Efficient Parallel Multi-Objective Optimization for Real-time Systems Software Design Exploration

Rahma Bouaziz*, Laurent Lemarchand†, Frank Singhoff†, Bechir Zalila*, Mohamed Jmaiel*‡
* ReDCAD Laboratory, University of Sfax, ENIS, B.P. 1173, 3038 Sfax, Tunisia
†Lab-STICC Laboratory, University of Bretagne Occidentale, UMR CNRS 6285, F-29200 Brest, France
‡Digital Research Center of Sfax, B.P. 275, Sakiet Ezzit, 3021 Sfax, Tunisia
rahma.bouaziz@redcad.org, {laurent.lemarchand, singhoff}@univ-brest.fr, {bechir.zalila, jmaiel.mohamed}@enis.tn

***Abstract*—Real-time embedded systems may be composed of a large number of time constrained functions. When such systems are implemented on top of multitasks real-time operating systems (RTOS), the functions have to be assigned to tasks of the target RTOS. This is a challenging work due to the large number of valid candidate functions to tasks assignment solutions. Moreover, the impact of the assignment on the system performance criteria (often conflicting) should be taken into account in the architecture exploration. The automation of the software design exploration by the use of metaheuristics such as multi-objective evolutionary algorithm (MOEA) is a suitable way to help the designers. Indeed, MOEAs approximate near-optimal alternatives at a reasonable time when compared to an exhaustive and exact search method. However, for large-scale systems (i.e having a huge number of functions) even a MOEA method is impractical due to the increased time required to solve a problem instance. This may raise a threat to the scalability of the software design exploration method. To tackle this problem, we present in this article a parallel implementation of the Pareto Archived Evolution Strategy (PAES) algorithm used as a MOEA for the software design exploration. The proposed parallelization method is based on the well-known master-slave paradigm. Additionally, it involves a new selection scheme in the PAES algorithm. Results of experimentations provide evidence that, on one hand, the parallel approach can considerably speed up the design exploration and the optimization processes. On the other hand, the proposed selection strategy improves the quality of obtained solutions as compared to the original PAES selection schema.**

Keywords*—Real-Time Embedded Systems, Design exploration, Multi-Objective Optimization, PAES, Parallelism, Master-Slave Model

I. Introduction

Real-time embedded systems are frequently designed according to multi-tasked architectures that have timing constraints to meet. Usually, the designer has to assign the functions of the real-time embedded system to a set of tasks. The verification of the timing constraints (referred to as schedulability analysis) is then performed at the design level. The design of software architectures of these systems involves dealing with a number of competing performance criteria. Improving one criterion leads to the degradation of another. Therefore, software architects must explore several architecture alternatives in order to design architectures that meet at best the trade-off between performance criteria. Due to the major increase in complexity and size of today real-time software architectures, the exploration and the selection are time-consuming, complex and error-prone tasks. This complexity motivates the automation of the design exploration process to help software designers.

The design space exploration for real-time embedded systems can be addressed as a multi-objective optimization (MOO) problem [1], [2], [3], [4]. Multi-Objective Evolutionary Algorithms (MOEAs) are metaheuristics that allow designers to find suboptimal (or near-optimal) solutions in a reasonable time when exact methods fail to handle large scale problems due to computing resource requirements.

In a previous work [1], we have addressed the problem of assigning functions to tasks in the design of software architectures of real-time systems. A MOO approach was applied to consider the trade-off between the number of preemptions and the overall laxity of tasks. The number of preemptions increases with the granularity of the assignments whereas the tasks laxities decrease when more functions are assigned to one task. We have chosen the Pareto Archived Evolution Strategy (PAES) [5] as a MOEA to investigate this trade-off.

a) Problem Statement: Design exploration problems involves multiple degrees of freedom in the exploration process, with a huge size of the design search space, and multi-objective evaluation complicates comparison between those solutions. MOEA methods usually require to explore huge portions of the search space because they should identify a (often large) set of Pareto-optimal solutions and not a single optimum like with single-objective optimization methods. This would result in a very large number of solution evaluations. Moreover, many real-world MOO problems, as the design exploration of real-time systems, have computationally expensive objective functions and constraints verification. This is the case when the objective functions and the constraints are derived from schedulability analysis. These two factors hinder MOEAs (e.g PAES) from solving efficiently such real-world problems.

b) Contributions of this Article: In this article, we propose an implementation of the PAES algorithm based on the well-known master-slave parallel paradigm [6]. With this coarse-grained approach, multiple candidate solutions are processed in parallel for checking constraints and evaluating objective functions. We show how this approach improves the efficiency of the architecture exploration with MOEAs for large-scale problem instances. Both the computational time and the quality of the resulting solution sets are improved. The former is due to the asynchronous parallel evaluations and the

latter is due to the use of a new selection scheme in PAES. This parallel scheme could also be applicable paired with other optimization criteria and design evaluation methods.

The remainder of the article is organized as follows. Section II introduces the multi-objective optimization (MOO), the PAES algorithm and gives an overview of our design exploration method to assign the functions of a real-time embedded system to tasks. Section III presents our approach, corresponding to the parallel multi-objective search model for the PAES algorithm. Section IV shows experimental results. Section V presents the related work and section VI concludes the article.

II. Background and Context

This section presents the basic concepts of the Multi-Objective Optimization (MOO) and introduces the PAES algorithm as the MOO technique used in our design exploration method. Then it gives an overview of our PAES-based methodology for determining the best trade-offs in the software design of a real-time application according to some competing performance criteria.

A. Multi-Objective Optimization (MOO)

Many engineering problems involve more than one competing objectives that need to be optimized simultaneously with respect to a set of constraints. Then, we have to evaluate the performance of candidate solutions in more than one objective [6]. Finally, the outcome of running MOO is a single or a set of solutions, with each solution representing a trade-off between objectives.

A single solution that yields the best value for all objectives rarely exists. Instead, a set of alternative solutions called *non-dominated* (or *Pareto optimal*) solutions are searched for. A solution is Pareto-optimal with respect to a set of objectives if there exists no other solution in the search space that improves on all of the objectives at once. These solutions constitute the *Pareto Set*. As depicted in Figure 1, the associated objective vectors correspond to the *Pareto Front* which represents the best trade-off set for the considered objectives.

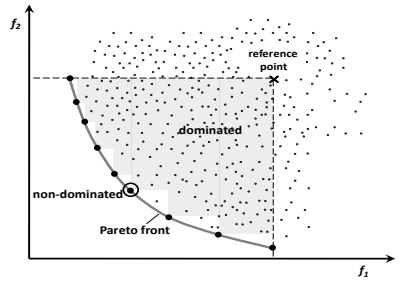

Fig. 1: Pareto front for two minimization objectives

Many metaheuristic algorithms that seek to approximate the Pareto set have been developed to solve MOO problems [7]. The key points for those algorithms are their accuracy (how close to the Pareto front are the values of the solutions they provide) and their efficiency (are the solutions numerous and well spaced over the whole Pareto front). A large amount of MOO techniques is derived from Multi-Objective Evolutionary Algorithms (MOEA). Evolutionary algorithms are meta-heuristic optimization algorithms inspired from nature, e.g. particle swarm (PSO), ant colony, simulated annealing, genetic algorithms (GA). The latter use biology-inspired mechanisms like *mutation* and *crossover* in order to refine a set of candidate solutions iteratively.

B. PAES MOEA

The Pareto Archived Evolution Strategy (PAES) [5] is a MOEA technique using archiving. It manipulates a single solution as opposed to other methods used for MOO such as GA or PSO. This is a key point in running very time-consuming evaluation functions such as those stemmed from the scheduling analysis. The sequential PAES schema is outlined in the pseudo code of algorithm 1.

Algorithm 1: Sequential form of PAES Algorithm

```
1  begin
2      Generate initial random solution c;
3      Evaluate c and add it to the archive A;
4      repeat
5          Mutate c to produce a new candidate solution
           m;
6          Evaluate m;
7          if c dominates m then
8              Discard m;
9          else if m dominates c then
10             Replace c with m;
11             Add m to the archive;
12         else if m is dominated by any member of A
           then
13             Discard m;
14         else
15             Apply test (c,m,A) to determine which
               becomes the new current solution and
               whether to add m to A;
16         end if
17     until termination condition is satisfied;
18 end
```

The PAES algorithm is based on a (1+1) evolution strategy which means that it maintains a single current solution (*parent*) and, at each iteration, generates a single new candidate (*offspring*). It makes it through a random *mutation*. This algorithm is confined to a local search, i.e. it performs only a small change (*mutation*) operator that moves from a current solution to a nearby neighbour. The mutation procedure is also specific for each problem, and depends on the way the solution is represented into a *chromosome*. The current solution is replaced at each iteration by its mutated offspring if the latter dominates or is in a less crowded region than its parent. Otherwise, the next iteration is realized keeping the same current individual as a basis for mutation. PAES maintains a list of some non-dominated solutions called archive used as reference set with respect to which each new candidate is being compared (lines 11,12).

C. Software Design Exploration Method for Real-Time Applications Based on PAES

In our research, we focus on real-time systems design and implementation. These systems may consist of a large number of time constrained high-level functions. During software architecture design, these functions must be assigned to tasks that will run the functions on the top of a RTOS. The main objective of our proposed method [1] is to guide the designer by providing solutions of functions to the tasks assignment, referred to as design (or architecture) alternatives. An optimal design from a scheduling point of view might be an architecture that minimizes preemptions while maximizing the overall laxity. A preemption occurs when a higher priority task τ_i is released during the execution of a lower priority task τ_j and when the scheduler interrupts the execution of τ_j to allow the task τ_i to run. The laxity of a task is the amount of time between the end of the task execution and its deadline. The number of preemptions increases with the granularity of the assignments whereas the tasks laxities decrease when more functions are assigned to one task. Therefore, these two goals are conflicting: one architecture may achieve high performance (maximum overall laxity) at high cost (high number of preemptions) or vice versa. This is the main motivation to rely on an iterative multi-objective search and optimization process by the mean of a MOEA, notably PAES. This proposal allows us to explore the design space for Pareto-optimal solutions in a reasonable time and then to help designers to reach a final design.

Figure 2 gives an overview of our design exploration methodology. The entry point is the function specification of a real-time system. From this specification, a first architecture is proposed so that each functional block is assigned to a single task. A scheduling analysis is performed on this initial architecture by the Cheddar scheduling simulator [8]. This analysis is achieved through a simulation-based schedulability test on the hyperperiod (the least common multiple of task periods). According to the addressed context (fixed priority and preemptive scheduling policy, periodic synchronous and independent tasks, and uniprocessor platform), the hyperperiod corresponds to the feasibility interval [9]. If the initial task set is schedulable then it will be considered as the initial solution to the PAES algorithm. Otherwise, the designer must adjust the timing parameters of the functional specification.

Once the initial solution is defined, we come to the multi-objective design exploration and optimization part. The latter involves the execution of the PAES algorithm. At each iteration, (1) an alternative design (mutated solution) is generated from the current solution by small random changes in the assignment of functions to tasks. The new candidate architecture should fulfil the timing constraints (i.e. schedulability of the task set) as well as the functions to tasks assignment constraints (2). This alternative design is evaluated according to the two performance criteria computed by Cheddar, i.e. the preemption cost and the overall laxity (3). Then, the optimization steps of PAES are performed such as the ranking, the archiving and the selection of the current solution for the next iteration (4). These steps are iterated until the termination condition of PAES is reached, e.g. number of iterations. The result is a set of schedulable alternative architectures that approximates the Pareto set.

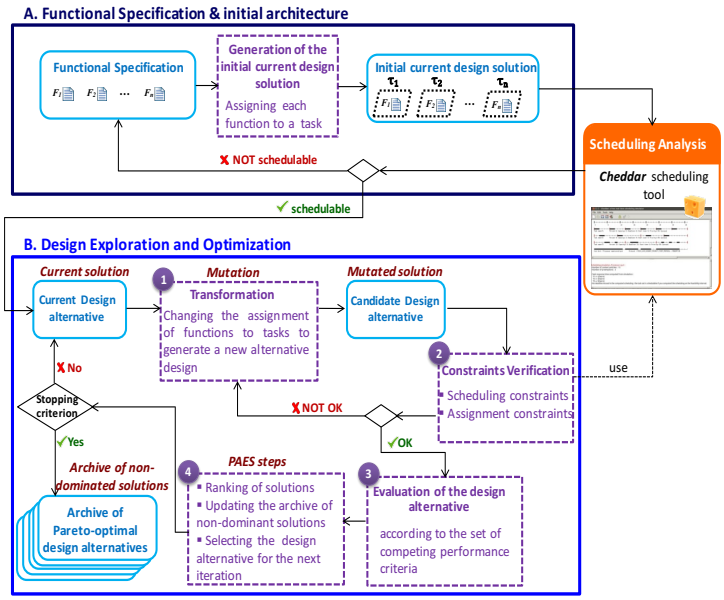

Fig. 2: Overview of the PAES-based Software Design Exploration Method

III. Towards Fast and More Efficient Software Design Exploration

In [1], we conducted experiments to assess the proposed design exploration method described in section II-C. For such experiments beyond 50 functions, the execution of our method takes several hours. This is due to the significant time spent to process the scheduling analysis and objectives evaluation of each investigated solution. Thus, to speed up the exploration and the optimization without degrading the quality of solutions, we propose a parallel implementation of our design exploration method.

Parallelizing the MOEA technique (i.e PAES) arises as a possible way of facing this drawback and obtain the results in a reasonable amount of time. The parallelization of MOEA techniques is interesting in different situations: when the evaluation function is time-consuming, when some mutation process is time-consuming, when there is a local search (hybrid algorithm), or when there is a sub-population searching algorithm. We are directly concerned with the 2 former aspects, since our evaluations are time-consuming and the mutation process includes schedulability analysis. We propose a parallel implementation of the PAES algorithm based on the master-slave paradigm, with coarse-grained asynchronous tasks performing the mutation and evaluation of solutions. We not only aim to speed up the search but to hopefully improve the solution quality. In the remaining of this section, we first detail the parallel scheme proposed for PAES and then we present our new selection strategy adapted to the proposed parallel scheme.

A. Parallel PAES implementation: The Master-Slave Asynchronous Model

According to [6], three main parallel paradigms are used in the MOEA domain for splitting the computational load across

several processors: the Island model, the Diffusion model and the master-slave model. The two former paradigms are dedicated for sub-population searching problems. However, we deal with the parallelization of PAES algorithm that maintains a single current solution. Furthermore, the master-slave model is a simple parallel programming paradigm for optimization techniques. Therefore, the case we consider here is a master-slave asynchronous framework (Figure 3). This model is aimed at distributing the objective function evaluations of the candidate solutions on several slave processors while a single master processor maintains the central archive and runs the selection part of the MOEA. As stated by the master-slave paradigm, slaves are started by the master, and interact exclusively with it, receiving solutions to process and sending back mutated solutions and associated objective values. The additional cost of the parallelization, that is mainly due to solution data transfers must be reasonable as compared to the computation costs, since data for a solution consist only in a set of function indexes and a few objective function values. For simplicity, the proposed parallel asynchronous PAES is called `PA-PAES` in the rest of the article.

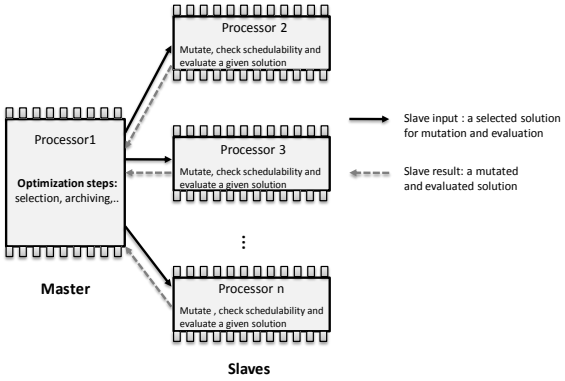

Fig. 3: Master-Slave Model

Generally speaking, this algorithmic framework needs the following components: (1) a selection strategy (based on crowded region with dominance, or indicator based), (2) a mutation and an evaluation procedures and eventually, (3) a local search procedure that outputs one or more neighbours. The selection strategy is explained in the next section, derived from the PAES original one. The mutation, evaluation and associated schedulability analysis procedures are the same as in our previous work [1]. The algorithm also takes as input a number S of slaves and a number I of iterations (or other more sophisticated convergence criteria for loop termination). We only use I for lack of simplicity. The master and slave computations in the parallel PAES implementation are described in Algorithms 2 and 3 respectively. Parallel evaluations (or other computing process) are realized asynchronously. This means that the master processor does not have to wait for all of the evaluations from the other processors (slaves). The results of the slave computations are taken into account as soon as they become available: when a slave sends an evaluated solution, the master archives it using the original PAES approach, and goes on the optimization by selecting a new solution to send to the slave for mutation and evaluation. This way, we save the overall computation time and resource usage, especially as the processing time of different solutions by slaves may be different.

Algorithm 2: Parallel Asynchronous PAES: Master algorithm

```
1  begin
2      start S slaves;
3      send to each of them a seed unevaluated solution;
4      iteration := 1;
5      while iteration ≤ I do
6          receive a set or a single evaluated solution(s)
           from any slave s;
7          compare it/them with the archive A;
8          update the archive A with non-dominated
           solutions;
9          if iteration < I − S then
10             select a candidate solution from A;
11             send it to idle slave s;
12         end if
13         iteration := iteration + 1;
14     end while
15     terminate slaves;
16     output A;
17 end
```

Algorithm 3: Parallel Asynchronous PAES: Slave algorithm

```
1  begin
2      while not terminated do
3          receive a candidate solution from master;
4          mutate it;
5          evaluate it;
6          if local search then
7              generate and evaluate neighbours;
8              discard locally dominated neighbours;
9          end if
10         send back to the master the evaluated
           solution(s);
11     end while
12 end
```

The algorithm needs a selection strategy for the search procedure. In the original PAES, the selection strategy, referred to as `local selection`, is working as follows: admitting the mutated solution as current solution or keeping current solution for the next iteration. Since selection (by the master) is overlapped with the slave computations that are expected to be time consuming, we can spend more time in the selection process, looking to the whole archive, instead of using solely the received solution from the slave as the potential candidate for the next iteration. Our new selection strategy is described in the next subsection.

B. Global Archive Selection Strategy

As opposed to the sequential version, there is no current solution maintained by the master process, but the solution sent to a slave for mutation and evaluation (lines 9-10, Algorithm 2) is selected from the archive as follows. We randomly choose a selection criterion among the followings:

- random criterion: a random solution from the archive is selected, to ensure the exploration process.
- one of the objective functions: one of the archive solutions which is within the 10% best for this objective is picked. This is an exploitation criteria for the most promising solutions.
- crowding criterion: a solution from the archive within the less crowded area is chosen. This helps in maintaining the diversity of the front.

These choices are equiprobable. The proposed selection strategy (referred to as `global selection`) is designed to balance between exploration and exploitation of the solution space.

IV. Experiments and Results

In this section, we present experiments that assess the efficiency of the above proposals when applied to our design exploration problem. We perform two evaluations. The first evaluation aims at investigating the efficiency (in terms of quality of fronts) of PAES with the new global archive selection strategy as compared to the original PAES (with the local selection strategy) for different problem scales. The second evaluation is performed in order (1) to check that the parallelization does not induce lost on fronts quality as compared to the sequential version, (2) and to assess its efficiency regarding to the temporal behavior.

In order to perform experiments, we need different test cases (with different sizes) of our design exploration problem. We propose and implement a function set generator. Experiments are conducted on a SMP 48 processors machine running Linux OS. Our method is implemented in Ada, with parallelization based onto the Ada concurrency features (tasking, synchronization and communication with Rendezvous or protected objects, ...). The developed software is available in the Cheddar Repository[1].

In the following, first we describe our test case generator (section IV-A). Then, we give an overview of the performance metrics used for measuring the results and assessing our proposals (section IV-B). In sections IV-C and IV-D, we present (i) experiments protocol, (ii) the results and (iii) their analysis of the two conducted evaluations.

A. Test Case Generator for the Design Exploration problem

In order to perform the experiments, we apply our PAES-based functions to tasks assignment method on synthetically generated function sets. Function periods are uniformly distributed between a set of a maximum of 10 different periods by function set following Goossens and Macq method [10], ensuring that the scheduling simulations have to be run on a limited feasibility interval. Processor utilization factors for each function F_i are tuned with the UUnifast algorithm [11], so that the sum of function utilizations is equal to the desired overall processor utilization factor for the function set. The overall processor utilization is fixed at 80% for all the experiments. Furthermore, the function deadlines are implicit, i.e. set to be equal to the periods. The capacities are set based on the generated periods and processor utilization factors.

[1] http://beru.univ-brest.fr/svn/CHEDDAR/trunk/src/framework/paes

B. Performance Metrics

In our evaluations, we used two metrics in order to evaluate the performance of the proposed method : (1) the speed-up metric (2) and quality of solution sets through the hypervolume metric.

Speed-up: is a classical way for measuring the efficiency of parallel algorithms [12]. The speed-up S_p is defined as $S_p = \frac{T_1}{T_p}$, with p the number of processors (i.e. slaves) involved in the parallel computations and T_x the execution time with x processors.

Hypervolume Indicator: In MOO techniques, the comparison between solution sets (fronts) is generally difficult since one set is not decidedly superior to another. Many *unary* metrics exist [6], that map a front to a single value thereby allowing us to easily compare the quality of produced fronts. These metrics take into account both the closeness of the obtained solutions to the optimal set (accuracy) and their spread across objective space (diversity). One of these metrics is the $hypervolume$ indicator [6]. Given a set of solutions and associated front (points in objective space), it computes a reference point dominated by all of the points of the front (see Figure 1). Then for each subset of points of interest (e.g produced by different algorithms), it provides as hypervolume the area bounded by the reference point and the considered subset (light grey area in the Figure 1). The subset ($front_i$) with the largest hypervolume is likely to present the best set of trade-offs.

For a test case, in order to compare results from different algorithms (PAES+Global Selection; PAES+Local selection; PA-PAES implementation and PAES-Sequential version), we compute the best and the worst results obtained (considering non dominated solutions) for each objective (min_1, min_2), (max_1, max_2) over the set of solutions provided by all of the runs of each algorithm $\bigcup_{a \in runs} front_a$. Moreover, in order to allow the objectives to contribute approximately equally, values are normalized according to a linear normalization technique defined by [13]:
$(x, y) \in front_a \implies (x', y') \in norm_a$
with $x' = \frac{x - min_1}{max_1 - min_1}$ and $y' = \frac{y - min_2}{max_2 - min_2}$.
The reference point used is $(1 + \epsilon, 1 + \epsilon)$, in order to ensure that every point in $norm$ is dominated, with a small value of ϵ, e.g 0.001. The resulting hypervolume computed for one front can grow up to $1 + \epsilon^2$ (when the front contains a single value, dominating all other values in fronts considered for comparison).

C. Evaluation 1: Global Archive Selection Method Evaluation

This evaluation aims at qualitatively comparing global selection regarding to local selection. A number of experiments were run in order to investigate the quality of solution sets for different function set sizes that range from 20 to 100 by step of 20 functions. For each size, we generate 5 different test cases (function sets) and each test case is processed with the following PAES versions: (1) Local selection with sequential PAES, (2) Local selection with PA-PAES$_4$ (4 slaves), (3) Global selection with sequential PAES, (4) Global selection with PA-PAES$_4$ (4 slaves). Each of these versions is run 5 times because of the stochastic nature of PAES. The number of PAES iterations for each run is fixed at 2000. For each

test case, we compute the average hypervolume value over the 5 runs for each of the considered PAES versions. Figure 4 reports, for each size, the hypervolume average value over all test cases of each PAES version.

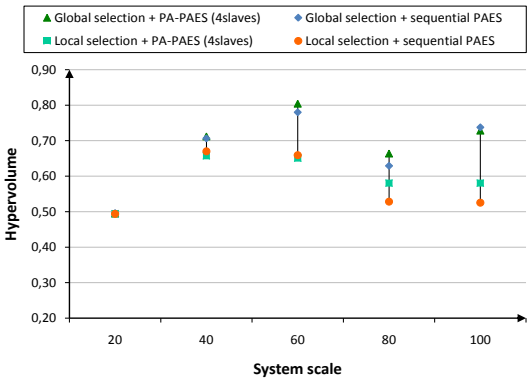

Fig. 4: Hypervolume comparison between the global selection and the local selection in sequential-PAES and PA-PAES$_4$ (with 4 slaves) implementations for different system scales (function set sizes)

This figure shows that the best hypervolume values are obtained by the PA-PAES$_4$ with the global selection strategy for the most scales. In addition, we can observe that the efficiency of the global selection with respect to the local selection is more significant for test cases with larger scales ($\geq 60 functions$), i.e. larger space search and problems. This conjecture is reinforced by the average hypervolume improvement between the global selection and local selection in both sequential PAES and PA-PAES$_4$, which are reported in table I. For example, for 20 functions test cases, the improvement is practically negligible. This can be explained by the fact that all versions lead to the same results for small-scale test cases. Moreover, for PA-PAES$_4$, we also computed the speed-ups for the performed experiments. The average speed-up for the overall experiments is about 3.8 which represents a promising speed-up value for a 4 slaves parallel version.

TABLE I: The average hypervolume improvement between global selection and local selection in sequential and parallel versions

Function set size	Sequential version Avg HV improvement (%)	Parallel version Avg HV improvement (%)
20	0.43	0.06
40	5,15	7,64
60	15,5	19.08
80	16.03	12.53
100	28.79	20.3

D. Evaluation 2: Parallel Method Evaluation

The second evaluation is performed in order to assess the efficiency of the parallel implementation (PA-PAES) comparing to the sequential one. We have run a set of experiments in a range of different slaves (4, 8 and 16) to observe speed-up of the parallel approach. These experiments are achieved on three problem scales, i.e. test cases of 50, 80 and 100 function sets. The selection strategy is set to the proposed global selection. For each scale, we generate 5 different test cases thanks to our function set generator. Each test case is processed 5 times by the sequential version as well as by the PA-PAES$_4$, PA-PAES$_8$ and PA-PAES$_{16}$. All these versions perform 2000 iterations.

Figure 5 provides the average speed-up against the number of slaves involved in the parallel computation. In this figure, we can notice that even if they are not linear, speed-ups increase regularly with the number of involved slaves, reaching up to 11.7 for 16 slaves with 100 functions test cases.

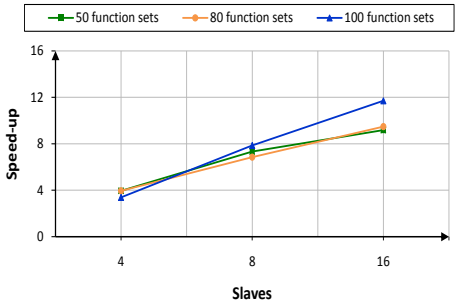

Fig. 5: Speedup of the parallel PAES against the number of slaves involved in the parallel computation

As shown in figure 6 quality is preserved, with almost roughly constant hypervolume values from 4 to 16 slaves.

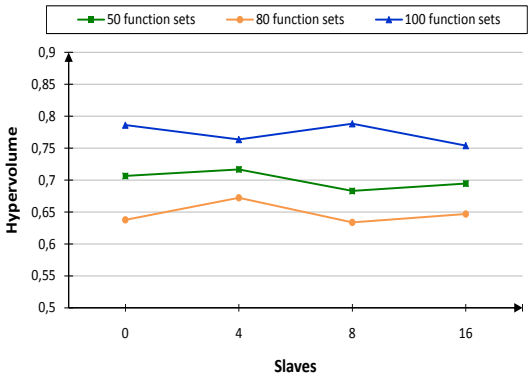

Fig. 6: Average Hypervolume against the number of slaves involved in the parallel computation

To conclude, the results of this evaluation provide evidence that the new implementation of our design exploration method allows us to handle more tractably large-scale problem instances (i.e. to ensure scalability).

V. Related Work

Our work comes within the scope of design exploration using multi-objective optimization techniques. Some research contributions have been developed in this field. Most of them are based on MOEAs. In [2], the authors proposed a method that explores architecture alternatives for real-time embedded systems, in order to produce architectures that fulfil at best a set of conflicting non-functional properties stemmed from the requirements of the addressed systems. This method is based on model transformations compositions and MOEA by means of the NSGA-II multi-objective optimization strategy. Koziolek

et al. [3] developed a framework called Peropteryx. It assists software architects during the design stage to approximate Pareto-optimal architectures by using NSGA-II. AQOSA [4] is another generic framework that provides an automated design exploration process based on a set of MOEAs namely NSGA-II, SPEA2, and SMS-EMOA. The proposed framework is evaluated through both a small-scale and a large-scale case study. Experimental results for both case studies show that all used MOEA techniques take a considerable amount of time to generate results, even for a few number of iterations.

This motivates the use of parallelism of MOEA techniques in the context of design exploration problems. All of those MOEA techniques are population-based ones, evaluating a set of solutions at each iteration, before exploiting them to update the front and go to the next iteration. This implies a large computational effort when the architectural solutions are costly to evaluate, as stated by [4].

Unlike these methods and frameworks, we apply an asynchronous parallel algorithm that exploits a solution as soon as it is evaluated in order to guide the optimization process, and takes also benefit from parallelization.

Parallelization of metaheuristics for multi-objective optimization have been widely studied in [6], [14]. For example, [14] presents a survey of parallel algorithms for MOO techniques. Few works address the parallelization of PAES. The algorithm `ppaes` [14] is based on a sub-population search model. It works with an island-based approach as follows: Each process runs the PAES algorithm and maintains its own local archive of non-dominated solutions. Periodically, processes exchange solutions. When results from all processes become available, the last step consists in building the final Pareto-front by merging local results. Results of experiments of `ppaes` show that it is not suitable for MOO problems with very expensive computational objective functions like our specific design exploration problem. Considering distinct populations seems not computationally efficient for design optimization with costly evaluations. In [15], the authors propose another parallelization of PAES by running simultaneous parallel evaluations at each iteration, and comparing results against the current solution at the end of each iteration. In order to generate candidate solutions to be evaluated at an iteration, they consider a prediction tree taking into account the probability of a mutated solution to be better than its parent. As our local selection scheme, this work tries to avoid useless evaluations, but still handles solutions synchronously like a population instead of taking them into account asynchronously. Again and in contrary, we focus on an asynchronous parallel implementation for PAES.

VI. Conclusion and Future Work

In this article, we propose a parallel implementation of the Pareto Archived Evolution Strategy (PAES) algorithm used as a multi-objective evolutionary strategy. The proposed parallel multi-objective strategy is applied to the problem of functions to tasks assignment of a real-time system in order to improve the applicability of this method for large-scale real-time systems. This implementation involves a master-slave asynchronous formulation. Furthermore, we proposed and investigated a new selection strategy for the PAES algorithm. The proposed approach is particularly suitable for multi-objective problems with large and variable time objective function computation. It was evaluated by a set of experiments on our problem by synthetically generated test cases. Results show an improvement not only in the execution time (e.g. with 8 slaves, the average speed-up is about 7.8 for test cases of 100 functions) but also in the quality of solutions when compared to our previous implementation. In the future, we plan to improve the efficiency of the parallel search model for high number of processors by investigating the solution choice like in [15]. We also plan to explore the applicability of the approach with other architecture exploration optimization goals.

References

[1] R. Bouaziz, L. Lemarchand, F. Singhoff, B. Zalila, and M. Jmaiel, "Architecture exploration of real-time systems based on multi-objective optimization," in *Engineering of Complex Computer Systems (ICECCS), 2015 20th International Conference on*, Dec 2015, pp. 1–10.

[2] S. Rahmoun, E. Borde, and L. Pautet, "Automatic selection and composition of model transformations alternatives using evolutionary algorithms," in *Proceedings of the 2015 European Conference on Software Architecture Workshops*. ACM, 2015, p. 25.

[3] A. Koziolek, H. Koziolek, and R. Reussner, "Peropteryx: automated application of tactics in multi-objective software architecture optimization," in *Proceedings of the joint ACM SIGSOFT conference–QoSA and ACM SIGSOFT symposium–ISARCS on Quality of software architectures–QoSA and architecting critical systems–ISARCS*. ACM, 2011, pp. 33–42.

[4] R. Li, R. Etemaadi, M. T. Emmerich, and M. R. Chaudron, "An evolutionary multiobjective optimization approach to component-based software architecture design," in *Evolutionary Computation (CEC), 2011 IEEE Congress on*. IEEE, 2011, pp. 432–439.

[5] J. D. Knowles and D. W. Corne, "Approximating the nondominated front using the pareto archived evolution strategy," *Evolutionary computation*, vol. 8, no. 2, pp. 149–172, 2000.

[6] C. A. C. Coello, G. B. Lamont, and D. A. V. Veldhuizen, *Evolutionary Algorithms for Solving Multi-Objective Problems*. Springer Science & Business Media, 2007.

[7] S. Bandyopadhyay and S. Saha, "Some single- and multiobjective optimization techniques," in *Unsupervised Classification*. Springer Berlin Heidelberg, 2013, pp. 17–58.

[8] F. Singhoff, A. Plantec, and P. Dissaux, "Can we increase the usability of real time scheduling theory? the cheddar project," in *International Conference on Reliable Software Technologies*. Springer, 2008, pp. 240–253.

[9] J. Goossens, E. Grolleau, and L. Cucu-Grosjean, "Periodicity of real-time schedules for dependent periodic tasks on identical multiprocessor platforms," *Real-time systems*, 2016.

[10] J. Goossens and C. Macq, "Limitation of the hyper-period in real-time periodic task set generation," in *In Proceedings of the RTS Embedded System (RTS01)*, 2001.

[11] E. Bini and G. C. Buttazzo, "Measuring the performance of schedulability tests," *Real-Time Systems*, vol. 30, no. 1-2, pp. 129–154, May 2005.

[12] E. Alba and M. Tomassini, "Parallelism and evolutionary algorithms," *Evolutionary Computation, IEEE Transactions on*, vol. 6, no. 5, pp. 443–462, 2002.

[13] C. M. Fonseca, J. D. Knowles, L. Thiele, and E. Zitzler, "A tutorial on the performance assessment of stochastic multiobjective optimizers," in *Third International Conference on Evolutionary Multi-Criterion Optimization (EMO 2005)*, vol. 216, 2005, p. 240.

[14] F. Luna, A. J. Nebro, and E. Alba, "Parallel evolutionary multiobjective optimization," in *Parallel Evolutionary Computations*. Springer, 2006, pp. 33–56.

[15] J. C. Calvo, J. Ortega, and M. Anguita, "Comparison of parallel multi-objective approaches to protein structure prediction," *The Journal of Supercomputing*, vol. 58, no. 2, pp. 253–260, 2011.