

Parallel Multi-objective Optimization using Master-Slave Model on Heterogeneous Resources

Sanaz Mostaghim, Jürgen Branke, Andrew Lewis, Hartmut Schmeck

Abstract—In this paper, we study parallelization of multi-objective optimization algorithms on a set of heterogeneous resources based on the Master-Slave model. The Master-Slave model is known to be the simplest parallelization paradigm, where a master processor sends function evaluations to several slave processors. The critical issue when using the standard methods on heterogeneous resources is that in every iteration of the optimization, the master processor has to wait for all of the computing resources (including the slow ones) to deliver the evaluations. In this paper, we study a new algorithm where all of the available computing resources are efficiently utilized to perform the multi-objective optimization task independent of the speed (fast or slow) of the computing processors. For this we propose a hybrid method using Multi-objective Particle Swarm optimization and Binary search methods. The new algorithm has been tested on a scenario containing heterogeneous resources and the results show that not only does the new algorithm perform well for parallel resources, but also when compared to a normal serial run on one computer.

I. INTRODUCTION

Stochastic iterative search methods such as Evolutionary Algorithms (EAs) and Particle Swarm Optimization (PSO) have been shown to solve many real-world applications for decades [16], [18]. These algorithms are naturally parallel; they contain several individuals which are being improved through generations. This parallel nature is particularly useful when implementing the algorithm on parallel computers. Many real-world applications have time consuming function evaluations and therefore parallelizing the evaluations on a set of available computing resources speeds up the optimization task. Also, parallelization is getting easier with the steady progress of new technologies such as Grid Computing [?]. Several Parallel Evolutionary Algorithms have been studied in the literature (e.g., in [23], [10], [7], [8]). There are three, main paradigms: the Island model, Master-Slave model and Diffusion model. The focus of this paper is the Master-Slave model, where a single processor maintains the optimization task and uses the other processors for objective function evaluations.

In this paper, we study parallelization of multi-objective optimization (MO) algorithms using the master-slave model on a heterogeneous set of processors. There is a major difference when parallelizing a multi-objective algorithm compared to a single objective one. The main difficulty is that when using a Multi-objective EA (MOEA) on a master-slave model, the master processor has to wait for all of the

evaluations from the other processors. Then it can apply a ranking method to the solutions to find the non-dominated front and continue the optimization. In an heterogeneous environment, the waiting time might be very long. In some cases, the fast processors can deliver twice the evaluations of a slow one. Here, we study a new algorithm which utilizes all of the computing resources from the slow to the very fast (as in a normal Grid). There are several Parallel MOEA methods which study different aspects [14], [4], [?] particularly the island model. The Master-Slave model is a straight-forward method and is the simplest parallelization paradigm when working in an homogeneous environment. However, to our knowledge no one has studied solving Multi-objective problems on heterogeneous systems using the Master-Slave model. Solving Multi-Objective problems on such systems is a challenging task:

- Dealing with heterogeneous resources, the aim is to use all of the computing resources but at the same time to be efficient in time i.e., the master node must make use of all of the available functions evaluations in every time step. Indeed, the master node encounters a trade-off between waiting for the slowest processor or continuing the optimization based on the available solutions.
- The solution of multi-objective optimization problems is usually a set of so called Pareto-optimal solutions. Most of the Multi-objective evolutionary algorithms (MOEAs) approximate these solutions by finding a set of non-dominated solutions. In every generation, a typical MOEA method applies a ranking-based method to the “entire” population to evaluate the solutions.
- Exploration in the search space is another important issue. The Master-Slave model is typically used to solve computationally expensive problems. The main task would be to employ a reasonable exploration technique.

One drawback of the Master-Slave model is the communication overhead between the processors. Here, we ignore this as we aim to solve very expensive optimization problems.

In this paper, we study the Multi-Objective Particle Swarm Optimization (MOPSO) method. In MOPSO, every single particle (Individual) follows its own best position and the position of a global guide. Hence, in the parallelization scheme, it is enough to find a guide for a particular particle to improve the particle’s position. This is an advantage comparing to other MOEA techniques such as SPEA2 [24] or NSGAI [12]. In SPEA2, all of the individuals are compared to each other and are given a strength and in NSGAI a ranking method finds different non-dominated fronts. In

Sanaz Mostaghim, Jürgen Branke and Hartmut Schmeck are with the Institute AIFB, University of Karlsruhe, Germany, (email: (mostaghim, branke, schmeck)@aifb.uni-karlsruhe.de). Andrew Lewis is with the School of Information and Communication Technology, Griffith University, Australia, email: a.lewis@griffith.edu.au.

these methods, the best evaluation is based on the entire population.

This paper is organized as follows. We briefly explain the Master-Slave model of parallelization in the next section. In Section III the parallel MOPSO for the Master-Slave model in homogeneous and heterogeneous environments is studied. Section IV is dedicated to the experiments which are based on a scenario of a set of heterogeneous resources. Finally, Section V concludes the paper.

II. MASTER-SLAVE MODEL OF PARALLELIZATION

The simplest parallelization paradigm for optimization is the Master-Slave model. This model is aimed at distributing the (objective function) evaluation of the individuals on several slave computing resources while a master resource executes the optimization procedure. The master node can also be used for evaluations. Figure 1 shows this paradigm.

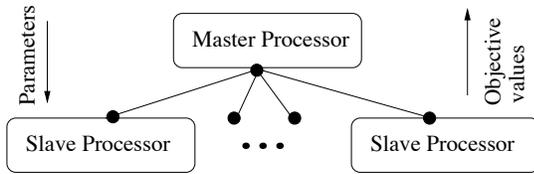


Fig. 1. Master-Slave Model [10]

This model is of great benefit when the function evaluations are very expensive. It is very natural to use parallel computers to solve expensive functions. However, the key issue is that in most of cases we deal with heterogeneous resources. Here, the central computing resource has to gather the information from all of the resources and has to perform the optimization steps (such as ranking etc.) based on the obtained solutions. However, waiting for the slowest processor might take a long time. The main questions when solving problems on heterogeneous computing resources are (a) how to efficiently search the space and (b) how to use all of the resources so that none of them stops working.

III. PARALLEL MULTI-OBJECTIVE PARTICLE SWARM

The main advantage of MOPSO compared to MOEAs is that the solutions survive until the last generation. A typical MOPSO method is shown in Algorithm 1. It starts with a random population of solutions (also called particles). Every particle i has a position in the search space and a velocity, denoted by x^i and v^i , respectively. The non-dominated particles are stored in an archive and the population is updated based on the positions of the global best particles P_{global} and their own personal memories P_{Best} :

$$x_{new}^i = x_{old}^i + v_{new}^i \quad (1)$$

$$v_{new}^i = wv_{old}^i + R_1(x_{old}^i - P_{Best}) + R_2(x_{old}^i - P_{global}) \quad (2)$$

where w , R_1 and R_2 denote the inertia weight and control parameters (1 and 2). In MOPSO, selecting the best personal position and the global positions has a great impact on the quality of the solutions [3], [21]. The global best is

Algorithm 1 MOPSO

Initiate population

repeat

 Find non dominated solutions; store in the archive

 Update the population and the archive

 Apply turbulence Factor

until Termination condition met

Return archive

Algorithm 2 Synchronous MOPSO

Initiate population

repeat

 Distribute evaluations of the population members on the computing resources

 Wait for all of the computing resources to finish evaluations

 Find non dominated solutions; store in the archive

 Update the population and the archive

 Apply turbulence factor

until Termination condition met

Return archive

usually selected from the archive. To avoid local optima in MOPSO, a percentage of the particles are randomly selected and moved in order to explore the search space. This is implemented using a turbulence factor. MOPSO is iteratively continued until a stopping criteria, such as a certain number of iterations or evaluations, is met. A very good survey about MOPSO can be found in [22].

Parallelizing MOPSO based on the Master-Slave model for a set of homogeneous resources can be easily integrated. Algorithm 2 shows a Parallel MOPSO on an homogeneous (synchronous) set of resources. In every iteration, the master resource distributes the evaluation tasks and waits for all of the resources to finish.

However, the key issue in dealing with heterogeneous (asynchronous) resources is that in the worst case of having only one slow computing resource, almost all of the computing resources remain idle while the master node waits for the slowest one. The main idea of this paper is to make use of all of the computing resources in a way that as soon as a reasonable amount of resources have completed evaluations the master resource continues optimizing. However, the results of the slow computers are also of interest as they also contain information about the search space. The first step in parallelization on an heterogeneous set of resources is to define a good set of initial particles and send them to the slave nodes for evaluation. In MOPSO usually a random set is used; here we propose to use a set of random particles defined by a Binary Search (BS) Method as explained below.

A. Exploration using Binary Search method

For searching the most unexplored regions of a search space, Binary Search Algorithms have been proposed in [17]. The search is started by selecting a random point in the

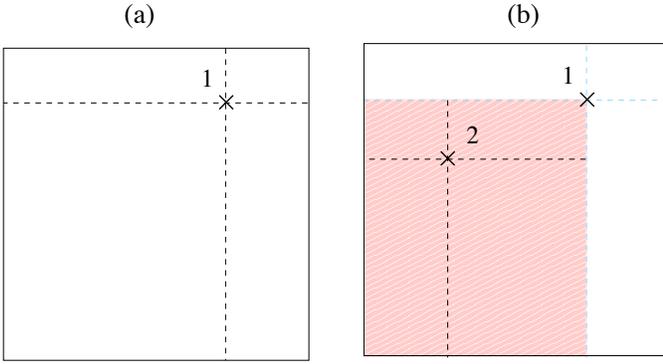


Fig. 2. An example of a Binary Search Method in a two dimensional space. In (a) one point is randomly selected in the search space. The next point is selected in the most empty region defined in (b).

search space. Then the most empty region is found in the space by computing the distances between the first point and the boundaries of the search space. Figure 2 (a) shows an example for a two dimensional space. The next point is randomly selected in that empty region (Figure 2 (b)). This is an iterative method and can be performed for any desirable number of solutions. These solutions are usually stored in a list.

Beside the Binary Search method, Voronoi diagrams can be used for exploration and exploitation in the search space [17], but Binary Search methods are very simple to implement. However, one disadvantage is that selecting a point from a large list of solutions requires a relative high computation time particularly for high (> 20) dimensional spaces. In the cases at which this study is aimed, it is assumed the time taken for objective function evaluations will dominate.

B. Parallel MOPSO on Heterogeneous resources

We propose a MOPSO which starts with an initial set of solutions based on the Binary Search (BS) method. These selected solutions are stored in a list and are sent to the slave processors for evaluation. The master node waits for N_s ($1 \leq N_s \leq N$) processors where N is the total number of available processors. After receiving those evaluations, the non-dominated solutions are stored in the archive and the master node sends new evaluation jobs to the N_s processors. Before sending the parameter sets, the master node qualifies the evaluation of the processors. If the solution found by a processor is:

- dominated by one of the archive members, the master processor selects a proper global guide from the archive and updates the position of the corresponding particle based on MOPSO. Then the master processor sends the new position to the processor for the evaluations.
- not dominated by any other archive member, the master processor gives the task of exploration to the corresponding processor. Based on the Binary Search Method, a proper random position in an unexplored region in the space is selected and is sent it to the

Algorithm 3 Asynchronous MOPSO (BS stands for the Binary Search method)

```

Initiate population using BS
Store the BS points in a List
Distribute evaluations of the population members on the
computing resources
repeat
  Wait for at least  $N_s$  of the computing resources to
  finish evaluations
  Find non dominated solutions; store in the archive
  For all of the dominated solutions
    Update using MOPSO
  For all of the non-dominated solutions
    Explore using BS
    Update the BS List
  Update the archive
  Redistribute the evaluations on the resources
until Termination condition met
Return archive

```

processor for evaluation.

Indeed, the exploration task done by the Binary Search method replaces the turbulence factor of the original MOPSO. The population members are stored in the Binary List over the generations. Algorithm 3 shows the parallel MOPSO on heterogeneous resources.

C. Discussion

In this algorithm, the parameter N_s , the least number of processors to wait for, must be known in advance. In fact, this parameter depends on the heterogeneous resources and the estimated load on them.

In our proposed Parallel MOPSO, we do not use a turbulence factor. Instead, we apply an intentional exploration to the non-dominated solutions using the Binary Search method. The reason is that the non-dominated solutions cannot be considerably improved by MOPSO as they build the global best particles. Therefore, their positions are stored in the archive and we assign them a new position based on the Binary Search method. On the other hand, the position of the dominated particles can be improved by the non-dominated ones and therefore the MOPSO method can be used effectively.

In the proposed asynchronous MOPSO algorithm, we not only store the non-dominated solutions in the archive, but we also store all the population members (explored by the Binary Search method) in a list. In fact, the list is very useful to obtain knowledge about the unexplored regions, in contrast to the idea of selecting any random position in parameter space using the turbulence factor. It must be mentioned that the size of the list has a great impact on the computation time. Here, we assume that the function evaluations require a relatively high computation time and that there is a limited number of solutions in the lists so that the computation time of finding the unexplored regions in the list can be neglected.

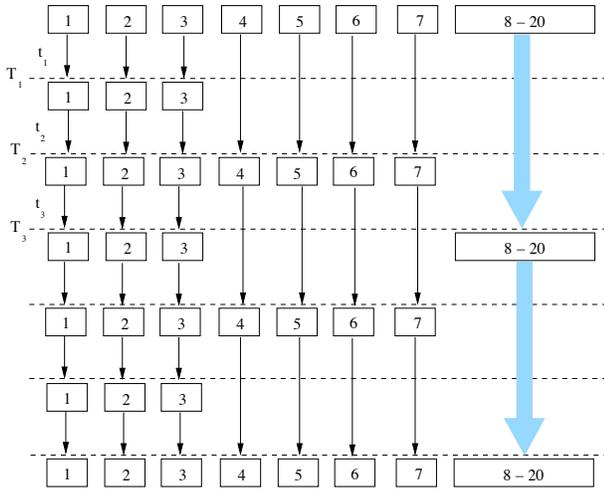


Fig. 3. 20 heterogeneous processors are illustrated. t_1 , t_2 and t_3 are equal. Here, a total time of $6T_1$ is shown where, after this time, all processors are synchronized again.

IV. EXPERIMENTS

In order to simulate the heterogeneous environment, we consider 20 processors as shown in Figure 3. Three processors (1-3) are fast and can finish the evaluations of the objective values in time T_1 , four processors (4-7) are slower and require time $T_2 = 2T_1$ for this task. The rest of the processors (8-20) are even slower, requiring $T_3 = 3T_1$.

In Figure 3, a total time of $6T_1$ is depicted. After this time, all processors are synchronized again,; therefore we call this a **cycle**. We run our tests for 20 such cycles (i.e. the slowest processors can process 40 evaluations in this time, the fastest processors can run 120 evaluations). For simplicity, the proposed asynchronous MOPSO method is called PMOPSO in the rest of the paper. For a more in-depth evaluation, we additionally compare the results of our new algorithm to a number of straightforward alternatives:

a) *Case1*: Instead of running a parallel MOPSO, a simple alternative would be just to run a single MOPSO on the fastest processor available for the same time. While this scenario uses less computational power overall, it will tell us how much we can benefit from parallelization.

b) *Case2*: We could also let a single processor run for as many evaluations as all processors in our heterogeneous example together. In the above scenario we span 56 evaluations per cycle. This leads to a population of 20 particles for 56 generations. This uses the same computational power as our heterogeneous MOPSO, but takes much more time because it is not run in parallel. In fact, this case should perform “better” than other cases as the computation time is not being considered.

c) *Case 3*: We define an homogeneous environment which uses 20 processors with the same speed as the slowest processor from the heterogeneous environment. This case is based on the Algorithm 2 and contains a population of 20 particles which are run for 20 cycles (40 generations). We refer to this case as MOPSO in the following.

For all of the MOPSO methods, we set the inertia weight value and turbulence factor at 0.4 and 0.1, respectively. The global best particles are selected based on the Sigma method proposed in [21] and we use the newest method for selecting the the local best particles [3]. From the description of the above cases, we expect that Case 1 will not perform as well as Case 2 and PMOPSO. Also the homogeneous case (Case 3) using slow processors cannot outperform Case 2. In the experiments, we want to observe the quality of PMOPSO compared to all of the cases.

A. Test Functions

The above Scenario and the proposed PMOPSO method must be independent from the selected test problems. Here, we use some standard test functions from the multi-objective optimization literature. These test functions are two- and three-objective optimization problems selected from [24], [13] as shown in Table I. These test functions have different

TABLE I
TEST FUNCTIONS

Test	Function	Constraints
ZDT3	$g(x_2, \dots, x_n) = 1 + 9(\sum_{i=2}^n x_i)/(n-1)$ $h(f_1, g) = 1 - \sqrt{f_1/g} - (f_1/g) \sin(10\pi f_1)$ $f_1(x_1) = x_1$	$x_i \in [0, 1]$ $n = 30$ $i = 1, 2, \dots, n$
ZDT1	$g(x_2, \dots, x_n) = 1 + 9(\sum_{i=2}^n x_i)/(n-1)$ $h(f_1, g) = 1 - \sqrt{f_1/g}$ $f_1(x_1) = x_1$	$x_i \in [0, 1]$ $n = 30$ $i = 1, 2, \dots, n$
FF	$f_1(\vec{x}) = 1 - \exp(-\sum_i (x_i - \frac{1}{\sqrt{n}})^2)$ $f_2(\vec{x}) = 1 - \exp(-\sum_i (x_i + \frac{1}{\sqrt{n}})^2)$	$n = 10$ $x_i \in [-4, 4]$
DTLZ	$f_1(\vec{x}) = (1 + g(x_M)) \cos(x_1 \pi/2) \cos(x_2 \pi/2)$ $f_2(\vec{x}) = (1 + g(x_M)) \cos(x_1 \pi/2) \sin(x_2 \pi/2)$ $f_3(\vec{x}) = (1 + g(x_M)) \sin(x_1 \pi/2)$ $g(x_M) = \sum_{i=3}^8 (x_i - 0.5)^2$	$x_i \in [0, 1]$ $n = 8$

properties. ZDT3 has a disconnected Pareto-optimal front where all others have connected fronts.

B. Evaluations

The tests are run for 10 different seeds and we use the best and the average attainment surfaces for the evaluations [19], [15]. For two objective tests, we compare the hyper volume values of the different attainment surfaces [24]. For evaluating the results of the three objective test function, we measure the average and standard error of the hyper volumes of different runs.

C. Results

Figures 4 and 5 show the obtained average and best attainment surfaces for the two objective test problems by PMOPSO, Case 1, Case 2 and Case 3 methods.

These results are compared in terms of their hyper volume values in Table II. We observe that the PMOPSO method obtains much better solutions than Case 1 and MOPSO methods. These results are even better than the results of Case 2 for the Tests ZDT1 and ZDT3. For the FF problem, the result are comparable. However, comparing PMOPSO with Case 2 is not a fair comparison as Case 2 is run for a longer time than PMOPSO.

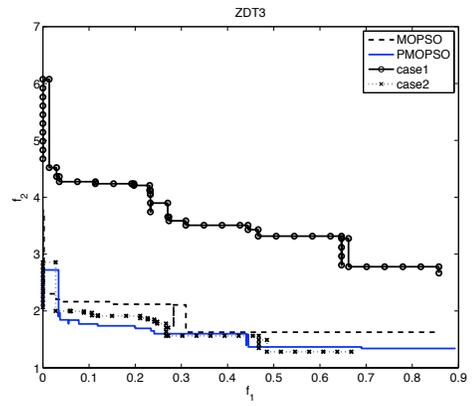
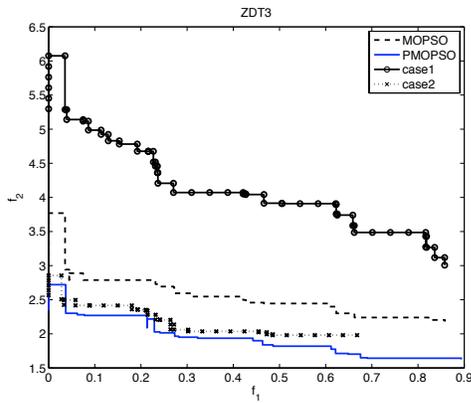
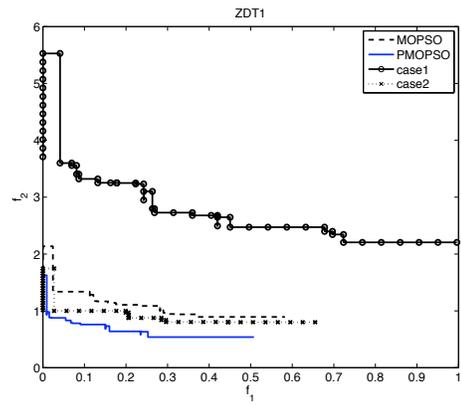
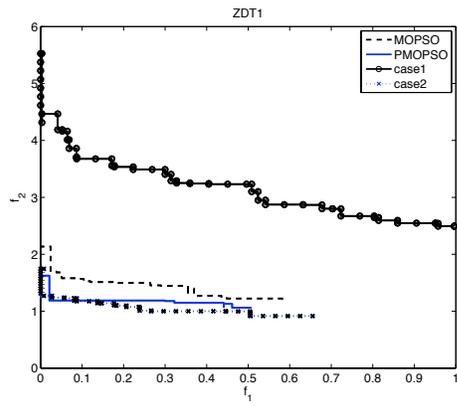
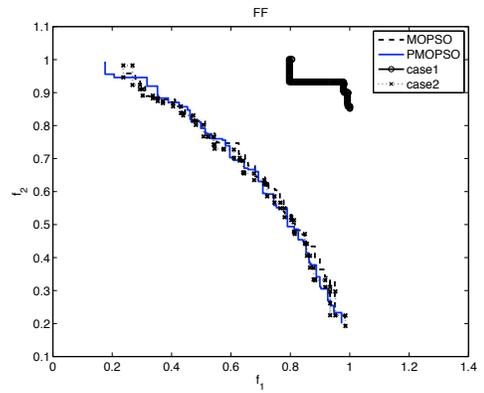
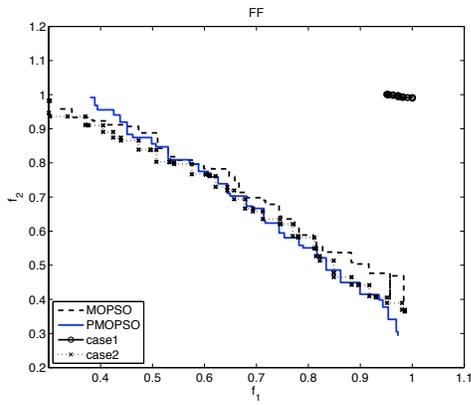


Fig. 4. The Average attainment surfaces for the FF, ZDT1 and ZDT3 test problems.

Fig. 5. The best obtained attainment surfaces for the FF, ZDT1 and ZDT3 test problems.

TABLE II
HYPER VOLUME VALUES FOR THE 2-OBJECTIVE PROBLEMS.

FF	Average	Best	Worst
PMOPSO	2.403	2.991	1.983
Case 1	0.625	1.053	0.502
Case 2	2.594	2.834	2.142
Case 3	2.512	2.795	1.542
ZDT1			
PMOPSO	2.873	3.502	2.438
Case 1	0.564	1.093	-0.169
Case 2	3.017	3.192	2.604
Case 3	2.632	3.034	1.866
ZDT3			
PMOPSO	1.986	2.425	1.454
Case 1	-0.366	0.274	-1.038
Case 2	1.702	2.431	1.339
Case 3	1.288	2.096	0.285

In all of the best attainment surfaces, PMOPSO outperforms the other methods. The Case 2 method obtains equal and better quality of solutions on average, compared to PMOPSO. The better performance of Case 2 compared to other methods was expected before starting the tests. Therefore the good solutions (or even better solutions than Case 2) of PMOPSO illustrate a considerable improvement of the results.

Table III shows the average and the standard error of the hyper volume value calculated for the 3-objective test problem over 10 runs. We observe that Case 2 has the highest

TABLE III
HYPER VOLUME VALUES FOR THE 3-OBJECTIVE PROBLEM.

FF	Average	Std. err
PMOPSO	7.0929	1.2×10^{-3}
Case 1	7.002	3.8×10^{-3}
Case 2	7.2539	5.69×10^{-4}
Case 3	7.1988	1.5×10^{-3}

hyper volume value, where Case 3 and PMOPSO methods have comparable values and all of them outperform Case 1. It can also be observed that Case 2 obtains results with lower standard error compared to others. The Case 3 method has the highest standard error value and this is due to the effects of the turbulence factor.

During the experiments, we noticed that the PMOPSO method performed better for the ZDT1 and TDZ3 problems than FF and DTLZ when compared to other cases. A possible reason for this feature could be the number of parameters. ZDT1 and ZDT3 test functions contain 30 parameters, where FF 10 and DTLZ have 8 parameters.

V. CONCLUSION AND FUTURE WORK

In this paper, we studied the Master-Slave model of parallelization to solve Multi-objective Problems on an heterogeneous set of resources. We proposed a new hybrid Parallel MOPSO method called PMOPSO which uses a Binary Search method to explore the unexplored regions of the search space and combined this with a MOPSO. PMOPSO is designed in a particular way so that the Master processor continues optimizing as soon as it receives evaluations from

a small number of processors. In this way we save the overall computation time. This approach was tested on a scenario containing a set of heterogeneous resources and compared with 3 other cases. The results show a good quality of solutions even when compared with a non-parallel case which takes a longer computation time.

The proposed approach is particularly suitable for very expensive multi-objective problems of real-world applications using an heterogeneous set of processors. In future, we want to study the parameter determining the minimum number of processors for which the Master node waits (denoted as N_s in the paper). In addition, we would like to analyze the quality of PMOPSO on other test problems with higher numbers of parameters and objectives.

REFERENCES

- [1] D. Abramson, A. Lewis, and T. Peachy. Nimrod/o: A tool for automatic design optimization. In *The 4th International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP 2000)*, 2000.
- [2] E. Alba and M. Tomassini. Parallelism and evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 6(5):443–461, 2002.
- [3] J. Branke and S. Mostaghim. About selecting the personal best in multi-objective particle swarm optimization. In T. P. Runarsson et al., editor, *Parallel Problem Solving from Nature*, LNCS, pages 523–532. Springer, 2006.
- [4] J. Branke, H. Schmeck, K. Deb, and M. Reddy. Parallelizing Multi-Objective Evolutionary Algorithms: Cone Separation. In *IEEE Congress on Evolutionary Computation*, pages 1952–1957, 2004.
- [5] Jürgen Branke, Andreas Kamper, and Hartmut Schmeck. Distribution of evolutionary algorithms in heterogeneous networks. In *Genetic and Evolutionary Computation Conference*, volume 3102 of LNCS, pages 923–934. Springer, 2004.
- [6] L. T. Bui, H. A. Abbass, and D. Essam. 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.
- [7] E. Cantu-Paz. A Survey of Parallel Genetic Algorithms. IlliGAL Report 97003, University of Illinois, 1997.
- [8] E. Cantu-Paz. Designing Efficient Master-slave Parallel Genetic Algorithms. IlliGAL Report 97004, University of Illinois, 1997.
- [9] E. Cantu-Paz. *Efficient and Accurate Parallel Genetic Algorithms*. Kluwer, 2000.
- [10] C. A. Coello Coello, D. A. Van Veldhuizen, and G. B. Lamont. *Evolutionary Algorithms for Solving Multi-Objective Problems*. Kluwer Academic Publishers, 2002.
- [11] K. Deb. *Multi-Objective Optimization using Evolutionary Algorithms*. John Wiley & Sons, 2001.
- [12] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan. A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II. In *Parallel Problem Solving from Nature VI (PPSN-VI)*, pages 849–858, 2000.
- [13] K. Deb, L. Thiele, M. Laumanns, and E. Zitzler. Scalable multi-objective optimization test problems. In *Congress on Evolutionary Computation*, pages 825–830. IEEE, 2002.
- [14] K. Deb, P. Zope, and A. Jain. Distributed computing of pareto-optimal solutions with evolutionary algorithms. In *International Conference on Evolutionary Multi-Criterion Optimization*, pages 534–549, 2003.
- [15] Carlos M. Fonseca and Peter J. Fleming. On the Performance Assessment and Comparison of Stochastic Multiobjective Optimizers. In *Parallel Problem Solving from Nature (PPSN IV), Lecture Notes in Computer Science*, pages 584–593, 1996.
- [16] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Publishing Company, Inc., Reading, Massachusetts, 1989.
- [17] Evan J. Hughes. Multi-objective Binary Search Optimisation. In *Evolutionary Multi-Criterion Optimization: Second International Conference (EMO)*, pages 72–87, 2003.
- [18] J. Kennedy and R. C. Eberhart. *Swarm Intelligence*. Morgan Kaufmann, 2001.

- [19] Joshua Knowles. A summary-attainment-surface plotting method for visualizing the performance of stochastic multiobjective optimizers. *IEEE Intelligent Systems Design and Applications (ISDA V)*, 2005.
- [20] S. Mostaghim, J. Branke, and H. Schmeck. Multi-objective particle swarm optimization on computer grids. In *The Genetic and Evolutionary Computation Conference*, volume 1, pages 869–875, 2007.
- [21] S. Mostaghim and J. Teich. Strategies for finding good local guides in multi-objective particle swarm optimization. In *IEEE Swarm Intelligence Symposium*, pages 26–33, 2003.
- [22] Margarita Reyes-Sierra and Carlos A. Coello Coello. Multi-objective particle swarm optimizers: A survey of the state-of-the-art. *International Journal of Computational Intelligence Research*, 2(3):287–308, 2006.
- [23] D. A. Van Veldhuizen, J.B. Zydallis, and G. B. Lamont. Considerations in engineering parallel multiobjective evolutionary algorithms. In *IEEE Transactions on Evolutionary Computation*, Vol. 7, No. 2, pages 144–173, April 2003.
- [24] E. Zitzler. *Evolutionary Algorithms for Multiobjective Optimization: Methods and Applications*. Shaker, 1999.