Parallel and Distributed MOEA/D with Exclusively Evaluated Mating and Migration

Yuji Sato
Faculty of Computer and Information Sciences
Hosei University
Tokyo, Japan
yuji@k.hosei.ac.jp

Mikiko Sato
School of Information and Telecommunication Engineering
Tokai University
Tokyo, Japan
mikiko.sato@tokai.ac.jp

Mads Midtlyng
Graduate School of Computer and Information Sciences
Hosei University
Tokyo, Japan
midtlyng.madsalexander.9c@stu.hosei.ac.jp

Minami Miyakawa
Department of Electronic Information Systems Engineering
Sinshu University
Nagano, Japan
miyakawa@shinshu-u.ac.jp

***Abstract*—This paper proposes a method for many-core-based large-scale parallel and distributed computation of MOEA/D, a decomposition-based evolutionary multi-objective optimization algorithm. Standard parallel MOEA/D on many-core environments provides fast execution time, but uniformity and diversity of the Pareto front may be lost. To avoid this problem, we propose a method of defining a virtual overlapping zone between partitions and selecting individuals for mating and migration by evaluating individual populations in this area using weight vectors of adjacent partitions. Using a two-objective constrained knapsack problem for evaluation, we compare the proposed method with standard single-core execution, no-migration parallel MOEA/D, and parallel MOEA/D with standard migration, and show that the proposed method is effective in improving diversity in solution searching while shortening execution time and increasing the accuracy of solution searching.**

Keywords—MOEA/D, parallel and distributed processing, many-core environment, multi-objective evolutionary algorithms

I. Introduction

In recent years, many technologies have been developed to realize "smart cities". The smart city concept integrates information and communication technology, and various physical devices connected to the cloud network to optimize the efficiency of city operations and services and connect to citizens. Therefore, most of real-world problems in the smart city are multimodal interface problems and/or multi-objective optimization problems involving several conflicting objectives.

Cloud systems may offer tens of thousands of virtual machines, terabytes of memories and exabytes of storage capacity. The current trend toward many-core architecture increases the number of cores even more dramatically: we may have more than a million cores to offer extremely massive parallelization. Next, concomitant with advances in modern computational science, the field of evolutionary computing is shifting rapidly to the massive computing era where optimization problems can be characterized by a large number of decision variables, a large number of conflicting objectives, and expensive evaluation functions. As a result, a recent trend in multi-objective evolutionary algorithms is to increase the population size to approximate the Pareto front with high accuracy, and research on parallel computation of multi-objective evolutionary algorithms (MOEAs) has begun [1-8].

However, many parallel speedup methods proposed for decomposition-based MOEAs like MOEA/D [9] are virtual parallelization techniques using multi-thread technology in multi-core processors that have a small number of physical CPU cores. These techniques store all individual information in shared memory and many of these prior studies cannot be guaranteed to improve performance when applied to massively parallel processing using many-core environments such as GPUs [10], supercomputers, and clouds.

In this paper, we propose a method for parallel speedup of MOEA/D. This method presupposes a many-core environment such as GPUs and seeks to prevent degradation in the accuracy of solution searching. For the benchmark problem similar to real-world problems, we use a two-objective knapsack problem with constraints. We evaluate and compare hypervolume (HV) [11] values and execution times of MOEA/D on a single CPU, simple parallel MOEA/D on each partitioned subpopulation, parallel MOEA/D on each partitioned subpopulation with the standard island migration model applied, and parallel MOEA/D on each partitioned subpopulation applying the proposed method to show the method's effectiveness.

II. Background and Related Works

A. Constrained Multi-objective Optimization Problem

Multi-objective optimization is a method that simultaneously optimizes multiple objective functions in a trade-off relationship. A constrained multi-objective optimization problem (CMOP) is defined by the following equations.

$$\begin{cases} Minimize\ f_i(\boldsymbol{x}) & (i = 1,\ 2,\ \ldots,\ m) \\ subject\ to\ g_j(\boldsymbol{x}) \geq 0 & (j = 1,\ 2,\ \ldots,\ k) \end{cases}$$

where f is an objective function vector which consists of m conflicting objective functions and g is a constrained function. The task is to find a set of $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$ minimizing/maximizing m objectives under the constraint functions.

978-1-7281-6929-3/20/$31.00 ©2020 IEEE

Fig. 1 shows a conceptual illustration of the optimization of two objectives. In multi-objective optimization, a currently known solution that has a good evaluation value for any particular objective function is called a non-inferior solution and is regarded as an optimal solution. As shown in Fig. 1, there are usually multiple non-inferior solutions. The set of non-inferior solutions is called the Pareto optimal solution. The surface formed by the Pareto optimal solution is called the Pareto front, and the purpose of a search algorithm such as MOEAs is to conduct searches so that the Pareto front takes a form that better satisfies the criteria of the objective function.

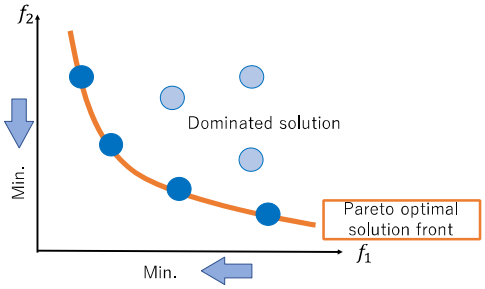


Fig. 1. Conceptual illustration of the objective space for two objectives optimization problems.

B. Conventional Parallel and Distributed MOEA/D

Basically, MOEAs can be categorized into dominance-based and decomposition-based algorithms. NSGA-II [12] and SPEA2 [13] are representative dominance-based algorithms. These techniques are based on the concept of Pareto dominance and the use of some kind of density estimator. NSGA-II uses solution ranking using non-dominated sorting and an estimator based on crowding distance [11], while SPEA2 applies the concept of strength and an estimator based on the distance to the k-nearest neighborhood. On the other hand, decomposition-based algorithm such as MOEA/D decomposes the original MOP into a number of single-objective subproblems constructed in a way such that the optimal solution to each of these subproblems is a point on the optimal Pareto front.

A recent trend in MOEAs has been to increase the population size to approximate the Pareto front with high accuracy [14]. Increasing the population size, however, results in an exponential increase in the computational complexity required for searching the Pareto optimal solutions. As a result, execution time can be a problem when applying such an approach to engineering applications. Research on parallel computation of MOEAs has thus begun. Below, in this paper, we deal with the parallelization of MOEA/D. We also use the Chebyshev (g^{te}) function to be minimized:

$$g^{te}(x,\lambda) = \max_{i\in\{1,\dots,m\}} \lambda^i \left| z_i^* - f_i(x) \right|$$

where x is a feasible solution, $\lambda = (\lambda^1, \dots, \lambda^m)$ is a positive weight vector, and $z^* = (z_1^*, \dots, z_m^*)$ is a reference point.

Several studies on parallel speedup of MOEA/D have already been reported. The simplest implementation method is the master-slave system. For single-objective optimization, different slave processing units(PU)s are assigned to each partitioned subgroup to evaluate individuals. The results of each generation are aggregated in the master PU to select parent individuals to generate next-generation individuals. Because the master-slave system has transfer time overhead incurred when sending data between the master PU and slave PUs, as the degree of parallelism increases the rate of speed improvement declines. This method is thus basically applied to small-scale or medium-scale parallel speedup.

Nebro et al. [2] proposed a parallel MOEA/D algorithm (pMOEA/D) that maintains diversity in solutions. pMOEA/D assigns a different slave PU to each partitioned subgroup constructed from several weight vectors. MOEA/D is executed independently on each subgroup, and individuals are gathered from each subgroup to the master PU according to a probability. Parent individuals are then selected from all the solutions in the master PU. In contrast to the standard master-slave method, pMOEA/D can reduce the number of data transfers. However, the overhead becomes greater in a many-core environment such as GPUs when processing is carried out to aggregate all individual information to the master PU.

Mambrini et al. [4] proposed a parallel decomposition method (PaDe). This algorithm assigns a different slave PU to each subgroup, which has a single-objective problem. MOEA/D is executed independently on each subgroup by evaluating the individuals of each group only with a function that aggregates each subgroup's weights. At a certain interval the worst T solutions of each PU (island) is replaced with the T best solutions from the neighboring islands by migration. This method can also be applied to implementation on GPUs. Such an implementation can bring about further speedup by storing not only all individual information in the microprocessor's shared memory but also individual information of each island in the local memory of streaming multiprocessors (SMs). On the other hand, because the each subgroup's individuals are evaluated by just the function aggregating the weight vectors of each PU, there is a high possibility that the distribution of Pareto optimal solutions between subgroups will become sparse.

Derbel et al. [5] proposed MP-MOEA/D, which brings copies of populations of solutions belonging to neighboring weight vectors to the each weight vector to mate solutions belonging to the weight vector. When implemented on multi-core processors, this algorithm is well-suited for improving the speed of thread-based parallelization, which uses shared memory, and high accuracy of solution-searching can be expected. GPU implementation is also possible. However, because the groups of individuals are stored in global memory, the rate of performance improvement does not increase greatly relative to the increase in the number of CPU cores.

Fig. 2 shows a GPU's block diagram. Unlike a multi-core processor, where all CPU cores access the same shared memory, a GPU has a small local memory within each streaming multiprocessor, which is capable of high-speed transfer, and a large global memory, which is slow. As a result, special care such as considering how to store data must be taken to ensure high-speed performance.

Many of the studies described above are parallelization speedup methods suitable for multi-core processors, which use shared memory. Only one method, PaDe, is suitable for GPUs,

which improve performance by storing individual information in SMs' local memory. However, this method cannot maintain sufficient solution accuracy. In this paper, we propose a method for large-scale parallel MOEA/D that is suitable for many-core architecture like GPUs.

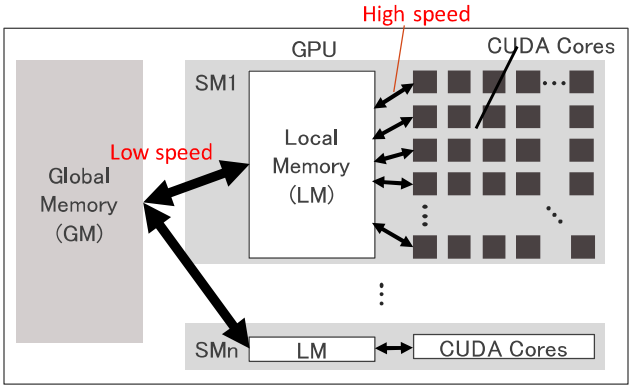


Fig. 2. Conceptual illustration of the objective space for two objectives optimization problems.

III. Proposed Method

A. Difficulties of Parallel MOEA/D on GPU

MOEA/D decomposes the original MOP into a number of single-objective subproblems. Let us denote (λ^1, …, λ^n) be a set of n uniformly distributed weight vectors defining n subproblems. MOEA/D archives a population P = (x^1, …, x^n), each individual x^i (i = 1, …, n) corresponding to a current best solution for one subproblem. Here, subproblems $i \in \{1, \ldots, n\}$ having disjoint T-neighbors $Ne(i)$, *i.e.* T closest weight vectors and corresponding current best solution. At each generation, for every subproblem i, two solutions are selected as parents at random from this T-neighbors $Ne(i)$ to generate an offspring solution x' and if $g^{te}(x', \lambda^i) < g^{te}(x^i, \lambda^i)$ then set $x^i = x'$.

Fig. 3 shows an example of a standard partitioning of the objective space when implementing parallel MOEA/D on a GPU. In this figure, the original multi-objective optimization problem is organized into ten single-objective subproblems using weight vector λ^1 to λ^{10} and assigned to four partitions (P_{1A} to P_{1B}, P_{2A} to P_{2B}, P_{3A} to P_{3B}, P_{4A} to P_{4B},). Here, weight vectors and the individual information assigned to each partition is stored in local memory in a different streaming multiprocessor (SM) on the GPU, and MOEA/D is executed in parallel in each SM. For example, in this figure, weight vectors λ^4, λ^5, individuals x^4, x^5 in the partition 2 are stored in SM2, and weight vectors λ^6, λ^7, individuals x^6, x^7 in the partition 3 are stored in SM3. Since SM consists of several tens of processing cores and local memory, it is possible to execute MOEA/D in parallel using hundred times more physical PUs than multi-core processors. On the other hand, in a weight vectors adjacent to a partition, such as λ^5 or λ^6 in Fig. 3, the individual information of the neighborhood assigned to the adjacent partition cannot be used, and the distribution of the Pareto optimal solution in the vicinity of the partition tends to be sparse. This problem cannot be solved by general island migration models. This is because an elite individual evaluated based on weight vectors belonging to an adjacent partition is not necessarily an elite individual when evaluated using weight vectors belonging to the partition. On the other hand, if individual information assigned to SM3 among individuals belonging to T-neighbors of weight vector λ^5 in partition 2 is moved to SM2 and evaluated, the accuracy of the solution search can be maintained. However, data transfer between SMs needs to be performed via global memory, and since the communication speed is 100 times slower than the communication speed in the SM, sufficient speedup cannot be expected.

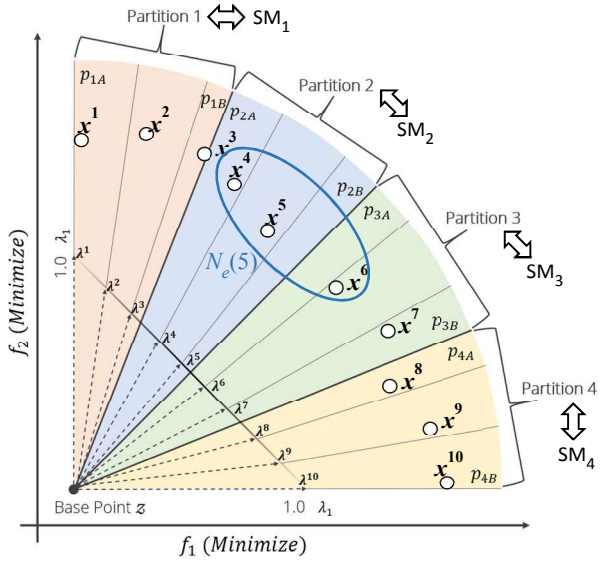


Fig. 3. Standard distribution of solutions in the case of four partitions.

B. Virtual overlap and exclusive evaluation

In order to solve the above problem, we propose a method of defining a virtual overlapping zone between partitions and performing exclusive evaluation on the overlapping zone. The size of the virtual overlapping zone is determined by considering the T-neighbor size. Fig. 4 shows the virtual overlapping zone for partition 3 as an example. In Fig. 4, the physical area of partition 3 is from P_{3A} to P_{3B}, but P_{2BB} and P_{3AA} are defined as the virtual overlapping zone between partition 2 and 3. Here, in addition to all the individual information and weight vector information of partition 3 from P_{3A} to P_{3B}, the weight vector information of overlapping zones P_{3AA} and P_{3BB} is stored in the local memory of SM3. That is, for the individual information, only the information in the partition is stored, and for the weight vectors for the individual evaluation, the information in the adjacent virtual overlapping zone is also stored in the SM.

Let's x_j^i, belong to T closest weight vectors $Ne(j)$ in the virtual overlapping zone, be a current best solution evaluated using the weight vector λ^i. In virtual overlapping zone, weight vector λ^i archives the current best solution x^i and x_j^i. Here, x^i is archived in the SM that stores λ^i, and x_j^i is archived in the adjacent SM that stores λ^j. Fig. 5 shows an image diagram of sharing a weight vectors near the boundary between two adjacent partitions in the virtual overlap region when T-neighbor size is 2. Weight vectors λ^1 to λ^5, individuals x^1 to x^4 and x_4^5 are stored in SM2 and similarly weight vectors λ^4 to λ^9, individuals x^5 to x^9 and x_5^4 are stored in SM3 to execute MOEA/D in parallel. Then, at an appropriate interval, migration is performed

to copy x_4^5 belonging to SM2 to SM3 and x_5^4 belonging to SM3 to SM2. In other words, with the partition boundaries as the boundary, selecting individuals for migration by evaluating individual populations in this area using weight vectors of adjacent partitions.

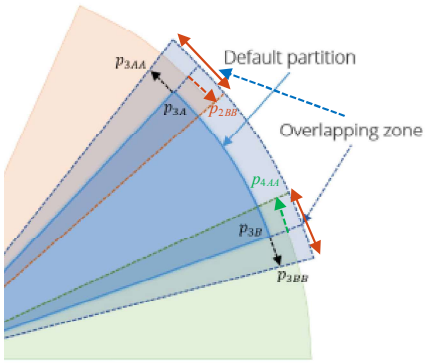


Fig. 4. Depiction of how partitions overlap with each other to create folds.

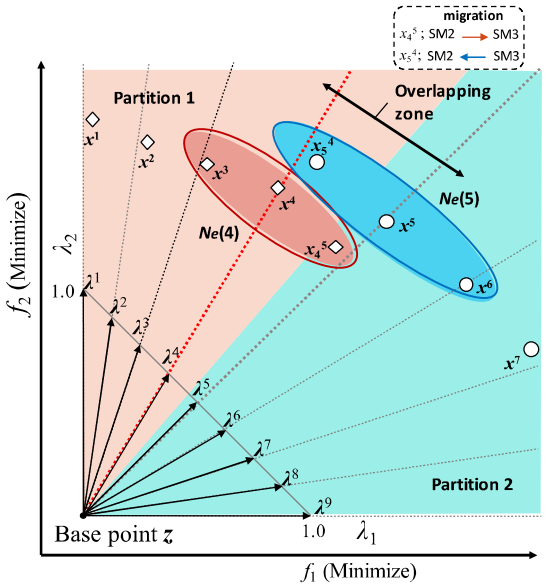


Fig. 5. Exclusively evaluated solutions inside a fold created between partitions.

The pseudo code to be executed by each SM is summarized in Algorithm 1. Each SM executes in parallel some sub-problem with respect to weight vectors which is obtained by decomposition of the standard MOEA/D. Basically, in standard MOEA/D on single-CPU, at each generation, for every sub-problem i, x^i and another solution is selected as parents at random from this T-neighbors $Ne(i)$ to generate an offspring solution x'. On the other hand, in a case of distributed MOEA/D on GPU, in a weight vectors adjacent to a partition, the individual information of the neighborhood assigned to the adjacent partition cannot be used. To cope with this problem, another individual for generating an offspring solution x' are selected from the combination of T-neighbors $Ne(i)^*$ belonging to the partition and elite individuals x_l^j (line 10) Next, if $g^{te}(x', \lambda^i) < g^{te}(x^i, \lambda^i)$ then set $x^i = x'$, and if offspring solution x' satisfies $g^{te}(x', \lambda^j) < g^{te}(x_l^j, \lambda^j)$ then set $x_l^j = x'$ (line 22). Here, the random number generated in the overlapping zone is used as the initial value of x_l^j.

Note that the algorithm proposed here can also be applied to implement parallel speed-up of MOEA/D on multi-core processors which use shared memory. All of individuals and weight vectors in each partition are stored in shared memory, and evaluations of individuals belonging to each weight vector in a partition are processed in parallel using multi-threading technology.

Algorithm 1: Pseudo code to be executed by each SM_k, $\boldsymbol{k} \in \{\mathbf{1}, \ldots, \boldsymbol{K}\}$, K: the number of SM

Input: $Ne(t)^*$: neighboring sub-problems belong to the partition k;
λ^i for every $\boldsymbol{i} \in Ne(t)^*$: neighbors' weight vectors;
x_j^i, belong to T closest weight vectors $Ne(j)$ in the virtual overlapping zone, be a current best solution evaluated using the weight vector λ^i; // Input data by migration

Output: x_l^j; // Output data by migration

```
1  INITIALIZE x^i for every i ∈Ne(t)^*, z*;
2  Until STOPPING CONDITION do
3      for i ∈Ne(t)^* do
4          y^i ← x^i;
5      end
       // migration: Input x_j^i from adjacent SM
6      for i ∈ Overlapping Zone do
7          if g^te(x_j^i, λ^i) < g^te(x^i, λ^i) then set x^i = x_j^i;
8      end
9      Repeat t_interval times: //
10         l ← rand(Ne(t)^* ∪ x_j^i);
11         y ← Crossover_Mutation_Repair (x^t, x^l);
12         for m ∈ {1, …, M} do
13             if z_m^* < f_m(y) then z_m^* ← f_m(y);
14         end
15         if g(y, λ^t) < g(x^t, λ^t) then
16             x^t ← y;
17         end
18         for i ∈Ne(t)^* do
19             if g^te(y, λ^i) < g^te(y^i, λ^i) then set y^i = y;
20         end
21         for j ∈ Overlapping Zone do
22             if g^te(y, λ^j) < g^te(x_l^j, λ^j) then set x_l^j = y;
23         end
           // migration: x_l^j → adjacent SM
24 end
```

IV. Evaluation

A. Experimental method

To verify the accuracy of solution searching when the proposed method is applied, a feasibility study was carried out using multi-core processors. A CPU core was assigned to each

partition, and each core independently held individual information. A function was provided to report solutions in partial overlapping zones by communicating between tasks.

Using the constrained knapsack problem described below, two-objective optimization problems were evaluated. Here, considering that using these problems for evaluation takes time and that a general island model has many design variables, we conduct a comparison evaluation targeting two items — hypervolume and execution time — for the cases of executing standard MOEA/D on a single CPU, no-migration parallel MOEA/D, parallel MOEA/D with standard migration and parallel execution of MOEA/D using the proposed method.

As constrained multi-objective optimization problems, we focus on *mk*-KPs [15]. The *mk*-KPs are defined in as follows.

$$\begin{cases} Maximize\ f_j(x) = \sum_{i=1}^{n} p_{ij} \times x_j \ (j = 1,2,\dots,m) \\ Subject\ to \sum_{i=1}^{n} w_{il} \times x_i \le c_l \ (l = 1,2,\dots,k) \end{cases} \quad (1)$$

The problem has n items and k knapsacks, and each item i has m profits p_{ij} (j = 1,2,...,m) and k weights w_{il} (l = 1,2,...,k). The task is to find a set of items $x = x_1, x_2, \cdots, x_n \in \{0,1\}^n$ maximizing m objectives while not exceeding k knapsack capacities c_l. The knapsack capacity c_l is defined as follows.

$$c_l = \varphi \sum_{i=1}^{n} w_{il} \ (l = 1,2,\dots,k) \quad (2)$$

φ is the feasibility ratio for each knapsack (constraint), and we can control the strictness of the constraints by varying φ. The *mk*-KP problem is different from the multi-objective knapsack problem (MOKP) [16] in that the numbers of objectives m and knapsacks k can be independently determined.

Parameters used in the experiment are listed in Table I and the test execution environment is summarized in Table II. Experimental results were taken to be the median of 31 trials.

TABLE I. EXPERIMENTAL PARAMETERS FOR KNAPSACK PROBLEM

Population Size	200
Degree of Parallelism	4, 8, 16 20
Number of Generations	5000
Neighborhood Size	5
Number of Folding Weight Vectors	5
Migration Interval	500 generations
Migration Size	5
Number of Objectives	2
Constraints	2
Crossover Rate	1.0
Mutation Rate	0.05
Number of Items	500
Elimination Rate	0.5

TABLE II. TEST EXECUTION ENVIRONMENT

CPU	Intel(R) Core(TM) i9-7920X CPU (Skylake-X, 12 cores, 24 threads, 2.90 GHz)
Memory	32GB DDR4 SDRAM
OS	Microsoft Windows™10 Pro (64-bit)
Compiler	Visual C++ 12.0

B. Experimental results and discussion

a) Examination of solution searching accuracy of no-migration parallel MOEA/D: Fig. 6 shows the Pareto fronts after MOEA/D was executed for 5,000 generations with the total number of individuals fixed at 200 for the cases of single-core execution (no parallelization) and no-migration parallel MOEA/D executed on four CPU cores (4 cores x 50 individuals), eight cores (8 cores x 25 individuals), 16 cores (16 cores x 12 individuals or 13 individuals), and 20 cores (20 cores x 10 individuals).

From Fig. 6, we see that by parallelizing MOEA/D, diversity at both edges of the Pareto front increases. On the other hand, even though the degree of parallelism increased, convergence to the Pareto front (in the same generation) is reduced, and the solution distribution near partition boundaries became sparse. The reason may be that some of the individuals in the vicinity of the weight vector T near the partition boundary are assigned to the adjacent partition and cannot be used to select parents.

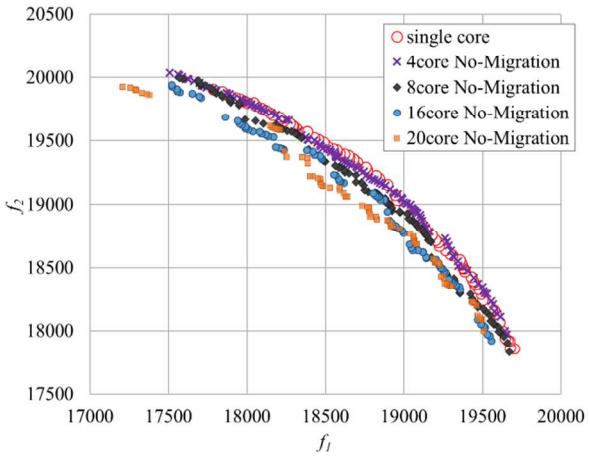


Fig. 6. Relationship between the degree of parallelism of no-migration parallel MOEA/D and solution searching accuracy.

b) Differences in the impact of different migration method: Next, to investigate the effects of migration due to parallelization in which 200 individuals were divided among 16 cores, the following cases of solution searching were carried out and their performance compared: obtaining Pareto solutions by each core without migration (16-core No-Migration); classical island model method, which like the proposed method sends the best solutions in each group to the neighboring group at the same migration interval (16-core Classical Migration); the proposed method (16-core Partially Overlapping); and, as reference, the conventional method using a single core (Single Core). The parameters of the proposed method were Number of

Generations, Number of Folding Weight Vectors, Migration Interval, and Migration Size as given in Table I.

Fig. 7 shows the Pareto fronts after MOEA/D was executed for 5,000 generations. We see that when compared to execution by a single core, parallelization reduces convergence to the Pareto front. On the other hand, with parallelization the solution distribution widens at both edges of the Pareto front (increased diversity). Also, when the standard island migration model was applied, while it is believed that there would be compatibility with the optimization problem, compared with no-migration both convergence to the Pareto front and the diversity of the solution distribution are reduced. The reason may be that even with the migration to adjacent partitions of elite individuals evaluated on the basis of weight vectors in the relevant partitions, this approach may not work effectively when individuals are evaluated with weight vectors in the adjacent partitions. On the other hand, when migration was carried out using the proposed virtual overlap approach and exclusive evaluation, convergence to the Pareto front and diversity of the solution distribution both improved compared with no-migration. In addition, the issue of the solution distribution being dispersed near partition boundaries due to parallelization is alleviated.

Next, the HV values and execution times under the same testing conditions were compared, as shown in Fig. 8. Here, when individuals are simply divided and parallel executions are carried out, the execution time improves 6.8-fold. However, solution searching accuracy falls slightly. As with the conventional island model, simply passing the best solutions within a group to neighboring groups at a constant time interval reduces diversity at both edges of the Pareto front and the HV value. In contrast, with the proposed overlap method the number of individuals searched by each core for solutions increases slightly. In addition, the time to process the migration increases. As a result, while execution time increased slightly, improvements in both execution time and solution searching accuracy could be achieved due to the effect of the proposed migration method.

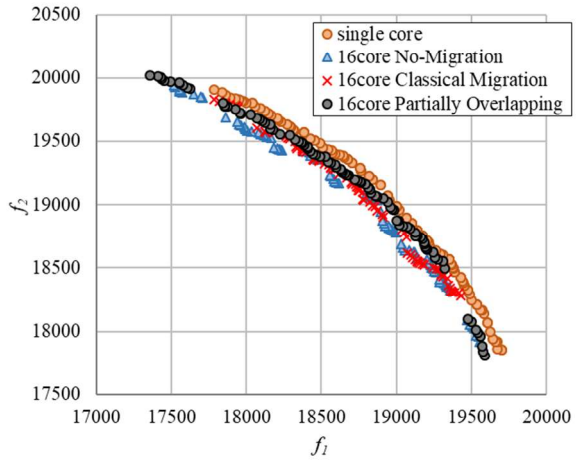


Fig. 7. Migration methods and shapes of Pareto fronts.

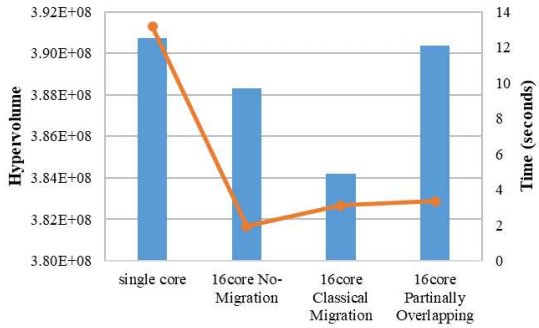


Fig. 8. Comparison of solution searching accuracy and execution time for 5,000 generations due to differences in migration.

c) Relationship between degree of parallelism and solution searching accuracy for same generation: From the above experiments, we see that the proposed method improves solution searching accuracy and diversity of solution distribution more than no-migration parallelization and the standard island migration model do. In addition, the problem of the solution distribution's becoming sparse near the partition boundaries due to parallelization tends to improve. We thus examined the relationship between solution searching accuracy (HV value) and execution time for the same number of generations by the proposed overlapping method when the total number of individuals is kept constant and the degree of parallelism (number of partitions) is changed. The number of generations was set at 5,000 and the other parameters of the overlap method were Number of Generations, Number of Folding Weight Vectors, Migration Interval, and Migration Size, as given in Table I.

Fig. 9 shows the relationship between solution searching accuracy (HV value) obtained after 5,000 generations and the required execution time while changing the degree of parallelism. Compared with a single CPU, execution time on 16 cores is improved a maximum of 4.3-fold. On the other hand, overall solution searching accuracy falls gradually as the degree of parallelism increases.

Fig. 10 shows a comparison of the shapes of the Pareto fronts obtained after 5,000 generations by four cores, 16 cores and a single CPU. It shows that compared with four cores, the solution distribution obtained by 16 cores became sparse near the partition boundaries.

Fig. 11 shows a comparison of HV value for 16 cores when under the same conditions while trying a combination of setting the migration interval to 100, 500 and 1000, and Fig. 12 shows a comparison of the shapes of the Pareto fronts for 16 cores, when increasing the total number of individuals to 400. The results show that reducing the migration intervals do not necessarily alleviates the problem of the solution distribution's becoming sparse to a certain extent, but show that there is an appropriate migration interval. However, increasing the total number of individuals and the number of individuals per core greatly ameliorates this problem. This is because the number of individuals was fixed at 200 and divided up among the cores, the number of individuals per core decreased too much. The solution distribution thus became sparse as the degree of

parallelism increases. Therefore, it is considered that this problem can be improved by increasing the number of individuals per core.

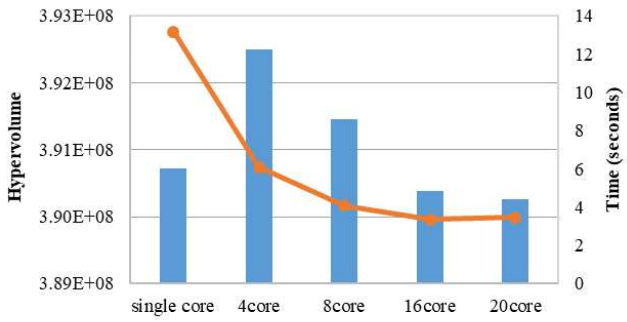


Fig. 9. Results of evaluation after 5,000 generations while changing degree of parallelism.

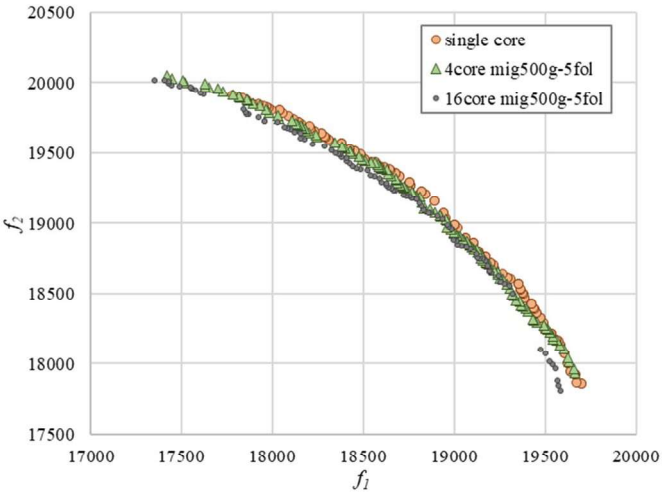


Fig. 10. Comparison between the non-dominated solutions with 4 cores, 16-core Virtual Overlap MOEA/D Method and single-core MOEA/D.

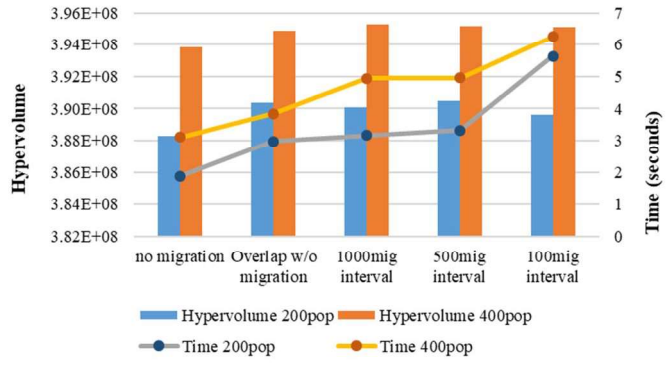
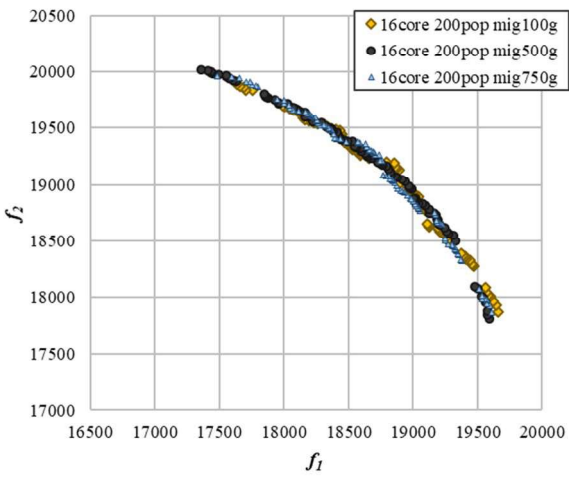
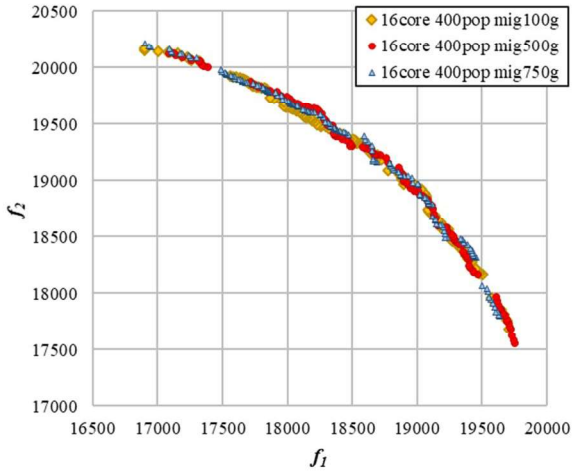


Fig. 11. Comparison of solution searching accuracy and execution time due to differences in migration interval with 16 cores.



(a) Pareto front for 200 populations



(b) Pareto front for 400 populations

Fig. 12. Comparison of the shapes of the Pareto fronts for 16 cores, due to the difference in population size.

d) Relationship between degree of parallelism and solution searching accuracy for same execution time: When considering practical applications, comparison on the basis of execution time is often more important than the number of generations. We thus investigated solution searching accuracy (HV value) by using the proposed method when the degree of parallelism of the CPU cores is changed (change in number of group partitions) while maintaining the total number of individuals at 200 and keeping the execution time fixed. For the evaluation time, 1 second, the time to evaluate 1,600 generations by 16 cores, was used. The parameters of the overlap method were Number of Folding Weight Vectors, Migration Interval, and Migration Size as given in Table I.

Fig. 13 shows the relationship between degree of parallelism and HV value when execution time is fixed. Evaluation time of one generation was reduced by increasing the degree of parallelism. As a result, when time is fixed, greater HV values are obtained as the number of executed generations increase in line with the increase in the number of cores.

Fig. 14 shows a plot of non-dominated solutions when eight cores and 16 cores are used. By increasing the number of cores, diversity at both edges of the solution distribution increases. This shows that raising the number of cores is effective at improving the HV value.

The results above show that the proposed method has both the effect of ameliorating the problem of the solution distribution near the partitioned boundaries becoming sparse and improving diversity near both edges of the Pareto front when MOEA/D is parallelized in a many-core environment like GPUs. In addition, the proposed method is effective in reducing execution time. However, our testing here is a feasibility study evaluating the use of multi-core processors. Going forward, it is necessary to evaluate using a many-core environment like GPUs. Also necessary are an evaluation using multiple types of test problems and an evaluation that increases the number of objectives.

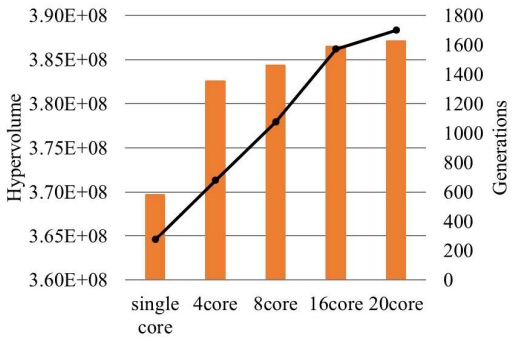


Fig. 13. HV values when execution time is fixed and the degree of parallelism is changed.

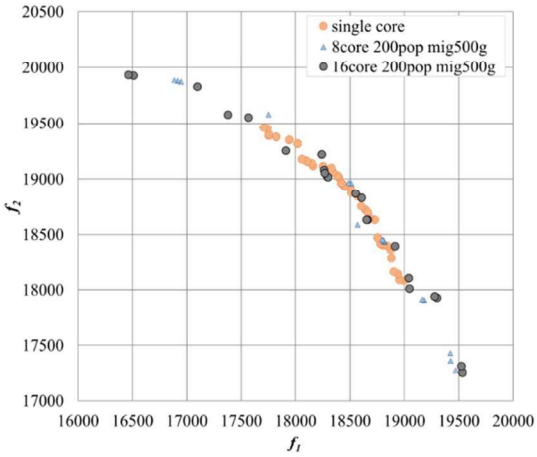


Fig. 14. Comparison between the non-dominated solutions with 8, 16 cores Virtual Overlapping Method of MOEA/D and single core MOEA/D.

V. Conclusion

This paper proposed a method for many-core-based large-scale parallel and distributed computation of MOEA/D. More specifically, we have proposed the method of defining virtual overlapping zones between partitions and selecting individuals for mating and migration by evaluating individual populations in these areas using weight vectors of adjacent partitions. By using a two-objective constrained knapsack problem for evaluation, we compared the proposed method with standard single core execution, no-migration parallel MOEA/D, parallel MOEA/D with standard migration, and showed that the proposed method is effective in improving diversity in solution searching around partition boundaries and extreme non-dominated solutions.

Acknowledgment

The authors would like to thank Prof. Toshio Hirotsu from Hosei University for their variable discussion about experimental results. This work was supported by JSPS KAKENHI Grant Numbers JP19K12162.

References

[1] Juan J. Durillo, Antonio J. Nebro, F. Luna and E. Alba, “A study of master-slave approaches to parallelize NSGA-II,” in Proceedings of the 2008 IEEE International Symposium on Parallel and Distributed Processing, 2008, pp. 1-8.

[2] A. J. Nebro and J. J. Durillo, “A Study of the Parallelization of the Multi-Objective Metaheuristic MOEA/D,” in International Conference on Learning and Inteligent Optimization (LION4), 2010, pp. 303–317.

[3] Hai-Lin Liu, Fangqing Gu, and Qingfu Zhan, “Decomposition of a Multiobjective Optimization Problem into a Number of Simple Multiobjective Subproblems,” IEEE Transactions on Evolutionary Computation, vol. 18, no. 3, pp. 450 – 455, June 2014.

[4] A. Mambrini and D. Izzo, “PaDe: A parallel algorithm based on the MOEA/D framework and the island model,” in Parallel Problem Solving from Nature (PPSN XIII), 2014, pp. 711–720.

[5] B. Derbel, A. Liefooghe, G. Marquet and E. Talbi, "A fine-grained message passing MOEA/D," in Proceedings. of 2015 IEEE Congress on Evolutionary Computation (CEC), 2015, pp. 1837–1844.

[6] F. Luna and E. Alba, “Parallel Multiobjective Evolutionary Algorithms,” Springer Handbook of Computational Intelligence, 2015, pp. 1017–1031.

[7] Y. Sato, M. Sato, and M. Miyakawa, “Distributed NSGA-II using the divide-and-conquer method and migration for compensation on many-core processors,” in 2017 21st Asia Pacific Symposium on Intelligent and Evolutionary Systems (IES), pp. 83–88, 2017.

[8] Y. Sato, M. Sato, and M. Miyakawa, “Distributed NSGA-II Sharing Extreme Non-dominated Solutions for Improving Accuracy and Achieving Speed-up,” in Proceedings. of 2019 IEEE Congress on Evolutionary Computation (CEC), 2019, pp. 3087-3094.

[9] Q. Zhang, H. Li., "MOEA/D: A Multiobjective Evolutionary Algorithm Based on Decomposition" in IEEE Transactions on Evolutionary Computation, Volume 11, Issue 6, 2007, pp. 712-731.

[10] Best Practice Guide – GPGPU. http://www.prace-ri.eu/IMG/pdf/Best-Practice-Guide-GPGPU-1.pdf#search=%27GPGPU+guide%27 (Cited on Jan. 23, 2020)

[11] K. Deb, Multi-Objective Optimization using Evolutionary Algorithms. Wiley; 1 edition, March 2, 2009.

[12] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, “A fast and elitist multiobjective genetic algorithm: Nsga-ii,” *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, Apr 2002.

[13] E. Zitzler, M. Laumanns, and L. Thiele, “SPEA2: Improving the strength pareto evolutionary algorithm for multiobjective optimization,” in Proceedings of the EURO-GEN 2001 Conference, 2001.

[14] H. Ishibuchi, N. Akedo, and Y. Nojima, “Behavior of multiobjective evolutionary algorithms on many-objective knapsack problems,” *IEEE Transactions on Evolutionary Computation*, vol. 19, no. 2, pp. 264–283, April 2015.

[15] D. P. Hans Kellerer, Ulrich Pferschy, Knapsack Problems. Springer, 2004.

[16] E. Zitzler, Evolutionary Algorithms for Multiobjective Optimization: Methods and Applications. PhD thesis, Swiss Federal Institute of Technology, Zurich, 1999.