13

Parallel Approaches for Multiobjective Optimization

El-Ghazali Talbi[1], Sanaz Mostaghim[2], Tatsuya Okabe[3], Hisao Ishibuchi[4], Günter Rudolph[5], and Carlos A. Coello Coello[6]

[1] Laboratoire d'Informatique Fondamentale de Lille
Université des Sciences et Technologies de Lille
59655 - Villeneuve d'Ascq cedex, France
talbi@lifl.fr

[2] Institute AIFB
University of Karlsruhe
76128 Karlsruhe, Germany
mostaghim@aifb.uni-karlsruhe.de

[3] Honda Research Institute Japan Co., Ltd.
8-1 Honcho, Wako-City, Saitama, 351-0188, Japan
okabe@jp.honda-ri.com

[4] Department of Computer Science and Intelligent Systems
Osaka Prefecture University
Osaka 599-8531, Japan
hisaoi@cs.osakafu-u.ac.jp

[5] Computational Intelligence Research Group
Chair of Algorithm Engineering (LS XI)
Department of Computer Science, University of Dortmund
44227 Dortmund, Germany Guenter.Rudolphuni-dortmund.de

[6] CINVESTAV-IPN (Evolutionary Computation Group)
Depto. de Computación, Av. IPN No 2508
Col. San Pedro Zacatenco, México, D.F., 07360 MEXICO
ccoello@cs.cinvestav.mx

Abstract. This chapter presents a general overview of parallel approaches for multiobjective optimization. For this purpose, we propose a taxonomy for parallel metaheuristics and exact methods. This chapter covers the design aspect of the algorithms as well as the implementation aspects on different parallel and distributed architectures.

Key words: Parallel algorithms, Parallel metaheuristics, Parallel multiobjective optimization, Parallel exact optimization

Reviewed by: Heinrich Braun, SAP AG, Walldorf, Germany
Jürgen Branke, University of Karlsruhe, Germany

13.1 Introduction

Multiobjective optimization problems are often NP-hard, complex and CPU time consuming. Exact methods can be used to find the exact Pareto front (or a subset of the front), but they are impractical to solve large problems as they are time and memory consuming. On the other hand, metaheuristics provide the approximated Pareto fronts in a reasonable time. However, they also remain time-consuming for solving large problems.

Parallel and distributed computing are used in the design and implementation of multiobjective optimization algorithms to speedup the search. Also, they are used to improve the precision of the used mathematical models, the quality of the obtained Pareto fronts, the robustness of the obtained solutions, and to solve large scale problems.

In this chapter, we present the main parallel models for metaheuristics and exact methods from the algorithmic design point of view. We consider continuous and combinatorial optimization problems as parallel models are suited either for combinatorial or continuous optimization problems. From the implementation point of view, we concentrate on the parallelization of multiobjective optimization algorithms on general-purpose parallel and distributed architectures as these architectures are the most widespread computation platforms. The rapid evolution of technology in terms of processors (multi-core), networks (Infiniband), and architectures (GRIDs, clusters) make the parallelization very popular.

Different architectural criteria which affect the efficiency of the implementation are shared memory / distributed memory, homogeneous / heterogeneous, dedicated / non dedicated, local network / large network. Indeed, these criteria have a strong impact on the deployment techniques such as load balancing and fault-tolerance. Depending on the type of the used architecture, different parallel and distributed programming environments such as message passing (PVM, MPI), shared memory (multi-threading, OpenMP), high throughput computing (Condor), and Grid computing (Globus) can be used.

This chapter is organized as follows. In the next section, we present the parallel models for designing metaheuristics for MOPs. In Section 3, we review the parallel models for exact algorithms. Section 4 deals with the implementation issues for metaheuristics and exact algorithms. Finally, we conclude the paper and discuss several lines for future research in Section 5.

13.2 Parallel Models for Metaheuristics

Different parallel models for metaheuristics have been proposed in the literature. They follow three major hierarchical models such as:

- Self-contained parallel cooperation (between different algorithms)

- Problem independent intra-algorithm parallelization
- Problem dependent intra-algorithm parallelization

where the last two models do not alter the behavior of the algorithms and therefore are generally used to speedup the search.

13.2.1 Level 1: Self-Contained Parallel Cooperation

Basic Concept

This group of parallel algorithms containing *the Island model* is used for parallel systems with very limited communication. In the island model, every processor runs an independent MOEA using a separate (sub)population. The processors might cooperate by regularly exchanging migrants which are good individuals in their subpopulations. These algorithms are also suitable for problems with large search spaces where a large population is being required. The large population is then being divided into several subpopulations.

In every processor, an optimization algorithm with selection and recombination operators is being carried out on a subpopulation. As written by Coello *et al.* (2002), there are several methods (also based on the island model) in the literature which we can categorize into two main groups. (1) Cooperating Subpopulations: These methods are based on partitioning the objective/search space. In this group, the population is divided into subpopulations. The number of subpopulations and the way the population is divided are the two key issues. (2) Multi-start Approach: Here, each processor independently runs an optimization algorithm.

Group 1: Cooperating Subpopulations

These algorithms attempt to distribute the task of finding the entire Pareto-optimal front among participating processors. By this way, each processor is destined to find a particular portion of the Pareto-optimal front. In fact, the population of a MOEA is divided into a number of independent and separate subpopulations resulting in several small separate MOEAs executing simultaneously which have the responsibility to find the (Pareto-)optimal solutions in their own search region. Each MOEA could have different operators, parameter values, as well as a different structure. In this model, some individuals within some particular subpopulations occasionally migrate to another one.

Generally, when distributing the task among the processors, the overlap between the solutions of two processors should be as small as possible. Also, the distribution algorithm must be scalable. Usually, the designer or a computational resource (master node) is responsible for distributing and dividing the population or the objective/search space.

In the literature, the very first approaches based on the island model do not directly divide the objective/search space into different regions, but implicitly result in the division as studied by Baita *et al.* (1995); Poloni (1995); Hiroyasu *et al.* (2000); Jozefowiez *et al.* (2002); Deb *et al.* (2003); Xiao and Armstrong (2003); de Toro Negro *et al.* (2004).

Baita *et al.* (1995) and Poloni (1995) use a local geographic selection scheme in which individuals are placed on a toroidal grid with one individual per grid intersection point. Hiroyasu *et al.* (2000) proposed the Divided Range Multi-Objective Genetic Algorithm (DRMOGA) in which the global population is sorted according to one of the objective functions (which is changed after a number of generations). Then, the population is divided into equally-sized sub-populations. Each of these sub-populations is allocated to a different processor in which a serial MOEA is applied. After a certain number of generations, the sub-populations are gathered and the process is repeated, but this time using some other objective function as the sorting criterion. The main goal of this approach is to focus the search effort of the population on different regions of the objective space. However, in this approach we cannot guarantee that the sub-populations remain in their assigned region. A similar approach is followed by de Toro Negro *et al.* (2004). Deb *et al.* (2003) use a modified domination criterion for assigning a specific region of the objective space to a processor.

Zhu and Leung (2002); Zhu (2002) proposed the Asynchronous Self-Adjustable Island Genetic Algorithm (aSAIGA) in which, rather than migrating a set of individuals, the islands exchange information related to their current explored region. Based on the information coming from other islands, a **self-adjusting** operation modifies the fitness of the individuals in the island to prevent two islands from exploring the same region. In a similar way to DRMOGA, this approach cannot guarantee that the sub-populations move tightly together throughout the search space, hence the information about the explored region may be meaningless.

Xiao and Armstrong (2003) use a generalized version of VEGA (Vector Evaluated Genetic Algorithm, Schaffer (1985)) to divide the population into subpopulations.

López-Jaimes and Coello (2005) proposed an approach called Multiple Resolution Multi-Objective Genetic Algorithm (MRMOGA), whose main idea is to encode the solutions using a different resolution in each island (heterogeneous nodes are assumed). Then, the variable decision space is divided into hierarchical levels with well-defined overlaps. Evidently, migration is only allowed in one direction (from low resolution to high resolution islands). A conversion scheme is required when migrating individuals, so that the resolution is properly adjusted. This approach uses an external population (or elitist archive) and the migration strategy considers such a population as well. The approach also uses a strategy to detect nominal convergence of the islands in order to increase their initial resolution. The rationale behind this approach is that the true Pareto front can be reached faster using this change of resolution in the islands, because the search space of the low resolution islands is proportionally smaller and, therefore, convergence is faster. This issue was originally identified by Parmee and Vekeria (1997) when they used an injection island strategy to solve a single-objective engineering optimization problem.

In the method proposed by Jozefowiez *et al.* (2005), each processor has its own population which is defined in the entire search space. The defined communication network between the processors is a ring where the processors send half of their populations to their two neighbors. The computations of a given processor do not begin until it has received the information from its two neighbors.

The first approach on dividing the objective space into several regions is introduced by Branke *et al.* (2004). This technique called Cone Separation divides the objective space into subspaces and assigns each subspace to one processor. They, however, do not divide the search space and therefore each processor explores the entire search space. The solutions outside the defined region in the objective space of each processor are considered as infeasible (although in reality they are feasible). Those infeasible solutions are migrated to other processors. This algorithm is scalable and there is no overlap between the solutions obtained by each processor. The so-called hypergraph has been used by Mehnen *et al.* (2004) to structure the populations in MOEAs and then applied it to parallel MOEAs. Streichert *et al.* (2005) refined the idea of the Cone Separation technique by using a clustering method for finding the right partitions in the objective space.

More recently, Bui *et al.* (2006) study an approach for dividing the search space. In their approach, they select a random (hyper-)sphere as the search space for every single processor. Then every processor runs a MOEA inside its defined region. The spheres are evaluated in terms of their solutions and their positions are being improved in the search space for the next iteration(s). This has been done beside other techniques like racing model using Multi-Objective Particle Swarm Optimization.

All of these methods work on processors which have similar properties in other words homogeneous systems. Mostaghim *et al.* (2007) study an approach which works asynchronously and is thus particularly suitable for heterogeneous computer clusters as occurring, e.g., in modern grid computing platforms.

Group 2: Multi-start Approach

This model introduced by Mezmaz *et al.* (2006) consists of several parallel local search algorithms which are independently run on several (also heterogeneous) processors. The basic idea of using such a model is that running several optimization algorithms with different initial seeds is more valuable than executing only one single run for a very long time. This is of particular importance for local search algorithms. Jozefowiez *et al.* (2007) use a parallel hybrid approach combining the multi-start model and the self contained parallel cooperation model. The Pareto front found by a parallel EA is partitioned and serves as a guide to multiple tabu search tasks.

Synchronous versus Asynchronous

Usually in MOEAs a set of non-dominated solution are found as the result of the optimization. In case of using the cooperating subpopulation model every single processor will cooperate to obtain one part of the non-dominated set. In elitist MOEAs like SPEA2 the non-dominated solutions are usually stored in an archive. In other algorithms like the NSGA-II, there is no archive as the main population contains the non-dominated solutions. In any of these cases, the set of non-dominated solutions must be updated as soon as a processor finishes its optimization task.

Apart from the way the subpopulations are created, we must ensure that the processors obtain good convergence and diversity of solutions. For this in some cases each processor can run the optimization several times as shown in Algorithm 3. Algorithm 3 is basically being used on a set of homogeneous

Algorithm 3 Synchronous cooperating subpopulations

```
Initiate subpopulations
repeat
  Wait for results of all processors
  Migration of individuals if any
  Update archive if any
until Termination condition met
Return archive
```

systems. The termination criterion could be a fixed number of runs on each processor (in many cases one iteration has been selected). In this algorithm "Initiate Subpopulation" deals with dividing the objective/search space in order to build the subpopulations. "Migration of individuals" refers to methods in which processors communicate with each other and exchange some of their individuals as migrants.

In reality, we typically deal with heterogeneous systems where this Algorithm is not suitable. In heterogeneous systems, there are different computing resources including very fast and very slow processors. According to Algorithm 3, all of the processors have to wait for the slowest one. In order to deal with these systems, Algorithm 4 is proposed. In this algorithm, whenever a processor returns its results, they can be immediately integrated into the archive. Based on the quality of the obtained archive a suitable new subpopulation can be selected for that processor. This makes the approach particularly suitable for heterogeneous computer clusters such as Grids, where very fast processors are used along with rather slow ones. It is not necessary to wait for the slowest processor to return its results. Here the processors can indirectly communicate through the archive. We must notice that migration is not straightforward as before.

Algorithm 4 Asynchronous cooperating subpopulations (Heterogeneous systems)

```
Initiate an empty archive
Initiate subpopulations
repeat
  if A processor returns results then
    Update archive
    Determine its new subpopulation
  end if
until Termination condition met
Return archive
```

Mostaghim *et al.* (2007) integrate a hypervolume based method into the optimization routine in every processor. For initializing a subpopulation, a guide is selected according to its marginal hypervolume. The hypervolume is the area dominated by all solutions stored in the archive (Chapter 14). The *marginal* hypervolume of a solution is the area dominated by the solution that is not dominated by any other solution. The guide is the solution from the archive which has not been selected before and which has the largest marginal hypervolume. After selecting the guide, a Multi-Objective Particle Swarm Optimization method is used to move its subpopulation toward the guide, hence searching the area around the guide.

13.2.2 Level 2: Problem Independent Parallel Intra-algorithm

Most of the metaheuristics are iterative methods. In this model, we will parallelize a single iteration of the algorithm. Our concern in this model are only search mechanisms which are problem independent such as the evaluation of the neighborhood in local search and the reproduction mechanism in evolutionary algorithms.

Basic Concept

During an optimization, we have to evaluate fitness values of candidates of solution (individuals). If we use benchmark problems/simple applications to evaluate fitness values, the calculation time is negligible. However, a real application sometimes needs huge computational time, e.g., using computational fluid dynamics (CFD), electro-magnetic field analysis, finite element method (FEM) etc. See Okabe *et al.* (2003); Okabe (2004). In this situation, total calculation time becomes too huge and it is generally impossible to obtain a certain result in a reasonable calculation time.

Let assume that the number of individuals, the maximum number of generations, the number of objectives and the calculation time of ith objective function are n, g, k and t_i, respectively. The total calculation time in evolutionary multiobjective optimization, denote T, can be easily calculated as follows:

$$T = gn \sum_{i=1}^{k} t_i + g\alpha = gnt + g\alpha, \tag{13.1}$$

where, α is the time that genetic operator needs in one generation and $t = \sum_{i=1}^{k} t_i$. If $n = 100$, $g = 500$, $\alpha \approx 0$ and $t = 3$ $(days)$ which is a certain real example using CFD solver, the total calculation time is about 411 years! Nevertheless, the problem should be optimized.

To tackle this problem, a parallel calculation is often used. The basic idea is shown in Fig. 13.1. This type of parallelization is called *master-slave model* or *global parallelization*, e.g., Branke *et al.* (2004); Cantu-Paz (1997a); Veldhuizen *et al.* (2003). The optimizer running on a master node carries out an overall calculation including initialization, crossover, mutation and selection except for evaluation of individuals. In evolutionary computation, several individuals exist in a population to be evaluated. However, the evaluation of each individual is completely independent from other evaluations. Therefore, in Fig. 13.1, each evaluation will be done on different slave nodes. The master node generates a population, e.g. car designs. Then, the master node distributes individuals to several independent slave nodes. In the slave nodes, the evaluations of individual, e.g. car design, are carried out simultaneously. Thereafter, the fitness values are gathered by the master node. Based on the fitness values, the master node selects promising individuals and generates new individuals by genetic operators. This flow is repeated until a given termination condition is met. Since several time-consuming evaluations are carried out at the same time, the total calculation time is dramatically reduced.

Calculation Time

Now, we will consider when we should parallelize a calculation using master-slave model. Assume that the total calculation time without/with parallelization and the number of nodes are T^{wo}, T^{w} and N, respectively. As an example, $n = N$ (the number of available nodes is the same as the number of individuals) is also assumed. Since a master node can be used not only for managing total calculation but also for fitness evaluation, a master node also contributes to fitness evaluation. One can easily obtain the following equations of T^{wo} and T^{w}:

$$T^{wo} = gnt + g\alpha = gNt + g\alpha, \tag{13.2}$$

$$T^{w} = g\alpha + gt + g(N-1)T_{DT}, \tag{13.3}$$

where, T_{DT} is the necessary time for data transfer from the master node to one slave node and from one slave node to the master node in one generation.

Now, the efficiency of parallelization, denoted as η, is calculated as:

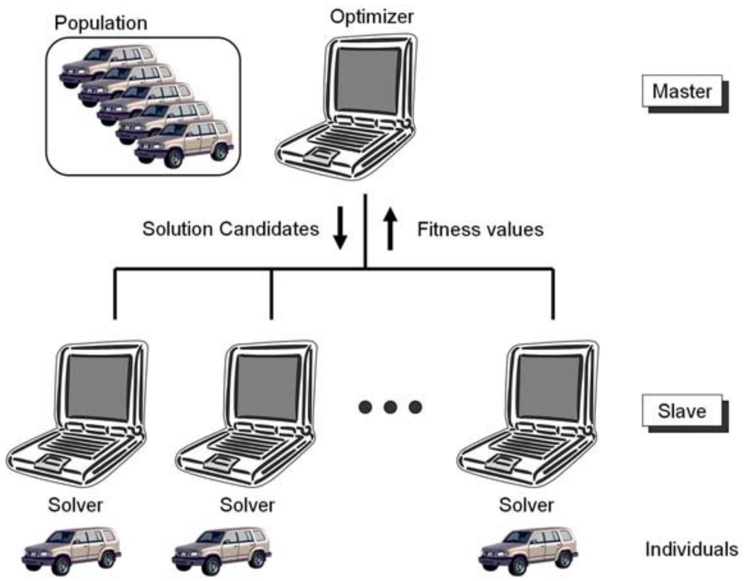


Fig. 13.1. Master-slave model for parallelization.

$$\eta = \frac{T^{wo}}{NT^{w}} \times 100(\%). \tag{13.4}$$

The numerator is the total resources for calculation when the parallelization is not used. The denominator is the total resources for calculation when parallelization is used. Since one master node and $(N-1)$ slave nodes are occupied for the time of T^w, the total resources are NT^w. If η becomes 100%, the parallelization is very useful. Oppositely, if η becomes 0%, the parallelization should not be done.

Using Eq. (13.2) and Eq. (13.3), Eq. (13.4) can be calculated as follows:

$$\eta = \frac{Nt + \alpha}{N(t + \alpha + (N-1)T_{DT})} \times 100. \tag{13.5}$$

Eq. (13.5) leads the following results:

- If $\alpha << t$ and $T_{DT} << t$, η is nearly 100%. This means that if the necessary calculation time for one fitness evaluation is sufficiently larger than α (for genetic operators) and T_{DT} (for data transfer), we should parallelize a calculation.
- If $\alpha \approx 0$, one can easily obtain the following relation:

$$\eta = \left(1 - \frac{(N-1)T_{DT}}{(N-1)T_{DT} + t}\right) \times 100. \tag{13.6}$$

This equation means that the smaller the value of t is, the worse is the efficiency of η.

Survey

From the beginning of the research for evolutionary algorithms on single objective optimization, parallelization technique has been paid attention due to population-based approach of evolutionary algorithms. There are many surveys in the literature, e.g., Schmeck *et al.* (2001); Bethke (1976); Adamidis (1994); Cantu-Paz (1997a,b). As a natural extension, parallelization is also used for evolutionary multiobjective optimization.

Stanley and Mudge (1995) propose the framework of parallel genetic algorithm called *Genetic Algorithm running on the INternet (GAIN)*. The usage of different architectures for parallel computation is often due to the fact that homogenous computers are not readily available. This situation leads to different computational time of fitness evaluations on slave nodes. If the computational time on a certain node is different from others, the efficiency of parallel computation decreases dramatically due to much idle time of faster slave nodes. To solve this problem, Stanley and Mudge propose the GAIN. Based on a given parameter that determines the maximum number of pending evaluations, the idle time is reduced. If the number of unevaluated individuals exceeds this number, the generation process sleeps. Otherwise, the generation process is carried out even if unevaluated individuals exist. The results of the GAIN show a robust and good performance.

Watanabe *et al.* (2002) extend an original master-slave model to maintain a higher diversity of the population. They call this extension as *Master-slave model with local cultivation (MSLC) model.* In this model, two randomly selected individuals are sent to a slave node. Using two individuals, most genetic operators are carried out in a slave node. However, in one generation, all individuals distributed to slave nodes are gathered and ranked again on the master node. Since most of calculation is done on slave node, the problem occurred on a master-slave node, i.e. higher computational cost of a master slave, is solved.

de Toro Negro *et al.* (2002) propose the parallel multiobjective evolutionary algorithm called *Parallel Single Front Genetic Algorithm (PSFGA)* as an extension of *Single Front Genetic Algorithm (SFGA)* based on master-slave model. The characteristic of the SFGA are as follows: Only the non-dominated individuals can join the recombination process, all non-dominated individuals are copied to the next population and the rest of individuals to complete the population are obtained by recombination and mutation of the non-dominated individuals. In PSFGA, the population is divided into several sub-populations based on fitness values. In the sub-population, the original SFGA is carried out. After execution of SFGA, all individuals are gathered by a master node. They conclude that parallelization is very helpful not only for the reduction of computational cost but also for the preservation of diversity.

Coello and Sierra (2004) study the parallelization of a coevolutionary multiobjective evolutionary algorithm. Based on the master-slave model, they parallelize their algorithm. The population is divided into several sub-population according to search region. In each generation, sup-populations cooperate or compete. In Coello and Sierra (2004), the parallel algorithm is compared with the serial (original) algorithm and shows better result from accuracy of solution and computational cost points of view.

Veldhuizen *et al.* (2003) discuss parallel evolutionary multiobjective optimization. In Veldhuizen *et al.* (2003), master-slave model, island model, diffusion model and hybrid model are discussed and the calculation time of them are also compared.

Dubreuil *et al.* (2006) analyze the master-slave model for distributed evolutionary computation theoretically. This paper builds a theoretical framework for the master-slave model and validates the framework empirically based on the Distributed BEAGLE C++ framework. They conclude that contrary to popular belief, the master-slave model can scale well.

Recently, many applications which need time-consuming fitness evaluations are successfully optimized using the master-slave model. Due to the page limitation, few of them are introduced, e.g., Jones *et al.* (1998); Sasaki *et al.* (2000); Okabe *et al.* (2003). Jones *et al.* (1998) parallelize a genetic algorithm on an aerodynamic and aeroacoustic optimization of airfoils. Despite the time-consuming multidisciplinary fitness evaluations, they successfully show good results by the usage of master-slave parallelization. Since their fitness evaluations need huge computational cost, their efficiency of parallelization achieves nearly 100%. Sasaki *et al.* (2000) optimize the design of a wing for supersonic transport using multiobjective genetic algorithm. To solve the huge computational cost, a simple master-slave model is used. They obtain the successful results with better performance. Okabe *et al.* (2003) optimize the shape of a micro heat exchanger problem using a commercial computational fluid dynamics software. To reduce huge computational cost, the algorithm is parallelized based on the master-slave model and successfully optimizes the shape. In Okabe *et al.* (2003), the necessary conditions of parallel optimization using a commercial solver are also discussed.

As introduced above, there are a lot of papers proposing new efficient method for the master-slave model and showing successful optimization results by master-slave model. Since real multiobjective optimization problems are more complicated, this type of parallelization will gather much attention in order to successfully obtain the optimal design of applications in reasonable time.

13.2.3 Level 3: Problem Dependent Parallelization

In this model, problem-dependent operations are parallelized. In general, the interest here is the parallelization of the evaluation of a single solution (different objectives and/or constraints). The parallel models may be based on the data partitioning or task partitioning. This model is useful in MOPs with time and/or memory intensive objectives and constraints. It may also be useful in MOPs with uncertainty which need in general an repeated evaluation of objective.

Basic Concept

In the last section, the evaluations in a generation are parallelized. However, even if the evaluations are parallelized, one fitness evaluation is sometimes still time-consuming. To solve this problem, we discuss the parallelization of each evaluation in this section. Possible parallelization of one fitness evaluations are listed as follows:

1. **Several solvers:**
   Consider a multiobjective optimization of a car design as an example. To design a car, several disciplines should be considered. Examples are to optimize the air flow around a car and the toughness of materials of a car. To optimize this problem, two independent solvers are necessary, i.e. CFD solver for the air flow and FEM for the toughness of materials. If we use one computer to evaluate two objectives, many users will firstly use the CFD to obtain the first objective function and secondly use the FEM to obtain the second objective function or vise versa. Some users will execute the CFD and the FEM at the same time. However, the total calculation time is nearly same with the above case because the computational resources are shared by two solvers. However, it is reasonable to execute the CFD and the FEM at the same time on different computers. Although the idle time, caused by the different calculation time of the CFD and the FEM, is not avoidable, the total calculation time becomes shorter.
2. **Decomposition of one fitness evaluation:**
   Consider the evaluation of a big product which consists of several parts. A simple idea to reduce the computational time is evaluation of each part and merging of them. Generally, CFD calculation for a big product is terribly time-consuming and has huge memory consumption. To tackle these problems, domain decomposition method (DDM) is often used in CFD research field, e.g., Elleighy and Tanaka (2001). Calculation domain is divided into several parts and assigned to different computers. Each computer calculates only the assigned part. To balance all calculation, the boundary condition is shared regularly. This division reduces the calculation time and memory consumption. However, since the boundary condition is shared regularly, rich connections among several computers are necessary. Furthermore, the user should take care of the division to reduce boundary and the balance of calculation cost on each computer.
3. **Multiple runs for one fitness evaluation:**
   Fitness evaluation sometimes needs several runs of a solver with different calculation conditions. An example is an optimization with uncertainty. Recently, robustness of fitness value against the variance of design parameters has gathered much attention, in particular, by researchers and

practitioners researching for a real application. In a real application, it is impossible to generate a product based on optimal design parameters because some variations are unavoidable. Therefore, it is very important to obtain a robust and (nearly) optimal design. To find robust and (nearly) optimal design, multiple runs of a solver are sometimes necessary. Assume that an optimizer obtains the design parameter x. To see the robustness against variance of the design parameter, the fitness values of $x + dx$ and $x - dx$ should also be evaluated. In this situation, it is reasonable to execute three solvers with different design parameters on different computers simultaneously. By simultaneous execution, calculation time will be reduced.

Calculation Time

For above situations, the total calculation time is considered here. Based on equations shown later, we will discuss when we should parallelize a calculation or not.

1. **Several solvers:**
Assume a k-objective optimization problem where the time t_i is necessary to evaluate the ith objective function and N nodes are available for calculation. As an example, $N = k$ is assumed. The total calculation time without/with parallelization can be obtained as:

$$t^{wo} = \sum_{i=1}^{k} t_i = \sum_{i=1}^{N} t_i, \tag{13.7}$$

$$t^{w} = \max(t_i) + T_{DT}, \tag{13.8}$$

here, t^{wo}, t^{w}, and T_{DT} are the necessary time for k objective evaluations without/with parallelization, and the necessary time for data transfer, see Fig. 13.2 (a). The maximum time of all t_i is denoted by $\max(t_i)$. Using these equations, the efficiency of parallelization, denote η', can be obtained as:

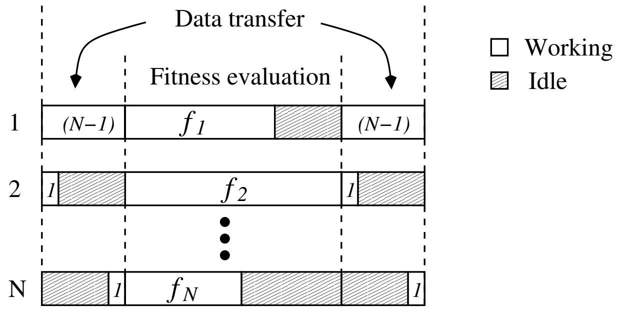
$$\eta' = \frac{t^{wo}}{Nt^{w}} \times 100 = \frac{\sum_{i=1}^{N} t_i}{N\max(t_i) + NT_{DT}} \times 100. \tag{13.9}$$

This equation leads the following results when T_{DT} is negligible:
- If all t_i are the same, the efficiency of parallelization is 100%.
- Otherwise, the efficiency is reduced due to idle time of the computer with a shorter calculation.

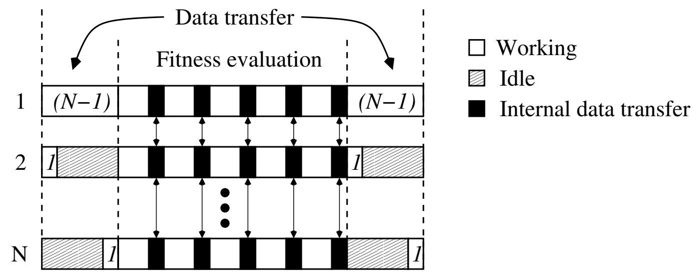
If T_{DT} is not negligible, the efficiency will be reduced. In the worst case, the efficiency is nearly 0% when $T_{DT} >> t_i$. This means that the parallelization should not be used.

2. **Decomposition of one fitness evaluation:**
Assume that N nodes are available for calculation and one problem will be decomposed into N sub-problems. The total calculation time of one problem and one sub-problem are assumed to be t_{all} and t_{sub}, respectively. Here, the decomposition is assumed as ideal, i.e., the time for all sub-problems is the same. The number of boundaries caused by decomposition and the time of internal data transfer per one boundary are assumed as B and T_{in}. By decomposition, each domain will be solved separately. However, to consider relations among neighbor domains, the boundary information should be adjusted regularly. One can obtain the following equations:
$$t^{wo} = t_{all} \tag{13.10}$$
$$t^{w} = t_{sub} + (N-1)T_{DT} + BT_{in}. \tag{13.11}$$
Here, the number of boundaries of each decomposed domain is assume to be the same with others. The variable of T_{DT} is the time of data transfer for initial data. Since t_{all} is approximately Nt_{sub}, one can obtain the following efficiency (see Fig. 13.2 (b)):
$$\eta' = \frac{t^{wo}}{Nt^{w}} \times 100 = \left(1 - \frac{(N-1)T_{DT} + BT_{in}}{(N-1)T_{DT} + BT_{in} + t_{sub}}\right) \times 100 \tag{13.12}$$
In domain decomposition method, T_{in} is generally very high. Therefore, by using rich connections among nodes, T_{in} should be reduced. Furthermore, the users should think of a way to reduce the number of boundaries, B.
3. **Multiple runs for one fitness evaluation:**
Following the same way of master-slave model, it is easy to obtain the following equation:
$$\eta' = \left(1 - \frac{(N-1)T_{DT}}{(N-1)T_{DT} + t}\right) \times 100, \tag{13.13}$$
here, N, T_{DT} and t are the number of necessary runs for one fitness value, the time for data transfer and the time for one fitness evaluation. As discussed in master-slave model, the calculation should be parallelized when $t >> T_{DT}$.

The three models for parallel metaheuristics may be used in conjunction within a hierarchical structure. In Meunier *et al.* (2000); Talbi and Meunier (2006), this hierarchical architecture has been adopted to solve a complex multiobjective network design problem. At level 1, a parallel self contained cooperative model based on evolutionary algorithms (island model) and local search has been used. At level 2, a parallel evaluation model for a steady state evolutionary algorithm, in which the evaluation phase of the algorithm is done in parallel and in an asynchronous manner. Those two first parallel model are independent of the target MOP. Finally at level 3, a parallel synchronous decomposition model, in which the evaluation of a single solution is carried out in parallel by partitioning the geographical domain.



(a) Several solvers



(b) Decomposition

Fig. 13.2. Necessary calculation time of several solvers and decomposition.

13.3 Parallel Models for Non-heuristic Methods

Parallelization of exact optimization methods, particularly branch and bound ones, has been largely studied in the literature (refer to Talbi (2006)). However, to the best of our knowledge it is rarely tackled in the multiobjective context. For example, in the 11th MCDM conference (1994) Antunes and Tsoukiás (1997) survey new developments in computer science and they try to explore their specific relevance for the field of MCDA. The field of *distributed computing* is mentioned (p. 382f.) with the potential to integrate different MC models and methods in a single MCDA system. The benefit of *parallel computing* is seen in decomposing the problems. But no reference to any existing work is given[1]. Although there were presentations of parallel approaches at MCDM conferences, these papers have been rarely included in the official proceedings. For example, in the 15th MCDM conference (2000)

[1] The potential benefits of fuzzy sets and neural networks have been discussed where evolutionary algorithms and related metaheuristics were not an issue at that time.

there were at least two papers on parallel MCDM methods[2] but none of them appeared in the official proceedings prepared by Köksalan and Zionts (2001). This might be the reason why there is only a short list of publications that is presented next.

13.3.1 High Level Parallel Models

High level parallel models embrace approaches in which sequential MCDM methods are run independently with no or occasional information exchange in parallel. The simplest example is probably given by the idea to run several instances of the same algorithm with scalarized objective functions but different weights. This approach yields an approximation of the Pareto front and set. More flexibility is provided by the OpTiX-II software framework described by Grauer and Boden (1997). Here, the user may specify which MCDM methods are run in parallel on workstation clusters, and which information they are going to exchange. Unfortunately, numerical results are given for single-objective optimization only.

13.3.2 Low Level Parallel Models

Low level parallel models represent approaches in which parts of the sequential MCDM method are parallelized. For example, in 1992 Galperin Galperin (1992) proposed a new unscalarized method for MCDM based on his concept of balance numbers. In this procedure m subproblems can be solved in parallel independently (p. 81) in each iteration. Apparently, the parallelization was outlined only but not realized.

In case of interrelated multiobjective linear (MOLP) problems Volkovich (1997) parallelize the search for local solutions. Again, there are no results regarding speedup or efficiency. The situation changes for the parallel method for MOLPs presented by Wiecek and Zhang (1997). They achieve a speedup about 27 when using 32 processors for large problems. They deploy the technique of task partitioning by using an ADBASE solver on each processor.

Another reason for using parallel hardware is raised with interactive MCDM methods: If you like to foster interactivity during robustness analysis and that the decision maker does not give up too early (due to impatience and/or time schedule) then you must take care about fast response times. For this purpose, Costa and Climaco (1994) calculated solutions in parallel when using multiple reference points on a four processor system. They extended their work on parallelization to other interactive MCDM methods: in the course of the ELECTRE III method the creditability indices that define a fuzzy outranking relation can be calculated independently (and hence in parallel) for different pairs of alternatives. Moreover, they also parallelized the four subproblems of the distillation algorithm used in this method. A speedup

[2] See the program at `http://mcdm2000.ie.metu.edu.tr/tentprog.htm`

about 7 was achieved for 16 processors by Dias *et al.* (1997). Similarly, the preference indices of the interactive PROMETHEE method can be calculated independently and therefore in parallel. Dias *et al.* (1998) achieve a speedup about 15 for 16 processors in the best case.

Low level parallel models are also used for exact, combinatorial multi-objective problems. In particular, in case of biobjective flowshop problems Dhaenens *et al.* (2006) discuss parallelization of the weighted sum method with dichotomic search: after a new solution is found two new searches are launched. Consequently, many processors are idle in the early phase of the search resulting in poor speedup/efficiency measures. A similar approach can be deployed for the two phase method—with same disadvantages as in the previous method proposed by Lemesre *et al.* (2007a). Speedups do not exceed 1.7 for four processors. Lemesre *et al.* (2007b) achieved similar performance for the partitioning parallel method. In the first phase they use task partitioning and then space partitioning.

Needless to say, hybridizations of metaheuristics and exact methods do have some potential in case of parallelization. Basseur *et al.* (2004) propose a parallel hybrid model combining an exact approach (branch and bound) and a metaheuristic. The parallel metaheuristic is used to approximate the Pareto front. The parallel branch and bound is used to solve sub-problems and to improve the quality of the obtained Pareto front.

We are aware that this list of publications is not complete. But it reveals that there is considerably less work on the parallelization of non-heuristic and exact methods than for metaheuristics. Finally, we like to emphasize that the deployment of parallel hardware for interactive environments might be fruitful also for multiobjective metaheuristics.

13.4 Parallel Implementation and Deployment

Parallel and distributed architectures can have different memories (shared/distributed), computation resources (homogeneous/heterogeneous), and networks (local/large). These different properties have a strong impact on the deployment technique such as load balancing and fault-tolerance. Depending on the type of architecture used, different parallel and distributed programming environments such as message passing (PVM, MPI), shared memory (multi-threading, OpenMP), high throughput computing (Condor), and Grid computing (Globus) can be used. We briefly study some of this issues in the following.

13.4.1 Shared Memory versus Distributed Memory

The main advantage of parallel MOP algorithms implemented on shared memory architectures such as SMPs and multi-core processors is the simplicity.

For example, it is easier to share data such as upper bounds in exact algorithms and best found approximated non-dominated set of solutions in meta-heuristics. However, parallel distributed architectures offer a more flexible and fault-tolerant programming platform. Indeed, the memory access contention in shared memory architecture make the number of processors limited for this type of architectures.

13.4.2 Homogeneous and Dedicated versus Heterogeneous and Non-dedicated

Most massively parallel machines (MPP) and clusters of workstations (COW) such as IBM SP3 are composed of homogeneous processors and are generally dedicated to the application. The proliferation of powerful workstations and fast communication networks have shown the emergence of heterogeneous network of workstations (NOW) as platforms for high-performance computing. COWs and NOWs constitute a low-cost hardware alternative to run parallel algorithms. However, the efficient scheduling of tasks and fault tolerant mechanisms in NOWs is more complex to design and analyze due to the heterogeneity of those architectures (processors, networks, etc.) and a higher probability of faults.

Melab *et al.* (2006a) focus on solving large size problems using the Condor environment. It is an open source framework originally intended to the design and deployment of the three parallel models for meta-heuristics on dedicated clusters and networks of workstations. Relying on the Condor programming environment, it enables the execution of these applications on volatile non dedicated heterogeneous computational pools of resources. Efficient load balancing and fault-tolerance mechanisms have been designed for this purpose. Experimentations have been carried out on more than 100 PCs originally intended for education. The obtained results are convincing, both in terms of flexibility and easiness at implementation, and in terms of efficiency, quality and robustness of the provided solutions at run time.

13.4.3 Tightly Coupled (Local Networks) versus Loosely Coupled (Large Networks)

Massively parallel machines, clusters and local networks of workstations may be considered as tightly coupled architectures. Large network of workstations and Grid computing platforms are loosely coupled and are affected by higher cost of communication. The larger the granularity of a model, the better suited is the model for large networks.

Since the granularity of the self-contained parallel cooperation models (level 1) is very high, they can be easily deployed on large scale architectures which are in general loosely coupled and have high communication cost.

This model is also scalable in terms of the number of processors which is not the case for the other models (Problem Independent Parallel Intra-Algorithm and Problem Dependant Parallelization). These models are interesting when the evaluation of a single solution is CPU-time consuming and/or Input/Output intensive.

Melab *et al.* (2006b) report some results on parallel cooperative multiobjective meta-heuristics on computational grids. They particularly focus on the island model and the multi-start model and their cooperation. They propose a checkpointing-based approach to deal with the fault tolerance issue of the island model. Nowadays, existing DispatcherWorker grid middlewares are inadequate for the deployment of parallel cooperative applications. Indeed, these need to be extended with a software layer to support the cooperation. Therefore, they propose a Linda-like cooperation model and its implementation on top of XtremWeb. This middleware is then used to develop a parallel meta-heuristic applied to a bi-objective Flow-Shop problem using the two models. The work has been experimented on a multi-domain education network of 321 heterogeneous Linux PCs. The preliminary results, obtained after more than 10 days, demonstrate that the use of grid computing allows to fully exploit effectively different parallel models and their combination for solving large-size problem instances.

In terms of exact methods, the high level is more appropriate for large networks. The most popular parallelization approach of the branch and bound algorithm consists in building and exploring in parallel the search tree representing the problem being tackled. The deployment of such parallel model on a grid raises the crucial issue of dynamic load balancing. The major question is how to efficiently distribute the nodes of an irregular search tree among a large set of heterogeneous and volatile processors. Mezmaz *et al.* (2007) propose a new dynamic load balancing approach for the parallel branch and bound algorithm on the computational grid. The approach is based on a particular encoding of the tree nodes allowing a very simple description of the work units distributed during the exploration. Such description optimizes the communications involved by the huge amount of load balancing operations. The approach has been applied to one instance of the bi-objective flow-shop scheduling problem. The application has been experimented on a computational pool of more than 1000 processors belonging to seven Nation-wide clusters. The optimal Pareto front has been generated within almost 6 days with a parallel efficiency of 98%.

13.5 Conclusion and Future Trend

Parallel and distributed computing are powerful and necessary ways to reduce the computation time of multiobjective optimization algorithms and/or improve the quality of the obtained solutions. This chapter presents a general overview of parallel approaches for multiobjective optimization. For this purpose, we have proposed a taxonomy for parallel metaheuristics and exact methods. We have covered both the design aspect of algorithms and implementation on different parallel and distributed architectures. Different parallel models have been proposed in the design of multiobjective optimization algorithms. These models are largely experimented on a wide range of academic and real-life MOPs in different domains. The presented models may be used in conjunction within a hierarchical structure.

Multiobjective optimization algorithms have been implemented and deployed on different type of parallel and distributed architectures: clusters and networks of workstations and shared memory parallel architectures. An efficient implementation must consider the characteristics of the target parallel model (granularity, synchronous, etc.) and architecture (homogeneity, dedicated, etc.). For example, fine granularity models cannot easily be deployed on large scale distributed systems.

In the last decade, Grid computing and Peer-to-Peer (P2P) computing have become a real alternative to traditional high performance computing architectures for the development of large-scale distributed applications. This is a great challenge as Grid and P2P-enabled frameworks for multiobjective optimization algorithms are emerging.

Designing generic software frameworks to deal with the design and efficient transparent implementation of distributed multiobjective optimization algorithms is another important aspect. Software frameworks such as PARADISEO offer transparent implementation of different parallel models on different architectures using suitable programming environments as written by Cahon *et al.* (2004) and Liefooghe *et al.* (2007)[3].

In future, more and more applications will be concerned by parallel multiobjective optimization in different domains such as MDO (Multi-disciplinary Design Optimization), life sciences and industrial applications. Also, designing the interactive multiobjective optimization approaches which requires real-time parallel solving of MOPs is another important challenge.

References

Adamidis, P.: Review of Parallel Genetic Algorithms Bibliography. Technical Report, Aristotle University of Thessaloniki (1994)

Antunes, C., Tsoukiás, A.: Against fashion: A travel survival kit in "modern" MCDA. In: Multicriteria Analysis:International Conference on Multiple Criteria Decision Making, pp. 378–389. Springer, Berlin (1997)

Baita, F., Mason, F., Poloni, C., Ukovich, W.: Genetic Algorithm with Redundancies for the Vehicle Scheduling Problem. In: Biethahn, J., Nissen, V. (eds.) Evolutionary Algorithms in Management Applications, pp. 341–353. Springer, Berlin (1995)

[3] See the web site: `http://paradiseo.gforge.fr` for more details.

Basseur, M., Lemesre, J., Dhaenens, C., Talbi, E.-G.: Cooperation between branch and bound and evolutionary approaches to solve a bi-objective flow shop problem. In: Ribeiro, C.C., Martins, S.L. (eds.) WEA 2004. LNCS, vol. 3059, pp. 72–86. Springer, Heidelberg (2004)

Bethke, A.D.: Comparison of Genetic Algorithms and Gradient-based Optimizers on Parallel Processors: Efficiency of Use of Processing Capacity. Logic of Computers Group Technical Report 197, University of Michigan (1976)

Branke, J., Schmeck, H., Deb, K., Reddy, M.: Parallelizing Multi-Objective Evolutionary Algorithms: Cone Separation. In: IEEE Congress on Evolutionary Computation, pp. 1952–1957 (2004)

Bui, L.T., Abbass, H.A., Essam, D.: Local models - an approach to distributed multiobjective optimization. Technical Report TR-ALAR-200601002, The Artificial Life and Adaptive Robotics Laboratory, University of New South Wales, Australia (2006)

Cahon, S., Melab, N., Talbi, E.-G.: ParadisEO: A framework for the reusable design of parallel and distributed metaheuristics. Journal of Heuristics 10(3), 357–380 (2004)

Cantu-Paz, E.: A Survey of Parallel Genetic Algorithms. IlliGAL Report 97003, University of Illinois (1997a)

Cantu-Paz, E.: Designing Efficient Master-slave Parallel Genetic Algorithms. IlliGAL Report 97004, University of Illinois (1997b)

Coello Coello, C.A., Reyes Sierra, M.: A study of the parallelization of a coevolutionary multi-objective evolutionary algorithm. In: Monroy, R., Arroyo-Figueroa, G., Sucar, L.E., Sossa, H. (eds.) MICAI 2004. LNCS (LNAI), vol. 2972, pp. 688–697. Springer, Heidelberg (2004)

Coello Coello, C.A., Van Veldhuizen, D.A., Lamont, G.B.: Evolutionary Algorithms for Solving Multi-Objective Problems. Kluwer Academic Publishers, New York (2002)

Costa, J.P., Climaco, J.N.: A multiple reference point parallel approach in MCDM. In: International Conference on Multiple Criteria Decision Making, pp. 255–263. Springer, New York (1994)

de Toro Negro, F., Ortega, J., Fernandez, J., Diaz, A.: PSFGA: a parallel genetic algorithm for multiobjective optimization. In: Euromicro Workshop on Parallel, Distributed and Network-based Processing, pp. 384–391 (2002)

de Toro Negro, F., Ortega, J., Ros, E., Mota, S., Paechter, B., Martín, J.M.: PSFGA: Parallel Processing and Evolutionary Computation for Multiobjective Optimisation. Parallel Computing 30(5–6), 721–739 (2004)

Deb, K., Zope, P., Jain, S.: Distributed computing of Pareto-optimal solutions with evolutionary algorithms. In: Fonseca, C.M., Fleming, P.J., Zitzler, E., Deb, K., Thiele, L. (eds.) EMO 2003. LNCS, vol. 2632, pp. 534–549. Springer, Heidelberg (2003)

Dhaenens, C., Lemesre, J., Melab, N., Mezmaz, M., Talbi, E.-G.: Parallel exact methods for multi-objective combinatorial optimization. In: Parallel Combinatorial Optimization, John Wiley and Sons, Berlin (2006)

Dias, L.C., Costa, J.P., Climaco, J.N.: Conflicting criteria, cooperating processors—some experiments on implementing a multicriteria support method on a parallel computer. Computers and Operations Research 24(9), 805–817 (1997)

Dias, L.C., Costa, J.P., Climaco, J.N.: A parallel implementation of the PROMETHEE method. European Journal of Operational Research 104(3), 521–531 (1998)

Dubreuil, M., Gagne, C., Parizeau, M.: Analysis of a Master-slave Architecture for Distributed Evolutionary Computations. IEEE Transactions on Systems, Man, and Cybernetics 36(1), 229–235 (2006)

Elleighy, W.M., Tanaka, M.: Domain Decomposition Coupling of FEM and BEM. Transactions of the Japan Society for Computational Engineering and Science 4, 107–111 (2001)

Galperin, E.A.: Nonscalarized multiobjective global optimization. Journal of Optimization Theory and Applications 75(1), 69–85 (1992)

Grauer, M., Boden, H.: OpTiX-II: A software environment for MCDM based on distributed and parallel computing. In: Multicriteria Analysis: International Conference on Multiple Criteria Decision Making, pp. 199–208. Springer, Berlin (1997)

Hiroyasu, T., Miki, M., Watanabe, S.: The New Model of Parallel Genetic Algorithm in Multi-Objective Optimization Problems—Divided Range Multi-Objective Genetic Algorithm—. In: IEEE Congress on Evolutionary Computation, July 2000, vol. 1, pp. 333–340. IEEE Computer Society Press, Piscataway (2000)

Jones, B.R., Crossley, W.A., Lyrintzis, A.S.: Aerodynamic and Aeroacoustic Optimization of Airfoils via a Parallel Genetic Algorithm. In: AIAA 98-4811 (1998)

Jozefowiez, N., Semet, F., Talbi, E.-G.: Parallel and hybrid models for multi-objective optimization: Application to the vehicle routing problem. In: Guervós, J.J.M., Adamidis, P.A., Beyer, H.-G., Fernández-Villacañas, J.-L., Schwefel, H.-P. (eds.) PPSN 2002. LNCS, vol. 2439, pp. 271–280. Springer, Heidelberg (2002)

Jozefowiez, N., Semet, F., Talbi, E.-G.: Enhancements of nsga ii and its application to the vehicle routing problem with route balancing. In: Talbi, E.-G., Liardet, P., Collet, P., Lutton, E., Schoenauer, M. (eds.) EA 2005. LNCS, vol. 3871, pp. 131–142. Springer, Heidelberg (2006)

Jozefowiez, N., Semet, F., Talbi, E.-G.: Target aiming pareto search and its application to the vehicle routing problem with route balancing. Journal of Heuristics 13(5), 455–469 (2007)

Köksalan, M., Zionts, S.: International Conference on Multiple Criteria Decision Making. Springer, Berlin (2001)

Lemesre, J., Dhaenens, C., Talbi, E.-G.: An exact parallel method for a bi-objective permutation flowshop problem. European Journal of Operational Research 177(3), 1641–1655 (2007a)

Lemesre, J., Dhaenens, C., Talbi, E.-G.: Parallel partitioning method (PPM): A new exact method to solve bi-objective problems. Computers and Operations Research 34(8), 2450–2462 (2007b)

Liefooghe, A., Jourdan, L., Talbi, E.-G.: Paradiseo-MOEO: A framework for evolutionary multi-objective optimization. In: Evolutionary Multi-objective Optimization, Japan, pp. 457–471 (2007)

López-Jaimes, A., Coello Coello, C.A.: MRMOGA: Parallel Evolutionary Multiobjective Optimization using Multiple Resolutions. In: IEEE Congress on Evolutionary Computation, Edinburgh, Scotland, September 2005, vol. 3, pp. 2294–2301. IEEE Computer Society Press, Los Alamitos (2005)

Mehnen, J., Michelitsch, T., Schmitt, K., Kohlen, T.: pMOHypEA: Parallel evolutionary multiobjective optimization using hypergraphs. Technical Report of the SFB Project 531 Computational Intelligence CI–189/04, University of Dortmund (2004)

Melab, N., Cahon, S., Talbi, E.-G.: Grid computing for parallel bioinspired algorithms. Journal of Parallel and Distributed Computing (JPDC) 66(8), 1052–1061 (2006a)

Melab, N., Mezmaz, M., Talbi, E.-G.: Parallel cooperative metaheuristics on the computational grid: A case study - the biobjective flow-shop problem. Parallel computing 32(9), 643–659 (2006b)

Meunier, H., Talbi, E.-G., Reininger, P.: A multiobjective genetic algorithm for radio network design. In: IEEE Congress on Evolutionary Computation, Orlando, USA, pp. 317–324 (2000)

Mezmaz, M., Melab, N., Talbi, E.-G.: Using the multi-start and island models for parallel multi-objective optimization on the computational grid. In: IEEE International Conference on e-Science and Grid Computing (e-Science'06), pp. 112–120 (2006)

Mezmaz, M., Melab, N., Talbi, E.-G.: An efficient load balancing strategy for grid-based branch and bound. Parallel computing 33(4-5), 302–313 (2007)

Mostaghim, S., Branke, J., Schmeck, H.: Multi-objective particle swarm optimization on computer grids. In: The Genetic and Evolutionary Computation Conference, vol. 1, pp. 869–875 (2007)

Okabe, T.: Evolutionary Multi-objective Optimization -On the Distribution of Offspring in Parameter and Fitness Space-. Shaker Verlag, Aachen (2004)

Okabe, T., Foli, K., Olhofer, M., Jin, Y., Sendhoff, B.: Comparative Studies on Micro Heat Exchanger Optimization. In: IEEE Congress on Evolutionary Computation, pp. 647–654 (2003)

Parmee, I.C., Vekeria, H.D.: Co-operative Evolutionary Strategies for Single Component Design. In: Bäck, T. (ed.) International Conference on Genetic Algorithms, pp. 529–536. Morgan Kaufmann, San Francisco (1997)

Poloni, C.: Hybrid GA for Multi-Objective Aerodynamic Shape Optimization. In: Winter, G., Periaux, J., Galan, M., Cuesta, P. (eds.) Genetic Algorithms in Engineering and Computer Science, pp. 397–416. Wiley & Sons, Chichester (1995)

Sasaki, D., Obayashi, S., Sawada, K., Himeno, R.: Multiobjective Aerodynamic Optimization of Supersonic Wings Using Navier-Stokes Equations. In: European Congress on Computational Methods in Applied Sciences and Engineering (2000)

Schaffer, D.J.: Multiple objective optimization with vector evaluated genetic algorithms. In: International Conference on Genetic Algorithms and Their Applications, pp. 93–100 (1985)

Schmeck, H., Kohlmorgen, U., Branke, J.: Parallel Implementations of Evolutionary Algorithms. In: Solutions to Parallel and Distributed Computing Problems, pp. 47–68 (2001)

Stanley, T.J., Mudge, T.: A Parallel Genetic Algorithm for Multiobjective Microprocessor Design. In: The Sixth International Conference on Genetic Algorithms, pp. 597–604 (1995)

Streichert, F., Ulmer, H., Zell, A.: Parallelization of multi-objective evolutionary algorithms using clustering algorithms. In: Coello Coello, C.A., Hernández Aguirre, A., Zitzler, E. (eds.) EMO 2005. LNCS, vol. 3410, pp. 92–107. Springer, Heidelberg (2005)

Talbi, E.-G.: Parallel combinatorial optimization. Wiley, Chichester (2006)

Talbi, E.-G., Meunier, H.: Hierarchical parallel approach for gsm mobile network design. Journal of Parallel and Distributed Computing 66(2), 274–290 (2006)

Van Veldhuizen, D.A., Zydallis, J.B., Lamont, G.B.: Considerations in Engineering Parallel Multiobjective Evolutionary Algorithms. IEEE Transactions on Evolutionary Computation 7(2), 144–173 (2003)

Volkovich, V.L.: Distributed multiobjective optimization problems and methods for their solution. In: International Conference on Multiple Criteria Decision Making, pp. 222–232. Springer, Berlin (1997)

Watanabe, S., Hiroyasu, T., Miki, M.: Parallel Evolutionary Multi-Criterion Optimization for Mobile Telecommunication Networks Optimization. In: Evolutionary Methods for Design, Optimization and Control, pp. 162–172 (2002)

Wiecek, M.M., Zhang, H.: A parallel algorithm for multiple objective linear programs. Computational Optimization and Applications 8(1), 41–56 (1997)

Xiao, N., Armstrong, M.P.: A specialized island model and its application in multiobjective optimization. In: Cantú-Paz, E., Foster, J.A., Deb, K., Davis, L., Roy, R., O'Reilly, U.-M., Beyer, H.-G., Kendall, G., Wilson, S.W., Harman, M., Wegener, J., Dasgupta, D., Potter, M.A., Schultz, A., Dowsland, K.A., Jonoska, N., Miller, J., Standish, R.K. (eds.) GECCO 2003. LNCS, vol. 2724, pp. 1530–1540. Springer, Heidelberg (2003)

Zhu, Z.-Y.: An Evolutionary Approach to Multi-Objective Optimization Problems. Ph.D. thesis, The Chinese University of Hong Kong (2002)

Zhu, Z.-Y., Leung, K.-S.: Asynchronous Self-Adjustable Island Genetic Algorithm for Multi-Objective Optimization Problems. In: IEEE Congress on Evolutionary Computation, Piscataway, New Jersey, May 2002, vol. 1, pp. 837–842 (2002)