Reference Point based Distributed Computing for Multiobjective Optimization

Tolga Altinoz
Department of Electrical and Electronics Engineering
Ankara University, Turkey
taltinoz@ankara.edu.tr

Kalyanmoy Deb
Department of Electrical and Computer Engineering
Michigan State University
East Lansing, MI 48864, USA
kdeb@egr.msu.edu

COIN Report Number 2015006

Abstract—**As the complexity of problem and/or the number of objectives increase, a large population has to be evaluated at each generation of algorithm, and this process needs more computational resources, or costs longer time with the same computation power. However, distributing the tasks into different processors (or cores) is a good solution to speed up the computational process overall. In this study, a novel and practical distributed computing approach for multiobjective evolutionary optimization algorithms is proposed. Instead of dividing the objective space into cones as proposed in an earlier study, a subset of points initialized on a hyper-plane spanning the entire objective space is assigned to different processors and the R-NSGA-II procedure is invoked to find the respective partial efficient front. This distributed computing approach reduces the overall computational effort compared to that needed with a single-processor method.**

Keywords—distributed computing; R-NSGA-II; evolutionary multiobjective optimization

I. Introduction

Finding the solutions of multiobjective optimization problems greatly depends on the complexity of the problem. The "complexity" is related to mathematical relationship between formulations, order of objective equations, shape of the underlying Pareto-optimal front, number of objectives, and dimension of search space. For a complex problem, a relatively large population size is required to get a reasonable solution by covering the search over a sufficient part of the search space. Evaluating a large population needs more computational power. Hence, as the complexity increases, the necessity to make fast computation rises [1]. In general, to obtain results of the complex real-world engineering problems needs days or weeks. Fortunately, with the introduction of parallel and distributed computers and associated libraries and toolboxes, the execution time can be reduced dramatically. Although evolutionary algorithms (EAs) are ideally suitable for parallelization, much of the distributed EA studies have concentrated on the following two manners: (i) use multiple processors to distribute the evaluation of population members and (ii) use island-based EAs with occasional migration of solutions among processors to constitute a parallel search.

The first approach, also known as the 'Master-slave model' [2] is the most preferred method (especially in multi-core CPU and GPU implementations) in the academic researches and industrial applications. The main reason is the ease of modifying the sequential code to parallel implementation. The master processor controls the distribution of data on slave processors. The master orders to run different parts of the program or the same program in a parallel fashion on all machines. This model is useful especially for GPU implementations. The theoretical speedup for this model is directly proportional to number of cores/processors. For example, if there is K number of processors then theoretically the algorithm becomes K times faster than sequential code.

The island model [3] allows the optimization algorithms to run independently at different processors, but processors have to share some information among each other regularly. After a specific number of generations, some solution candidates at each processor are shared to each other (called 'migration'). This model is suitable for clusters that are heterogeneous, physically distant from each other and communication between processors is limited or expensive to achieve. The 'diffusion model' [4] is a special increased bandwidth version of the island model, because individuals at each processor need information/data more often from other processors. This model is suitable for distributed machines with parallel units like GPU.

Although, the master-slave approach is straightforward and can at most provide almost a linear speed-up in computations with number of processors, the second approach is promising, intellectually intriguing, and has the potential to produce a super linear speed-up [5] with an increase in number of processors.

Evolutionary multi-objective optimization (EMO) algorithms aim at finding multiple trade-off Pareto-optimal solutions, instead of a single optimal solution at the end of their tasks. There have been a few studies in parallelizing an EMO procedure [1-4, 6], but the topic has a tremendous scope of using both types of parallelizations discussed above. Two different distributed computing methods were suggested by the second author and his collaborators in the past [3]. The first approach used a different cone-domination principle for each processor and the relevant portion of the Pareto-optimal front was then found by independently running an EMO algorithm in each processor [3, 4]. Occasional migration of non-dominated solutions from one processor was sent to other processors for information exchange. This idea is simple and uses a key concept (emphasizing non-dominated solutions) of multi-objective optimization; however, one significant criticism of the study was its limitation to work on problems having convex-shaped efficient fronts. For problems having non-convex efficient fronts, the approach is not expected to find intermediate solutions in the non-convex part of the efficient front. In the second study [6], the objective space is explicitly divided by defining explicit artificial constraints and each divided part of the objective space was declared feasible for each processor. The location of the constraints was determined adaptively based on the range of the non-dominated front discovered by each processor. Although this approach can be applied generically to handle any shape of an efficient front, the idea is algorithmically unappealing and a brute force constraint-based strategy is used to distribute the computing task.

In this study, we borrow another key concept from multi-objective optimization and follow the island model approach. Instead of cone domination, we use a reference point based approach [7] in which for each processor a set of contiguous reference points are pre-assigned to find a part of the efficient front. Due to the use of reference-point based approach, the overall algorithm is generically applicable to solve problems having any shape of the efficient frontier.

The remainder of the paper is constructed as follows. Section II gives a brief description of the existing cone domination and constraint-based approaches for distributed computing of the efficient front. Section III gives a detail description of our proposed method. The implementation is divided into two sub-sections based on number of objectives of problems. Section IV presents the results of our proposed approach. The last section outlines the conclusion and future work for this study.

II. Existing Distributed Computing Approaches For Multiobjective Optimization

Here, we concentrate on two island-based distributed computing approaches which have been proposed in the past.

A. *Cone-Domination Approach*

For a two-objective minimization problem, the usual domination principle makes all solutions in the positive quadrant of a solution x dominated by it (Fig 1(a)). An alternative view of this definition is that solution x dominates all solutions lying in a positive cone in the objective space spanning a 90 degree angle. Thus, all solutions that do not get dominated by any other solutions in the feasible search space are, by definition, Pareto-optimal solutions. The respective objective space points are called efficient solutions and the corresponding trade-off boundary formed by efficient solutions is called an 'efficient' front. This definition can be easily extended for any number of objectives. However, if the definition of the dominating cone is expanded (to be more than 90 degrees in two-objective problems, for example), a subset of the original efficient front will become the new cone-dominated front, as shown in Fig 1(b).

In an island approach proposed elsewhere [3], a different cone was assigned to each processor in a systematic manner so that every feasible original Pareto-optimal solution becomes a member of the cone-dominated Pareto-optimal set for at least one of the processors. Although the concept is interesting, the cone-domination principle works for problems having a convex efficient front and will fail to identify the entire efficient front for non-convex problems.

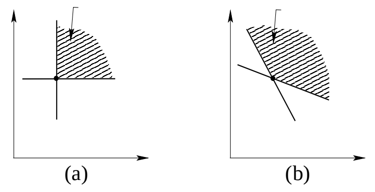

Fig 1: Original domination and cone domination concepts are illustrated.

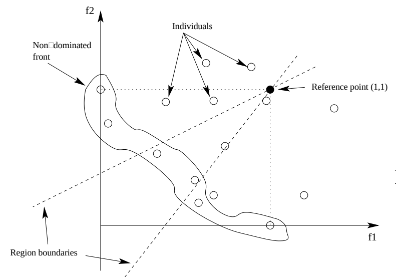

Fig 2: Constraint-domination approach is illustrated.

B. *Cone-Domination Approach*

To have a generic strategy which will work on both convex and non-convex problems, artificial constraints were added for each processor to define the scope of the search process for each processor [6], as shown in Fig. 2. Each processor used the same domination principle. The method can be extended for higher number of objectives, but due to deterministic division of the objective space, an exponentially large number of processors are needed for a linear increase in objectives. The original study [3] proposed an adaptive placement of the constraints to alleviate this problem somewhat

III. Proposed Distributed Computing Approach

The proposed idea used the concept of reference point based multi-objective optimization [7]. Like in MOEA/D [8] or NSGA-III [9, 10] procedures, in the first step, we begin with a set of trade-off reference points, as shown in Fig. 3(a) for a two-objective problem. For higher objectives, reference points are created uniformly on a higher-dimensional hyper-plane.

The points on the line (or a plane) for each processor are determined as the second step. In Fig. 3(b), equally distributed points are demonstrated. The number of points and distribution of them affect the performance of R-NSGA-II on each processor. If the points are selected too close each other, number of points and computation time of algorithm increase. Otherwise, solutions don't spread enough. Therefore, in this study, equally distributed points are considered and distributed. When line (in bi-objective) is equally divided and points are obtained, the borders of points between neighborhood processors may cause overlaps. Hence, in this study, different border points are considered and evaluated as cases.

The motivation of this approach is to distribute points to processors (third step) and find Pareto solutions closest to these point(s) by using reference-point based multi-objective optimization algorithms. In the next section, R-NSGA-II algorithm [7] is used.

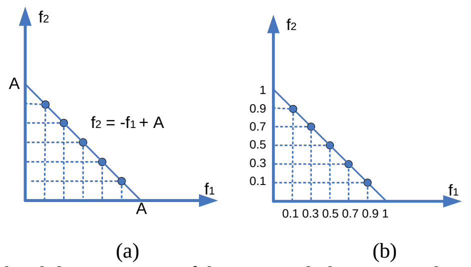

Fig. 3. Graphical demonstration of the proposed idea on two-objective space.

A. *Reference point based NSGA-II or R-NSGA-II*

Reference point based NSGA-II (R-NSGA-II) is a modified version of original NSGA-II with the tournament selection, the real-coded SBX recombination, and the polynomial mutation operators [11]. Instead of the crowding distance operator, R-NSGA-II considers the normalized Euclidean distance between reference point(s) and population members. The aim of the algorithm is to emphasize non-dominated solutions that are closest to the supplied reference point(s). In the event of multiple reference points, since a widely spread out reference points are supplied in the objective space (as in Fig 3(b)), R-NSGA-II is expected to find a widely separated set of Pareto-optimal front at the end. Since, we use R-NSGA-II concept in our proposed distributed computing approach, we discuss some details of the R-NSGA-II method.

The normalized distance is calculated as follows:

1- First, the combined population (parent and newly created offspring populations) of size 2N is classified into a number of non-dominated fronts of increasing domination level, as done in NSGA-II [11]. To choose N members for the next generation, the crowding distance operator of NSGA-II is replaced with the following two steps.

2- For each non-dominated front starting with the first one, calculate a normalized and weighted Euclidean distance d_{ref}, as follows:

$$d_{ref} = \sqrt{\sum_{i=1}^{M} w_i \left(\frac{f_i - z_i}{f_i^{\max} - f_i^{\min}} \right)^2}, \qquad (1)$$

where M is the number of objectives, $\boldsymbol{f}$ is the objective vector, $\boldsymbol{z}$ is the particular reference point, $\boldsymbol{f}^{\max}$ and $\boldsymbol{f}^{\min}$ are the maximum and minimum objective vectors, and $\boldsymbol{w}$ is a pre-specified weight vector. In the absence of any preference information, it was proposed to use a vector of ones for $\boldsymbol{w}$.

3- The ε-clearing idea is applied next to maintain a diverse set of solutions near each reference point. First, the closest population member $\boldsymbol{x}_R$ for reference point R on each front is identified and saved as the top-most member of a sorted list. Thereafter, all neighboring front solutions within a normalized Euclidean distance of ε from $\boldsymbol{x}_R$ are temporarily cleared for further consideration. The next closest solution to the same reference point on the front is then identified and declared as the next member on the sorted list. This process is continued until all first front members are considered. Thereafter, the procedure is repeated for the second front members and so on. After considering all fronts, if still the size of the sorted list is less than the population size, the cleared members are considered starting from front one. A similar clearing mechanism is followed while picking solutions. This procedure is stopped as soon as the sorted list has N members.

In this paper, R-NSGA-II algorithm is used as main optimization algorithm for each processor as explained in the previous section. However, each processor uses a different set of reference points. Ideally, a non-overlapping set of reference points will be desired, but the choice of niching parameter ε is also an important parameter to set. We address these issues in the following section. Due to the use of a fraction of reference points for each processor, each is expected to find a part of the efficient front and when all processors have finished their runs, they together should have found representative points on the entire efficient front.

IV. Proof-Of-Principle Results

In this section, we consider two-objective ZDT1 problem to illustrate the working of our proposed R-NSGA-II based distributed computing algorithm. The ZDT1 problem requires minimization of two objectives f_1 and f_2. Here, we consider a 30-dimensional $\boldsymbol{x}$ vector; $\boldsymbol{x}=\{x_1,x_2,\ldots,x_{30}\}$. The equally distributed points on the line between points at each axis are assigned for two processors as given in Fig. 4. There are two dependent points {a,b} should be properly assigned. If solutions overlap, then total performance of distributed algorithm decreases. Therefore, points near the border of processors are investigated before presenting the results for more complex problems.

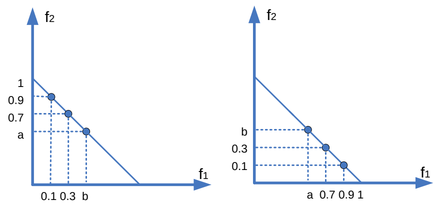

Fig. 4. Point allocation for two processors.

Four cases are considered in this study (population size is 50 for each processor). The first case is the overall result of R-NSGA-II without distributed computing (single processor, population size is 100). Other cases are graphically shown in Fig. 5 is run with 50 population for each processor.

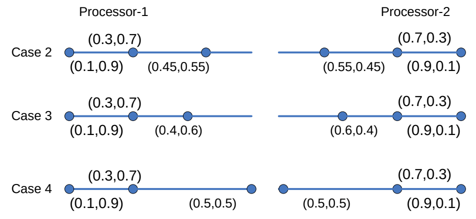

Fig. 5. Three cases are considered for determinaton of bordering points for two processors.

Each case considers three points for each processor marked with their location on the hyper-plane in Fig. 5. Table I gives the statistical results for 10 independent runs of Cases 1-4 until the hyper-volume metric is greater than 0.794 with hyper-volume reference point is (1.0646,1.0646). The number of generations and required number of total function evaluations for two processors are given in Table I. Best results are shown in bold.

TABLE I. FOUR CASE EVALUATION FOR DISTRIBUTED COMPUTING, WHERE MIN ISTHE MINIMUM, MAX IS THE MAXIMUM, STD.DEV. IS THE STANDARD DEVIATION, MEAN ISTHE AVERAGE OF INDEPENDENT RUNS, GEN IS THE AVERAGE NUMBER OF GENERATIONS PER PROCESSOR, AND FUNC.EVA. IS THE TOTAL NUMBER OF FUNCTION EVALUATIONS.

	Case - 1		**Case - 2**	
	Gen.	*Func. Eva.*	*Gen.*	*Func. Eva.*
Min	**54**	**6056**	55	6170
Max	**66**	**7331**	77	8539
Std.Dev.	**3.68**	**399.3**	7.78	844.2
Mean	**59.4**	**6633.5**	66.3	7376.4
	Case - 3		**Case - 4**	
	Gen.	*Func. Eva.*	*Gen.*	*Func. Eva.*
Min	52	5829	58	6493
Max	86	9566	84	9363
Std.Dev.	11.2	1238.4	9.17	994.8
Mean	63.8	7125.9	68.4	7626.7

Among Cases 2-4 (two processors), Case 3 requires the smallest number of generations and function evaluations. For an average performance, about 7,126 function evaluations are required in total for both processors. Thus, although none of the two-processor cases is able to reduce the function evaluations from a single-process run, the proposed idea seems to work in distributing points in two independent processors so as to have an equivalent hyper-volume to that obtained in the single-processor run. This may be attributed to the fact that ZDT1 problem is relatively easy to solve and no real advantage is found by using a distributed computing effort the way it is performed here.

To demonstrate the spread of solutions found by both processors, we randomly select points from one run (out of 10 independent runs performed) of the two-processor Case-3 for problem ZDT1 and show them in Fig. 6. A nicely distributed set of objective points can be observed from the figure.

Next, we show the performance of the proposed method on a non-convex problem. ZDT2 problem is solved by using Case-3 parallelization with identical parameter settings and the obtained points are presented in Fig. 7. A good spread of solutions is obtained.

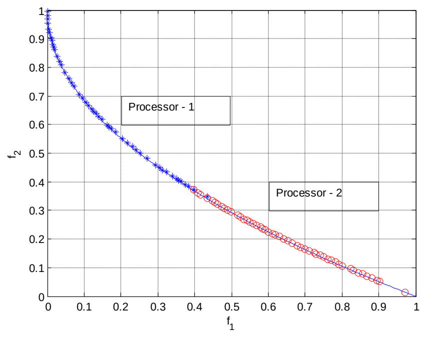

Fig. 6. Obtained non-dominated solutions of ZDT1 using two processsors.

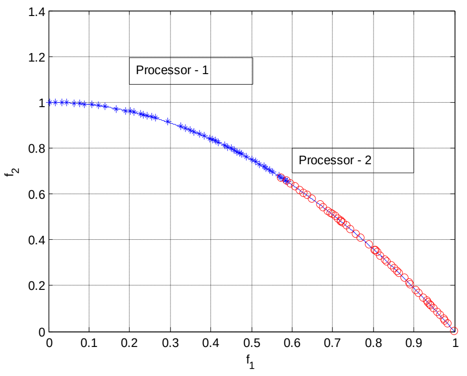

Fig. 7. Obtained non-dominated solutions of ZDT2 using two processsors.

Next, we divide all reference points into three groups to investigate the effect of three processors. The reference points are graphically shown in Fig. 8. Here, we deliberately use a single point for one of the processors so as to investigate the efficacy of using a single point instead of a set of reference points in every processor.

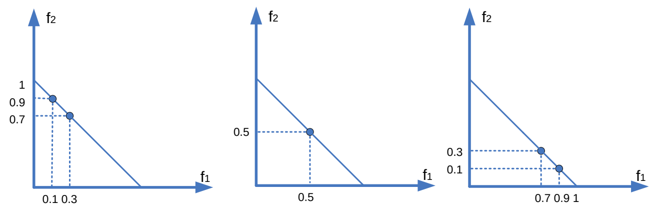

Fig. 8. Reference point allocation for three processors.

Table II presents the results for the three-processor division on ZDT1 problem. The single-processor run uses 150 members in population and multi-processor run using 50 members in each of them. 10 independent runs are performed for each case. Although 60% less function are evaluated per processor, 20% more generations are needed overall. From Table I and II, it is concluded that as the number of processors increase the number of function evaluation starts to decrease but the number of generation increases. In any case, the single-processor performance is still better for the ZDT1 problem.

TABLE II. THREE PROCESSORS FOR DISTRIBUTED COMPUTING WHERE MIN ISTHE MINIMUM, MAX IS THE MAXIMUM, STD.DEV. IS THE STANDARD DEVIATION, MEAN ISTHE AVERAGE OF INDEPENDENT RUNS, GEN IS THE AVERAGE NUMBER OF GENERATIONS PER PROCESSOR, AND FUNC.EVA. IS THE TOTAL NUMBER OF FUNCTION EVALUATIONS.

	Single Processor		**Three Processors**	
	Gen.	*Func. Eva.*	*Gen.*	*Func. Eva.*
Min	**39**	**6560**	50	8435
Max	**55**	**9281**	70	11731
Std.Dev.	**4.71**	**780.89**	5.71	947.5
Mean	**47.3**	**7949**	56.6	9496.7

Figs. 9 and 10 show the results on ZDT1 using three and four processors for 10 independent runs. The reference points used for four processors are: {[0.1,0.9]-[0.2,0.8]; [0.3,0.7]-[0.4,0.6]; [0.6,0.4]-[0.7,0.3]; [0.8,0.2]-[0.9,0.1]}. In each case, we choose an appropriate ε value so that there is no overlap between obtained solutions of each processor.

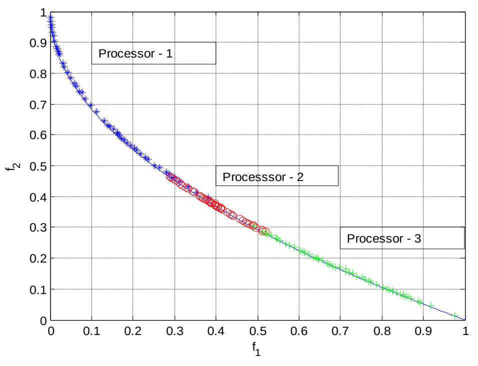

Fig. 9. Obtained non-dominated solutions of ZDT1 using three processsors.

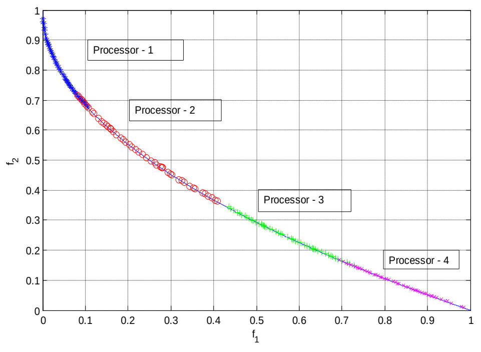

Fig. 10. Obtained non-dominated solutjons of ZDT1 using four processsors.

V. An Efficient Implementation And Results

The results obtained in the previous section indicate that the reference point based concept works in distributing the task of finding efficient points uniformly by multiple processors. In fact, due to the control of reference points the obtained distribution in obtained trade-off solutions is much better than those obtained using the cone-domination or constraint based distributing computing approaches executed earlier [3,6]. However, in none of the multi-processor case, we were able to achieve a computational gain compared to the single-processor runs. In this section, we propose a delayed distributed computing approach to alleviate this computational issue. First, we build up the necessary background for our proposed delayed approach. A population-based EMO requires a certain population size (N_{req}) depending on its operators and representation used to solve a problem. Assuming that a single-processor EMO is run with this population size, an P-processor run must then distribute its individual population sizes (for each process) such that overall population size of N_{req}. Without loss of generality, this means the setting of a population size of N_{req}/P for each processor. But, since each processor is trying to solve the original problem, albeit with a smaller scope of the efficient front, this reduced population size may not be adequate for each processor to bring its population close to Pareto-optimal front quickly.

TABLE III. Two processors for Delayed distributed Computing Approach for ZDT1 Problem. Min isthe minimum, Max is the maximum, Std.Dev. is the standard deviation, Mean isthe average of 10 independent runs. Gen Is The Average Number Of Generations And Func.Eva. Is The *Total* Number Of Function Evaluations needed for both processors.

	T_{delay}=5		T_{delay}=10		T_{delay}=15		T_{delay}=20	
	Gen.	*Func. Eva.*	*Gen.*	*Func. Eva.*	*Gen.*	*Func. Eva.*	*Gen.*	*Func. Eva.*
Min	55	6151	47	5307	48	5384	**49**	**5467**
Max	158	17527	72	8006	65	7209	**59**	**6586**
Mean	75.7	8429.6	57.4	6415.3	57.5	6422.3	**52.8**	**5902.1**
Std.Dev.	31.41	3469.3	8.07	875.67	5.1	555.17	**2.85**	**320.24**
	T_{delay}=25		T_{delay}=30		T_{delay}=35		T_{delay}=40	
	Gen.	*Func. Eva.*	*Gen.*	*Func. Eva.*	*Gen.*	*Func. Eva.*	*Gen.*	*Func. Eva.*
Min	47	5269	45	5061	48	5344	43	4830
Max	67	7468	71	7921	63	7020	95	10538
Mean	55.8	6235.7	53	5932.3	53.5	5979.2	55.7	6238.8
Std.Dev.	6.16	675.76	7.6	834.47	5.46	607.99	15.26	1676.49

TABLE IV. Three processors for Splitted distributed Computing Approach for ZDT1 Problem. Func.Eva. indicates total number of function evaluations needed by all three processors.

	T_{delay}=5		T_{delay}=10		T_{delay}=15		T_{delay}=20	
	Gen.	*Func. Eva.*	*Gen.*	*Func. Eva.*	*Gen.*	*Func. Eva.*	*Gen.*	*Func. Eva.*
Min	44	7471	42	7109	41	6984	35	5960
Max	303	50018	217	35940	197	32785	52	8718
Mean	132.8	22054.1	81.4	13586.5	60.4	10130.5	42.9	7223.4
Std.Dev.	88.37	14544.8	51.1	8419.29	48.07	7971.22	4.97	821.21
	T_{delay}=25		T_{delay}=30		T_{delay}=35		T_{delay}=40	
	Gen.	*Func. Eva.*	*Gen.*	*Func. Eva.*	*Gen.*	*Func. Eva.*	*Gen.*	*Func. Eva.*
Min	36	6100	36	6054	**36**	**6091**	40	6735
Max	46	7726	56	9406	**46**	**7801**	43	7242
Mean	40.4	6816.2	40	6746	**39.9**	**6730.1**	40.7	6870.5
Std.Dev.	8.87	480.43	6.37	1054.23	**2.96**	**497.61**	0.94	155.26

TABLE V. TWO PROCESSORS FOR DELAYED DISTRIBUTED COMPUTING APPROACH FOR ZDT2 PROBLEM.

	T_{delay}=5		**T_{delay}=10**		**T_{delay}=15**		**T_{delay}=20**	
	Gen.	*Func. Eva.*	*Gen.*	*Func. Eva.*	*Gen.*	*Func. Eva.*	*Gen.*	*Func. Eva.*
Min	61	6808	60	6692	53	5954	48	5399
Max	91	10014	82	9110	80	8894	83	9230
Mean	78.6	8722.5	72.1	8028.1	62.7	7004.2	63.5	7092
Std.Dev.	10.6375	1153.24	9.1706	998.94	9.2742	1015.67	13.032	1436.39
	T_{delay}=25		**T_{delay}=30**		**T_{delay}=35**		**T_{delay}=40**	
	Gen.	*Func. Eva.*	*Gen.*	*Func. Eva.*	*Gen.*	*Func. Eva.*	*Gen.*	*Func. Eva.*
Min	53	5970	49	5474	40	4501	**44**	**4953**
Max	75	8293	89	9905	72	8017	**78**	**8683**
Mean	65.4	7294.6	62.8	7009.9	61.4	6865	**60.1**	**6716.1**
Std.Dev.	7.04	751.57	12.46	1369.17	9.68	1063.42	**9.97**	**1092.82**

TABLE VI. THREE PROCESSORS FOR SPLITTED DISTRIBUTED COMPUTING APPROACH FOR ZDT2 PROBLEM.

	T_{delay}=5		**T_{delay}=10**		**T_{delay}=15**		**T_{delay}=20**	
	Gen.	*Func. Eva.*	*Gen.*	*Func. Eva.*	*Gen.*	*Func. Eva.*	*Gen.*	*Func. Eva.*
Min	55	9280	58	9651	40	6803	44	7438
Max	75	12517	79	13210	79	13138	78	13024
Mean	67	11216	69.2	11567.8	60.5	10133.8	59.3	9938.8
Std.Dev.	6.6667	1081.09	7.8712	1319.34	12.0393	1959.87	11.7289	1936.01
	T_{delay}=25		**T_{delay}=30**		**T_{delay}=35**		**T_{delay}=40**	
	Gen.	*Func. Eva.*	*Gen.*	*Func. Eva.*	*Gen.*	*Func. Eva.*	*Gen.*	*Func. Eva.*
Min	43	7260	**42**	**7062**	35	5906	40	6768
Max	75	12502	**56**	**9405**	72	12048	70	11650
Mean	58.4	9782.4	**50.2**	**8436.2**	53.5	8968	50.4	8471
Std.Dev.	11.17	1828.36	**4.66**	**772.87**	10.55	1749.82	8.55	1396.73

One way to alleviate this issue is to delay the use of the multi-processor distributed computing approach until a certain number of generations. To exploit the availability of P processors, the first T_{delay} number of generations can be performed using the master-slave approach and the remaining generations are processed using the above-discussed reference point based the island model approach. Fig. 11 illustrates this hybrid parallelization approach.

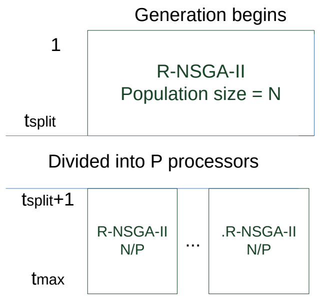

Fig. 11. Obtained nondominated solutions of DTLZ2 for a) three and b) four processsors.

A. Two-objective Problems

We implement this hybrid approach and present results on ZDT1 and ZDT2 problems using two and three processors. Results are summarized in Tables III and IV for T_{delay} = 5 to 40 for two and three processors for ZDT1 problem and Tables V and VI for ZDT2 problems, respectively, Tables give the statistical results for 10 independent runs until the hyper-volume metric is greater than 0.794 for ZDT1 problem and 0.460 for ZDT2 problem with hyper-volume reference point is (1.0646, 1.0646). Tables show a remarkable result. The use of multiple processors allowed a smaller number of total function evaluations (11% smaller for 2-processors, 15% smaller with 3-processors for ZDT1, for example) needed to find a set of points having an identical hyper-volume as that obtained using the single-processor method.

To understand the effect of T_{delay} better, the average number of total two-processor function evaluations over 10 runs with respect to T_{delay} is shown in Fig. 12 with data taken from Tables III and IV for ZDT1. It is clear that for two processor case, there exists an optimal plateau from T_{delay} = 15 to 40 generations for which both the number of generations and function evaluations are small. For the three processor case, T_{delay} = 20 to 40 are better. Interestingly, for both two and three processors $T_{\text{delay}} \geq 20$ perform well. This is remarkable in the sense that the use of multiple processors is able to help reduce the overall computational effort. In fact, a comparison between the two tables indicates that the three processors make a bigger advantage than a two-processor case.

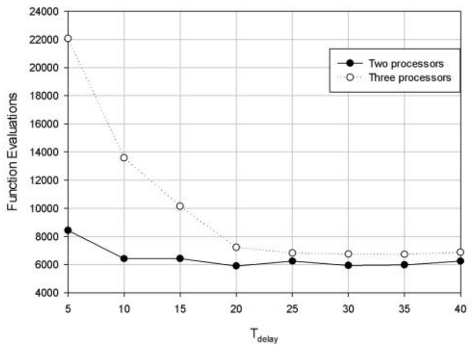

Fig. 12. Parametric study of T_{delay} for ZDT1 problem.

Table VII summarizes the results obtained using a single-processor, two-processor with the original reference point based approach, and two-processor with delayed approach. It is clear that the delayed approach (shown in bold) is computationally faster. The same can be observed for the three-processor case for ZDT1 problem in Table VIII.

TABLE VII. TWO PROCESSORS FOR DISTRIBUTED COMPUTING FOR ZDT1 PROBLEM.

	Single Proccessor		**Two Processors**		T_{delay}**=30**	
	Gen.	*Func.Eval.*	*Gen.*	*Func.Eva.*	*Gen.*	*Func.Eva.*
Min	54	6056	52	5829	**45**	**5061**
Max	66	7331	86	9566	**71**	**7921**
Mean	59.4	6633.5	63.8	7125.9	**53**	**5932.3**
Std.Dv.	3.68	399.3	11.2	1238.4	**7.6**	**834.47**

TABLE VIII. THREE PROCESSORS FOR DISTRIBUTED COMPUTING FOR ZDT1 PROBLEM.

	Single Proccessor		**Three Processors**		T_{delay}**=30**	
	Gen.	*Func.Eval.*	*Gen.*	*Func.Eva.*	*Gen.*	*Func.Eva.*
Min	39	6560	50	8435	**36**	**6054**
Max	55	9281	70	11731	**56**	**9406**
Mean	47.3	7949	56.6	9496.7	**40**	**6746**
Std.Dv.	4.71	780.89	5.71	947.5	**6.37**	**1054.23**

Next, the hybrid approach is implemented for ZDT2 problem using two and three processors. Similar analysis is performed and results are summarized in Tables IX and X for two and three processors with T_{delay} = 30, respectively. No definite conclusion can be drawn for ZDT2, as the best performance is shared between single processor and multi-processor delayed approaches. Although a clear advantage was observed for ZDT1 for the multi-processor delayed approach, in ZDT2 the results are mixed. Further investigations on more difficult problems are needed to make better conclusions.

Fig. 13 gives the average number of function evaluations for two and three processors results, which are given in Tables V and VI. The best performance is obtained for $T_{\text{delay}} \geq 30$ generations.

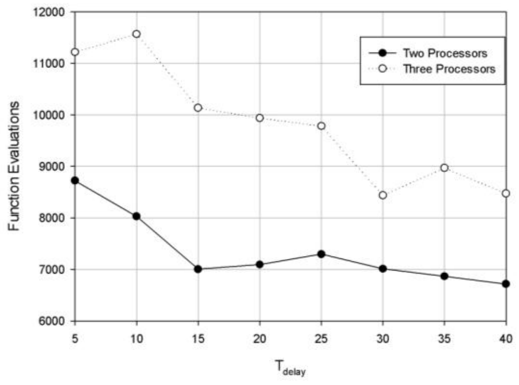

Fig. 13. Average number of function evaluations with respect to T_{delay} plot for two and three processor fot ZDT2 problem.

TABLE IX. THREE PROCESSORS FOR DISTRIBUTED COMPUTING FOR ZDT2 PROBLEM.

	Single Proccessor		**Two Processors**		T_{delay}**=30**	
	Gen.	*Func.Eval.*	*Gen.*	*Func.Eva.*	*Gen.*	*Func.Eva.*
Min	53	5795	59	6575	**49**	**5474**
Max	**76**	**8311**	80	8881	89	9905
Mean	**59.3**	**6508.2**	67.7	7547.7	62.8	7009.9
Std.Dv.	**7.08**	**764.07**	5.85	646.93	12.46	1369.17

TABLE X. THREE PROCESSORS FOR DISTRIBUTED COMPUTING FOR ZDT2 PROBLEM.

	Single Proccessor		**Three Processors**		T_{delay}**=30**	
	Gen.	*Func.Eval.*	*Gen.*	*Func.Eva.*	*Gen.*	*Func.Eva.*
Min	**30**	**4949**	51	8579	42	7062
Max	56	9211	77	12860	**56**	**9405**
Mean	**44.3**	**7293**	65.7	11001	50.2	8436.2
Std.Dv.	6.46	1064.54	7.32	1210.77	**4.6619**	**772.87**

B. Three-Objective Problems

Next, we consider DTLZ2 problem [12] for T_{delay}=30. First, we need to define suitable clusters of reference points for each processor which all must lie on a unit hyper-plane (Fig. 14(a)). As mentioned, Das and Dennis's [13] approach can be used for this purpose to create a set of uniformly distributed reference points on the hyper-plane. Then, we require clustering the points into *P* divisions for a *P*-processor run. Fig. 14(b) shows the division of points for two processors.

Fig. 14(c) and Fig. 15(a) presents the obtained results for three and four processors for the DTLZ2 problem in which the hyper-plane is split into three and four divisions accordingly.

To demonstrate further working, the delayed approach is applied to three-objective DTLZ1 and DTLZ3 problems using two processors and results are shown in Fig. 15(b) and 15(c).

VI. CONCLUSIONS

In this study, a novel distributed computation approach based on the reference point based concept for solving multi-objective optimization problems has been proposed. The proposed method is based on assigning reference points on a predefined hyper-plane in the objective space for different processors. The extensive results presented here have shown that the proposed approach is useful and easy to apply for multi-objective problems. Importantly, unlike an existing cone-domination based approach, it works on both convex and concave problems. We have also proposed a delayed distributed computing approach in which the distributed approach begins after certain numbers of initial generations are elapsed. In general, the delayed approach has been computationally faster in arriving at the same hyper-volume than a single processor EMO, but further investigations are needed to make the idea perfect. The use of a migration policy to take help from one processor to another should improve the performance of the overall approach and we plan to investigate it in the near future. We also plan to extend the delayed reference point based approach to many-objective optimization problems as well.

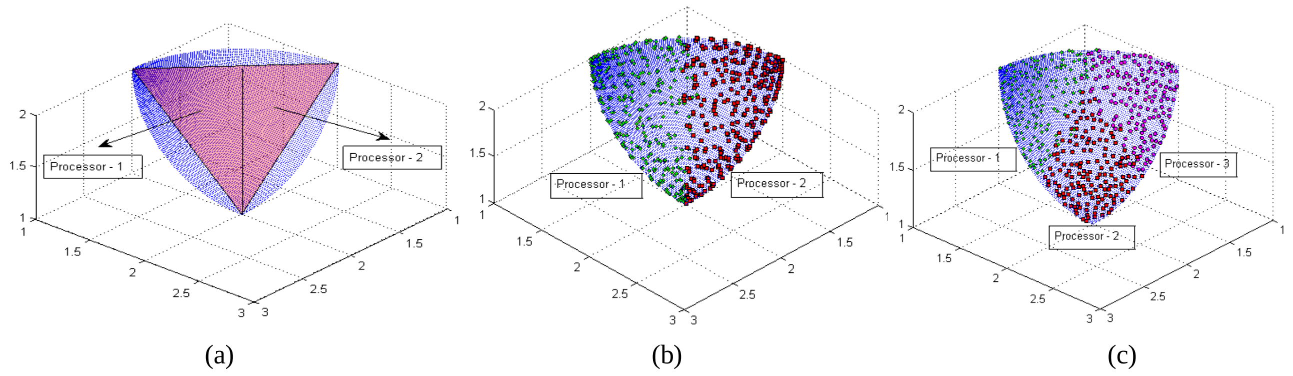

Fig. 14. Division strategy obtained non-dominated solutions of DTLZ2 for (a) two-processor case and (b) three-processor case.

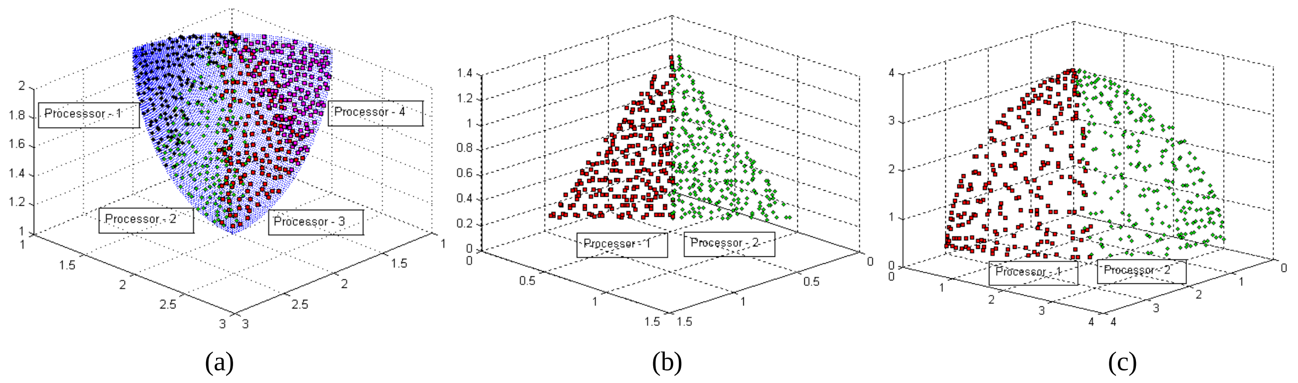

Fig. 15. Division strategy obtained non-dominated solutions of(a) DTLZ2 for four-processor case,(b) DTLZ1 for two-processor case, and (c) DTLZ3 for two-processor case.

Acknowledgment

O. Tolga Altinoz has been supported by the grant from TUBITAK under 2214/A Research Scholar Program.

References

[1] N. Melab, M. Mezmaz, and E. Talbi,"Parallel cooperative meta-heuristics on the computational grid: A case study: the bi-objective flow-shop problem," Parallel computing, vol. 32, no. 9, pp. 643-659, 2006.

[2] S. Mostaghim, J. Branke, A. Lewis, and H. Schmeck, "Parallel multi-objective optimization using master-slave model on heterogeneous resources," IEEE Evolutionary Computation CEC, pp. 1981-1987. 2008

[3] K. Deb, P. Zope, and A. Jain. Distributed computing of pareto-optimal solutions with evolutionary algorithms. In C. M. Fonseca, P. J. Fleming, E. Zitzler, K. Deb, and L. Thiele, editors, Evolutionary Multi-Criterion Optimization, volume 2632 of LNCS, pages 534–549. Springer, 2003.

[4] H. Schmeck, U. Kohlmorgen, and J. Branke. Parallel implementations of evolutionary algorithms. In A. Zomaya, F. Ercal, and S. Olariu, editors, Solutions to Parallel and Distributed Computing Problems, pages 47–66.Wiley, 2001.

[5] E. Zitzler, K. Deb, and L. Thiele. Comparison of multiobjective evolutionary algorithms. Evolutionary Computation, 8(2):125–148, 2000.

[6] J. Brake, H. Schemek, k. Deb, and S.M. Reddy, "Parallelizing multi-objective evolutionary algorithms: cone separation," Congress on Evolutionary Computation, pp. 1952 - 1957, 2004.

[7] K. Deb, J. Sundar, U. B. R. N, and S. Chaudhuri, "Reference pointbased multi-objective optimization using evolutionary algorithm," International Journal of Computational Intelligence Research, pp. 273–286,2006.

[8] Q. Zhang, and H. Li, "MOEA/D: A Multiobjective Evolutionary Algorithm Based on Decomposition," IEEE Transactions on Evolutionary Computation, vol. 11, no. 6, pp. 712-731, 2007.

[9] K.Deb, and H. Jain, "An Evolutionary Many-Objective Optimization Algorithm Using Reference-Point-Based Nondominated Sorting Approach, Part I: Solving Problems With Box Constraints," IEEE Transactions on Evolutionary Computation, vol. 18, no. 4, pp. 577-601, 2014.

[10] H. Jain, and K. Deb, "An Evolutionary Many-Objective Optimization Algorithm Using Reference-Point Based Nondominated Sorting Approach, Part II: Handling Constraints and Extending to an Adaptive Approach," IEEE Transactions on Evolutionary Computation, vol. 18, no. 4, pp. 602-622, 2014.

[11] K. Deb, a. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitistmultiobjective genetic algorithm: NSGA-II," IEEE Transactions onEvolutionary Computation, pp. 182–197, 2002.

[12] K. Deb, L. Thiele, M. Laumanns and E. Zitzler. "Scalable Multi-Objective Optimization Test Problems,". CEC 2002, pp. 825-830, 2002

[13] I. Das and J. Dennis, "Normal-boundary intersection: A new method for generating the Pareto surface in nonlinear multicriteria optimization problems," SIAM Journal of Optimization, vol. 8, no. 3, pp. 631–657, 1998.