A

Parameter Space Representation of Pareto Front to Explore Hardware-Software Dependencies

VINCENZO CATANIA, University of Catania
ANDREA ARALDO, Université Paris-Sud and Télécom ParisTech
DAVIDE PATTI, University of Catania

The Embedded Systems Design requires conflicting objectives to be optimized with an appropriate choice of hardware/-software parameters. A simulation campaign can guide the design in finding the best trade-offs, but, due to the big number of possible configurations, it is often unfeasible to simulate them all. For these reasons, Design Space Exploration (DSE) algorithms aim at finding near-optimal system configurations, by simulating only a subset of them.

In this work we present PS, a new multi-objective optimization algorithm and evaluate it in the context of the embedded system design. The basic idea is to recognize *interesting* regions, i.e. regions of the configuration space that provide better configurations with respect to other ones. PS evaluates more configurations in the interesting regions, while less thoroughly exploring the rest of the configuration space. After a detailed formal description of the algorithm and the underlying concepts, we show a case study involving the hardware/software exploration of a VLIW architecture. Qualitative and quantitative comparisons of PS against a well-known multi-objective genetic approach demonstrate that, while not outperforming it in terms of Pareto dominance, the proposed approach can balance the uniformity and granularity qualities of the solutions found, obtaining more extended Pareto-fronts that provide a wider view of the potentiality of the designed device. Therefore, PS represents a further valid choice for the designer, when objective constrains allow it.

Categories and Subject Descriptors: D.2.2 [**Design Tools and Techniques**]

General Terms: Design, Algorithms, Performance

Additional Key Words and Phrases: DSE, MultiObjective Optimization, Genetic Algorithms

ACM Reference Format:
Vincenzo Catania, Andrea Araldo and Davide Patti, 2014. Parameter Space Representation of Pareto Front to Explore Hardware-Software Dependencies. *ACM Trans. Embedd. Comput. Syst.* V, N, Article A (January YYYY), 23 pages.
DOI : http://dx.doi.org/10.1145/0000000.0000000

1. INTRODUCTION AND MOTIVATION

The design of an embedded system requires different conflicting objectives (energy consumption, performance, area) to be fulfilled. In addition, it has to cope with an increasing demand and a reduced time-to-market [WSTS 2013]. In many real cases, no analytical results are known to predict the relation between system configurations and objectives. In such cases a high level system simulation is the only way to have a picture of how system parameters impact on the objectives. A key problem is that the size of the design space to be simulated grows with the product of cardinalities of each parameter, resulting in an intractable number of simulations.

In these cases a Design Space Exploration (DSE) strategy is required, i.e. an algorithm that selects only a tractable subset of all possible configurations. The final result of a design space exploration is a subset of configurations called *Pareto set* ([Pareto 1896], see also Def. 3.1). The set of the corresponding tuples of objective values is called *Pareto front*: it represents the trade-off between

Author's addresses: V. Catania, DIEEI University of Catania; A. Araldo, Université Paris-Sud and Télécom ParisTech; D. Patti, DIEEI University of Catania.

DOI : http://dx.doi.org/10.1145/0000000.0000000

objectives, i.e. for each point of the Pareto front there is no other configuration performing better for all the objectives (see Fig. 1).

In this work we present PS, a multi-objective design exploration algorithm that introduces the concept of *interesting* and *uninteresting regions* of the configuration space. The main idea can be simplified as follows: at each iteration of the algorithm, the configuration space is divided in regions. A score is given to each region based on the *innovation* provided by its configurations in the Pareto front, i.e. how different these configurations are, in terms of objective values, with respect to the ones previously found. The regions with high score are considered the most *interesting* and, at the next iteration, more configurations are simulated there, rather than in the rest of the configuration space. Several geometrical transformations (splitting, merging etc.) are involved in the Parameter Space (aka configuration space), hence the name PS for the algorithm.

From a more abstract but interesting perspective, the proposed algorithm mimics the attention shift in human beings. For example, authors in [Kahneman 1973] models human attention as a single resource with a limited budget that must be distributed among tasks, preferring the task which the focus is on. Authors in [Sperling and Weichselgartner 1995] describe the visual attention as a spotlight that focuses, each time, on a particular region of the visual space in order to deeply process it. Similarly, PS focuses its "spotlight" only on the interesting regions and thoroughly explore them distributing the simulation budget on them. In other words, with PS we propose a new perspective on the configuration space, where parameter interdependencies play the fundamental role of making some regions more capable of generating hardly predictable solutions: catching this sort of "chaoticity" in the parameter space is the main idea behind the strategy that is being introduced in this work.

The paper is organized as follows. First, Sec. 2 places our work in the context of related effort. Then we give a theoretical formulation of the algorithm. In particular, in Sec. 3 we provide formal definitions of the concepts and the operations that will be used to describe PS algorithm. In Sec. 4 we give the theoretical description of PS. Finally in Sec. 5 we show a case study involving the exploration of the hardware/software parameters of a Very Long Instruction Word (VLIW) architecture, performing a qualitative and quantitative comparison of PS against the state-of-the-art of multi-objective genetic based approaches.

2. RELATED WORK AND OUR CONTRIBUTION

Many different design space exploration algorithms have been proposed in literature. Exploring the design space of embedded systems is difficult not only because of the huge size of the parameter space, but also due to the tight *dependencies* between parameters, which impact the objectives. Due to this complexity, exploration algorithms are usually heuristic and, to a large extent, rely on simple intuitions rather then formal arguments.

Some exploration strategies assume some knowledge about the system parameters, their semantics and their impact on design objectives. The *Dependency analysis*, proposed in [Givargis et al. 2002], is meant to take advantage of the parameter dependencies. A "dependency graph" is constructed and clusters – i.e. subsets of strongly dependency-connected parameters – are recognized. Each cluster is exhaustively evaluated and its "local" Pareto set is found. Then, all local Pareto sets are merged. In this way, a series of local exhaustive evaluations are performed instead of an exhaustive evaluation of all the possible configurations. Some problems arise: i) In real scenarios, parameter interdependencies may be so deep that it may be difficult to recognize independent clusters. This may lead to the need of an evaluation of almost all the possible configurations. ii) A designer may not have a precise and complete a-priori picture of the dependencies among parameters; for this reason the need of "automated approaches for computing interdependencies" is declared in the same paper. These drawbacks are not present in PS, since, although it leverages dependencies as in [Givargis et al. 2002], it is able to detect them automatically, requiring no a-priori knowledge. Moreover, our algorithm is able to capture also "local dependencies", i.e. dependencies that emerge only among certain ranges of parameters and may not hold when considering the entire ranges. This cannot be modeled in Dependency Analysis.

A preliminary estimation of hardware/software dependencies in order to reduce the exploration of the design space is addressed in [Catania et al. 2008], where authors propose to deal with the software parameters separately, statistically testing their average effect on objectives before the actual exploration is performed. Also in [Patti et al. 2014] authors analyze the impact of including compilation parameters into the design space exploration. The results of both works confirm the important role that hardware/software dependencies play in a multi-objective exploration and thus justify the choice of explicitly including the two classes of parameters in the design space without making any distinction between them from the algorithm's perspective. In fact, one of the purposes of the proposed PS algorithm is to avoid the need of any a priori knowledge about those dependencies, letting the algorithm itself discover regions where interaction among parameters generates new Pareto-optimal points (see the PS scoring system described in the next sections).

Abraham, Rau and Schreiber proposed in [Abraham et al. 2000] to decompose the system under evaluation into components that minimally interact with each other. Pareto sets for each component are found and, provided that "monotonicity" exists, the complete Pareto set is computed merging the component Pareto sets. Roughly speaking (see section 4 of the same paper for more details), monotonicity property guarantees that the best system can be obtained as a composition of the best components. This approach would perfectly work if all the components were perfectly isolated, i.e., if there were no dependencies among them. But real scenarios rarely expose monotonicity property, and thus inaccuracies may occur, as stated in the same paper.

Other approaches, as [Fornaciari et al. 2001; Ascia et al. 2002], are based on the concept of *sensitivity analysis*, i.e. measuring how much the objective varies when varying each parameter. Parameters are ordered based on their sensitivity. The first parameter (the most sensitive one) is varied, while keeping the other parameters fixed to arbitrary values, and its best value is found. The same procedure is repeated for the second parameter and so on. The limited accuracy shown by these approaches can be explained by the limited and rigid exploration of the parameter space: after fixing the best first parameter value, there are no more chances to consider different values. It is worth noticing that this approach can not capture "local sensitivity", i.e. the objectives may be more sensitive to some ranges of a parameter and less sensitive to other ranges of the same parameter. Moreover, it can not capture "combined sensitivity", i.e. the objectives may be more sensitive to a range of a parameter p_1, only when other parameters are within certain ranges, and less sensitive to the same range of p_1 when the other parameters are in different ranges. On the contrary, our algorithm can capture both local and combined sensitivity.

Authors of [Dellnitz et al. 2005; Dellnitz and Witting 2009] propose three different methods for the numerical approximation of the Pareto set. The first two require the knowledge of the gradient of the objective functions, which is usually not available in embedded system design, since the closed form expression of the objective functions is not known a priori. The third one, called "Sampling algorithm", does not need this requirement and is the closest to the PS approach presented in this work, although the mechanism is different. At every iteration a set of regions of configuration space is constructed with the aim to cover the Pareto set. The size of these regions becomes smaller at each iteration so that the covering becomes tighter. A random set of test points are evaluated in each new region; only the regions containing non-dominated points will be considered in the next iteration. An important feature differentiates it from PS. In the Sampling algorithm only the selected regions are examined, even though there is no guarantee that they contain all the Pareto points. As a consequence, the Sampling algorithm may erroneously and prematurely neglect some regions that may possibly contain a non-negligible number of Pareto points. The PS algorithm avoids this, since even though a region has not provided non-dominated points, it is not neglected, but it is populated with fewer test points, allowing to find its other non-dominated points. From this behavior also comes another important difference. The Sampling algorithm of [Dellnitz et al. 2005] keeps reinforcing the analysis of the regions selected in the previous iterations. In other words, it keeps zooming in on the regions that have been considered "interesting", i.e. rich of non-dominated points, in the previous iterations. The philosophy behind our algorithm is different. A region, which has been considered interesting at certain iterations, may not be in the future, since it has been thoroughly analyzed. On

the other hand, a region, initially uninteresting, may turn out to be interesting in the future, once all the other regions have been already exploited. Therefore, from an iteration to the next, our algorithm zooms in on regions that are currently interesting and zooms out from the uninteresting ones, spanning in a broader and more dynamical way the configuration space. In addition, in the Sampling algorithm only the number of non-dominated points is accounted to see how interesting a region is. On the contrary, we also consider the *innovation*, as already anticipated in the introduction.

The most well documented and widespread class of algorithms adopted in the design space exploration of embedded systems is represented by the Evolutionary Multi-objective Optimization Algorithms [Coello et al. 2002]. For this reason, this is the class of algorithms that we will compare to ours in the experiments of Sec. 5. Evolutionary approaches have many advantages: they provide good accuracy, they are problem-independent and do not require any a-priori knowledge of the system. The strong point of these approaches is that they consist in an exploration that is sufficiently broad (the mutation avoids being rigidly restricted to limited parts of parameter space), and, at the same time, not too scattered, as it is guided by the performances of the already evaluated configurations. Different variants of these algorithms have been proposed in literature, including mixed approaches as in [Ascia et al. 2011], where authors specifically focus on increasing the performance of an evolutionary exploration strategy, proposing a fuzzy-based technique to avoid the actual simulation of a certain number of configurations. In particular, in this work we will compare the proposed approach against a well known SPEA2 multi-objective genetic algorithm [Knowles et al. 2006] which is, to the best of our knowledge, one of the most appropriate and tested for the particular scenario we are assuming. We claim that the approach presented in this paper has most of the benefits of the evolutionary approaches, although the rationale is completely different.

3. UNDERLYING CONCEPTS AND THEORETICAL FORMULATION

In this section we provide the concepts and the formal definitions that we will use later to express PS, the algorithm we propose. First, we give the definitions of Pareto set and Pareto front that are the basis of Design Space Exploration. Then, we sketch the fundamental idea behind our algorithm. Finally, we provide a formal definition of the regions, their properties and the operations that the algorithm will perform on them.

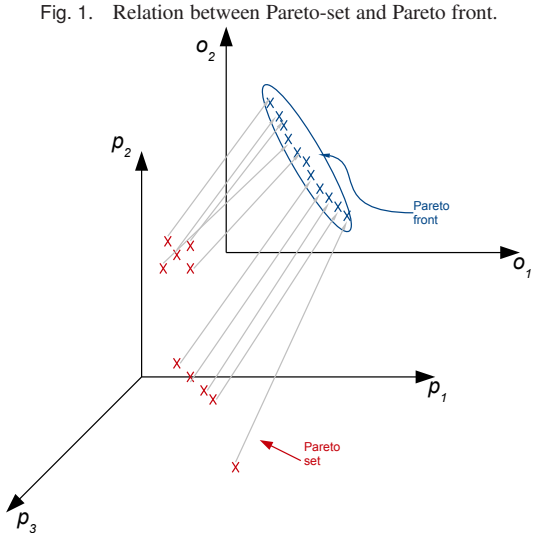
In this section and the following one, we will deliberately keep our formulation as general as possible, even if the focus of this paper is the embedded system design space exploration. The goal is to make the generality of PS emerge. We argue that it is a strong point for PS, as for genetic algorithms, since (i) it permits to apply it to a wide range of different problems and (ii) it can be effectually used when the embedded system design is particularly challenging and no much a-priori knowledge is available to the designer. As already pointed in Sec. 2, the generality of genetic algorithms has made them widely deployed with respect to the other approaches. Thanks to the generality of PS, we will not need to modify or adapt it in order to apply it to the case of embedded system design, as we will show in Sec. 5. On the one hand, this legitimate our choice of keeping our formulation general. On the other hand, more importantly, this confirms the capability of PS to automatically adapt to the problem under investigation, without any explicit intervention or knowledge of the designer, which is required, for example, by [Givargis et al. 2002; Abraham et al. 2000; Dellnitz et al. 2005] .

3.1. Pareto set and Pareto front

Let S be a parametrized system with n parameters. The generic parameter $p_i,\ i \in \{1,2,\ldots,n\}$ can take any value in the set V_i. A *configuration* $\mathbf{c}$ of the system S is a n-tuple $\langle v_1, v_2, \ldots, v_n \rangle$ in which $v_i \in V_i$ is the value fixed for the parameter p_i. The *configuration space* (or *design space*) of S [which we will indicate as $\mathscr{C}(S)$] is the complete range of possible configurations [$\mathscr{C}(S) = V_1 \times V_2 \times \ldots \times V_n$]. Naturally, not all the configurations of $\mathscr{C}(S)$ can really be mapped on S, but only a subset is feasible. We call this subset *feasible configuration space* of S and indicate it as $\mathscr{C}^*(S)$.

Let m be the number of objectives to be optimized (e.g. power, cost, performance, etc.). An *evaluation function* $E : \mathscr{C}^*(S) \times \mathscr{B} \longrightarrow \Re^m$ associates each feasible configuration of S to an m-tuple

Fig. 1. Relation between Pareto-set and Pareto front.



of values corresponding to the objectives when any application belonging to the set of benchmarks $\mathscr{B}$ is executed.

Given a system S, an application $b \in \mathscr{B}$ and two configurations $\mathbf{c}', \mathbf{c}'' \in \mathscr{C}^*(S)$, $\mathbf{c}'$ is said to *dominate* (or *eclipse*) $\mathbf{c}''$, if, given $\mathbf{o}' = E(\mathbf{c}', b)$ and $\mathbf{o}'' = E(\mathbf{c}'', b)$, it results that $\mathbf{o}' \leq \mathbf{o}''$ and $\mathbf{o}' \neq \mathbf{o}''$, where vector comparisons are component-wise and are true only if all of the individual comparisons are true ($o'_i \leq o''_i \ \forall \ i = 1, 2, \ldots, m$). To indicate that $\mathbf{c}'$ dominates $\mathbf{c}''$, we use the notation $\mathbf{c}' \succ \mathbf{c}''$.

Definition 3.1 (*Pareto set and Pareto front*). The *Pareto-optimal set* of S (or, shortly, Pareto-set) for the application b is the set:

$$\mathscr{P}(S,b) = \left\{ \mathbf{c} \in \mathscr{C}^*(S) \mid \nexists \, \mathbf{c}' \in \mathscr{C}^*(S), \mathbf{c}' \succ \mathbf{c} \right\}$$

that is, the set of configurations $\mathbf{c} \in \mathscr{C}^*(S)$ not dominated by any other configuration. The configurations belonging to the Pareto-optimal set are called *Pareto-optimal configurations*, while the *Pareto-optimal front* (or, shortly, Pareto-front) is their image, i.e. the set:

$$\mathscr{P}_F(S,b) = \{ \mathbf{o} | \mathbf{o} = E(\mathbf{c}, b), \mathbf{c} \in \mathscr{P}(S,b) \}$$

The aim of a Design Space Exploration (DSE) strategy is to give a good approximation of the Pareto-optimal set for a system S and an application b, simulating as few configurations as possible. As we will show later, at each iteration i, PS calculates an approximation $\mathscr{P}_i$ of the Pareto-set that approaches it as iterations go on. The image of $\mathscr{P}_i$ is an approximation of the Pareto-front. The simulation effort is distributed more on the regions that provide configurations in $\mathscr{P}_i$ which introduce more *innovation*, i.e. whose image in the objective space is more distant, thus "more different", from the previous Pareto-front approximation points.

3.2. Sketch of the Idea

In the design process, there may be "local decisions" for which the designer can take advantage of well-known results or heuristics. For example, if we consider a highly parallel computer architecture in a configuration with few registers, it is well-known that increasing the number of ALUs does not bring tangible better performances but only cause an increase in area occupation. An intelligent exploration algorithm would not waste time evaluating configurations with many ALUs and few registers. If p_1 is the parameter "number of registers" and p_2 is the parameter "number of ALUs",

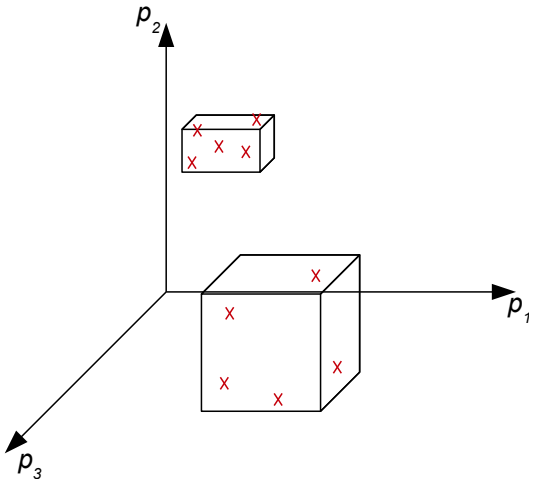


Fig. 2. In this example 5 configurations are evaluated in a small region and in a bigger region. It can be seen that the small region is more crowded.

we can say that the region:

$$R = \{ (v_1, v_2) \in V_1 \times V_2 | \, v_1 < s_1, v_2 > s_2 \}$$

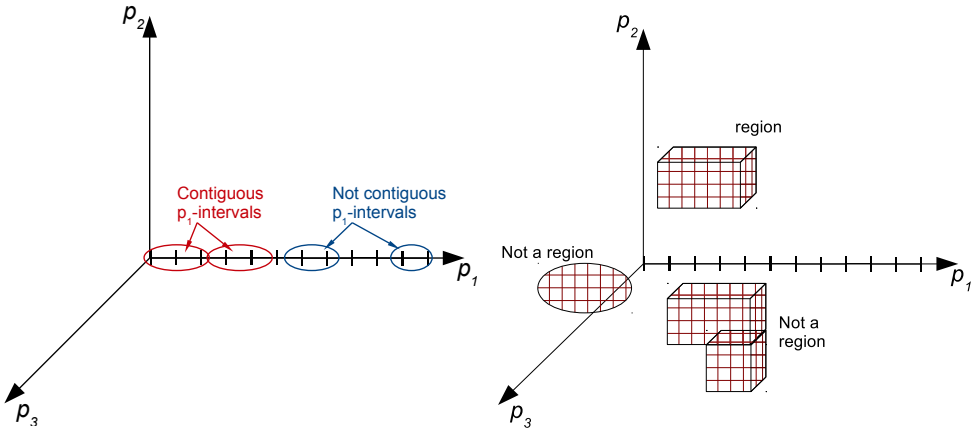
(where s_1 and s_2 are some threshold values) is *uninteresting*. This clearly shows that there are cases when, if a parameter lies in a certain range, there are ranges of the other one that are not worth being explored. Intuitively, the Cartesian product of these ranges is an *uninteresting region*.

The problem is that, in complex scenarios, not all the relations of this kind are known in advance. Some dependency may be unknown. Therefore, as stated in Sec. 2, setting up an exploration trying to take into account as much dependencies as possible is practically unfeasible. We need a methodology to "automatically" recognize interesting or uninteresting regions without requiring a priori knowledge to the designer, that is what our algorithm does.

We also consider, in measuring how much interesting a region is, the *innovation* that it adds. If, during a design space exploration, a Pareto front approximation is temporarily calculated, adding a new Pareto front point near the previous ones is not a considerable innovation, because the way it fulfills the objectives is similar to the other already evaluated configurations. On the contrary, adding a new Pareto front point that is distant from previous ones is remarkable, since it lets the experimenter discover potential performance that have not been considered yet. At each iteration, PS "zooms in" on the most interesting regions while "zooming out" from the rest of configuration space. In order to algorithmically implement this behavior, at each iteration PS assigns the same number N of simulations to each region. If a region R is interesting, it is split in k sub-regions before the next iteration starts. Therefore, in the next iteration, N simulations are assigned to each sub-region. As a result, we are populating the original region R with $k \cdot N$ simulations. On the contrary, uninteresting regions are merged, so that each one will receive less effort in the next iteration. In other words, areas of the configuration space that are more promising are split in small sub-regions, while the other ones are blended in big regions. As illustrated in Fig. 2, assigning N simulations to each region makes small ones well populated while bigger ones sparsely explored.

3.3. Formal definitions

We start by defining the parameter interval and the concept of interval contiguity. Using these definitions, we construct the region and define the region contiguity. Then we define the operations of splitting and merging regions. Finally, we provide the definition of separation.



(a) Examples of contiguous and non contiguous intervals. (b) Examples of regions.

Fig. 3. Intervals and regions.

Since regions will be defined as Cartesian products of parameter intervals, we need the following definition.

Definition 3.2 (*Parameter interval*). Let p_i be a parameter and $a_i, b_i \in V_i, a_i < b_i$. A p_i- interval is

$$[a_i..b_i] = [a_i, b_i] \cap V_i$$

i.e. $[a_i..b_i]$ is the set of values of V_i lying in $[a_i, b_i]$.

We consider only parameters p_i with ordering, i.e. such that $\forall a, b \in V_i$ it is possible to say $a < b$ or $a = b$ or $a > b$. The following definition of interval contiguity will be the base of the concept of region contiguity, used to define merging and splitting operations.

Definition 3.3 (*Contiguous intervals*). Two intervals are contiguous if and only if they do not overlap and merging them results in a new interval. More formally, let $[a_i..b_i]$ and $[x_i..y_i]$ be two p_i-intervals. They are said to be contiguous (see Fig. 3(a)) if and only if

$$\begin{array}{l} [a_i..b_i] \cap [x_i..y_i] = \emptyset \text{ and} \\ [a_i..b_i] \cup [x_i..y_i] \text{ is a } p_i\text{- interval or } [x_i..y_i] \cup [a_i..b_i] \end{array} \quad (1)$$

Definition 3.4 (*Region*). A region is a Cartesian product of parameter intervals, as illustrated in Fig. 3 (b). More formally, if $p_1, \ldots, p_n$ are the parameters of the system, and $[a_i..b_i]$ is a p_i-interval, a region is a set of the following form:

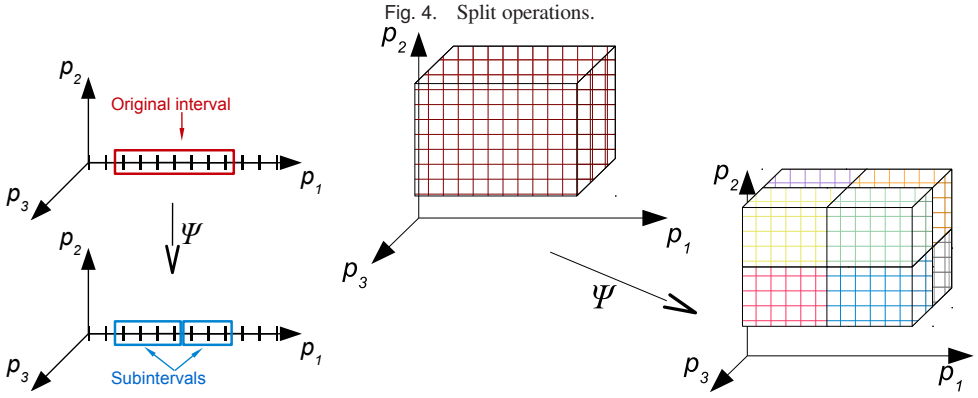
$$R = [a_1..b_1] \times \cdots \times [a_n..b_n] = \prod [a_i..b_i]$$

We will refer later to "big" and "small" regions. The sense of these terms is related to the cardinality of regions.

Definition 3.5 (*Interval and region comparison*). A p_i-interval $[a_i..b_i]$ is said to be bigger then another p_i-interval $[c_i..d_i]$ if and only if $|[a_i..b_i]| > |[c_i..d_i]|$. Similarly, a region R_1 is said to be *bigger* then R_2, if $|R_1| > |R_2|$

Note that with $|\cdot|$ we indicate the cardinality of a set. Therefore, $|[a_i..b_i]|$ is the number of values of V_i lying between a_i and b_i and has not to be confused with $b_i - a_i$. Similarly $|R_i|$ is the number of configurations inside the region R_i.

Fig. 4. Split operations.



(a) Splitting an interval. (b) Splitting a region.

As already explained, we will split interesting regions and merge uninteresting ones. The region splitting and merging operations are defined starting from the corresponding operations defined on intervals.

Definition 3.6 (*Splitting an interval*). Given a p_i interval $[a_i..b_i]$ with more than one element, i.e. $|[a_i..b_i]| > 1$, the splitting operation produces as result two contiguous intervals $[a_i..c_i], [d_i..b_i]$ such that

$$\begin{cases} [a_i..c_i] \cap [d_i..b_i] & = \emptyset \\ [a_i..c_i] \cup [d_i..b_i] & = [a_i..b_i] \\ |[a_i..c_i]| & = \left\lceil \frac{|[a_i..b_i]|}{2} \right\rceil \end{cases} \tag{2}$$

We define the split operator ψ as the one that, when applied to the intervals, gives the result of the split operation, i.e., in the case above, $\psi[a_i..b_i] = \{[a_i..c_i], [d_i..b_i]\}$. If $|[a_i..b_i]| = 1$, the interval cannot be split and $\psi[a_i..b_i] = [a_i..b_i]$.

The interval split is illustrated in Fig. 4 (a). Note that, when splitting an interval, the resulting intervals must satisfy two conditions: (i) they must be contiguous and, merging them, the initial interval must be obtained; (ii) the point in which we cut the interval is not arbitrary, but it is close to the middle.

Definition 3.7 (*Splitting a region*). Consider a region $R = \prod [a_i..b_i]$.The split operator ψ divides it into a set of sub-regions obtained splitting each p_i-interval and combining the resulting sub-intervals with the Cartesian product, as shown in Fig. 4 (b). Formally:

$$\psi(R) = \left\{ \prod [x_i..y_i] \,\middle|\, [x_i..y_i] \in \psi[a_i..b_i] \right\} \tag{3}$$

The merging operation will be defined only on region pairs that satisfy the conditions provided in the following definition.

Definition 3.8 (*Contiguous regions*). Let $R_1 = \prod [a_i \ldots b_i]$ and $R_2 = \prod [x_i..y_i]$ be two regions. They are said to be contiguous if and only if they do not overlap and, merging them, a new region can be obtained. Formally, R_1 and R_2 are contiguous if and only if there exists a j such that:

$$\begin{array}{l} [a_i..b_i] = [x_i..y_i] \text{ for } i \neq j \text{ and} \\ [a_j..b_j] \text{ and } [x_j..y_j] \text{ are contiguous intervals} \end{array} \tag{4}$$

We call p_j the *contiguity parameter* for the two regions.

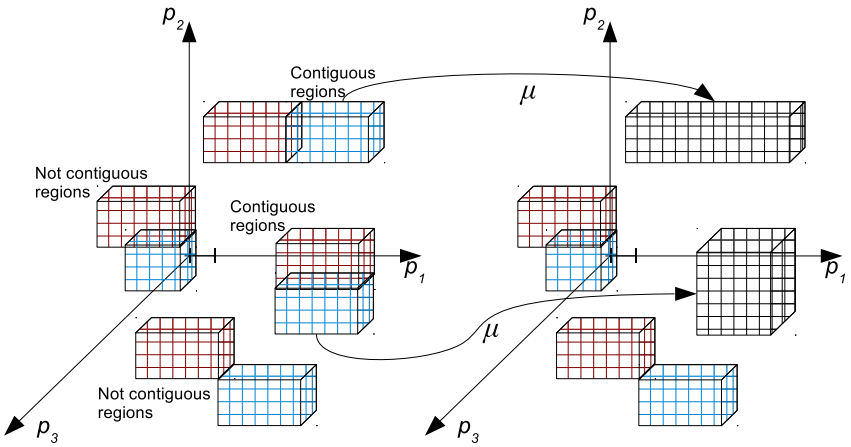


Fig. 5. Contiguous and non contiguous regions. The right figure represents the regions after merging operation μ is applied.

Examples of contiguous and non contiguous regions are given in Fig. 5. The two regions at the top, for example, are contiguous because their union can be expressed as the Cartesian product of parameter intervals, and thus they can be merged to obtain a new bigger region. On the contrary, the two leftmost regions on the left are not contiguous.

We now formally describe the merging operations.

Definition 3.9 (*Merging intervals*). Let $[a_i \dots b_i]$ and $[x_i..y_i]$ be two contiguous p_i-intervals. The merge operation applied to them produces a new p_i-interval of the following form:

$$\mu \begin{pmatrix} [a_i..b_i] \\ [x_i..y_i] \end{pmatrix} = \begin{cases} [a_i..y_i] & \text{if } b_i < x_i \\ [x_i..b_i] & \text{if } y_i < a_i \end{cases} \tag{5}$$

Definition 3.10 (*Merging regions*). Let $R_1 = \prod [a_i \dots b_i]$ and $R_2 = \prod [x_i..y_i]$ be two contiguous regions and let p_j be the contiguity parameter. The merge operation μ applied to R_1 and R_2 produces a new region of the following form:

$$\mu(R_1,R_2) = \prod_{i<j} [a_i..b_i] \times \mu \begin{pmatrix} [a_j..b_j] \\ [x_j..y_j] \end{pmatrix} \times \prod_{i>j} [a_i..b_i] \tag{6}$$

Note that $\mu(R_1,R_2)$ is mathematically equivalent to $R_1 \cup R_2$. However, we use the notation above to enforce the requirement that R_1 and R_2 must be to contiguous regions.

In Sec. 3.2 we observed that, in order to estimate how interesting a region is, it is crucial to understand the novelty it introduces, i.e. how distant the Pareto-optimal configurations discovered in that region are from the Pareto-optimal configuration previously discovered. The concept of distance is expressed by the operator defined in the next definition. We prefer to call it *separation* rather than *distance*, because usually distance denotes a commutative operator while our definition of separation is non commutative.

Definition 3.11 (*Separation*). Let $\mathbf{c}', \mathbf{c}'' \in \mathscr{C}^*(S)$ be two configurations of the system S, $b \in \mathscr{B}$ a benchmark application and $\mathbf{o}', \mathbf{o}'' \in \Re^m$ the representation of the configurations in the objective space, i.e. $\mathbf{o}' = E(\mathbf{c}', b)$, $\mathbf{o}'' = E(\mathbf{c}'', b)$, where E is the evaluation function. The separation between $\mathbf{c}'$ and $\mathbf{c}''$ is

$$s(\mathbf{c}' \to \mathbf{c}'') = \sum_{i=1}^{m} \left| \frac{o_i' - o_i''}{o_i'} \right|$$

where o_i' and o_i'' are the i-th components of $\mathbf{o}'$ and $\mathbf{o}''$, respectively.

The separation between $\mathbf{c}'$ and $\mathbf{c}''$ measures how much we must vary the image of the former to obtain the image of the latter. Note that the separation is a *normalized* measure, thanks to o'_i at the denominator. This is important since objectives are usually heterogeneous, e.g. delay and power consumption have different unities of measurement and different scale. If we did not use a normalized measure, in presence of objectives whose absolute values are small and other objectives with big values, the notion of distance would have been biased toward the latter.

4. PS ALGORITHM

In this section we present PS, the algorithm that we propose to perform the Design Space Exploration (DSE). As any other DSE algorithm, its input is the parametrized system S and a benchmark application b, while the output is an approximation of the Pareto-optimal set $\mathscr{P}(S,b)$. PS is iterative and we will describe, separately, the initialization phase, the operations performed inside each iteration and the condition that triggers the termination.

At a glance, at each iteration simulations are performed. Regions are classified based on the innovation introduced by their configurations in the Pareto-front so far calculated. The most interesting regions are split, so that each sub-region will receive more evaluation effort in the next iteration. On the contrary, uninteresting regions are merged.

By a slightly abuse of notation, considering a subset I of the configuration space, we will use $\mathscr{P}(I,b)$ to denote the non-dominated feasible configurations in I, i.e.

$$\mathscr{P}(I,b) = \left\{\mathbf{c} \in I \cap \mathscr{C}^*(S) | \nexists\, \mathbf{c}' \in I \cap \mathscr{C}^*(S), \mathbf{c}' \succ \mathbf{c}\right\} \tag{7}$$

Since in each run of the algorithm we will consider only one benchmark application $b \in \mathscr{B}$, we will omit it, replacing, for example, $\mathscr{P}(I,b)$ with $\mathscr{P}(I)$.

Each iteration is called *era*. The i-th era is characterized by:

— the set of regions $\mathscr{R}_i$ in which the configuration space $\mathscr{C}(S) = V_1 \times \cdots \times V_n$ is partitioned
— an approximation $\mathscr{P}_i$ of the Pareto-optimal set.
— a set $test_i$ of at most K configurations evaluated in the era.

K is a parameter that must be set before the algorithm runs. Note that $\mathscr{R}_i$ is a partition of the configuration space, i.e.

$$\begin{array}{l} \bigcup_{R \in \mathscr{R}_i} R = \mathscr{C}(S) \text{ and} \\ R' \cap R'' = \emptyset, \forall R', R'' \in \mathscr{R}_i \end{array} \tag{8}$$

We now formally describe PS.

4.1. Initialization

We fix $\mathscr{R}_0$, $\mathscr{P}_0$ and $test_0$ for the era 0 as follows. Era 0 has only one region that is the whole parameter space, i.e. $\mathscr{R}_0 = \{V_1 \times \cdots \times V_n\}$. A random set $test_0$ of K configurations is evaluated. The Pareto set $\mathscr{P}_0 = \mathscr{P}(test_0)$ is calculated.

4.2. Iteration steps

For era_i with $i > 0$ the following operations must be performed.

(1) In each era i, a set $test_i$ of K configurations will be evaluated. We call them *test configurations* and selecting them is exactly the goal of this step. We select only *new* configurations, i.e. the ones that have not been evaluated in past eras. In other words, $test_i$ must not overlap with $\bigcup_{j=0}^{i-1} test_j$. Since all regions R in the configuration space partition $\mathscr{R}_i$ must be properly explored, we have to distribute the test configurations among all these regions. For this reason, $\frac{K}{|\mathscr{R}_i|}$ configurations are randomly extracted from the new configurations of each region $R \in \mathscr{R}_i$. We use $test_{R,i}$ to denote the set of the selected configurations and, obviously, $\bigcup_{R \in \mathscr{R}_i} test_{R,i} = test_i$. In case the new configurations in R are less then $\frac{K}{|\mathscr{R}_i|}$, all of them are selected. In this case

$test_{R,i} = R \setminus \bigcup_{j=0}^{i-1} test_j$, where $\bigcup_{j=0}^{i-1} test_j$ is the set of the configurations already evaluated in the past.[1]

(2) All configurations on $test_i$ are simulated.

(3) The Pareto set $\mathscr{P}_i$ for era_i is defined as the set of non-dominated configurations among the ones evaluated so far, i.e.

$$\mathscr{P}_i \triangleq \mathscr{P}\left(\bigcup_{j\leq i} test_j\right) = \mathscr{P}\left(test_i \cup \mathscr{P}_{i-1}\right)$$

where the last equality can easily be proved.

(4) The test configurations that are dominated, i.e. do not belong to $\mathscr{P}_i$, are neglected, while an *innovation score* is given to the other ones, computed on the base of their separation from the non-dominated configurations of the past eras. Recalling that s is the separation (see Def. 3.11), the innovation score of a non-dominated test configuration $\mathbf{c}^*$ is defined as

$$is(\mathbf{c}^*) = \min\{s(\mathbf{c}^* \to \mathbf{c}) | \mathbf{c} \in \mathscr{P}_{i-1}\}$$

In other words, a configuration $\mathbf{c}^*$ is characterized by as much innovation as more separated it is from the configurations of the previous era.

(5) Every region $R \in \mathscr{R}_i$ is given an innovation score defined as the sum of the innovation scores of its non-dominated test configurations:

$$is(R) = \sum\{is(\mathbf{c}) | \mathbf{c} \in \mathscr{P}_i \cap test_{R,i}\}$$

(6) The total innovation and average score for the entire era is calculated:

$$\begin{array}{l} is_{TOT,i} = \sum_{R\in\mathscr{R}_i} is(R) \\ is_{av,i} = is_{TOT,i}/|\mathscr{R}_i| \end{array} \tag{9}$$

(7) The configuration space partition $\mathscr{R}_i$ is divided in three subsets, that we will comment later:

(a) *high innovation regions*, whose innovation score exceeds the average score of the previous era by a factor of α, i.e.

$$\mathscr{R}_{i,h} = \{R \in \mathscr{R}_i | is(R) > \alpha \cdot is_{av,i-1}\}$$

where α is a constant defined by the designer (for example $\alpha = 1.2$).

(b) *low innovation regions*, whose score is positive but modest, i.e.

$$\mathscr{R}_{i,l} = \{R \in \mathscr{R}_i | 0 < is(R) \leq \alpha \cdot is_{av,i-1}\}$$

(c) *no innovation regions*, whose innovation is null, i.e.

$$\mathscr{R}_{i,n} = \{R \in \mathscr{R}_i | is(R) = 0\}$$

(8) The goal of this step is preparing the regions that will compose the configuration space partition of the next era, i.e. $\mathscr{R}_{i+1}$. The following operations are performed:

(a) Each high innovation region $R \in \mathscr{R}_{i,h}$ is split to obtain $\psi(R)$, as stated in Def. 3.7. We use $\psi\left(\mathscr{R}_{i,h}\right)$ to denote the set of regions obtained as explained.

(b) Low innovation regions are left unchanged.

(c) No innovation regions are merged, two by two. To do so, pairs of no innovation contiguous regions (R_1, R_2) are selected. We use $\mathscr{M}_i$ to represent the set of these pairs. This set is constructed such that pairs are independent, i.e. there cannot be two of them sharing a region. In other words, taking whatever $(R_1,R_2),(R_3,R_4) \in \mathscr{M}_i$, $R_i \neq R_j, \forall i,j \in \{1,2,3,4\}, i \neq j$. We merge each pair of these regions, i.e. $\forall (R_1,R_2) \in \mathscr{M}_i$ we calculate $\mu(R_1,R_2)$, according to Def. 3.10. We stress that the regions that belong to each selected pair must be contiguous, otherwise it is impossible to merge them. We use $\mu(\mathscr{M}_i)$ to denote the set of the merged regions, i.e. $\mu(\mathscr{M}_i) = \bigcup_{(R_1,R_2)\in\mathscr{M}_i} \mu(R_1,R_2)$

[1] Note that if such regions exist, the total number of evaluations in era i will be less than K

(9) The set of regions $\mathscr{R}_{i+1}$ for the following era is composed of the sub-regions resulting from splitting operations ($\psi\left(\mathscr{R}_{i,h}\right)$), the low innovation regions unchanged $\mathscr{R}_{i,l}$, the result of the merging operations ($\mu(\mathscr{M}_i)$) and the no innovation regions that was not possible to merge $\left(\mathscr{R}_{i,n} \setminus \bigcup_{(R_1,R_2)\in\mathscr{M}_i}\{R_1,R_2\}\right)$. Formally:

$$\mathscr{R}_{i+1} \triangleq \psi\left(\mathscr{R}_{i,h}\right) \cup \mathscr{R}_{i,l} \cup \left(\mathscr{R}_{i,n} \setminus \bigcup_{(R_1,R_2)\in\mathscr{M}_i}\{R_1,R_2\}\right) \cup \mu(\mathscr{M}_i) \tag{10}$$

4.3. Termination Condition

The algorithm terminates at step k if

$$is_{TOT,k}, is_{TOT,k-1}, \ldots, is_{TOT,k-\beta} \leq \gamma \cdot is_{TOT,k-(\beta-1)}$$

where β and γ are selected by the experimenter. In other words, the algorithm terminates if, in the last β iterations, the "amount of innovation" has been small. Therefore, it is very probably that carrying on iterating will not considerably change the Pareto-front approximation already formed.

It is not recommended to chose $\beta = 1$, because it is possible that the $k-1$-th and k-th Pareto front approximations are not very different but next ones could be. In other words, it is unsafe to terminate as soon as the Pareto front does not considerably change between two consecutive eras. For this reason, we take into account the history of the Pareto fronts.

A pseudo-code representation of PS algorithm is in Alg.1.

4.4. Exploration vs. Exploitation in PS

The proposed algorithm has been designed in order to be inherently able to balance exploration and exploitation, which are "two cornerstones of problem solving by search" [Eiben and Schippers 1998]. While a detailed survey on the exploration vs. exploitation trade-off can be found in [Črepinšek et al. 2013], here we limitedly recall that *exploration* is the capability of an algorithm to consider different parts of the configuration space, not being confined only to a limited subset, and thus not being trapped in local optima. Following the terminology of [Weise et al. 2012], if the exploration is not sufficiently wide, the algorithm suffers from *premature convergence*, i.e. it starts to analyze only a narrow portion of the configuration space, thus ignoring potential Pareto configurations that are outside this portion. When premature convergence happens, there is insufficient *diversity* [Weise et al. 2012], i.e. all the provided solutions are similar. On the other hand, *exploitation* is the capability to recognize and densely analyze the parts of configuration space that have many Pareto configurations.

While in genetic algorithms exploration and exploitation are guaranteed by mutation, crossover and selection, in our algorithm these guarantees are provided for the following reasons:

— Merging and splitting operations are dynamic processes. A portion of the parameter space may be populated by a large number of small regions in some eras, and thus a large number of test configurations. However, after having been sufficiently explored, the innovation score of these regions may decrease, they may be merged and therefore that portion of configuration space may be covered by a small number of large regions in next eras. This is a desirable property of PS, since, after a number of eras of intense exploration of a configuration space part, this exploration may turn out to be enough thorough and so it may be time for other parts to be minutely scanned. PS dynamics can be summarized as follows: while eras alternate, PS zooms out from the configurations subspaces already sufficiently scanned in the past, whereas it zooms in on subspaces which provide novelty (guaranteeing exploitation), meaning that they have not been properly analyzed yet (this helps exploration).

Algorithm 1: PS Algorithm

Input : S
Output: $\tilde{\mathscr{P}}$ // Approximation of Pareto-set

//Initialization
$\mathscr{R}_0 \Leftarrow \{V_1 \times \cdots \times V_n\}$;
$\mathscr{P}_0 \Leftarrow$ extract_non_dominated($test_0$);

$i \Leftarrow 0$;
while *!termination_condition* **do**
  // Iteration steps
  $test_i = \emptyset$;
  foreach $R \in \mathscr{R}_i$ **do**
    $\mathscr{N}_{R,i} \Leftarrow$ configurations in R not evaluated so far;
    $|\mathscr{N}_{R,i}| \leq K/|\mathscr{R}_i|$?
      $test_{R,i} \Leftarrow \mathscr{N}_{R,i}$:
      $test_{R,i} \Leftarrow K/|\mathscr{R}_i|$ randomly selected configurations from $\mathscr{N}_{R,i}$;
    $test_i \Leftarrow test_i \cup test_{R,i}$
  end
  simulate ($test_i$);
  $\mathscr{P}_i \Leftarrow \mathscr{P}(test_i \cup \mathscr{P}_{i-1})$;
  foreach $\mathbf{c}^* \in \mathscr{P}_i$ **do**
    $is(\mathbf{c}^*) \Leftarrow \min\{s(\mathbf{c} \to \mathbf{c}) | \mathbf{c} \in \mathscr{P}_{i-1}\}$;
  end
  foreach $R \in \mathscr{R}_i$ **do**
    $is(R) = \sum\{is(\mathbf{c}) | \mathbf{c} \in \mathscr{P}_i \cap test_{R,i}\}$;
  end
  $is_{TOT,i} = \sum_{R \in \mathscr{R}_i} is(R)$
  $is_{av,i} = is_{TOT,i}/|\mathscr{R}_i|$

  //Region classification
  $\mathscr{R}_{i,h} = \{R \in \mathscr{R}_i | is(R) > \alpha \cdot is_{av,i-1}\}$;
  $\mathscr{R}_{i,l} = \{R \in \mathscr{R}_i | 0 < is(R) \leq \alpha \cdot is_{av,i-1}\}$;
  $\mathscr{R}_{i,n} = \{R \in \mathscr{R}_i | is(R) = 0\}$;

  // Splitting and merging operations
  $\psi(\mathscr{R}_{i,h}) \Leftarrow \{\psi(R) | R \in \mathscr{R}_{i,h}\}$;
  construct $\mathscr{M}_i$; // the set of pair of contiguous no innovation regions
  $\mu(\mathscr{M}_i) = \bigcup_{(R_1,R_2) \in \mathscr{M}_i} \mu(R_1,R_2)$;

  // Prepare the configuration space partition for the next iteration
  $\mathscr{R}_{i+1} \triangleq \psi(\mathscr{R}_{i,h}) \cup \mathscr{R}_{i,l} \cup \left(\mathscr{R}_{i,n} \setminus \bigcup_{(R_1,R_2) \in \mathscr{M}_i} \{R_1,R_2\}\right) \cup \mu(\mathscr{M}_i)$;
end
$\tilde{\mathscr{P}} \Leftarrow \mathscr{P}_i$

— No portion of parameter space is neglected since, at each era, all the regions, even the least interesting ones, have the chance to be explored with new evaluated points (providing exploration), even though some regions are less densely explored than others.

To summarize, thanks to the above characteristics, PS is able to recognize the regions where the best configurations are, while avoiding premature convergence, not being trapped in a narrow subset of configuration space.

4.5. Algorithm Parameters and Scalability

The parameters that determine the algorithm are the configurations budget for each era K and the threshold factor α. From a functional point of view, K is somewhat similar to the population size of a genetic approach, determining how many configurations should be evaluated in the subsequent step of the algorithm. However, as seen in the previous subsection, the way this K configurations are distributed across the parameters space is completely different. Authors of [Zitzler et al. 2000] found that increasing K not necessarily has a considerable impact on the performance of the algorithm, and this holds also in our algorithm (see for example Figure 7 and Subsection 5.3). Nonetheless, while in the genetic approach K must be sufficiently large in order to guarantee the diversity of the Pareto set configurations, in our algorithm the diversity is not impacted by K since, by construction, it is guaranteed by the way the configuration space is partitioned, independently of the number of configurations evaluated at each iteration.

On the other side, parameter α determines when a region is considered of high innovation. High values of α mean that a region will hardly be elected as high-innovation, producing less splitting operations. This slows down zooming in on portions of configuration space and makes the distribution of the simulated configurations more uniform, since there are fewer small regions with a high test configuration concentration. The opposite happens when small values of α are used. In this introductory work we are not explicitly focused on investigating how these parameters could be fine tuned, nevertheless we conducted some preliminary tests to confirm the relative stability of *PS* with different choices and to select some "reasonable" average values to perform the design space exploration, as we will discuss in Sec. 5.

With regard to the performance of the proposed approach, we observe it is related to the total number of different configurations evaluated during the exploration. We can safely make such assumption, because the process of evaluating a new configuration is the time-dominating one among all the operations performed. As we will show in the next Section, this is especially true in the hardware/software design space exploration scenario we are addressing, since benchmark compilation and execution times are orders of magnitude bigger than the splitting/merging operations and innovation scores computations. The i-th era would provide a new set $test_i$ of K configurations to be tested. However, we cannot know a priori how many of these will actually be new configurations, i.e. requiring a process of evaluation. Thus, the number of total new simulations N_{tot} is bounded as follows:

$$K \leq N_{tot} \leq K \times E_{tot}$$

where E_{tot} is the number of eras executed and assuming that at least K evaluation should be performed in order to obtain the first Pareto $\mathscr{P}_0$.

4.6. On the parallelizability of PS

While the detailed analysis and the evaluation of a parallel implementation of the proposed algorithm is not in the scope of this paper, we provide in this subsection some arguments to show different strategies that could be applied to make PS execution parallelizable.

Following the approach used in the "Master-Slaves parallel genetic algorithms" [Cantú-Paz 1998; Borkar and Mahajan 2014], we can distinguish a master and different slave processing nodes. At each era the master assigns a set of regions to each slave, which, in turn, independently performs the analysis and calculates the innovation score of the assigned regions. At the end of each era, the master collects the results from the slaves, performs the splitting/merging operations and the next era starts.

Another possible approach is closer to the "Island scheme" for GA parallelization [Borkar and Mahajan 2014]. The configuration space is initially divided in n macro-regions, each one assigned to a different processing node, which autonomously performs all the PS steps. When merging and splitting, each processing node consider the assigned macro-region as the entire parameter space. Note that, while the Island scheme for GA parallelization requires a migration operator, that is not easy to properly tune, we do not have special requirements. On the other hand, one of the macro-

regions may begin to provide very low innovation at a certain era. In this case, dedicating to it the same number K of test functions as to the other macro-regions is inefficient. To solve this problem, the number of test functions assigned to each macro-region should be periodically renegotiated, in favor of the macro-regions providing most innovation.

Opposite to the "Island scheme", a novel parallelization technique permitted by PS can consist of different instances of PS, each one running on the whole configuration space but having its own configuration space partition that separately evolves. Each PS instance independently performs the zoom-out and zoom-in on configuration space, splitting and merging its own regions. All PS instances share the indication of the configurations already tested, so that a configuration already tested by some PS instance, will not be tested again by some other PS instance. In addition, PS instances share the set of non-dominated configurations. Therefore, when a PS instance evaluates the innovation score of its regions (steps 4 and 5 of Sec. 4.2), it considers also the solutions found by the other PS instances. In this way, the configuration space is spanned in parallel by multiple PS instances, that, although autonomous, are not isolated; on the contrary, they continuously exchange information and influence each other, thanks to the sharing of the the set of already simulated configurations and the set of non-dominated ones. Otherwise stated, all the PS instances (independently but jointly) contribute to the formation of the Pareto-set.

We defer the evaluation of these different proposals to our future research, the aim of this section being just to show that our algorithm has the potentiality of being parallelized, thus leaving room for performance improvement.

5. A CASE STUDY: VLIW HARDWARE/SOFTWARE DESIGN SPACE

Although PS has been formally introduced and described as a general multi-objective optimization algorithm, in this work we will embedded system design scenario, since its properties are the ones that suggested to the authors the concepts of Pareto-based innovation and region scoring system.

In particular, we use a parametrized Very Long Instruction Word (VLIW) architecture [Kathail et al. 2000] as testbed. The philosophy behind a VLIW system makes this a natural choice for several reasons:

— *Multi-Objective trade-offs*: VLIW architectures allow designers to balance power, energy and performances objectives, resulting in extended Pareto sets which are an ideal ground for our testing purposes focused on parameter-space representation
— *Software Compiler awareness*: code scheduling for the execution of applications is statically obtained by the compiler which performs code transformations "being aware" of the underlying hardware configuration. This tight hardware/software dependence makes the VLIW scenario perfectly suitable for investigating hardly predictable parameter interactions.

In this section, we will also briefly describe the parametrized platform used as testbed for the experiments, the general evaluation flow along with the high-level estimation models and applications used as benchmarks.

5.1. Design Space: Hardware and Software Parameters

Conceptually, system parameters can be classified in three main categories: *processor*, *memory subsystem* and *compiler*. The first two categories are directly related to the physical implementation of the system, and thus parameters of those classes are usually referred as "hardware parameters". On the other side, all the parameters affecting the behavior of the compiler are referred as "software parameters", being directly involved in the process of generating the application executable running on the underlying hardware.

Processor hardware parameters are directly related to the VLIW core and include the size of the register files and the number of functional units of each type. On the software side, a set of the most impacting code compilation parameters has been selected. Table I shows the complete set of parameters included in the design space and their possible values. The total size of the design space, which grows with the product of parameter cardinalities, is about 1.33×10^{14}.

Table I. Design Space

Parameter	Values	Type
General Purpose Registers	16, 32, 64, 128	HW
Floating Point Registers	16, 32, 64, 128	HW
Predicate Instructions registers	32, 64, 128	HW
Branch Target Registers	16, 32, 64	HW
Control Registers	32, 64, 128	HW
Integer Functional Units	1, 2, 3, 4, 5, 6	HW
Floating Point Units	1, 2, 3, 4, 5, 6	HW
Memory Units	1, 2, 3, 4	HW
Branch Units	1, 2, 3, 4	HW
L1 Data Cache Size (KB)	1, 2, 4, 8, 16, 32, 64, 128	HW
L1 Data Cache Blocksize	32, 68, 128	HW
L1 Data Cache Associativity (KB)	1, 2, 4	HW
L1 Instruction Cache Size (KB)	1, 2, 4, 8, 16, 32, 64, 128	HW
L1 Instruction Cache Blocksize	32, 68, 128	HW
L1 Instruction Cache Associativity (KB)	1, 2, 4	HW
L2 Unified Cache Size (KB)	32, 64, 128, 256, 512	HW
L2 Unified Cache Blocksize	32, 68, 128	HW
L2 Unified Cache Associativity (KB)	2, 4, 8	HW
tcc_region: Scope of action of the compiler	basic block, super block, hyper block	SW
max_unroll_allowed: unroll iterations allowed	none, medium, high	SW
regroup_only: Avoids inlining	yes, no	SW
do_classic_opti: Classical optimizations	yes, no	SW
do_prepass_scalar_scheduling: Performs a prepass schedule	yes, no	SW
do_postpass_scalar_scheduling: Performs a schedule after	yes, no	SW
do_modulo_scheduling: Modular scheduling of loop code	yes, no	SW
memvr_profiled: Memory-dependencies profiling	yes, no	SW

Table II. Benchmarks

Benchmark	Application
Alloca_test	Memory array allocation test
Bmm	Matrices multiplication and elements sum
Fir_int	Finite impulse response
Mm	Floating point matrices multiplication
Sha256	Cryptocurrency header Hashing
Wave	Wavefront computation

As regards the class of benchmark being considered, a set of applications representative of some common frequently running code kernels has been selected. Table II shows the set of applications along with a brief description.

5.2. Configuration Evaluation Flow

To evaluate and compare the performance indexes of different architectures for a specific application, one needs to simulate the architecture running the code of the application. To make architectural exploration possible both the compiler and the simulator have to be retargetable. The open source project Trimaran [Trimaran 2010] provides these tools and, together with the estimation framework EPIC-Explorer [Ascia et al. 2003], has been adopted as a tested and flexible platform for carrying out the experiments.

We will now show a functional scheme highlighting the main blocks of the EPIC-Explorer framework and the interface with the Trimaran tools. The input for the whole evaluation flow is the source file of an application benchmark and the configuration of the architecture being evaluated. With reference to Figure 6 this input is represented by the blocks *App.c* and *Config*.

The application (App.c) is first compiled by the Trimaran's compiler front-end (IMPACT). At the end of the compilation flow, Trimaran supplies a simulation library (Emulib) which makes it possible to execute the VLIW code produced by Elcor, generating a file (Stats) containing the execution statistics (e.g., instruction mix, execution cycles, utilization of functional units, etc.). A cache sim-

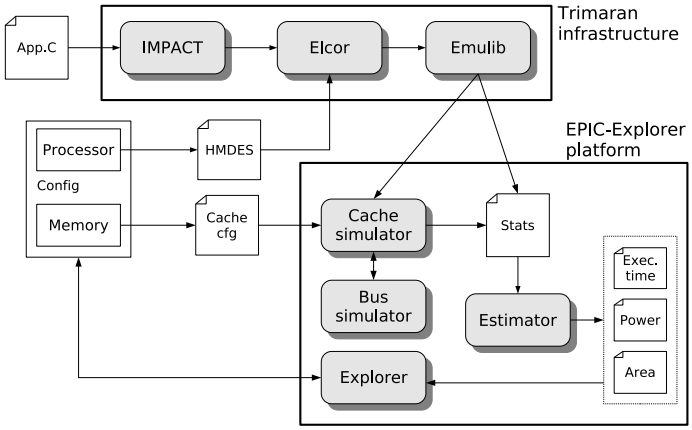


Fig. 6. Block diagram of the framework.

ulator, along with a bus simulator, is used to gather information about the behavior of the memory hierarchy in terms of miss rate and data/address traffic on the interconnection buses.

Together with the configuration of the system, the statistics produced by simulation contain all the information needed to apply the area, performance and power consumption estimation model implemented in the *Estimator* component. These are high-level abstraction models that allow to quickly get an estimation while still providing a discrete level of accuracy (for a detailed description of the models see also [Ascia et al. 2003]).

Finally, the results obtained by these models are the input for the *Explorer* component. This component executes an optimization algorithm, the aim of which is to modify the parameters of the configuration so as to minimize the three cost functions (area, execution time and energy/power consumption).

5.3. Comparison Setup

The results presented in the next section compare PS with one widely adopted Multi-objective Genetic Algorithm [Knowles et al. 2006]. To the best of our knowledge, genetic approaches in general still represent the best compromise between efficiency (i.e. time required for exploration) and accuracy of the reported Pareto Sets. Note that, for the reasons described above, mono-objective approaches have been explicitly excluded from this analysis. We also exclude other approaches (e.g. dependency based [Givargis et al. 2002]) that make use of some a-priori knowledge about the role and the semantics of each parameter, since our aim is evaluate the capacity of focusing on interesting parameter regions without any external judgment or help based on heuristics. In this work, when referring to genetic approach (or GA), we consider the widely spread *SPEA2* [Zitzler et al. 2001] variant of the Multi-objective Genetic Algorithm, remanding to works like [Zitzler et al. 2000] [Zitzler et al. 2003] for a detailed analysis on how it could differ from other genetic approaches and what could be expected from similar multi-objective strategies.

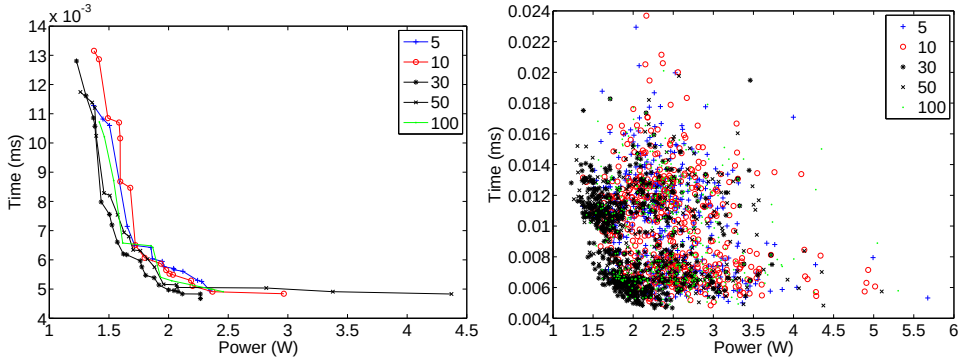
Although an analogy between the concepts of "era" and "generation" can be considered, comparisons have been carried out using as reference a fixed amount of total unique simulations, instead of a given number of eras/generations . The purpose is to make the comparison fair and accurate from the algorithm performance perspective, since a fixed budget of different simulations is more directly related to the actual amount of time required to execute the exploration.

The set of default algorithm tuning parameters used for both the genetic and PS approaches is shown in Table III. Given the similarity, both in terms of meaning and functionality, between the parameter K and the population size of the genetic approach, they both have been set to the same values. As discussed when introducing the main PS algorithm concepts, a sensitivity analysis on

Table III. Algorithm Parameters

Algorithm	Parameter	Default Value
GA	population size	30
GA	crossover probability	0.8
GA	mutation probability	0.1
PS	era budget (K)	30
PS	innovation threshold (α)	1
both PS and GA	Total simulations budget	100-500

Fig. 7. Impact of increased K values on Pareto fronts and distribution for the alloca_test application.



the algorithm parameters is beyond the scope of this work. Nevertheless, we conducted several tests with different values of K, similar to the ones shown in Figure 7, which refers to the alloca_test application and a total budget of 500 configurations. We found that initially increasing values of K seems to improve both Pareto front (as in the picture on the left) and objective distribution (on the right), but, after some threshold value, K does not seem to provide any benefit, even returning to slightly worse results for the last values of K considered (100). Similar trends have been found for the other benchmarks (not included for space reasons), showing on average a value of K around 30 as reasonable sweet spot for the objectives. A complete resource with the whole set of results and data obtained in the tests performed can be found online at [Catania et al. 2014].

5.4. Evaluation metrics

A qualitative evaluation of Paretos is probably the simplest way to have a quick glance on algorithm behavior and results. However, in tests involving a comparison between different exploration strategies, this approach could not help in capturing the whole picture, missing some important properties related to the distribution of the solutions found. In order to evaluate the Paretos from a quantitative point-of-view we will consider two metrics, namely, the *variation range* and the *average normalized absolute dispersion error*. Both of them permit to quantify the extent of the Pareto front that is an important factor to consider when evaluating its quality. Indeed, authors of [Zitzler et al. 2000; Weise et al. 2012] observe that an exploration algorithm should preferably provide Pareto fronts whose objective function range is large. In other words, following the terminology of [Weise et al. 2012], a *uniform convergence* should be guaranteed. Pareto fronts with large objective function ranges permit to investigate in a broader way the potentials of the designed device. This is important, in general, since if only a narrow portion of the Pareto front is provided due to non-uniform convergence, the decision maker may neglect some important options [Weise et al. 2012]. Having a wide Pareto front becomes essential, in particular, when designing a device that may be required to satisfy different constraints. For example, the decision maker may want to design two different multi-purpose processors. The first design may adapt to situations in which the processor is used for real time elaboration (thus requiring low execution time) and is attached to the powerline (thus not having stringent power dissipation constraints). The second design, on the contrary, may fit

situations in which the energy is provided by a battery (stringent power constraints) but only background computations are performed (long execution time is tolerable). These two processors may be successfully designed maintaining the same architecture and simply using different values for the design parameters. A wide Pareto front helps in achieving this goal.

For a given objective, the *variation range* represents the ratio between the maximum and the minimum value observed for that objective.

Another metric is the average *normalized absolute dispersion error*: it measures the average absolute difference between the distribution of points in the objective space and an ideal distribution in which the points are uniformly distributed over the objective space. Formally, let O be the image, in the objective space, of the configurations visited by the design space exploration. The generic element of O (*i.e.*, a solution) is a pair (p,t) where p and t are the average power and execution time, respectively. The two-dimensional objective space is then partitioned by a $M_x \times M_y$ mesh. For each tile T_i, $i = 1,2,\ldots,M_xM_y$ of the mesh, let N_i be the number of points in O which fall in T_i. The average absolute error, E_i, for T_i is the absolute value of the difference between N_i and the ideal number of solutions, $\overline{N}$, which should fall in T_i in case of uniform distribution. Such $\overline{N}$ can be simply computed as the ratio between the cardinality of O and the number of tiles. Thus,

$$E_i = |N_i - \overline{N}|,$$

where $\overline{N} = |O|/(M_xM_y)$. The average normalized absolute dispersion error (*ANADE*) is the average of E_i normalized to the maximum absolute error E_{max}:

$$ANADE = \frac{\sum_{i=1}^{M_xM_y} E_i/(M_xM_y)}{E_{max}},$$

where E_{max} can be computed as the average absolute error in the worst case in which all the solutions fall in a single tile:

$$E_{max} = \frac{(M_xM_y - 1)\overline{N} + |\overline{N} - |O||}{M_xM_y}.$$

5.5. Results

In this Section we first analyze the results of GA and PS approaches from a qualitative perspective, showing how Pareto curves compare when plotted on the bidimensional space of power and performance objectives. Next, we perform a quantitative analysis of the results in order to more accurately figure out in which features/properties the two approaches differentiate.

Fig. 8 and Fig. 9 show the resulting Pareto fronts for two scenarios of 100 and 500 simulation budget, respectively. The set of algorithm parameters involved and their values are the ones previously shown in Table III. It should be pointed out again that the number of generations (or eras) has been chosen so that only the given amount of different simulations is actually performed. Thus, when accounting these simulations only unique simulations have been considered, removing repetitions and unfeasible configurations.

Low budget Pareto sets are shown in Fig. 8. Note that with such a small number of configurations being investigated, the two sets are often overlapping, being also strongly dependent upon the application being considered. These results are obtained in about 5 eras and demonstrate that, with such a limited budget, PS can compete with genetic approach while even discovering very different Pareto fronts. Considering the extended simulation budget as depicted in Fig. 9, we can qualitatively observe that even if PS does not strictly outperforms GA in terms of dominance, it shows in some cases Pareto fronts whose extension is larger. We can intuitively relate this to the "novelty-based" score system, so that instead of focusing on getting better points inside a given range PS tries to enlarge the range itself, resulting in less dense but more extended Pareto fronts. Of course we cannot classify this different behavior as being strictly better or worse when compared to the genetic one: other design factors should be evaluated on a case-by-case basis, e.g. the desired granularity of the results, objective range constraints or estimated error of the adopted models.

Fig. 8. Pareto fronts found by PS and GA for a fixed budget of 100 configurations.

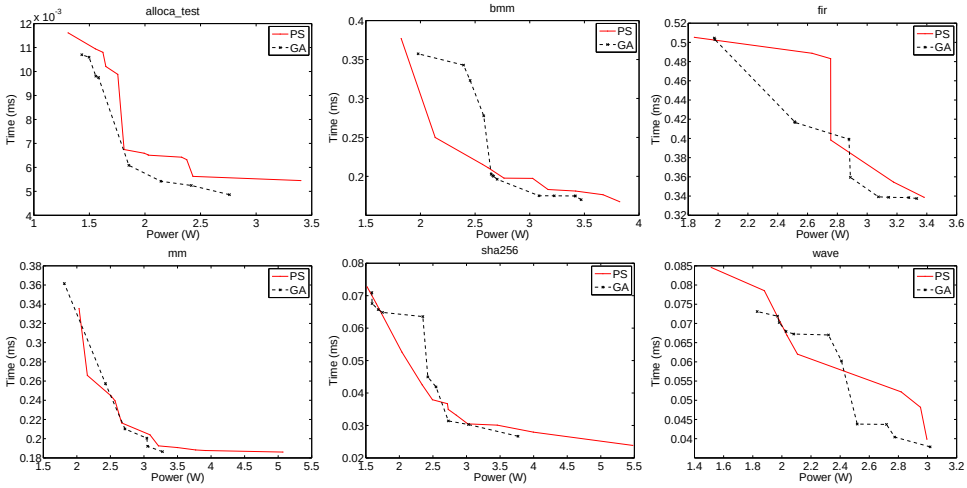
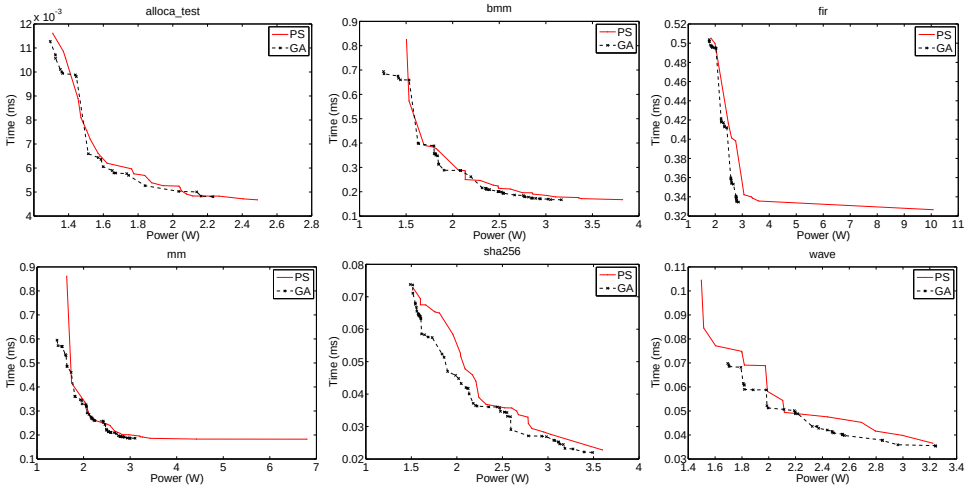


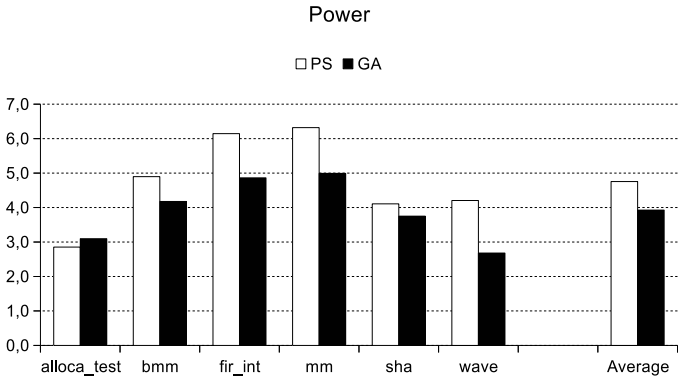
Fig. 9. Pareto fronts found by PS and GA for a fixed budget of 500 configurations.



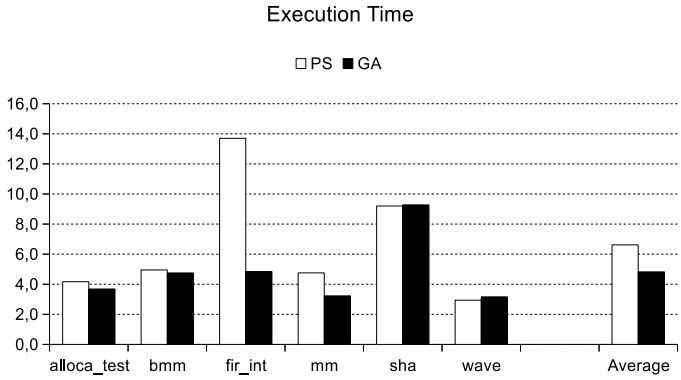
To better evaluate results from a different perspective, we adopt the *variation range* and the *average absolute dispersion error*, the two quantitative metrics described and motivated in Sec. 5.4. In particular, a comparison between the variation ranges of both power dissipation and execution time, obtained by PS and GA for different benchmarks, is shown in Fig. 10(a) and (b), respectively. As it can be observed, the PS exploration provides solutions which fall on a range that is, on average, wider than the one provided by a GA exploration for power dissipation and execution time, respectively.

Fig. 10(c) shows the average normalized absolute dispersion errors for different benchmarks for PS and GA exploration. As it can be observed, GA shows a tendency in discovering more uniformly dispersed points (i.e. higher values in the chart). These results, together with the variation ranges of Fig. 10(a) and (b) confirm the density vs. extension trade-offs when comparing PS and GA approaches. We highlight that PS shows a stronger *uniform convergence* (discussed in Sec. 5.4), i.e.

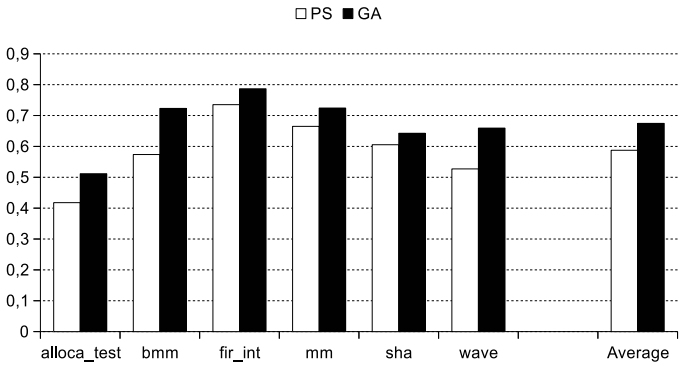
Fig. 10. (a)(b) Variation range obtained by PS and GA for different benchmarks and average absolute dispersion errors (c).



(a)



(b)



(c)

it provides a more extended Pareto front, avoiding the designer to neglect some important options, and offering a wider view of the potentials of the designed device.

Finally, it should be pointed out that the final output of PS execution is not only the Pareto-optimal set of configurations, but also a partitioning of the parameter space in more or less innovative regions, obtained as a result of the merge/splitting operators described in Sec. 4. Exploiting the information carried out by this partitioning could be an interesting added-value of PS to be investigated in future developments of the proposed strategy.

6. CONCLUSIONS

In this work we presented PS, a multi-objective strategy which introduces the concept of Parameter Space representation of Pareto Front. A case study of a VLIW architecture involving strong hardware/software dependencies has been analyzed to evaluate the PS effectiveness with different amounts of simulations budget. A qualitative/quantitative comparison against a widespread multi-objective genetic approach showed that the proposed strategy can result in a different distribution of Pareto points which balances solutions granularity and improved Pareto extension.

REFERENCES

Santosh G. Abraham, B. Ramakrishna Rau, and Robert Schreiber. 2000. *Fast Design Space Exploration Through Validity and Quality Filtering of Subsystem Designs*. Technical Report HPL-2000-98. HP Laboratories Palo Alto.

Giuseppe Ascia, Vincenzo Catania, Alessandro G. Di Nuovo, Maurizio Palesi, and Davide Patti. 2011. Performance evaluation of efficient multi-objective evolutionary algorithms for design space exploration of embedded computer systems. *Applied Soft Computing* 11, 1 (2011), 382 – 398. DOI: http://dx.doi.org/10.1016/j.asoc.2009.11.029

Giuseppe Ascia, Vincenzo Catania, and Maurizio Palesi. 2002. Tuning Methodologies for Parameterized Systems Design. In *International Workshop on System-on-Chip for Real-Time Applications*. Banff, Canada.

Giuseppe Ascia, Vincenzo Catania, Maurizio Palesi, and Davide Patti. 2003. EPIC-Explorer: A Parameterized VLIW-based Platform Framework for Design Space Exploration. In *First Workshop on Embedded Systems for Real-Time Multimedia (ESTIMedia)*. Newport Beach, California, USA, 65–72.

P.S. Borkar and A.R. Mahajan. 2014. Types and applications of Parallel Genetic Algorithm. *Internation Journal of Advanced Research in Computer Science and Software Engineering* 4, 4 (2014).

Erick Cantú-Paz. 1998. A Survey of Parallel Genetic Algorithms. *Calculateurs Paralleles* 10 (1998).

Vincenzo Catania, Andrea Araldo, and Davide Patti. 2014. Complete results of Paramspace tests. (2014). http://ricerca.dieei.unict.it/PS_TEST/

Vincenzo Catania, Maurizio Palesi, and Davide Patti. 2008. Reducing Complexity of Multiobjective Design Space Exploration in VLIW-based Embedded Systems. *ACM Trans. Archit. Code Optim.* 5, 2, Article 11 (Sept. 2008), 33 pages. DOI: http://dx.doi.org/10.1145/1400112.1400116

Carlos A. Coello Coello, David A. Van Veldhuizen, and Gary B. Lamont. 2002. *Evolutionary Algorithms for Solving Multi-Objective Problems*. Vol. 5. Kluwer Academic Publishers.

Matej Črepinšek, Shih-Hsi Liu, and Marjan Mernik. 2013. Exploration and exploitation in evolutionary algorithms: a survey. *ACM Computing Surveys (CSUR)* 45, 3 (2013), 35.

Michael Dellnitz, Oliver Schütze, and Thorsten Hestermeyer. 2005. Covering Pareto sets by multilevel subdivision techniques. *Journal of optimization theory and applications* 124, 1 (2005), 113–136.

M Dellnitz and K Witting. 2009. Computation of robust Pareto points. *International Journal of Computing Science and Mathematics* 2, 3 (2009), 243–266.

Agoston E Eiben and CA Schippers. 1998. On evolutionary exploration and exploitation. *Fundamenta Informaticae* 35, 1 (1998), 35–50.

William Fornaciari, Donatella Sciuto, Cristina Silvano, and Vittorio Zaccaria. 2001. A Design Framework to Efficiently Explore Energy-Delay Tradeoffs. In *9th. International Symposium on Hardware/Software Co-Design*. Copenhagen, Denmark, 260–265.

Tony Givargis, Frank Vahid, and Jörg Henkel. 2002. System-Level Exploration for Pareto-Optimal Configurations in Parameterized System-on-a-Chip. *IEEE Transactions on Very Large Scale Integration Systems* 10, 2 (Aug. 2002), 416–422.

D. Kahneman. 1973. *Attention and Effort*. Prentice-Hall.

Vinod Kathail, Michael S. Schlansker, and B. Ramakrishna Rau. 2000. *HPL-PD Architecture Specification: Version 1.0*. Technical Report. Compiler and Architecture Research HP Laboratories Palo Alto HPL-93-80.

Joshua Knowles, Lothar Thiele, and Eckart Zitzler. 2006. *A tutorial on the performance assessment of stochastive multiobjective optimizers*. Technical Report 214. Computer Engineering and Networks Laboratory, ETH Zurich.

Vilfredo Pareto. 1896. *Cours D'Economie Politique*. Vol. I–II. F. Rouge, Lausanne.

Davide Patti, Maurizio Palesi, and Vincenzo Catania. 2014. Merging Compilation and Microarchitectural Configuration Spaces for Performance/Power Optimization in VLIW-Based Systems. In *Modern Trends and Techniques in Computer Science*, Radek Silhavy, Roman Senkerik, Zuzana Kominkova Oplatkova, Petr Silhavy, and Zdenka Prokopova (Eds.). Advances in Intelligent Systems and Computing, Vol. 285. Springer International Publishing, 203–212. DOI: http://dx.doi.org/10.1007/978-3-319-06740-7_18

G. Sperling and E. Weichselgartner. 1995. Episodic Theory of the Dynamics of Spatial Attention. *Psycological Review* 102 (1995), 503–532.

Trimaran 2010. An Infrastructure for Research in Instruction-Level Parallelism. http://www.trimaran.org/. (2010).

Thomas Weise, Raymond Chiong, and Ke Tang. 2012. Evolutionary optimization: Pitfalls and booby traps. *Journal of Computer Science and Technology* 27, 5 (2012), 907–936.

WSTS 2013. World Semiconductor Trade Statistics Bluebook. http://www.wsts.org/. (2013).

Eckart Zitzler, Kalyanmoy Deb, and Lothar Thiele. 2000. Comparison of Multiobjective Evolutionary Algorithms: Empirical Results. *Evolutionary Computation* 8, 2 (2000), 173–195.

Eckart Zitzler, Marco Laumanns, and Lothar Thiele. 2001. SPEA2: Improving the Performance of the Strength Pareto Evolutionary Algorithm. In *EUROGEN 2001. Evolutionary Methods for Design, Optimization and Control with Applications to Industrial Problems*. Athens, Greece, 95–100.

Eckart Zitzler, Lothar Thiele, Marco Laumanns, Carlos M. Fonseca, and Viviane Grunert da Fonseca. 2003. Performance Assessment of Multiobjective Optimizers: An Analysis and Review. *IEEE Transactions on Evolutionary Computation* 7, 2 (April 2003), 117–132.