An OpenMP Parallel Genetic Algorithm for Design Space Exploration of Heterogeneous Multi-processor Embedded Systems

Conference Paper · January 2020

DOI: 10.1145/3381427.3381431

4 authors, including:

Vittoriano Muttillo
Università degli Studi dell'Aquila
26 PUBLICATIONS **71** CITATIONS

SEE PROFILE

Paolo Giammatteo
Università degli Studi dell'Aquila
13 PUBLICATIONS **23** CITATIONS

SEE PROFILE

An OpenMP Parallel Genetic Algorithm for Design Space Exploration of Heterogeneous Multi-processor Embedded Systems

Vittoriano Muttillo, Paolo Giammatteo, Giuseppe Fiorilli, Luigi Pomante
University of L'Aquila, L'Aquila, Italy
{vittoriano.muttillo,paolo.giammatteo,luigi.pomante}@univaq.it,giuseppe.fiorilli@student.univaq.it

ABSTRACT

Heterogeneous multiprocessor platforms are becoming widespread in the embedded system domain, mainly for the opportunity to improve timing performance and to minimize energy/power consumption and costs. Therefore, when using such platforms, it is important to adopt a Design Space Exploration (DSE) strategy that considers compromises among different objectives. Existing DSE approaches are generally based on evolutionary algorithms to solve Multi-Objective Optimization Problems (MOOPs) by minimizing a linear combination of weighted cost functions (i.e., Weighted Sum Method, WSM). In this way, the main issues are related to reduce timing execution while trying to improve the evolutionary algorithm performance, introducing strategies that attempt to bring better solutions. Code parallelization is one of the most used approaches in this field, but no standard methods have been released since different aspects could affect the performance. This approach leads to exploit parallel and distributed processing elements in order to implement evolutionary algorithms. In the latter case, if we consider genetic algorithms, it is possible to talk about Parallel Genetic Algorithms (PGA). Considering this context, this paper focuses on DSE for heterogeneous multi-processor embedded systems and introduces an improvement that reduces execution time using parallel programming languages (i.e., OpenMP) inside the main genetic algorithm approach, while trying to lead to better partitioning solutions. The descriptions of the adopted DSE activities and the OpenMP implementation, validated by means of a case study, represent the core of the paper.

CCS CONCEPTS

• **Computer systems organization → Embedded systems**; **Embedded hardware**;

KEYWORDS

Design Space Exploration, Heterogeneous Multi-Processor, Embedded Systems, Multi-objective optimization, Parallel Genetic Algorithm

WOODSTOCK'97, July 1997, El Paso, Texas USA

ACM ISBN 123-4567-24-567/08/06…$15.00
https://doi.org/10.475/123_4

1 INTRODUCTION

In the last thirty years, there has been an exponential increase in the diffusion and evolution of Information and Communication Technologies (ICT), especially in the Embedded Systems domain. As a consequence, the presence of embedded electronic digital HW/SW systems in everyday life is constant and, often, almost invisible. In addition to the main Functional Requirements (FR, i.e., what the system shall do), it is possible to identify several Non-Functional ones (NFR, i.e., other requirements that shall be satisfied but not related to the functionality) normally relevant in the embedded systems domain (e.g., cost, size, power/energy, etc.) and well known in advance. NFR are associated with related design requirements and related metrics, i.e., measurable properties of the system implementation. The co-existence of FR and NFR is the most relevant challenge. Indeed, different HW and SW components can be selected and organized in order to optimize power consumption, size, cost, etc. Unfortunately, such optimizations are normally in contrast among them (e.g., performance vs. size) and a proper trade-off needs to be accepted. Unfortunately, there are no fully engineered general methodologies defined for this purpose and often the best option is still to refer to designers' experience. SW tools that support designers in order to reduce costs and overall complexity of systems development are even more of fundamental importance.

A classical HW/SW system design flow partitions a set of functionality into SW-components, implemented by means of high-level programming languages (e.g., C/C++) that runs on specific processors, and HW-components, described by means of domain-specific languages such as VHDL or SystemC. Co-Simulation activities are then required to check constraints fulfillment, taking into account both communication and interfacing issues. In this scenario, an Electronic System-level entry is a good starting point for an HW/SW Co-Design flow. Considering an ESL methodology, the main design issues are modeling FR/NFR and validating them before implementing the system itself. In such a context, one of the most critical issues is always related to Design Space Exploration (DSE) activities. DSE is related to the approach, whether automated or not, used in order to find the best HW/SW partitioning and mapping for the final system implementation. These approaches usually rely on optimization problems, where there was more than one objective function that can be considered (i.e., minimize cost, power consumption, maximize performance, throughput, etc.), but most of the time objectives are often conflicting. This kind of optimization issues are called "Multi-objective optimization problems" (MOOP), where multiple objectives have to be optimized simultaneously. A classical approach to solve an MOOP is the Weighted Sum Method (WSM), which assigns a weight to each objective function so that the

problem is converted to a Single-Objective Optimization Problem (SOOP) with a scalar cost function. In this manner, the multi-criteria optimal design problem is changed into a single objective optimal design one and could be solved in a relatively straightforward way, using also a Genetic Algorithm (GA) procedure. GAs operate with a set of individuals, called population, which normally are created randomly. While the search method is evolving, the population converges to a single solution that dominates any other individuals.

The specific research questions formulated in this study are the following:

- **RQ1**: *How is possible to reduce automatic DSE-GA execution time without loss of accuracy and diversity?*
- **RQ2**: *Which paradigm can be used in order to guarantee bounded timing behavior?*
- **RQ3**: *In which manner the automatic DSE configuration drives the choice of the possible algorithm implementations?*

To answer these questions, this paper presents a PGA approach oriented to reduce execution time in the utility function evaluation, using OpenMP directives and population partitioning.

The remainder of the paper is organized in the following steps. Section II presents some preliminaries and theoretical aspects related to PGA. Section III illustrates the reference DSE approach used. Section IV shows the proposed PGA approach. Section V presents some preliminary experimental results. Finally, Section VI closes the paper with some conclusions and future works.

2 PRELIMINARIES

According to literature, researchers were interested in reducing the execution time and resources required by GAs. One of the latest trends in this field is to exploit parallel and distributed executions (i.e., Parallel Genetic Algorithm, PGA). Genetic operations on individuals in the population can easily be carried out in parallel (creation, evaluation, mutation, and crossover). The basic idea of most parallel programs is to divide tasks into smaller sub-tasks and to execute them simultaneously on different cores. This "divide and conquer" approach can be applied to genetic algorithms in different ways. The basic difference among these several implementations is to use a single population or to divide the latter into several sub-populations. Starting from these differences, PGAs can be classified into three main categories [1]:

- GA master-slave with a global single population (global parallelization): there is a single population (as well as in classical GAs), but individuals fitness evaluation is performed in parallel on various cores;
- Coarse-grained GAs: GA with large-grained multiple populations (also known as Parallel Distributed Genetic Algorithms, PDGA, or Island-Based Genetic Algorithms, IBGA), that includes the subdivision of the population into small-sized sub-populations which exchange individuals with migration operations.
- Fine-grained GAs: GAs with a single-grained population (also known as Cellular PGA, CPGA): GA characterized by a spatial population subdivided into sub-populations smaller than the classical case. Selection and reproduction operations are restricted to sub-groups, and neighborhood overlapping is allowed. The ideal case is to have only one individual for each processing unit (sub-population with only one individual).

Fig. 1 shows the schematic view of the three types of PGA [1]. Since the size of sub-populations in coarse-grained and fine-grained GAs is smaller than in the serial case, we should expect these to converge firstly towards solutions. However, when comparing the performance of serial and parallel algorithms, the quality of the solutions found must also be taken into consideration. Indeed PGAs converge more quickly, but it is also true that the quality of the solutions could be lower than in the classic case. Furthermore, it should be noted that while global parallelization does not affect the normal behavior of the genetic algorithm (master-slave), the coarse-grained and fine-grained GAs change the way the algorithm works [1].

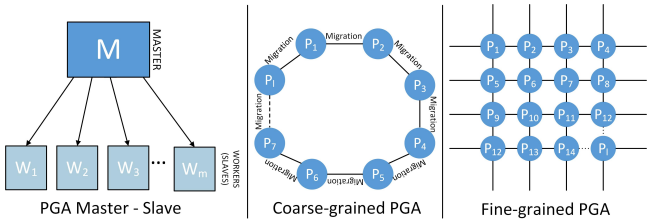

Figure 1: Parallel Genetic Algorithm Types.

The master-slave PGA commonly parallelizes the evaluation of individuals, as the fitness value of each individual is independent of the rest of the population. Therefore, there is no need for communication at this stage. Individuals are evaluated by assigning a fraction of the population to each available processor. The master-slave PGA can be also efficiently implemented on shared memory or distributed memory architectures. In the case of a shared memory multiprocessor, the population is shared and each processor can read the assigned individuals and return the result of the assessments without conflicts. On distributed memory computers, the population can be saved on a single processor. Usually, the number of individuals assigned to the various processors is constant but in some cases, it may be necessary to balance the computational load using a dynamic scheduling algorithm.

Other aspects of a PGA that can be parallelized may be the evaluation of individuals. However, these operations are so simple that in most cases the time required to send individuals back and forth between processors would compensate for any gain in performance. This overhead in communications also hinders the parallelization of the selection operation as many of these techniques need information on the entire population and therefore require continuous communications. Ultimately, the master-slave GAs with a single global population have the advantage of being easily implementable and leading to an effective parallelization, especially when the evaluation of individuals requires a great computational effort. Moreover, they have the characteristic of not altering the search method of the serial GA, so as to be able to apply all their theory directly.

In such a context, Abramson et al. [2] have implemented a PGA with global parallelization for the problem of railway timetables on two target platforms, one shared memory (Encore Multimax with 16 processors) and the other with distributed memory (Fujitsu AP1000

with 128 processors). Their study showed a significant speed-up using up to 16 processors on two computers, while, if other processors are added to the calculation, the speed-up degraded due to the increase of communications. The work in [3] presents a hybrid parallel genetic algorithm (HPGA) based on two-layer parallelism of processes and threads, and it is used to solve reference benchmark functions. The solution uses a hybrid coarse-grained master-slave PGA, integrating MPI and OpenMP parallel programming model. Finally, the paper in [4] describes an optimization method for parameter estimation of one-dimensional groundwater reactive transport problems using a PGA. Results indicate that the OpenMP FORTRAN parallel implementation, used on an Intel quad-core desktop computer, has a linear speedup with an increasing number of processors. Furthermore, the performance of the optimization algorithm is sensitivity to the various genetic algorithm (GA) parameters, including initial population size, number of generations, and parameter bounds. It is for all these reasons that the master-slave approach has been chosen as the reference PGA.

3 REFERENCE DESIGN SPACE EXPLORATION

In the context of Heterogeneous Multi-Processor Embedded Systems (HMPESs), this work adopts the ESL HW/SW co-design flow presented in [5][6][7], where the most critical development step is the semi-automatic *Design Space Exploration* one. The main goal of this approach is to try to fully address the problem of both automatically suggest an HW/SW partitioning of the system specification and map the partitioned entities onto an automatically defined heterogeneous multi-processor architecture. The DSE is constituted of two iterative activities: *HW/SW Partitioning, Mapping and Architecture Definition* and *Timing Co-Simulation*.

The former involves several stages, from the definition of the solution space, the encoding concerning the decision variable space, and the definition of the objective functions and general MOOP. The main problem is to map application processes on a set of basic HW components (called *Basic HW Components*, BCs) selected by the decision maker. It is worth noting that the MOOP is modeled as a minimizing problem (i.e. the "objective functions" are "cost functions"), defined as follows:

Definition 3.1. (*Reference Design Space Exploration MOOP*).

$$\begin{aligned} \min_{\bar{x}} \quad & \bar{F}(\bar{x}) = [F_1(\bar{x}), \ldots, F_k(\bar{x})]^\intercal \\ \text{s.t.} \quad & \bar{x} \in \Omega = \{\bar{x} \in \mathbb{N}^n_{>0} : x_i \leq b\} \end{aligned} \tag{1}$$

where the value b is the total number of considered BCs. The $\bar{x}$ vector represents application processes and the values are BC instances in the decision variable space. So the solution space is bounded by the total number of BCs. Each solution is mapped on a discrete value, and constrained by the maximum number of allowed BCs instances.

The cost functions depend on different metrics evaluated/estimated during the co-design flow. Then, a multi-objective genetic algorithm (MOGA) is used to find an approximation of the Pareto front in a single run, where, starting from the phenotype space, the solution has been encoded considering application processes and BCs.

Considering the decision variable space size, it is possible to evaluate the number of solutions as the permutations with repetition of n processes, that compose the solution vectors $\bar{x}$, allocated on b BCs, so the space size is b^n. It is worth noting that the feasible design space Ω is convex, but we cannot say anything about the feasible solution space.

Starting from this definition, a GA is used to solve the HW/SW partitioning problem. Normally, the decision variables $\bar{x}$ have been represented in GAs by using binary strings [8], but some authors have studied the convenience of using other alternatives, suggesting the utilization of more natural representations, such as the real-valued encoding [8]. For our purposes, an integer-based representation of individuals has been considered. Each individual is characterized by a "fitness", which is the value of the cost functions, calculated in correspondence of the specific considered metric for each individual.

Applying a WSM to the MOOP considered in this work, it is possible to define the utility function that quantifies the quality of each individual of the GA population. The cost functions (called *indexes*) and the methods used to evaluate them at each iteration have been defined in [5][6][7]. In this context, the instance of an individual $\bar{x}$ is defined as a vector where the indexes represents processes and the values represents BC instances. This is shown in Fig. 2.

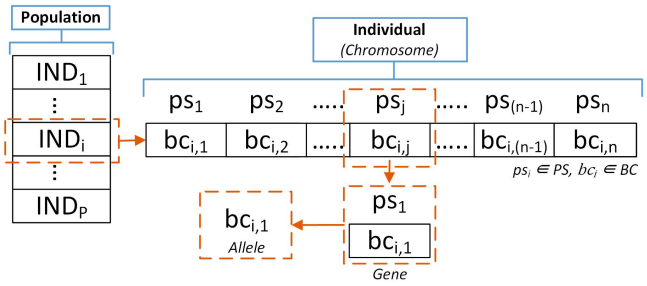

Figure 2: Genetic Algorithm Individual Set. Each individual is composed by different genes, where ps_j are application processes, and $bc_{i,j}$ are BC instances

Finally, the *Timing Co-Simulation* activity considers suggested mapping/architecture items to actually check for timing constraint satisfaction. It is performed using an HW/SW Timing Co-Simulator fully described in [9] that uses an off-the-shelf unifying statement-level metric evaluated as described in [10]. This simulation activity is out of the scope of this paper, while it will be considered in future works.

4 PROPOSED APPROACH

The proposed approach has been shown in Fig. 3. S_i, W_j and F_k represents respectively the sub-population size, the number of workers/threads considered, and the number of objective functions evaluated in parallel. The input considers also GA parameters (the type of mutation, crossover, selection, etc.), and the specific use case, while it is possible to change OpenMP directives and configuration. The outputs are DSE solutions (with performance values and degree of goodness respect to the optimal ones) and the execution time associated with the different configurations.

OpenMP has been chosen as a reference parallel programming language for PGA implementation. OpenMP [11] is a specification

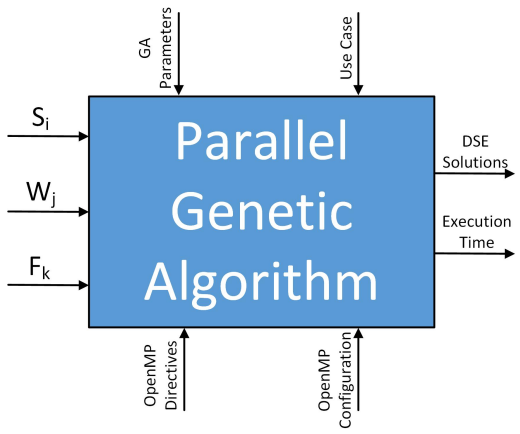

Figure 3: Proposed PGA Approach

for a set of compiler directives, library routines, and environment variables that can be used to specify high-level parallelism in FORTRAN and C/C++ programs. It instructs the compiler to organize parallel sections of code in a specific manner and helps to parallelize execution of an application. It is based on a fork-join model. The implementation of OpenMP based on GCC, called libgomp [12], has been selected to provide support to the execution of parallelized applications that use this library.

A global parallelization has been implemented, where a master-slave PGA approach has been used. More in details, this work considers 3 types of PGA implementation:

(1) Type 1: Parallel evaluation of objective functions for each individual (evaluates objective functions on different threads);
(2) Type 2: Parallel evaluation of utility function (split the population into subsets);
(3) Type 3: Hybrid approach, combining the previous two approaches.

The first case has been implemented using the *#pragma omp parallel { .... }* preprocessing directive, using OpenMP threads to evaluate a custom number of objective functions directly proportional to threads number. The OpenMP *Section* construct has been used to avoid race conditions between the different threads. In such a way it is possible to limit the code sections executed by several distinct threads (defined by the *#pragma omp section* directive). In such a way, functional parallelism is ensured in which each processor operates on a set of different and independent instructions using the section construct.

The second scenario has been implemented in two different ways. The former involves the use of *#pragma omp parallel for* directive, the latter considers the use of *section* construct to evaluate the utility function for each sub-population. In the first case, the *for* construct splits the loop so that each team thread manages a predetermined portion of the loop itself. The construct splits the population and distributes equally the sub-population to each thread. This solution may not be the most performing, while sometimes it is useful to decide a priori on how to divide the computational load among the threads. The directive *schedule (type[,chunk])* is used to manage the chunk size (i.e., the dimension of the portion of the loop assigned to each thread in the current team). The optional chunk variable must be an integer or a scalar expression evaluated outside the for construct. The type can be *static*, *dynamic* or *guided*. The first type divides the population into equal sections that are statically assigned to threads with round-robin order. It is recommended when the work can be evenly spread across cores. The *dynamic* type splits the population into sets with the dimension equal to chunk. Each thread receives the population set dynamically. It is recommended when several iterations had an unpredictable time, so it is necessary a workload balance among the different cores. The last type decreases the population size exponentially until you get to the minimum default chunk size. This is a hybrid solution between the first two types. Considering the use of Section directive, the population is divided into equal sub-population (driven by the number of threads used) and the sub-sets are assigned to the distinct sections. The *private(i)* directive avoid to skip fitness function evaluation, when the for index *i* is shared among different threads.

The final hybrid solution implemented performs the creation of distinct parallel blocks using the directive *parallel* to evaluates the fitness function on a specific sub-population. The population is divided into equal sub-population assigned to a single section that evaluates a fixed number of objective functions, depending on the number of threads used in the system. Fig. 4 shows this last solution, where it is possible to change OpenMP number of threads and sections.

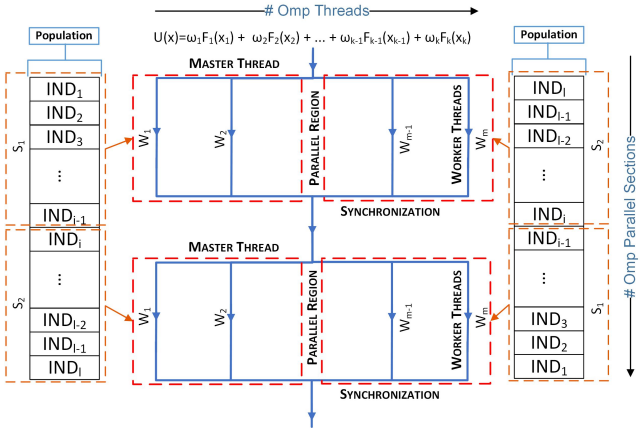

Figure 4: Hybrid Implementation

5 EXPERIMENTAL RESULTS

All the comparative studies on multi-objective genetic algorithms agree that the elitism and diversity operations lead to better results for the MOOPs. However, the implementation of this type of operation requires a computational effort and additional resources to the system. As an example, Fig. 5 shows two DSE solutions in two different scenarios, where the elitism is present or not.

The considered use case, or the Time-to-completion (TTC) timing constraint that drives the DSE steps, and the genetic algorithm parameters are described in [13]. From this example it is clear that elitism is one of the most important feature needed by a Genetic

Table 1: Parallelism performance table

	500000 individuals				1000000 individuals			
Impl.	Best Case (s)	Speed-up	Worst Case (s)	Speed-up	Best Case (s)	Speed-up	Worst Case (s)	Speed-up
1.	34	-	38	-	71.5	-	76.5	-
2.	36	-5.56%	38	0	74.7	-4.28%	82.3	-7.05%
3.	30,1	11,47%	32,6	14,21%	62,1	13,15%	71,8	6,14%
4.	29	14,7%	32	15,79%	59	17,48%	65,5	14,38%
5.	27,4	19,41%	29	23,68%	57	20,28%	62,7	18,03%
6.	29,5	13,23%	31,6	16,84%	62,7	12,31%	65,8	13,99%
7.	28,6	15,88%	31	18,42%	60,5	15,38%	65,2	14,77%
8.	35,5	-4,22%	37,5	1,32%	76	-5,92%	83	-7,83%

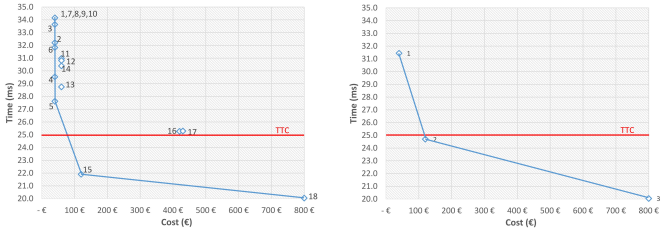

Figure 5: Design Space Exploration with elitism (left) and with classical approach (right, no elitism)

algorithms, but it introduces timing overheads (as shown in the last figure in [14]), and a correct way to implement this functionality is one of the main problem in this field. The next paragraph describe the results obtained using the proposed PGA approach, while the reference use case and the GA configurations have been extracted from [13]. The PGA has been tested on a PC with Intel Core i5-2430M dual-core (2.40 GHz), 4GB di RAM and Linux Ubuntu distribution. The hyper-threading processor can manage up to 4 threads in parallel.

This section presents the results in terms of execution times resulting from the use of parallelism. In particular, 100 tests were performed with each of the proposed implementations to test the performance of each of them and evaluate the speed-up obtained with respect to the serial case. Table 1 summarizes the results obtained, showing the best and worst case compared to the serial execution for each implementation and the speed-up obtained.

The different test and configuration are listed below:

(1) Serial;
(2) Type 1 with parallel evaluation of objective functions for each individuals with 2 threads;
(3) Type 2 with parallel evaluation of utility function using sub-population sets and the *omp parallel for* with 2 threads;
(4) Type 2 with parallel evaluation of utility function using sub-population sets and the *omp parallel for* with 4 threads;
(5) Type 2 with parallel evaluation of utility function using sub-population sets and the *omp parallel for* with 2 threads, schedule guided;
(6) Type 2 with parallel evaluation of utility function using sub-population sets and *omp parallel sections* with 2 sections and 2 threads;
(7) Type 2 with parallel evaluation of utility function using sub-population sets and *omp parallel sections* with 4 sections and 4 threads;
(8) Type 3 (i.e., hybrid approach) with parallel evaluation of objective functions using sub-population and *omp parallel sections* with 2 parallel flows and 2 sections (as shown i Fig. 4).

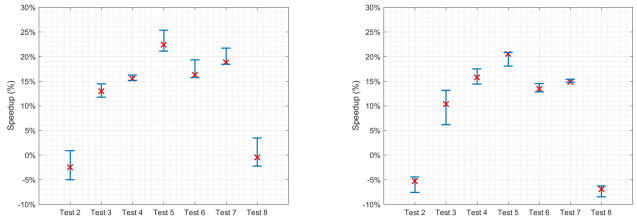

Figure 6: Speed-up plot, with $5 \cdot 10^5$ individuals (left) e 10^6 individuals (right)

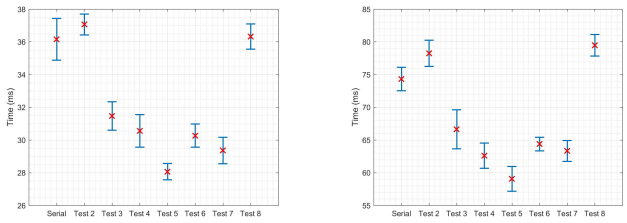

Figure 7: Execution Time plot, with $5 \cdot 10^5$ individuals (left) e 10^6 individuals (right)

Table 1 shows that the solution (5) is the best in terms of timing performance and speed-up ($\simeq$ 20%), for the two specific tests (with $5 \cdot 10^5$ and 10^6 populations size). This solution considers only the use of *omp for* with the guided directive schedule. It is worth noting that there is no optimal chunk size, but it is needed an empirical test to find this "optimal" value. In this work, the best results were found with 5000 chunk size for the first test ($5 \cdot 10^5$ population size) and 20000 chunk size for the second test (10^6 populations

size). Another interesting result is related to the increase in the number of threads from 2 to 4. The main results show that the performance increase more with a huge population, and less with a small population. This trend is not proportional to the number of threads or population size. Furthermore, the first solution degrades the algorithm performance while the parallel evaluation of each objective function is less than the overhead introduced by OpenMP parallelization, respect to threads synchronization and access to individuals saved in the shared memory access.

Figure 6 shows the speed-up distribution for $5 \cdot 10^5$ individuals e 10^6 individuals. The mean best solution in terms of performance is the configuration (5), as said before, while the solution (4) do not change drastically respect to the population size (In the other solutions the increased individual set degrades performance very much).

Finally, Figure 7 presents execution time distribution related to the different tests. The less standard deviation is related to the test number (6), so if the designer is interested in a predictable implementation, this solution can be used as a reference one.

6 CONCLUSIONS AND FUTURE WORKS

This work has presented a DSE approach extended to implement a parallel genetic algorithm able to reduce execution time while it does not degrade the goodness of the final DSE solution founded. An approach to finding the best PGA configuration is presented, where the designer can changes input parameters to achieve desired performances. Results show the need to do not fix the PGA configuration using OpenMP, while the performances depend on population size and GA configuration. Future works will exploit PGA on different technologies and parallel programming languages (GPU/CUDA, multi/many cores/OpenMPI, etc..), and try to implement also fine-grained or coarse-grained PGA, to compare performances and solution founded still taking into account decision maker preferences in the whole design flow.

ACKNOWLEDGMENTS

This work has been partially supported by the ECSEL RIA 2016 MegaM@Rt2 and AQUAS projects.

REFERENCES

[1] Erick Cantú-Paz. A survey of parallel genetic algorithms. *CALCULATEURS PARALLELES, RESEAUX ET SYSTEMS REPARTIS*, 10:30, 1998.

[2] D.Abramson, G. Mills, and S. Perkins. Parallelisation of a genetic algorithm for the computation of efficient train schedules. *Proceedings of the 1993 Parallele Computing and Trnsputers Conference*, pages 139–149, 1993.

[3] Z. Wang, T. Ju, D. Cui, and X. Hei. A study of hybrid parallel genetic algorithm model. In *2011 Seventh International Conference on Natural Computation*, volume 2, pages 1038–1042, July 2011.

[4] J. Torlapati and T. P. Clement. Using parallel genetic algorithms for estimating model parameters in complex reactive transport problems. *Processes*, 7:640, 09 2019.

[5] L. Pomante. Hw/sw co-design of dedicated heterogeneous parallel systems: an extended design space exploration approach. *IET Computers Digital Techniques*, 7(6):246–254, November 2013.

[6] L. Pomante. System-level design space exploration for dedicated heterogeneous multi-processor systems. In *ASAP 2011 - 22nd IEEE International Conference on Application-specific Systems, Architectures and Processors*, pages 79–86, Santa Monica, CA, USA, Sept 2011. IEEE.

[7] C. Brandolese, W. Fornaciari, L. Pomante, F. Salice, and D. Sciuto. Affinity-driven system design exploration for heterogeneous multiprocessor soc. *IEEE Transactions on Computers*, 55(5):508–519, 2006.

[8] T. Back. *Evolutionary algorithms in theory and practice. Evolutions strategies, evolutionary programming, genetic algorithms.* Oxford University Press, 1996.

[9] D. Ciambrone, V. Muttillo, L. Pomante, and G. Valente. Hepsim: An esl hw/sw co-simulator/analysis tool for heterogeneous parallel embedded systems. In *2018 7th Mediterranean Conference on Embedded Computing (MECO)*, pages 1–6, 2018.

[10] V. Muttillo, V. Stoico, G. Valente, F. D'Antonio, L. Pomante, and F. Salice. Cc4cs: An off-the-shelf unifying statement-level performance metric for hw/sw technologies. In *Companion of the 2018 ACM/SPEC International Conference on Performance Engineering*, volume 2018-January, pages 119–122, 2018.

[11] The OpenMP Team. The openmp api specification for parallel programming. www.openmp.org, 2019.

[12] *GNU libgomp*, 2018 (accessed: 29.11.2019). https://gcc.gnu.org/onlinedocs/libgomp/.

[13] V. Muttillo, G. Valente, and L. Pomante. Criticality-driven design space exploration for mixed-criticality heterogeneous parallel embedded systems. In *Proceedings of the 9th Workshop and 7th Workshop on Parallel Programming and RunTime Management Techniques for Manycore Architectures and Design Tools and Architectures for Multicore Embedded Computing Platforms*, PARMA-DITAM '18, pages 63–68, New York, NY, USA, 2018. ACM.

[14] V. Muttillo, G. Fiorilli, and T. Di Mascio. Tuning dse for heterogeneous multiprocessor embedded systems by means of a self-equalized weighted sum method. In *Proceedings of the 10th and 8th Workshop on Parallel Programming and RunTime Management Techniques for Many-core Architectures and Design Tools and Architectures for Multicore Embedded Computing Platforms*, PARMA-DITAM 2019, pages 1:1–1:4, New York, NY, USA, 2019. ACM.