Expert Systems With Applications 45 (2016) 385–399

Contents lists available at ScienceDirect

Expert Systems With Applications

journal homepage: www.elsevier.com/locate/eswa

Multi-objective kernel mapping and scheduling for morphable many-core architectures

Nuno Neves [a,c], Rui Neves [b,c], Nuno Horta [b,c], Pedro Tomás [a,c], Nuno Roma [a,c,*]

[a] INESC-ID, Rua Alves Redol, 9, 1000–029 Lisboa, Portugal
[b] Instituto de Telecomunicações, Av. Rovisco Pais, 1, 1049–001 Lisboa, Portugal
[c] Instituto Superior Técnico, Universidade de Lisboa, Av. Rovisco Pais, 1, 1049–001 Lisboa, Portugal

ARTICLE INFO

Keywords:
Optimization
Design methodologies
Multi-core
Reconfigurable architectures
Run-time and dynamic reconfiguration
Energy efficiency

ABSTRACT

A new optimization framework to maximize the performance and efficiency of morphable many-core accelerators is proposed. The devised methodology supports the co-existence of multiple optimization goals and constraints (e.g., computational performance, power, energy consumption and runtime reconfiguration overhead) by relying on a design space exploration approach based on a convenient adaptation of a Multi-Objective Evolutionary Algorithm. In accordance, the proposed algorithm allows the generation of a comprehensive set of execution plans, specifically targeting an efficient runtime adaptation of the processing elements instantiated in morphable slots of the processing structure. The conducted experimental evaluation shows significant gains in terms of the attained performance and energy efficiency when considering both highly parallel and data dependent applications, achieving peak power dissipation and energy consumption reductions as high as 54% and 45%, respectively.

1. Introduction

The increasing demand for computational processing power that has been observed along the last decade has driven the development of heterogeneous systems, typically composed of a host General Purpose Processor (GPP) and one or more accelerating devices, such as Graphical Processing Units (GPUs), Field-Programmable Gate Arrays (FPGAs) or Application Specific Integrated Circuits (ASICs), each typically integrating multiple processing elements (PEs). While these systems already allow for significant application acceleration, it is of fundamental importance to improve the offered processing efficiency, while complying with the strict power and energy constraints of both embedded and high performance computing systems. In particular, although it has been widely recognized that energy consumption can lead to important constraints in the processing performance of mobile computing platforms, the observed divergence between device-level energy-efficiency gains and transistor-density (Esmaeilzadeh, Blem, St Amant, Sankaralingam, & Burger, 2011) may require that, in future computing systems, some of the transistors must remain dimmed or powered down most of the time. A solution for such a problem relies on the exploitation of runtime dynamic morphing of processor architectures, by clock/power gating unused hardware resources, by applying Dynamic Voltage and Frequency Scaling (DVFS) techniques or by relying on fine- or coarse-grained hardware reconfiguration (Petrica, Izraelevitz, Albonesi, & Shoemaker, 2013). In particular, by dynamically adapting the PEs architectures to the application kernels and by adopting a carefully management of the available processing resources, not only in terms of its execution, but also in terms of its energy requirements, it is possible to attain high computing performance and energy efficiency (Venkatesh et al., 2011).

However, while several programming frameworks have already been developed with the specific purpose of efficiently exploiting GPUs for general purpose computation (CUDA/OpenCL), morphable hardware accelerators have not received so much attention, specially those deployed on reconfigurable technology. In fact, although some existing frameworks (e.g. Xilinx Vivado, Altera SDK for OpenCL, Maxeler Technologies' MaxCompiler) already provide the means for translating kernels into low-level hardware implementations and for mapping them into FPGAs, the ability for runtime morphing the architectures is usually not taken into account. As a consequence, the development of a new framework capable of efficiently and dynamically mapping and scheduling an application's kernels into the PEs of a morphable accelerator is still highly required.

The herein proposed framework tackles the described problem by incorporating a design space exploration (DSE) tool to derive, in compile-time, a set of execution plans that describe not only an

* Corresponding author at: ECE Department, INESC-ID, IST, Universidade de Lisboa, Rua Alves Redol, 9, 1000–029 Lisboa, Portugal. Tel.: +351213100311.
E-mail addresses: Nuno.Neves@inesc-id.pt (N. Neves), Rui.Neves@lx.it.pt (R. Neves), Nuno.Horta@lx.it.pt (N. Horta), Pedro.Tomas@inesc-id.pt (P. Tomás), Nuno.Roma@inesc-id.pt (N. Roma).

http://dx.doi.org/10.1016/j.eswa.2015.10.004
0957-4174/

efficient mapping of tasks to PEs, but also the scheduling of those mappings to morphable processing hardware. Furthermore, by taking advantage of a multi-objective optimization (MOO) technique, it is possible to generate multiple execution plans, each establishing a different compromise between the application's performance, system power consumption and energy efficiency. Hence, by carefully selecting (in runtime) the most appropriate execution plan and by considering the time and energy overheads resulting from the communication and the reconfiguration/adaptation process, it is possible to make an application execution as adaptive and energy-efficient as possible.

To validate the proposed algorithm and scheduling methodologies, they were conveniently integrated into a previously developed morphable many-core accelerator, managed and controlled by an integrated Hypervisor module. The conducted experimental evaluation in a reconfigurable hardware scenario shows that the proposed multi-objective optimization approach provides significant energy savings under a set of both static and dynamic application workloads, reaching gains as high as 45%.

The remaining of the manuscript is organized as follows: Section 2 summarizes the background and the related work and enumerates the contributions of the herein proposed approach; Section 3 introduces a generic model of the considered processing platform composed by multiple reconfigurable/adaptable accelerators, together with the presentation of its underlying application specifications; Section 4 describes the proposed DSE algorithm, including the definition of the optimization goals and all of the intervening operators included in the adopted Objective Evolutionary Algorithm (MOEA); the algorithm is then validated in Section 5, by considering two case studies; finally, Section 6 concludes the manuscript by discussing and addressing the main contributions and achievements.

2. Related work

The new optimization framework that is herein proposed represents a considerable extension of other MOO design approaches that have been presented in the literature to optimize highly heterogeneous many-core processing systems. Some examples of related contributions comprise: code optimization (Balaprakash, Tiwari, & Wild, 2014; Neugebauer, Marwedel, & Engel, 2015); static and dynamic task scheduling (Behnamian, Zandieh, & Ghomi, 2009; Camelo, Donoso, & Castro, 2011; Cheng, Shiau, Huang, & Lin, 2009; Daoud & Kharma, 2011; Sheikh & Ahmad, 2013; Xu, Li, Hu, & Li, 2014); and a number of DSE approaches based on high-level and system-level synthesis (Erbas, Cerav-Erbas, & Pimentel, 2006; Gschwandtner, Durillo, & Fahringer, 2014; Holzer, Knerr, & Rupp, 2007; Krishnan & Katkoori, 2006), application mapping in multi-processor systems (Mariani et al., 2010; Palermo, Silvano, & Zaccaria, 2009) and heterogeneous systems (Erbas, Erbas, & Pimentel, 2003), as well as routing and communication topology optimizations (Glaß, Lukasiewycz, Wanka, Haubelt, & Teich, 2008). However, although some of these solutions already target morphable processing architectures, they do not take advantage of the runtime reconfiguration/adaptation capabilities of the underlying hardware support.

On the other hand, some DSE algorithms have been proposed to specifically target dynamically reconfigurable platforms. In Miramond and Delosme (2005), it is proposed a tool that defines different contexts for reconfigurable circuits, which are dynamically switched in runtime through partial reconfiguration. The underlying tasks are then assigned to each configuration through spatial and temporal partitioning, by using a local search algorithm. In Czarnecki and Deniziak (2008), it is proposed the usage of conditional task graphs, allowing to model mutually exclusive tasks. According to this algorithm, all tasks are initially assigned to only one GPP module and new solutions are produced using iterative improvement methods. Also based on conditional task graphs, the Evolutionary Algorithm (EA) proposed in Shang and Jha (2002) is used to determine the amount of used resources and to assign tasks to PEs, followed by a re-mapping and scheduling algorithm that makes use of the dynamic reconfiguration capabilities of FPGAs. Another algorithm targeting multi-mode systems is proposed in Wildermann, Reimann, Ziener, and Teich (2011), where it is assumed that the different operating modes can share the same hardware resources by means of partial reconfiguration. A symbolic encoding is also proposed, by combining a SAT solver and an MOEA, allowing the system synthesis for allocation, binding and placement of partially reconfigurable modules, both temporally and spatially.

Although the above referred frameworks and algorithms already take advantage of dynamic reconfiguration, they are mostly focused in HW/SW co-synthesis and HLS problems. In contrast, the herein proposed approach considers CPU-coupled morphable heterogeneous accelerators integrating a number of predefined reconfigurable/adaptable regions, allowing the dynamic morphing of their PE architectures according to energy and runtime execution requirements.

2.1. Background

A rather preliminary approach to the herein proposed optimization strategy was initially considered in the morphable many-core acceleration platform proposed in Neves, Mendes, Chaves, Tomás, and Roma (2015a). Such adaptive processing structures is managed by an Hypervisor module, implemented either in the host computer or locally in the accelerator device. The Hypervisor is responsible for permanently monitoring the accelerator's PEs execution in order to define convenient scheduling decisions (in real-time), and to issue appropriate reconfiguration commands that adapt the architecture of each individual morphable region to the instantaneous characteristics and requirements of the application under execution. Moreover, the devised Hypervisor module can also deploy runtime optimization policies to further promote the performance and energy efficiency, thus guaranteeing an extended battery life and/or minimum power consumption (e.g. in mobile computing platforms), or minimum energy costs in high performance computing clusters. Such devised optimization policies are based on intrinsic and/or external requirements that may demand the system to reduce the power consumption in runtime (e.g. by turning off processing cores) or to ensure a minimum and stationary performance level, while complying with strict peak power or energy constraints. Nevertheless, such an approach incurs in significant overheads that affect the overall efficiency of the platform, since the Hypervisor is required to perform a significant amount of operations in runtime, namely: (i) read the performance counters of the PEs; (ii) process the obtained information through the set of optimization policies; and (iii) schedule and trigger the appropriate reconfiguration commands to the accelerator.

2.2. Contributions

The optimization methodology that is now proposed complements the Hypervisor-based platform proposed in Neves et al. (2015a) with a comprehensive set of DSE optimization tools, executed at compile time. These tools provide a new abstraction level of the underlying morphable processing structures, particularly fitted to the application kernels being migrated from the CPU. With such an approach, the Hypervisor can be provided with further information about the application behavior and requirements, thus mitigating the implicit overheads, and even allowing the anticipation of the required reconfigurations and scheduling decisions, thus hiding the adaptation procedure behind the application execution.

Hence, the main contributions of the proposed DSE framework can be summarized as follows:

- Definition of a generic system specification model for morphable many-core accelerators, specially devised to support the modeling of the runtime adaptation of the processing and data-communication structures, comprising both application kernel and architecture models, and related through system and application-specific mapping constraints.
- Specification of a multi-constraint multi-objective DSE problem targeting the optimization of the execution and of the runtime reconfiguration procedure of a morphable many-core accelerator (regarding execution latency, peak power dissipation and energy consumption) for each running application.
- Extension of the well-known Non-dominated Sorting Genetic Algorithm II (NSGA-II; Deb, Pratap, Agarwal, and Meyarivan, 2002) (specially suited and commonly used in multiple constraint problems) for the deployment of a novel DSE algorithm that generates the mappings between the executed kernels and the different PE architectures that may be implemented in each reconfigurable/adaptable accelerator. Contrasting with the existing solutions, the adopted methodology includes the adaptability of the accelerators architecture in the optimization procedure and considers the runtime reconfiguration latency and power dissipation overheads.
- Definition of a set of adaptive execution plans (properly optimized in terms of the resulting processing time, peak power and energy consumption) that are conveniently prepared to be subsequently provided to an accelerator's adaptation and task scheduling engine. Such execution plans are used to bridge the proposed compile-time framework with the existing morphable many-core acceleration platform (Neves et al., 2015a), and are shown to greatly improve the control and management routines of the platform's Hypervisor, as well as the resulting overall system efficiency, by eliminating most runtime scheduling and reconfiguration/adaptation process decision overheads.

In order to better demonstrate the contributions of the proposed framework, Section 5.4 will present a comparative discussion with the covered related work and state-of-the-art. Such discussion will also consider the application of the proposed strategies in ASIC technologies, by relying on clock/power gating and DVFS techniques.

3. Morphable heterogeneous framework

The proposed framework targets generic heterogeneous systems comprising morphable and scalable many-core accelerators (based either on FPGA or ASIC devices) tightly coupled with a host CPU, as illustrated in Fig. 1. To ease the description, the presentation that follows adopts the same prototyping environment proposed in Neves et al. (2015a) based on FPGA technologies, by extensively exploiting their reconfigurable characteristics. Nevertheless, similar processing structures may also be implemented in ASIC devices, by exploiting power/clock gating methodologies (allied with DVFS) to dynamically adapt the processing structures. Besides the morphing processing structures, the proposed framework also comprises a comprehensive set of optimization strategies to define the implicit mapping and reconfiguration procedures.

3.1. Morphable many-core processing platform

In order to keep the description of the proposed technique as generic as possible, some assumptions shall be made concerning the accelerator topology. Accordingly, any component of the system that can be changed in runtime is assumed to be a morphable slot. Such slots not only will support the implementation of the processing clusters but also of the interconnection network between them, shared memory controllers and convenient first-in-first-out (FIFO) interfaces that chain adjacent clusters (see Fig. 1). To keep up with those changes, the static part of the accelerator was assumed to maintain the necessary interfaces to all the described components. It is composed of the communication network that connects the Hypervisor and the host CPU to the several processing clusters, as well as all the required interfaces to all the morphable slots. Hence, while all these components may be mapped on reconfigurable logic (when FPGAs are used), they cannot be changed by the Hypervisor.

3.2. Adaptive execution plan

Besides deploying the previously referred optimization policies, the Hypervisor is also able to use the set of optimized *adaptive execution plans*, obtained at compile-time with a DSE algorithm, to efficiently execute the underlying application. Such execution plans comprise a time-ordered execution queue containing (see Fig. 2): (i) the mappings of the several operations or kernels (herein defined

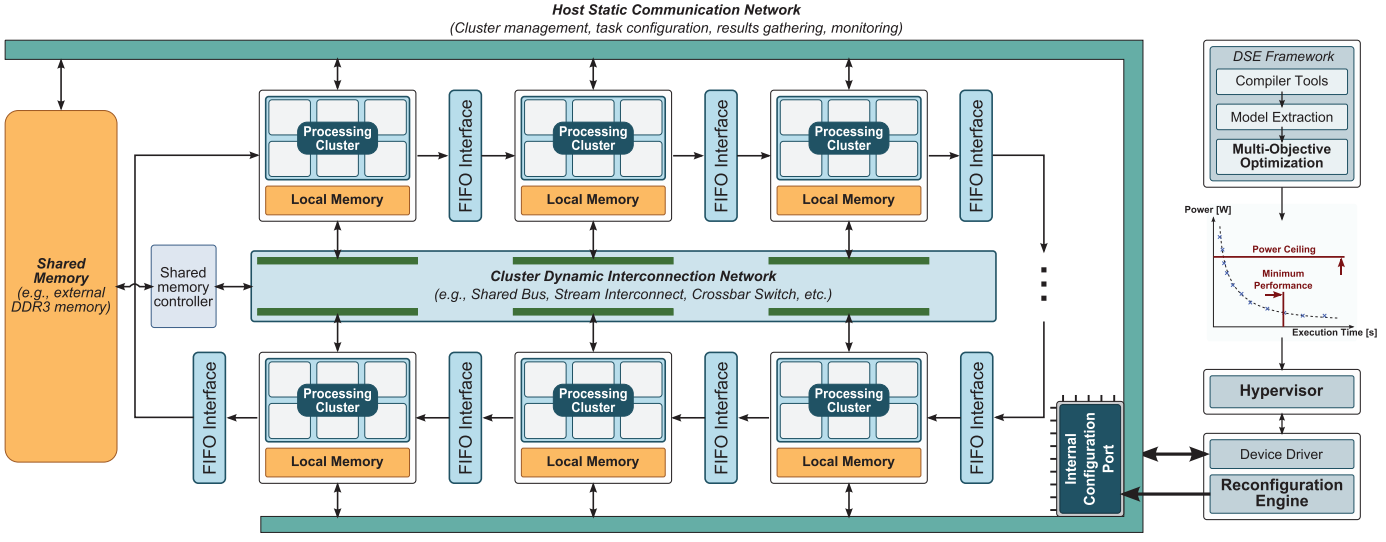


Fig. 1. Conceptual representation of the target morphable heterogeneous architectures.

TASK	ARCH	SLOT	REC	TIME
⋮				
7	4	1	no	5
2	3	2	no	5
5	2	3	no	5
4	1	2	yes	13
3	1	1	yes	27
6	2	3	no	30
⋮				

Fig. 2. Execution plan example. If two tasks (TASK column) are mapped to different components (ARCH column) and subsequently assigned to the same morphable slot (SLOT column), a reconfiguration must be triggered (REC column) before the second task begins.

as tasks) to the different PE architectures of the accelerator; and (ii) the assignments of those mappings and PE architectures to the available set of morphable slots, together with any required reconfiguration commands. This way, an execution plan can be represented as a list of tuples (TASK,ARCH,SLOT,REC,TIME), where TASK is a task identification, ARCH is the mapped architecture identification, SLOT is the morphable slot for that architecture, REC indicates whether reconfiguration is required prior to the task execution and TIME is the assigned starting time.

To take advantage of the obtained execution plans, the Hypervisor operates in one of two modes. On the *Dispatcher* mode, the Hypervisor simply follows a given execution plan, issuing the appropriate tasks and reconfiguration commands to the accelerator. This way, the Hypervisor will switch between execution plans, according to runtime conditions that affect the power and energy consumption, and/or the processing throughput, in order keep the execution as efficient as possible. On the *Dynamic Scheduler* mode, the Hypervisor extracts (from the execution plan) the mappings between tasks and the appropriate architectures, and dynamically schedules the tasks' execution and reconfiguration commands during runtime. Moreover, by monitoring the amount of data flowing between the PEs, it switches between execution plans in real-time, in order to balance the data flowing in and out of dependent tasks, ensuring system adaptation to applications with dynamic workload variations.

3.3. System specification model

The proposed DSE algorithm makes use of a formal representation of both the underlying application and of the target processing structure. In particular, it adopts a general system specification model (Erbas et al., 2006) for the representation of the data flow between the several application kernels and the hardware components, in order to define a new representation that also supports morphable systems.

3.3.1. Application model definition

A data-flow graph is used to represent the application kernels and their interactions (Huang, Sethuraman, & Chapman, 2008; Stefanov, Zissulescu, Turjan, Kienhuis, & Deprette, 2004). In particular, it is assumed that a pre-processing framework module elaborates such a graph for each application kernel, extracting its inherent characteristics, which will be used as a starting point for the herein defined model.

In a first step, the proposed framework must be provided with the set of different types of supported operations (including communication). Such operations can range from fine-grained, such as integer multiplications or floating-point additions, to coarse-grained operations, such as entire functions or kernels. This way, each task, t_i, is represented by:

n_k—Number of operations (or kernels) of type-k.
d_s—Size of its assigned data chunk.

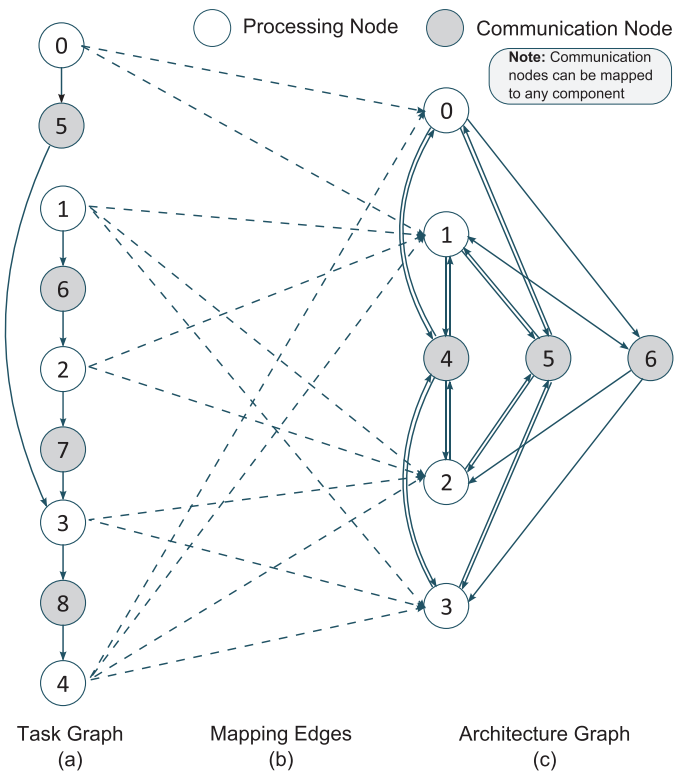


Fig. 3. Example of a system specification comprehending the task (a) and the architecture (c) graphs, together with the set of mapping edges (b) between them. The values inside each node of the graphs correspond to unique identifications for each task (a) and component (c) of the system.

Based on this assumption, the application is represented by a data-flow task graph, $G(V_T, E_T)$, composed of a set of tasks V_T. This set comprises both *processing tasks* ($V_T^p \subset V_T$) and *communication tasks* ($V_T^c \subset V_T$). The task graph also comprises the set of edges, E_T and the data flow between them. Each edge is given by the tuple $e = (t_i, t_j)$, with $t_i, t_j \in V_T$. An example task graph is shown in Fig. 3(a).

3.3.2. Architecture model definition

A similar formalization is applied to the definition of the architecture model. This way, an architecture is represented by a data-flow architecture graph, $G(V_C, E_C)$, composed of a set of components, V_C (comprising both *processing components*, $V_C^p \subset V_C$, and *communication channels*, $V_C^c \subset V_C$), and the set of edges, E_C, that represent the data flow between them. Each edge is given by the tuple $e = (c_i, c_j)$, with $c_i, c_j \in V_C$. Every component, c_i, represents a morphable module and is defined by:

ca_i — component architecture identification;
s_i — morphable slot used by the component;
l_{i_k} — Execution latency for operations (or kernels) of type k in the component (-1 if the type of operation cannot be executed).
ls_i — Communication setup overhead (only for communication channels).
lr_i — Component reconfiguration/adaptation time.
p_i — Power consumption of the component.
pr_i — Reconfiguration/adaptation power consumption.

The latency of each communication channel, i, is given by adding its setup overhead (ls_i) to the delay of transferring a single data block (l_{ki}) times the amount of transferred data.

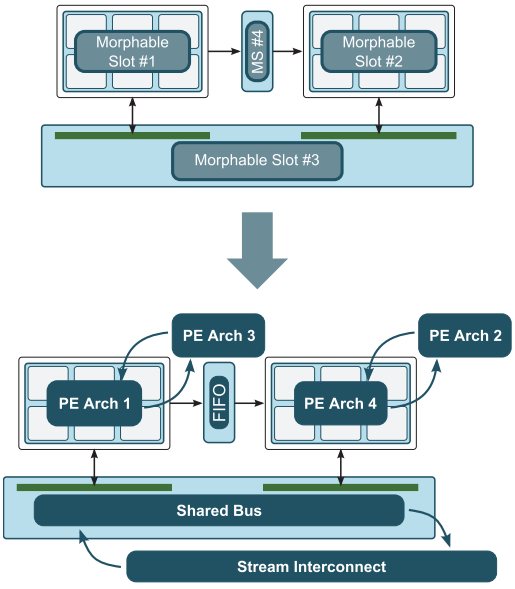


Fig. 4. Example of an architecture model, represented by both the location and organization of the morphable slots and by the possible component assignments to each of those slots. Each morphable slot can accommodate only one architecture at the time. Each component is given an unique numeric identification, later used for the architecture data-flow graph (see Fig 3(c)).

One of the main challenges posed by the representation of such a morphable system is concerned with the definition of a formal model that properly represents the adaptivity of the system. For such purpose, each of the available reconfigurable regions is assumed as a morphable slot for a set of components supported by the system. This way, each component may be replicated in all its compatible slots in the model, and each replication is assumed as a different component. Hence, if two consecutive tasks are mapped to two different components in the same slot, a reconfiguration command must be triggered in between, as it will be later described. By default, two types of slots are considered: (i) PE or cluster of PE slots, reserved for processing components; and (ii) slots for the communication channels. Fig. 4 depicts an example architecture model, corresponding to the same system specification previously presented in Fig. 3(c).

3.3.3. *Mapping constraints*

Having defined both the application and the architecture models, a set of mapping edges that represent each possible mapping of each task to all the components of the system is created, by simultaneously parsing both graphs. For each pair of processing task t_i and processing component c_j, a mapping edge m_{ij} is created if that component can execute all types of operations of that task. In particular, if two consecutive processing tasks are mapped to the same component, the communication task between them can also be mapped to that same component with zero latency (detailed below). A typical example of such case corresponds to data stored in a local cache or a scratch-pad memory element. A full system specification for the previously referred task and architecture models is shown in Fig. 3.

4. Design space exploration methodology

Real MOO problems are characterized by a set of conflicting objective functions, to which it is not possible to find a single solution that simultaneously minimizes (or maximizes) all the objectives. The resulting set of solutions, usually denoted as the *Pareto optimal* front, contains the points representing the best trade-off solutions in the objective space. Each of these best compromise solutions are not dominated by any other solution in the objective space (Deb, 2008).

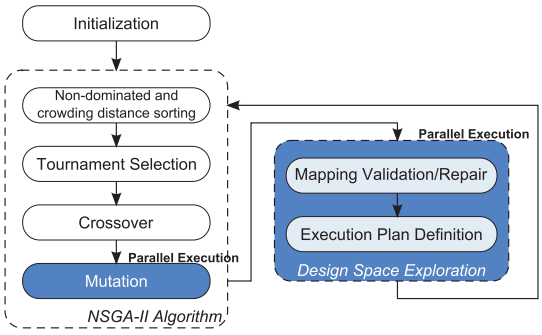


Fig. 5. Proposed NSGA-II extension (highlighted blocks) for DSE.

A wide variety of methodologies have been proposed to solve MOO problems (Deb, 2008). Among the most used for solving DSE problems are the MOEAs, since these algorithms are particularly well-suited for simultaneously dealing with solutions in large sets. Moreover, each DSE problem can be regarded as a highly constrained MOO problem (Deb, 2008), since a vast number of mapping and architectural constraints are inherently imposed. Hence, by applying genetic evolutionary approaches, where potential solutions are encoded as chromosomes in individuals of a given population, it is possible to obtain the set of *non-dominated* solutions by evolving the population through a number of generations. The population, itself, is evolved by applying operations based on nature's evolution mechanisms, such as selection, reproduction and mutation.

4.1. *Algorithm*

The proposed DSE methodology was built by abstracting its underlying MOEA, allowing it to be easily modified in order to use different and alternative algorithms. In fact, although the most commonly used MOEAs are very-well suited for MOO problems, different algorithms may be better suited and can be easily used, depending on the properties of the search space (such as the dimensionality and the shape of the Pareto optimal front).

One of the most well established MOEAs is the NSGA-II (Deb et al., 2002), characterized by a very fast convergence and a very good spread of solutions near the Pareto optimal front. This is achieved by including a fast non-dominated sorting approach, followed by a selection operator that, together, create a mating pool by combining the parent and offspring populations and by subsequently selecting the best (with respect to fitness and spread). Furthermore, the NSGA-II algorithm is specially well-suited for solving problems with multiple constraints (Deb et al., 2002), as it is the case of the targeted DSE problem.

By taking the NSGA-II as the core algorithm for the proposed DSE methodology, a new encoding of the solution space is proposed, based on the information collected from the system specification models defined in Section 3.3. Appropriate crossover and mutation operators have also been devised according to the proposed encoding, as well as convenient *mapping validation/repair* and *execution plan definition* algorithms (see the corresponding flowchart in Fig. 5). This *mapping validation/repair* algorithm is used to decode, validate and repair the set of solutions generated by the NSGA-II algorithm. Conversely, the *execution plan definition* algorithm is used not only to generate the time-ordered task and reconfiguration command execution queues (see Fig. 2), corresponding to the generated solutions, but also to calculate their corresponding objective functions. The resulting function values are then sent back to the selection operator of the NSGA-II algorithm for evaluation, thus closing the optimization loop.

In order to accelerate the convergence of this optimization algorithm, its underlying parallelism is fully exploited by adopting a multi-threaded implementation to significantly speedup its

execution. Such concurrency is made available since some operations process each individual independently. In particular, the mutation, mapping validation/repair and execution plan definition operators can fully exploit parallelism, while the sorting, selection and crossover operators require interactions between individuals.

4.2. Problem definition

As previously referred, the main goal of the proposed DSE methodology is to optimally map the application kernels to a given morphable accelerator, implemented in an FPGA or ASIC device (as described in Section 3). in accordance, the implicit DSE problem should be optimized for *latency, peak power dissipation* and *total energy consumption*, and these three optimization goals are defined as the aimed objective functions of the considered algorithm. Hence, it should be possible to obtain a set of optimized execution plans for the considered application kernels that represent the best possible trade-offs for these three goals.

For each individual (p) from a given population (P_t) in its tth generation, the DSE problem is defined as the multi-constraint multi-objective problem:

$$\min_{p \in P_t} f_p = (f_{L_p}, f_{P_p}, f_{E_p}), \quad \text{subject to (6)–(9)} \tag{1}$$

The objective functions are formally defined according to the system specification models and variables, as described in Section 3.3. The first objective function (f_L), targeting the minimization of the system latency (in execution cycles), is given by the finishing time of all the enqueued tasks:

$$f_L = \max_{t_i \in V_T} (s_{t_i} + l_{t_i}), \tag{2}$$

where the finishing time of a given task ($t_i \in V_T$) is obtained by adding its starting time (s_{t_i}) and its latency (l_{t_i}). Moreover, each task latency is computed by considering its assigned data chunk size (d_s), its respective number of operations (n_k) and the latency (l_{j_k}) of each operation on its mapping component (m_{ij}):

$$l_{t_i} = d_s \times \left(\sum_k l_{j_k} \times n_k \right) \times m_{ij}, \quad \text{with} \quad t_i \in V_T, \quad t_j \in V_C \tag{3}$$

For the other two objective functions, a time frame (t) is defined to represent a period of time (with duration l_t), where no changes occur in the architecture and in the execution state of the accelerator's PEs. The set of time frames corresponding to the total execution time of a given solution is given by T. Hence, the second objective function (f_P) minimizes the peak power dissipation (in Watts) at any given time frame:

$$f_P = \max_{t \in T} \sum_{t_i \in V_C} (p_i + pr_i \times r_i) \times a_i \, . \tag{4}$$

It is calculated based on the current allocation of each component (a_i), its power dissipation (p_i) and its reconfiguration/adaptation power consumption (pr_i), in case such reconfiguration occurs (r_i). The a_i and r_i variables take a binary value (0 or 1) indicating whether a component is allocated and whether reconfiguration/adaptation is required, respectively.

Finally, the third objective function (f_E) minimizes the total energy consumption (in Joules):

$$f_E = \sum_{t \in T} \left(\sum_{t_i \in V_C} (p_i + pr_i \times r_i) \times a_i \right) \times l_t, \tag{5}$$

The consumed energy in each time frame (t) is calculated by adding the total instantaneous power dissipation of the system at that time and multiplying it by the duration of the time frame (l_t). Finally, the energy consumption corresponding to all time frames is added up.

To ensure that the DSE algorithm generates a set of feasible solutions representing possible mappings of tasks to possible architectures, it must be defined as a constrained problem. Such constraints are herein enumerated:

1. Each processing task must be mapped to a single processing component:

$$\sum_{a \in V_C^p \subset V_C} m_{ai} = 1, \quad \text{for each} \quad t_i \in V_T^p \subset V_T \tag{6}$$

2. Each communication task must be mapped either to a processing component or to a communication channel:

$$\sum_{a \in V_C^p \subset V_C} m_{ai} + \sum_{b \in V_C^c \subset V_C} m_{bi} = 1, \quad \text{for each} \quad t_i \in V_T^c \subset V_T \tag{7}$$

3. When two adjacent processing tasks are mapped to the same component, the communication task between them must be mapped to that same component:

$$\begin{aligned} &\text{If} \quad m_{ai} = 1 \quad \text{and} \quad m_{aj} = 1 \quad \text{and} \quad \exists\, t_k \in V_T \\ &\text{such that} \quad \exists\, e_x = (t_i, t_k), \quad e_y = (t_k, t_j) \in E_T, \\ &\qquad\qquad \text{then} \quad m_{ak} = 1, \\ &\qquad \text{where} \quad t_i, t_j \in V_T \quad \text{and} \quad a \in V_C \end{aligned} \tag{8}$$

4. When two adjacent processing tasks are mapped to separate components, the communication task between them must be mapped to a communication channel that connects both components:

$$\begin{aligned} &\text{Let} \quad t_k \in V_T\text{: } \exists\, e_x = (t_i, t_k), \quad e_y = (t_k, t_j) \in E_T, \\ &\quad \text{if} \quad m_{ai} = 1 \quad \text{and} \quad m_{bj} = 1, \quad \text{then} \quad m_{ck} = 1, \\ &\text{iff} \quad \exists\, c_c \in V_C\text{: } \exists\, e_w = (c_a, c_c), \quad e_z = (c_c, c_b) \in E_C, \\ &\qquad\qquad \text{where} \quad c_a, c_b \in V_C \quad \text{and} \quad t_i, t_j \in V_T \end{aligned} \tag{9}$$

The last constraint refers to the actual values of the objective functions. These values are only required to be non-negative, since the system specification model maximizes the value of each function, given the tasks' latency, component area and power characteristics.

4.3. Individual encoding

Having defined the DSE problem, the next required step is to encode the solutions of the problem in individuals of a given population. As it is common practice in genetic algorithms (Deb, 2008), the encoding of the solutions is referred to as a *chromosome*, unique to each individual, randomly generated and modified by the algorithm. Each chromosome is posteriorly decoded and evaluated, according to a set of problem related operators.

In the considered DSE problem, the solutions represent possible mappings of tasks to PEs and communication channels, according to the set of constraints defined above. This way, each solution may be simply represented as an array, where each index represents a different task and the corresponding value depicts the mapping of that task to a given component. However, such a simple representation leads to a great number of infeasible mappings (e.g. mapping of a task to a component that cannot execute all its operations). Hence, instead of a single value, each cell of this array comprehends a list that contains all the feasible mappings of its corresponding task, being the head of that list the current mapping (see Fig. 6). Accordingly, the head of the list is denoted as *gene*, while the rest of the list is named *gene repair list*. Hence, although very rarely and greatly depending on the system architecture, the only situation where an infeasible solution can occur is whenever two communicating tasks are mapped to two components that have no communication channels between them. While such situation is not a common architectural design, it may occur, specially in hierarchical architectures.

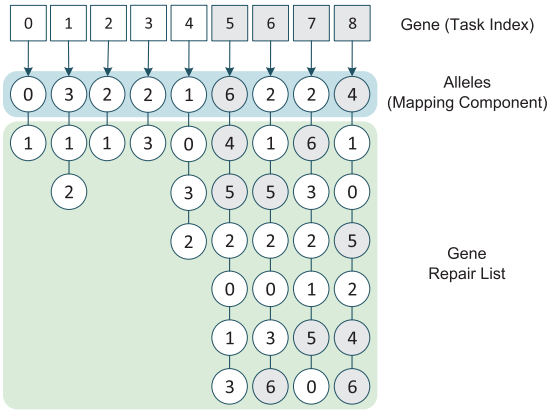


Fig. 6. Individual encoding, representing a possible solution for the specification defined in Fig. 3.

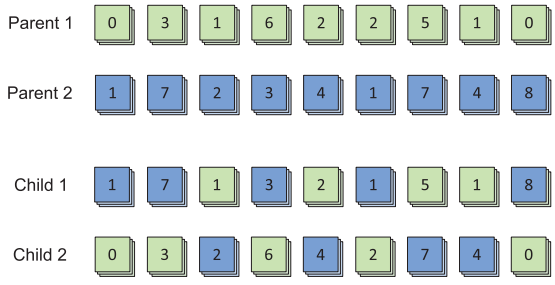


Fig. 7. Multi-point crossover operator.

4.4. Genetic operators

Since a custom encoding was specifically defined for the considered DSE algorithm, the genetic operators, i.e. *crossover* and *mutation*, must also be defined accordingly.

4.4.1. Crossover

This operator is based on the multi-point crossover method, characterized by providing the best diversity among the offspring population (Whitley & Sutton, 2012). Usually, this operator randomly selects which genes of the chromosome array of two parent individuals are cross-copied to two newly created child individuals, as shown in Fig. 7. However, since each element of the devised chromosome is also composed of a repair list, instead of only copying the gene, both are copied to the new individual. This ensures the feasibility property mentioned above.

4.4.2. Mutation

Most existing encodings (Pilato, Loiacono, Ferrandi, Lanzi, & Sciuto, 2008) that rely on repair lists to maintain feasible results either require several mutation and repair steps, or do not entirely exploit the diversity imposed by the operator (Erbas et al., 2003). To overcome both issues, a two-step mutation operator was devised (see Fig. 8). Such operator, denoted as *Swap-Scramble* mutation, starts by first swapping the gene value with a random value from the repair list. Next, a subset of the repair list is randomly selected and the values of that subset are randomly re-arranged, or scrambled. This way, no additional procedures are required and more diversity is achieved in the population.

4.5. DSE operators

To complete the optimization loop, the *mapping validation/repair* and the *execution plan definition* DSE operators are defined to evaluate

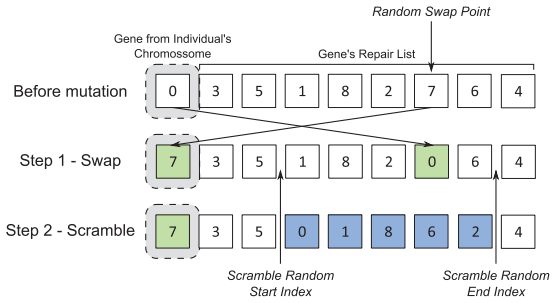


Fig. 8. Two-step *Swap-Scramble* mutation operator.

and repair the set of solutions obtained from NSGA-II and to generate the resulting execution plans.

4.5.1. Mapping validation/repair operator

Given the devised solution encoding, constraints (6) and (7) are implicitly verified. In accordance, only communication-related infeasible mappings may occur, i.e. if constraints (8) or (9) are not satisfied. Instead of immediately discarding such solutions, a repair algorithm (Algorithm 1) that takes advantage of the considered gene repair list was also devised. Whenever constraint (8) is violated, the algorithm immediately maps the communication task to the component to which its source and sink tasks are mapped to. On the other hand, when constraint (9) is violated, the algorithm tries to find (and map) an alternative component that will connect the components. If such a component is found, the algorithm immediately maps the communication task to it; otherwise, the solution is deemed unfeasible, and subsequently discarded.

Algorithm 1 Mapping validation/repair operator.

Input: $mappings[] \Rightarrow$ chromosome, $SSpec \Rightarrow$ system specification
Variables: $tc \Rightarrow$ communication task, $tsrc \Rightarrow$ source task, $tsk \Rightarrow$ sink task, $m(t) \Rightarrow$ mapping of task t
Output: $feasibleMapping$ (`true`/`false`)

```
1:  feasibleMapping ← true
2:  for all communication tasks do
3:    if m(tsrc) = m(tsk) = m(tc) then
4:      continue
5:    else if m(tsrc) = m(tsk) ≠ m(tc) then
6:      m(tc) ← m(tsrc)
7:    else if m(tsrc) ≠ m(tsk) then
8:      search component that connects m(tsrc) and m(tsk) and is
        present in tc repair list
9:      if component found then
10:       m(tc) ← component
11:     else
12:       feasibleMapping ← false
13:       return
14:     end if
15:   end if
16: end for
17: return
```

4.5.2. Execution plan definition operator

The final step of the algorithm comprehends the definition of the execution plan. This operator (see Algorithm 2) serves two purposes: (i) generation of the execution queue for the current solution; and (ii) calculation of the three objective functions. The execution plan definition is based on a common iterative scheduling procedure. By following the devised task graph, an ordered priority list is

Algorithm 2 Execution plan definition operator.

Input: *mappings*[] ⇒ chromosome, *SSpec* ⇒ system specification
Variables: *alloc*[] ⇒ slot configuration, *state*[] ⇒ slot state, *candidates*[] ⇒ candidate priority list, *inProgress*[] ⇒ running tasks, *tk.map* ⇒ mapping of a task
Output: *queue*[], *peak_power*, *latency*, *t_energy*

1: initialize *queue*[] based on *mappings*[] and *SSpec*
2: *candidates*[] ← input tasks
3: *peak_power*, *t_energy* ← 0
4: *time* ← 1
5: **repeat**
6: **for each** task *tk* in *candidates*[] **do**
7: *ready* ← 1
8: **for each** *source* in *tk* **do**
9: **if** *source.start* ≠ −1 **and** *source.start* + *source.latency* < *time* **then**
10: **continue**
11: **else**
12: *ready* ← 0; **break**
13: **end if**
14: **end for**
15: **if** *ready* = 1 **and** *state*[*tk.map.slot*] = 0 **then**
16: **if** *alloc*[*tk.map.slot*] ≠ *tk.map.arch* **then**
17: *tk.latency* ← *tk.latency* + *tk.map.rec_time*
18: *tk.power* ← *tk.power* + *tk.map.rec_power*
19: **end if**
20: *alloc*[*tk.map.slot*] ← *tk.map.arch*
21: *state*[*tk.map.slot*] ← 1
22: *tk.start* ← *time*
23: *inProgress*[] ← *tk* (in order of finish time)
24: *queue*[] ← *tk*
25: add all *tk.edges*[] to *candidates*[]
26: **end if**
27: **end for**
28: add up *power* for all *tk* in *inProgress*[]
29: *peak_power* ← *max*(*peak_power*, *power*)
30: *tk* ← *pop*(*inProgress*[])
31: *ntime* ← *tk.finish*
32: *state*[*tk.map.slot*] ← 0
33: *t_energy* ← *t_energy* + *power* × (*ntime* − *time*)
34: *time* ← *ntime*
35: **until** all tasks enqueued
36: *latency* ← *time*
37: **return** *queue*[]

kept, including all candidate tasks that are waiting to be initiated. Accordingly, a given task can be started if the morphable slot of its mapping component is not in use. As soon as the slot becomes free, the architecture of the mapped component is checked against the former component previously allocated to that slot. In case the architectures are different, a reconfiguration command must be triggered before the task can be started. Furthermore, every time there is a change in the configuration of the system (i.e. a reconfiguration or a component's execution starts or finishes), its total power dissipation is calculated, as well as the time period while it stays constant. This way, the peak power dissipation can be obtained by comparing those measures and by choosing the minimum. The total energy consumption is calculated by adding the products of the power measures by their corresponding durations. Finally, the computed latency corresponds to the time when the last task completes its execution.

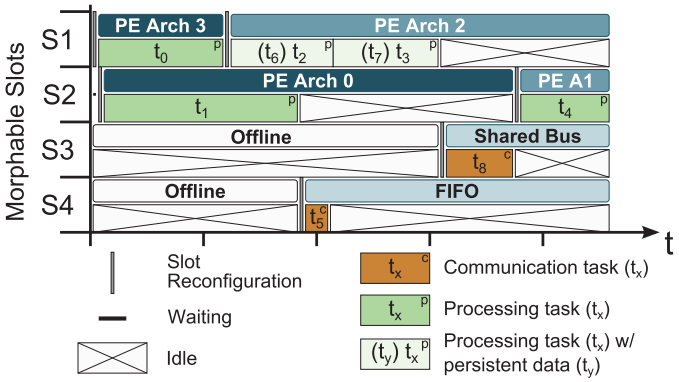


Fig. 9. Example of an execution queue, obtained with Algorithm 2 for the encoding in Fig. 6, corresponding to the mapping of the application described with the task graph in Fig. 3(a) to the architecture in Fig. 4. Note that when two adjacent tasks are mapped to the same component, the communication task between them is mapped to that same component with zero latency (shown between parenthesis in the second task's execution).

4.6. Resulting optimized execution plans

Each solution that is obtained from the considered DSE algorithm is represented by an execution plan. This plan contains the time-ordered execution queue (as shown in Fig. 9), the mappings and the required reconfiguration operations for the generated solution. Then, a post-processing routine selects a representative subset of the obtained optimal solutions (based on the system's processing requirements) and provides them to the Hypervisor in the form of execution plans. In turn, the Hypervisor uses these execution plans in one of the two possible modes, as described in Section 3.2.

On the *Dispatcher* mode, the Hypervisor strictly follows the order in the execution queue, corresponding to the current execution model, and issues the tasks to their mapping component in its corresponding morphable slot, as well as the required reconfiguration commands. In case there is a change in the execution environment, such as a low-power scenario, the Hypervisor switches the considered execution plan and continues the execution from where it left.

On the *Dynamic Scheduler* mode, instead of strictly following an execution plan, it monitors the PE throughput variations and switches between execution plans in real-time, such as to guarantee a balance between the data flowing into and out of each PE and the task execution balance according to the application dependencies. To ensure compliance with the application performance and system power and energy constraints, it dynamically selects the most appropriate execution plan from a list of feasible candidates, as provided by the DSE algorithm.

5. Experimental results

To experimentally validate the proposed DSE methodology, the architectural model of the morphable heterogeneous computing platform proposed in Neves et al. (2015a) was used as the base architecture. This platform (whose conceptual representation is shown in Fig. 1) integrates 7 processing clusters, each composed of up to 15 cores (the actual value depends on the area requirements of the cores), together with several morphable slots for the interconnection network, and various FIFO buffers between the processing clusters. To attain the aimed application performance, system peak power and energy constraints, the platform offers an extensive run-time adaptation supported on real-time monitoring of the processing cores performance (while they execute an application's kernels) and on an embedded Hypervisor that selects and ensures the implementation of the most convenient execution plan among the set of

execution plans generated by the proposed DSE algorithm. Just as in Neves et al. (2015a), an FPGA technology process was adopted for the prototype based on a Xilinx Virtex-7 FPGA (XC7VX485T). Nevertheless ASIC technology processes could equally be used by relying on convenient adaptations, as previously discussed in Section 5.3. The testing platform was run on a 6-core Intel Core i7-3930K CPU, running at 3.20GHz, with 4 × 8GB 1.6GHz DDR3 RAM memories. The characteristics of each architecture component in terms of its timing, area and power dissipations were obtained with Xilinx EDK 14.5 and the XPS Power Estimation tools.

To generate the set of execution plans, the proposed DSE algorithm was implemented in C programming language, by using POSIX threads library to attain the aimed high efficiency and portability. Moreover, the adopted genetic algorithm (NSGA-II) was configured with a population of 200 individuals for 2000 generations (corresponding to an execution time between 5 and 10 seconds) and crossover and mutation probabilities of 0.95 and 0.2, respectively. This initial configuration was based on the ranges of parameter values suggested in Deb et al. (2002) and further fine-tuned by observing the algorithm's evolution, in terms of convergence characteristics and spread of the resulting solutions in the Pareto front.

To demonstrate and evaluate the optimization capabilities offered by the proposed DSE algorithm, two different case studies were considered: one relying on the *Dispatcher* mode to ensure the commitment of the performance, peak power and energy constraints; and other relying on the *Dynamic Scheduler* mode, to be able to instantaneously adapt to the application workload variations.

5.1. Case study A: Optimization strategy overview

The first case study is used not only to demonstrate one of the possible outcomes of the DSE algorithm, showing platform morphing under the *Dispatcher* mode, but it also shows that the introduced changes to the NSGA-II do not affect its convergence properties. This is particularly important, since it validates the optimization capabilities of the proposed framework. For this evaluation, the same arithmetic benchmark that was used in Neves et al. (2015a) to validate the adopted reconfigurable and morphable platform is used. The benchmark consists of four different kernels, each performing distinct vector operations, as shown in Fig. 10a.

To execute the benchmark under a set of performance, peak power and energy constraints, the architecture model depicted in Fig. 10b is

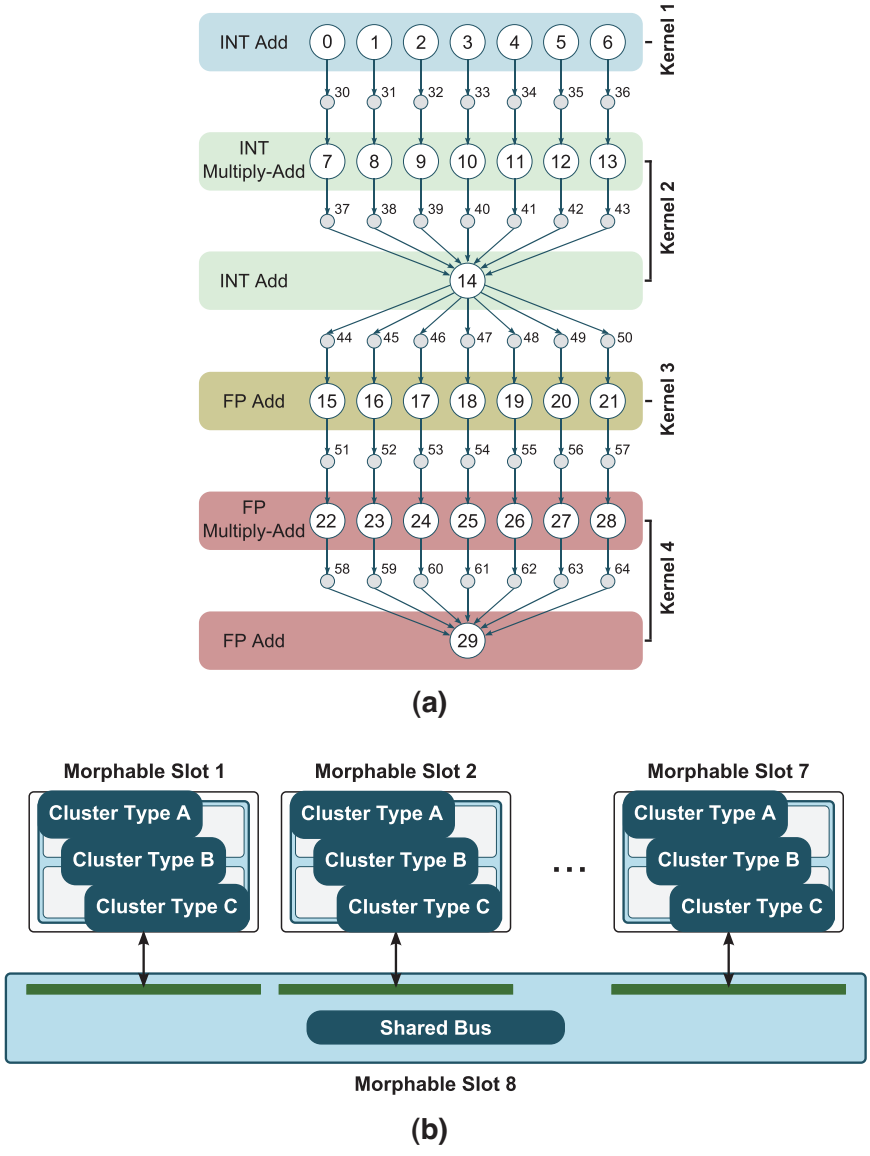


Fig. 10. Arithmetic benchmark application model (a) and the corresponding architecture configuration (b).

Table 1
Arithmetic kernel benchmark case study (operation latency is shown in average clock cycles per operation, CCPO).

Components (number of cores)		Power [mW]	INT latency [CCPO]		FP latency [CCPO]		Communication	Reconfiguration	
			Add	Multiply	Add	Multiply	Latency [CCPO]	Latency [ms]	Power [mW]
Cluster type A	(15)	615.20	2.15	4.38	25.09	51.39	0	10	44
Cluster type B	(12)	636.58	2.44	2.19	16.78	15.83	0	10	44
Cluster type C	(8)	600.65	3.66	3.28	3.79	3.79	0	10	44
Shared bus	(1)	168.10	n.a.	n.a.	n.a.	n.a.	10	2	44
Control power		367.9 mW							
Operating frequency		100 MHz							

used, where each morphable slot can be reconfigured to instantiate one of three possible cluster types (A, B and C), each composed of a set of homogeneous PEs, or turned off to save power. Although all the underlying PE architectures support all the instantiated benchmark arithmetic operations, the simpler architectures rely on software libraries (instead of hardwired instructions) to execute the most complex arithmetic operations (e.g. multiplication and FP operations). In particular, the most complex architecture (Type C) provides hardware specific units for every type of operations (integer and FP addition and multiplication), while the intermediate architecture (Type B) only includes hardware units for integer addition and multiplication (no FP operations) and the simpler architecture (Type A) only includes arithmetic units for simple integer operations (no multiplication or FP operations). These differences result in different processing latencies for each kernel, and in different area requirements for the corresponding PEs, which in turn results in more or less PEs instantiated in each cluster.

The characteristics of each instantiated cluster are shown in Table 1, together with the latency of each vector operation on every PE architecture (normalized in clock cycles per operation, CCPO) and their corresponding power dissipation. It also includes the reconfiguration/adaptation latency and inherent power, as well as the power dissipation of the system's underlying control logic. This way, a comprehensive model of the entire morphable system could be established and applied to the adopted DSE algorithm. The DSE algorithm was run by assuming input vectors of about 15 million cells for this arithmetic benchmark, processed in the same order as in Neves et al. (2015a).

The obtained Pareto front is depicted in the plots from Fig. 11, representing the interactions between each objective function in the three-dimensional search space (namely, between peak power dissipation and latency; between energy consumption and latency; and between peak power dissipation and energy consumption, plotted in Fig. 11a, b and c, respectively). These plots depict a good convergence of the obtained solutions, showing that the convergence properties of the NSGA-II algorithm were unaffected by the imposed changes. Moreover, the observed spread of the solutions (also unaffected) results in several levels of optimization, representing different trade-offs between the targeted objective functions. This provides the Hypervisor with a greater flexibility than the original system (Neves et al., 2015a) to adapt the execution to the system runtime requirements.

The execution plans depicted in Fig. 12 represent two of the best optimized objective function solutions taken from the Pareto front (see Fig 11). In particular, Fig. 12a depicts a solution that represents the best execution-time optimized plan, mostly trading-off latency for peak power consumption. By analyzing the execution queue of this solution, it can be concluded that even though it is best optimized for latency, it also presents some reduction of the peak power dissipation when running Kernels 1 and 2. On the other hand, the solution depicted in Fig. 12b illustrates the execution plan characterized by the lowest peak power dissipation, although resulting in a higher execution time. In this case, the system uses two slots in parallel (for a short period of time) to run Kernels 1 and 2 (with lower latencies), while only one slot is used for Kernels 3 and 4, corresponding to the most time consuming kernels.

The fact that the solution depicted in Fig. 12b represents, respectively, 54% and 45% lower peak power dissipation and energy consumption values than those obtained with the solution depicted in Fig. 12a, clearly shows that the Hypervisor (in *Dispatcher* mode) can fully take advantage of the execution plans generated by the DSE algorithm, when adapting the execution to the system runtime requirements. These results are sustained by the observed Pareto front, comprehending solutions that represent different trade-offs between the considered optimization targets, thus providing different levels of performance, peak power dissipation and energy consumption.

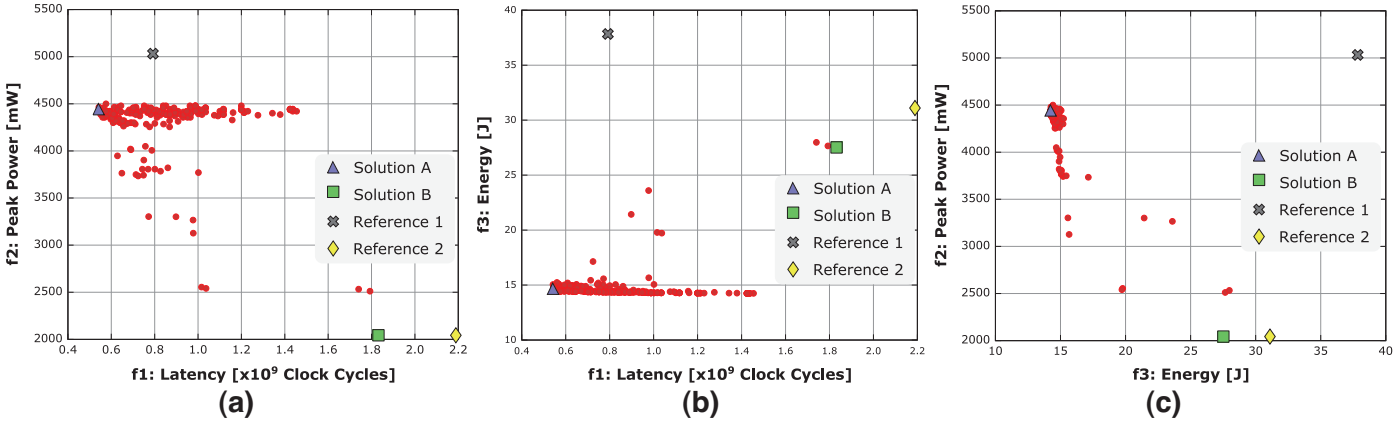
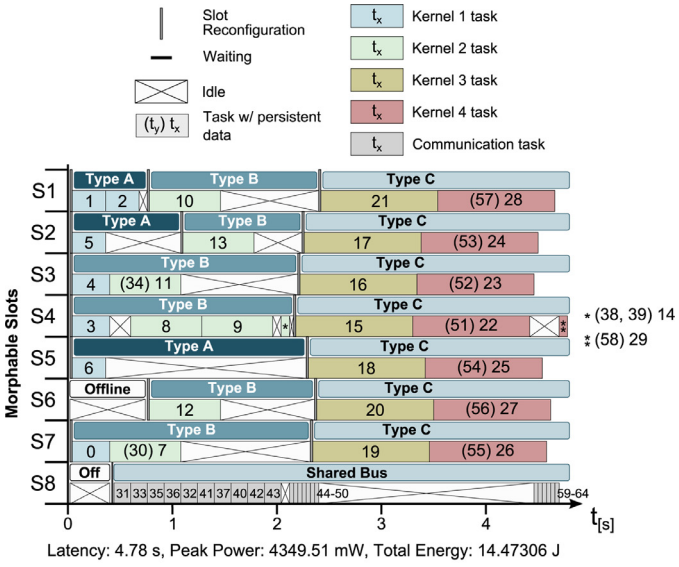
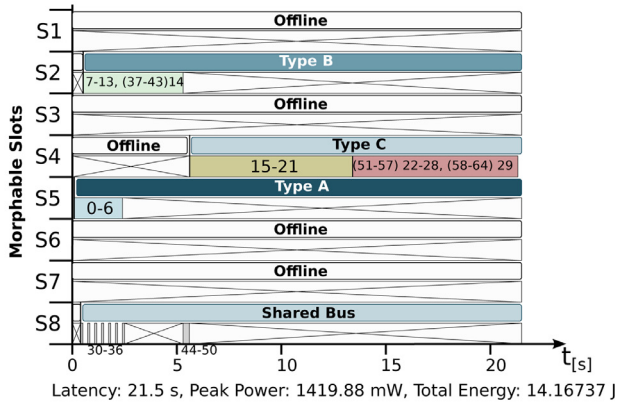


Fig. 11. Interactions between the three objective functions in the Pareto front. (a) Interaction between peak power dissipation (Eq. (4)) and latency (Eq. (2)); (b) interaction between energy consumption (Eq. (5)) and latency (Eq. (2)); and (c) interaction between peak power dissipation (Eq. (4)) and energy consumption (Eq. (5)). The marked solutions represent the execution plans shown in Fig. 12 (*Solution A* and *Solution B*) and the reference runs obtained with the platform proposed in Neves et al. (2015a) (*Reference 1* and *Reference 2*).



(a) Execution time optimization.



(b) Low-power optimization.

Fig. 12. Execution plans, referring to the application and architecture models from Fig. 10, representing two solutions obtained from the resulting Pareto front. Task identifications shown between parenthesis represent communication tasks between two adjacent tasks mapped to the same component. In this case, the data corresponding to the communication tasks persists in the component, while switching the processing tasks.

In fact, when compared with the Hypervisor's optimization policies proposed in Neves et al. (2015a), the herein proposed approach not only allows the Hypervisor to adapt the system to real-time requirements (by switching the execution plans) without any runtime penalization to re-evaluate the best suited PE configuration, but also allows a greater flexibility to cope with such requirements, given the number of generated solutions.

5.2. Case study B: Optimization under variable workloads

The execution complexity of several existing applications is directly related to the characteristics of the input data. Moreover, user-defined parameters often provide additional levels of complexity. As a result, such applications are usually very difficult to model in order to be used together with optimization algorithms, since the models must take into account those parameters and the variable characteristics of the input data itself. As such, this second case study depicts how the execution of such applications can be optimized by using the proposed DSE algorithm. In fact, it can be demonstrated that by varying certain parameters in the application model, different search granularities and intensities are attained, resulting in different levels of output data size and latency in each phase. This can be achieved by running the DSE algorithm with those different parameter levels, thus generating different execution plans for the considered scenarios. The execution plans can then be used by the Hypervisor to adapt the system architecture according to the application's real-time requirements.

The presented case study relies on a proof-of-concept benchmark based on a biological sequence alignment application model (Mahram & Herbordt, 2015), composed of multiple processing phases and comprehending several existing dedicated processing architectures for each of those phases. According to current state-of-art approaches, a heuristic indexed search within the whole data set is initially performed, in order to identify regions of interest in a reference sequence for a number of query sequences to be analyzed. Then, a filtering phase is followed, that further refines the search. The execution of both phases is performed according to specific user-defined parameters. The third and final phase of the algorithm comprehends a local alignment algorithm, which takes the resulting interest regions, performs the local alignment and returns the corresponding score results.

In order to demonstrate how such an application can take advantage of the proposed DSE algorithm, illustrative models of the complexity and output data size of each phase are derived, by taking into account the characteristics of those applications (Mahram & Herbordt, 2015; Neves et al., 2015b). Also, existing processing architectures, specially optimized for each phase, are used as the base architecture. The *Latency* and *Output Size* columns in Table 2 represent approximate models that are used to estimate the latency and output data size of each execution phase, depending on user-defined parameters and on the size of the reference and query sequences. For the *Indexed Search* phase latency, the K value represents a user-defined parameter that defines the size of the search seed, while M and N represent the sizes of the reference and query sequences, respectively. The output size formula defines the amount of resulting pointers (addresses) corresponding to the obtained regions of interest, i.e., the number of Hits. In this equation, β represents a data dependent value that reflects the characteristics of the reference sequence. The *Filtering* phase refines the indexed search results, by further reducing the number of regions of interest (Hits), resulting in a smaller subset of potential sites (P. Sites) for local alignment.

Table 2
Biological sequence alignment case study: Component specifications and application model equations.

Execution phase	Components (number of cores)		Power [mW]	Latency [CCPO]	Output size [number of cells]	Reconfiguration	
						Latency [ms]	Power [mW]
Index Search	Index ASIP	(14)	701.30	$[K + \log_2(M-K)] \times \frac{N}{K}$	$\text{Hits} = \frac{N \times (M-K)}{K^2} \times \beta$	10	44
Filtering	MB-LITE M	(12)	636.58	$\text{Hits} \times \log_2(\text{Hits}) + \text{Hits}$	$\text{P.Sites} = \text{Hits} \times R/\gamma^2$	10	44
Local Alignment	Align ASIP	(5)	610.50	$\text{P.Sites} \times 2N^2/\alpha$	-	10	44
Communication	Stream Interconnect	(1)	101.78	4	-	2	44
Communication	FIFO	(1)	62.30	1	-	1	44
Control power	367.9 mW						
Operating frequency	100 MHz						

The latency and data output size of this phase depend on the amount of Hits resulting from the Indexed Search phase, where R is a user-defined parameter to adjust the filtering intensity, while γ is a variable characterizing the similarity between the query and the reference sequences. The last phase implements a *Local Sequence Alignment* based on a dynamic programming approach. Its latency depends on the number of P. Sites resulting from the filtering phase and on the size of the query sequences. The α parameter is a speedup factor that depends on the used alignment algorithm (e.g. the striped SIMD search algorithm (Farrar, 2007) provides a 6x speedup factor).

The adopted structure considers the existence of seven morphable slots, allowing the instantiation of an homogeneous cluster of PEs in each slot, each cluster composed of a different number of PEs, and each PE specially adapted for each processing phase (kernel). In particular, the *Indexed Search* phase is executed on the Application Specific Instruction-set Processor (ASIP) architecture proposed in Sebastião, Dias, Roma, and Flores (2014), referred here as `Index ASIP`; the *Filtering* phase is executed on a MB-LITE soft-core architecture (Kranenburg & van Leuken, 2010), denoted as `MB-LITE M`; and the *Local Alignment* phase is performed on the ASIP architecture proposed in Neves et al. (2015b), named `Align ASIP`.

Although a total number of seven morphable slots are considered, slots 1, 2 and 3 are reserved for each of the processing architectures, as shown in Fig. 13c, such as to ensure that at least one single cluster of each of the three co-existing architectures is always present in the system, interconnected in a pipelined manner (according to the application model in Fig. 13a). Further details about each component are presented in Table 2.

Hence, by varying the user-defined parameters and the intrinsic characteristics of the input data set, a number of different scenarios can be exploited by the DSE algorithm, resulting in distinct execution plans that will be used by the Hypervisor to adapt the execution in real-time. Although the latency and output sizes have a direct relation with the characteristics of the input data set, in the considered case the β and γ values were set to 0.7 and 0.5, respectively, representing a fair differentiation across the query and reference sequences, respectively. The α value was set to 6, corresponding to the SIMD striped search algorithm (Farrar, 2007). Furthermore, two levels of user-defined parameters were used, representing different scenarios for a wide relaxed search ($K = 100$, $R = 0.1$) and a fine tuned intensive search ($K = 15$, $R = 0.01$).

By extracting the mappings of the execution plans resulting from the DSE algorithm, the adaptive architecture configurations shown in Fig. 14 were derived and provided to the Hypervisor, as described in Section 3.2. This way, the Hypervisor (in *Dynamic Scheduler* mode) can adapt the architecture (in runtime) in two different abstraction levels. On a high management level, the Hypervisor monitors the result of the application and switches the execution plan between a scenario corresponding to a relaxed search (e.g., Fig. 14a) or to a more intensive search (e.g., Fig 14b), depending on user requirements or on the input data characteristics. On a lower management level, the Hypervisor adapts the architecture in each morphable slot depending on the data flows existing between phases of the application, according to the execution plan being considered at that instant.

The mapped processing cluster types and morphable slot allocations, shown in Fig. 14, represent execution-time optimized solutions for the considered scenarios. It is possible to verify the different

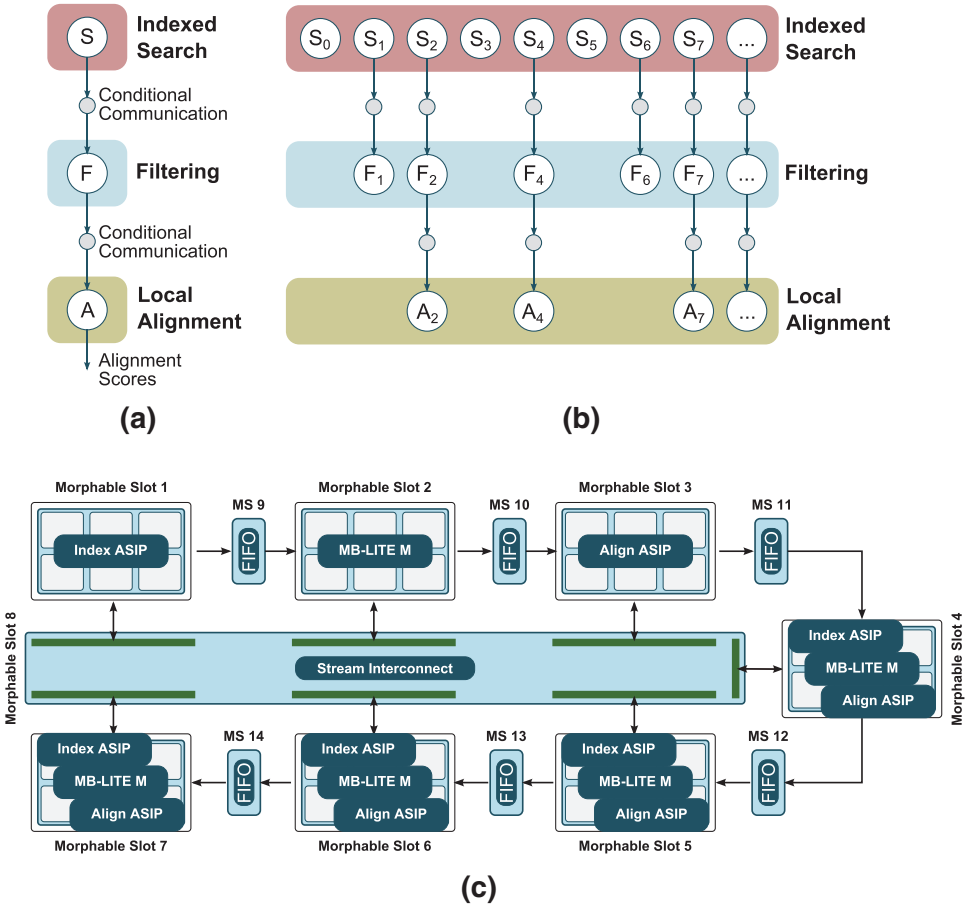
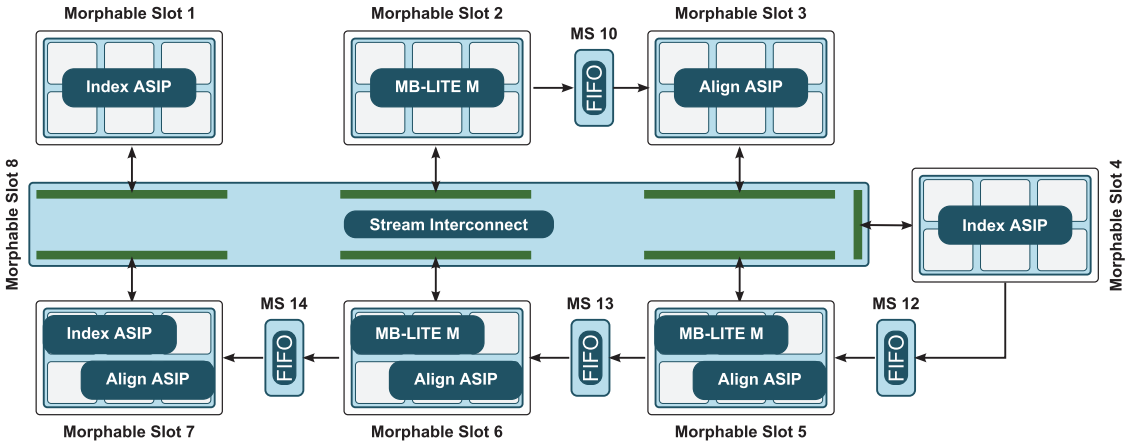
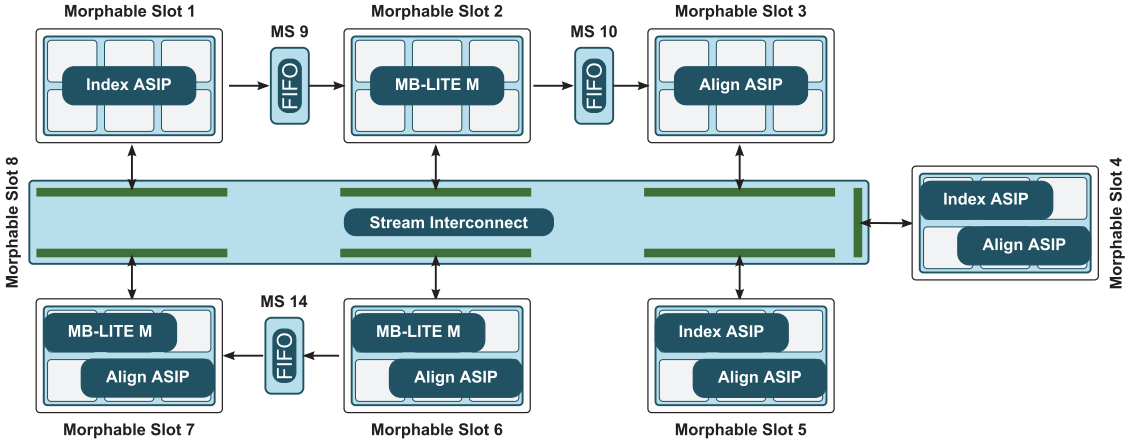


Fig. 13. Biological sequence alignment application model (a), example execution showing dynamic workload variation (b) and architecture configuration (c). Although the application model accounts for every possible outcome for the communication between phases, the size of the output of each phase (if any) is only known during runtime.



(a) Wide search with relaxed parameters.



(b) Intensive search.

Fig. 14. Two resulting execution plans for the biological sequence alignment application. Even though the plans are optimized for different execution scenarios, the Hypervisor can still fine tune the configuration of the system according to the amount of data outputted by each phase.

number of cores of each architecture allocated according to the intensity of each phase of the application, resulting from the chosen input parameters. As should be expected, for a wide and relaxed search, the Indexed Search phase is characterized by a lower complexity and results in few regions of interest, leading to a relaxed Filtering phase and to few potential alignment sites for the Local Alignment phase. This results in an even allocation of types of processing clusters, as shown in Fig. 14a. On the other hand, for a fine tuned intensive search, the complexity of the Indexed Search phase is higher and finds significantly more regions of interest, which even after an intensive Filtering phase, results in a large amount of potential alignment sites for the Local Alignment phase. The solution for this scenario results in larger numbers of allocated `Index ASIPs` and `Align ASIPs` (see Fig. 14b), allowing a more flexible balancing of architectures by the Hypervisor, depending on the real-time throughput requirements.

5.3. Application of DSE to DVFS and power-gating

Although the presented prototyping of the proposed DSE framework was supported on runtime dynamic reconfigurable systems based on FPGA devices, the algorithm and the system specification models can be equally applied to systems deploying other approaches to attain the aimed energy efficiency, such as DVFS or clock- and power-gating. In accordance, the modeling of such techniques can be defined by using an approach entirely similar to the one that was adopted for the FPGA. However, in this case, instead of a dynamically reconfigurable system, the algorithm assumes a static instantiation of every single component of the specification model, not requiring any specific modeling for the reconfiguration of the morphable slots. This results in a heterogeneous multi-processor system, with different numbers of processing cores for each available architecture, where the DSE algorithm simply defines appropriate clock/power gating procedures to all unused components of the system, instead of reconfiguring their architecture. Hence, depending on the runtime application requirements, the dataset to be processed is always transferred to the most suited processing architecture of the system.

In accordance, the modeling of these DVFS techniques is relatively straightforward: the clock- and power-gating are modeled the same way as the FPGA reconfiguration process, i.e. the reconfiguration time and power are replaced by the respective synchronization time and power overheads, required to ensure the switching (on/off) of the system components. Similarly, the properties of each component have to be taken into account by this modeling, where the throughput and power consumption are replaced by a function of both the operating frequency and the supplied voltage.

The inclusion of the models corresponding to these DVFS techniques in the proposed DSE framework greatly increases the range of

processing systems that may benefit from these optimization capabilities: from highly dynamic systems implemented in reconfigurable devices (FPGAs) to multi-processor homogeneous or heterogeneous systems based on ASIC chips.

5.4. Discussion with related work

The presented experimental results demonstrate the optimization capabilities that are offered by the proposed DSE algorithm, when compared with the execution of the Hypervisor-based platform proposed in Neves et al. (2015a). As described in Neves et al. (2015a), the former platform deploys several runtime optimization policies that allow the Hypervisor to adapt the processing architecture to the instantaneous real-time system requirements, without any prior knowledge of the characteristics of the application under execution. This way, the Hypervisor can only monitor the PEs execution and any external condition that may affect it. The gathered information is then used to adapt the morphable system through a runtime exploration of the best suited configuration. As a consequence, several significant overheads that affect the efficiency of the platform are imposed. Specifically, the Hypervisor is required to: (i) read the performance counters of the PEs; (ii) process the obtained information through the set of optimization policies; and (iii) trigger the appropriate reconfiguration commands to the accelerator.

On the other hand, the proposed DSE methodology, coupled with the set of compilation tools that extract the application's model, allow the achievement of a higher level of optimization, since the Hypervisor is now provided with further information about the application's behavior and requirements, through the set execution plans generated by the proposed framework. Accordingly, the significant overheads that were observed in the former Hypervisor platform (Neves et al., 2015a) are now significantly mitigated, since the Hypervisor is no longer required to intensively monitor the execution of the PEs. Moreover, according to the execution plan under consideration, it can anticipate the required reconfigurations, hiding the adaptation process behind the application execution.

This overhead reduction can be clearly ascertained with the two execution scenarios of *Case study A* (see Fig. 11) that compare the results obtained with the original platform (Neves et al., 2015a) with the corresponding set of solutions obtained with the proposed DSE algorithm. The two reference solutions correspond to: (i) an execution time optimization scenario (*Reference 1*); and (ii) a 2 Watt peak power dissipation limitation scenario (*Reference 2*), executed in the original platform. From the plots depicted in Fig. 11, it can be clearly observed that such solutions are not as efficient as those obtained with the proposed DSE algorithm, corresponding to *Solution A* (depicted in Fig. 12a) and *Solution B* (depicted in Fig. 12b).

Hence, from the two presented case studies, it can be observed that the proposed framework represents a novel kernel mapping approach to FPGA-based (or ASIC-based) accelerators, by taking advantage of a well-known MOEA and by taking into account the dynamic reconfiguration/adaptation of the underlying hardware. Although several contributions exploit similar MOO techniques for code optimization (Balaprakash et al., 2014; Neugebauer et al., 2015), dynamic task scheduling (Camelo et al., 2011; Daoud & Kharma, 2011; Sheikh & Ahmad, 2013; Xu et al., 2014) and static application mapping in multi-processor systems (Mariani et al., 2010; Palermo et al., 2009) and heterogeneous systems (Erbas et al., 2003), none of them takes advantage of the offered dynamic reconfiguration/adaptation capabilities. In fact, the only contributions that take such capabilities into account mostly target HW/SW co-synthesis and HLS problems (Czarnecki & Deniziak, 2008; Gschwandtner et al., 2014; Miramond & Delosme, 2005; Shang & Jha, 2002; Wildermann et al., 2011). Instead, the herein proposed approach targets a system-level optimization of CPU-coupled morphable heterogeneous accelerators, resulting in a compile-time tool that can efficiently map an application's kernels to the morphable PEs. Moreover, it was demonstrated that this DSE optimization may be applied to a vast application domain, including not only simple but also complex and highly data dependent routines.

6. Conclusions

A new DSE algorithm based on a variation of the NSGA-II MOEA is proposed in this manuscript. The algorithm targets efficient mappings of an application's kernels to a morphable many-core accelerator structure, by exploiting runtime dynamic reconfiguration/adaptation of the processing architecture. To attain this objective, a novel encoding of the solution space, together with a specially devised DSE operator were defined, resulting in adaptive and highly optimized execution plans. Such plans are then used by a Hypervisor management module to control (in real-time) the dynamic adaptation of the accelerator architecture.

Two case studies were used to evaluate the gains and advantages offered by the proposed algorithm. In a first case study, different execution plans, targeting different optimization objectives, were used to adapt the execution according to external system constraints, such as performance or power dissipation. The obtained results showed reductions up to 54% and 45% in peak power dissipation and energy consumption, when compared to non-optimized solutions. The last case study presented a proof-of-concept prototype illustrating how highly configurable and data dependent applications can take advantage of the devised algorithm. In this case, different input parameters (according to user-defined requisites) were took into consideration for the optimized adaptation of the morphable slots, while still maintaining fine tuning functionalities to adapt the overall execution to the input dataset. Finally, the presented contributions of the proposed optimization framework were discussed and compared in terms of the achieved improvements to the covered state-of-the-art and previous related works. Moreover, the application of the proposed strategies in ASIC technologies, by relying on clock/power gating and DVFS techniques was also considered.

The combination of the proposed optimization framework with other application analysis techniques (e.g. polyhedral analysis) is envisaged as a future challenge, enabling the deployment of a complete development toolchain for morphable accelerators. On the other hand, the possibility of including alternative performance metrics other than the provided latency, peak power dissipation and energy consumption is also regarded as highly promising improvement directions to the proposed framework, allowing a broader range of optimization goals. Such improvements can be performed by also considering alternative base MOEAs, in order to deal with possible convergence degradations resulting from the adoption of different levels of dimensionality (i.e. higher numbers of objective functions) and specific properties of the solution space. Moreover, considering that the proposed DSE algorithm targets CPU-coupled accelerators, another possible enhancement direction would be to model both the CPU processing architecture and its high-speed link to the accelerator, allowing the application to be completely mapped in the entire optimization system, accounting for the inherent overheads of the kernel off-loading to the accelerator.

Acknowledgements

This work was partially supported by national funds through Fundação para a Ciência e a Tecnologia (FCT) under projects PTDC/EEA-ELC/117329/2010, UID/CEC/50021/2013 and research grant SFRH/BD/100697/2014.

References

Balaprakash, P., Tiwari, A., & Wild, S. M. (2014). Multi objective optimization of hpc kernels for performance, power, and energy. In *High performance computing systems. performance modeling, benchmarking and simulation* (pp. 239–260). Springer.

Behnamian, J., Zandieh, M., & Ghomi, S. F. (2009). Parallel-machine scheduling problems with sequence-dependent setup times using an aco, SA and VNS hybrid algorithm. *Expert Systems with Applications, 36*(6), 9637–9644.

Camelo, M., Donoso, Y., & Castro, H. (2011). MAGS—An approach using multi-objective evolutionary algorithms for grid task scheduling. *International Journal ofApplied Mathematics and Informatics, 5*(2), 117–126.

Cheng, S.-C., Shiau, D.-F., Huang, Y.-M., & Lin, Y.-T. (2009). Dynamic hard-real-time scheduling using genetic algorithm for multiprocessor task with resource and timing constraints. *Expert Systems with Applications, 36*(1), 852–860.

Czarnecki, R., & Deniziak, S. (2008). Co-synthesis of dynamically reconfigurable SOPCs specified by conditional task graphs.. *Open Cybernetics & Systemics Journal, 2*, 206–218.

Daoud, M. I., & Kharma, N. (2011). A hybrid heuristic–genetic algorithm for task scheduling in heterogeneous processor networks. *Journal of Parallel and Distributed Computing, 71*(11), 1518–1531.

Deb, K. (2008). Multi-objective optimization using evolutionary algorithms. *Wiley Paperback Series*. Wiley.

Deb, K., Pratap, A., Agarwal, S., & Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation, 6*(2), 182–197.

Erbas, C., Cerav-Erbas, S., & Pimentel, A. D. (2006). Multiobjective optimization and evolutionary algorithms for the application mapping problem in multiprocessor system-on-chip design. *IEEE Transactions on Evolutionary Computation, 10*(3), 358–374.

Erbas, C., Erbas, S. C., & Pimentel, A. D. (2003). A multiobjective optimization model for exploring multiprocessor mappings of process networks. In *IEEE/ACM/IFIP International conference on hardware/software codesign and system synthesis* (pp. 182–187). ACM.

Esmaeilzadeh, H., Blem, E., St Amant, R., Sankaralingam, K., & Burger, D. (2011). Dark silicon and the end of multicore scaling. In *International symposium on computer architecture (ISCA)* (pp. 365–376). IEEE/ACM.

Farrar, M. (2007). Striped Smith-Waterman speedsdatabase searches six times over other SIMD implementations. *Bioinformatics, 23*(2), 156–161.

Glaß, M., Lukasiewycz, M., Wanka, R., Haubelt, C., & Teich, J. (2008). Multi-objective routing and topology optimization in networked embedded systems. In *International conference on embedded computer systems: Architectures, modeling, and simulation (SAMOS)* (pp. 74–81). IEEE.

Gschwandtner, P., Durillo, J. J., & Fahringer, T. (2014). Multi-objective auto-tuning with insieme: Optimization and trade-off analysis for time, energy and resource usage. In *Euro-par 2014 parallel processing* (pp. 87–98). Springer.

Holzer, M., Knerr, B., & Rupp, M. (2007). Design space exploration with evolutionary multi-objective optimisation. In *International symposium on industrial embedded systems (SIES)* (pp. 126–133). IEEE.

Huang, L., Sethuraman, G., & Chapman, B. (2008). Parallel data flow analysis for OpenMP programs. In *A practical programming model for the multi-core era* (pp. 138–142). Springer.

Kranenburg, T., & van Leuken, R. (2010). MB-LITE: A robust, light-weight soft-core implementation of the MicroBlaze architecture. In *Design, automation and test in Europe conferenceand exhibition (DATE)* (pp. 997–1000). European Design and Automation Association.

Krishnan, V., & Katkoori, S. (2006). A genetic algorithm for the design space exploration of datapaths during high-level synthesis. *IEEE Transactions on Evolutionary Computation, 10*(3), 213–229.

Mahram, A., & Herbordt, M. C. (2015). NCBI BLASTP on high-performance reconfigurable computing systems. *ACM Transactions on Reconfigurable Technology and Systems, 7*(4), 33:1–33:20.

Mariani, G., Brankovic, A., Palermo, G., Jovic, J., Zaccaria, V., & Silvano, C. (2010). A correlation-based design space exploration methodology for multi-processor Systems-on-Chip. In *Design automation conference (DAC)* (pp. 120–125). ACM.

Miramond, B., & Delosme, J.-M. (2005). Design space exploration for dynamically reconfigurable architectures. In *Conference on design, automation and test in Europe* (pp. 366–371). IEEE Computer Society.

Neugebauer, O., Marwedel, P., & Engel, M. (2015). Multi-objective aware communication optimization for resource-restricted embedded systems. In *Proceedings of the 28th international conference on architecture of computing systems (ARCS'15)* (pp. 1–8). VDE.

Neves, N., Mendes, H., Chaves, R., Tomás, P., & Roma, N. (2015a). Morphable hundred-core heterogeneous architecture for energy-aware computation. *IET Computers & Digital Techniques, 9*(1), 49–62.

Neves, N., Sebastião, N., Matos, D., Tomas, P., Flores, P., & Roma, N. (2015b). Multicore SIMD ASIP for next-generation sequencing and alignment biochip platforms. *IEEETransactions on Very Large Scale Integration (VLSI) Systems, 23*(7), 1287–1300.

Palermo, G., Silvano, C., & Zaccaria, V. (2009). ReSPIR: A response surface-based Pareto iterative refinement for application-specific design space exploration. *Transactions on Computer-Aided Design of Integrated Circuits and Systems, 28*(12), 1816–1829.

Petrica, P., Izraelevitz, A. M., Albonesi, D. H., & Shoemaker, C. A. (2013). Flicker: A dynamically adaptive architecture for power limited multicore systems. In *ACM SIGARCH computer architecture news* (pp. 13–23). ACM.

Pilato, C., Loiacono, D., Ferrandi, F., Lanzi, P. L., & Sciuto, D. (2008). High-level synthesis with multi-objective genetic algorithm: A comparative encoding analysis. In *IEEE congress on evolutionary computation (IEEE world congress on computational intelligence)* (pp. 3334–3341). IEEE.

Sebastião, N., Dias, T., Roma, N., & Flores, P. (2014). Optimized ASIP architecture for compressed BWT-Indexed Search in bioinformatics applications. In *International conference on high performance computing and simulation (HPCS)* (pp. 527–534). IEEE.

Shang, L., & Jha, N. K. (2002). Hardware-software co-synthesis of low power real-time distributed embedded systems with dynamically reconfigurable FPGAs. In *Asia and South Pacific design automation conference* (p. 345). IEEE Computer Society.

Sheikh, H. F., & Ahmad, I. (2013). Dynamic task graph scheduling on multicore processors for performance, energy, and temperature optimization. In *International green computing conference (IGCC* (pp. 1–6). IEEE.

Stefanov, T., Zissulescu, C., Turjan, A., Kienhuis, B., & Deprette, E. (2004). System design using Khan process networks: the Compaan/Laura approach. In *Design, automation and test in europe conference and exhibition: 1* (pp. 340–345). IEEE.

Venkatesh, G., Sampson, J., Goulding-Hotta, N., Venkata, S. K., Taylor, M. B., & Swanson, S. (2011). Qscores: Trading dark silicon for scalable energy efficiency with quasi-specific cores. In *International symposium on microarchitecture (ISCA)* (pp. 163–174). IEEE/ACM.

Whitley, D., & Sutton, A. (2012). Genetic algorithms—A survey of models and methods. In G. Rozenberg, T. Bäck, & J. Kok (Eds.), *Handbook of natural computing* (pp. 637–671). Springer Berlin Heidelberg.

Wildermann, S., Reimann, F., Ziener, D., & Teich, J. (2011). Symbolic design space exploration for multi-mode reconfigurable systems. In *IEEE/ACM/IFIP International conference on hardware/software codesign and system synthesis* (pp. 129–138). ACM.

Xu, Y., Li, K., Hu, J., & Li, K. (2014). A genetic algorithm for task scheduling on heterogeneous computing systems using multiple priority queues. *Information Sciences, 270*, 255–287.