# A unified view of parallel multi-objective evolutionary algorithms

El-Ghazali Talbi

# A Unified View of Parallel Multi-objective Evolutionary Algorithms

Prof. El-Ghazali Talbi

*University Lille 1, Bat.M3 59655 Villeneuve d'Ascq, France*

**Abstract**

This paper describes a unified view of parallel evolutionary algorithms for multi-objective optimization problems. The parallel optimization algorithms are detailed from both design and implementation aspects. The proposed taxonomy is based on three hierarchical parallel models. Moreover, various parallel architectures are taken into account. The performance assessment issue of parallel multi-objective evolutionary algorithms (MOEA) is also presented. This work can be extended to any population-based metaheuristics such as particle swarm and scatter search.

*Keywords:* `Parallel evolutionary algorithms, metaheuristics, multi-objective optimization`

## 1. Introduction

Many multi-objective optimization problems (MOPs), such as in finance, logistics/transportation, engineering design and life science, are complex. Indeed most of the practical and academic MOPs are NP-complete. Moreover they are CPU time and memory consuming. Even if the use of multi-objective evolutionary algorithms (MOEAs) allows to decrease the computational complexity of the algorithms, this complexity remains important for a great majority of MOPs in multiple domains of applications, in which the constraints and the

☆This work is part of a project that has received funding from the *European UnionH2020 research and innovation programme* under grant agreement No 692286.

objective functions associated to the problem are resource intensive and the size of the decision and/or objective space is important.

Additionally, the fast evolution of technology in the development of networks (local networks such as Infiniband or wide area networks such as optical networks), data storage, and processors (e.g. multi-core processors, vector units, GPUs, ARM, FPGAs), make the use of high-performance computers popular. Such parallel machines represent an interesting opportunity for the development of parallel multi-objective evolutionary algorithms. Indeed, sequential machines are reaching technological limitations. Even workstations and laptops are composed of multi-core processors and GPUs. The ratio between cost and performance is constantly decreasing. The proliferation of puissant processors and networks have shown the apparition of architectures such as many-cores, GPUs, networks of workstations (NOWs), clusters of processors (COWs), Grids and Clouds for high-performance computing.

High-performance computing architectures are used in the development of MOEAs for the enclosed goals:

- **Reduce the time to approximate the Pareto front**: the objective here is to speedup the search time. For instance this improves the development of interactive optimization algorithms which is an important aspect for multi-criteria decision making. It is an important issue for a given family of MOPs in which there are hard constraints on time such as in dynamic MOPs and operational MOPs such as time-critical control and planning [1].

- **Enhance the properties of the Pareto front**: parallel models for MOEAs enable to enhance the properties of the Pareto front. Indeed, exchanging knowledge between optimization heuristics will modify their behaviour in exploring the decision and objective spaces associated to the MOP. The main objective in a cooperation between algorithms is to improve the quality of the Pareto front. One can improve both the search time and the convergence/diversification of the Pareto front. A parallel

model for MOEAs may be more efficient than a sequential one even on a single machine.

- **Resolve large instances of MOPs**: parallel MOEAs enable to resolve big instances of MOPs. Those big instances cannot be resolved on a sequential computer. One can also resolve more accurate and then more complex models of MOPs. Enhancing the accuracy of models increases the complexity of the associated problems. Additionally, many MOPs need the handling of big databases such as machine learning of big data. Parallel MOEAs can be used with systems such as Hadoop and Spark which provide significant computing power in the mining of big databases [2].

- **Enhance the robustness**: robustness may be measured in terms of solving in an effective manner different MOPs and various instances of a given MOP. It can also represent the sensitivity of the MOEA to its parameters.

In this taxonomy a clear difference is made between the design issue and the implementation issue of parallel MOEAs. A unified description of parallel MOEAs is proposed. In the implementation issue, we will focus on the efficiency of parallel MOEAs on various types of parallel machines. The impact of parallel languages and programming environments, such as message passing and shared memory are analyzed. Several criteria from the architecture point of view are taken into account such as: sharing of memory, homogeneity of resources, multiplicity of users, and network locality. Those considered criteria will influence on the development of efficient load balancing and fault-tolerant approaches.

The organisation of the paper is as follows. In section 2 the main concepts of multi-objective optimization are given. Section 3 details the main parallel models in the design of MOEAs. In section 4 we will focus on the parallel implementation aspects of MOEAs. The most important concepts of parallel machines related to the parallel implementation of MOEAs are underlined. The

performance metrics used to assess the performance of parallel MOEAs are also detailed.

## 2. Multi-objective optimization

**Definition 2.1** (Multi-objective optimization problem)**.** A multi-objective optimization problem (MOP) may be defined as:

$$(MOP) = \begin{cases} min\ F(x,p) = (f_1(x), f_2(x), \ldots, f_n(x)) \\ s.c.\ x \in S \end{cases} \quad (1)$$

where $n$ ($n \geq 2$) is the number of objective functions, $x = (x_1 ..., x_k)$ is the vector representing the decision variables, $S$ represents the set of feasible solutions in the decision space associated with equality and inequality constraints, and explicit bounds. $F(x) = (f_1(x), f_2(x) ..., f_n(x))$ represents the objective space to be optimized[1].

**Definition 2.2** (Pareto dominance)**.** An objective vector $u = (u_1, \ldots, u_n)$ is said to dominate $v = (v_1, \ldots, v_n)$ (denoted by $u \prec v$) if and only if no component of $v$ is smaller than the corresponding component of $u$ and at least one component of $u$ is strictly smaller, i.e.

$$\forall i \in \{1, \ldots, n\} :\ u_i \leq v_i\ \wedge\ \exists i \in \{1, \ldots, n\} :\ u_i < v_i.$$

**Definition 2.3** (Pareto Optimality)**.** A solution $x^* \in S$ is Pareto Optimal[2] if for every $x \in S$, $F(x)$ does not dominate $F(x^*)$, i.e. $F(x) \not\prec F(x^*)$.

A MOP contain a set of solutions known as the *Pareto optimal set*. The image of this set in the objective space is denoted as the *Pareto front*. This set of solutions represents the compromise solutions between the different conflicting objectives. The main goal in the resolution of a MOP is to obtain the

[1] It is assumed, without loss of generality, the minimization of all the objectives.

[2] The Pareto optimal solutions are also known under the name of *acceptable solutions*, *efficient*, *not-dominated*, *non-inferior*.

Pareto optimal set and, consequently, the Pareto front. Notwithstanding, when metaheuristics (e.g. evolutionary algorithms) are applied, the goal becomes to obtain an approximation of the Pareto optimal set having two properties: convergence to the Pareto optimal front and uniform diversity. The first property ensures the generation of near-optimal Pareto solutions, while the second property indicates a good distribution of the obtained solutions around the Pareto optimal front, so that no valuable information is lost.

## 3. Design of parallel MOEAs

In the design of parallel MOEAs, three main parallel hierarchical models are identified (Tab. 1):

- **Parallel algorithmic-level for MOEAs:** in this model, a parallel algorithm is composed of cooperating MOEAs. This model is not dependant on the target MOP to be solved. If the different MOEAs are independent, the search will be similar to the sequential execution of the algorithms. However, the cooperative model will modify the behaviour of the MOEAs and allow to enhance the quality of the obtained Pareto front.

- **Parallel iteration-level for MOEAs:** in this parallel model of MOEAs, an iteration of a MOEA is parallelized. This model is not dependant on the target MOP to be solved. This model will not modify the behaviour of the MOEA. The main goal is to speedup the search time of MOEAs manipulating large populations of individuals.

- **Parallel solution-level for MOEAs:** in this model of MOEAs, the algorithm will handle in parallel a single solution of the decision space. This model is dependant on the target MOP to be solved. It consists in the parallel evaluation of the different objectives or constraints associated to the MOP. Let us notice that in many MOEAs, the most costly operation is the evaluation of solutions. This model will not modify the behaviour of the algorithm. It deals with improving the search time of the algorithm.

Table 1: Parallel models of MOEAs in solving MOPs.

| Parallel level | MOP dependent | Behaviour | Granularity | Objective |
|---|---|---|---|---|
| Algorithm | No | Modified | MOEA Algorithm | Quality & Speedup |
| Iteration | No | Non modified | MOEA Iteration | Speedup |
| Solution | Yes | Non modified | MOEA Solution | Speedup |

### *3.1. Algorithmic model of MOEAs*

In the algorithmic parallel model of MOEAs, different evolutionary algorithms are executed in parallel. They may be independent or cooperative in solving MOPs.

#### *3.1.1. Independent algorithmic model of MOEAs*

In the independent algorithmic model of parallel MOEAs, the various MOEAs are executed without any exchange of information. The various MOEAs are initialized with different populations. Different parameters are used for the MOEAs such as the probability of crossover and mutation. Each component of a MOEA may be developed differently: operators (e.g. mutation, crossover), representation, objective functions, constraints, diversity preservation, fitness assignment, and elitism. Some other parameters will depend on the problem formulation. For instance one can use different reference points to generate various MOPs problems solved by the same MOEA [3]. The reference points can be uniformly distributed within a region that covers the Pareto Frontier.

This model is straightforward to develop. The master/workers pattern is well adapted to this parallel model. A worker implements a MOEA. The master defines the various parameters used by the workers and agregates the best found Pareto front from those obtained by the different workers. In addition to speedup the MOEA, this parallel model allows to improve its robustness [4].

This parallel model evokes the question below: is it comparable to launch $k$ MOEAs during a given time $t$ and to launch a single MOEA during $k * t$? The

answer relies on the characteristics of the decision and objective spaces of the MOP (e.g. distribution of the Pareto optimal solutions).

*3.1.2. Cooperative algorithmic model of MOEAs*

In the cooperative algorithmic model of MOEAs, the various MOEAs are cooperating and then exchanging extracted knowledge from the search with the intent to compute a better and more robust Pareto front [5]. In most MOEAs an archive is maintained outside the current population. This archive is composed generally of all generated Pareto optimal solutions.

In the development of this parallel model for cooperative MOEAs, the following questions occur:

- **The communication criterion (When?)**: the communication of data between the MOEAs can be carried out in a *blind* (probabilistic or periodic) manner or using an adaptive criterion. Periodic communication arises in each MOEA after a given number of iterations; this kind of exchange is synchronous. In probabilistic communication, an exchange operation is carried out after each iteration with a given probability. In adaptive communication, the exchange of information is conducted by some knowledge extracted from the multi-objective search. As an example one can use any information about the improvement of the quality of the Pareto front. A traditional criterion can be associated to the update of the Pareto archive when a new Pareto solution is found.

- **The communication topology (Where?)**: the exchange topology denotes for each MOEA node its neighbor(s). It defines the source and the destination of the information exchanged. The most popular topologies are the regular ones such as mesh, hypercube and rings [6]. Dynamic and random topologies can also be used [7].

- **The exchanged knowledge (What?)**: here we have to define the knowledge extracted from the search to be exchanged between the MOEAs. Generally, the exchanged knowledge consists of:

- Non-dominated solutions: the selection strategy of a MOEA can be based on the non-dominated solutions generated during the search. A given number or percentage of the Pareto set can be selected to be communicated to neighbors.
- Multi-objective search knowledge: except the Pareto solutions, other informations extracted from the multi-objective search can be communicated such as performance indicators of the local Pareto set, or utopian solutions [7].

- **The consolidation policy (How?)**: this question is related to the strategy applied when the communicated knowledge is received. Generally, the local knowledge of a MOEA is renewed with the received one. For instance, when a MOEA receives some non-dominated solutions from its neighbors, an update of its local population or archive is carried out. Many classical replacement approaches (e.g. deterministic, stochastic) can be used. One can use a deterministic elitist approach in which the Pareto solutions from the merging of the local Pareto set and the received Pareto set is selected.

A small number of parallel evolutionary algorithms have been developed for multi-objective optimization compared to single objective optimization [4]. The popular parallel model of evolutionary algorithms, the cellular model, may be reduced tho the migration model in which an island consists of a single solution. The cell of the grid is represented by a single solution. The selection strategy is based on the neighbors of the solution. Consequently, this parallel model supports more diversification than in a sequential search. In [8], it has been shown for many practical applications that parallel cellular models for MOEAs are more efficient in the generation of the Pareto front than sequential MOEAs.

Some parallel models of MOEAs may be based on the partitioning of the objective space. Indeed, a given MOEA may be concerned by a subset of the objective functions (Fig. 1). A MOEA can also manage a single objective. Another strategy may be based on using various scalarization strategies of the objectives set such as different weights of a linear aggregation [9]. This approach has been

used in the MOEA/D algorithm which is based on the decomposition of the objective space [10][7]. Other parallel models can be based on the partitioning of the decision space [11][12]. Hence a given MOEA will focus its search on a partition of the decision space.

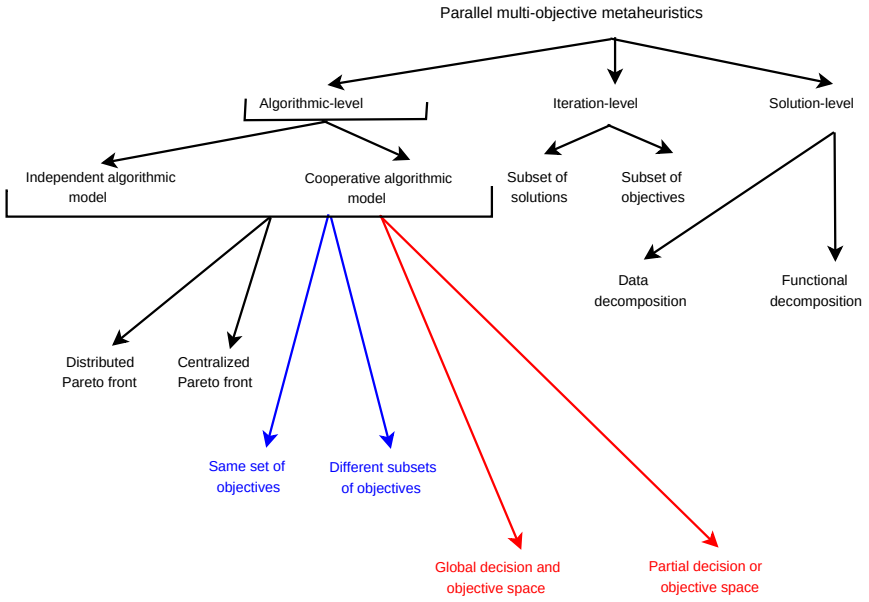


Figure 1: A taxonomy of parallel MOEAs for multi-objective problems.

Another classification criterion may be related to the cenralized or distributed ways to generate the whole Pareto front. Two strategies may be developed (Fig. 1):

- *Centralized approach*: in this approach, the Pareto set is represented in a shared global memory. All MOEAs update this centralized structure in parallel [13] [14][15][16].

- *Distributed approach*: here the Pareto set is distributed between the different MOEAs. Each MOEA maintains a local copy of the Pareto set. Then, a global update is carried out at the end of the computation [17][18][19].

### *3.2. Iteration-based model for MOEAs*

The iteration-based parallel model for MOEAs consists in the parallelization of a single iteration of evolutionary algorithms. This model consists generally in handling in parallel the population of individuals. The CPU-time consuming component of any MOEA is the fitness assignment of the solutions composing the population. This parallel model does not depend on the target MOP. Only problem-independant search components are parallelized such as the evaluation of the objective function and the update of the population. Moreover, there is no modification of the behaviour of the algorithm. The goal is only to accelerate the search process. This is a straightforward and popular model for parallel MOEAs. Pratically, many MOPs are characterized by expensive objective functions. As an example, many computational engineering optimization problems are concerned by complex solvers using surrogate models from finite element methods, electromagnetics, fluid dynamics [20][21]. In multi-disciplinary design optimization, complex simulations are handled to evaluate the objective functions associated to the problem [22].

In parallel MOEAs the population of solutions may be controled in parallel. In a classical master/workers model, the master will handle the initialization of the population, the selection and the replacement phases, while the variation phase (e.g. mutation, crossover) and the fitness assignment phases are managed by the workers [17] (Fig. 2). In the fitness assignment and the update of the Pareto archive, the Pareto ranking is a costly operation that can also be parallelized [23].

Two different strategies may be developed:

- **Synchronous approach:** this approach is equivalent to a sequential MOEA. All operations are handled in a synchronous way and then finalised before starting a new iteration. The master has to wait for all results from the whole set of workers. Then a new iteration of the evolution mechanism can be handled.

- **Asynchronous approach:** in the asynchronous approach, the master

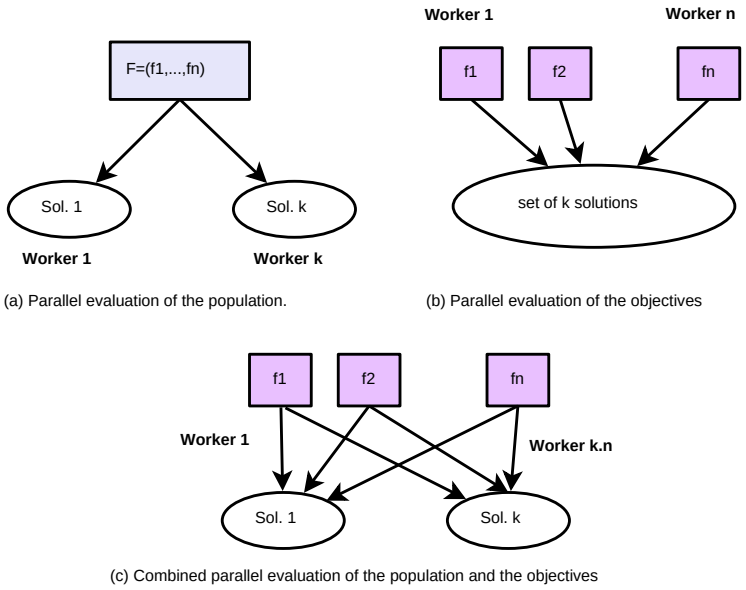


Figure 2: The iteration-based parallel model for MOEAs.

process can handle a new iteration before all the workers finalize their process. For instance in asynchronous steady-state MOEAs, the variation and the evaluation phases may be handled in parallel. Two queues of solutions with fixed sizes are managed in parallel: the queue of solutions to be evaluated and the queue of already evaluated solutions. The free workers will handle the solutions of the first queue. Hence the master process handle the selection phase from the second queue without waiting the results from all the workers. This asynchronous model is not equivalent to the sequential one. Indeed, there is no guarantee that the order of selecting and replacing the solutions is the same than in sequential algorithms.

In swarm based multi-objective algorithms, knowledge other than solutions is used to generate new solutions. This knowledge must be shared by all agents of the swarm. As an example, in ant-based optimization, this knowledge is represented by the pheronome matrix. The master process must distribute this matrix to each worker. An agent of the swarm is handled by a worker. Each

worker acquires the pheromone matrix, generate and assign a fitness to the new solution and then return the result to the master. The pheromone matrix is updated once all results are collected from the workers [24][25].

### *3.3. Solution-based model for MOEAs*

In this parallel model the main goal is to reduce the search time of a MOEA. This model focus on the parallelisation of the functions treating a single solution such as the calculation of objective function and constraints. Two different decomposition strategies may be applied[26][9][27] (Fig. 1):

- **Decomposition of data:** many big MOPs can be decomposed according to the data involved in the problem. Hence the objective functions and/or the constraints can be evaluated in parallel for each partition [28]. Depending on the target MOP, a partition can be a geographical area, a data structure or a database.

- **Decomposition of functions:** the objectives or the contraints of a MOP can be decomposed in many sub-functions. Those sub-functions can be handled in parallel by a set of workers. The master needs to collect and combine the results from the workers to assign a global fitness to a solution in terms of convergence and diversification. The parallelism degree of such a parallel model is defined by the numer of sub-functions associated to the objectives and constraints.

### *3.4. Models combination for parallel MOEAs*

The various parallel models of MOEAs can be associated in a single parallel model using a hierarchical organization [29]. Hence this combined scalable parallel model will offer a high degree of parallelism $k * m * n$ where $k$ represents the number of independent or cooperating MOEAs, $m$ represents the population size and $n$ represents the number of sub-functions or partitions of the data associated to the MOP.

## 4. Implementation issues of parallel MOEAs

This section will focus on the efficient implementation of the proposed parallel models of MOEAs on different types of high-performance machines.

### *4.1. High-performance computing architectures and programming environments*

High-performance computing architectures are emerging rapidly. The classification criteria for parallel machines having a great influence in the efficient implementation of parallel MOEAs are: shared or distributed memory, homogeneity or heterogeneity of machines, multiple users sharing of the machines, and locality of the communication network. All the proposed parallel models of MOEAs will be analyzed according to those criteria.

**Shared/distributed memory**: the set of processors of a shared-memory machine are linked through a global shared memory. Various topologies have been developed for the communication network (e.g. multistage crossbar, shared bus). This kind of parallel machines are comfortable in terms of programming in a way that classical sequential paradigms can be reused. However this kind of architecture has limited scalability and an important cost. A popular example of such high-performance machines are multi-core architectures and symmetric multiprocessors.

In a distributed memory machine, the processors have their local memory. The processing elements are coupled by a communication network using various topologies such as rings, hypercubes and fat-trees. This type of machines is much more complicated to program but is more scalable. Communication of information must be handled by message passing. The most popular example of distributed memory machines is the clusters of processors (COWs).

**Homogeneous/heterogenous machines**: the homogeneity of resources (e.g. processors, networks) is an important characteristic that has to be taken into account. One can qualify clusters of processors by the homogenity of the processing elements. Some networks of workstations (NOWs) can be distinguished by the heterogeneity of the processors and the communication networks.

**Single/multiple users**: many high-performance computing machines such as COWs are non-shared among multiple users. Those machines are dedicated for a single user at a given time. However, some parallel machines such as NOWs can be shared by multiple users.

**Local-area network/wide-area network (WAN)**: local-area networks (LAN) are generally the adopted networks for massively parallel architectures and clusters. Wide-area networks represent loosely coupled networks and are used in grid computing systems and large network of workstations. Those networks are characterized by a highest cost in terms of communication.

A grid is generally composed of a large set of distributed heterogeneous resources which are administrated across multiple domains [30]. Two different families of Grids may be found:

- **High-Performance Grid**: in high-performance Grids (HPC Grid) a dedicated high-performance network is used to interconnect the different clusters and supercomputers. HPC Grids are generally non shared by multiple users. At a given time a single application is executed on a subset of processors.

- **Desktop Grid**: in desktop Grids, many private workstations are connected through a large scale networks (e.g. internet). Desktop Grids are generally shared by many applications and users.

**Volatile/non-volatile parallel machines**: a volatile machine is a parallel architecture in which there is a dynamic availability of resources. For instance, in a desktop Grid, there is a volatility in terms of processors. Moreover, the probability of failure (e.g. shutdown) of processors is important.

The following table 2 summarizes the properties of the main HPC machines according to the proposed criteria.

Embedded systems, which are small size, reduced complexity and low-power architectures, can be used in the development of parallel efficient MOEAs [31]. The most popular specific architectures are Graphical Processing Unit (GPU), Field Programmable Gate Arrays (FPGAs) and ARM family of processors.

Table 2: Propoerties of the main HPC machines.

| Criteria | Shared Memory | Homogeneity | Resource Sharing | Network | Volatility |
|---|---|---|---|---|---|
| **SMPs Multi-cores** | Yes | Yes | Yes/No | LAN | No |
| **COWs** | No | Yes/No | No | LAN | No |
| **NOWs** | No | No | Yes | LAN | Yes |
| **HPC-Grid** | No | No | No | WAN | No |
| **Desktop-Grid** | No | No | Yes | WAN | Yes |

FPGAs are implemented hardware devices by a programming process [32]. We expect that the exploitation of FPGAs (e.g. Xilinx) will be widespread in the implementation of parallel MOEAs especially for some challenging and highly used applications such as in bioinformatics [33][34]. Some parallel implementations of MOEA have been carried out on specific architectures (e.g. microcontrollers) [35]. Initially GPUs have been dedicated to graphics rendering and then generalized to numerical compuations. GPUs are based on a SIMD (Single Instruction Multiple Data Streams) structure [36]. Many efficient implementations of parallel MOEAs on GPUs have been proposed in the literature [37][38][31][39]. ARM is a family of reduced instruction set computing (RISC) architectures for processors. They are extensively used in consumer electronic devices such as smartphones and tablets. The implementation of parallel evolutionary algorithms on embedded systems based on single-core and multi-core ARM processors starts to be explored [40].

The target parallel architecture will affect the selection of the suited programming environment for the implementation of a parallel MOEA. Shared-memory or message passing are the most popular programming paradigms that can be used. For shared-memory HPC machines, there are three main options:

- **Multi-threading:** a thread is a light process. Multiple threads composing a process share the same memory space. An important issue in multi-threading is the recovery between computation and communication.

Multi-threaded programming may be used within libraries (e.g. Pthreads) [41] or programming languages (e.g. Java threads) [42].

- **OpemMP and CUDA:** OpenMP (Open Multi-Processing) and CUDA are the most popular programming environment based on shared memory concepts. Various programming langages (e.g. Fortran, C and C++) are interfaced with OpenMP and Cuda [43]. CUDA is used for the GPUs and its portability is limited to Nvidia GPUs. The CUDA framework can be accessed by a directive based language such as OpenACC (Open Accelerators). The OpenACC environment has advantages both in efficiency and portability for programming heterogeneous systems composed of GPU and multi-cores.

Message passing is the main paradigm for distributed memory environments. The tasks composing a program communicate by message exchange using a synchronous or asynchronous communications. The most popular environment is MPI (Message Passing Interface).

### *4.2. Performance assessment of parallel MOEAs*

Speedup and efficiency are the most popular performance measures to evaluate the scalability of parallel MOEAs [44]. They define the gain obtained by the parallelization of the program. The speed-up $S_N$ is the ratio between the sequential time $T_1$ on a single processor and the execution time $T_N$ on $N$ processors

$$S_N = \frac{T_1}{T_N}$$

The *wall-clock time* can be used instead of the *CPU time*. The wall-clock time can be defined as the whole program time which includes the input and output processes. The CPU time is defined as the time which corresponds to the execution of the program. A super-linear (resp. linear) speedup is achieved if $S_N > N$ (resp. $S_N = N$) [45]. In most of the cases a sub-linear speedup $S_N < N$ is performed, caused by the overhead of the communications costs.

The efficiency $E_N$ with $N$ processors can be formulated as the ratio between the speed-up $S_N$ and the number of processors $N$.

$$E_N = \frac{S_N}{N}$$

When all processors are fully exploited the efficiency will be equal to 100%.

The speedup measure may have different formulations. The selection of a given formulation will depend on the goal of the performance assessment process. One can use the *absolute* speedup, when the sequential time $T_1$ is defined as the best known sequential time to solve the problem. In multi-objective optimization using MOEAs, it is hard to make out the best sequential program. Hence this definition is almost never used. One can use the *relative* speedup when the sequential time $T_1$ is formulated as the parallel algorithm implemented on a single processor. If we take into account the bottlenecks of the architectures where the MOEAs are going to be run (e.g., maximum bandwidth between memory and CPU on shared memory machines), those metrics represent upper bounds. Other metrics to evaluate the performance of parallel MOEAs can be used. An important issue can be the *energy consumption*, for example the energy and the power dissipation to find the Pareto front. Another aspect can be related to the *financial cost* if the implementation of parallel MOEAs is carried out on commercial clouds. Hence the performance evaluation can be itself formulated as a multi-objective optimization problem dealing with speedup, energy consumption and financial aspects.

Various termination criteria may be applied for the algorithms:

- **Number of iterations:** a given a priori fixed number of iterations can be defined. This is the most popular measure for parallel MOEAs. One can obtain a superlinear speedup (i.e. $S_N > N$) [46]. It is justified by the properties of the parallel machine in which there are more resources than a sequential machine. For instance, when having more cache memory in the parallel architecture, the swapping time will be more important for the sequential program.

- **Convergence:** a given quality of solution may be fixed as an achievement. One can use this measure to assess the effectiveness of a parallel MOEA. This measure is useful only for algorithmic-level parallel models of MOEAs, in which the behaviour of the parallel algorithm is different from the sequential one. A super-linear speedup may be obtained and is justified by the properties of the parallel search, in which the landscape of the decision space is not visited by a parallel search in the same order as a sequential search. When dealing with stochastic MOEAs, the average speedup

$$S_N = \frac{E(T_1)}{E(T_N)}$$

must be used.

### *4.3. Main characteristics of parallel MOEAs*

The *granularity* of a parallel MOEA is the most important property of its performance. It can be defined as the ratio between the computation and the communication times. The achieved speedup is better when the granularity is larger. The parallel models of MOEAs have a declining granularity from large grain (algorithmic-level) to fine-grain (solution-level). The maximum amount of parallel processes composing a parallel MOEA represents its *degree of concurrency*.

An important issue in the implementation of parallel MOEAs is the scheduling of the different processes. Various scheduling techniques may be applied:

- **Static**: in static scheduling, the number of processes of MOEAs and their placement are identified before the execution (at compilation). It is an effective strategy for homogeneous architectures, non-volatile and non-shared HPC machines. Undeniably, for heterogeneous architectures, there is a difference in load or power among processors. Hence the time for an iteration of MOEAs is deduced from the least powerful processor or the highly loaded, and then many processes are often idle.

- **Dynamic**: in dynamic scheduling, the placement of processes is defined during their execution but the number of processes is fixed at compile time. It is an effective strategy for shared architectures in which the load of processors cannot be predicted at compile time. It is also an important issue for *non-regular* parallel MOEAs where the execution time of the different processes cannot be efficiently estimated at compile time. It can happen for instance when the evaluation time of the different objective functions depends on the solution itself.

- **Adaptive**: in adaptive scheduling, the set of processes is changing dynamically. The number of processes may vary according to the load of the processors. For instance, a process can be created (resp. killed) when a processor becomes idle (resp. overloaded). Adaptive scheduling is essential for highly volatile HPC machines such as desktop Grids.

Fault-tolerance is an important concern for CPU-time intensive MOEAs implemented on loosely tolerant HPC machines such as NOWs and Grids. One has to apply application-level checkpointing and recovery strategies which are more efficient than system-level ones. Minimal information is stored for application-level checkpointing such as the population of solutions and the iteration number.

### *4.4. Parallel algorithmic-level for MOEAs*

**Granularity:** the parallel algorithmic-level model has a relatively big granularity. Communication cost is generally reduced compared to the execution cost. The communication cost will depend on the frequency of communication and the amount of exchanged knowledge (e.g. set of Pareto solutions). This model is the most appropriate for large scale HPC machines and networks such as NOWs and Grids. The frequency of migration (resp. size of communicated knowledge) must be associated to the latency (resp. bandwidth) of the network. The communication topology between MOEAs can be defined in correlation with the interconnection network.

**Degree of concurrency:** in terms od scalability, the degree of concurrency of the parallel algorithmic-level MOEA is bounded by the number of involved MOEAs. This number is generally tuned according to the effectiveness of the parallel MOEAs and the resources of the target HPC machine. In general there is a threshold in which increasing the number of MOEAs will not allow to improve the quality of results.

**Asynchronous versus synchronous exchange:** the communication exchange in the algorithmic-level parallel model can be synchronous or asynchronous. In the synchronous exchange of knowledge, the MOEAs carry out a synchronization step at a predefined generation. This operation guarantees the same evolution stage for all MOEAs. The synchronous communication is not fault tolerant and then is less efficient on a computational Grid. Indeed a fault of a single MOEA involves the blocking of the global model in a volatile environment.

In the asynchronous approach, a decision criterion for migration is associated to each MOEA [6]. At each iteration, if the criterion is satisfied, the MOEA exchanges some knowledge with the neighbors. If the target HPC machine is heterogeneous, the MOEAs may be at various evolution steps leading to the *super-solution* or *non-effect* situations, i.e. the reception of bad solutions at an advanced stage will generally not bring any improvements for the local population. In the inverse situation, the exchange will precipitate the convergence. An important advantage of asynchronous exchange is the non-blocking operation. It will be more critical for shared HPC machines such as desktop Grids and NOWs (e.g. multiple users). As the load of processors and networks is heterogeneous, the use of synchronous exchange will deteriorate the performances of the parallel MOEA. Indeed, the least powerful processor will arbitrate the performance.

In a volatile HPC machine such as desktop Grids, it is difficult to preserve regular topologies such as torus and rings. The fault of a processor makes necessary a dynamic reconfiguration of the logical topology. This is a costly op-

eration which makes the migration of knowledge inefficient. The use of random and adaptive topologies is recommended for highly volatile HPC machines such as desktop Grids.

**Scheduling:** in the algorithmic-level model of parallel MOEAs, the processes correspond to MOEAs. One can use the following strategies for scheduling MOEAs:

- Static: in static scheduling, the set of MOEAs is fixed and related to the number of processors. Moreover, a static placement of the MOEAs on the processors is carried out. The placement is not modified during the search.
- Dynamic: in dynamic scheduling, the different MOEAs are dynamically placed on the processors. A migration of the MOEAs is carried out during the search according to the load state of the processors.
- Adaptive: in adaptive scheduling, the number of MOEAs implicated in the search can fluctuate dynamically. For instance, a new MOEAs is recovered when a processor becomes idle, and a MOEA is interrupted when a processor becomes busy.

**Fault tolerance:** in this parallel model of MOEAs, the memory state required to checkpoint a parallel MOEA is represented by the search memory of the MOEA (i.e. population of individuals, Pareto archive, iteration number).

### *4.5. Parallel iteration-level for MOEAs*

**Granularity:** the parallel iteration-level model for MOEAs has a medium granularity. The granularity will be determined by the ratio between the evaluation of a sub-population (i.e. partition) and its communication cost. The model is effective when the size of the sub-populations is important or the evaluation of the multiple objectives is time-consuming.

**Scalability:** this model has a degree of concurrency which is bounded by the size of the population. Solving MOPs with many objectives and using large

population sizes will enhance the scalability of the parallel iteration-level model of MOEAs.

**Asynchronous/synchronous exchange:** using asynchronous communications will enhance the effectiveness of the parallel iteration-level model of MOEAs. The evaluation of sub-populations and the generation of solutions must be done in an asynchronous way. For instance, steady-state algorithms may be used in the reproduction phase [47]. Asynchronism will be more crucial for heterogeneous or volatile and shared HPC machines [48]. It is also an essential property for MOPs in which the computation time of the objective functions is not deterministic and depends on the carried solution.

The main advantages of the asynchronous model are fault tolerance and robustness. The synchronous model is blocking and therefore less efficient on a heterogeneous grids. The disappearance of an evaluating process needs the redistribution of its solutions to other processes. Then, it is crucial to keep the solutions not yet evaluated.

**Scheduling:** in this parallel model, processes correspond to the construction/evaluation of a subset of solutions. Hence, the different scheduling strategies will differ as follows:

- Static: in static scheduling, a partitioning of the population is carried out at compilation time. Is it well adapted for homogeneous non-shared machines in which the population is splitted into equal size partitions related to the number of processors. A static placement of the partitions on the processors is carried out. For heterogeneous non-shared machines, the size of each partition must be initialized in relation to the performance of the nodes. Moreover, the static mapping approach is not efficient for variable computational costs of equal partitions. For instance, it can happen for optimization problems where various CPU-times are associated to the evaluation of different solutions.

- Dynamic: in dynamic scheduling, a migration of tasks can be handled

during the execution of MOEAs. The migration decisions depend on the load of processors. In general the number of processes is related to the size of the population and the number of objectives. Many processes may be scheduled on the same processor. The approach based on the master-workers cycle stealing may be used. To each worker is first assigned a small number of individuals. Once it has performed its iterations the worker requests from the master additional individuals. All the workers are stopped once the final result is returned. Faster and less loaded processors handle more solutions than the others. This approach allows to reduce the execution time compared to the static scheduling approach.

- Adaptive: in adaptive scheduling the number of generated partitions depend on the load of the target HPC machine. This approach is more efficient for shared, volatile and heterogeneous machines such as desktop Grids.

**Fault-tolerance:** the memory of the parallel iteration-level model needed for checkpointing is collected from the partitions. The partitions are composed of a set of (partial) solutions and their associated fitnesses.

### *4.6. Parallel solution-level for MOEAs*

**Granularity:** the parallel solution-level has the smallest granularity. In the functional decomposition model, the granularity is the ratio between the evaluation cost of the sub-functions and the communication cost of an individual. In the data decomposition model, the granularity is the ratio between the evaluation of a data partition and its communication cost.

This parallel model is less appropriate for large-scale distributed machines in which the communication cost (in terms of bandwidth and/or latency ) is significant, such as in computational Grids. Indeed, its implementation is often restricted to clusters or shared memory machines.

**Scalability:** the degree of concurrency of the model is limited by the number of data partitions or sub-functions. The use of the solution-level parallel model

in conjunction with the other parallel models helps to extend the scalability of parallel MOEAs.

**Asynchronous/synchronous communications:** the implementation of this model is always synchronous. In general a master/workers paradigm is implemented. The master must wait for all partial results to compute the global value of the objective functions. The execution time $T$ will be bounded by the maximum time $T_i$ of the different tasks. An exception occurs for highly-constrained MOPs, where feasibility of the solution is first checked. The master terminates as soon as a given process detects that the individual does not satisfy a given constraint. Due to its highly synchronization steps, this parallel model is worth applying to MOPs in which the computations needed at each iteration are CPU-time consuming. The relative speedup may be computed as follows:

$$S_n = \frac{T}{\alpha + T/n}$$

where $\alpha$ is the communication cost and $n$ is the number of processors.

**Scheduling:** in the parallel solution-level model, processes correspond to data partitions in the data decomposition model and to sub-functions in the functional decomposition. The different scheduling approaches can be defined as follows:

- Static: in static scheduling, the date or sub-functions are decomposed into equal size partitions. A static placement between the data partitions (or sub-functions) and the processors is carried out. This static scheme is efficient for parallel homogeneous non-shared HPC machines. For a heterogeneous non-shared machine, the size of each partition in terms of sub-functions or data must be tuned according to the performance of the processors.

- Dynamic: dynamic scheduling is crucial for shared parallel machines and MOEAs characterized by variable costs for the sub-functions or data partitions. Dynamic load balancing may be easily achieved by evenly distributing at run-time the sub-functions or the data among the nodes. In

MOPs where the computing cost of the sub-functions is unpredictable, dynamic scheduling is necessary.

- Adaptive: in adaptive scheduling, the number of generated data partitions or sub-functions vary in function of the load of the processors. This approach is more appropriate to shared, volatile and heterogeneous parallel machines such as desktop Grids.

**Fault-tolerance:** the memory of the solution-level parallel model needed for the checkpointing mechanism is straightforward. It is composed of the solution(s) and their partial objective values.

Depending on the target parallel machine, table 3 shows a guideline for the efficient implementation of the various models of parallel MOEAs. For each model (algorithmic-level, iteration-level, solution-level), the table presents its characteristics according to the highlighted criteria (granularity, scalability, asynchronism, scheduling and fault-tolerance).

Table 3: Efficient implementation of parallel MOEAs.

| **Property** | **Algorithmic-level** | **Iteration-level** | **Solution-level** |
|---|---|---|---|
| Granularity | Coarse (Frequency of exchange, size of information) | Medium (Nb. of solutions per partition) | Fine (Eval. sub-functions, eval. data partitions) |
| Scalability | Number of MOEAs | Neighborhood size, populations size | Nb. of sub-functions, nb. data partitions |
| Asynchronism | High (Information exchange) | Moderate (Eval. of solutions) | Exceptional (Feasibility test) |
| Scheduling and Fault-tolerance | MOEA | Solution(s) | Partial solution(s) |

## 5. Conclusions and perspectives

The development of parallel MOEAs allows to improve speedup of the search, the quality of the approximated Pareto front, the robustness, and to solve large

scale MOPs. The distinction between parallel design and implementation issues of MOEAs is crucial to analyze parallel MOEAs. The essential conclusions of this paper can be summarized as follows:

- In terms of design of parallel MOEAs, the various parallel models have been unified. Three complementary hierarchical models have been highlighted: algorithmic-level, iteration-level and solution-level parallel models.

- In terms of implementation of parallel MOEAs, we deal with the question of an efficient placement of a parallel MOEA on a given parallel machine. The focus was made on the main criteria of parallel machines that have an impact on the efficiency of parallel MOEAs.

This paper illustrates the unified parallel models on evolutionary algorithms. However, the proposed framework can be extended to any population-based metaheuristic such as scatter search, particle swarm, and ant colonies.

In the next years, the main perspective is to attain Exascale performance. The emanation of heterogeneous and hybrid HPC machines build of multi-cores and many-cores will accelerate the accomplishment of this goal. In terms of parallel programming environments, cloud computing will become an important alternative to traditional high performance computing for the development of MOEAs that harness large scale computational resources. This is an important challenge as nowadays cloud software frameworks for parallel MOEAs are just emerging.

In the future design of high-performance computers, the ratio between power and performance will be important for sustainable supercomputing. The power represents the electrical power consumption of the machine. In the last year, the greenest HPC machines in the world doubled their energy efficiency, thanks to the manycore processors, GPUs and ARM based architectures. Exascale HPC machines operating at less than 20MW will be realizable in two years.

In terms of solving multi-objective optimization problems, parallel MOEAs constitute inevitable strategies for big real-life applications such as engineering design and machine learning. Parallel MOEAs represent an important solution to solve dynamic real-time MOPs [49], large-scale and many-objective optimization problems [50]. Moreover, parallel models for MOPs with random or epistemic uncertainties have to be developed. The sources of uncertainty in MOPs are due to many factors such as environment parameters, decision variables and objectives functions.

## References


[1] M. Camara, J. Ortega, F. D. Toro, A single front genetic algorithm for parallel multi-objective optimization in dynamic environments, Neurocomputing 72 (16) (2009) 3570 – 3579.

[2] C. Barba-Gonzalz, J. Garca-Nieto, A. J. Nebro, J. F. Aldana-Montes, Multi-objective big data optimization with jmetal and spark, in: Evolutionary Multi-Criterion Optimization. EMO 2017. Lecture Notes in Computer Science, Vol.10173, 2017, pp. 16–30.

[3] J. Figueira, A. Liefooghe, E.-G. Talbi, A. Wierzbicki, A parallel multiple reference point approach for multi-objective optimization, European Journal of Operational Research 205 (2) (2010) 390–400.

[4] D. A. van Veldhuizen, J. B. Zydallis, G. B. Lamont, Considerations in engineering parallel multi-objective evolutionary algorithms, IEEE Trans. on Evolutionary Computation 7 (2) (2003) 144–173.

[5] G. Yao, Y. Ding, Y. Jin, K. Hao, Endocrine-based coevolutionary multi-swarm for multi-objective workflow scheduling in a cloud system, Soft Computing 21 (15) (2017) 4309–4322.

[6] R. H. Gómez, C. Coello, E. Alba, A parallel version of SMS-EMOA for many-objective optimization problems, in: International Conference on Parallel Problem Solving from Nature, 2016, pp. 568–577.

[7] W. Ying, Y. Xie, Y. Wu, B. Wu, S. Chen, W. He, Universal partially evolved parallelization of moea/d for multi-objective optimization on message-passing clusters, Applied Soft Computing (18) (2017) 5399–5412.

[8] S. Nesmachnow, Parallel multiobjective evolutionary algorithms for batch scheduling in heterogeneous computing and grid systems, Comp. Opt. and Appl. 55 (2) (2013) 515–544.

[9] T. J. Stanley, T. Mudge, A parallel genetic algorithm for multi-objetive microprocessor design, in: Proc. of the Sixth Int. Conf. on Genetic Algorithms, 1995, pp. 597–604.

[10] J.-J. Durillo, Q. Zhang, A. Nebro, E. Alba, Distribution of computational effort in parallel moea/d., in: LION Learning and Intelligent Optimization, Springer, 2011, pp. 488–502.

[11] E.-G. Talbi, S. Mostaghim, H. Ishibushi, T. Okabe, G. Rudolph, C. C. Coello, Multiobjective optimization: Interactive and evolutionary Approaches, Vol. 5252 of LNCS, Springer, 2008, Ch. Parallel approaches for multi-objective optimization, pp. 349–372.

[12] S. Mostaghim, J. Branke, H. Schmeck, Multi-objective particle swarm optimization on computer grids, in: Genetic and Evolutionary Computation Conference (GECCO'07), ACM, London, UK, 2007, pp. 869–875.

[13] M. Basseur, F. Seynhaeve, E.-G. Talbi, Adaptive mechanisms for multi-objective evolutionary algorithms, in: Congress on Engineering in System Application CESA'03, Lille, France, 2003, pp. 72–86.

[14] F. de Toro, J. Ortega, E. Ros, S. Mota, B. Paechter, J. Martín, PSFGA: Parallel processing and evolutionary computation for multiobjective optimisation, Parallel Computing 30 (5-6) (2004) 721–739.

[15] C. Coello, M. Reyes, A study of the parallelization of a coevolutionary multi-objective evolutionary algorithm, in: MICAI'2004, LNAI Vol.2972, 2004, pp. 688–697.

[16] A. J. Nebro, F. Luna, E.-G. Talbi, E. Alba, Parallel multiobjective optimization, in: E. Alba (Ed.), Parallel metaheuristics, Wiley, 2005, pp. 371–394.

[17] J. Rowe, K. Vinsen, N. Marvin, Parallel GAs for multiobjective functions, in: Proc. of the 2nd Nordic Workshop on Genetic Algorithms and Their Applications (2NWGA), 1996, pp. 61–70.

[18] S. Duarte, B. Barán, Multiobjective network design optimisation using parallel evolutionary algorithms, in: XXVII Conferencia Latinoamericana de Informática CLEI'2001, 2001.

[19] K. E. Parsopoulos, D. K. Tasoulis, N. G. Pavlidis, V. P. Plagianakos, M. N. Vrahatis, Vector evaluated differential evolution for multiobjective optimization, in: Proc. of the IEEE 2004 Congress on Evolutionary Computation (CEC'04), 2004.

[20] G. Zavala, A. Nebro, F. Luna, C. Coello, A survey of multi-objective metaheuristics applied to structural optimization, Structural and Multidisciplinary Optimization 49 (4) (2014) 537–558.

[21] G. P. Rangaiah, Multi-objective optimization: techniques and applications in chemical engineering, Vol. 5, World Scientific, 2016.

[22] M. Zhang, W. Gou, L. Li, F. Yang, Z. Yue, Multidisciplinary design and multi-objective optimization on guide fins of twin-web disk using kriging surrogate model, Structural and Multidisciplinary Optimization 55 (1) (2017) 361–373.

[23] A. Lančinskas, J. Žilinskas, Approaches to parallelize pareto ranking in nsga-ii algorithm, in: International Conference on Parallel Processing and Applied Mathematics, Springer, 2011, pp. 371–380.

[24] A. M. Mora, P. García-Sánchez, J. J. M. Guervós, P. A. Castillo, Pareto-based multi-colony multi-objective ant colony optimization algorithms: an island model proposal, Soft Computing 17 (7) (2013) 1175–1207.

[25] M. Pedemonte, S. Nesmachnow, H. Cancela, A survey on parallel ant colony optimization, Applied Soft Computing 11 (8) (2011) 5181–5197.

[26] N. Srinivas, K. Deb, Multiobjective optimization using non-dominated sorting in genetic algorithms, Evolutionary Computation 2 (3) (1995) 221–248.

[27] R. Szmit, A. Barak, Evolution strategies for a parrallel multi-objective genetic algorithm, in: D. W. et al. (Ed.), Proc. of the Genetic and Evolutionary Computation Conference, Morgan Kaufmann, 2000, pp. 227–234.

[28] E. Talbi, S. Cahon, N. Melab, Designing cellular networks using a parallel hybrid metaheuristic on the computational grid, Computer Communications 30 (4) (2007) 698–713.

[29] E. G. Talbi, S. Cahon, N. Melab, Designing cellular networks using a parallel hybrid metaheuristic on the computational grid, Comput. Commun. 30 (4) (2007) 698–713.

[30] I. Foster, C. K. (eds.), The grid: Blueprint for a new computing infrastructure, Morgan Kaufmann, San Fransisco, 1999.

[31] J. J. Moreno, G. Ortega, E. Filatovas, J. A. Martinez, E.-M. Garzon, Using low-power platforms for evolutionary multi-objective optimization algorithms, Journal of Supercomputing 73 (1) (2017) 302–315.

[32] R. Zeidman, Designing with FPGAs and CPLDs, Elsevier, USA, 2002.

[33] M. Letras, A. Morales-Reyes, R. Cumplido, A scalable and customizable processor array for implementing cellular genetic algorithms, Neurocomputing 175 (2016) 899–910.

[34] P. Santos, J. C. Alves, J. C. Ferreira, A reconfigurable custom machine for accelerating cellular genetic algorithms, U. Porto Journal of Engineering 2 (2) (2016) 2–13.

[35] G. Mamdoohi, A. F. Abas, K. Samsudin, N. H. Ibrahim, A. Hidayat, M. A. Mahdi, Implementation of genetic algorithm in an embedded

microcontroller-based polarization control system, Engineering Applications of Artificial Intelligence 25 (4) (2012) 869–873.

[36] A. R. Brodtkorb, T. R. Hagen, M. L. SæTra, Graphics processing unit (gpu) programming strategies and trends in gpu computing, J. Parallel Distrib. Comput. 73 (1) (2013) 4–13.

[37] Z. Li, Y. Bian, R. Zhao, J. Cheng, A fine-grained parallel multi-objective test case prioritization on gpu, in: Proceedings of the 5th International Symposium on Search Based Software Engineering (SSBSE '13), Vol. 8084, Springer, St. Petersburg, Russia, 2013, pp. 111–125.

[38] T. V. Luong, E. Taillard, N. Melab, E.-G. Talbi, Parallelization strategies for hybrid metaheuristics using a single gpu and multi-core resources, in: C. A. C. Coello, V. Cutello, K. Deb, S. Forrest, G. Nicosia, M. Pavone (Eds.), Parallel Problem Solving from Nature - PPSN XII, Vol. 7492 of Lecture Notes in Computer Science, Springer, 2012, pp. 368–377.

[39] S. Gupta, G. Tan, A scalable parallel implementation of evolutionary algorithms for multi-objective optimization on GPUs, in: IEEE Congress on Evolutionary Computation (CEC), IEEE, 2015, pp. 1567–1574.

[40] A. Nasrollahzadeh, G. Karimian, A. Mehrafsa, Implementation of neuro-fuzzy system with modified high performance genetic algorithm on embedded systems, Applied Soft Computing, to appear 2017.

[41] D. R. Butenhof, Programming with POSIX threads, Addison-Wesley, USA, 1997.

[42] P. Hyde, Java thread programming, Sams, 1999.

[43] B. Chapman, G. Jost, R. VanderPas, D. J. Kuck, Using OpenMP: Portable Shared Memory Parallel Programming, MIT Press, USA, 2007.

[44] V. Kumar, A. Grama, A. Gupta, G. Karypis, Introduction to parallel computing: Design and analysis of algorithms, Addison Wesley, 1994.

[45] E. G. Talbi, P. Bessière, Superlinear speedup of a parallel genetic algorithm on the supernode, SIAM News 24 (4) (1991) 12–27.

[46] B. Dorronsoro, G. Danoy, A. J. Nebro, P. Bouvry, Achieving super-linear performance in parallel multi-objective evolutionary algorithms by means of cooperative coevolution, Computers & Operations Research 40 (6) (2013) 1552–1563.

[47] M. Depolli, R. Trobec, B. Filipič, Asynchronous master-slave parallelization of differential evolution for multi-objective optimization, Evolutionary Computation 21 (2) (2013) 261–291.

[48] S. Mostaghim, J. Branke, A. Lewis, H. Schmeck, Parallel multi-objective optimization using master-slave model on heterogeneous resources, in: IEEE Congress on Evolutionary Computation CEC'2008, 2008, pp. 1981–1987.

[49] Y. Dujardin, D. Vanderpooten, F. Boillot, A multi-objective interactive system for adaptive traffic control, European Journal of Operational Research 244 (2) (2015) 601–610.

[50] R. Cheng, Y. Jin, M. Olhofer, B. Sendhoff, Test problems for large-scale multiobjective and many-objective optimization, IEEE Transactions on Cybernetics 1 (99) (2016) 1–14.