeneous multicore processors on thread scheduling. IEEE Micro. 2008;28(3):17–25.
8. Chitlur N., Srinivasa G., Hahn S., Gupta P. K., Reddy D., Koufaty D. A., Brett P., Prabhakaran A., Zhao L., Ijih N., Subhaschandra S., Grover S., Jiang X., Iyer R. Quickia: Exploring heterogeneous architectures on real prototypes. 2012:433–440.
9. Damak T., Werda I., Bilavarn S., Masmoudi N. Fast prototyping h.264 deblocking filter using esl

tools. Transactions on Systems, Signals & Devices, Issues on Communications and Signal Processing. 2013;8(3):345–362.
10. DeVuyst M., Kumar R., Tullsen D. M. Exploiting unbalanced thread scheduling for energy and performance on a cmp of smt processors. Proceedings of the 20th International Conference on Parallel and Distributed Processing. 2006:140–140.
11. Ding H., Huang M. Polypc: Polymorphic parallel computing framework on embedded reconfigurable system. 27th International Conference on Field Programmable Logic and Applications. 2017:1 –8.
12. Duhem F., Muller F., Aubry W., Gal B. Le, Négru D., Lorenzini P. Design space exploration for partially reconfigurable architectures in real-time systems. Journal of Systems Architecture. 2013;59(8):571 –581.
13. Duhem F., Muller F., Bonamy R., Bilavarn S. Fortress: a flow for design space exploration of partially reconfigurable systems. Design Automation for Embedded Systems. 2015;19(3):301–326.
14. Ghiasi S., Keller T., Rawson F. Scheduling for heterogeneous processors in server systems. Proceedings of the 2Nd Conference on Computing Frontiers. 2005:199–210.
15. Hong C., Benkrid K., Iturbe X., Ebrahim A., Arslan T. Efficient on-chip task scheduler and allocator for reconfigurable operating systems. Embedded Systems Letters. 2011;3(3):85–88.
16. Imes C., Hoffmann H. Minimizing energy under performance constraints on embedded platforms: Resource allocation heuristics for homogeneous and single-isa heterogeneous multi-cores. SIGBED Rev. 2015;11(4):49–54.
17. Javaid H., Shafique M., Henkel J., Parameswaran S. System-level application-aware dynamic power management in adaptive pipelined mpsocs for multimedia. 2011:616–623.
18. Kathail V., Hwang J., Sun W., Chobe Y., Shui T., Carrillo J. Sdsoc: A higher-level programming environment for zynq soc and ultrascale+ mpsoc. Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays. 2016.
19. Leech C., Kazmierski T. J. Energy efficient multi-core processing. ELECTRONICS. 2014;18(1):3–10.
20. Liang T., Feng L., Sinha S., Zhang W. Paas: A system level simulator for heterogeneous computing architectures. 27th International Conference on Field Programmable Logic and Applications. 2017:1 –8.
21. Liu S., Pittman R. N., Forin A., Gaudiot J. L. Achieving energy efficiency through runtime partial reconfiguration on reconfigurable systems. ACM Trans. Embed. Comput. Syst. 2013;12(3):72:1–72:21.
22. Nguyen T., Gurumani S., Rupnow K., Chen D. Fcuda-soc: Platform integration for field-programmable soc with the cuda-to-fpga compiler. Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays. 2016:5–14.
23. Ozkan M. A., Reiche O., Hannig F., Teich J. Fpga-based accelerator design from a domain-specific language. Proceedings of the 26th International Conference on Field-Programmable Logic and Applications. 2016.
24. Paulsson K., Hubner M., Becker J. Dynamic power optimization by exploiting self-reconfiguration in xilinx spartan 3-based systems. Microprocessors and Microsystems. 2009;33(1):46 –52.
25. Schwiegelshohn F., Hubner M. An application scenario for dynamically reconfigurable fpgas. Proceedings of the 9th International Symposium on Reconfigurable and Communication-Centric Systems-on-Chip. 2014.
26. Singh P., Chinta V. Survey report on dynamic power management, Chicago, USA: University of Illinois, Chicago (ECE Department); 2008.
27. Snavely A., Tullsen D. M. Symbiotic jobscheduling for a simultaneous mutlithreading processor. SIGPLAN Not. 2000;35(11):234–244.
28. Steffen N. Fast artificial neural network library. http : //leenissen.dk/f ann/wp/. 2013.
29. Sterpone L., Carro L., Matos D., Wong S., Fakhar F. A new reconfigurable clock-gating technique for low power sram-based fpgas. Design, Automation & Test in Europe (DATE), 2011. 2011:1–6.
30. Teodorescu R., Torrellas J. Variation-aware application scheduling and power management for chip multiprocessors. Proceedings of the 35th Annual International Symposium on Computer Architecture. 2008:363–374.
31. Tuan T., Lai B. Leakage power analysis of a 90nm fpga. IEEE Custom Integrated Circuits Conference, 2003. 2003:433–440.

32. Wang Q., Gupta S., Anderson J. H. Clock power reduction for virtex-5 fpgas. Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays. 2009:13–22.
33. Weller D., Oboril F., Lukarski D., Becker J., Tahoori M. Energy efficient scientific computing on fpgas using opencl. Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays. 2017:247–256.
34. Winter J. A., Albonesi D. H., Shoemaker C. A. Scalable thread scheduling and global power management for heterogeneous many-core architectures. Proceedings of the 19th International Conference on Parallel Architectures and Compilation Techniques. 2010:29–40.
35. Wipliez M., Roquier G., Nezan J. F. Software code generation for the rvc-cal language. Journal of Signal Processing Systems. 2011;63(2):203 –213.

VITAE

Robin Bonamy received his M.E. degree in Electronics and Embedded Systems in 2009 and the Ph.D. degree from University of Rennes, France, in 2013. His research interests are reconfigurable devices, power and energy models and design space exploration especially for energy consumption reduction. Since 2015 he is consultant in innovation based on digital electronic technologies. With a global approach he allows the successful development of an idea to the prototype. He is currently focused on low power, wireless devices and security in IoT.

Sébastien Bilavarn received the B.S. and M.S. degrees from the University of Rennes in 1998, and the Ph.D. degree in electrical engineering from the University of South Brittany in 2002 (at formerly LESTER, now Lab-STICC). Then he joined the Signal Processing Laboratories at the Swiss Federal Institute of Technology (EPFL) for a three year post-doc fellowship to conduct research with the System Technology Labs at Intel Corp., Santa Clara. Since 2006 he is an Associate Professor at Polytech'Nice-Sophia school of engineering, and LEAT Laboratory, University of Nice-Sophia Antipolis - CNRS. His research is in the general field of system level design and methodologies for very high energy efficient processing systems, addressing key issues related to heterogeneous, reconfigurable and multiprocessor architectures on a great number of challenging french, european and international collaborative research projects.

Fabrice Muller received the Ph.D. degree in electrical engineering from the University of Nantes in 2000 and he joins the Polytech'Nice-Sophia engineering school and I3S Laboratory as Associate Professor. From 2007, he changes for the LEAT/CNRS Laboratory. In December 2011, he got the H.D.R (Habilitation à Diriger des Recherches) on his work on multiprocessor architectures, dynamically distributed and reconfigurable for real-time applications. Since 2018, he is at Polytech'Lab. He is currently Director of the Department of Electronic Systems Engineering at Polytech Nice Sophia and in charge of the quality management ISO9001:2015. He focuses on co-design and reconfigurable architectures on FPGA targets, hardware task placement/partitioning and scheduling. He also works on implementations of hardware/software RTOS and on future adaptive architectures. He was involved as co-author in the software Cofluent Design created in 2003, acquired by Intel in 2011. Since 2012, he currently designs a FoRTReSS tool box (Flow for Reconfigurable archiTectures in Real-time SystemS) for the evaluation of the energy consumption at system level and on the generation of a middleware which makes it possible to execute and schedule applications modeled under FoRTReSS on embedded architectures supporting Linux.

François Duhem holds a MS. Degree in electronics engineering from Polytech'Nice-Sophia Antipolis (University of Nice, France). In 2012, he completed his PhD thesis in the Laboratory of Electronics, Antennas and Telecommunications (LEAT) in Sophia Antipolis on a methodology for designing dynamically and partially reconfigurable architectures on FPGAs targeting real-time applications. After four years working for Thales Research & Technology on high-performance computing architectures based on FPGAs and heterogeneous Systems-on-Chip, he joined the French Alternative Energies and Atomic Energy Commission (CEA) in 2017 as a research scientist on digital design of spintronic integrated circuits.

Simon Heywood is a software developer who enjoys building reliable solutions to complex problems. From 2001 to 2017 he was an R&D engineer at Thales, where he participated in a number of national and European research projects related to networks, information security, and embedded systems. He has a bachelor's degree in Physics from the University of Warwick and a master's degree in Computer Science and Engineering from Chalmers University of Technology. He currently works as a consultant for HiQ in Gothenburg, Sweden.

Philippe Millet is the head of the high performance computing laboratory at Thales Research and Technology (TRT) where he leads a team of 10 people consisting of doctors, engineers and PhD students.He is also leading the high performance computing architecture research topic throughout the Thales group. For this activity, he writes proposals, leads research projects and defines the strategy for the research to be conducted according to the business units needs, the upper management strategy and funding opportunities. He is an expert in software and hardware embedded real-time architectures and is experienced in team lead and project management.

Fabrice Lemonnier is a senior Research Engineer at Thales Research and Technology (TRT). He is leader of High Performance Computing lab in which they are research activities on architectures and tools for embedded HPC systems. Before joining TRT, he has worked on software interface standardization for military avionic modules (RTOS and communication) for THALES Airborne Systems. In THALES Optronics, he has worked on methods to improve the algorithm architecture adequation for image processing and on parallel architectures to implement intensive computing algorithms. In 1996, he received the PhD degree from Ecole des Mines de Paris on dedicated architectures for image processing specifically Mathematical Morphology segmentation.