Hindawi
Mathematical Problems in Engineering
Volume 2019, Article ID 1713636, 13 pages
https://doi.org/10.1155/2019/1713636

Research Article

A Dual Heterogeneous Island Genetic Algorithm for Solving Large Size Flexible Flow Shop Scheduling Problems on Hybrid Multicore CPU and GPU Platforms

Jia Luo and Didier El Baz

LAAS-CNRS, Université de Toulouse, CNRS, Toulouse, France

Correspondence should be addressed to Jia Luo; 824629061@qq.com

Received 7 November 2018; Revised 16 January 2019; Accepted 19 February 2019; Published 13 March 2019

Academic Editor: Thomas Hanne

The flexible flow shop scheduling problem is an NP-hard problem and it requires significant resolution time to find optimal or even adequate solutions when dealing with large size instances. Thus, this paper proposes a dual island genetic algorithm consisting of a parallel cellular model and a parallel pseudo-model. This is a two-level parallelization highly consistent with the underlying architectures and is well suited for parallelizing inside or between GPUs and a multicore CPU. At the higher level, the efficiency of the island GA is improved by exploring new regions within the search space utilizing different methods. In the meantime, the cellular model keeps the population diversity by decentralization and the pseudo-model enhances the search ability by the complementary parent strategy at the lower level. To encourage the information sharing between islands, a penetration inspired migration policy is designed which sets the topology, the rate, the interval, and the strategy adaptively. Finally, the proposed method is tested on some large size flexible flow shop scheduling instances in comparison with other parallel algorithms. The computational results show that it not only can obtain competitive results but also reduces execution time.

1. Introduction

The flexible flow shop (FFS) scheduling problem focuses on improving machine utilization and reducing makespan. Some works on solving small size FFS are concerned on exact methods [1, 2] to find the optimal solution. However, conventional optimization techniques always fail in industry application as the problem sizes in the real world are much bigger and the computational cost is increased. Therefore, there is a growing interest in developing heuristic methods to solve large complex FFS problems [3, 4]. Although these approaches cannot guarantee finding the optimal solution, there is a sizable probability that an adequate solution is found in a reasonable time.

The genetic algorithm (GA) is one of the most widely known heuristic methods and is one of the best approaches in solving FFS problems. But when GAs are applied to large or complex problems, there is a conflict between searching better solutions and decreasing execution time. In contrast to classical GAs, the island GA [5] divides the population into a few relatively large subpopulations. Each of them works as an island and is free to converge towards its own suboptimum. At some points, a migration operator is utilized to exchange individuals among islands. This design imitates the nature in a better way which increases the search diversification [6]. Furthermore, it is one of the most famous models to exploit parallelism in GAs. Nevertheless, due to the finite island size and the same genetic operator configuration in each island, island GAs are apt to yield premature convergence [7]. Meanwhile, this design has to be carried out with high respect to the underlying architectures for parallelization implementations.

With the unprecedented evolution of GPUs and multicore CPUs, almost all modern computers are equipped with both. Some comparisons between their performances for GA applications were discussed [8], but the cooperation between the two in this domain was rarely concerned. These facts have motivated the design of a heterogeneous island GA that keeps better population diversity and is well suited for parallelization on GPUs and a multicore CPU. In this paper, we seek to address it and its application to solve large size FFS problems. Specially, the contributions of our work are summarized as follows:

(1) A dual heterogeneous island model is proposed where the 2D variable space of the cellular GA and the complementary parent strategy of the pseudo-GA keep the population diversity.

(2) A two-level parallelization highly consistent with the underlying architecture is implemented which is well suited for parallelizing inside or between GPUs and a multicore CPU.

(3) A penetration inspired migration policy is designed so that it can share good individuals effectively by setting the topology, the rate, the interval and the strategy adaptively.

The remaining sections of this paper are organized as follows. Section 2 introduces related works. Section 3 describes the research problem. Section 4 presents the design of the dual heterogeneous island GA on hybrid multicore CPU and GPU platforms. Section 5 displays the numerical experiments and the analysis of the results. Finally, Section 6 states the conclusions.

2. Related Works

When the population size is p and there are q islands, only p/q individuals work with GA operators in one island. Extremely, each island may have only 2 individuals [9, 10]. The selection and the elitist strategy in GAs decrease the subpopulation diversity in one island after several generations. Although the migration at some points can help to create new individuals, this influence is restricted. What is worse, an inappropriate implementation of migration mechanism may cause genetic drift and leads to converging towards a local optimum. One approach for dealing with this problem is the heterogeneous island GA which makes distinction among subpopulations by different configurations. Herrera et al. [11] presented the gradual distributed real-coded GA that applied different crossover operators to different subpopulations. Alba et al. [12] encompassed the actual parallelization of the gradual distributed real-coded GA on a cluster of 8 homogeneous PCs. In [13], Miki et al. designed a parallel GA using nCUBE-2E where different islands had different parameter settings. Although these heterogeneous algorithms have improved the solutions' quality, their implementations are usually executed on homogeneous architectures or even on a monoprocessor. In these cases, different islands can work in parallel but GA operations inside one island are executed in a sequential way. In addition to proposed heterogeneous island GAs, some works were carried out to evaluate the performance of heterogeneous computing architectures for island GAs. In [14], a homogenous island GA was run at the same time on different types of machines which obtained superlinear speedup. García-Sánchez et al. [15] studied benefits from setting the subpopulation sizes according to each heterogeneous node's computational power. García-Valdez et al. [16] tested the randomized parameter setting strategy for heterogeneous workers in pool-based GAs. Despite promising results from leveraging computational capabilities of a heterogeneous cluster, these methods must face some common challenges such as lost connections, low bandwidth, abandoned work, security, and privacy. Moreover, the above-mentioned designs generally are hard to profit the computation capability from GPUs or heterogeneous environment mixed with multicore processors and many-core processors.

In the last decade, Graphics Processing Units (GPUs) have gained widespread popularity as computing accelerators for computational intelligence. Langdon [17] surveyed genetic programming use with GPU and showed the fastest genetic programming is based on an interpreter rather than compilation. Krömer et al. [18] provided a brief overview of the latest research on the design, implementation, and applications of parallel nature-inspired metaheuristics-based methods on the GPUs. In [19], a systolic genetic search was designed to explicitly exploit the high degree of parallelism available in GPU architectures while a cellular evolutionary algorithm framework implemented on GPUs [20] was presented by Soca et al. Since the cooperation between GPUs and a multicore CPU is stable and secure, some efforts have considered to utilize both and enjoy their compute capabilities maximally. Dabah et al. [21] proposed 5 accelerated branch and bound algorithms for solving the blocking job shop scheduling problem where two of them presented hybridization between the multicore CPU approach and the GPUs-based parallelization approach. Benner et al. [22] discussed a hybrid Lyapunov solver where the intensive parts of the computation were accelerated using GPUs while executing the remaining operations on a multicore CPU. In [23], Bilel et al. introduced a CPU-GPU cosimulation framework where synchronization and experiment design were performed on CPU and node's processes were executed in parallel on GPUs. These studies have confirmed the interest to design a scheme that exploits GPUs and a multicore CPU in efficient ways. However, simultaneous parallelization on two sides and its implementation for island GAs are not yet concerned.

Several researches have tried island GAs to solve shop scheduling problems. On one hand, some works consider the improvement for solutions' quality. Kurdi [24] studied an island GA to solve job shop scheduling problems where three islands worked with different mutation operators and worst individuals adapting to their environment migrated first. In [25], Defersha et al. considered an island GA with a k-way tournament selection, five kinds of crossover, and six kinds of mutation applied by different probabilities for a flexible job shop scheduling problem with lot streaming. On the other hand, the rest applications imposed parallelization to island GAs and analyzed the speedup. Zajicek et al. [26] proposed an accelerated island GA for solving a flow shop scheduling where all computations were carried out on GPUs. Huang et al. discussed a flow shop scheduling problem with fuzzy processing times and fuzzy due dates in [27] and its implementation on CUDA. But none of them have so far, to the best

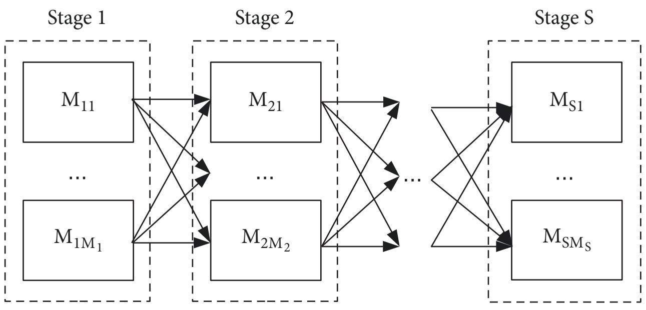

Figure 1: A flexible flow shop layout.

of our knowledge, tried to solve shop scheduling problems by heterogeneous island GAs parallelized on GPUs and a multicore CPU. All the above-mentioned efforts provide us with a starting point for designing a dual heterogeneous island GA that keeps a better population diversity and that is well suited for parallelization on hybrid multicore CPU and GPU platforms. This implementation will help to find a good solution for large size FFS problems in the real world within a short response time.

3. Problem Definition

The FFS is a multistage production process as illustrated in Figure 1. An instance of the FFS problem considers a set of J jobs ($1 \leq j \leq J$). Each of them consists of a set of S stages ($2 \leq s \leq S$). At every stage, there is a set of M_s machines ($1 \leq m \leq M_s$) and at least one stage has more than one machine. All jobs need to go through all stages in the same order and only one machine is selected for processing on each stage. There is no precedence between operations of different jobs, but there is precedence among operations due to the jobs' processing cycles. Preemptive operations are not allowed. A feasible solution is described by jobs' sequence on target machines M_{js}. The processing time of job j at stage s on machine m is abbreviated as P_{jsm}. Usually, it is known with the release time R_j and the due time D_j. The objective function to minimize the total tardiness and the makespan is represented by $WT * \sum T_j + C_{max}$ using the classification scheme of Bruzzone et al. [28], where WT indicates the priority of the first objective. As a minimization problem, the fitness function of an individual is transferred from the objective function by $\max(E_{max} - (WT * \sum T_j + C_{max}, 0)$, where E_{max} is the estimated maximum value of the objective function. The FFS problem is NP-hard in essence [29]. When dealing with large size instances, it requires huge resolution time to find optimal or even adequate solutions.

4. Dual Heterogeneous Island Genetic Algorithm on Hybrid Multicore CPU and GPU Platforms

4.1. Dual Heterogeneous Island Strategy. The general framework of the proposed dual heterogeneous island strategy is shown in Figure 2. There is the same number of individuals on each island where island A works with the cellular GA [30] and island B works with the pseudo-GA [31]. As two islands are exploring new regions within the search space utilizing different methods, it helps enlarge the scope of the search process and increase the chances of avoiding premature convergence. Moreover, individuals from heterogeneous islands have obtained different characters during the independent evolution procedure. In this case, the benefit of migration is increased. At the software level, three sublevels are considered according to the source of the heterogeneity:

(i) Genotype level: as a feasible solution is described by jobs' sequence on target machines, the chromosome is displayed by a string of length $J \times S$ and is indexed from 0 to $J \times S - 1$. The i-th gene states the index of the target machine for job $\lfloor i/S \rfloor + 1$ at stage $\{i/S\} + 1$ and each gene has two layers. The upper layer is designed for the cellular GA where the i-th gene is presented by an integer number. At the lower layer, the i-th gene is expressed by binary numbers to work with the complementary parent strategy of the pseudo-GA.

(ii) Operator level: the cellular GA starts with random initialization and maps individuals on a 2D grid. An individual is limited to communicate with individuals from the nearby area and use them for crossover and mutation. The neighborhoods overlapping makes good solutions disseminate through the entire population. This design allows a better exploration of the search space with respect to decentralization. The pseudo-GA has a similar process with the standard GA, but implements no selection and no mutation. The dynamic complementary strategy [31] initializes all bits of one parent randomly with a binary value of either 0 or 1. Meanwhile, the bit located at the same position of the other parent is assigned with the opposite binary value. The crossover is executed between the fixed pair of individuals to make their offspring always complementary to each other. In this case, the search ability is enhanced since higher population diversity is got without gene loss and the maximum Hamming distance is kept all long.

(iii) Parameter level: the execution of the crossover operator and the mutation operator are determined by the crossover rate and the mutation rate. Their values for the cellular GA and the pseudo-GA on different islands are set differently.

4.2. Parallelization on Hybrid Multicore CPU and GPU Platforms. As far as the hardware level is concerned, there are two obvious advantages to parallelize the dual heterogeneous island GA on hybrid multicore CPU and GPU platforms:

(i) Widespread HPC resources: nowadays, almost all modern computers are equipped with GPUs and a multicore CPU. The cooperation between them is through their inner connections which is stable and secure. With the development of CUDA [32], it is convenient to use enabled GPUs for general

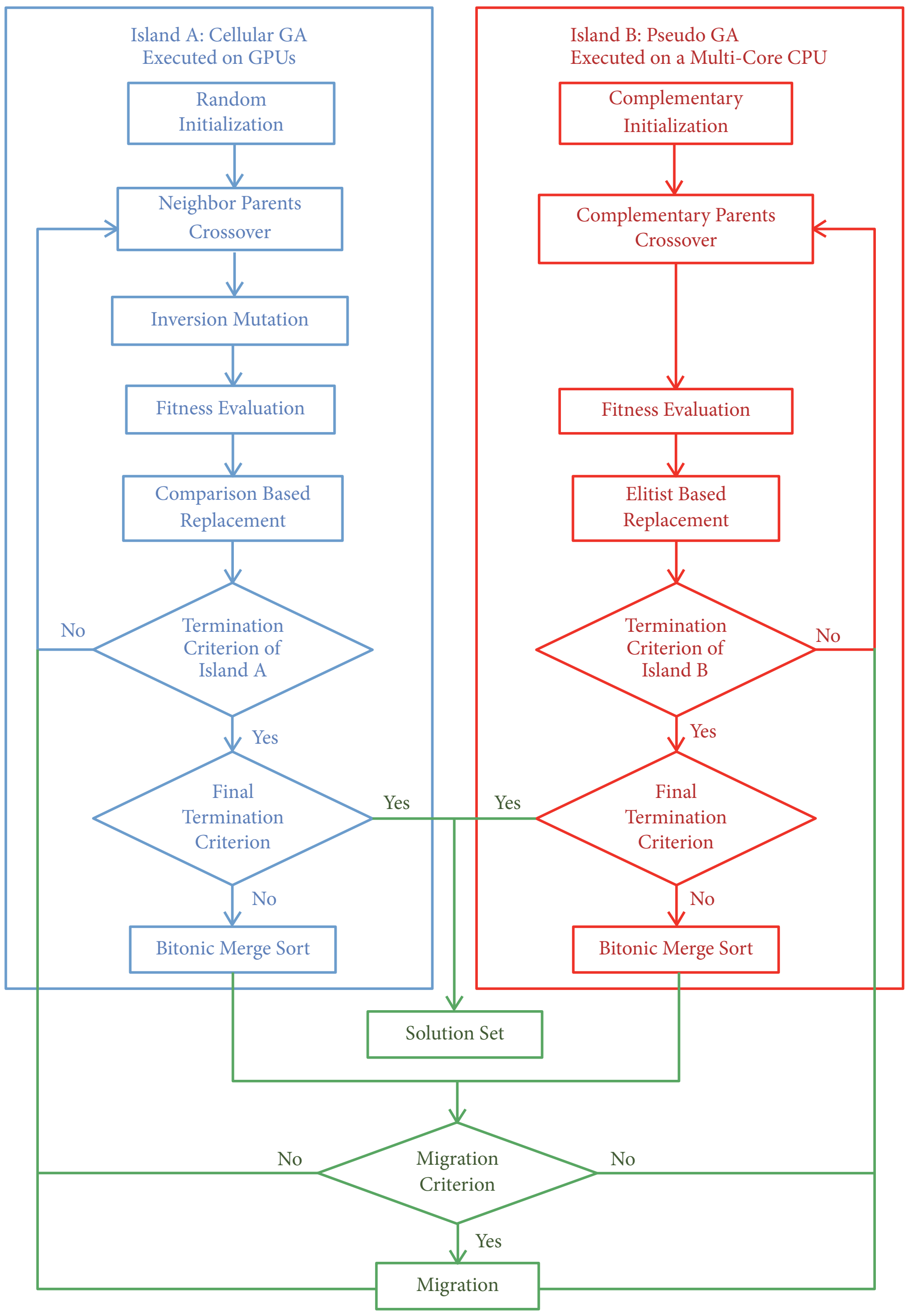
Island A: Cellular GA
Executed on GPUs
Random Initialization
Neighbor Parents Crossover
Inversion Mutation
Fitness Evaluation
Comparison Based Replacement
Termination Criterion of Island A
No
Yes
Final Termination Criterion
Yes
No
Bitonic Merge Sort
Island B: Pseudo GA
Executed on a Multi-Core CPU
Complementary Initialization
Complementary Parents Crossover
Fitness Evaluation
Elitist Based Replacement
Termination Criterion of Island B
No
Yes
Final Termination Criterion
Yes
No
Bitonic Merge Sort
Solution Set
Migration Criterion
No
No
Yes
Migration

FIGURE 2: The general framework of the dual heterogeneous island GA.

purpose processing. On the other hand, concurrency platforms allowing the coordination of multicore resources facilitate programming on multicore CPUs. Moreover, in addition to the parallelization on GPUs or on a multicore CPU at the lower level, the GPUs and the multicore CPU can work concurrently at the higher level to maximally use computing resources.

(ii) High consistency with the proposed GA: the cellular GA maps individuals on a 2D grid and GA operations are designed with respect to this structure. The CUDA threads are grouped into 2D blocks that are organized in a 2D grid [33]. Thus, the cellular GA can be entirely parallelized on GPUs with the absolutely matching architectures. On the other hand, only the crossover, the fitness evaluation, and the replacement are kept in the pseudo-GA. The crossover is performed between fixed complementary parents. The fitness evaluations of individuals are independent. Since no global information is required, all four loops in the above two

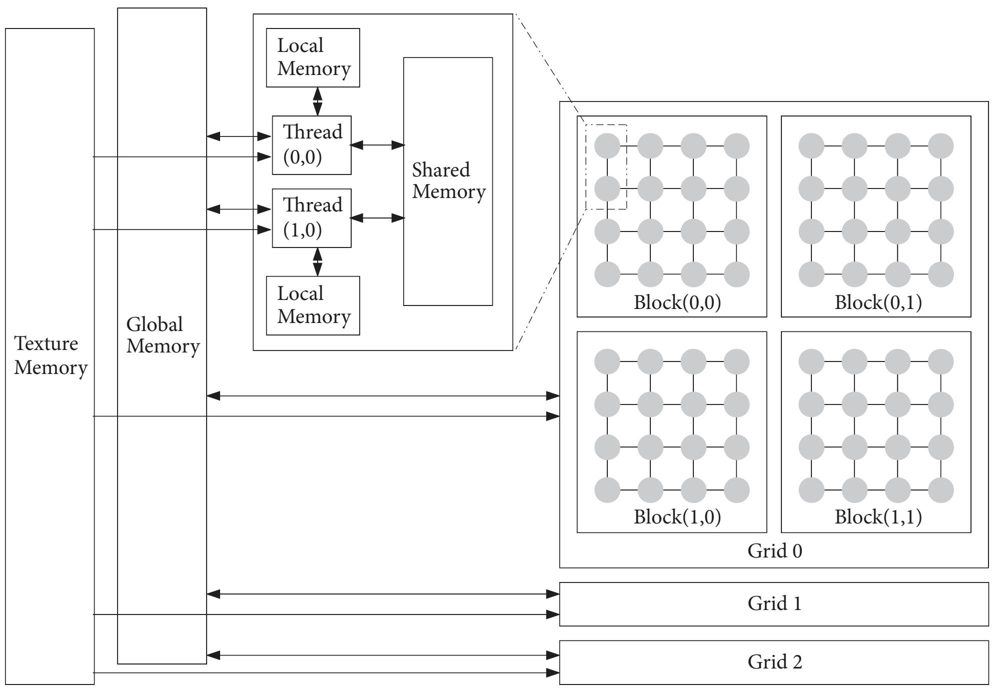

Figure 3: The hierarchy of threads and different types of memory of CUDA framework.

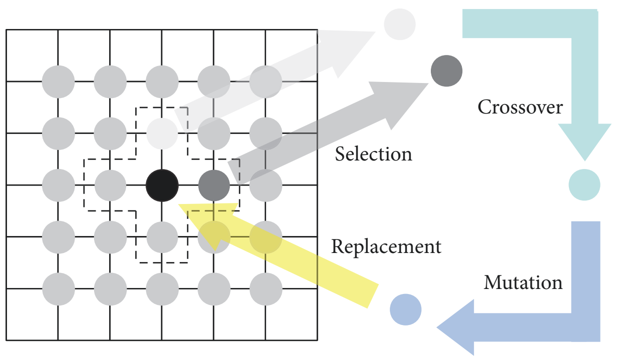

Figure 4: The population structure of the cellular GA and its operation procedure.

steps can be easily handled on a multicore CPU in parallel.

There are four types of memory in CUDA and their hierarchy is shown in Figure 3. Each CUDA thread handles one cellule of the cellular GA with GA operators on the 2D grid as illustrated in Figure 4. As the texture memory of CUDA is designed to gain an increase in performance for accelerating access patterns with spatial locality [34], we design the neighborhood area of the cellular GA as to be surrounded by the dashed lines in Figure 4. Firstly, the algorithm recombines two individuals selected from the nearby area to generate a new one. Afterwards, this new individual undertakes the mutation and replaces the original individual if its solution is better. Then, all individuals are sorted according to their fitness values using the Bitonic-Merge sort [35], if the cellular GA meets the island termination criterion but not the final termination criterion. During the full procedure, individuals' information is placed in the global memory while the neighbors' information is stored in the texture memory. The selection, the crossover, the replacement, and the Bitonic-Merge sort are executed through the global memory while the fitness evaluation and the mutation are handled via the local memory.

When the GPUs are occupied by executing the cellular GA, the pseudo-GA is run on a multicore CPU by OpenMP [36] which is an API supporting multiplatform shared memory multiprocessing programming. In this case, the GA operators on two heterogeneous islands are working in parallel on the host (a multicore CPU) and the device (GPUs) simultaneously. At the end, the Bitonic-Merge sort [35] is accomplished by the OpenMP-based code in a similar way as the cellular GA on CUDA.

4.3. Migration Policy. The migration between islands is controlled by the topology, the rate, the interval, and the strategy. To decrease the number of parameters that need to be set manually, we develop a migration policy inspired by the penetration theory [37] where a migration threshold value θ is set $(0 \le \theta \le 1)$. The execution of migration is decided by this value and there is more likely for individuals to migrate when $\theta = 1$. Moreover, the migration rate α and the migration direction indicator β are formulated as in

$$\alpha = \begin{cases} 1-\beta & 1-\beta < \theta \\ 0 & 1-\beta \ge \theta \end{cases} \tag{1}$$

Table 1: The experimental relative data of the large FFS problem.

WT	100
P_{jsm}	$U[1,5]$
R_j	$U[0,\overline{P}]$, where $\overline{P} = \sum_j \sum_s (\sum_m P_{jsm}/M_s)$
D_j	$R_j + \overline{P}_j(1+\sigma)$, where $\sigma = U[0,2]$ and $\overline{P}_j = \sum_s (\sum_m P_{jsm}/M_s)$

Table 2: Wilcoxon signed ranks test results.

Comparison	Generations	Job number = 100			Job number = 200			Job number = 300		
		R^+	R^-	p-value	R^+	R^-	p-value	R^+	R^-	p-value
heterogeneous GA versus cellular GA	500	0.00	1275.00	0.000	0.00	1275.00	0.000	0.00	1275.00	0.000
	1000	1103.00	172.00	0.000	1051.00	224.00	0.000	728.00	574.00	0.382
	1500	1257.00	18.00	0.000	1272.00	3.00	0.000	1275.00	0.00	0.000
	2000	1263.00	12.00	0.000	1275.00	0.00	0.000	1275.00	0.00	0.000
heterogeneous GA versus pseudo GA	500	1275.00	0.00	0.000	1275.00	0.00	0.000	1275.00	0.00	0.000
	1000	1275.00	0.00	0.000	1275.00	0.00	0.000	1275.00	0.00	0.000
	1500	1275.00	0.00	0.000	1275.00	0.00	0.000	1275.00	0.00	0.000
	2000	1088.00	187.00	0.000	1274.00	1.00	0.000	1275.00	0.00	0.000

$$\beta = \begin{cases} \dfrac{\mathrm{fit}_A}{\mathrm{fit}_B} & \mathrm{fit}_A < \mathrm{fit}_B \\ \dfrac{\mathrm{fit}_B}{\mathrm{fit}_A} & \mathrm{fit}_A > \mathrm{fit}_B \end{cases} \quad (2)$$

Here, fit_A and fit_B are the best individual's fitness value of subpopulation A on island A and subpopulation B on island B. In a certain generation, we calculate the above functions and carry out three steps as follows:

(i) If $1-\beta < \theta$, the migration is executed. Otherwise, do nothing.

(ii) The topology of migration is determined by the ratio of fit_A to fit_B. If $\mathrm{fit}_A/\mathrm{fit}_B > 1$, the migration is from subpopulation A to subpopulation B. If $\mathrm{fit}_A/\mathrm{fit}_B < 1$, the migration direction is reversed. If $\mathrm{fit}_A/\mathrm{fit}_B = 1$, no migration is implemented.

(iii) When the migration is carried, $\alpha \times p$ individuals with best fitness values in the emigrant subpopulation are selected to replace $\alpha \times p$ individuals with worst fitness values in the immigrant subpopulation.

The migration policy is executed by the CPU where results of the cellular GA on GPUs are sent back to the CPU at this moment. With this design, the topology, the rate, the interval, and the strategy no longer need to be considered manually. New merged individuals with good genes can be transited quickly and the execution time is saved by preventing ineffective information sharing.

5. Numerical Experiments

To analyze the performance of the proposed algorithm, we compare its solutions' quality and execution time with the parallel cellular GA and the parallel pseudo-GA. The population size is kept as 512 for all tested GAs while the subpopulation size for each island of the heterogeneous GA is 256. The crossover rate and the mutation rate of the cellular GA are set as 1.00 and 0.05, respectively [30], while the crossover rate of the pseudo-GA is equal to 0.75 [31]. The cellular GA from the dual heterogeneous GA keeps the same crossover rate and mutation rate as the cellular GA. Similarly, the pseudo-GA from the dual heterogeneous GA keeps the same crossover rate as the pseudo-GA. Moreover, to better check the influence of the migration, the migration threshold is fixed as 1.00. As large size FFS problems are concerned in this paper, the analyzed instances are characterized by 300 jobs with 4 stages and there are 2 available machines at each stage. Other experimental relative data are defined in Table 1. All parameters settings will be employed throughout the study, unless a particular case is stated explicitly.

The experimental platform is based on the Intel Xeon E5640 CPU with 2.67GHz clock speed and four cores. The GPU code implementation is carried out using CUDA 8.0 on NVIDIA Tesla K40, with 2880 cores at 0.745 GHz and 12 GB GDDR5 global memory. All programs are written in C, except for the GPU kernels in CUDA C. Tables 2–4 and Figures 5–9 display results of 2000 generations and they are average values of 50 runs.

5.1. Test on the Migration Policy Execution Gap. Even the topology, the rate, the interval, and the strategy are set adaptively when the migration policy is carried in a certain generation. We still need to test when to execute it since the migration policy needs call back results on GPUs and too frequent data exchange between the device and the host may weaken the performance of the proposed method. As it is displayed in Figure 5, the migration policy execution gap is increased from 10 generations to 800 generations and the island GA has a risk to fall in a local optimum if this value is either too small or too big. As a result, it finds that an inappropriate migration can also lead to the premature convergence, besides homogeneous genetic

Table 3: Solutions comparison among different GAs with different settings of crossover rate and mutation rate.

	Crossover rate of cellular GA	Mutation rate of cellular GA	Crossover rate of pseudo GA	Generations=500		Generations=1000		Generations=1500		Generations=2000	
				Avg.	Best	Avg.	Best	Avg.	Best	Avg.	Best
Cellular GA	0.75	0.05		326535.97	321666.09	326526.59	321666.09	326526.59	321666.09	326526.59	321666.09
	0.75	0.1		327179.02	320788.47	327143.56	320724.06	327143.56	320724.06	327143.56	320724.06
	0.75	0.15		325808.54	321231.75	325677.47	320496.38	325677.47	320496.38	325677.47	320496.38
	0.825	0.05		324897.77	321255.38	324884.82	321255.38	324881.76	321255.38	324881.76	321255.38
	0.825	0.1		324865.60	316669.25	324834.08	316537.31	324843.08	316537.31	324834.08	316537.31
	0.825	0.15		325315.74	320469.34	325204.08	319631.25	325204.08	319631.25	325204.08	319631.25
	0.9	0.05		322973.18	317704.62	322962.50	317661.12	322962.49	317661.12	322962.49	317661.12
	0.9	0.1		322522.17	317559.59	322462.99	317474.69	322462.99	317474.69	322462.99	317474.69
	0.9	0.15		323264.19	316454.66	323144.37	315824.53	323144.37	315824.53	323144.37	315824.53
Pseudo GA			0.75	346659.15	341943.28	332607.32	330122.19	323422.25	320771.78	317254.68	315120.00
			0.825	346662.04	343632.38	332508.13	328620.69	322942.15	319277.88	316814.56	313505.47
			0.9	346037.89	340650.22	332287.24	328877.31	322846.37	319892.25	316400.91	313716.28
Heterogeneous GA	1.0	0.05	0.75	333811.44	328588.75	320091.08	315801.28	312876.46	309713.34	309885.90	306500.03

Table 4: Solutions comparison among different GAs with different settings of island size.

	Island amount (Island size)	Generations=500		Generations=1000		Generations=1500		Generations=2000	
		Avg.	Best	Avg.	Best	Avg.	Best	Avg.	Best
Island cellular GA	2 (256)	332343.94	328144.34	332343.94	328144.34	332343.94	328144.34	332343.94	328144.34
	8 (64)	350598.23	346120.12	344591.80	339518.22	340262.62	333733.62	336830.04	332179.34
	32 (16)	363540.32	360575.00	359771.15	359697.66	356632.84	352502.31	354473.03	351107.38
Island pseudo GA	2 (256)	338342.11	334721.91	324257.65	321016.00	317882.28	314946.06	315330.97	312015.50
	8 (64)	337958.95	334247.50	330252.24	323065.22	324689.42	317601.56	320122.33	312462.94
	32 (16)	360301.02	357540.59	357444.52	354706.56	355326.44	352325.09	353490.46	348249.91
Heterogeneous GA	2 (256)	333811.44	328588.75	320091.08	315801.28	312876.46	309713.34	309885.90	306500.03

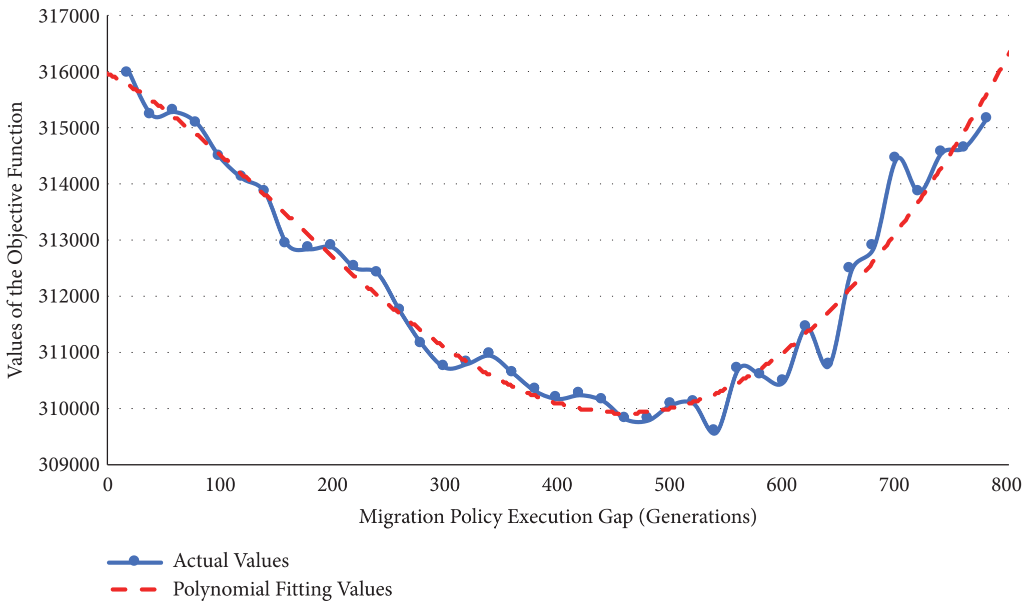

Figure 5: The influence of the migration policy execution gap for the heterogeneous GA.

operator configurations and limited subpopulation sizes. Following the polynomial fitting values, the best performance for the tested instance is obtained when the migration policy execution gap is around 500 generations and we keep this setting for the remaining tests in this paper.

5.2. Comparison Test on the Solutions' Quality with Different Size Problems. The convergence trends of different GAs with different size problems are illustrated in Figures 6, 7, and 8, respectively. Although the specific designs of the cellular GA and the pseudo-GA can help increase population diversity, each one has a significant shortage. As shown in Figures 6–8, the objective function value of the cellular GA decreases faster than other two methods at the beginning while it is stuck in a local optimum after several generations. The solution quality of the pseudo-GA is better than the cellular GA at the end of the evolution, but its convergence speed is much slower. The proposed design combines the merits from the cellular GA and the pseudo-GA while eliminating their shortages by the independent evolution and the penetration migration. It is easy to find elbows in the convergence curve of the heterogeneous GA and they always appear around the generations where the migration policy is executed. This phenomenon witnesses the process of how the premature convergence is avoided thanks to two heterogeneous islands are connected by the penetration migration. Thus, the proposed design has a larger chance than the cellular GA to find the global optimum while it converges faster than the pseudo-GA. Moreover, this advantage is even more distinguished when this method is taken to solve larger size problems.

To confirm the conclusion that we have got from the convergence trend among different GAs, the Wilcoxon signed ranks test [38] is taken for characterizing the behavior of the heterogeneous GA, in 1x1 comparisons with the cellular GA and the pseudo-GA considering different size problems. In this test, the difference between the performance scores of the two algorithms on n instances is recorded. The differences are ranked according to their absolute values. Let R^+ be the sum of ranks for the instances in which the first algorithm outperforms the second and R^- the sum of ranks for the opposite. T is set to be the smallest of the sums, T =

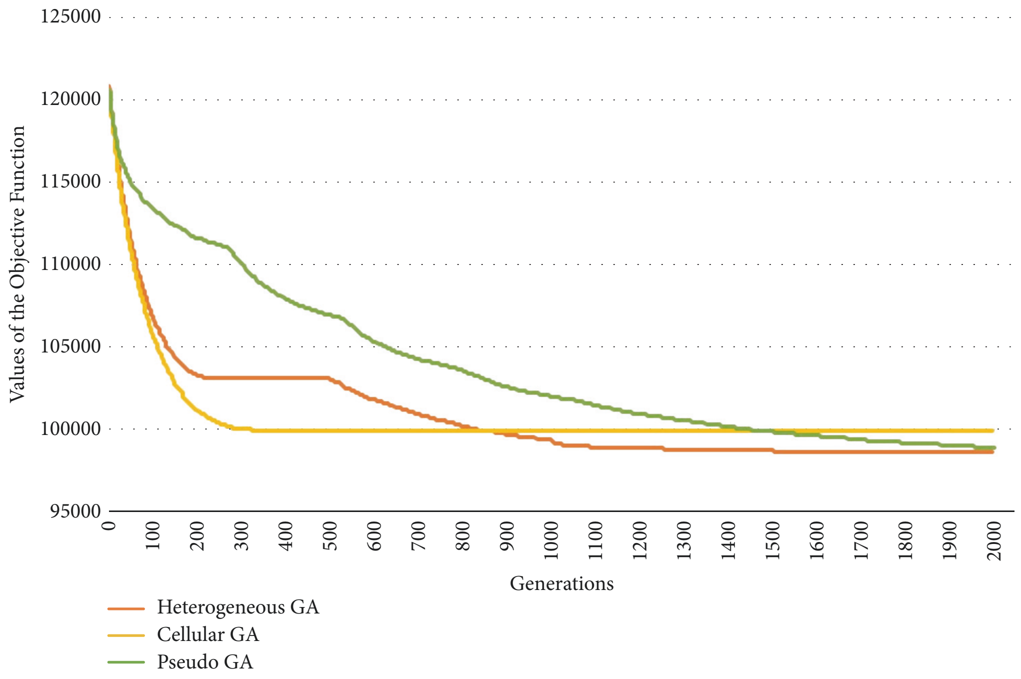

Figure 6: The convergence trend among different GAs (job number = 100).

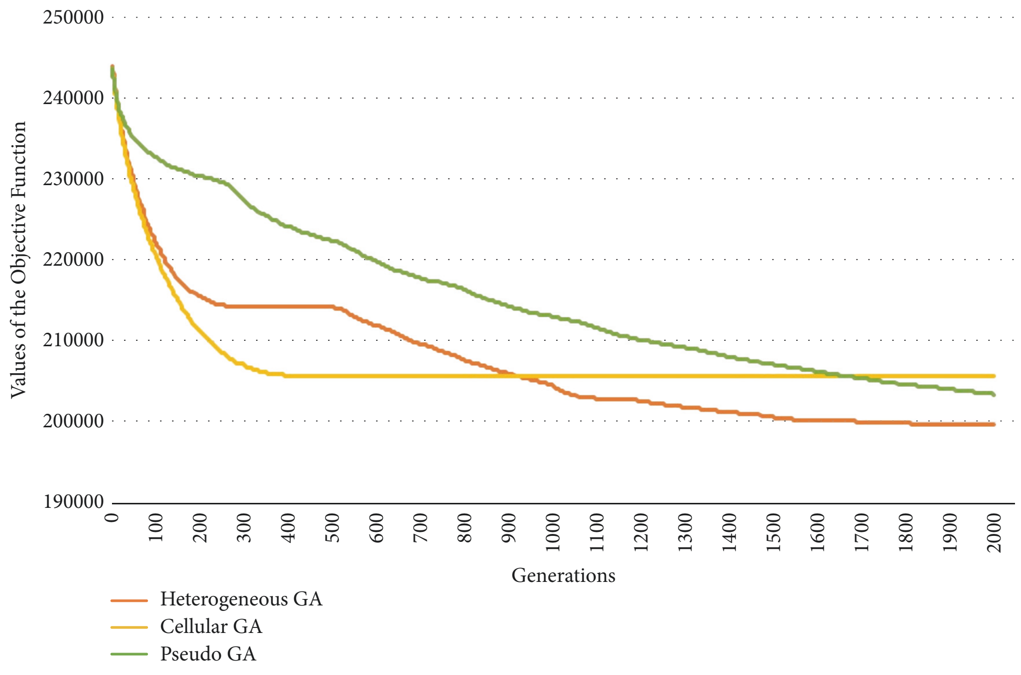

Figure 7: The convergence trend among different GAs (job number = 200).

$\min(R^+, R^-)$. If T is less than or equal to the value of the distribution of Wilcoxon for n degrees of freedom, it means a given algorithm outperforms the other one, with the p value associated. Table 2 shows the R^+, R^-, and p values computed for all the pairwise comparisons concerning the heterogeneous GA. All values have been computed by SPSS [39]. As the table states, the heterogeneous GA shows a significant improvement over the cellular GA and the pseudo-GA starting from 1500 generations for three different size problems with a level of significance equals to 0.01. On the other hand, the cellular GA is better at finding good solutions at the beginning stages but the R^- values keep decreasing when the generations are increasing. The pseudo-GA performs worse than the heterogeneous GA in most of the cases while the R^- values become increased at the end of the evolution.

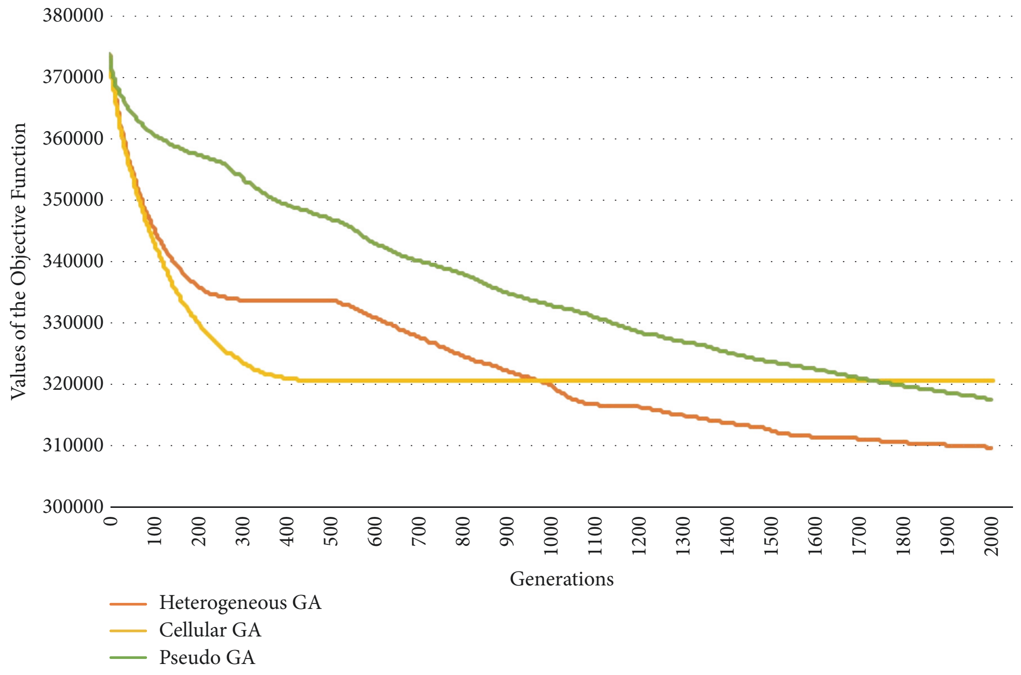

FIGURE 8: The convergence trend among different GAs (job number = 300).

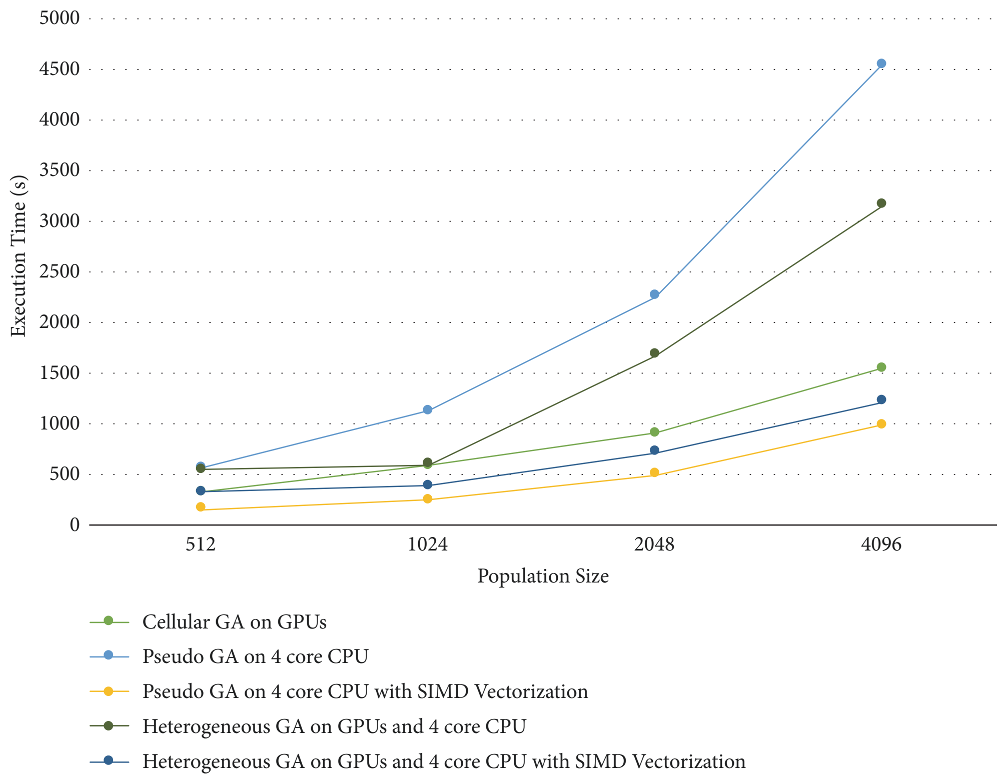

FIGURE 9: The execution time comparison among different parallel GAs.

5.3. Comparison Test on the Solutions' Quality with Different Parameters. Considering the existing experiences, the most appropriate crossover rate ranges between 0.75 and 0.9 [40] and the mutation rate should be much lower than the crossover rate [41]. Therefore, we firstly compare the efficiency of the heterogeneous GA with the cellular GA and the pseudo-GA by three groups of crossover rates and three groups of mutation rates as shown in Table 3. The results state that the crossover rate and the mutation rate do have some impact on the algorithm performance. However, the influence is limited. Besides, the heterogeneous GA can always find better solutions with the average value and the best value than other two GAs after 500 generations.

As a next step, we try to divide the population of the cellular GA and the pseudo-GA into a few relatively large subpopulations and check their performance as island cellular GAs and island pseudo-GAs with the heterogeneous model. In this case, each island works independently as the regular cellular GA or the regular pseudo-GA. Islands are interconnected with a single ring. An island can accept an individual with the best fitness value from the neighbor to overwrite the one with the worst fitness value as the migration. The migration interval is kept as 500 generations as the heterogeneous GA. As the results displayed in Table 4, the heterogeneous GA overcomes both island cellular GAs and island pseudo-GAs after 500 generations. Moreover, it verifies one phenomenon that the finite island size and the same genetic operator configuration in each island may make island GAs be apt to yield premature convergence.

5.4. Comparison Test on the Execution Time. To check the execution time among these parallel GAs, we consider different population sizes from 512 to 4096. The cellular GA is fully carried on GPUs. The pseudo-GA is generated on a four-core CPU with or without SIMD vectorization. The two islands of the heterogeneous GA are generated on GPUs and a CPU simultaneously. Similarly, the pseudo-GA from the dual heterogeneous GA is parallelized on the four-core CPU with or without SIMD vectorization. The SIMD vectorization is executed via SSE2 [42], as far as this experiment platform is available. Concerning results in Figure 9, the heterogeneous GA on the hybrid platform takes less execution time than the pseudo-GA on a 4-core CPU as the heterogeneous design can be well parallelized on both sides simultaneously. However, it loses to the cellular GA because the number of individuals executed on GPUs and the threads occupancy is twice as much as the heterogeneous GA on the hybrid platform. Fortunately, the performance of the heterogeneous structure gets improved significantly when the computation capability on the four-core CPU is enhanced by the SIMD vectorization. It points out the importance of computation capability balance between the host and the device when the proposed approach is implemented where the weak side may become as a bottleneck and reduces the overall effectiveness. Finally, because the pseudo-GA only deals with binary integers whose storage size is small, the contribution of the SIMD vectorization is impressive and the pseudo-GA on a four-core CPU with vectorization overcomes the others.

6. Conclusions and Future Works

A dual heterogeneous island GA was proposed in this paper. It was composed of a cellular GA on GPUs and a pseudo-GA on a multicore CPU where the 2D variable space of the cellular GA and the complementary parent strategy of the pseudo-GA kept the population diversity. This structure was highly consistent with the underlying architecture which can be fully parallelized inside or between GPUs and a multicore CPU. Since the two islands evolved independently in different ways, a penetration inspired migration was designed to share information between them and to decrease the risk of premature convergence. For solving some large instances of the FFS problem, it firstly found out the importance of an appropriate migration implementation. Otherwise, the migration could cause genetic drift and lead to a convergence towards a local optimum. The second test showed the proposed method obtained better solutions with different size problems because of the merits from two different islands and confirmed the efficiency of the penetration migration. The third test further checks its efficiency by comparing the results with the cellular GA and the pseudo-GA who both have tuned parameters. Finally, the effectiveness of the dual heterogeneous island GA was displayed by comparison tests with other parallel methods and pointed that the balance of computation capability between the host and the device had a great influence on its overall performance.

Three main areas that deserve further study are identified. The first issue is to automatically decide the island size on two platforms according to the computation capability of GPUs and a multicore CPU. A second line of interest is to analyze the impact of the migration policy execution gap when it is carried out asynchronously and compare its influence with the current synchronous design. As multi-GPU systems and multi-CPU systems have become more and more common in recent years, finally we would like to test multi-island cellular GAs and multi-island pseudo-GAs on a multi-GPU system and a multi-CPU system with a modified migration policy.

Data Availability

All data used to support the findings of this study is available from the first author upon request.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

This work is supported by a scholarship from the China Scholarship Council (CSC). Moreover, Didier El Baz is grateful for the help of NVIDIA Corporation for the donation of the Tesla K40 GPUs used in this work and the authors would like to express their gratitude to the editors and the reviewers for their helpful comments.

References

[1] S. H. Bhatt, M. B. Kiran, A. B. Bhesdadiya, A. P. Singh, and P. V. Shah, *Review on Branch and Bound technique in Flexible Manufacturing System Scheduling*, 2017.

[2] F. Xu, W. Weng, and S. Fujimura, “Energy-efficient scheduling for flexible flow shops by using MIP,” in *Proceedings of the IIE*

Annual Conference, vol. 1040, Institute of Industrial and Systems Engineers (IISE), 2014.

[3] J. Behnamian, "Diversified particle swarm optimization for hybrid flowshop scheduling," *Journal of Optimization in Industrial Engineering*, 2018.

[4] H. Öztop, D. T. Eliiyi, M. Fatih Tasgetiren, and Q.-K. Pan, "Iterated greedy algorithms for the hybrid flowshop scheduling with total flow time minimization," in *Proceedings of the 2018 Genetic and Evolutionary Computation Conference, GECCO 2018*, pp. 379–385, ACM, July 2018.

[5] E. Cantú-Paz, "A survey of parallel genetic algorithms," *Calculateurs Paralleles, Reseaux Et Systems Repartis*, vol. 10, no. 2, pp. 141–171, 1998.

[6] Z. Konfrist, "Parallel genetic algorithms: advances, computing trends, applications and perspectives," in *Proceedings of the 18th International Parallel and Distributed Processing Symposium*, pp. 162–169, Santa Fe, NM, USA, 2004.

[7] A. A. Gozali and S. Fujimura, "Localization strategy for island model genetic algorithm to preserve population diversity," in *Proceedings of the International Conference on Computer and Information Science*, pp. 149–161, Springer, May 2017.

[8] J. Jaros and P. Pospichal, "A fair comparison of modern CPUs and GPUs running the genetic algorithm under the knapsack benchmark," in *Proceedings of the European Conference on the Applications of Evolutionary Computation*, pp. 426–435, Springer, April 2012.

[9] T. Hiroyasu, M. Miki, M. Hamasaki, and Y. Tanimura, "A new model of distributed genetic algorithm for cluster systems: dual individual DGA," in *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications*, vol. 1, pp. 477–483, 2000.

[10] T. Hiroyasu, M. Miki, M. Hamasaki, and Y. Tanimura, "A new model of parallel distributed genetic algorithms for cluster systems: dual individual DGAs," in *Proceedings of the International Symposium on High Performance Computing*, pp. 374–383, Springer, October 2000.

[11] F. Herrera and M. Lozano, "Gradual distributed real-coded genetic algorithms," *IEEE Transactions on Evolutionary Computation*, vol. 4, no. 1, pp. 43–63, 2000.

[12] E. Alba, F. Luna, A. J. Nebro, and J. M. Troya, "Parallel heterogeneous genetic algorithms for continuous optimization," *Parallel Computing*, vol. 30, no. 5-6, pp. 699–719, 2004.

[13] M. Kaneko, M. Miki, and T. Hiroyasu, "A parallel genetic algorithm with distributed environment scheme," in *PDPTA*, 2000.

[14] E. Alba, A. J. Nebro, and J. M. Troya, "Heterogeneous computing and parallel genetic algorithms," *Journal of Parallel and Distributed Computing*, vol. 62, no. 9, pp. 1362–1385, 2002.

[15] P. García-Sánchez, G. Romero, J. González et al., "Studying the effect of population size in distributed evolutionary algorithms on heterogeneous clusters," *Applied Soft Computing*, vol. 38, pp. 530–547, 2016.

[16] M. García-Valdez, L. Trujillo, J. J. Merelo-Guérvos et al., "Randomized parameter settings for heterogeneous workers in a pool-based evolutionary algorithm," in *Proceedings of the International Conference on Parallel Problem Solving from Nature*, pp. 702–710, Springer, September 2014.

[17] W. B. Langdon, "Graphics processing units and genetic programming: An overview," *Soft Computing*, vol. 15, no. 8, pp. 1657–1669, 2011.

[18] P. Krömer, J. Platoš, and V. Snášel, "Nature-inspired metaheuristics on modern GPUs: State of the art and brief survey of selected algorithms," *International Journal of Parallel Programming*, vol. 42, no. 5, pp. 681–709, 2014.

[19] M. Pedemonte, F. Luna, and E. Alba, "Systolic genetic search, a systolic computing-based metaheuristic," *Soft Computing*, vol. 19, no. 7, pp. 1779–1801, 2015.

[20] N. Soca, J. L. Blengio, M. Pedemonte, and P. Ezzatti, "PUGACE, a cellular evolutionary algorithm framework on GPUs," in *Proceedings of the 2010 6th IEEE World Congress on Computational Intelligence, WCCI 2010 - 2010 IEEE Congress on Evolutionary Computation, CEC 2010*, pp. 1–8, IEEE, July 2010.

[21] A. Dabah, A. Bendjoudi, A. AitZai, D. El-Baz, and N. N. Taboudjemat, "Hybrid multi-core CPU and GPU-based B&B approaches for the blocking job shop scheduling problem," *Journal of Parallel and Distributed Computing*, vol. 117, pp. 73–85, 2018.

[22] P. Benner, P. Ezzatti, D. Kressner, E. S. Quintana-Ortí, and A. Remón, "A mixed-precision algorithm for the solution of Lyapunov equations on hybrid CPU-GPU platforms," *Parallel Computing*, vol. 37, no. 8, pp. 439–450, 2011.

[23] B. R. Bilel, N. Navid, and M. S. M. Bouksiaa, "Hybrid CPU-GPU distributed framework for large scale mobile networks simulation," in *Proceedings of the 2012 IEEE/ACM 16th International Symposium on Distributed Simulation and Real Time Applications, DS-RT 2012*, pp. 44–53, October 2012.

[24] M. Kurdi, "An effective new island model genetic algorithm for job shop scheduling problem," *Computers & Operations Research*, vol. 67, pp. 132–142, 2016.

[25] F. M. Defersha and M. Chen, "A coarse-grain parallel genetic algorithm for flexible job-shop scheduling with lot streaming," in *Proceedings of the International Conference on Computational Science and Engineering, 2009. CSE'09*, vol. 1, pp. 201–208, IEEE, August 2009.

[26] T. Zajıcek and P. Šucha, "Accelerating a Flow Shop Scheduling Algorithm on the GPU," *Eraerts*, vol. 143, 2011.

[27] C.-S. Huang, Y.-C. Huang, and P.-J. Lai, "Modified genetic algorithms for solving fuzzy flow shop scheduling problems and their implementation with CUDA," *Expert Systems with Applications*, vol. 39, no. 5, pp. 4999–5005, 2012.

[28] A. A. G. Bruzzone, D. Anghinolfi, M. Paolucci, and F. Tonelli, "Energy-aware scheduling for improving manufacturing process sustainability: a mathematical model for flexible flow shops," *CIRP Annals - Manufacturing Technology*, vol. 61, no. 1, pp. 459–462, 2012.

[29] J. N. D. Gupta, "Two-stage, hybrid flowshop scheduling problem," *Journal of the Operational Research Society*, vol. 39, no. 4, pp. 359–364, 1988.

[30] P. Vidal and E. Alba, "Cellular genetic algorithm on graphic processing units," in *Nature Inspired Cooperative Strategies for Optimization (NICSO 2010)*, pp. 223–232, Springer, Berlin, Germany, 2010.

[31] Q. Chen, Y. Zhong, and X. Zhang, "A pseudo genetic algorithm," *Computing and Applications*, vol. 19, no. 1, pp. 77–83, 2010.

[32] https://developer.nvidia.com/cuda-toolkit.

[33] B. Plazolles, D. El Baz, M. Spel, V. Rivola, and P. Gegout, "SIMD monte-carlo numerical simulations accelerated on GPU and xeon phi," *International Journal of Parallel Programming*, vol. 46, no. 3, pp. 584–606, 2018.

[34] J. Sanders and E. Kandrot, *CUDA by Example: An Introduction to General-Purpose GPU Programming*, Addison-Wesley Professional, 2010.

[35] M. Pharr and R. Fernando, *Gpu Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation*, Addison-Wesley Professional, 2005.

[36] http://www.openmp.org/.

[37] J. Gu, X. Gu, and M. Gu, "A novel parallel quantum genetic algorithm for stochastic job shop scheduling," *Journal of Mathematical Analysis and Applications*, vol. 355, no. 1, pp. 63–81, 2009.

[38] J. Derrac, S. García, D. Molina, and F. Herrera, "A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms," *Swarm and Evolutionary Computation*, vol. 1, no. 1, pp. 3–18, 2011.

[39] https://www.ibm.com/analytics/spss-statistics-software.

[40] J. D. A. R. Schaffer, *A Study of Control Parameters Affecting Online Performance of Genetic Algorithms for Function Optimization*, San Meteo, Calif, USA, 1989.

[41] J. A. Cabrera, A. Simon, and M. Prado, "Optimal synthesis of mechanisms with genetic algorithms," *Mechanism and Machine Theory*, vol. 37, no. 10, pp. 1165–1177, 2002.

[42] https://en.wikipedia.org/wiki/Streaming_SIMD_Extensions.