Solving the dynamic energy aware job shop scheduling problem with the heterogeneous parallel genetic algorithm

Jia Luo, Didier El Baz, Rui Xue, Jinglu Hu

Solving the Dynamic Energy Aware Job Shop Scheduling Problem with the Heterogeneous Parallel Genetic Algorithm

Jia Luo[a,b,c,d], Didier El Baz [b], Rui Xue[a,1*], Jinglu Hu [c]
a College of Economics and Management, Beijing University of Technology, Beijing, China
b LAAS-CNRS, Université de Toulouse, CNRS, Toulouse, France
c Graduate School of Information, Production, and Systems, Waseda University, Kitakyushu, Japan
d International Research Fellow of Japan Society for the Promotion of Science

Abstract

Integrating energy savings into production efficiency is considered as one essential factor in modern industrial practice. A lot of research dealing with energy efficiency problems in the manufacturing process focuses solely on building a mathematical model within a static scenario. However, in the physical world shop scheduling problems are dynamic where unexpected events may lead to changes in the original schedule after the start time. This paper makes an investigation into minimizing the total tardiness, the total energy cost and the disruption to the original schedule in the job shop with new urgent arrival jobs. Because of the NP hardness of this problem, a dual heterogeneous island parallel genetic algorithm with the event driven strategy is developed. To reach a quick response in the dynamic scenario, the method we propose is made with a two-level parallelization where the lower level is appropriate for concurrent execution within GPUs or a multi-core CPU while codes from the two sides can be executed simultaneously at the upper level. In the end, numerical tests are implemented and display that the proposed approach can solve the problem efficiently. Meanwhile, the average results have been improved with a significant execution time decrease.

Key words

Job shop scheduling, Energy efficiency, Dynamic scheduling, Parallel genetic algorithm, Multi-core processing, GPU computing

[1]* Corresponding Author

1. Introduction

Energy costs due to production have been traditionally treated as externalities that must be incurred [1]. With an increasing interest for industrial sustainability, integrating energy savings into production efficiency is considered as one essential factor in factory practice. There are two kinds of approaches studying energy saving in manufacturing systems [2]: avoiding peak power consumption and reducing the overall energy cost. The first one [3, 4, 5] shifts load at energy peaks when the maximum available energy is met. The second one [6, 7, 8] focuses on decreasing the total energy cost in manufacturing system by subdividing it and switching them among different types and different levels. Most of these research works focus solely on building a mathematical model within a static scenario. However, unexpected events may lead changes in the preset schedule after the start time. Few works focus on dynamic energy aware shop scheduling problems and most of them [9, 10, 11] were solved by the complete rescheduling with a risk in instability. Moreover, scheduling problems in dynamic scenarios are more complicated than scheduling problems in static scenarios and the time cost to obtain the optimal solution or even a high-quality solution is heavy. Therefore, an approach proposing an appropriate updated schedule within a reasonable time is highly desirable in this case.

Parallel computing has been widely used for years. The multi-core CPU can run multiple instructions at the same time on separate cores to increase the overall speed while Graphics Processing Units (GPUs) are many-core processor devices providing a highly multi-threaded environment using the Single Instruction, Multiple Threads (SIMT) model. Since most of latest computers are furnished with a multi-core CPU and GPUs, the execution on both is an effective strategy to utilize hardware in an efficient way. Investigation on solving scheduling problems in manufacturing processes by parallel computing methods [12] has received increasing attention in the last decades. However, the sophisticated issue as energy aware shop scheduling in dynamic scenarios was never considered as best as we are aware. On the other side, there is a great number of successful cases [13,14, 15] proving that parallel GAs are reliable for solving shop scheduling problems. But most of them either only use the CPU, the GPUs or two of them in sequence which may end up to an underuse of computing resources due to the hardness of designing schemes that efficiently exploit simultaneously different hardware architectures. Thus, the design of parallel GAs on hybrid CPU–GPU frameworks for solving dynamic energy aware shop scheduling problems is a known research challenge following previous works, and this is what we are trying to solve in this paper.

An investigation into minimizing the total tardiness and the total energy cost in the job shop with new urgent arrival jobs is concerned in this paper. To avoid the shortages of the complete rescheduling, the schedule repair with the event driven strategy is utilized to represent the updated solution. Since the response time is sensitive in the dynamic scenario, a dual heterogeneous island parallel GA executed simultaneously on GPUs and a multi-core CPU is utilized. Numerical tests confirm the proposed method can be implemented to solve the problem efficiently and effectively. Our main work can be summed up as follows:

1. A dynamic energy aware job shop scheduling model is studied which seeks a trade-off among the total tardiness, the energy cost and the disruption to the original schedule;

2. A parallel GA is adapted for solving the proposed problem, whose lower level is appropriate for concurrent execution within GPUs or a multi-core CPU while codes from the two sides can be executed simultaneously at the upper level;

3. Numerical experiments have been carried out and witness that the adapted parallel GA can not only solve the proposed problem efficiently but also improve the average results with a

significant execution time decrease.

The remaining of our work consists of 5 sections. In section 2, the literature review is presented. Section 3 exposes the research problem and formulates the mathematical model. Section 4 discusses the design of the parallel GA on hybrid CPU–GPU frameworks and its implementation for solving dynamic energy aware shop scheduling problems. Afterwards, computational tests and a case study are conducted in section 5. Finally, conclusions are stated in section 6.

2. Literature review

Due to environmental concerns and continuously rising cost, there is an increasing interest in energy saving in traditional industrial processes. Since moving the production activities in off-peak periods or inserting idle times for machine may not be acceptable with intense production process or fixed working time shifts [1], minimizing the overall energy cost is considered one main solution. Meng et al. discussed the total energy consumption for flexible job shop scheduling problems in [16] and solved it by six new mixed integer linear programming models. He et al. [17] proposed a model synthesizing the optimization of energy consumption and makespan while the optimal solutions were obtained by the Tabu search. Meanwhile, the GA or the improved GA is one powerful and frequently used method to deal with the total energy cost integrated scheduling problems. Liu et al. [6] developed a non-dominant sorting GA and obtained the Pareto front for a bi-objective job shop scheduling problem that investigated into minimizing total electricity cost and total tardiness. Similarly, a modified multi-objective GA was studied in [7] and it was utilized to solve a multi-machine job shop scheduling model with emission aware issues. In one word, numerous efforts have been given to combine the traditional shop scheduling efficiency with the overall energy cost. However, the models used in these researches are deterministic in which the number of jobs is a fixed value [6]. As an ongoing reactive process where the presence of a variety of unexpected disruptions is usually inevitable [18], the static scheduling obviously cannot meet the requirements in most real-world environments.

Literature on dynamic scheduling has considered a significant number of works dealing with new arrival jobs and their effects in various manufacturing systems [18]. Most efforts concentrated only on the efficiency improvement for traditional scheduling problems while neglecting the energy cost. In the dynamic scenario, complete rescheduling and schedule repair are the two most common used strategies. An improved particle swarm optimization was adopted in [9] to allocate the new jobs and the previous remaining operations simultaneously for an energy saving dynamic scheduling problem. Zhang et al. studied the dynamic rescheduling considering energy consumption in [10] where optimal solutions were found by a GA with the complete rescheduling strategy. Even the complete rescheduling provides the optimal solutions, it can result in instability and disruption in manufacturing flows, leading to tremendous production costs [19]. On the opposite, schedule repair only attempts to revise part of the originally created schedule for responding to the production environment changes. In [2], Pach et al. set up flexible manufacturing systems using potential fields where resources could switch to less energy consumption mode by sensing the intentions from products. Zeng et al. [20] presented a particle swarm optimization to solve the energy consumption based dynamic scheduling problem by introducing idle time windows. To sum up, some efforts concerning energy efficient scheduling problems in dynamic scenarios have been conducted. It shows that the schedule repair strategy is more practical to deal with the dynamic manufacturing system in the real world. But many limitations are still remaining that must be taken into account. One for instance is to get the appropriate updated schedule within a reasonable time, especially for large size manufacturing applications.

With the huge evolution of multi-core CPUs and GPUs, some works have considered the cooperation between them to maximally utilize their compute capability. A parallelization mixing

the multi-core CPU and the GPUs was studied by Dabah et al in [12] where a group of blocking job shop scheduling problems were solved efficiently. In [21], Hawick et al. described the use of threading approaches and multi-core CPUs to control independent GPU devices to speed up scientific simulations. Hossam et al. [22] introduced a parallel implementation of hybrid CPU/GPU in which CPU and GPU work cooperatively and seamlessly, combining benefits of both platforms. All these works have verified that a scheme exploiting a multi-core CPU and GPUs corporately can increase the hardware occupation and achieve a speedup. However, this strategy is rarely implemented for GAs, in particular for the implementation of parallel GAs to solve dynamic energy aware shop scheduling problems, as far our knowledge is concerned.

Considering the above-mentioned requirements, we seek to study parallel GAs for solving the dynamic energy aware job shop scheduling problem on hybrid CPU–GPU frameworks. All the previous studies have afforded us with a starting point to design a GA that is well suited for parallelization on different architectures. Moreover, this implementation is efficient to provide appropriate solutions for large dynamic energy aware job shop scheduling problems within a short response time.

3. Problem statement

3.1 EDJSP description

The Job Shop scheduling Problem (JSP) is a NP-hard problem [23] in which there are several jobs and each job consists of a certain amount of operations. One operation is processed by a particular machine and every job is assigned to a group of machines following a predetermined route [6]. As a layout shown in Figure 1, job A and job B need to be processed by 4 machines and their processing routines are fixed as Machine 0-2-1-3 and Machine 2-0-3-1, respectively.

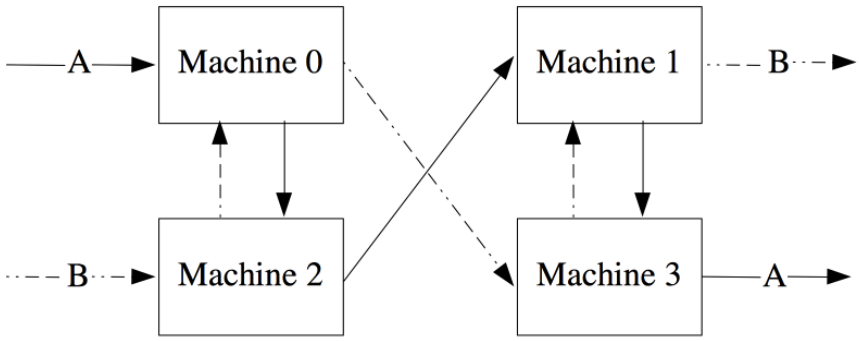

Fig. 1. A job shop layout

The Energy aware Dynamic Job Shop scheduling Problem (EDJSP) is an extension of the JSP with machine speed scaling [24] in which machines are available to be set at different speed levels when dealing with different jobs. The processing time and the energy cost of one operation processed on one machine at a set speed level are known. When a higher speed level is chosen, the processing time is shortened but with an energy cost increase. When machines start to handle original jobs following the preset schedule, a batch of new jobs may arrive and these jobs are requested to be processed as soon as possible. Therefore, the production line conducts them immediately with the highest speed level. The operations being processed are terminated and need to be rescheduled with other unfinished operations of original jobs based on the insertion of new urgent arrival jobs. The updated schedule using the schedule repair refers to some local adjustment of the original one. There are two possible schedule repair measures for the impact caused by the schedule changes [25]: (1) the deviation from the original jobs starting times, (2) the deviation from the original sequence. In this paper, a measure modified from (1) is taken into consideration where each original

job has an importance weight and a larger importance weight indicates a higher penalty for delaying the finishing time of original jobs from the original schedule. If one operation of a new urgent arrival job is added before one operation of an original job on the same machine, a higher speed level with less processing time but more energy cost is required to make the original job to be completed as close as to its finishing time in the original schedule. Clearly, there are conflicts among the minimization of total tardiness, the minimization of total energy cost and the minimization of disruption to the original schedule. Thus, a trade-off must be made among them. Because of the NP hardness of the JSP, the EDJSP is a NP-hard problem and more complicated than the JSP.

3.2 Mathematical model of EDJSP

A description of the notations referred within the remaining sections of this paper is summarized in Table 1.

Table 1 The used notations

Notation	Description
j, i, l, x, z	Job indices
s, t, y	Operation indices
m	Machine index
p, q, w	Speed level indices
n	Amount of original jobs
n'	Amount of new urgent arrival jobs
r1	Amount of completed operations of original jobs before the rescheduling point
r2	Sum of completed operations of original jobs before the rescheduling point and operations of new urgent arrival jobs
o_j	Amount of operations of job j
g	Amount of machines
h	Amount of speed levels
J	Set of original jobs, $J = \{0,1,2, \dots, n-1\}$
J'	Set of new arrival jobs, $J' = \{0,1,2, \dots, n'-1\}$
O_j	Set of operations of job j, $O_j = \{0,1,2, \dots, o_j-1\}$
M	Set of machines, $M = \{0,1,2, \dots, g-1\}$
L	Set of speed levels, L = $\{0,1,2, \dots, h-1\}$
R_j	Release time of job j, $j \in J \cup J'$
D_j	Due time of job j, $j \in J \cup J'$
T_j	Tardiness of job j, $j \in J \cup J'$
M_{js}	Target machine handling operation s of job j, $j \in J \cup J', s \in O_j$
RS	Rescheduling point
P_{jsmp}	Processing time when operation s of job j handled by target machine m at speed level p, $j \in J \cup J', s \in O_j, m \epsilon M, p \epsilon L$
Q_{jsmp}	Energy cost when operation s of job j handled by target machine m at speed level p, $j \in J \cup J', s \in O_j, m \epsilon M, p \epsilon L$
Z_{jsmp}	Boolean variable, it is equal to 1 if operation s of job j is handled by target machine m at speed level p, otherwise, it equals to 0, $j \in J \cup J', s \in O_j, m \epsilon M, p \epsilon L$
S_{jsm}	Original start time of operation s of original job j on machine m, $j \in J, s \in O_j, m \epsilon M$

S'_{jsm}	New start time of operation s of job j on machine m, $j \in J \cup J', s \in O_j, m \in M$
TT	Total tardiness of all jobs
ET_{max}	Estimated maximum value of TT
ET_{min}	Estimated minimum value of TT
TE	Total energy cost
EE_{max}	Estimated maximum value of TE
EE_{min}	Estimated minimum value of TE
wt_j	Importance weight of original job j, $j \in J$
DEV	Weighted finishing time deviation of the updated schedule from the original one
ED_{max}	Estimated maximum value of DEV
ED_{min}	Estimated minimum value of DEV
α, β, γ	Weight of each normalized objective function.
θ	Migration threshold value, $0 \leq \theta \leq 1$
λ	Migration rate, $0 \leq \lambda \leq 1$
φ	Migration policy execution gap, the frequency to perform the migration policy as defined in equation (12).
fit_A	The best individual's fitness value of subpopulation A on island A
fit_B	The best individual's fitness value of subpopulation B on island B
a, b, c, f	Gene indices in a chromosome
v_j	Index of occurrence time of job j
u_j	Occurrence time of job j
U	Set of occurrence time of a job number, $U = \{0,1,2, \dots, u_j - 1\}$
k	Current generation number of the GA
$X(k)$	Operation permutation of original schedule at generation k
$Y(k)$	Speed level permutation of original schedule at generation k
$X'(k)$	Operation permutation of new schedule at generation k
$Y'(k)$	Speed level permutation of new schedule at generation k
o_{js}	Operation s of job j
d, e	Indices for operations on machine m
o'_m	Number of operations on machine m before operation s of job j is assigned on it
O'_m	Set of operations on machine m before operation s of job j is assigned on it, $O'_m = \{0,1,2, \dots, o'_m - 1\}$
N	The number of orthogonal arrays in the Taguchi method
F	The response values in the Taguchi method
P_{cec}	The crossover rate of the cellular GA
P_{cem}	The mutation rate of the cellular GA

P_{cac}	The crossover rate of the classic GA
P_{cam}	The mutation rate of the classic GA

To minimize the total tardiness, the total energy cost and the delay caused by the schedule changes, the formal mathematical model of the EDJPS is derived from the mathematical models presented in [25, 26]. The formalization is given as follows.

Objective Function:

Min: $\alpha \times \frac{TT-ET_{min}}{ET_{max}-ET_{min}} + \beta \times \frac{TE-EE_{min}}{EE_{max}-EE_{min}} + \gamma \times \frac{DEV-ED_{min}}{ED_{max}-ED_{min}}$ (1)

Subject to:

$$S'_{j0M_{j0}} \geq R_j \quad j \in J \cup J' \tag{2}$$

$$S'_{j(s+1)M_{j(s+1)}} \geq S'_{jsM_{js}} + \sum_{p \in L} P_{jsM_{js}p} \times Z_{jsM_{js}p} \quad j \in J \cup J', s \in O_j, s > 0, p \in L \tag{3}$$

$$S'_{itM_{it}} \geq S'_{jsM_{js}} + \sum_{p \in L} P_{jsM_{js}p} \times Z_{jsM_{js}p} \tag{4}$$
$$j \in J \cup J', i \in J \cup J', j \neq i, s \in O_j, t \in O_i, M_{js} == M_{it}, p \in L, S_{jsm} \leq S_{itm}$$

$$TT = \sum_{j \in J} max\left(S'_{j(o_j-1)M_{j(o_j-1)}} + \sum_{p \in L} P_{j(o_j-1)M_{j(o_j-1)}p} \times Z_{j(o_j-1)M_{j(o_j-1)}p} - D_j, 0\right) \tag{5}$$

$$\sum_{p \in L} Z_{jsM_{js}p} = 1 \quad j \in J \cup J', s \in O_j \tag{6}$$

$$TE = \sum_{j \in J} \sum_{s \in O_j} \sum_{p \in L} Q_{jsM_{js}p} \times Z_{jsM_{js}p} \tag{7}$$

$$S'_{jsM_{js}} \geq RS \quad j \in J, s \in O_j, S_{jsM_{js}} + \sum_{p \in L} P_{jsM_{js}p} \times Z_{jsM_{js}p} \geq RS \tag{8}$$

$$S'_{jsM_{js}} = S_{jsM_{js}} \quad j \in J, s \in O_j S_{jsM_{js}} + \sum_{p \in L} P_{jsM_{js}p} \times Z_{jsM_{js}p} < RS \tag{9}$$

$$R_j \geq RS \quad j \in J' \tag{10}$$

$$DEV = \sum_{j \in J} wt_j \times max((S'_{j(o_j-1)M_{j(o_j-1)}} + \sum_{p \in L} P_{j(o_j-1)M_{j(o_j-1)}p} \times Z_{j(o_j-1)M_{j(o_j-1)}p}) - (S_{j(o_j-1)M_{j(o_j-1)}} + \sum_{q \in L} P_{j(o_j-1)M_{j(o_j-1)}q} \times Z_{j(o_j-1)M_{j(o_j-1)}q}), 0) \tag{11}$$

In this optimization problem, S'_{jsm} and Z_{jsmp} are the decision variables. A weighted additive utility function with three normalized objectives is described as (1) where all objectives can be assessed on the same scale. The linear weighted sum approach is taken for this application instead of the Pareto optimal solution for two reasons. Firstly, the most widely used parallel cellular model on GPUs is still immature for solving multi-objective problems where the main stream implementation manages a central Pareto front sequentially [26, 27]. Second, most of the existing literature on the multi-objective job shop scheduling problems adopt the linear weighted sum approach [28] whose computational complexity is relatively lower. Meanwhile, this design is suitable for dealing with large size problems in the dynamic scenario by obtaining an adequate renewed scheduling plan in a reasonable time.

Constraints (2) and (3) enforce that the first operation can only be processed after the release time while the others are authorized to start after its precedent one. The precedence for sequencing operations on machines is insured by constraint (4). Moreover, equation (5) defines the total tardiness of original jobs. As far as the energy cost, constraint (6) states each operation can only be handled by one machine with a fixed speed level whereas the total energy cost is given by equation (7). Finally, constraint (8), (9) and (10) impose the definition of rescheduling and equation (11) indicates the weighted finishing time deviation of the updated schedule from the original one.

4. Solving approach

4.1 Event-driven strategy

With the event-driven policy, rescheduling is triggered in response to an unexpected event that alters the current system status [18]. In the case of EDJSP, the unexpected event is considered as an arrival of urgent jobs. These jobs are requested to be processed as soon as possible even if the original schedule has started. Operations that are being executed need to be terminated and unfinished operations of original jobs must be rearranged in order to leave the machines available to firstly handle urgent jobs. Thus, new urgent arrival jobs are assigned to machines with the highest speed levels at the beginning when the rescheduling is triggered. If the amount of new urgent arrival jobs is not unique, they are scheduled as the regular JSP with the objective of minimizing the total tardiness. Unfinished operations of original jobs are considered at the next step according to the remaining spaces on machines. A dual heterogeneous island parallel GA on hybrid CPU–GPU frameworks is adapted to generate an adequate schedule for them in a limited time. The flow of the event-driven strategy is summarized as in Figure 2.

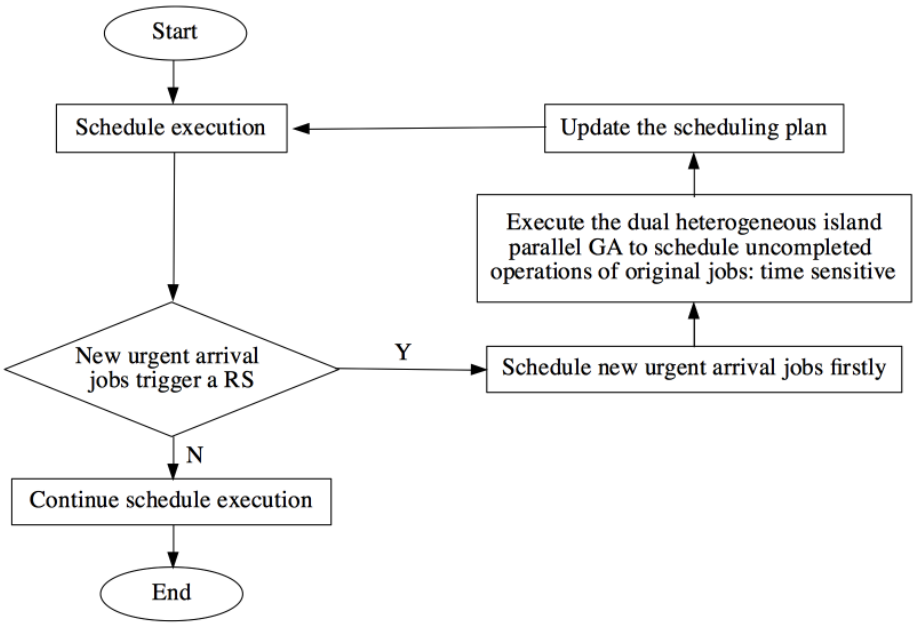

Fig.2. The flow of the event-driven strategy for the EDJSP

4.2 Dual heterogeneous island parallel GA on hybrid CPU–GPU frameworks

The general procedure of the dual heterogeneous island parallel GA on hybrid CPU–GPU frameworks is a further development from our previously designed parallel GA and its implementations to solve large scale flexible flow shop scheduling in static scenarios [30]. The algorithm divides the population into two islands. There is an identical amount of individuals on every island in which island A deals with the cellular GA [31] and island B deals with the classic GA [32]. At a certain point, a migration operation is executed to swap individuals between them. The procedure of the dual heterogeneous island parallel GA on hybrid CPU–GPU frameworks is shown in Fig.3. As far as the software and the hardware levels are concerned, four obvious advantages of this design are summarized as follows:

- Since the cellular GA and the classical GA get new search points in the exploring space using different mechanisms, this design enlarges the range of the searching process and decreases the probability that premature convergence occurs.

- Because of the independent evolution, individuals from heterogeneous islands obtain distinct characters from different solution range. Therefore, the performance of migration is enhanced.

- With respect to the underlying architectures, the cellular GA is designed to be entirely executed in parallel on GPUs while the classic GA can be partially parallelized on a multi-core CPU.

- To utilize the computing resources maximally, codes from the host and the device can also be executed simultaneously, in addition to the parallelization within GPUs or a multi-core CPU.

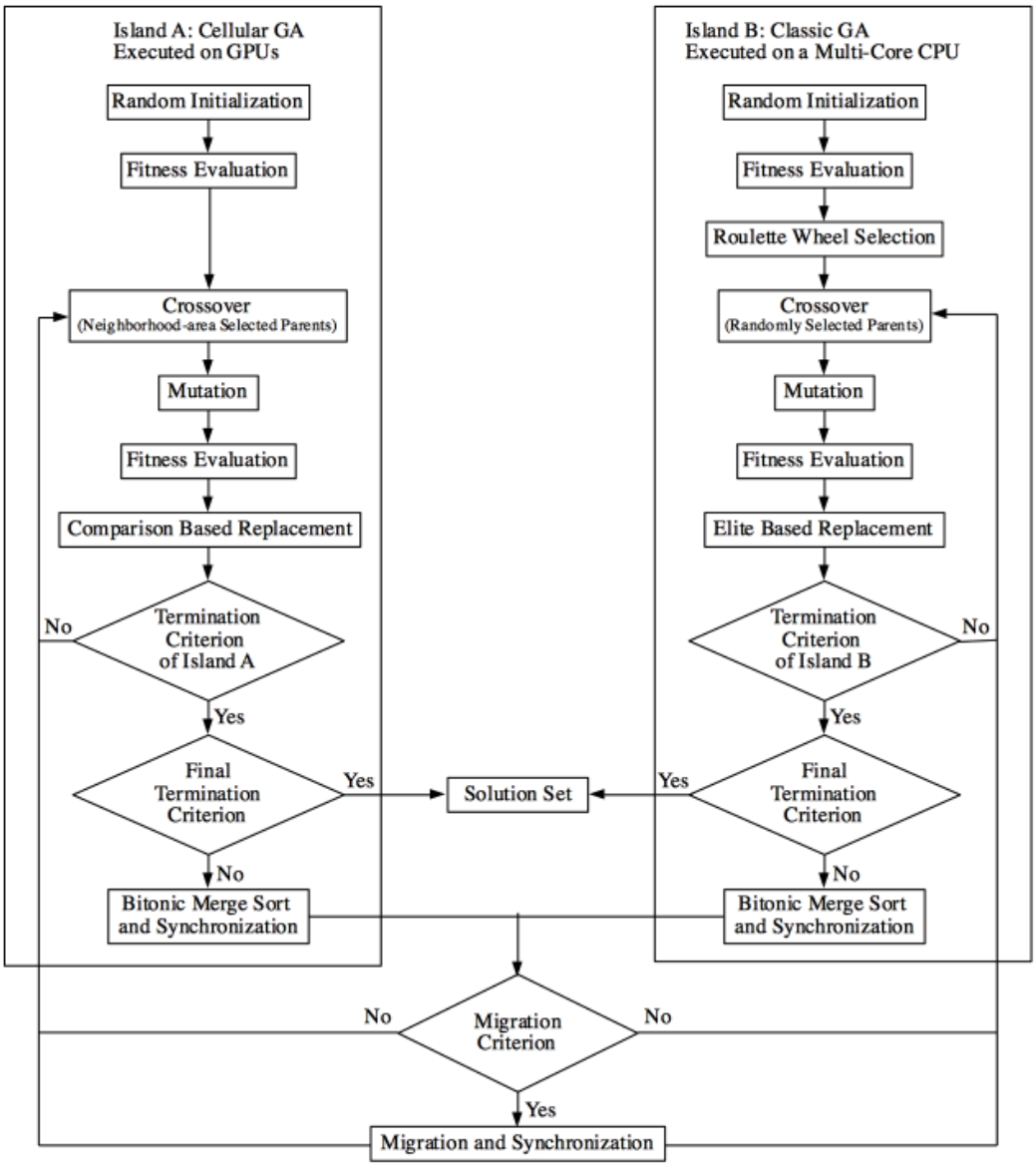

Fig.3. The procedure of the dual heterogeneous island parallel GA

The implementation for the cellular GA on island A is done with the Compute Unified Device Architecture (CUDA) for GPUs. All CUDA threads in a grid execute the same kernel function where a grid is organized as 2D array of blocks and each block is arranged as 2D array of threads [33]. The cellular GA maps individuals in a grid environment [34]. Therefore, it can be completely parallelized on CUDA and absolutely matches the underlying architectures. Since the texture memory of CUDA is optimized for 2D spatial locality [35], the overlapping communication region of the cellular GA is designed to be circled as in Fig. 4. At first, two parent individuals are selected from this region and their chromosomes are reassembled to produce a new offspring individual. After the mutation operation, this newly constituted individual substitutes the initial one only if its fitness value is better. Upon the fitness values, all individuals are finally ordered by the Bitonic-Merge sort [36], if the algorithm complies with the island termination condition but not with the final termination condition. The crossover, the replacement and the Bitonic-Merge sort are managed via the global memory whereas the mutation and the fitness evaluation are performed thought the local memory.

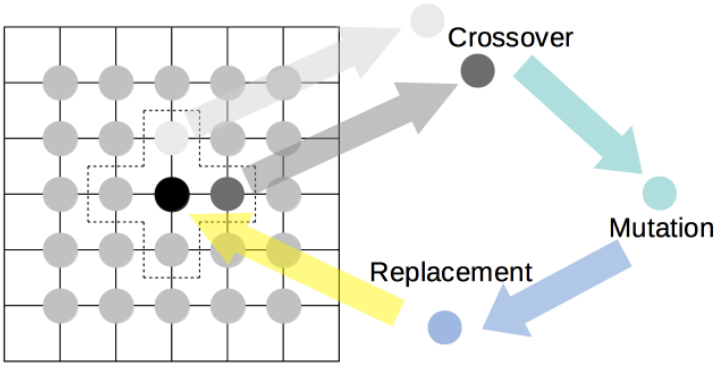

Fig.4 The procedure to generate new solutions by the cellular GA

Because of the high frequency use of the roulette wheel selection, the classic GA on island B utilizes it for selecting individuals from the population according to their fitness values. As the next step, two randomly paired parent individuals carry out the crossover and the new offspring individual implements the mutation. After these steps, the all-time best individual is maintained and is used to replace the worst individual in the current generation. At the end, all individuals are sorted under the same condition as the cellular GA on GPUs. To respect the original mechanism without requesting specific underlying architecture, a master-slave model is utilized to parallelize the classic GA. In this case, only the fitness evaluation and the Bitonic-Merge sort are performed on a multi-core CPU as slave nodes while the rest procedures are handled at the master side in sequential. When this procedure is implemented, the OpenMP [37] (Open Multi-Processing) is used for programming parallel threads in multi-core applications.

When a migration point is reached, individuals executed on GPUs are transferred to the CPU and the migration between the two islands is carried out by the CPU. To reduce the amount of factors required to be controlled manually, the migration's execution is decided by a migration threshold value θ. Moreover, the migration rate λ is formulated as

$$\lambda = \{1 - min\{\frac{fit_A}{fit_B, \frac{fit_B}{fit_A}}\} 0 \qquad \begin{matrix} 1 - min\{\frac{fit_A}{fit_B, \frac{fit_B}{fit_A\}} < \theta \\ 1 - min\{\frac{fit_A}{fit_B, \frac{fit_B}{fit_A\}} \geq \theta \end{matrix} \tag{12}$$

The migration is only executed when $1 - min\{\frac{fit_A}{fit_B, \frac{fit_B}{fit_A\}} < \theta$ where the λ percent individuals with the best fitness values are exchanged between the two islands to replace the λ percent individuals with the worst fitness values in the others. This mechanism helps to diffuse the best individuals efficiently while saving the execution time by avoiding useless information sharing.

4.3 Hybrid encoding representation

To solve the EDJSP, a modified operation-based encoding is adopted for representing the chromosomes. In terms of the schedule of original jobs, the chromosome contains two permutations: operation permutation X(k) (13) and speed level permutation Y(k) (14). X(k) utilizes the operation-based encoding where each job is represented by a natural number and each number is present as many times as the number of operations of the job it represents [38]. By scanning X(k) from left to right, the v_j^{th} occurrence of a job j refers to the v_j^{th} operation in the technological sequence of this job [39]. According to the example provided in [40], a feasible solution for a 3× 3 job shop is presented as [2, 1, 0, 0, 1, 2, 2, 1, 0] where 2 on the fifth gene position (indexed from

0) indicates the operation 1 (after the operation 0) of job 2 as it is the 1st occurrence (after the 0th occurrence) of number 2. Thus, X(k) can be translated to a list of ordered operations as $[o_{20}, o_{10}, o_{00}, o_{01}, o_{11}, o_{21}, o_{22}, o_{12}, o_{02}]$. Moreover, each element $y_a(k)$ shows the selected speed level of its related element $x_a(k)$ on the target machine.

$$X[k] = [\, x_0(k),\ x_1(k),\ \cdots, x_a(k), \cdots, x_{\sum_{j\in J} \quad o_j - 1}(k)\,] \tag{13}$$

where $x_a(k) \epsilon [0, n-1], u_j == o_j$.

$$Y[k] = [\, y_0(k),\ y_1(k),\ \cdots, y_a(k), \cdots, y_{\sum_{j\in J} \quad o_j - 1}(k)\,] \tag{14}$$

where $y_a(k) \epsilon [0, h-1]$.

To leave machines available to conduct new urgent arrival jobs firstly with the highest speed level and rearrange unfinished operations of original jobs, the chromosome of the updated schedule also includes an operation permutation $X'(k)$ (15) and a speed level permutation $Y'(k)$ (16). The initialization rule for both are shown in Algorithm 1. Moreover, the decoding rule is displayed in Algorithm 2.

$$X'(k) = [\, x'_0(k),\ x'_1(k),\ \cdots, x'_a(k), \cdots, x'_{\sum_{j\in J\cup J'} \quad o_j - 1}(k)\,] \tag{15}$$

where $x'_a(k) \epsilon [0, n + n' - 1], u_j == o_j$.

$$Y'(k) = [\, y'_0(k),\ y'_1(k),\ \cdots, y'_a(k), \cdots, y'_{\sum_{j\in J\cup J'} \quad o_j - 1}(k)\,] \tag{16}$$

where $y'_a(k) \epsilon [0, h-1]$.

Algorithm 1. The initialization rule of permutations $X'(k)$ and $Y'(k)$

a ← 0;
for $b \leftarrow 0\ to\ \sum_{j\in J} o_j - 1$ **do**
  j ← $x_b(0)$;
  s ← v_j;
  p ← $y_b(0)$;
  if $S_{jsM_{js}} + P_{jsM_{js}p} \times Z_{jsM_{js}p} \le RS$ **then**
    $x'_a(0) \leftarrow x_b(0)$;
    $y'_a(0) \leftarrow y_b(0)$;
    a ← a+1;
  end if
end for
r1← a;
for $c \leftarrow 0\ to\ \sum_{i\in J'} o_i - 1$ **do**
    $x'_a(0) \leftarrow x_c(0)$;
    $y'_a(0) \leftarrow y_c(0)$ $//highest\ speed\ level$;
  a ← a+1;
end for
r2← a;
for $a \leftarrow r2\ to\ \sum_{l\in J\cup J'} o_l - 1$ **do**
    initialize $x'_a(0)$ following the rule of operation-based encoding;
  initialize $y'_a(0)$ randomly in the range of machine speed level;
end for

Algorithm 2. The decoding rule

for $a \leftarrow r2\ to\ \sum_{j\in J\cup J'} o_j - 1$ **do**
  j ← $x'_a(k)$;
  s ← v_j;
  p ← $y'_a(k)$;
  if s == 0 **then**
    if $R_j \le RS$ **then**
    $S'_{jsM_{js}} \leftarrow RS$;
  else
    $S'_{jsM_{js}} \leftarrow R_j$;
  end if
else
  for $b \leftarrow 0\ to\ \sum_{i\in J\cup J'} o_i - 1$ **do**
    if i == j and $v_i == s-1$ **then**
      w ← $y'_b(k)$;
    end if
  end for
  if $S_{j(s-1)M_{j(s-1)}} + P_{j(s-1)M_{j(s-1)}w} \times Z_{j(s-1)M_{j(s-1)}w} \le RS$ **then**
    $S'_{jsM_{js}} \leftarrow RS$;
  else
    $S'_{jsM_{js}} \leftarrow S'_{j(s-1)M_{j(s-1)}} + P_{j(s-1)M_{j(s-1)}w} \times Z_{j(s-1)M_{j(s-1)}w}$;
  end if
end if
d← 0;
for $c \leftarrow r1\ to\ a$-1 **do**
  l ← $x'_c(k)$;
  t ← v_l;
  if $M_{js} == M_{lt}$ **then**
    $O'_{M_{js}}[d] \leftarrow o_{lt}$;
    d ← d+1;
  end if
end for
Sort elements in $O'_{M_{js}}[d]$ in ascending order by the starting time;
for $e \leftarrow 0\ to\ o'_m - 1$ **do**
  x ← job number in $O'_{M_{js}}[e]$;
  y ← operation number of job x in $O'_{M_{js}}[e]$;
  for $f \leftarrow 0\ to\ \sum_{z\in J\cup J'} o_z - 1$ **do**
    if z == x and $v_z == y$ **then**
      q ← $y'_f(k)$;
    end if
  end for
  if $[S'_{jsM_{js}},\ S'_{jsM_{js}} + P_{jsM_{js}p} \times Z_{jsM_{js}p}) \cap [S'_{xyM_{xy}},\ S'_{xyM_{xy}} + P_{xyM_{xyq}} \times Z_{xyM_{xyq}}) \neq \emptyset$ **then**
    $S'_{jsM_{js}} \leftarrow S'_{xyM_{xy}} + P_{xyM_{xyq}} \times Z_{xyM_{xyq}}$;
  end if

end for
end for

4.4 Crossover and mutation operators

To work with the modified operation-based encoding, the operation-based order crossover [6] is utilized as the crossover operator and works for genes in the chromosome within the range $[r2\ , \sum_{j\in J\cup J'} \quad o_j - 1]$. Firstly, it randomly chooses the same operations from two paired parents. The loci of chosen operations are preserved and copied to their own offspring. Afterwards, remaining operations are transmitted to the offspring of the other parent to fill the missing genes while their original orders are also kept. The crossover procedure for a 5×3 job shop example is shown in Fig.5 where job 0, job 1, job 2, job 3 are original jobs, job 4 is a new urgent arrival job and each machine has 3 speed levels. The integers in red indicates genes out of the range $[r2\ , \sum_{j\in J\cup J'} \quad o_j - 1]$ while the integers in blue mark the loci of randomly chosen operations.

Before crossover	Parent 1	$X'(k)$ = [2, 0, 1, 4, 4, 4, 0, 1, 3, 2, 2, 1, 3, 3, 0] $Y'(k)$ = [2, 1, 0, 2, 2, 2, 1, 2, 0, 1, 0, 2, 0, 1, 2]
	Parent 2	$X'(k)$ = [2, 0, 1, 4, 4, 4, 3, 1, 1, 3, 3, 0, 2, 0, 2] $Y'(k)$ = [2, 1, 0, 2, 2, 2, 2, 2, 0, 0, 2, 1, 0, 1, 1]
	↓	
After crossover	Offspring 1	$X'(k)$ = [2, 0, 1, 4, 4, 4, 3, 1, 3, 0, 2, 1, 2, 3, 0] $Y'(k)$ = [2, 1, 0, 2, 2, 2, 2, 2, 0, 1, 0, 2, 0, 1, 2]
	Offspring 2	$X'(k)$ = [2, 0, 1, 4, 4, 4, 0, 1, 1, 3, 3, 2, 3, 0, 2] $Y'(k)$ = [2, 1, 0, 2, 2, 2, 1, 2, 0, 0, 2, 1, 0, 1, 1]

Fig.5 An example of the operation-based order crossover

The swap mutation is used for $X'(k)$ where different arbitrary genes within the range $[r2, \sum_{j\in J\cup J'} o_j - 1]$ are chosen and exchange values. Concerning $Y'(k)$, unfixed number of genes are substituted by randomly generated values within the range, aside from the original ones. Following the above example, this procedure is illustrated in Fig.6 where genes in green illustrate the execution of mutation.

Before mutation	$X'(k)$ = [2, 0, 1, 4, 4, 4, 0, 1, 3, 2, 2, 1, 3, 3, 0] $Y'(k)$ = [2, 1, 0, 2, 2, 2, 1, 2, 0, 1, 0, 2, 0, 1, 2]
	↓
After mutation	$X'(k)$ = [2, 0, 1, 4, 4, 4, 0, 1, 0, 2, 2, 1, 3, 3, 3] $Y'(k)$ = [2, 1, 0, 2, 2, 2, 1, 0, 0, 1, 0, 2, 0, 1, 2]

Fig.6. An example of the mutation

5. Numerical Tests

Test 1 checks the efficiency and the effectiveness of the dual heterogeneous island parallel GA on hybrid CPU–GPU frameworks for solving the energy aware JSP while test 2 evaluates the performance of EDJSP by a case study. All the experiments have been made using the Intel Xeon E5640 CPU which has four CPU-cores, 2.67 GHz clock speed each and NVIDIA Tesla K40 with CUDA cores and 12 GB GDDR5 of global memory.

5.1 Evaluation

The energy aware JSP without taking into account new urgent arrival jobs is the first to be concerned. In this case, six large size problems are generated as in [41]. These instances are referred to as “easy problems” or “hard problems” with names EASY 20× 10, EASY 20× 20, EASY 50× 10, HARD 20× 10, HARD 20× 20 and HARD 50× 10. EASY 20× 10 and HARD 20× 10 are 20-job, 10-machine problems; EASY 20× 20 and HARD 20× 20 are 20-job, 20-machine problems; EASY 50× 10 and HARD 50× 10 are 50-job, 10-machine problems. Every job consists of the same amount of operations as the amount of machines, while one operation is always handled by a single machine. Moreover, every machine has 5 speed levels. As far as the easy problems are concerned, the machine procedure constraints for each job are generated randomly. As an alternative, the hard problems divide the machines into two sets. Each job must pass firstly through the first set, then through the second one. The ordering within the two sets of machines is generated randomly. The data relative to the experience is defined in Table 2.

Table 2. The data relative to the experience of the energy aware JSP

$P_{jsM_{js}p}$	U[1, 5]
$Q_{jsM_{js}p}$	$\delta \times P^2_{jsM_{js}p}$, where δ=U[2, 4]
R_j	U[0, $\underline{P}$], where $\underline{P} = \sum_j \quad (\sum_s \quad (\sum_p \quad P_{jsM_{js}p}/h)/o_j)$
D_j	$R_j + \underline{P}_j \times (1+\sigma)$, where σ=U[0, 2] and $\underline{P}_j =$ $\sum_s \quad (\sum_p \quad P_{jsM_{js}p}/h)$
α	1
β	1

To verify the performance of the proposed algorithm, we compare its solution quality and execution time with the parallel cellular GA on GPUs and the parallel classic GA on a multi-core CPU. For these tested three GAs, the population sizes are all kept as 512 (16×16×2) while each island's subpopulation size of the dual heterogeneous island parallel GA on hybrid CPU–GPU frameworks is 256 (16×16). The final termination criterion is set as 2000 generations. Moreover, the results shown in Fig.7, Fig.8, Fig.9, Fig.10, Fig.11, Fig.12, Table 5 and Table 6 are obtained by 30 independent runs while the results displayed in Table 7 are the average values of 5 runs. Since the parameter configuration has a huge impact to the performance of algorithms, the Taguchi method [42] is used to calibrate the parameters of the tested GAs. As most common optimality criteria of shop scheduling problems are about minimization, the signal to noise ratio (S/N) of the Taguchi method used to assess the performance in our case is calculated as:

$$\text{S/N ratio} = -10 \times (sum(F^2)/N) \tag{17}$$

Table 3. The parameters and their levels.

GAs	Parameters	Parameter Level		
		Level 1	Level 2	Level 3
Dual heterogeneous island parallel GA	P_{cac}	0.6	0.7	0.8
	P_{cam}	0.03	0.06	0.09
	P_{cec}	0.6	0.7	0.8
	P_{cem}	0.03	0.06	0.09
	θ	0.9	0.95	1.00
	φ	100	200	300
Parallel classic GA	P_{cac}	0.6	0.7	0.8
	P_{cam}	0.03	0.06	0.09
Parallel cellular GA	P_{cec}	0.6	0.7	0.8
	P_{cem}	0.03	0.06	0.09

As the migration of the proposed GA is carried out by the CPU in which individuals executed on GPUs are transferred to the CPU at this point, its performance may be weakened because of the frequent data exchange. Therefore, we need also test the migration policy execution gap for the dual heterogeneous island parallel GA on hybrid CPU–GPU frameworks, in addition to the crossover rate, the mutation rate and the migration threshold value. The parameters and their levels are given in Table 3. The Minitab software [43] is used to obtain the S/N ratios and the standard deviations in the Taguchi method for each GA. The L_{27} design is selected for the dual heterogeneous island parallel GA while the L_9 is selected for the parallel classic GA and the parallel cellular GA. The orthogonal array of each design is presented in Table 4 and Table 5 respectively. Regarding the S/N ratios of three GAs displayed in Fig.7, Fig.9 and Fig.11 and the standard deviations presented in Fig.8, Fig.10 and Fig.12 separately, we select their parameters levels as shaded in Table 3.

Table 4. The orthogonal array L_{27}

Run order	P_{cac}	P_{cam}	P_{cec}	P_{cem}	θ	φ
1	1	1	1	1	1	1
2	1	1	1	1	2	2
3	1	1	1	1	3	3
4	1	2	2	2	1	1
5	1	2	2	2	2	2
6	1	2	2	2	3	3
7	1	3	3	3	1	1
8	1	3	3	3	2	2
9	1	3	3	3	3	3
10	2	1	2	3	1	2
11	2	1	2	3	2	3
12	2	1	2	3	3	1
13	2	2	3	1	1	2
14	2	2	3	1	2	3
15	2	2	3	1	3	1
16	2	3	1	2	1	2
17	2	3	1	2	2	3
18	2	3	1	2	3	1
19	3	1	3	2	1	3
20	3	1	3	2	2	1
21	3	1	3	2	3	2
22	3	2	1	3	1	3
23	3	2	1	3	2	1
24	3	2	1	3	3	2
25	3	3	2	1	1	3
26	3	3	2	1	2	1
27	3	3	2	1	3	2

Table 5. The orthogonal array L_9

Run order	P_{cac}/P_{cec}	P_{cam}/P_{cem}
1	1	1
2	1	2
3	1	3
4	2	1
5	2	2
6	2	3
7	3	1
8	3	2
9	3	3

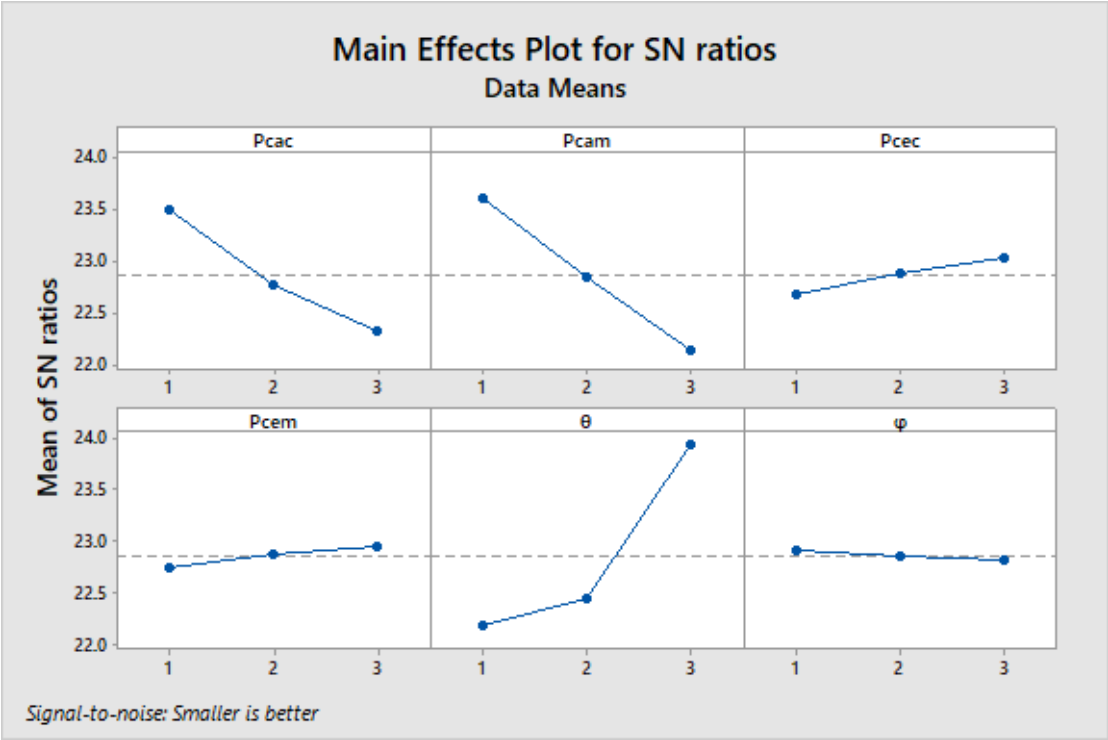

Fig.7. The S/N ratio of the dual heterogeneous island parallel GA

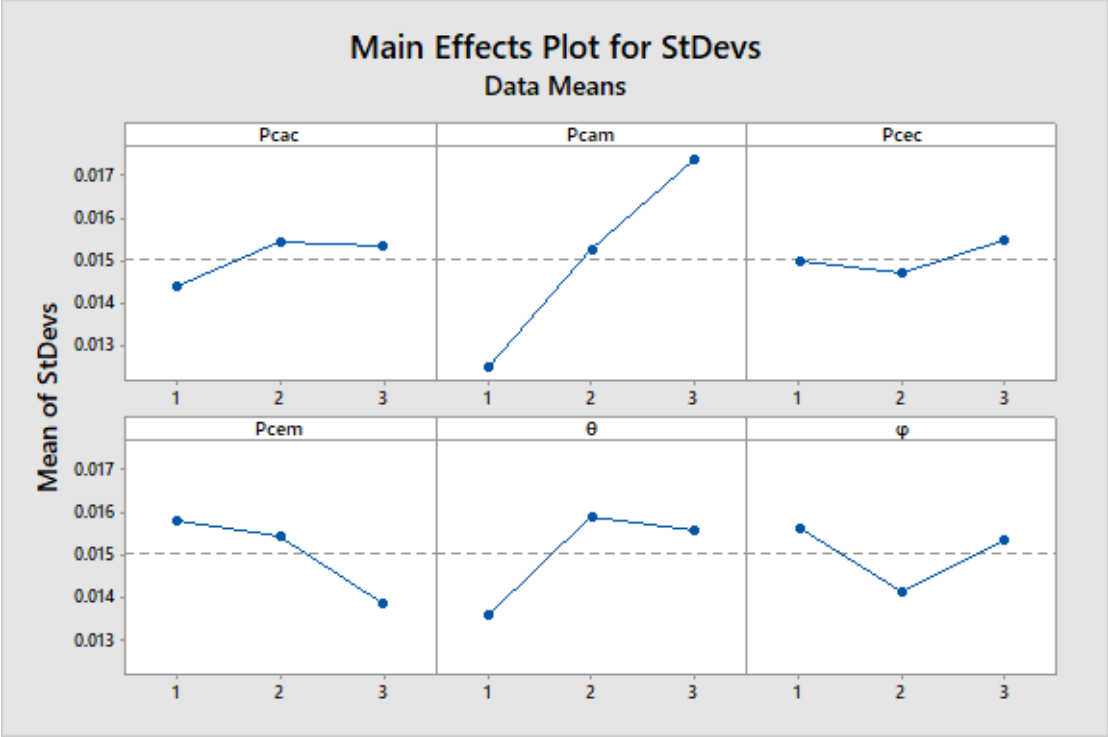

Fig.8. The standard deviation of the dual heterogeneous island parallel GA

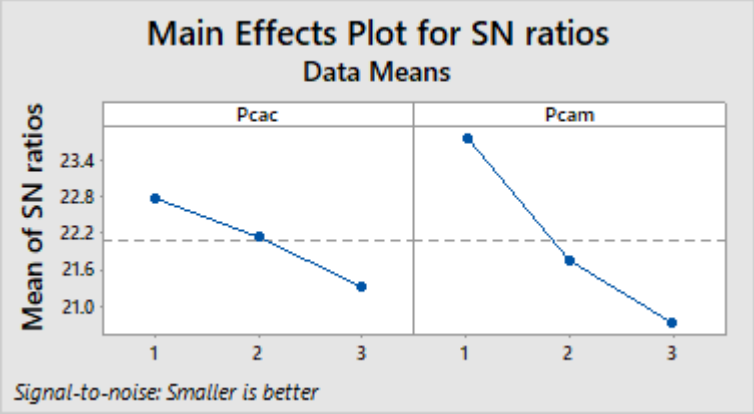

Fig.9. The S/N ratio of the parallel classic GA

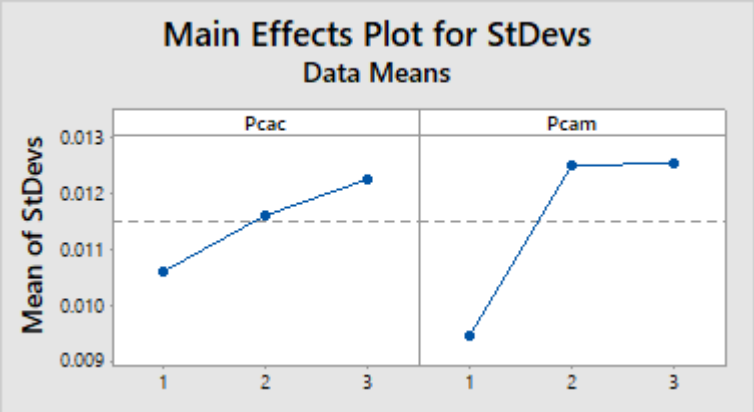

Fig.10. The standard deviation of the parallel classic GA

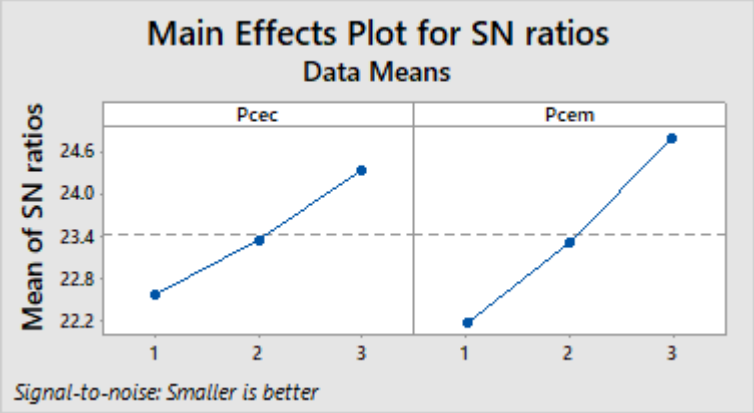

Fig.11. The S/N ratio of the parallel cellular GA

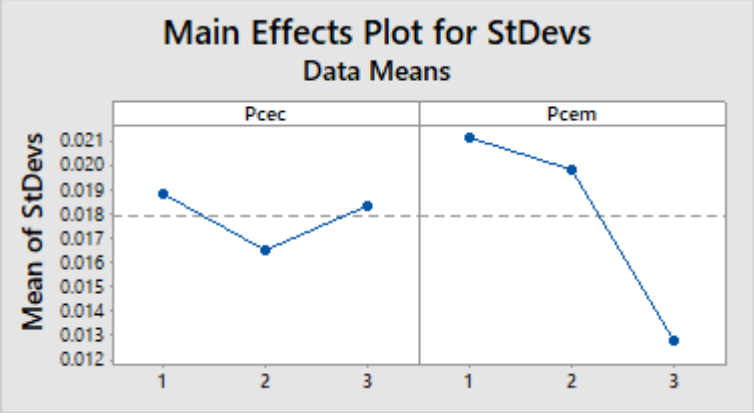

Fig.12. The standard deviation of the parallel cellular GA

Since the decentralized population in the parallel cellular GA allows to keep the population's diversity for longer [44], it works as strongly as the parallel classic GA and even defeats the parallel classic GA for half of the cases as shown in Table 6. Because of the separated evolution and the enhanced migration, the dual heterogeneous island parallel GA integrates the advantages from the parallel cellular GA and the parallel classic GA. Therefore, it can always get the best performance for all tested problems with the average value. To confirm this efficiency, the Wilcoxon signed ranks test [45] is utilized to compare the performance of the dual heterogeneous island parallel GA with the other two considered parallel GAs. Table 7 displays the R^-, R^+ and p-values computed by SPSS [46] where the dual heterogeneous island parallel GA shows an improvement over the parallel cellular GA for instances EASY 20× 10, EASY 50× 10, HARD 20× 10, HARD 20× 20 and HARD 50× 10, over the parallel classic GA for instances EASY 20× 10, EASY 20× 20, EASY 50× 10, HARD 20× 10 and HARD 20× 20 when the significance level equals to 0.1.

Table 6. The solutions' quality comparison

Problems	Parallel heterogeneous GA		Parallel cellular GA		Parallel classic GA	
	Average	Best	Average	Best	Average	Best
EASY 20× 10	**0.0481**	**0.0224**	0.0551	0.0308	0.0623	0.0490
EASY 20× 20	**0.0901**	0.0568	0.0911	**0.0468**	0.1007	0.0831
EASY 50× 10	**0.0585**	**0.0110**	0.1290	0.0378	0.0643	0.0292
HARD 20× 10	**0.0412**	0.0258	0.0441	**0.0227**	0.0747	0.0499
HARD 20× 20	**0.1025**	**0.0421**	0.1750	0.0916	0.1219	0.0849
HARD 50× 10	**0.0706**	0.0236	0.0932	**0.0075**	0.0709	0.0291

Table 7. The Wilcoxon signed ranks test results

Comparison	Problems	R^-	R^+	p-value
parallel heterogeneous GA versus parallel cellular GA	EASY 20× 10	136.00	329.00	0.047
	EASY 20× 20	204.00	261.00	>0.1
	EASY 50× 10	18.00	447.00	0.000
	HARD 20× 10	151.00	314.00	0.094
	HARD 20× 20	14.00	451.00	0.000
	HARD 50× 10	126.00	339.00	0.028
parallel heterogeneous GA versus parallel classic GA	EASY 20× 10	44.00	421.00	0.000
	EASY 20× 20	109.00	359.00	0.011
	EASY 50× 10	137.00	328.00	0.049
	HARD 20× 10	0.00	465.00	0.000
	HARD 20× 20	108.00	357.00	0.010
	HARD 50× 10	225.00	240.00	>0.1

R^-: value of the objective function got by the parallel heterogeneous GA > value of the objective function got by the parallel cellular GA (parallel classic GA)
R^+: value of the objective function got by the parallel heterogeneous GA < value of the objective function got by the parallel cellular GA (parallel classic GA)

The execution time of three parallel GAs with different population sizes are shown in Table 8. Because of the simultaneous execution on both sides, the dual heterogeneous island GA on the hybrid platform overcomes the parallel cellular GA on GPUs and the parallel classic GA on a multi-core CPU in most cases. This phenomenon is even more remarkable when the difference of the execution time on two islands is smaller. However, the advantage from the hybrid platform may be reduced because the overall performance is limited to the island who takes longer execution time. Therefore, it indicates the significance of computation capability balance between the multi-core CPU and the GPUs when the dual heterogeneous island GA is implemented. For some extreme situations, the weak node may perform as a bottleneck and decreases the global effectiveness.

Table 8. The execution time comparison

Problems	Population Size	Parallel heterogeneous GA (island of parallel cellular GA, island of parallel classic GA)	Parallel cellular GA	Parallel classic GA
EASY 20× 10	16×16×2	**475 s** (474 s, 233 s)	504 s	657 s
	32×32×2	**966 s** (802 s, 936 s)	1185 s	2248 s
	64×64×2	3927 s (2058 s, 3800 s)	**3615 s**	8321 s
EASY 20× 20	16×16×2	1731 s (1730 s, 826 s)	2106 s	**1602 s**
	32×32×2	**3556 s** (3555 s, 3346 s)	5530 s	6428 s
	64×64×2	**14060 s** (9587 s, 13472 s)	16864 s	28166 s
EASY 50× 10	16×16×2	**3082 s** (3081 s, 1342 s)	3408 s	3239 s
	32×32×2	**5600 s** (5467 s, 5405 s)	8350 s	12433 s
	64×64×2	**22627 s** (14162 s, 21748 s)	24776 s	48073 s
HARD 20× 10	16×16×2	**472 s** (472 s, 239 s)	507 s	660 s
	32×32×2	**986 s** (806 s, 955 s)	1183 s	1950 s
	64×64×2	3986 s (2046 s, 3859 s)	**3627 s**	8732 s
HARD 20× 20	16×16×2	**1729 s** (1729 s, 837 s)	2097 s	1935 s
	32×32×2	**3511 s** (3510 s, 3353 s)	5426 s	7437 s
	64×64×2	**14088 s** (9411 s, 13509 s)	16323 s	28508 s
HARD 50× 10	16×16×2	**3048 s** (3048 s, 1357 s)	3428 s	3268 s
	32×32×2	**5641 s** (5506 s, 5444 s)	8454 s	12121 s
	64×64×2	**22676 s** (14259 s, 21795 s)	25046 s	48427 s

5.2 Case study

A modified job shop instance incorporating machine speed scaling and new urgent arrival jobs is developed based on the well know 10×10 problem (10 jobs, 10 machines) from Muth and Thompson [47] (MT10) as a case study. There are 10 original jobs and 3 new urgent arrival jobs. Each machine has 5 speed levels. New urgent jobs arrive around 30% of the makespan of the original schedule. The operation sequence of original jobs and their processing times on target machine at speed level 0 are collected from MT10. On the other hand, the values for new urgent arrival jobs are generated following the rule of "hard problems" in subsection 5.1 evaluation. The values of energy cost at level 0 is set as $Q_{jsM_{js}0} = \delta \times P^2_{jsM_{js}0}$, where δ=U[2, 4]. The release times (R_j) of original jobs are fixed as 0 while the due times are generated as $D_j = \underline{P_j} \times (1 + \sigma)$, where σ=U[0, 2] and $\underline{P_j} = \sum_s \quad (\sum_p \quad P_{jsM_{js}p}/h)$. Concerning the importance weight of original jobs, we make $wt_0 = wt_1 = 4$, $wt_j = 2$ for j = 2, 3, …, 7 and $wt_8 = wt_9 = 1$. All details are shown in Table 9. Moreover, the processing time and the energy cost when operation s of job j handled by target machine m at different levels is defined as P_{jsmp}=$P_{jsm0} \times V_p$ and Q_{jsmp}=$Q_{jsm0} \div V_p$, respectively, where $V = (1, 1.3, 1.55, 1.75, 2.1)$. Finally, we keep the values of α, β equal to 1 while a very large constant is assigned to γ which indicates the importance of the schedule repair strategy.

The best-found solution of the original schedule is shown by the Gantt chart in Fig.13. Since new urgent jobs arrive at time 600, all operations are being operated at this moment need to be canceled and leave machines available for processing them firstly. In this case, some machines are occupied at some periods after scheduling new urgent arrival jobs. Therefore, unfinished operations of

original jobs are rearranged to make use of machines only when they are idle. By implementing the schedule repair strategy, the best-found solution illustrated by the Gantt chart of the updated schedule in Fig.14 presents that the processing time of some operations is obviously decreased. As a result, most original jobs' finishing time are only delayed slightly which is confirmed by the details displayed in Table 10.

In addition to the M10, we have extended another four classic cases from the literature to test the relationship among the three objectives of the EDJSP. The problems ABZ5 and ABZ7 are two problems from [48]. The problems LA35 and LA40 are two problems from [49]. The operation sequence of these jobs and their processing times on target machines are treated as original jobs at speed level 0 in the EDJSP. The importance weights of original jobs are randomly drawn integers from the interval [1, 4]. The amount of new urgent arrival jobs is an integer generated randomly from U [1, 10] while their arriving time is set by a random value from a uniform distribution on the interval [0, the makespan of the original schedule]. Moreover, the other settings are kept the same as the MT10 based EDJSP.

Table 9. The case data of the M10 based EDJSP

Jobs	M_{js} $P_{jsM_{js}0}$ $Q_{jsM_{js}0}$										wt_j	R_j	D_j
	0,	1	2	3	4	5	6	7	8	9			
J_0	29	78	9	36	49	11	62	56	44	21	4	0	787
	2732	22255	184	3729	8905	261	7849	10985	7219	1151			
	0	2	4	9	3	1	6	5	7	8			
J_1	43	90	75	11	69	28	46	46	72	30	4	0	1096
	5859	25571	16498	396	11116	2999	4796	5571	16324	3438			
	1	0	3	2	8	5	7	6	9	4			
J_2	91	85	39	74	90	10	12	89	45	33	2	0	1587
	30407	24102	5696	11450	19091	315	423	19723	4446	3161			
	1	2	0	4	6	8	7	3	9	5			
J_3	81	95	71	99	9	52	85	98	22	43	2	0	2050
	17491	27291	19422	33401	237	8060	21768	36629	1711	6783			
	2	0	1	5	3	4	8	7	9	6			
J_4	14	6	22	61	26	69	21	49	72	53	2	0	1450
	606	126	1546	12666	2229	10107	1711	6160	12115	6022			
	2	1	5	3	8	9	0	6	4	7			
J_5	84	2	52	95	48	72	47	65	6	25	2	0	1945
	27497	15	9080	30657	6690	16749	7013	13934	86	1507			
	1	0	3	2	6	5	9	8	7	4			
J_6	46	37	61	13	32	21	32	89	30	55	2	0	1415
	5410	2748	14764	596	3033	1042	2920	30266	3340	11800			
	2	0	1	5	4	6	8	9	7	3			
J_7	31	86	46	74	32	88	19	48	36	79	2	0	1005
	2720	15213	5903	14670	3078	16246	1198	5121	4872	19509			
	0	1	3	5	2	9	6	7	4	8			
J_8	76	69	76	51	85	11	40	89	26	74	1	0	1265
	20250	17948	12094	7397	18308	289	5980	20515	1459	21613			
	1	0	2	6	8	9	5	3	4	7			
J_9	85	13	61	7	64	76	47	52	90	45	1	0	2182
	23242	429	12595	141	14008	17143	8648	9555	16289	6382			
	2	1	0	4	3	8	9	7	5	6			
J_{10}	16	58	22	24	53	9	57	63	92	43		600	879
	831	12305	1099	1657	10418	175	6634	13903	31562	4829			
	3	1	4	0	2	7	9	6	8	5			
J_{11}	6	48	14	66	24	2	85	73	19	99		600	859
	114	7273	574	14278	1344	15	16379	14031	1136	37449			
	4	2	0	1	3	5	9	8	6	7			
J_{12}	99	90	63	14	31	27	15	2	51	33		600	806
	35989	27021	14863	409	2265	2298	662	9	5711	3161			

Table 10. The original jobs' finishing time comparison of the M10 based EDJSP

	Job 0	Job 1	Job 2	Job 3	Job 4	Job 5	Job 6	Job 7	Job 8	Job 9
Original Schedule	632.05	1091.80	1555.15	1817.90	1485.05	1535.80	1390.05	987.45	1431.40	1838.65
Updated Schedule	688.10	1092.30	1579.25	1824.40	1472.20	1331.65	1455.85	990.00	1991.45	1851.20
Difference	56.05	0.5	24.1	6.5	0	0	65.8	2.55	560.05	12.55

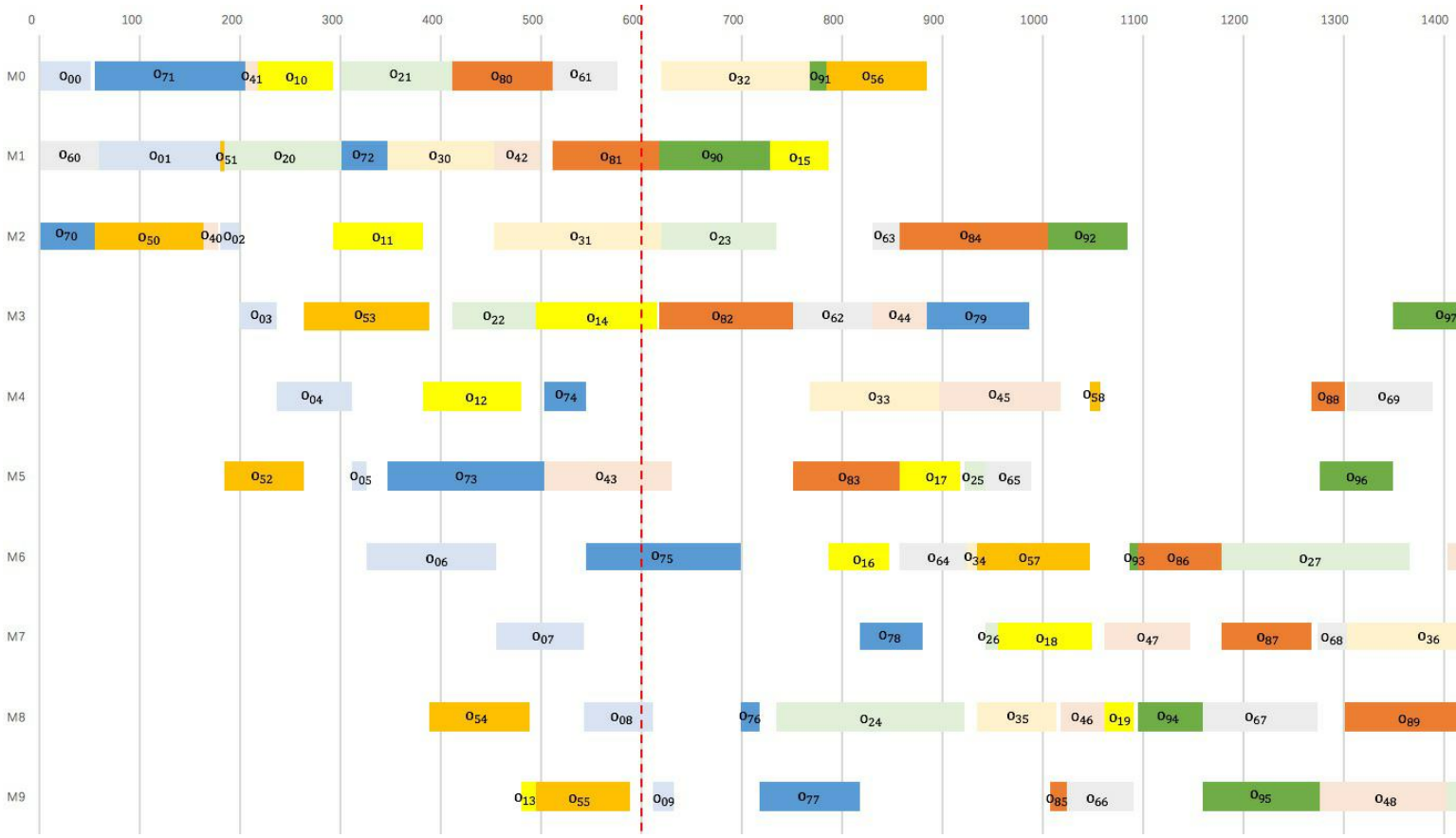

Fig. 13. The Gantt chart of the best-found solution of the original schedule for the M10 based EDJSP

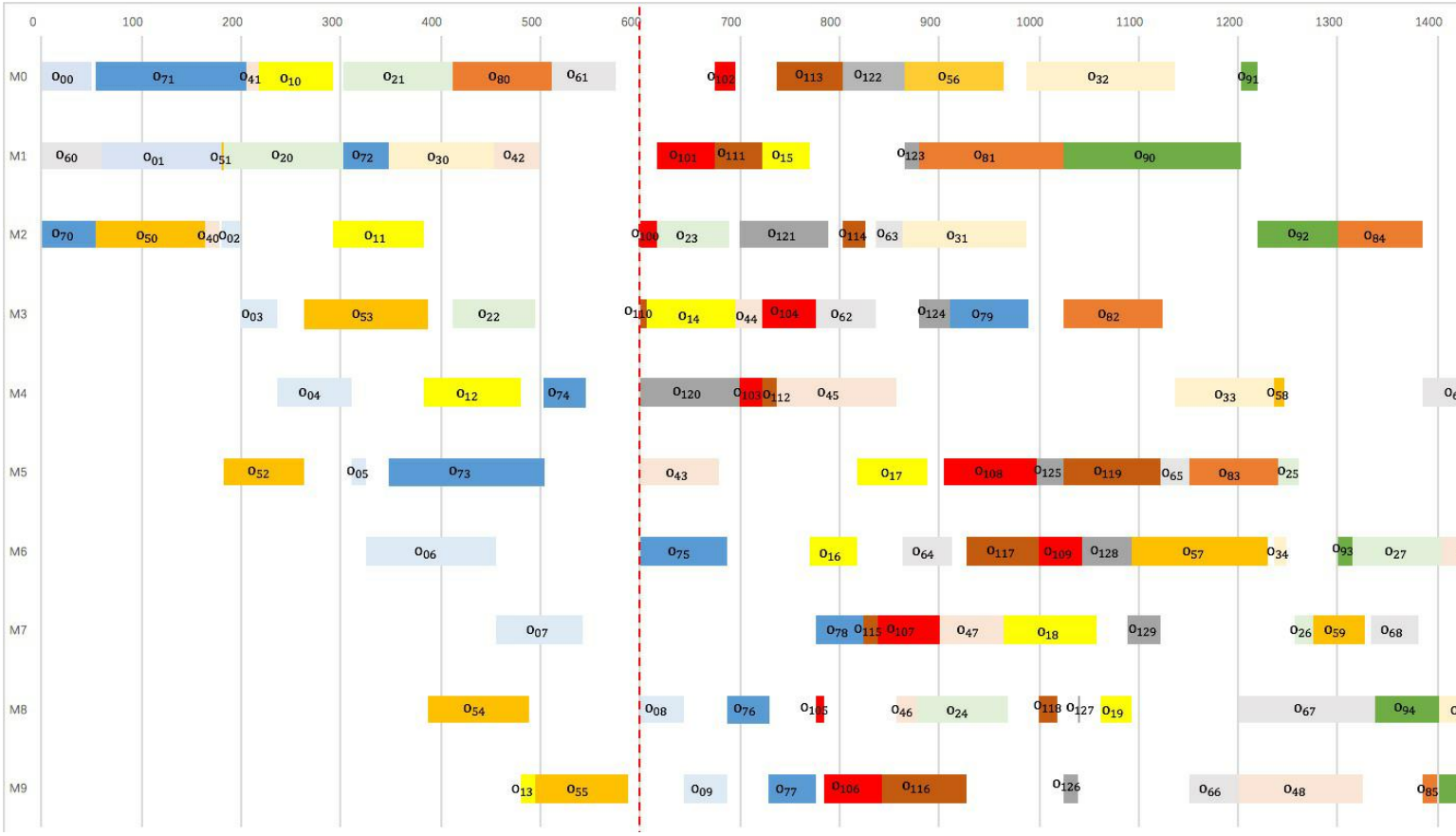

Fig. 14. The Gantt chart of the best-found solution of the updated schedule for the M10 based EDJSP

Table 11. The relationship among three objectives

Problems	NO. of machines	NO. of jobs	Weight of each normalized objective function			$\frac{TT - ET_{min}}{ET_{max} - ET_{min}}$
			α	β	γ	
MT10	10	10	100	10	1	0.0469
			100	1	10	0.0464
			10	100	1	0.1864
			1	100	10	0.1813
			10	1	100	0.0460
			1	10	100	0.0520
				Standard deviation		0.0702
ABZ5	10	10	100	10	1	0.0845
			100	1	10	0.0794
			10	100	1	0.2169
			1	100	10	0.2292
			10	1	100	0.1043
			1	10	100	0.1098
				Standard deviation		0.0674
ABZ7	15	20	100	10	1	0.0937
			100	1	10	0.0904
			10	100	1	0.2049
			1	100	10	0.2199
			10	1	100	0.1100
			1	10	100	0.1136
				Standard deviation		0.0579
LA35	10	30	100	10	1	0.2354
			100	1	10	0.2363
			10	100	1	0.3075
			1	100	10	0.2973
			10	1	100	0.2405
			1	10	100	0.2412
				Standard deviation		0.0333
LA40	15	15	100	10	1	0.1814
			100	1	10	0.1800
			10	100	1	0.2832
			1	100	10	0.2852
			10	1	100	0.1903
			1	10	100	0.1928
				Standard deviation		0.0508

Because of the relationship among the total tardiness, the total energy cost and the disruption to the original schedule, the decision maker can achieve their preference through controlling the importance weight of each normalized objective function. The dual heterogeneous island parallel GA was run 30 times for the above mentioned five EDJSP cases with different settings of α, β, γ and the average results are displayed in Table 11. It can be observed that the third objective is the most sensitive one to the importance weight in all cases while the second objective is the least. Thus, in industrial practice, decision makers are suggested to pay more attention to minimize the values of the disruption to the original schedule and the total tardiness while limiting the total energy cost in a reasonable range. Moreover, three different scenarios are analyzed underneath corresponding to different combinations of α, β and γ.

Scenario 1: When the decision-maker only wants to consider the minimum total tardiness, the importance weights can be set to α=100, β=1 and γ=10. The disruption to the original schedule is the most sensitive one among three objectives. For the problems ABZ7 and LA35, its standard deviation is more than the double of the standard deviation of the total energy cost. Therefore, when there is no specific preference between the disruption to the original schedule and the total energy cost, the latter one can be neglected.

Scenario 2: When the decision-maker only wants to consider the minimum total energy cost, the importance weights can be set to α=1, β=100 and γ=10. Although the gap of standard deviation between the total tardiness and the disruption to the original schedule is not so significant, the difference may still be obvious for some cases as the problem LA35. Thus, after the minimization of the total energy cost, the decision makers are advised to control the disruption to the original schedule prior to the total tardiness.

Scenario 3: When the decision-maker only wants to consider the minimum disruption to the original schedule, the importance weights can be set to α=10, β=1 and γ=100. The total tardiness is also very sensitive to the importance weight. For the problems MT10 and ABZ7, its standard deviation is quite close to the standard deviation of the disruption to the original schedule. Hence, a relative larger weight should be assigned to the total tardiness rather than the total energy cost in this case.

6. Conclusions and Future Works

In this paper, an investigation into minimizing the total tardiness, the total energy cost and the disruption to the original schedule in the job shop with new urgent arrival jobs was studied. To provide an adequate renewed scheduling plan in a reasonable time, a dual heterogeneous island parallel GA executed simultaneously on different parallel platforms was adopted. This design consisted of a cellular GA on GPUs and a classic GA on a multi-core CPU which was totally compliant with the underlying architectures of two-level parallelization. To improve the performance of the utilized GAs, the Taguchi method was used to calibrate their parameters firstly in the evaluation. Afterwards, the proposed method presented that it could obtain better solutions for solving six large size energy aware JSP through the integration of advantages from two different islands. In the meantime, it decreased the execution time obviously because of the simultaneously parallel execution on the host and the device while indicating the significance of computation capability balance between two sides. Concerning the EDJSP in the case study, the best-found solution of the updated schedule was shown by the Gantt chart. Compared with the original schedule, the processing time of some operations was significantly decreased. Finally, an experiment was carried to analyze the relationship among three objectives with different

importance weights. After a discussion around three scenarios, some useful suggestions were made for industrial practice.

In the future, the Pareto optimal solution will be considered to solve the dynamic energy aware shop scheduling problems. It can be easily found in the literature that the Pareto optimal solution is a common approach to deal with the multi-objective optimization problems, apart from the linear combination method. The ranking and crowding mechanisms from the NSGA II [50] are the mostly used strategy in the area. However, the non-dominated set of solutions managed during the optimization procedure is generally structured as the centralized Pareto front [26, 27]. This strategy is hard to achieve parallelism in the population level. On the other hand, any partial parallelization on GPUs may lead to frequent communication overheads and offset the effectiveness. Therefore, developing a fine-grained Pareto based approach mapping onto GPUs underlying architecture and achieving the full parallelization deserves further study.

Acknowledgement

This work is supported in part by the Japan Society for the Promotion of Science. Moreover, Didier El Baz is grateful to 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.

Reference

[1].Paolucci, M., Anghinolfi, D., & Tonelli, F. (2017). Facing energy-aware scheduling: a multi-objective extension of a scheduling support system for improving energy efficiency in a moulding industry. Soft Computing, 21(13), 3687-3698.

[2].Pach, C., Berger, T., Sallez, Y., Bonte, T., Adam, E., & Trentesaux, D. (2014). Reactive and energy-aware scheduling of flexible manufacturing systems using potential fields. Computers in Industry, 65(3), 434-448.

[3].Fang, K., Uhan, N., Zhao, F., & Sutherland, J. W. (2011). A new shop scheduling approach in support of sustainable manufacturing. In Glocalized solutions for sustainability in manufacturing (pp. 305-310). Springer, Berlin, Heidelberg.

[4]. Bruzzone A A G, Anghinolfi D, Paolucci M, et al. Energy-aware scheduling for improving manufacturing process sustainability: A mathematical model for flexible flow shops[J]. CIRP Annals-Manufacturing Technology, 2012, 61(1): 459-462.

[5].Xu F, Weng W, Fujimura S. Energy-Efficient Scheduling for Flexible Flow Shops by Using MIP[C]//IIE Annual Conference. Proceedings. Institute of Industrial and Systems Engineers (IISE), 2014: 1040.

[6].Y. Liu, H. Dong, N. Lohse, S. Petrovic, N. Gindy, An investigation into minimising total energy consumption and total weighted tardiness in job shops, Journal of Cleaner Production 65 (2014) 87-96.

[7].Q. Yi, C. Li, Y. Tang, Q. Wang, A new operational framework to job shop scheduling for reducing carbon emissions, in: Automation Science and Engineering (CASE), 2012 IEEE International Conference, IEEE, 2012, pp. 58-63.

[8].M. Dai, D. Tang, A. Giret, M. A. Salido, W. D. Li, Energy-efficient scheduling for a flexible flow shop using an improved genetic-simulated annealing algorithm, Robotics and Computer-Integrated Manufacturing 29 (5) (2013) 418-429.

[9].Tang D, Dai M, Salido M A, et al. Energy-efficient dynamic scheduling for a flexible flow shop using an improved particle swarm optimization[J]. Computers in Industry, 2016, 81: 82-95.

[10].Zhang L, Li X, Gao L, et al. Dynamic rescheduling in FMS that is simultaneously considering energy consumption and schedule efficiency[J]. The International Journal of Advanced Manufacturing Technology, 2016, 87(5-8): 1387-1399.
[11].Luo, J., Fujimura, S., El Baz, D., & Plazolles, B. (2018). GPU based parallel genetic algorithm for solving an energy efficient dynamic flexible flow shop scheduling problem. Journal of Parallel and Distributed Computing.
[12].Dabah, A., Bendjoudi, A., AitZai, A., El Baz, D., & Taboudjemat, N. N. (2018). Hybrid multi-core CPU and GPU-based B&B approaches for the blocking job shop scheduling problem. Journal of Parallel and Distributed Computing, 117, 73-86.
[13].Spanos, A. C., Ponis, S. T., Tatsiopoulos, I. P., Christou, I. T., & Rokou, E. (2014). A new hybrid parallel genetic algorithm for the job-shop scheduling problem. International Transactions in Operational Research, 21(3), 479-499.
[14].Zajıcek, T., & Sucha, P. (2011). Accelerating a Flow Shop Scheduling Algorithm on the GPU. eraerts, 143.
[15].Somani, A., & Singh, D. P. (2014, August). Parallel Genetic Algorithm for solving Job-Shop Scheduling Problem Using Topological sort. In Advances in Engineering and Technology Research (ICAETR), 2014 International Conference on (pp. 1-8). IEEE.
[16]. Meng, L., Zhang, C., Shao, X., & Ren, Y. (2019). MILP models for energy-aware flexible job shop scheduling problem. Journal of Cleaner Production, 210, 710-723.
[17]. He, Y., Liu, F., Cao, H. J., & Li, C. B. (2005). A bi-objective model for job-shop scheduling problem to minimize both energy consumption and makespan. Journal of Central South University of Technology, 12(2), 167-171.
[18].Ouelhadj, D., & Petrovic, S. (2009). A survey of dynamic scheduling in manufacturing systems. Journal of scheduling, 12(4), 417.
[19].Le, C. V., & Pang, C. K. (2013). Fast reactive scheduling to minimize tardiness penalty and energy cost under power consumption uncertainties. Computers & Industrial Engineering, 66(2), 406-417.
[20].Zeng, L., Zou, F., Xu, X., & Gao, Z. (2009, June). Dynamic scheduling of multi-task for hybrid flow-shop based on energy consumption. In Information and Automation, 2009. ICIA'09. International Conference on (pp. 478-482). IEEE.
[21]. Hawick, K. A., Leist, A., & Playne, D. P. (2010). Mixing multi-core CPUs and GPUs for scientific simulation software.
[22]. Hossam, M. A., Ebied, H. M., & Abdel-Aziz, M. H. (2013, November). Hybrid cluster of multicore CPUs and GPUs for accelerating hyperspectral image hierarchical segmentation. In 2013 8th International Conference on Computer Engineering & Systems (ICCES) (pp. 262-267). IEEE.
[23].Lenstra, J. K., Kan, A. R., & Brucker, P. (1977). Complexity of machine scheduling problems. In Annals of discrete mathematics (Vol. 1, pp. 343-362). Elsevier.
[24].Fang, K., Uhan, N. A., Zhao, F., & Sutherland, J. W. (2013). Flow shop scheduling with peak power consumption constraints. Annals of Operations Research, 206(1), 115-145.
[25].Wu, S. D., Storer, R. H., & Pei-Chann, C. (1993). One-machine rescheduling heuristics with efficiency and stability as criteria. Computers & Operations Research, 20(1), 1-14.
[26].Luna F., Alba E. (2015) Parallel Multiobjective Evolutionary Algorithms. In: Kacprzyk J., Pedrycz W. (eds) Springer Handbook of Computational Intelligence. Springer Handbooks. Springer, Berlin, Heidelberg
[27]. Talbi, E. G. (2019). A unified view of parallel multi-objective evolutionary algorithms. Journal of Parallel and Distributed Computing, 133, 349-358.
[28]. Shen, X., Zhang, M., & Fu, J. (2014). Multi-objective dynamic job shop scheduling: a survey and prospects. Int J Innov Comput Inf Control, 10(6), 2113-2126.

[29]. Zhang, R., & Chiong, R. (2016). Solving the energy-efficient job shop scheduling problem: a multi-objective genetic algorithm with enhanced local search for minimizing the total weighted tardiness and total energy consumption. Journal of Cleaner Production, 112, 3361-3375.
[30]. Luo, J., & El Baz, D. (2019). A Dual Heterogeneous Island Genetic Algorithm for Solving Large Size Flexible Flow Shop Scheduling Problems on Hybrid multi-core CPU and GPU Platforms. Mathematical Problems in Engineering.
[31]. Alba, E., & Dorronsoro, B. (2008). Cellular genetic algorithms. Operations research/computer science interfaces.
[32]. J. H. Holland, Genetic algorithms, Scientific American 267 (1) (1992) 66-73.
[33].Plazolles, B., El Baz, D., Spel, M., Rivola, V., & Gegout, P. (2017). SIMD Monte-Carlo Numerical Simulations Accelerated on GPU and Xeon Phi. International Journal of Parallel Programming, 1-23.
[34]. Danoy, G., Gaspar Pinto, F., Dorronsoro, B., & Bouvry, P. (2010). Hybrid Cellular Genetic Algorithm for Global Trajectory Optimization Problem. In International Conference on Metaheuristics and Nature Inspired Computing (pp. 1-2).
[35].Sanders, J., & Kandrot, E. (2010). CUDA by example: an introduction to general-purpose GPU programming. Addison-Wesley Professional.
[36].Pharr, M., & Fernando, R. (2005). Gpu gems 2: programming techniques for high-performance graphics and general-purpose computation. Addison-Wesley Professional.
[37].http://www.openmp.org/
[38]. May, G., Stahl, B., Taisch, M., & Prabhu, V. (2015). Multi-objective genetic algorithm for energy-efficient job shop scheduling. International Journal of Production Research, 53(23), 7071-7089.
[39]. Park, B. J., Choi, H. R., & Kim, H. S. (2003). A hybrid genetic algorithm for the job shop scheduling problems. Computers & industrial engineering, 45(4), 597-613.
[40]. Liu, M., & Wu, C. (2008). Intelligent optimization scheduling algorithms for manufacturing process and their applications. National Defense Industry Press, 334.
[41]. Storer, R. H., Wu, S. D., & Vaccari, R. (1992). New search spaces for sequencing problems with application to job shop scheduling. Management science, 38(10), 1495-1509.
[42]. Taguchi, G. (1990). Introduction to Quality Engineering, Tokyo. Asian Productivity Organization.
[43].http://www.minitab.com
[44].Dorronsoro, B., & Bouvry, P. (2013). Cellular genetic algorithms without additional parameters. The Journal of Supercomputing, 63(3), 816-835.
[45].Derrac, J., García, S., Molina, D., & Herrera, F. (2011). A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. Swarm and Evolutionary Computation, 1(1), 3-18.
[46].https://www.ibm.com/analytics/spss-statistics-software
[47].Muth, J. (1963). Probabilistic learning combinations of local job-shop scheduling rules. Industrial Scheduling.
[48]. Adams, J., Balas, E., & Zawack, D. (1988). The shifting bottleneck procedure for job shop scheduling. Management science, 34(3), 391-401.
[49]. Lawrence, S. (1984). Resouce constrained project scheduling: An experimental investigation of heuristic scheduling techniques (Supplement). Graduate School of Industrial Administration, Carnegie-Mellon University.
[50]. Deb, K., Pratap, A., Agarwal, S., & Meyarivan, T. A. M. T. (2002). A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE transactions on evolutionary computation, 6(2), 182-197.