Parallel genetic algorithms to solve dynamic task scheduling problems efficiently by taking into account the energy

Jia Luo

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par l'Université Toulouse 3 - Paul Sabatier

Présentée et soutenue par

Jia LUO

Le 18 janvier 2019

Algorithmes génétiques parallèles pour résoudre des problèmes d'ordonnancement de tâches dynamiques de manière efficace en prenant en compte l'énergie

Ecole doctorale : **EDMITT - Ecole Doctorale Mathématiques, Informatique et Télécommunications de Toulouse**

Spécialité : **Informatique et Télécommunications**

Unité de recherche :
LAAS - Laboratoire d'Analyse et d'Architecture des Systèmes

Thèse dirigée par
Didier EL BAZ

Jury

Mme Laetitia JOURDAN, Rapporteur
M. Enrique ALBA, Rapporteur
M. Christian ARTIGUES, Examinateur
M. David DEFOUR, Examinateur
M. Didier EL BAZ, Directeur de thèse

Acknowledgement

It is a real pleasure to thank the people who have helped me to finish this thesis. First and the most, I would like to show my deepest gratitude to my Ph.D. supervisor Dr. Didier El BAZ in LAAS du CNRS, France and to my host professor Prof. Shigeru, FUJIMURA from the Graduate School of Information, Production and Systems, Waseda University, Japan. They are respectable, responsible and resourceful scholars who have guided and helped me in every matter during my study. Dr. Didier EL BAZ has always supported me to understand how to find the direction of my research with his broad knowledge, creative thinking and deep insights on the subject of parallel and distributed computing and its applications to combinatorial optimization problems. I am also especially grateful to Prof. Shigeru FUJIMURA for giving me the opportunity of an exchange study in his team for three months. During my staying in Japan, he has given me nice suggestions for building the model of energy efficient dynamic scheduling problems and has encouraged me technically and spiritually. It is really a wonderful feeling to do research under their supervision and I would treasure this period of time as my most valuable memory.

I would like to thank all colleagues in the CDA team of LAAS DU CNRS, Bastien PLAZOLLES, Bailal FAKIH, Andrei DONCESCU, Li ZHU for their useful advices, research assistantship and relationship for my Ph. D. study. I want to express my gratitude to my friends in Toulouse, Menglin HE, Xue HAN, Yiran, CHEN, Xin YI and others for their accompaniment and encouragement during three years.

Of course, I would like to express a special acknowledgement to my parents for their understanding and supporting for all these years.

Abstract

Specialty: Information and Telecommunications

Family name: LUO

Given name: Jia

Thesis delivered at: LAAS, UPS Toulouse

Title: Parallel Genetic Algorithms for Solving Energy Efficient Dynamic Shop Scheduling Problems

Due to new government legislation, customers' environmental concerns and continuously rising cost of energy, energy efficiency is becoming an essential parameter of industrial manufacturing processes in recent years. Most efforts considering energy issues in scheduling problems have focused on static scheduling. But in fact, scheduling problems are dynamic in the real world with uncertain new arrival jobs after the execution time. In this thesis, two energy efficient dynamic scheduling problems are studied. Model I analyzes the total tardiness and the makespan with a peak power limitation while considering the flexible flow shop with new arrival jobs. A periodic complete rescheduling approach is adopted to represent the optimization problem. Model II concerns an investigation into minimizing total tardiness and total energy consumption in the job shop with new urgent arrival jobs. An event driven schedule repair approach is utilized to deal with the updated schedule.

As an adequate renewed scheduling plan needs to be obtained in a short response time in dynamic environment, two parallel Genetic Algorithms (GAs) are proposed to solve these two models respectively. The parallel GA I is a CUDA-based hybrid model consisting of an island GA at the upper level and a fine-grained GA at the lower level. It combines metrics of two hierarchical layers and takes full advantage of CUDA's compute capability. The parallel GA II is a dual heterogeneous design composed of a cellular GA and a pseudo GA. The islands with these two different structures increase the population diversity and can be well parallelized on GPUs simultaneously with multi-core CPU.

Finally, numerical experiments are conducted and show that our approaches can not only solve the problems flexibly, but also gain competitive results and reduce time requirements.

Key words: Shop Scheduling, Energy Efficiency, Dynamic Scheduling, Parallel Genetic Algorithms, GPUs, multi-core CPU

Résumé

Spécialité: Informatique et Télécommunications

Nom: LUO

Prénom: Jia

Thèse effectuée au: LAAS, UPS Toulouse

Titre de la Thèse en français : Algorithmes génétiques parallèles pour résoudre des problèmes d'ordonnancement de tâches dynamiques de manière efficace en prenant en compte l'énergie

Du fait de nouvelles législations gouvernementales et de la prise de conscience environnementale des consommateurs ainsi que de la hausse du coût de l'énergie, l'efficacité énergétique est devenue un paramètre essentiel des processus industriels ces dernières années. La plupart des avancées en ce qui concerne les économies d'énergie dans les problèmes d'ordonnancement se sont focalisées sur l'ordonnancement statique. Mais en fait, ces problèmes sont dynamiques dans le monde réel. Dans cette thèse, deux problèmes d'ordonnancement dynamique efficace énergiquement sont étudiés. Le Modèle I analyse le retard total et la durée de production avec une limite de puissance tout en tenant compte d'un flux dynamique de nouvelles tâches. Un rééchelonnement complet périodique est adopté. Le Modèle II vise à réduire au minimum le retard total et la consommation d'énergie totale dans le traitement des tâches en tenant compte de nouvelles tâches prioritaires. Une approche basée sur la réparation de la planification des événements est utilisée pour traiter la mise à jour de l'ordonnancement.

Comme un nouveau plan d'ordonnancement adéquat doit être obtenu dans un temps de réponse court dans un environnement dynamique, deux Algorithmes Génétiques parallèles (AG) sont proposés pour résoudre ces deux modèles. L'algorithme parallèle AG I est une méthode hybride basée sur CUDA consistant en un modèle AG insulaire au niveau supérieur et un modèle AG fin, au niveau inférieur. Il combine les métriques de deux couches hiérarchiques et tire pleinement parti des capacités de calcul de la plateforme CUDA. L'algorithme AG II est conçu avec une double hétérogénéité qui résulte de l'utilisation d'un AG cellulaire parallèle et d'un pseudo AG parallèle. Avec

ces deux structures différentes, les ilots augmentent la diversité de la population et peuvent être simultanément parallélisés sur des GPU et un processeur multi-cœur.

Enfin, des solutions numériques sont présentées et analysées ; elles montrent que nos approches peuvent non seulement résoudre les problèmes de manière flexible, mais également obtenir des solutions avantageuses et réduire les temps de calcul.

Mots clefs: Ordonnancement des tâches, efficacité énergétique, ordonnancement dynamique, algorithmes génétiques parallèles, GPU, CPU multi-cœur.

Content

List of Tables

List of Figures

List of Algorithms

Chapter I. Introduction

This Chapter introduces the research that has been done in this thesis. The main work was undertaken at the Laboratory for Analysis and Architecture of Systems of National Center for Scientific Research (LAAS-CNRS), Toulouse, France with the funding from China Scholarship Council. This study was supervised by Dr. Didier EL BAZ in the Distributed Computing and Asynchronism team at LAAS DU CNRS and was cooperated with Prof. Shigeru FUJIMURA at Waseda University, Japan. The CDA team at LAAS DU CNRS focuses on scientific research results in the fields of high performance computing, parallel computing and distributed computing with application to combinatorial optimization and numerical simulation problems [1] while Prof. Fujimura's research interests are on production management and production scheduling [2]. Therefore, this thesis was carried out in the background of solving shop job scheduling problems by parallel algorithms.

The shop scheduling problem is one of the best known combinatorial optimization problems. In this problem, a set of jobs needs to be scheduled on a set of machines under certain specific optimization criterions. According to the restrictions on the technological routes of the jobs, most of the job shop scheduling works' concern on the three basic types [3]: a flow-shop (each job is characterized by the same technological route), a job-shop (each job has a specific route) and an open-shop (no technological route is imposed on the jobs). Moreover, flexible shops also catch a lot of attention that is a combination of a shop scheduling problem and a parallel machine scheduling problem [4]. Nowadays, energy efficiency is becoming an essential parameter of industrial manufacturing processes, mostly due to new government legislation, customers' environmental concerns and continuously rising cost of energy. About one half of the world's total energy is currently consumed by the industrial sector [5] and its energy consumption has nearly doubled over the last 60 years [6]. Because of a

growing economical competitive landscape and higher environmental norms, it is now vital for manufacturing companies to integrate energy efficiency when dealing with traditional shop scheduling problems. Dynamic optimization problems are problems in which changes occur over time [7]. Scheduling problems are dynamic in the real world with unexpected events after the start time. Dynamic scheduling problems are more complex than static scheduling problems. A lot of methods have been proposed to solve this kind of problems [8]. In the dynamic environment, generating adequate results in a reasonable response time is a key point that cannot be ignored. Due to the hardness, the time cost to obtain an adequate solution for shop scheduling problems is heavy. Furthermore, few works have studied the dynamic scheduling problem with energy efficient demand. Thus, more efforts need be donated to solve energy efficient dynamic shop scheduling problems. Nonetheless we have to face the challenge that new integrated energy requirements in a dynamic environment lead to the complexity of the considered problem to be higher and ask longer execution time to get acceptable solutions.

The Genetic Algorithm (GA) is considered as one of the most efficient method to solve shop scheduling problems. It is a stochastic search algorithm based on the principle of natural selection and recombination [9] and has been successfully applied to solve many difficult optimization problems. However, there is an increase in the required time to find adequate solutions when GAs are applied to complex and large problems. Particularly, repeated fitness function evaluation is often the most prohibitive and limiting segment when GAs are chosen to find an optimal solution for high-dimensional or multimodal implementations. As a consequence, efforts to make GAs faster are deeply demanded and parallel implementation is considered as one of the most promising choices. Generally, there are different ways of exploiting parallelism in GAs [10]: master-slave models, fine-grained models, island models, and hybrid models. The master-slave model is the only one that does not affect the behavior of the algorithm by distributing the evaluation of fitness function to slaves. The fine-grained model works with a large spatially population. The evolution operations are restricted to a small neighborhood with some interactions by overlap structure. The island model divides population into subpopulations. These subpopulations on independent islands are free to converge towards different sub-optima and a migration operator can help mix good

features that emerge locally. The hybrid model combines any two of the above methods. Lots of researches have been carried on parallel GAs to solve shop scheduling problems on different architectures whereas most of them concerned only the traditional case with schedule efficiency.

In the last decades, High Performance Computing (HPC) has become well-known. Super computers and parallel processing techniques are used to solve complex computational issues. By leveraging both administration and parallel computational techniques, this technology's aim is to develop parallel processing algorithms and systems [11]. HPC systems give the ability to provide sustained performance through the concurrent use of computing resources. As a result, it is widely used for solving complex problems and performing research activities through computer modeling, simulation and analysis. The implementation on multi-core processors and many-core processors is one common way to use hardware efficiently for HPC applications. The multi-core CPU is a typical multi-core processor in which a single computing component is equipped with two or more independent processing cores. The instructions have no difference with ordinary CPU instructions, but multiple instructions can be run on separate cores simultaneously to increase overall speed. In theory, parallel implementations may achieve speedup near the number of cores in the best case. On the other hand, Graphics Processing Units (GPUs) are many-core processor devices providing a highly multi-threaded environment using the Single Instruction, Multiple Threads (SIMT) model. To achieve general-purpose parallel computation on GPUs, the Compute Unified Device Architecture (CUDA) [12] was developed in 2006. It is a framework that takes the maximum advantage of the low-lying hardware using an industry standard programming language [13]. These developments provide a nice point for exploring the parallel GAs. However, few studies have been conducted to integrate parallel computing in GAs to solve dynamic energy efficient scheduling problems, because of the complexity that is caused.

Overall, this thesis focuses on solving energy efficient dynamic shop scheduling problems with parallel GAs. Limiting the peak power is one of the main way when shop scheduling deals with energy efficiency, because electricity consumption and operating costs of manufacturing plants are usually charged based on the peak power demand from electricity providers [14]. Meanwhile, minimizing the total energy consumption

within the traditional scheduling problem is an alternative solution as delaying production activities may not be acceptable in manufacturing. Regarding the dynamic scheduling, two strategies are generally used. The complete rescheduling regenerates a new schedule from scratch while the schedule repair refers to some local adjustment of the current schedule [8]. Furthermore, the rescheduling point also has a great influence on the results. If the schedule is executed until some fixed period begins, it is considered as the periodic policy. On the other hand, the event driven policy triggers the rescheduling once any unexpected event happens. In this thesis, two energy efficient dynamic shop scheduling problems are studied. According to the above-mentioned classification schemes, their features are marked as in Table 1.

Table 1 Features of two proposed energy efficient dynamic shop scheduling models

	Model I	Model II
Problem Types	Flow Shop	Job Shop
Flexible Shops	Yes	No
Energy Efficiency	Peak Power	Total Energy Consumption
Reschedule Point	Periodic	Event Driven
Reschedule Strategy	Complete Reschedule	Reschedule Repair

In order to solve the energy efficient shop scheduling problem efficiently and achieve a speedup to meet the short response in the dynamic environment, two parallel GAs are developed in this thesis. The first one is taken for solving Model I. It is a hybrid model consisting of an island GA at the upper level and a fine-grained GA at the lower level. Since the fine-grained model obtains good population diversity when dealing with high-dimensional variable spaces and the island model converges faster by subpopulations, this design combines metrics from two levels to gain competitive results. Meanwhile, the hybrid structure achieves the maximum speedup through its high consistence with the CUDA framework. The second one is implemented to Model II that is composed of a cellular GA and a pseudo GA. The 2D variable spaces of the cellular model and the complementary parent strategy of the pseudo model keep the population diversity while a penetration inspired migration policy shares information between them. Furthermore, this heterogeneous structure can be well parallelized on GPUs simultaneously with multi-core CPU and enjoys parallel computing resources from two sides. The features of two parallel GAs are summarized as in Table 2.

Table 2 Features of two proposed parallel GAs

	Parallel GA I	Parallel GA II
Components	Island GA, Fined-grained GA	Cellular GA, Pseudo GA
Components' Relationship	Hierarchical	Horizontal
Platform	CUDA	CUDA, OpenMP [15]

The rest of this thesis is organized as follows: Chapter 2 presents the state of the art with respect to the recent works on solving shop scheduling problems using parallel GAs. Chapter 3 describes the two proposed parallel GAs and analyzes their performance by some instances of the flexible flow shop scheduling problem. An energy efficient dynamic flexible flow shop scheduling model using the peak power value with consideration of new arrival jobs is discussed in Chapter 4. It is solved by the CUDA-based hybrid GA with the predictive reactive complete rescheduling strategy. Chapter 5 studies a model of job shop scheduling problem in dynamic environment concerning the traditional schedule efficiency, the total energy consumption and the reschedule cost. The dual heterogeneous Island GA is used to solve this problem with the event driven schedule repair policy. Finally, Chapter 6 states the conclusions.

Reference

[1]. https://www.laas.fr/public/en/cda

[2]. https://www.waseda.jp/fsci/gips/other-en/2015/09/08/2172/

[3].Graham, R. L., Lawler, E. L., Lenstra, J. K., & Kan, A. R. (1979). Optimization and approximation in deterministic sequencing and scheduling: a survey. In Annals of discrete mathematics (Vol. 5, pp. 287-326). Elsevier.

[4].Werner, F. (2011). Genetic algorithms for shop scheduling problems: a survey. Preprint, 11, 31.

[5].EIA (2009) International energy outlook 2009. May 2009 2.

[6].EIA (2010) Annual energy review 2009. Report no. DOE/EIA0384(2009); August 2010

[7].Khouadjia, M. R., Sarasola, B., Alba, E., Jourdan, L., & Talbi, E. G. (2011, May). Multi-environmental cooperative parallel metaheuristics for solving dynamic

optimization problems. In Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), 2011 IEEE International Symposium on(pp. 395-403). IEEE.

[8].Ouelhadj, D., & Petrovic, S. (2009). A survey of dynamic scheduling in manufacturing systems. Journal of scheduling, 12(4), 417.

[9].Holland, J. H. (1992). Genetic algorithms. Scientific american, 267(1), 66-73.

[10].Cantú-Paz, E. (1998). A survey of parallel genetic algorithms. Calculateurs paralleles, reseaux et systems repartis, 10(2), 141-171.

[11].https://www.techopedia.com/definition/4595/high-performance-computing-hpc

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

[13].Munawar, A., Wahib, M., Munetomo, M., & Akama, K. (2009). Hybrid of genetic algorithm and local search to solve MAX-SAT problem using nVidia CUDA framework. Genetic Programming and Evolvable Machines, 10(4), 391.

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

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

Chapter II. Related Works

II.1 Introduction

The shop scheduling problem is a classical optimization problem. One instance of the problem consists of a set of n jobs J_0, J_1, …, J_j , ..., J_{n-1} and a set of o machines M_0, M_1, …, M_m, …, M_{o-1}. Each job J_i comprises a number of g stages S_0, S_1, …, S_s, …, S_{g-1}. The processing time of one step of job J_i on a particular machine is denoted as an operation and is abbreviated by (j, s, m). Usually, its value is known as P_{jsm} with the release time R_j and the due time D_j. Additionally, other required conditions are shown in Table 3.

Table 3 Other required conditions for shop scheduling problems

NO.	Description
1	Each operation of a job must be processed by one and only one machine.
2	Each machine can process no more than one operation at a time.
3	Each job is available for processing after the release time.
4	Setup times for job processing and machine assignment times between stages are not taken into consideration.
5	There is infinite intermediate storage between machines.

When a feasible schedule is given, we can compute for each J_j: the completion time C_j, the tardiness $T_j = \max\{0, C_j - D_j\}$, and the unit penalty $U_j = 1$ if $C_j > D_j$, otherwise 0. The most common optimality criteria are the minimization of the makespan C_{max}, the minimization of the sum of the weighted completion time $\sum w_j C_j$, the minimization of the sum of the weighted tardiness $\sum w_j T_j$, and the minimization of the sum of the weighted unit penalty $\sum w_j U_j$, or any combination among them.

There are three ways to classify the scheduling problem in manufacturing systems by the machine environment, the job characteristics and the optimization criterion [1]. However, three basic types: the flow-shop, the job-shop and the open-shop, have caught the most attestation. In a flow-shop, each job passes through the machines with the same order whereas a job-shop enables specified jobs have possibly different machine orderings. In an open-shop, there is no particular route imposed on jobs. Meanwhile, lots of works concern the combination of a shop scheduling and a parallel machine scheduling, in which at least one stage consists of several parallel machines [2]. The flexible flow shop and the flexible job shop are two types that are the most considered.

Most shop scheduling problems are known as strong NP-hard problems [3]. Many works to solve it by exact methods and meta-heuristic methods have been done. However, this class of problems requires complex and time-consuming solution algorithms. Although the speed of the best supercomputer increases 10 times each 3 or 4 years, this increase has only a little influence on the size of solvable problems [4]. Therefore, efforts to coordinate algorithms with HPC accelerators to solve shop scheduling problems efficiently and effectively are deeply desirable. In this Chapter, works on solving shop scheduling problems using parallel GA are presented. It showcases the most representative publications in this field by the categorization of parallel GAs and analyzes their designs based on the frameworks.

II.2 Genetic Algorithms with Scheduling Problems in Manufacturing Systems

II.2.1 Simple Genetic Algorithms

A simple GA [5] starts with a randomly generated initial population consisting of a set of individuals. An individual is representated by a chromosome. For flow shop problems, a standard chromosome consists of a string of length n, and the i-th gene contains the index of the job at position i [2]. An individual describes a feasible schedule of jobs' sequence on target machines. For job shop problems, there are two ways of chromosome representation: direct way and indirect way. The direct way is similar with the way for flow shop problems: a feasible schedule is directly encoded

into the chromosome, whereas the chromosome in the indirect way shows a sequence of dispatching rules for job assignment [6]. As no imposed technological routes of the jobs for open shop problems, both of the encoding approaches for the flow shop and the job shop can be applied in this case. The fitness value of each individual is used to evaluate the current population. It is related to the objective function value of shop scheduling problems at the point represented by a chromosome. Since most common optimality criteria of shop scheduling problems are about minimization. The fitness function FIT(i) of an individual i usually can be transferred as [2]:

$$\mathrm{FIT(i)} = \max\left(\mathrm{E_{max}} - \mathrm{F_i(S_i)}, 0\right) \tag{2.1}$$

where $\mathrm{F_i(S_i)}$ denotes the objective function value of a feasible schedule from individual i and $\mathrm{E_{max}}$ is the estimated maximum value of the objective function.

As the values of objective function for shop scheduling problems are generally positive, some papers measure the fitness function FIT(i) as:

$$\mathrm{FIT(i)} = \frac{1}{\mathrm{F_i(S_i)}} \tag{2.2}$$

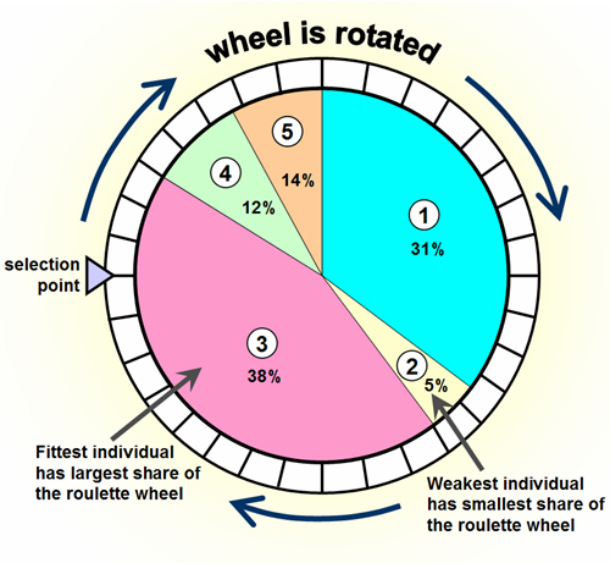

Figure 1 An example of the roulette wheel selection

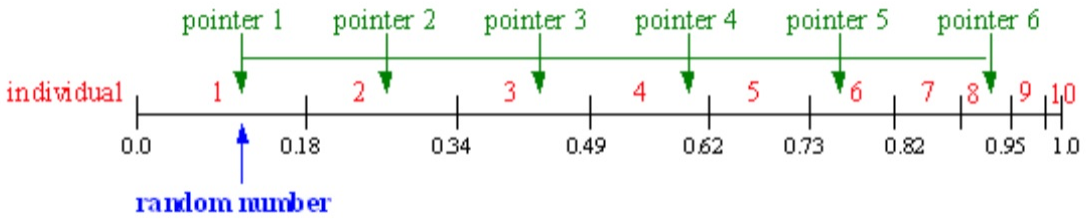

Figure 2 An example of the stochastic universal sampling

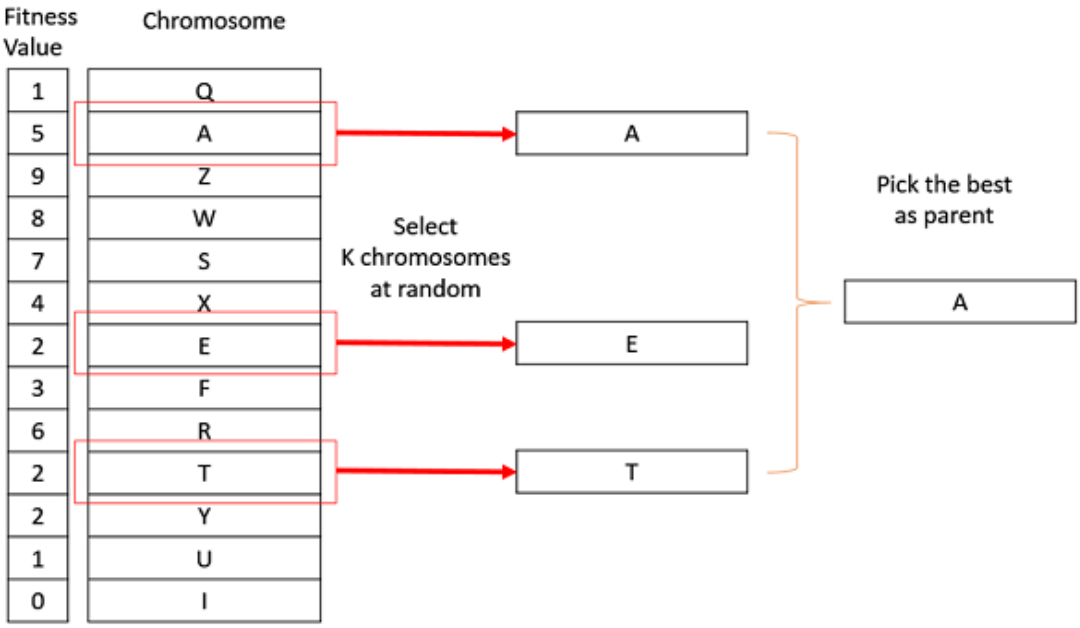

Figure 3 An example of the tournament selection

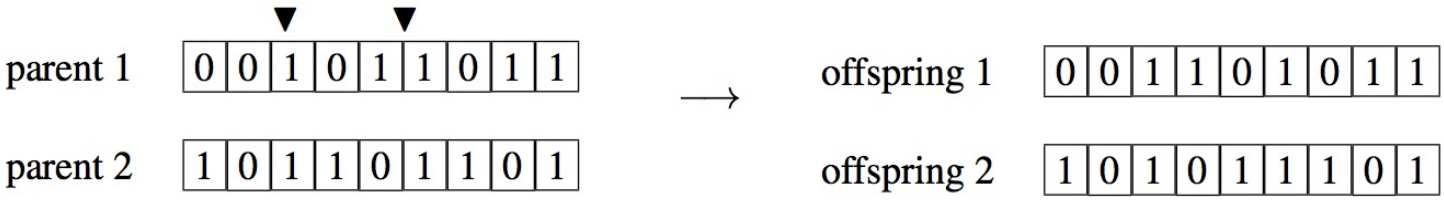

Figure 4 An example of the 2-point crossover

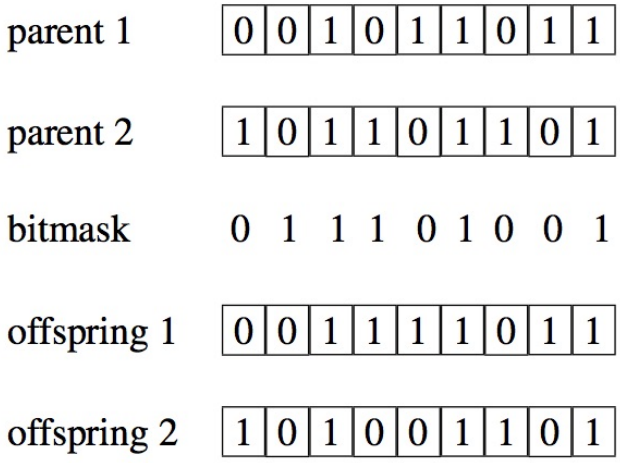

Figure 5 An example of the uniform crossover

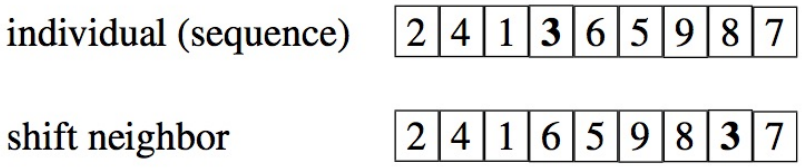

Figure 6 An example of the shift mutation

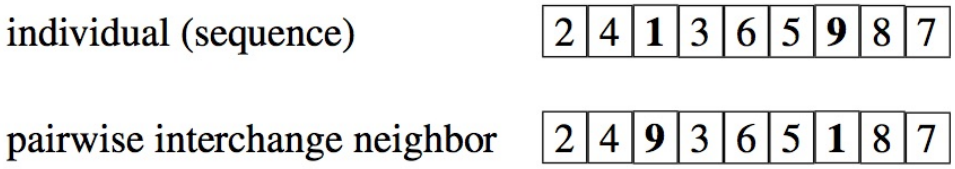

Figure 7 An example of the pairwise interchange mutation

Three GA operations: selection, crossover and mutation, work on these chromosomes to get new search points in a state of space. Usually, individuals are first selected through a fitness-based process. For shop scheduling problems, solutions with larger fitness values are more likely to be selected. Some well-known methods are implemented in this step: the roulette wheel selection [7] (see Figure 1), the stochastic universal sampling [8] (see Figure 2), the tournament selection [9] (see Figure 3) and so on [10]. Next, the crossover takes two random individuals kept after selection and exchanges random sub-chromosomes. The classic methods are the n-point crossover [2] (see Figure 4) and the uniform crossover [2] (see Figure 5). Due to particular requirements of different shop scheduling problems, additional steps may be required to repair the illegal offspring caused by the crossover. The mutation then alters some

random value within a chromosome. Different from the binary encoding, the mutation for shop scheduling problems works often based on the neighborhoods e.g. shift mutation (insertion neighborhood) [2] (see Figure 6) or pairwise interchange mutation (swap neighborhood) [2] (see Figure 7) to respect feasible solutions. Population evaluation is executed after these three steps. Sometimes, an elitist strategy is hired afterwards to keep limited number of individuals with the best fitness values to the next generation. This process repeats until the termination criteria have been satisfied. The full procedure is stated in Table 4.

Table 4 The pseudo-code of simple GA

```
1:  initialize();
2:  while (termination criteria are not satisfied) do
3:      Generation++
4:      Selection();
5:      Crossover();
6:      Mutation();
7:      FitnessValueEvaluation();
8:  end while
```

II.2.2 Master-Slave Genetic Algorithms

The master-slave GA is known as global parallel GA as well. It keeps a single population as a simple GA that is stored at the master side. In this case, each individual is free to compete and mate with any other. Since the fitness value calculations of individuals are independent without any communication with others, the slaves take care of fitness evaluation in parallel. Data exchange occurs only when sending and receiving tasks between the master and slaves. Obviously, frequent communication overhead offsets some performance gains from slaves' computing. However, as master-slave GA is the easiest parallel model to be implemented and does not assume underlying architecture, it is still very efficient when the evaluation is complex and requires considerable computation. The structure [11] and the steps of this parallel model are presented in Figure 8 and Table 5 respectively.

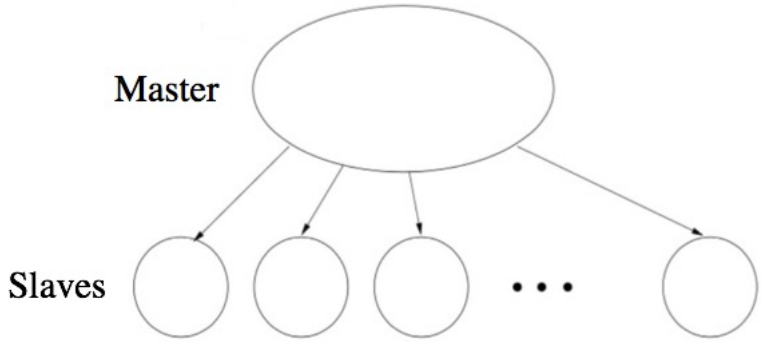

Figure 8 The scheme of master-slave GA

Table 5 The pseudo-code of master-slave GA

```
1:  Initialize();
2:  while (termination criteria are not satisfied) do
3:     Generation++
4:     Selection();
5:     Crossover();
6:     Mutation();
7:     Parallel_FitnessValueEvaluation_Individuals();
9:  end while
```

II.2.2.1 Job Shop Scheduling Problems

AitZai et al. modeled the job shop scheduling problem with blocking using the alternative graph with conjunctive arcs and alternative arcs in [12]. In addition to a parallel branch and bound method, two master-slave GA parallelization methods were also presented. The first one was based on CPU networking with a star network of inter-connected computers. On the opposite, the second one worked on GPU with some memory management respecting to CUDA framework. Numerical tests were carried on a station equipped with CPU ×2: Intel Xeon E5620 and GPU: NVIDIA Quadro 2000 01 Go GPU. With a population size 1056 and a limited total execution time 300s, the master-slave GA using GPU could get maximum 15 times more explored solutions than the GA using CPU. Moreover, a related earlier work was introduced by AitZai in [13]. In order to improve the solution of job shop scheduling problems, Somani et al. [14] imposed a topological sorting step to the GA before the fitness value calculation, which

was used to generate the topological sequences of directed acyclic graph. The parallel implementation of the proposed GA in CUDA environment consisted of two kernels. The former one was used for making the topological sequences by the help of topological sorting, while the later one was hired to calculate the makespan from the longest path algorithm. The crossover and the mutation were performed between two randomly selected schedules on CPU. Experiments was setup with Intel(R) Xeon(R) E5-2650 @ 2.00 GHz and NVIDA Tesla C2075 (448 cores) and results have shown the proposed GA performed around 9 times faster for large-scale problems than the sequential GA.

Another job shop scheduling problem was studied by Mui et al. [15] where a prior-rule was used to create active schedules. The selection combined the idea of an elitist strategy and a roulette wheel selection, whereas the crossover took a GT algorithm implemented on three parents and the mutation used neighborhood searching technique. With this design, the main part of the GA could be computed independently. In the parallel environment, a master-slave model was employed where the slaves performed the GA evolutionary operators concurrently and the master searched the global optimum among optimal results received from slaves. The proposed method was run on the CSS computer server system with 6 computers, in which each computer had a Pentium-4 CPU with 4GB free of ram. Empirical results have shown the master-slave GA with 6 processors could save 3 to 4 times the execution time compared to the sequential version.

II.2.2.2 Flow Shop Scheduling Problems

A master-slave GA dealing with a single population and a group of local subpopulations was presented in [16] for a flow shop problem. This method involved a master scheduler and a set of slave processors. The master scheduler ran the GA operators (partial replacement selection, cycle crossover and swapping mutation) of all individuals sequentially. When the evolution of one individual was finished, it was placed in the unassigned queue from which the master scheduler partitioned the fitness value calculation to slave processors in batches. The choice of candidate slave processors was made upon the involved communication overhead and their

computational potential. The available resources among slave processors in the distributed system could vary over time. Moreover, all individuals were maintained in the master scheduler synchronously. The proposed GA was implemented on a laptop with Prentium IV core 2 Duo 2.53 GHz CPU. The outputs showed the new algorithm could be 9 times faster maximally than the results of serial GA achieved by the Lingo 8 software.

Attentions to use master-slave GA to shop scheduling problems have been increased in the last decade and the work is carried with various underlying architectures. Since only independent tasks are executed on slaves without communication cost among them, both the conventional GA and any improved GAs can be implemented with it easily. Although the communication between the master and the slaves is an impediment in speed, it still performs well to solve shop scheduling problems whose fitness value calculation is complex and requires considerable computation.

II.2.3 Fine-grained Genetic Algorithms

The fine-grained GA can also be called as neighborhood GA, diffusion GA or massively parallel GA. The main idea is to map individuals of a single GA population on a spatial structure. An individual is limited to compete and mate with its neighbors, while the neighborhoods overlapping makes good solutions disseminate through the entire population. This model obtains good population diversity when dealing with high-dimensional variable spaces [17]. Meanwhile, it is easy to be placed in any 2D grid, as many massively parallel architectures are designed with this topology. However, we cannot neglect the great influence from the spatial structure, which generally has little chance to be modified. The scheme [11] and the implemented process of the fine-grained GA are shown in Figure 9 and Table 6 separately.

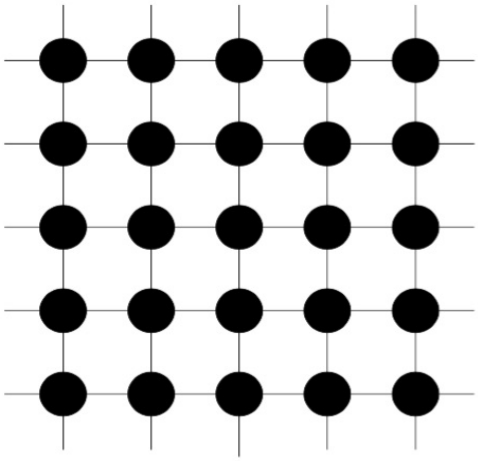

Figure 9 The scheme of fine-grained GA

Table 6 The pseudo-code of fine-grained GA

```
1:  Initialize();
2:  while (termination criteria are not satisfied) do
3:      Generation++
4:      Parallel_NeighborhoodSelection_Individuals();
5:      Parallel_NeighborhoodCrossover_Individuals();
6:      Parallel_Mutation_Individuals();
7:      Parallel_FitnessValueEvaluation_Individuals();
8:  end while
```

II.2.3.1 Job Shop Scheduling Problems

A fine-grained GA solving job shop scheduling problems was considered by Tamaki et al. [18]. In this paper, the selection was executed locally in a neighborhood of each population. The objectives of this neighborhood model were to improve search in the GA by suppressing favorably the premature convergence phenomena, and to reduce computational time by implementing it on a parallel computer at the same time. The method was then modified as an absolute neighborhood model and implemented on Transputer. Transputer was a MIMD (Multi-Instruction Multi-Data) machine with microprocessors, featuring integrated memory and serial communication links. Through several computational experiments for job shop scheduling problems, the parallel GA with 16 processors could shorten the calculation time dramatically.

However, as Transputer did not equip with shared memory, the data exchange was handled through communication operations. Therefore, the calculation time reduction was not able to reach an ideal level. Lin et al. [19] investigated parallel GAs on job shop scheduling problems with a direct solution representation, which encoded the operation starting times. The GA operators were inspired by the G&T algorithm with the random selection, the THX (time horizon exchange) crossover and the THX mutation. Two hybrid models built up by the fine-grained GA with a 2D torus topology and the island GA connected in a ring were discussed in this paper. The first one was an embedding of the fine-grained GA into the island GA, in which each subpopulation on the ring was a torus. The migration on the ring was much less frequent than within the torus. In the second model, the connection topology used in the island GA was one which is typically found in the fine-grained GA and a relatively large number of nodes were used. The migration frequency kept the same in the island GA. Those two methods were carried on a Sun Ultra 1 which was a family of Sun Microsystems workstations based on the 64-bit Ultra SPARC microprocessor with a single population GA, two island GAs of different subpopulation sizes and one torus fine-grained GA. The execution time comparison was only made between the single population GA and two island GAs with the speedup of 4.7 and 18.5 respectively. Regarding to solutions' quality, best results were obtained by the hybrid model consisting of island GAs connected in a fine-grained GA style topology by combing the merits from them.

Compared with other two kinds of parallel GA, it seems the implementation of fine-grained GA for shop scheduling problems is rare and outdated, no matter the amount of related papers or the various types of treated problems. Along the development of modern computing accelerators with 2D grid environment, like GPU, this implementation has a lot of potential in the near future. Apart form manufacturing systems, the fine-grained GA is also used for task scheduling problems [20]. It is another type of scheduling problems that focuses on minimizing the makespan as well but for a set of tasks to be executed in multiprocessor systems. In this domain, the fine-grained model is treated sometimes as parallel cellular GA [21].

II.2.4 Island Genetic Algorithms

The island model is the most famous for the research on parallel GAs. In some papers, it may be called as coarse-grained models, multi-deme models, multi-population models, migration models or distributed models. Unlike previous parallel GAs, this model divides the population into a few relatively large subpopulations. Each of them works as an island and is free to converge towards its own sub-optima. At some points, a migration operator is utilized to exchange individuals among islands. These configurations make the average population fitness improve faster and mix good local feature efficiently [11]. The main idea of this parallelization is a simple extension of the serial GA while the island model based underlying architecture is easily available. Therefore, the island GA dominates the work on parallel GAs for shop scheduling problems. A sketch [11] and a brief outline about this algorithm are illustrated in Figure 10 and Table 7 distinctly.

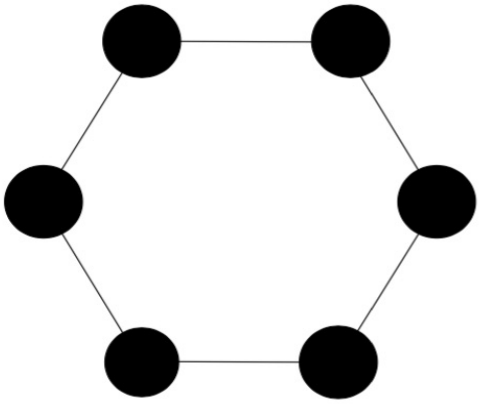

Figure 10 The scheme of island GA

Table 7 The pseudo-code of island GA

```
1:  Initialize();
2:  while (termination criteria are not satisfied) do
3:     Generation++
4:     Parallel_SubSelection_Islands();
5:     Parallel_SubCrossover_Islands();
6:     Parallel_SubMutation_Individuals ();
7:     Parallel_FitnessValueEvaluation_Individuals();
8:     if (generation % migration interval==0)
9:        Parallel_Migration_Islands();
10:    end if
11: end while
```

II.2.4.1 Job Shop Scheduling Problems

Park et al. [22] studied a hybrid GA and its parallel version for job shop scheduling problems with an operation-based representation. Concerning the parallel GA, the population was divided into two or four subpopulations. Each subpopulation acted as a single-population GA, where some individuals could migrate from one subpopulation to another at certain intervals. As four population initialization methods, four crossover operators and two selection operators were proposed in this paper, different subpopulations were equipped with different settings to help them evolve independently. Beside, the migration was implemented synchronously with a static ring type connection scheme. Experiments were carried out on a PC with Pentium II 350 and 64MB main memory with MT, ORB and ABZ benchmark problems [19]. The outputs confirmed the island GA improved not only the best solution but also the average solution from results of single GA. Asadzadeh et al. addressed a parallel agent-based GA for a job shop scheduling problem in [23]. Chromosomes of the population, indicating feasible schedules for problem instances were created by the management agent and the execute agent. Afterwards, the management agent divided it into subpopulations with the same size and sent each of them to processor agents. Each processor agent located on a distinct host and executed GA with a roulette wheel selection, a partially matched crossover and a subsequent gene mutation on its

subpopulation independently. Different subpopulations communicated by exchanging migrants through the synchronization agent. The number of processor agents was fixed at eight in the experiments. Furthermore, those agents formed a virtual cube amongst themselves and each of them had three neighbors. JADE middleware was used to implement this method, which was a software development framework aiming at developing multi-agent systems. Compared with the serial agent-based GA, the suggested algorithm obtained much short schedule lengths and had higher convergence speed with large size problems. In [24]. Gu et al. constructed a stochastic job shop scheduling problem by a stochastic expected value model. It was solved by a parallel quantum GA organized by the island model with a hybrid star-shaped topology. The information communication was performed through a penetration migration at the upper level and through a quantum crossover at the lower lever. Besides, the roulette wheel selection, the cycle crossover and the Not Gate mutation were designed as GA operators. Computational tests were run on a PC with a Pentium Processor with clock speed of 1.66 GHZ. On the average, the advised method had a better performance of generating optimal or near-optimal solutions with fast convergence speed than a GA or a quantum GA for large instance problems. Spanos et al. [25] designed a parallel GA for solving job shop scheduling problems with an elitist strategy based selection, a path relinking crossover and a swap mutation. The parallelization was set following the islands paradigm. However, one subpopulation merged with another one once the individuals inside stagnated, where the Hamming distance of more than half individuals were less than a predefined value. The process continued until there was only one subpopulation. Experiments were performed on a commodity workstation with a Pentium IV CPU running at 2 GHz with 1 GB RAM, The results indicated the addressed algorithm managed to attain a comparable performance with five recent approaches.

II.2.4.2 Flow Shop Scheduling Problems

Huang et al. [26] discussed flow shop scheduling problems with fuzzy processing times and fuzzy due dates, where the possibility and necessity measures with exact formulas were adopted to maximize the earliness and tardiness simultaneously. A modified GA was designed to solve the problems with the random keys, the parameterized uniform crossover and the immigration. If Pt was the family of chromosomes in the t-th

generation, then |Pt| denoted the population size of Pt. The next generation was made of a% best chromosomes from Pt, b% chromosomes for taking crossovers, and c% chromosomes generated randomly as immigrations, where a+b+c=100. In order to get more efficient convergence, an idea of the longest common substring and rearranging of the chromosomes chosen in the mating pool were also imposed in the algorithm. The full procedure was coded with CUDA by separating the whole population into blocks using the block size of 256 or 128. Circumventing to load the random keys of all chromosomes to global memory, one chromosome was distributed to a block so that all random keys could fit in the shared memory. Although there was no migration among blocks, the idea was organized based on the island GA. In the case of 200 jobs, the numerical simulations on a 2.33 GHz Intel Core2 Quad desktop computer with 2 GB of RAM, and an NVIDIA GeForce GTX285 graphics card showed that the proposed GA combining with CUDA parallel computation got 19 times speedup. Similarly, Zajicek et al. [27] proposed a homogeneous parallel GA model on the CUDA architecture, where all computations were carried out on the GPU in order to reduce communication between CPU and GPU. The main idea was based on an island GA with a tournament selection, an arithmetic crossover and a Gaussian mutation. Experiments were carried on a system with AMD Phenom II X4 945 3.0 GHz processor and NVIDIA Tesla C1060 GPU. Some instances of the flow shop scheduling problem were solved with speedup from 60 to 120 comparing to the equivalent sequential CPU version.

Bożejko et al. proposed a parallel GA for flow shop scheduling problems in [28]. The algorithm was based on an island model. To implementations, a Multi-Step Crossover Fusion was used to construct a new individual using the best individuals of different subpopulations and worked with the migration operator to complete the communication between different islands. Tests were performed on 4-processors Sun Enterprise 4x400 MHz under the Solaris 7 operating system, which is a MIMD machine of processors without shared memory. Four crossover operators and four mutation operators were considered as GA operators. The efficiency of the island GA was activated with the combination of three strategies: with the same or different start subpopulations, as independent or cooperative search islands and with the same or different genetic operators. Results turned out the strategy of starting the computation from different

subpopulations on every processor with different crossover operators and cooperation was significantly better than others. The improvement of the distance to reference solutions and the improvement of the standard deviation were at the level of 7% and 40% respectively, comparing to the sequential GA. A related work by the same team to minimize the total weighted completion time for the flow shop problem with a special case of a single machine was solved by a similar island GA in [29]. The results noted the 8-processors implementation performed the best.

II.2.4.3 Open Shop Scheduling Problems

Kokosinski et al. [30] studied an open shop scheduling problem and two greedy heuristics, LPT-Task and LPT-Machine, were proposed for decoding chromosomes represented by permutations with repetitions. The GA operators constituted a 2-elements tournament selection, a linear order crossover and a swap mutation or an invert mutation with constant or variable mutation probabilities. An island GA with a migration strategy was applied to the parallel version in which every island sent its best emigrants to all other islands and received immigrants from them. Incoming individuals replaced the chromosomes of host subpopulation randomly. The experimental platform was a PC with Pentium 4 processor (3.06 GHz) and 1 GB RAM. Unfortunately, this parallelization did not reveal obvious advantages in the results. A non-preemptive open shop scheduling problem was discussed by Harmanani et al [31]. Except a feasible solution, a chromosome in this paper included a scratch area through which a ReduceGap operation communicated to GA operators: the crossover and the mutation. A hybrid island GA was hired to organize the parallelization where neighboring islands shared their best chromosomes every G_N generation and all islands broadcasted their best chromosome to all other islands every L_N generations, where $G_N \ll L_N$. Islands were connected through an Ethernet network and used the Message Passing Interface (MPI) on a Beowulf cluster. The experiments were executed on a cluster of five machines that were running Linux and MPI. The outputs presented that the proposed method converged to a good solution quickly before it saturated with a speedup between 2.28 and 2.89 for large instances. A similar work was carried by Ghosn et al. in [32] later.

Regarding to solve shop scheduling problems by the island GA, various researches have been done with different architectures. We can discover that the works with GPU pay heavier attention on speedup gained from the island GA. On the opposite, the others consider more the improvement for solutions quality and convergence speed. Few implementations have discussed them simultaneously with a fair comparison. Besides, the island connection topology is varied from different papers with different migration strategies. Some of the designs are carried with respect to the underlying architectures, whereas the others are proposed from supporting theories. However, a completely understanding for the effects of migration is still missing as far our knowledge is concerned.

II.2.4.4 Flexible Shop Scheduling Problems

Defersha et al. [33] considered an island GA for a flexible job shop scheduling problem with lot streaming. In this case, the batch of each job was split into certain number of unequal consistent sublots. Each sublot of a job underwent a number of operations in a fixed sequence where each operation could be processed by one of several eligible machines. Three commonly used migration topologies: ring, mesh and fully connected were discussed in this paper with a k-way tournament selection, five kinds of crossover and six kinds of mutation applied by different probabilities. A parallel computation environment was composed more than 250 interconnected workstations each having an 8-core Intel Xeon 2.8GHz processor was used for experiments. Test problems were run using up to 48 cores and taking MPI for communication. With all problems considered, there were makespan reductions through the island GA. Meanwhile, empirical studies presented the impact from its different parameters. Regarding to topologies, the fully connected one outperformed other two. Three migration policies: random-replace-random, best-replace-random and best-replace-worst were tested. Results showed the island GA was not much sensitive to the change of migration policy while the best-replace-random migration policy performed slightly better. The same authors built a mathematical model for a flexible job shop scheduling problem incorporating sequence-dependent setup time, attached or detached setup time, machine release dates, and time lag requirements in [34]. Like the previous work, the GA operators constituted a k-way tournament selection, three assignment operators and two sequencing operators

applied by different probabilities. However, islands were connected with a randomly topology which employed randomly generated migration routes for each communication epoch. The method was tested on a similar experimental platform. Results of medium size problems showed the island GA helped improve the solutions quality and it converged to a better solution within the allowable computational time for large size problems where the single GA failed.

An island GA for flexible flow shop scheduling problems was addressed by Belkadi et al. [35] where genome constituted one assignment chromosome and a sequencing chromosome. The GA was implemented on a biprocessor architecture with a roulette wheel selection, a uniform crossover and a mutation similar to the crossover but operated only on the sequencing scheduling chromosomes. Four combinations from two island connected typologies (ring and grid with two dimensions) and two replacement strategies (best and random) were tested. The results noted those two parameters did not have significant influence in the variation of makespan. Regarding to the subpopulation size and its related subpopulation amount, the quality of the solution decreased progressively at the same time as the number of subpopulations increased based on the experiments. However, when the complexity of the problem rose up, this influence reduced. Finally, outputs stated the migration interval was the parameter that had the decision influence to the island GA where the quality of the solution improved gradually with increasing migration frequency. A comparison between the island GA and the sequential GA was also carried in this paper. According to empirical results, the island GA always obtained a smaller makespan while its performance of execution time was only discussed with theoretical values based on two processors. Rashidi et al. [36] studied flexible flow shop scheduling problems with unrelated parallel machines, sequence-dependent setup times and processor blocking to minimize the makespan and the maximum tardiness. Different weights were assigned to two criteria to transform the problem into a single-objective function. The individuals inside one island sought for their own single-objective function, and all islands worked in parallel for Pareto optimal solutions. The paired weights in different islands are different with a small deviation between each successive pairs. After executing the conventional GA operators, a local search step or a Redirect procedure were implemented to further cover the Pareto solutions. A comparison was carried between

the island GA without or with a local search step and a Redirect procedure where the later one indicated better performance.

As a combination of a shop scheduling problem and a parallel machine scheduling problem, the complexity of flexible shop scheduling problems is increased. According to previous work, the implementation of parallel GAs for this kind of specific problems is only referred by the island GA. In addition to design the algorithm, some papers have considered the influence from the migration by the connection topology, the migration rate, the migration interval and the migration strategy. A good cooperation of these parameters could decentralize the searching space and enlarge the diversity level to make a GA have better performance while enjoying a speedup from computing accelerations. However, current implementations are still limited. Most of the works address only the improvement to solutions quality. Experimental results to analyze the speedup gained from the island GA are not sufficient. As the increased complexity will lead to longer execution time, it is interesting to consider GPU to solve related problems whose native topology is suitable for the island GA with thousands parallel computing threads.

II.3 Design of HPC Frameworks-Based Genetic Algorithms for Shop Scheduling Problems

The preliminary work of parallel GAs for shop scheduling problems is implemented by fine-grained models on distributed memory machines. Although the results are outdated, impressive reduction for the execution time has been achieved. As the fine-grained GA is easy to be placed on a spatial structure, to coordinate this design with some modern HPC accelerators with 2D grid architecture, such as CUDA, is supposed to optimize its performance. Moreover, with new requirements from manufacturing systems in the real life, the complexity of shop scheduling problems is increasing. The 2D grid topology could organize a greater amount of threads to work in parallel, which is more efficient to help find optimal results of strong NP hard problems with large instances. The other problems from the operations research family solved in this way [37, 38] could be persuasive evidences.

The MIMD machine also works with the island GA at the earlier stage. Soon, it is improved to a parallel computation environment or a computer cluster equipped with multiple processors or multi-core processors. The commonly used parallel processing library MPI is generally chosen for information sharing through the migration. Meanwhile, GPU is involved with its special memory management to work with this design. As there is no strict underlying architecture limitation to implement the island GA when dealing with shop scheduling problems, the islands connected topology is varied. According to the collected papers, the ring topology is used most frequently. But it is hard to judge which topology performs the best. Besides, the cooperated influence between islands connected topology and other migration parameters cannot be neglected. Fortunately, the average results confirm the implementations of island GAs for shop scheduling problems are able to improve solutions quality and gain a speedup with reasonable migration design. As this model dominates not only the work on parallel GAs for shop scheduling problems but also parallel GAs for other applications, it still has a lot of potential in the future with the popularity of computing nodes providing multiple processors or multi-core processors.

Since the master slave GA does not assume underlying computer architecture, any parallel computing environment has the chance to use this design without worrying about sharing information. The most time consuming part for GAs to shop scheduling problems is the fitness value calculation that requires even much longer execution time with large problems. Therefore, GPU equipped with much more parallel threads is considered to have better performance among several choices.

II.4 Conclusion

As one kind of important problem in combinatorial optimization, applying parallel GAs for solving shop scheduling problems have caught heavy attention since last few decades. This Chapter addressed some of the most representative publications in this domain and the reviews were classified by the most common parallel GA categories: master-slave models, fine-grained models and island models. An independent section for hybrid models combining two of the above methods was not set, as the related work was few. These we have considered in this Chapter were assigned to one of the three

basic models according to their main designs. Most works of parallel GAs to search optimal results for scheduling problems in manufacturing systems are currently managed by the island GA. However, the future of implementing the other two parallel models to this field is promising as well by the development of modern computing accelerators with more parallel threads.

Reference

[1].Graham, R. L., Lawler, E. L., Lenstra, J. K., & Kan, A. R. (1979). Optimization and approximation in deterministic sequencing and scheduling: a survey. In Annals of discrete mathematics (Vol. 5, pp. 287-326). Elsevier.

[2].Werner, F. (2011). Genetic algorithms for shop scheduling problems: a survey. Preprint, 11, 31.

[3].Ullman, J. D. (1975). NP-complete scheduling problems. Journal of Computer and System sciences, 10(3), 384-393.

[4].Bożejko, W. (2010). A new class of parallel scheduling algorithms. Oficyna wydawn. Politechniki Wrosławskiej.

[5].Holland, J. H. (1992). Genetic algorithms. Scientific american, 267(1), 66-73.

[6].Cheng, R., Gen, M., & Tsujimura, Y. (1996). A tutorial survey of job-shop scheduling problems using genetic algorithms—I. Representation. Computers & industrial engineering, 30(4), 983-997.

[7].http://www.edc.ncl.ac.uk/highlight/rhjanuary2007g02.php/

[8]. http://www.geatbx.com/docu/algindex-02.html#P416_20744

[9].https://www.tutorialspoint.com/genetic_algorithms/genetic_algorithms_parent_selection.htm

[10].Jebari, K., & Madiafi, M. (2013). Selection methods for genetic algorithms. International Journal of Emerging Sciences, 3(4), 333-344.

[11].Cantú-Paz, E. (1998). A survey of parallel genetic algorithms. Calculateurs paralleles, reseaux et systems repartis, 10(2), 141-171.

[12].AitZai, A., Boudhar, M., & Dabah, A. (2013). Parallel CPU and GPU computations to solve the job shop scheduling problem with blocking.

[13].AitZai, A., Benmedjdoub, B., & Boudhar, M. (2012). A branch and bound and parallel genetic algorithm for the job shop scheduling problem with blocking. International Journal of Operational Research, 14(3), 343-365.
[14].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.
[15].Mui, N. H., Hoa, V. D., & Tuyen, L. T. (2012, December). A parallel genetic algorithm for the job shop scheduling problem. In Signal Processing and Information Technology (ISSPIT), 2012 IEEE International Symposium on (pp. 000019-000024). IEEE.
[16].Akhshabi, M., Haddadnia, J., & Akhshabi, M. (2012). Solving flow shop scheduling problem using a parallel genetic algorithm. Procedia Technology, 1, 351-355.
[17].Kohlmorgen, U., Schmeck, H., & Haase, K. (1999). Experiences with fine-grained parallel genetic algorithms. Annals of Operations Research, 90, 203-219.
[18]. Tamaki, H. (1992). A paralleled genetic algorithm based on a neighborhood model and its application to the jobshop scheduling. Parallel Problem Solving from Nature 2, 573-582.
[19].Lin, S. C., Goodman, E. D., & Punch, W. F. (1997, April). Investigating parallel genetic algorithms on job shop scheduling problems. In International Conference on Evolutionary Programming (pp. 383-393). Springer, Berlin, Heidelberg.
[20].Pinel, F., Dorronsoro, B., & Bouvry, P. (2013). Solving very large instances of the scheduling of independent tasks problem on the GPU. Journal of Parallel and Distributed Computing, 73(1), 101-110.
[21].Alba, E., & Dorronsoro, B. (2009). Cellular genetic algorithms (Vol. 42). Springer Science & Business Media.
[22].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.
[23].Asadzadeh, L., & Zamanifar, K. (2010). An agent-based parallel approach for the job shop scheduling problem with genetic algorithms. Mathematical and Computer Modelling, 52(11-12), 1957-1965.

[24].Gu, J., Gu, X., & Gu, M. (2009). A novel parallel quantum genetic algorithm for stochastic job shop scheduling. Journal of Mathematical Analysis and Applications, 355(1), 63-81.

[25].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.

[26].Huang, C. S., Huang, Y. C., & Lai, P. J. (2012). Modified genetic algorithms for solving fuzzy flow shop scheduling problems and their implementation with CUDA. Expert Systems with Applications, 39(5), 4999-5005.

[27].Zajıcek, T., & Sucha, P. (2011). Accelerating a Flow Shop Scheduling Algorithm on the GPU. eraerts, 143.

[28].Bożejko, W., & Wodecki, M. (2003, September). Parallel genetic algorithm for the flow shop scheduling problem. In International Conference on Parallel Processing and Applied Mathematics (pp. 566-571). Springer, Berlin, Heidelberg.

[29].Bożejko, W., & Wodecki, M. (2004, June). Parallel genetic algorithm for minimizing total weighted completion time. In International Conference on Artificial Intelligence and Soft Computing (pp. 400-405). Springer, Berlin, Heidelberg.

[30].Kokosiński, Z., & Studzienny, Ł. (2007). Hybrid genetic algorithms for the open-shop scheduling problem. IJCSNS, 7(9), 136.

[31].Harmanani, H. M., Drouby, F., & Ghosn, S. B. (2009, March). A parallel genetic algorithm for the open-shop scheduling problem using deterministic and random moves. In Proceedings of the 2009 Spring Simulation Multiconference(p. 30). Society for Computer Simulation International.

[32].Ghosn, S. B., Drouby, F., & Harmanani, H. M. (2016). A parallel genetic algorithm for the open-shop scheduling problem using deterministic and random moves. Int. J. Artif. Intell, 14(1), 130-144.

[33].Defersha, F. M., & Chen, M. (2009, August). A coarse-grain parallel genetic algorithm for flexible job-shop scheduling with lot streaming. In Computational Science and Engineering, 2009. CSE'09. International Conference on (Vol. 1, pp. 201-208). IEEE.

[34].Defersha, F. M., & Chen, M. (2010). A parallel genetic algorithm for a flexible job-shop scheduling problem with sequence dependent setups. The international journal of advanced manufacturing technology, 49(1-4), 263-279.

[35].Belkadi, K., Gourgand, M., & Benyettou, M. (2006). Parallel genetic algorithms with migration for the hybrid flow shop scheduling problem. Advances in Decision Sciences, 2006.

[36].Rashidi, E., Jahandar, M., & Zandieh, M. (2010). An improved hybrid multi-objective parallel genetic algorithm for hybrid flow shop scheduling with unrelated parallel machines. The International Journal of Advanced Manufacturing Technology, 49(9-12), 1129-1139.

[37].Boyer, V., & El Baz, D. (2013, May). Recent advances on GPU computing in operations research. In Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2013 IEEE 27th International (pp. 1778-1787). IEEE.

[38].Boyer, V., El Baz, D., & Salazar-Aguilar, M. A. (2017). GPU Computing Applied to Linear and Mixed Integer Programming.

Chapter III. Two Efficient New Parallel Genetic Algorithms

III.1 Introduction

According to related works, there are different ways of exploiting parallelism in GAs [1]: master-slave models, fine-grained models, island models, and hybrid models. The master-slave model is the only one that does not affect the behavior of the algorithm by distributing the evaluation of fitness function to slaves. The fine-grained model works with a large spatially population. The evolution operations are restricted to a small neighborhood with some interactions by overlap structure. The island model divides population into subpopulations. These subpopulations on independent islands are free to converge towards different sub-optima and a migration operator can help mix good features that emerge locally. The hybrid model combines any two of the above methods. Despite the fact that the island model dominates the work on parallel GAs, it is hard to conclude that the island GA overcomes other models since the comparison cannot be made in absolute terms.

As far as solving shop scheduling problems by parallel GAs, we could find two drawbacks from the former works. Firstly, few implementations have considered the execution time reduction and the solutions' quality improvement simultaneously with a fair comparison. Particularly, the GPU-based parallel GA is usually used to gain the maximum possible speedup while compromising with the solutions' quality. Regarding as the most frequent used GPUs architecture, the previous discussed parallel GAs on CUDA also have not conquered this problem when they were chosen for solving shop scheduling problems. Second, island GAs generally have a high risk to lose the

population diversity due to the same genetic operator configurations and the limited sub-population sizes. Despite some promising results from leveraging computational capabilities of a cluster to improve its performance, these methods must face some common challenges such as lost connections, low bandwidth, abandoned work, security and privacy [2]. Meanwhile, it is hard to control its performance with multiple migration parameters. Thus, proposing a method to optimize the performance of island GA with a better population diversity and less time requirement in a stable and secure parallel environment is represented as a not fully resolved topic.

Therefore, designing a parallel GA that is highly consistent with the CUDA framework while balancing conflicts between the solutions' quality and the execution time remains an open research challenge. Similarly, crafting a heterogeneous island model that is well suited for parallelizalizing inside or between GPUs and multi-core CPU is considered as an efficient way to overcome the shortages of island GAs and is extremely desired. In this Chapter, we seek to address two efficient new parallel GAs and their application to flexible flow shop scheduling problems (FFSP). The parallel GA I is a CUDA-based hybrid model consisting of an island GA at the upper level and a fine-grained GA at the lower level. It combines metrics of two hierarchical layers and takes full advantage of CUDA's compute ability. The parallel GA II is a dual heterogeneous design composed of a cellular GA and a pseudo GA. The islands with these two different structures increase the population diversity and can be well parallelized on GPUs simultaneously with multi-core CPU.

III.2 Problem Definition

For an easy presentation, we summarize the notations used along the rest of this Chapter in Table 8.

Table 8 A description of notations used in all formulae in Chapter III

Notation	Description
n	Number of jobs
g	Number of stages
o_s	Number of machines at stage s.

j	Job index
s	Stage index
m	Machine index
J	Set of jobs, $J = \{1,2,3,\dots,n\}$
S	Set of stages, $S = \{1,2,3,\dots,g\}$
M_s	Set of machines at the stage s, $s \in S, M_s = \{1,2,3,\dots,o_s\}$
R_j	Release time of job j, $j \in J$
D_j	Due time of job j, $j \in J$
P_{jsm}	Processing time when job j at stage s is to be processed on machine m, $j \in J, s \in S, m \in M_s$
S_{js}	Start time of job j at stage s, $j \in J, s \in S$
M_{js}	Target machine handling job j at stage s, $j \in J, s \in S$
T_j	Total tardiness, $j \in J$
C_{max}	Completion time of the last job, i.e., the makespan
WT	Weight for the total tardiness in the objective function
p	Gene index in a chromosome
ρ	Individual index
N	Number of individuals
PoP	Set of individuals, $PoP = \{0,1,2,\dots,N-1\}$
G	Set of genes, $G = \{0,1,2,\dots,n \times g - 1\}$
I	Abbreviation of individual
T	Abbreviation of thread
C	Number of islands
D	Number of individuals per islands
A, B	Island indices
α	Migration rate
β	Migration direction indicator
θ	Migration threshold value
fit_A	The best individual's fitness value of sub-population A on island A
fit_B	The best individual's fitness value of sub-population B on island B
a	$a \in (0,1)$

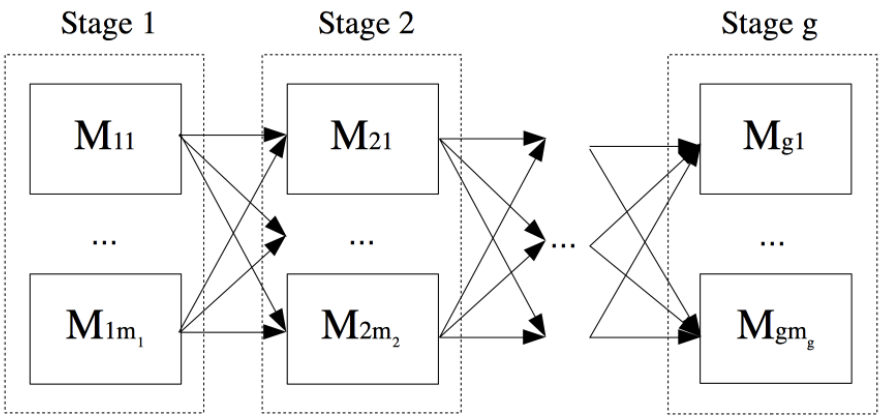

Figure 11 A flexible flow shop layout

The Flexible Flow Shop scheduling Problem (FFSP) is a multistage production process that has two or more stages in series as illustrated in Figure 11. There is at least one machine in each stage, and at least one stage has more than one machine. All jobs need to go through all stages in the same order before they are completed. On each stage, one machine is selected for processing a given operation. 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 of the FFSP is described by jobs' sequence on target machines. Furthermore, the formulation is given as follows.

Objective function:

$$\min: WT * \sum_{j \in J} T_j + C_{max} \tag{3.1}$$

Constraints:

$$T_j = \max(S_{jg} + P_{jgM_{jg}} - D_j, 0) \quad j \in J \tag{3.2}$$

$$C_{max} = \max_j(S_{jg} + P_{jgM_{jg}}) \quad j \in J \tag{3.3}$$

$$S_{j1} \geq R_j \quad j \in J \tag{3.4}$$

$$S_{js} \geq S_{j\,s-1} + P_{j\,s-1\,M_{j\,s-1}} \quad j \in J, s \in S, s > 1 \tag{3.5}$$

$$S_{js} + P_{jsM_{js}} \leq S_{is} \quad j \in J, i \in J, s \in S, j \neq i,\ M_{js} == M_{is},\ S_{js} \leq S_{is} \tag{3.6}$$

The decision variables in the mathematical model are M_{js} and S_{js}. As two scheduling objectives are considered, it is formulated as a single additive objective function (3.1) by aggregating the total tardiness and the makespan with the weight WT. As tardy jobs

typically cause penalty costs [3] and have a great influence on customer satisfaction, the weight WT indicates the priority of the first objective. Constraints (3.2) and (3.3) define the tardiness of jobs and the makespan separately. The precedence among operations due to the jobs' processing cycles is presented by constraints (3.4) and (3.5), while constraint (3.6) establishes the precedence caused by the sequencing on machines.

The population of GA consists of a set of individuals and is initialed by random values. An individual is represented by a chromosome. For the FFSP, a chromosome is composed of a string of length $n \times g$, and the p-th gene states the index of the target machine for job $\lfloor p/g \rfloor + 1$ at stage $\{p/g\} + 1$. For instance, assume that 3 jobs with 2 production stages are scheduled in a flexible flow shop and there are 2 parallel machines at each stage. A chromosome can be expressed as [1,2,1,2,2,1] which indicates [job 1 at stage 1 processed on machine 1, job 1 at stage 2 processed on machine 2, job 2 at stage 1 processed on machine 1, job 2 at stage 2 processed on machine 2, job 3 at stage 1 processed on machine 2, job 3 at stage 2 processed on machine 1]. The fitness function FIT (ρ) of an individual ρ is transferred from the above-mentioned objective function (Eq. (3.1)) as

$$\mathrm{FIT}(\rho) = \max\left(E_{max} - \left(WT * \sum_{j \in J} T_j + C_{max}\right), 0\right) \tag{3.7}$$

where E_{max} is the estimated maximum value of the objective function.

III.3 A CUDA-Based Hybrid Genetic Algorithm

III.3.1 Hybrid Model

To achieve general-purpose parallel computing on GPUs, the Compute Unified Device Architecture (CUDA) [4] was developed in 2006. It is a framework that takes the maximum advantage of the low-lying hardware using an industry standard programming language [5]. The parallel threads of CUDA are grouped into blocks that are organized in a grid as shown in Figure 12. The hierarchisation of threads is related to the memory hierarchy. There are three distinct levels of memory and their features are stated as follows [6].

- Global memory: As the largest memory of CUDA, it is accessible to all threads, but exhibits the highest latency.
- Shared memory: It enables threads only within a block whose access is much faster than the global memory.
- Local memory: It presents the lowest latency whereas it is only available to one thread with few KB of storage.

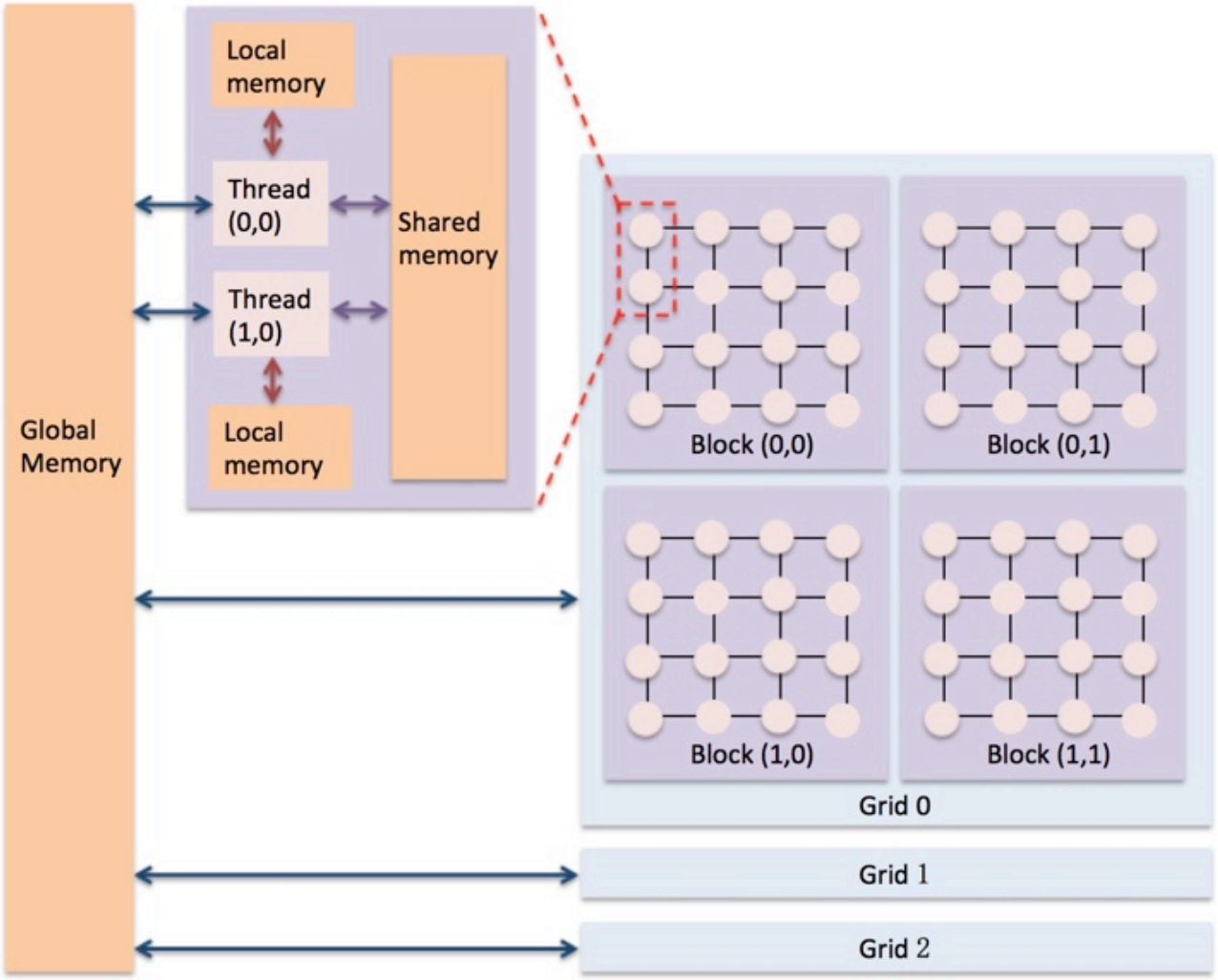

Figure 12 The hierarchy of threads and different types of memory of CUDA

With respect to the CUDA framework, our parallel GA consists of two levels, a fine-grained GA at the lower level and an island GA at the upper level, as presented in Figure 13. There is a correspondence between the hybrid parallel GA components and the hierarchy of CUDA threads and the details are displayed in Table 9. At the lower level, each CUDA thread processes one GA individual. Because of the 2D grid structure, the

GA individuals can get connected completely. Selection, crossover, mutation and fitness value calculation are generated mainly via the local memory to enjoy its lowest latency unless imperative information exchange among individuals is done through the global memory. On the other hand, one block on the CUDA framework represents one island in the GA at the upper level. An elitism based replacement inside the island and a migration among islands are carried. The shared memory is chosen to complete these works primarily while the overwriting is processed via the global memory synchronously. The procedure of the hybrid parallel GA with memory management is expressed in Figure 14.

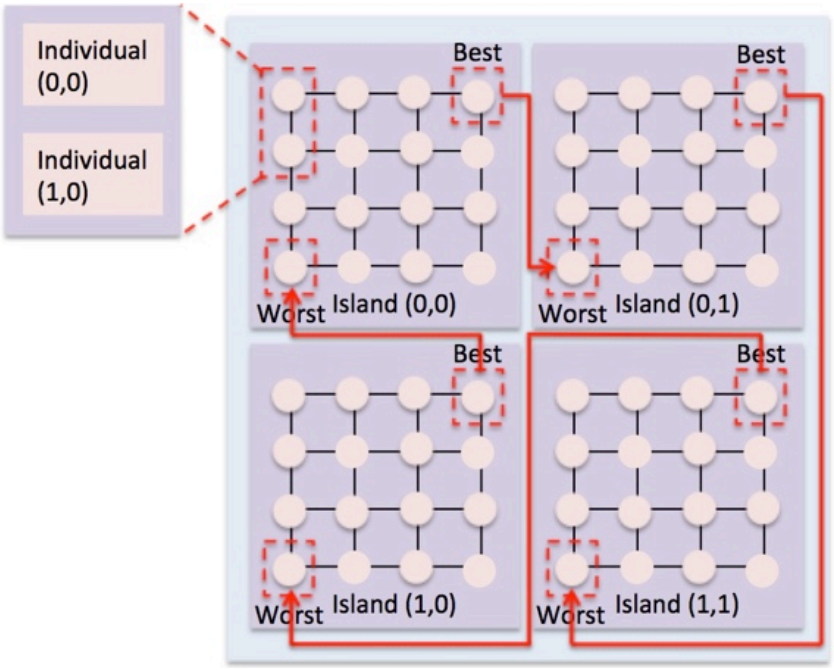

Figure 13 The hierarchy of hybrid GA

Table 9 The correspondence between the hybrid parallel GA components and the hierarchy of CUDA threads

Hybrid GA components	CUDA underlying architecture
Individual	Thread
Island	Block
Population	Grid

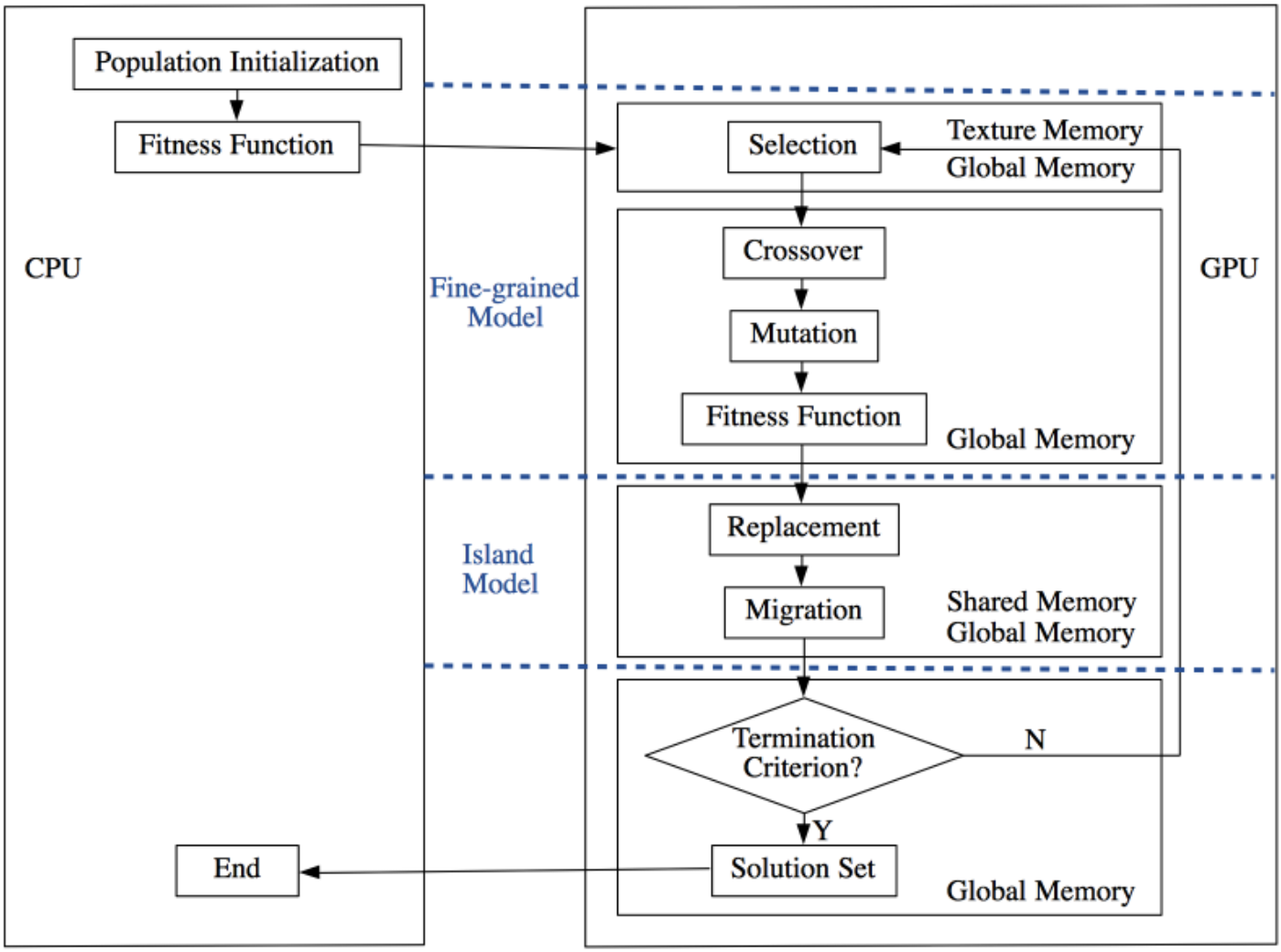

Figure 14 The procedure of hybrid parallel GA with memory management

According the evolution theory and the underlying architecture of CUDA, several advantages can be gained by the hybrid model over the simple one:

- At the lower level, the fine-grained model obtains good population diversity by dealing with high-dimensional variable spaces [7]. Limitation of interactions among individuals prevents the premature convergence. A reasonable neighborhood size with GA operators may disseminate the good solutions across the entire population.

- At the upper level, the island model increases the convergence speed by subpopulations as the long-held principle in Population Genetics: favorable traits spread faster when the demes are small than when the demes are large [1]. An

appropriate island size with a proper migration interval is able to optimize this performance.

- CUDA is built up with the 2D grid environment that matches perfectly the structure of the fine-grained GA. Thousands of CUDA threads are powerful to deal with large size individuals concurrently without increasing the time complexity. Meanwhile, GA operators are carried out using the fastest local memory.

- As CUDA threads are grouped into blocks, they are compatible to the mechanism of the island GA that divides the population into a few relatively large subpopulations. Isolated GA islands can work on CUDA blocks in parallel by the shared memory while some information sharing are executed through the global memory.

III.3.2 Genetic Algorithm Operators

- Selection: Because the 2D grid is adopted as the spatial population structure where each grid point contains one individual, the conventional selection operation is modified to suit the neighborhood configuration as in Figure 15. The selection area is defined with a certain diameter where the target individual is placed at the center of a grid. Among individuals within the selection area, the tournament selection is used where the individual with the best fitness value is selected to replace the target one.

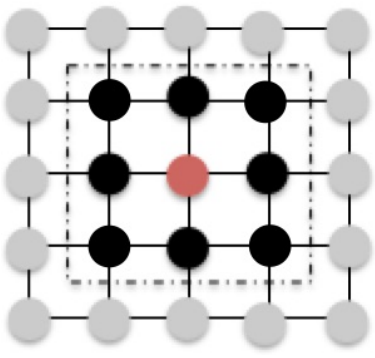

Figure 15 The selection operation of hybrid GA

- Crossover: We pair individuals with neighbors (See Figure 16) rather than selecting two from population randomly. This strategy does not require global

information sharing and is appreciated to work on the 2D grid architecture. Meanwhile, a risk that the GA converges to the local minima can be eliminated by its cooperation with the selection. Afterwards, a single point crossover is executed if a specified crossover probability is satisfied.

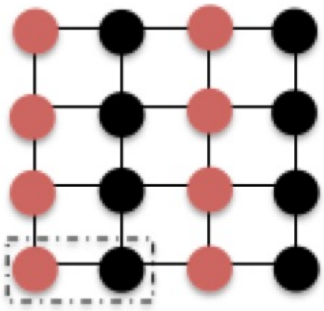

Figure 16 The crossover operation of hybrid GA

- Mutation: Any individual in the population gets a random number generated on the interval 0 to 1. If it is smaller than the default mutation rate, genes in the chromosome are replaced by random values in the range, apart from the original ones.

- Replacement: The individual whose fitness value is the largest in history within one island is kept. Then it is used to replace the individual whose fitness value is the smallest in this island. As one island is presented as one CUDA block, this operation is carried through the shared memory.

- Migration: Islands are interconnected as a single ring as shown in Figure 17. An island can only accept an individual with the largest fitness value from one neighbor to overwrite the individual with the smallest fitness value. The shared memory is utilized to search the best individual and the worst individual within one island while the overwriting is processed via the global memory synchronously.

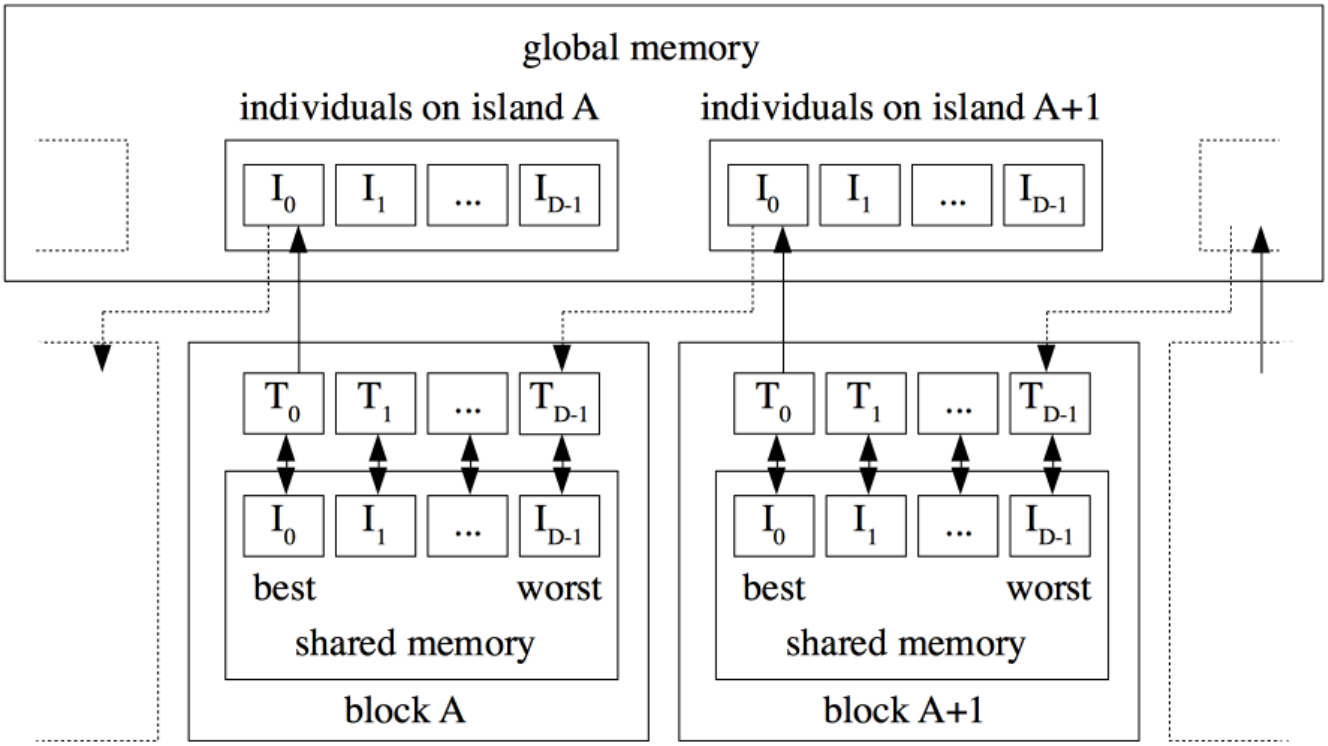

Figure 17 The single ring migration among islands

III.3.3 Numerical Experiments

To analyze the performance of parallel GA I, we consider several instances and all instances are characterized by 50 jobs with 4 stages and 3 available machines at each stage. The crossover rate and the mutation rate are set as 0.9 and 0.1, respectively. Other experimental relative data are defined in Table 10. 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 0.745GHz and 12 GB GDDR5 global memory. All programs are written in C, except for the GPU kernels in CUDA C. All results display the average value of 100 runs.

Table 10 The experimental relative data of the FFSP

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

III.3.3.1 Controlling Parameters Sensitive Analysis Test

As the maximum threads amount per block on CUDA is 1024. We keep the population size as 1024 (32×32). There are three controlling parameters in this proposed method: the island size, the selection diameter and the migration interval. They are set by different numbers: Island Size (IS) = 4 (2×2), 16 (4×4), 64 (8×8), 256 (16×16), 1024 (32×32) individuals, Selection Diameter (SD) = 3, 9, 15, 21, 27 individuals, Migration Interval (MI) = 10, 20, 30, 40, 50 generations. Figure 18 illustrates the convergence trend with combinations of different values. As the graph with all parameters' setting is a little confusing as shown by the first one. We separate it as 7 sub-graphs.

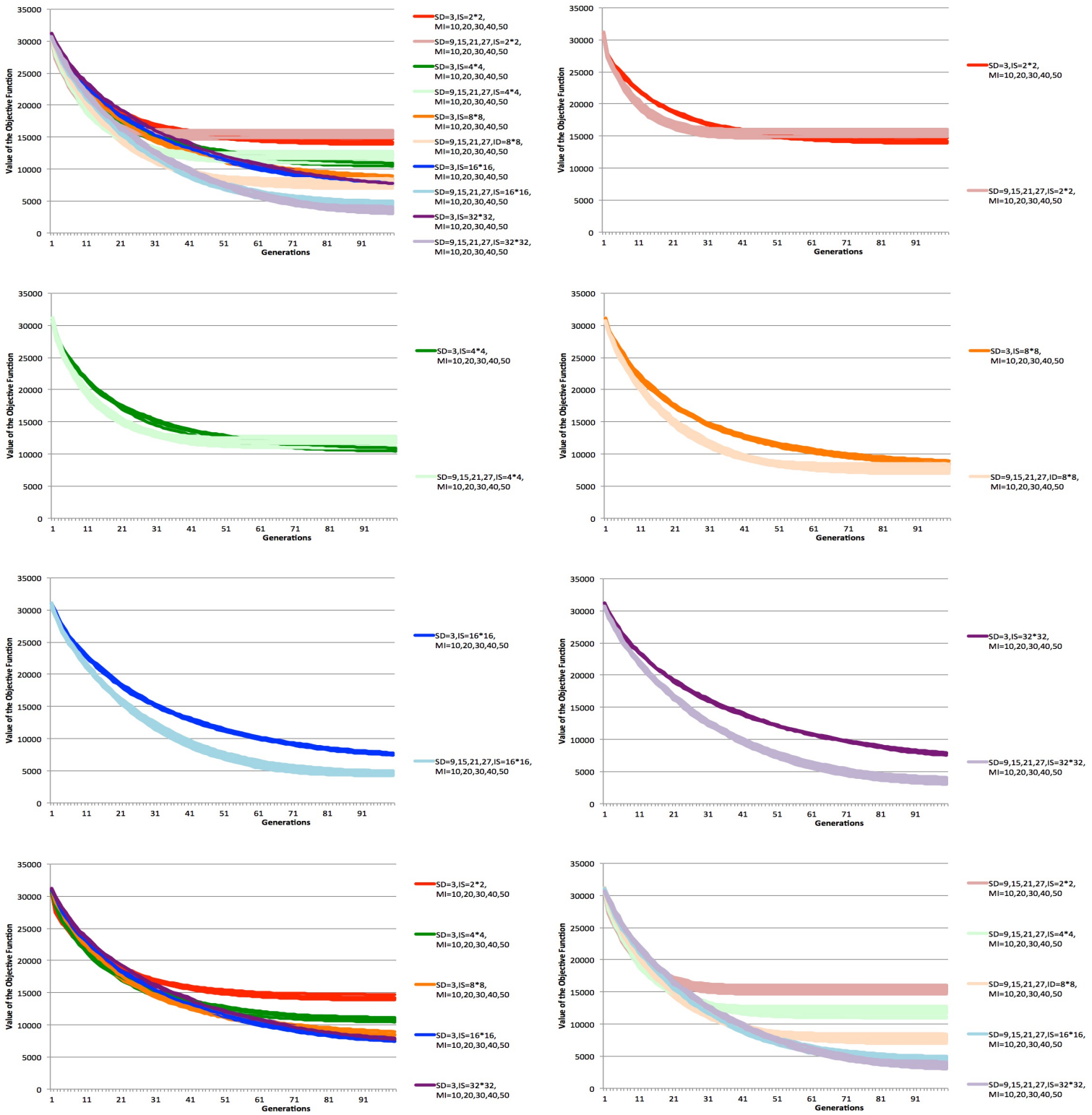

Figure 18 The sensitive analysis on controlling parameters

The island size keeps increasing from the second sub-graph to the sixth. It shows that the small selection diameter generally leads to an early convergence, whereas there is not much improvement after it reaches a medium size. Meanwhile, this influence is

more distinct when the island size is larger. As larger selection area requires larger memory, we suggest a medium size diameter value for implementations. For the following two tests, we keep its value as 9. The last two sub-graphs display the influence of the island size. As a result, a relatively larger island size makes the performance better. This trend is more obvious with groups of medium and large size selection diameters. Since the maximum threads amount per block on CUDA is 1024, the best performance is achieved by an island size 32×32. Moreover, there is no significant change when the migration interval is increased. Due to the additional cost caused by migration, it is advised not to have small migration intervals.

III.3.3.2 Comparison Test on Solution Quality

The simple GA is taken as a comparison to check the performance of parallel GA I. As the roulette wheel selection is the most frequently used selection strategy [8], we take it for the simple GA while the single point crossover is executed with randomly paired individuals. Meanwhile, the mutation operation is kept the same as the CUDA-based parallel GA. The convergence trends of the average result and the best result obtained from them are displayed in Figure 19.

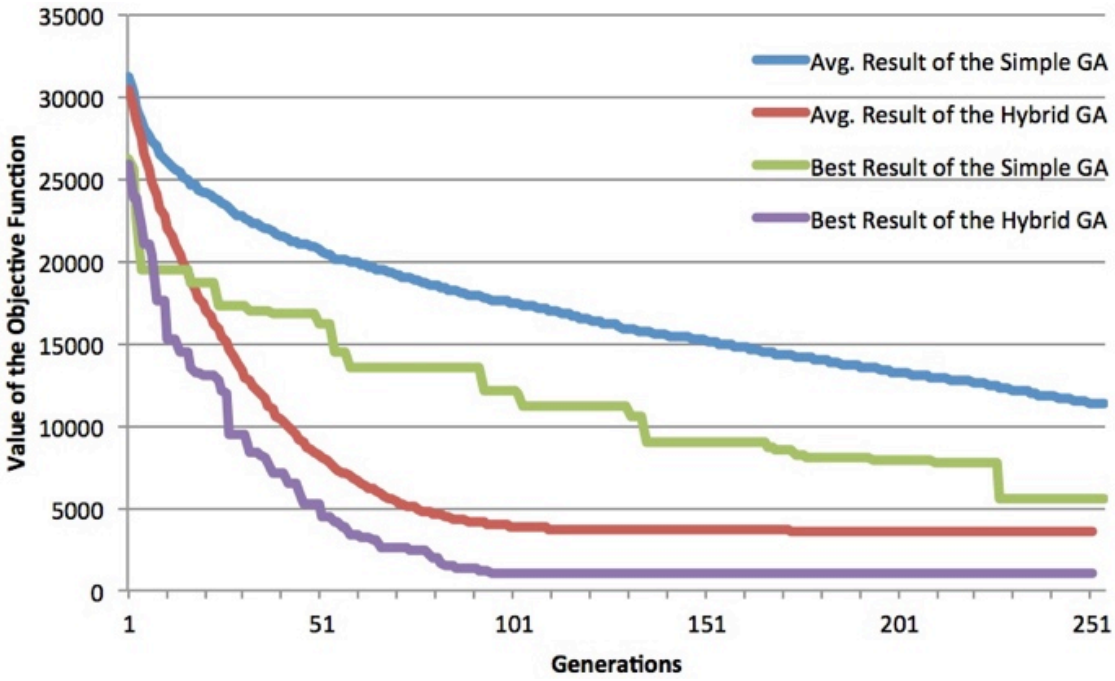

Figure 19 The solution quality comparison between the parallel GA I and the simple GA

Fine-grained models obtain good population diversity when dealing with high-dimensional variable spaces [7] and island models converge faster by subpopulations [1]. By combining the merits from them, we could find the parallel GA on CUDA always gains better performance with the average value and the best value of the objective function than the simple GA.

III.3.3.3 Comparison Test on Execution Time

For fair comparison, we do not only use the single CPU, but also take the multi-core CPU to contrast the execution time with GPUs for implementations of the simple GA, the master-slave GA and the proposed parallel GA I separately. The multi-core CPU code is run by OpenMP [9] that is an application programming interface (API) supporting multi-platform shared memory multiprocessing programming. Different implementations used to obtain the execution time are noted in Table 11.

Table 11 Different implementations used to obtain the execution time

Hgpu	Proposed hybrid parallel GA I over NVIDIA K40 GPU
Scpu	Simple GA over Intel Xeon E5640 CPU with one core
MSmulticpu	OpenMP based master-slave GA over Intel Xeon E5640 CPU with four cores
MSgpu	Master-slave GA over NVIDIA K40 GPU
Hcpu	Proposed hybrid GA over Intel Xeon E5640 CPU with one core
Hmulticpu	OpenMP based hybrid GA over Intel Xeon E5640 CPU with four cores

The speedups of the parallel GA I to the compared GAs are displayed as in Figure 20. Since parallel implementation is one of the most promising options to accelerate GAs, parallel GAs always work faster than serial GAs. Although, the proposed hybrid GA on CUDA does not win against other parallel algorithms with the small size population, the performance has been improved dramatically by increasing this latter parameter. Moreover, we expect that it can achieve further acceleration for more complicated problems.

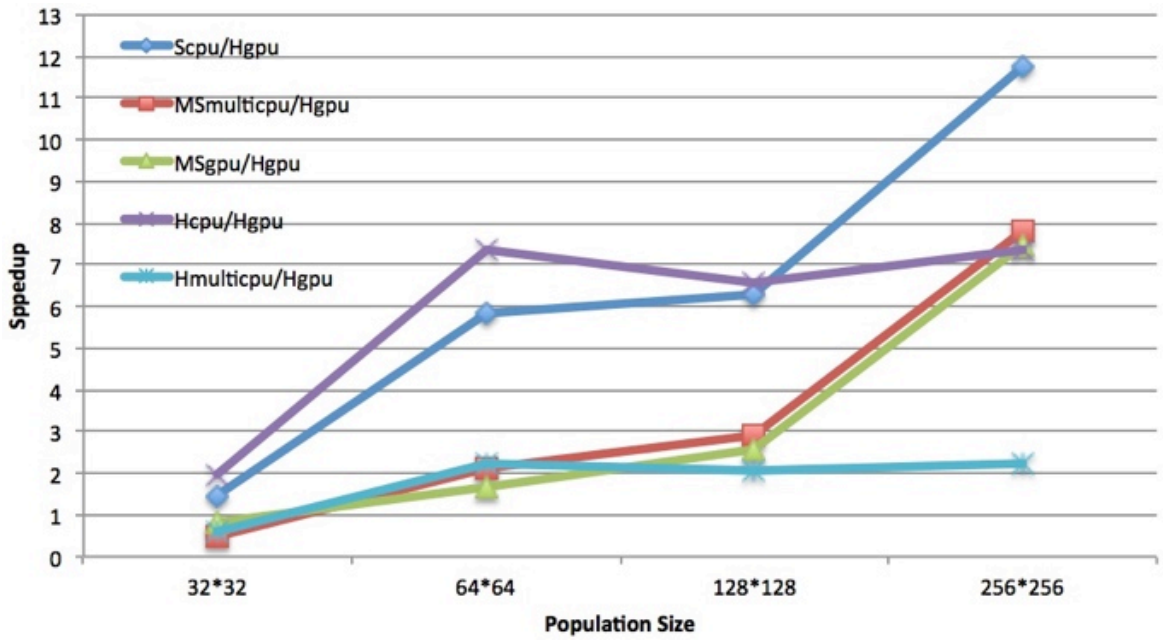

Figure 20 The execution time comparison among the parallel GA I and other implementations

III.4 A Dual Heterogeneous Genetic Algorithm

III.4.1 Dual Heterogeneous Island Strategy

Island GAs may lose the population diversity due to the limited sub-population size and the same genetic operator configurations. When the population size is N and there are C sub-populations, only N/C individuals work with GA operators in one island. Moreover, the selection and the elitist strategy in GAs decrease the sub-population diversity in one island after several generations. Although the crossover and the mutation can help create new individuals, the influence is restricted. At this moment, a migration from heterogeneous sub-populations can increase the local diversity and has a chance to produce better genes by working with GA operators. Thus, we propose a heterogeneous island strategy as in Figure 21 where the internal structures and the GA operations of each island are different.

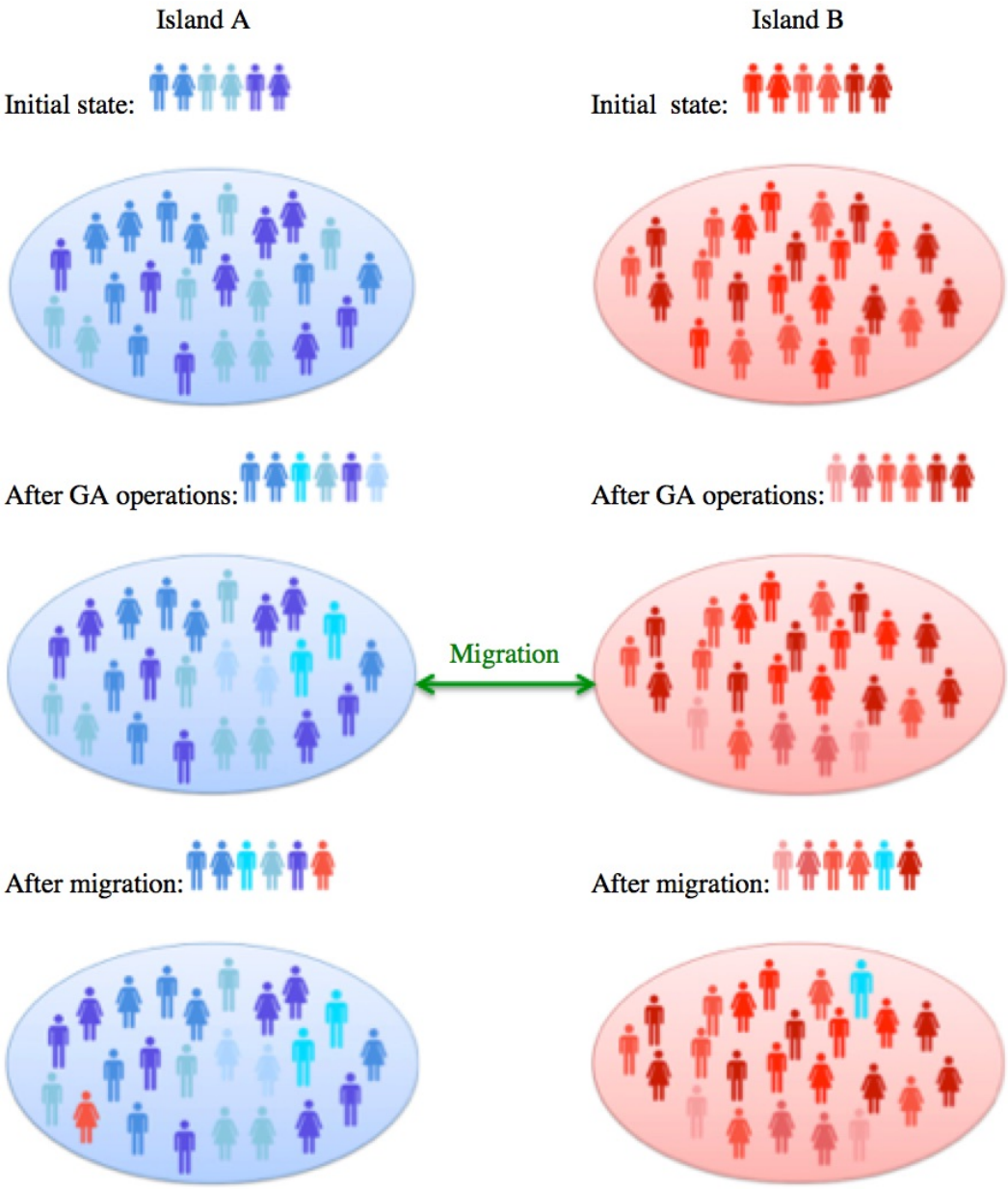

Figure 21 The dual heterogeneous island GA model

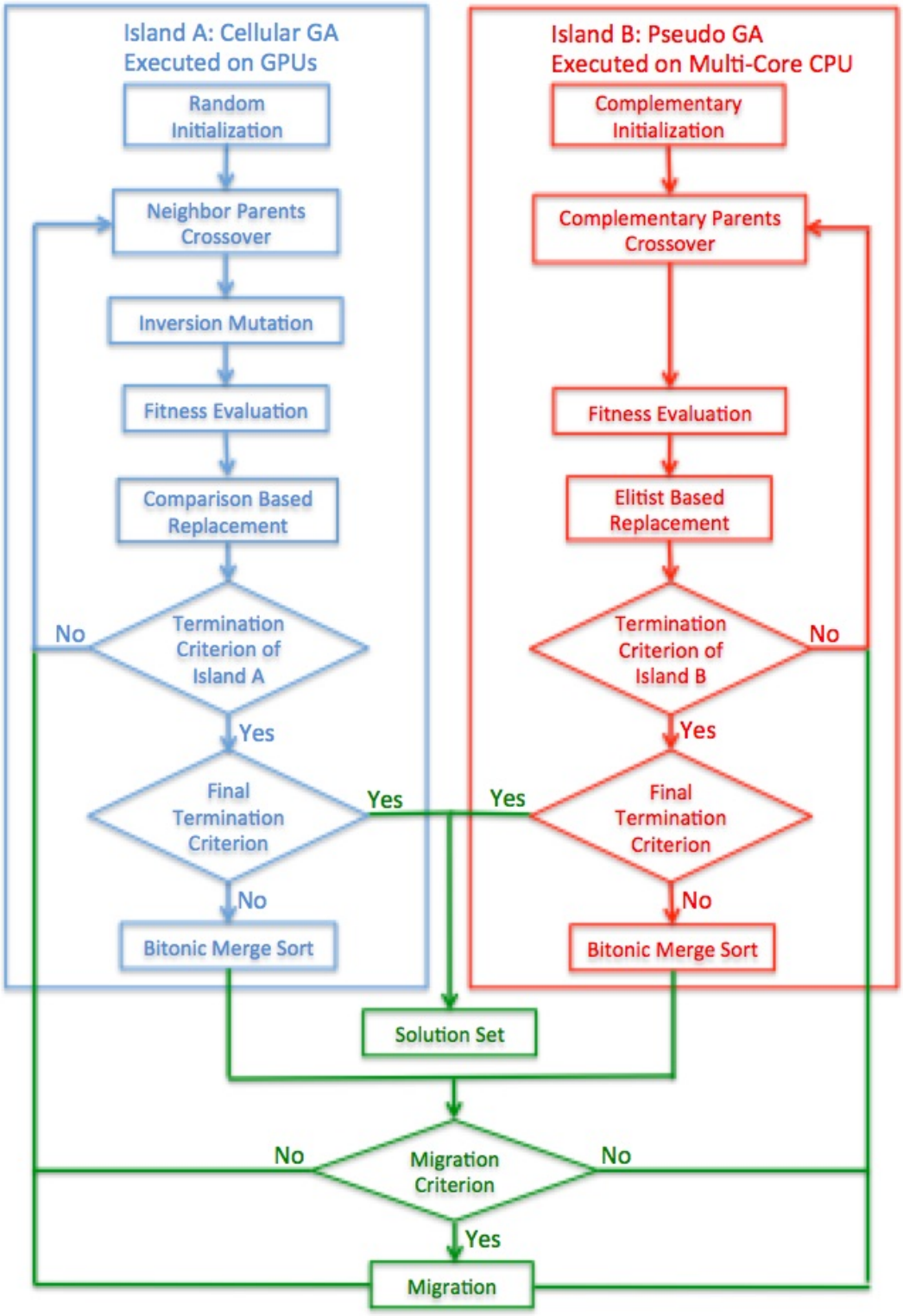

Figure 22 The procedure of dual heterogeneous island GA

The algorithm structures and the interaction between them are shown in Figure 22. At the software level, we can consider three sublevels according to the source of the heterogeneity [10].

- Operator level: There are two islands with the same amount of individuals where island A works with the cellular GA [11] and island B works the pseudo GA [12]. The cellular GA starts with random initialization and maps individuals on a two-dimensional grid. An individual is limited to compete and mate with its neighbors, while 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 initializes every pair of parents with complementary chromosomes. The crossover is executed between the offspring from the same parents, during which the parents are completely replaced by their own children. In this case, search ability is enhanced since higher population diversity is got without gene lost.

- Genotype level: The chromosome is modified to have two layers in this case. The upper layer is designed for the cellular GA and keeps the same structure as in Chapter III.2 Problem Definition where the p-th gene is displayed by an integer number. On the other hand, the p-th gene is expressed by binary numbers to work with the dynamic complementary initialization strategy [12] (See Table 12) of the pseudo GA at the lower layer.

Table 12 An example of the dynamic complementary initialization strategy

Bit	0^{th} gene	1^{st} gene	2^{nd} gene	…	p^{th}gene	…	$(n\times g-2)^{th}$gene	$(n\times g-1)^{th}$gene
Individual 0	1	1	0	…	a	…	0	0
Individual 1	0	0	1	…	1-a	…	1	1

- 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.

III.4.2 Penetration Migration Policy

The performance of migration between islands is controlled by the topology, the rate, the interval and the strategy. In order to decrease the amount of parameters need to be set manually, we develop a migration polity inspired by the penetration theory. The penetration theory [13] (See Figure 23) takes into account, the molar concentration, the temperature and the cubage of a liquor. It assumes that in mixing two kinds of liquor

with the same temperature and cubage but with different molar concentration, the liquor with the smaller concentration will move to the one with a bigger concentration value.

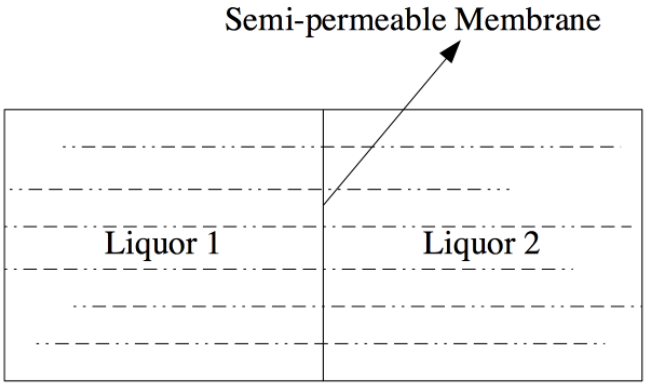

Figure 23 An example of the penetration model

Following the penetration theory, a migration threshold value θ is set ($0 \leq \theta \leq 1$). 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 equation (3.8) and equation (3.9), respectively:

$$\alpha = \begin{cases} 1-\beta & 1-\beta < \theta \\ 0 & 1-\beta \geq \theta \end{cases} \tag{3.8}$$

$$\beta = \begin{cases} \mathrm{fit_A}/\mathrm{fit_B} & \mathrm{fit_A} < \mathrm{fit_B} \\ \mathrm{fit_B}/\mathrm{fit_A} & \mathrm{fit_A} > \mathrm{fit_B} \end{cases} \tag{3.9}$$

Here, $\mathrm{fit_A}$ and $\mathrm{fit_B}$ are the best individual's fitness value of sub-population A on island A and sub-population B on island B. In a certain generation, we calculate the above functions and carry out three steps as follows:

- If $1-\beta < \theta$, the migration is executed. Otherwise, do nothing.
- The topology of migration is made by the ratio of $\mathrm{fit_A}$ to $\mathrm{fit_B}$. If $\mathrm{fit_A}/\mathrm{fit_B} > 1$, the migration is from sub-population A to sub-population 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.
- When the migration is carried, $\alpha \times N$ individuals with best fitness values in the emigrant sub-population are selected to replace $\alpha \times N$ individuals with worst

fitness values in the immigrant sub-population.

With this design, the migration policy needs no longer to fix the topology, the rate, the interval and the strategy manually. New merged individuals with good genes can be transited fast and the execution time is saved by preventing ineffective information sharing.

III.4.3 Parallelization on GPUs and multi-core CPU

As far as the hardware level, there are two obvious benefits to parallelize the dual heterogeneous island GA in GPUs and multi-core CPU environment:

- Widespread HPC resources: Nowadays, almost all modern computers are equipped with GPUs and multi-cores CPUs. The coordination between them is through their inner connection which is stable and secure. With the development of CUDA, it is convenient to use enabled GPUs for general purpose processing. On the other hand, concurrency platforms allowing coordination of multicore resources facilitate programming on multi-core CPUs. Moreover, in addition to the parallelization on multi-core CPU or GPUs at the lower level, the multi-core CPU and the GPUs can work concurrently at the higher level to maximally use computing resources.
- High consistency with the proposed GA: The cellular GA maps individuals on a two-dimensional grid, it can be entirely parallelized on GPUs. 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 for loops in the above two steps can be easily handled on multi-core CPU in parallel.

As texture catches of CUDA are designed to gain an increase in performance accelerating access patterns with spatial locality [14], we design the neighborhood area of the cellular GA as shown in Figure 24. Individuals' information and GA operators are placed and executed through the global memory while the neighbors' information are stored in the texture memory. Each CUDA thread handles one cellule of the cellular GA. It firstly recombines two individuals selected from the neighborhood area to

generate a new one. Afterwards, this new individual undertakes the mutation as designed in parallel GA I and replaces the original individual if its solution is better. Finally, individuals are sorted according to their fitness values using the Bitonic-Merge sort [15] (See Figure 25), if the cellular GA meets the island termination criterion but not the final termination criterion.

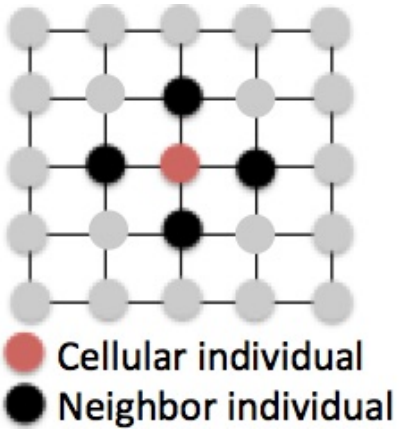

Figure 24 The neighborhood area of cellular GA

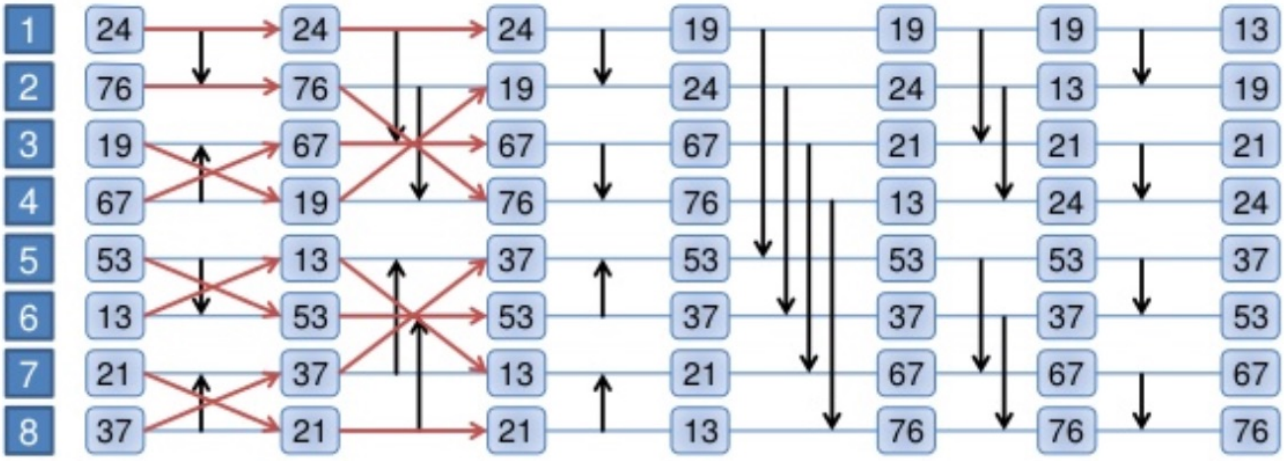

Figure 25 An example of the Bitonic-Merge sort

With the two layers encoding, the target machine index is presented by binary numbers at the lower level that is compatible with the dynamic-complementary initialization strategy [12]. When the GPUs are occupied by executing the cellular GA, the pseudo GA is run on multi-core CPU by OpenMP. In this case, the GA operators on two heterogeneous islands are working in parallel on the host (multi-core CPU) and the device (GPUs) simultaneously. At the end, the Bitonic-Merge sort [14] is accomplished by the OpenMP-based code in a similar way as the cellular GA on CUDA.

III.4.4 Numerical Experiments

To analyze the performance of parallel GA II, we compare its solutions' quality and execution time with the cellular GA and the pseudo GA. The population size is kept as 512 for all tested GAs while the sub-population size for each island of heterogeneous GA is 256. The crossover rate and the mutation rate of cellular GA are set as 1.00 and 0.05 respectively [11], while the crossover rate of pseudo GA is equal 0.75 [12]. 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 check the influence of migration, the migration threshold is fixed as 1.00. All analyzed instances are characterized by 300 jobs with 4 stages and 2 available machines at each stage. Other experimental relative data and the experimental platform are kept the same as the parallel GA I. The following table and figures display results of 2000 generations and they are average values of 50 runs.

III.4.4.1 Migration Check Interval Test

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 26, 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 onto premature convergence, besides homogeneous genetic 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.

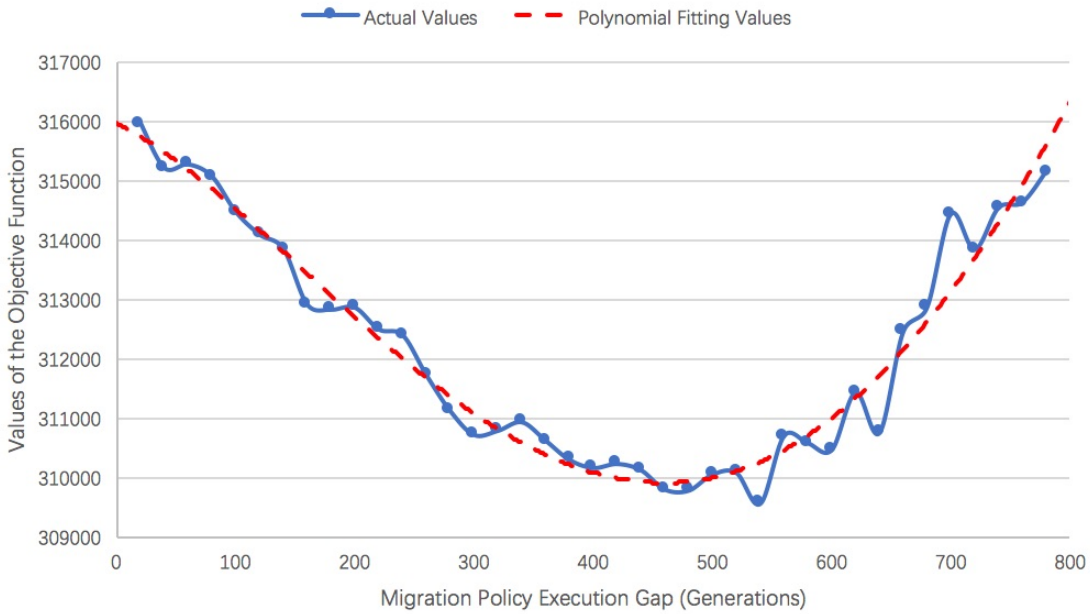

Figure 26 The influence of the migration policy execution gap for the heterogeneous GA

III.4.4.2 Comparison Test on Solution Quality

The solutions' quality of different GAs are shown in Table 13. Although the specific designs of cellular GA and the pseudo GA can help increase population diversity, the proposed method combines the merits from both and optimizes the performance by independent evolution and penetration migration. Thus, the heterogeneous GA overcomes them with better solutions and less variance. This effect is also confirmed by the convergence trend among three GAs in Figure 27. Moreover, there are elbows in the convergence curve of the designed approach 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 connected by the penetration migration.

Table 13 The solutions' quality comparison among the parallel GA II and other GAs

Different GAs	Best	Average	Variance
Heterogeneous GA	306500.03	309885.90	2003059.14
Cellular GA	314467.50	320648.18	6792896.04
Pseudo GA	314636.59	317683.23	2963668.96

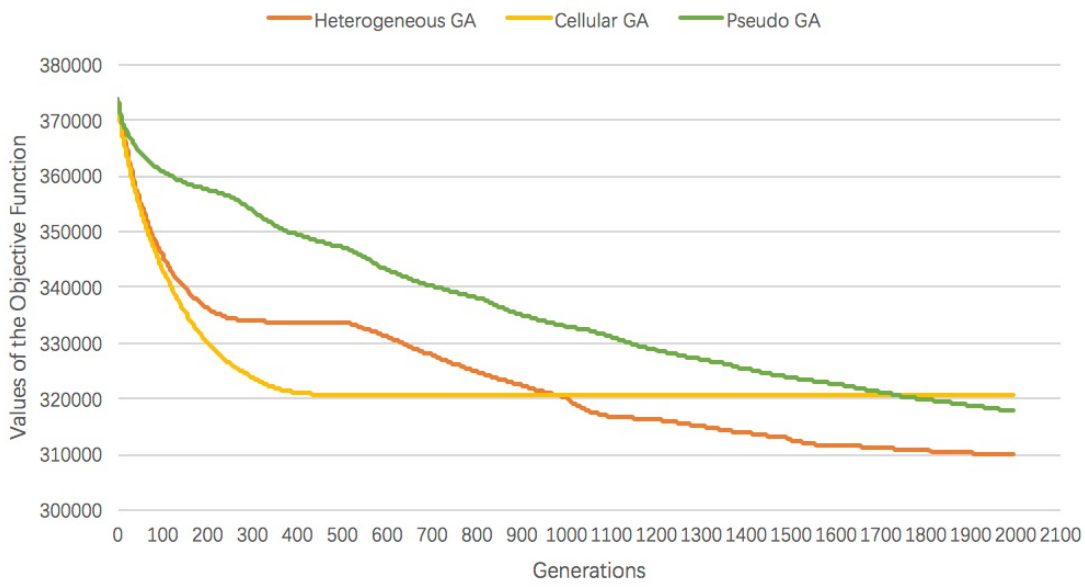

Figure 27 The convergence trend among the parallel GA II and other GAs

III.4.4.3 Comparison Test on Execution Time

Figure 28 displays the execution time of three GAs with increasing population size. It is easy to discover the heterogeneous island GA takes less execution time than the pseudo GA on multi-core CPU as the heterogeneous design can be well parallelized on GPUs simultaneously with multi-core CPU. However, the cellular GA performs best because the amount of individuals executed on GPUs and the threads occupancy are twice as much as the heterogeneous GA.

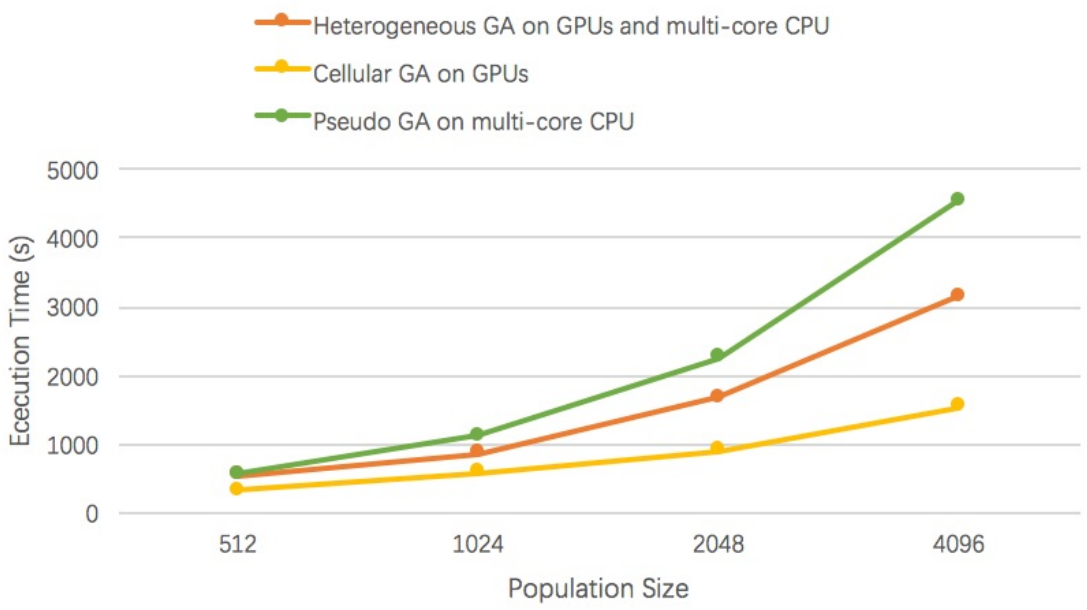

Figure 28 The execution time comparison among the parallel GA II and other GAs

III.5 Conclusions

Two efficient new parallel GAs are developed in this Chapter. The parallel GA I consists of an island model at the upper level and a fine-grained model at the lower level. This hybrid design is highly consistent with the CUDA framework and combines metrics from two levels by keeping population diversity and increasing convergence speed. The controlling parameters configuration test witnesses that the performance of this method is optimized with a medium size selection diameter, a relatively large island size and a wide range size migration interval. Through numerical experiments, the parallel GA I overcomes the simple GA by obtaining better results and taking less execution time. The parallel GA II is working with a dual heterogeneous island model that is composed of a cellular GA on GPUs and a pseudo GA on multi-core CPU. The 2D variable spaces of the cellular model and the complementary parent strategy of the pseudo model keep the population diversity while a penetration inspired migration policy helps share information between them. Furthermore, this heterogeneous design can be well parallelized on GPUs simultaneously with multi-core CPU. For solving some instances of FFSP, numerical experiments have shown that this approach cannot only get competitive results but also reduces execution time.

Reference

[1].Cantú-Paz, E. (1998). A survey of parallel genetic algorithms. Calculateurs paralleles, reseaux et systems repartis, 10(2), 141-171.

[2].García-Valdez, M., Trujillo, L., Merelo-Guérvos, J. J., & Fernández-de-Vega, F. (2014, September). Randomized parameter settings for heterogeneous workers in a pool-based evolutionary algorithm. In International Conference on Parallel Problem Solving from Nature (pp. 702-710). Springer, Cham.

[3].Parmee, I. C. (2009). Adaptive Computing in Design and Manufacture.

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

[5].Munawar, A., Wahib, M., Munetomo, M., & Akama, K. (2009). Hybrid of genetic algorithm and local search to solve MAX-SAT problem using NVIDIA CUDA framework. Genetic Programming and Evolvable Machines, 10(4), 391.

[6].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, 46(3) 589-606

[7].Kohlmorgen, U., Schmeck, H., & Haase, K. (1999). Experiences with fine-grained parallel genetic algorithms. Annals of Operations Research, 90, 203-219.

[8].Zhong, J., Hu, X., Zhang, J., & Gu, M. (2005, November). Comparison of performance between different selection strategies on simple genetic algorithms. In Computational Intelligence for Modelling, Control and Automation, 2005 and International Conference on Intelligent Agents, Web Technologies and Internet Commerce, International Conference on (Vol. 2, pp. 1115-1121). IEEE.

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

[10].Alba, E., Luna, F., Nebro, A. J., & Troya, J. M. (2004). Parallel heterogeneous genetic algorithms for continuous optimization. Parallel Computing, 30(5-6), 699-719.

[11].Vidal, P., & Alba, E. (2010). Cellular genetic algorithm on graphic processing units. In Nature Inspired Cooperative Strategies for Optimization (NICSO 2010) (pp. 223-232). Springer, Berlin, Heidelberg.

[12].Chen, Q., Zhong, Y., & Zhang, X. (2010). A pseudo genetic algorithm. Neural Computing and Applications, 19(1), 77-83.

[13].Gu, J., Gu, X., & Gu, M. (2009). A novel parallel quantum genetic algorithm for stochastic job shop scheduling. Journal of Mathematical Analysis and Applications, 355(1), 63-81.
[14]. J. Sanders, E. Kandrot, CUDA by example: an introduction to general-purpose GPU programming, Addison-Wesley Professional, 2010.
[15].Pharr, M., & Fernando, R. (2005). Gpu gems 2: programming techniques for high-performance graphics and general-purpose computation. Addison-Wesley Professional.

Chapter IV. Parallel GA I with Periodic Complete Rescheduling for Solving an Energy Efficient Dynamic FFSP Using the Peak Power Value

IV.1 Introduction

About one half of the world's total energy is currently consumed by the industrial sector [1] and its energy consumption has nearly doubled over the last 60 years [2]. Thus energy efficiency is becoming an essential parameter of industrial manufacturing processes, mostly due to new government legislation, customers' environmental concerns and continuously rising cost of energy. Because of a growing economical competitive landscape and higher environmental norms, it is now vital for manufacturing companies to reduce their energy consumption and to become more environment-friendly. To limit peak power consumption is one important approach to support energy efficient production, because electricity consumption and operating costs of manufacturing plants are usually charged based on the peak power demand from electricity providers [3]. This method takes load shifting to reduce energy use at the utility's peak demand [4]. Fang et al. [5] presented a multi-objective mixed-integer programming model of the flow shop scheduling problem that considers peak power load, energy consumption, and associated carbon footprint in addition to cycle time. Bruzzone et al. [6] proposed the integration of an energy aware scheduling module with an advanced planning and scheduling system in order to control the peak consumption, while accepting a possible increase in the total tardiness. Xu et al. built two mixed-

integer programming models in [3] to achieve a global optimal solution between the peak power and the traditional production efficiency without compromise on computing efficiency. To sum up, numerous works have focused on shop scheduling problems with peak power limitation in static perspective. But, due to frequently inevitable new arrival jobs in the production environment, a fixed preset scheduling plan could not meet the requirement.

Scheduling problems are dynamic in the real world with uncertain expected events after the start time. Dynamic scheduling problems are more complex than static scheduling problems. A lot of methods have been utilized to solve this kind of problems [7]. However, only a few of them considered the efficiency of shop scheduling problem with energy efficient demand and they were generally solved by the predictive reactive approach with complete rescheduling. Tang et al. [8] adopted an improved particle swarm optimization to search for the Pareto optimal solution of dynamic flexible flow shop scheduling problems that minimize energy consumption and makespan. A predictive reactive strategy was used to allocate the new jobs and the previous remaining operations simultaneously after the rescheduling point. Zhang et al. studied the dynamic rescheduling in flexible manufacturing systems considering energy consumption and schedule efficiency in [9] with a new goal programming math model. Optimal solutions were found by a genetic algorithm with the complete rescheduling strategy and the period policy. In a word, some efforts to solve energy efficient dynamic shop scheduling problems have been carried out. However, limitations still remain and must be tackled. A typical one is to obtain the renewed adequate scheduling plan in a reasonable response time. Particularly, complete rescheduling requires prohibitive computation even it performs the best to maintain optimal solutions.

In recent years, various algorithms using GPUs have been successfully applied to generate optimized results for shop scheduling problems with impressive time decrease. In addition to the already stated works in Chapter II, Melab et al. [10] indicated a parallel branch and bound algorithm based on a GPU-accelerated bounding model on flow shop scheduling benchmarks to improve the performance by optimizing data access management. Czapinski et al. [11] implemented a Tabu search method with GPUs for the solutions of permutation flow shop scheduling problems, which is 89 times faster than the CPU version. These cases have confirmed that the parallel

algorithms on GPUs have good performance in solving shop scheduling problems. However, it is also revealed that few studies have been conducted to integrate GPUs computing in dynamic energy efficient shop scheduling problems, because of the complexity that is caused.

As far as the above-mentioned requirements, the total tardiness and the makespan with a peak power limitation are analyzed in this Chapter while considering the flexible flow shop with new arrival jobs. A periodic complete rescheduling approach is adopted to represent the optimization problem. Furthermore, due to the fact that an adequate renewed scheduling plan needs to be obtained in a short response time in dynamic environment, a priority based hybrid parallel GA on GPUs is implemented. The efficiency and the effectiveness of the proposed approach are validated through computational tests.

IV.2 Problem Definition

For an easy presentation, we summarize the notations used along the rest of this Chapter in Table 14.

Table 14 A description of notations used in all formulae in Chapter IV

Notation	Description
j, i, i'	Job indices
s, s', s''	Stage indices
m	Machine index
t	Time period index
n	Number of original jobs
n'	Number of new arrival jobs
r	Number of original operations assigned to machines before the rescheduling point
g	Number of stages
o_s	Number of machines at the stage s.
H	Time horizon
J	Set of original jobs, $J = \{0,1,2, \dots, n-1\}$

J'	Set of new arrival jobs, $J' = \{0,1,2, \dots, n' - 1\}$
S	Set of stages, $S = \{0,1,2, \dots, g - 1\}$
M_s	Set of machines at the stage s, $s \in S, M_s = \{0,1,2, \dots, o_s - 1\}$
T	Set of time periods, $T = \{1,2,3, \dots,H\}$
RS	Rescheduling point
R_j	Release time of job j, $j \in J \cup J'$
D_j	Due time of job j, $j \in J \cup J'$
P_{jsm}	Processing time when job j at stage s is to be processed on machine m, $j \in J \cup J', s \in S, m \in M_s$
Q_{jsm}	Average power consumption when job j at stage s is to be processed on machine m, $j \in J \cup J', s \in S, m \in M_s$
Q_{max}	Power's peak
WT	Weight for the total tardiness in the objective function
u_{jst}	Boolean variable, $j \in J \cup J', s \in S, t \in T$
S_{js}	Start time of job j at stage s, $j \in J \cup J', s \in S$
M_{js}	Target machine handling job j at stage s, $j \in J \cup J', s \in S$
Q_t	Total power consumption at time period t, $t \in T$
T_j	Total tardiness, $j \in J \cup J'$
C_{max}	Completion time of the last job, i.e., the makespan
k	Current generation number of the GA
$X(k)$	Target machine matrix at generation k
$Y(k)$	Priority matrix at generation k
$Z(k)$	Order matrix at generation k
C	A very large constant, $C \epsilon R_+$
ρ	Individual index

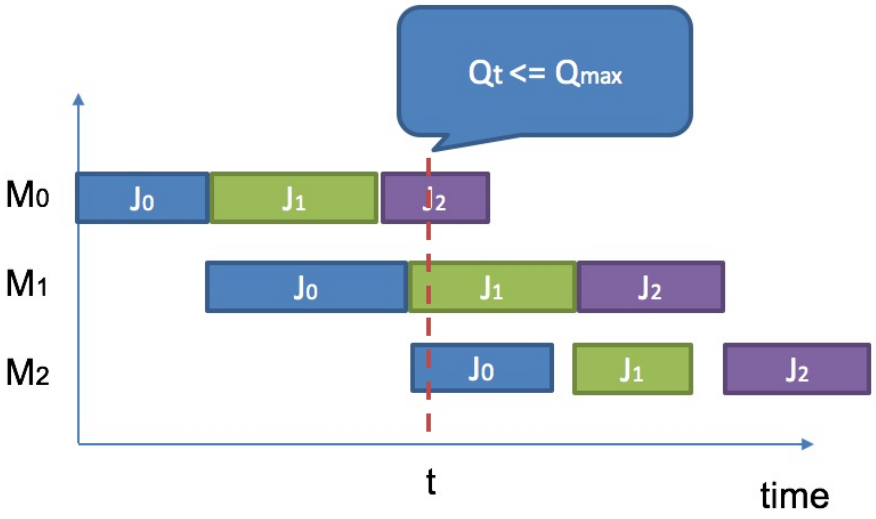

Figure 29 An example of EDFFSP using the peak power value

An Energy efficient Dynamic Flexible Flow Shop scheduling Problem (EDFFSP) is a further development of the FFSP. A set of new jobs may arrive after the start of the original plan. They should be processed sequentially and non-preemptively from the beginning of the rescheduling point with the remaining uncompleted operations of original jobs. In addition to the processing time, the power consumption of one step of J_i (J_i') on a particular machine is known. Furthermore, there is a power's peak limitation when the system operates as illustrated in Figure 29. As an FFSP is considered to be NP-hard in essence and difficult to solve, the EDFFSP is a NP-hard combinatorial optimization problem and more complex than the FFSP. To achieve the power's peak limitation and minimize the traditional makespan and the total tardiness objectives, the formal mathematical model for the EDFFSP is an extension of the mathematical model presented in [3, 6] to cover rescheduling. The formulation is given in the following.

Objective function:

$$\min: WT * \sum_{j \in J \cup J'} T_j + C_{max} \tag{4.1}$$

Constraints:

$$T_j = \max(S_{j\,g-1} + P_{j\,g-1 M_{j\,g-1}} - D_j, 0) \quad j \in J \cup J' \tag{4.2}$$

$$C_{max} = \max_j(S_{j\,g-1} + P_{j\,g-1\,M_{j\,g-1}}) \quad j \in J \cup J' \tag{4.3}$$

$$S_{j0} \geq R_j \quad j \in J \cup J' \tag{4.4}$$

$$S_{js} \geq S_{j\,s-1} + P_{j\,s-1\,M_{j\,s-1}} \quad j \in J \cup J', s \in S, s > 0 \tag{4.5}$$

$$S_{js} + P_{jsM_{js}} \leq S_{is} \quad j \in J \cup J', i \in J \cup J', s \in S, j \neq i,\ M_{js} == M_{is},\ S_{js} \leq S_{is} \tag{4.6}$$

$$Q_{max} \geq Q_t \quad t \in T \tag{4.7}$$

$$Q_t = \sum_{j \in J \cup J'} \sum_{s \in S} Q_{jsMjs} * u_{jst} \quad t \in T \tag{4.8}$$

$$u_{jst} = \begin{cases} 1 & j \in J \cup J', s \in S, S_{js} \leq t < S_{js} + P_{jsM_{js}} \\ 0 & otherwise \end{cases} \tag{4.9}$$

$$RS \leq S_{js} \quad j \in J \cup J', s \in S \tag{4.10}$$

The decision variables in this mathematical model are M_{js} and S_{js}. The objective function (4.1) and constraints (4.2)-(4.6) keep the same as the FFSP stated in Chapter III.2 but including new arrival jobs. Constraint (4.7) introduces the power's peak by an upper bound whereas the power consumption during a certain period is expressed by constraint (4.8). Constraint (4.9) gives the definition of a Boolean variable u_{jst}. It is equal to 1 if job j at stage s is being processed at time period t. Finally, constraint (4.10) imposes the definition of rescheduling.

IV.3 Solving Approach

IV.3.1 Periodic Complete Rescheduling Strategy

In the periodic policy, schedules are re-generated at the rescheduling points that occur with regular intervals and gather all new arrival jobs' information. To solve the EDFFSP, operations are assigned to machines in order, following the original schedule until the reschedule point. New arrival jobs and uncompleted operations of original jobs are processed in terms of the updated schedule executed by the optimization algorithm within a short response. The parallel GA I is chosen for solving this problem with a complete rescheduling strategy which is better in maintaining optimal solutions, but is rarely achievable in practice due to the prohibitive computation time [12]. Figure 30 summarizes the flow of the periodic complete rescheduling process.

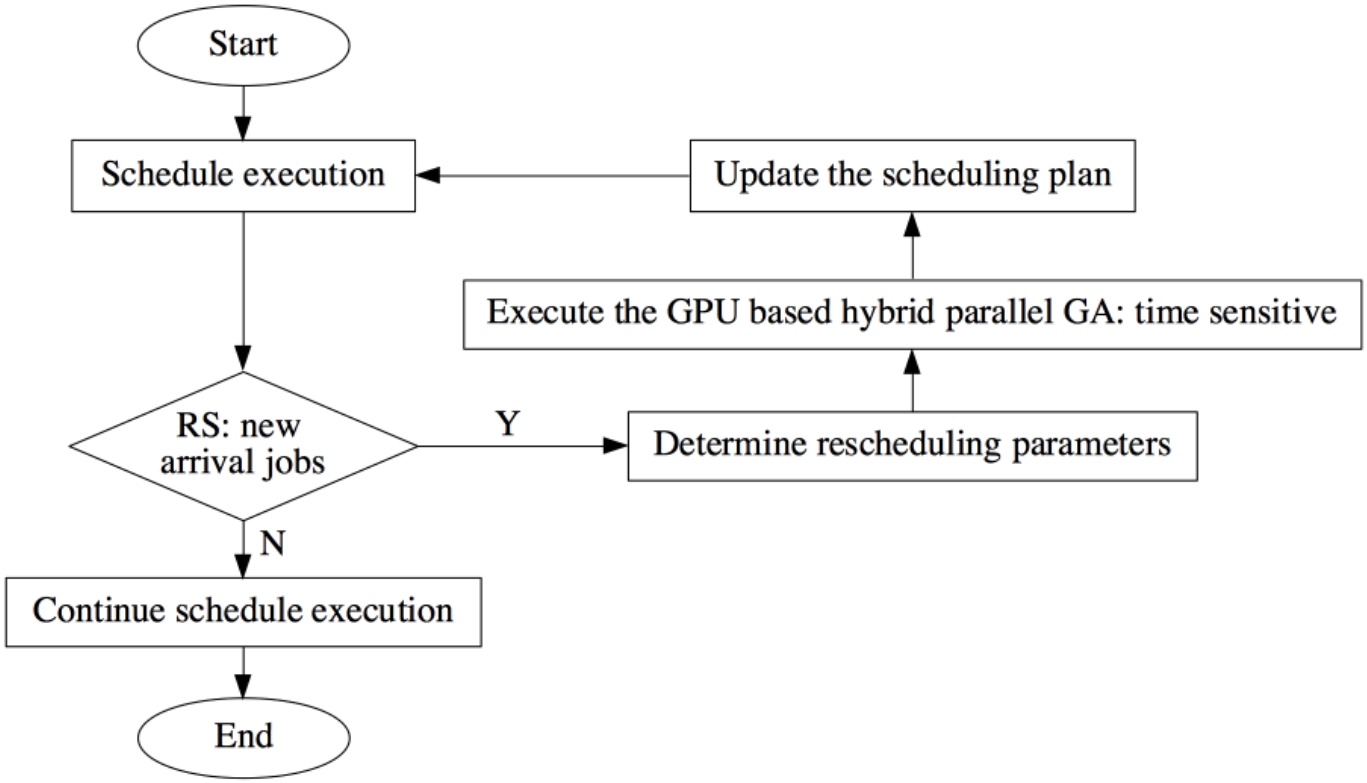

Figure 30 The flow of periodic complete rescheduling process for the EDFFSP

IV.3.2 Priority-Based Encoding Representation

According to the problem description in Chapter IV.2, a target machine matrix X(k), stored on the GPU global memory with $n + n'$ rows and g columns, is presented in (4.11).

$$X(k) = \begin{bmatrix} x_{00}(k) & x_{01}(k) & \cdots & x_{0(g-1)}(k) \\ x_{10}(k) & x_{11}(k) & \cdots & x_{1(g-1)}(k) \\ \vdots & \vdots & x_{js}(k) & \vdots \\ x_{(n+n'-1)0}(k) & x_{(n+n'-1)1}(k) & \cdots & x_{(n+n'-1)(g-1)}(k) \end{bmatrix} \quad (4.11)$$

Where $x_{js}(k) \epsilon [0, o-1] \cup \{-1\}, j \in J \cup J', s \in S$.

Moreover, (4.12) shows a $(n + n') \times g$ matrix placed on the GPU global memory that expresses the priority relation among operations.

$$Y(k) = \begin{bmatrix} y_{00}(k) & y_{01}(k) & \cdots & y_{0(g-1)}(k) \\ y_{10}(k) & y_{11}(k) & \cdots & y_{1(g-1)}(k) \\ \vdots & \vdots & y_{js}(k) & \vdots \\ y_{(n+n'-1)0}(k) & y_{(n+n'-1)1}(k) & \cdots & y_{(n+n'-1)(g-1)}(k) \end{bmatrix} \quad (4.12)$$

Where $y_{js}(k) \epsilon [1, g\times(n+n') - r] \cup \{-1\}, j \in J \cup J', s \in S$.

Each element of matrix X(k) indicates the machine number that deals with job j at stage s at generation k while each element of matrix Y(k) is used to sequence operations assigned to machines. The values for the EDFFSP are defined as:

- if job j at stage s is started or completed before the start time of the rescheduling point, both element $x_{js}(k)$ and element $y_{js}(k)$ are equal to -1. This includes:

  Case 1: job j at stage s of the original job is accomplished.

  Case 2: job j at stage s of the original job is being executed.

- if job j at stage s is assigned to a machine after the start time of the rescheduling point, element $x_{js}(k)$ is equal to a random integer representing the target machine handling job j at stage s. Similarly, elements $y_{js}(k)$ is also generated randomly from the range starting from 1 to the amount of unassigned operations. Moreover the value of element $y_{js}(k)$ is unique, where the larger the value of the random integer represents higher priority. This includes:

  Case 1: job j at stage s of the original job remains to be processed.

  Case 2: job j at stage s of the new arrival job must be processed.

In this representation, each chromosome of the parallel GA consists of one target machine matrix and one priority matrix, representing a feasible schedule. In the decoding step, elements of a matrix Z(k) (4.13) generated from the matrix X(k) and the matrix Y(k) are designed to address the assignment order of uncompleted operations. Element $z_{js}(k)$ is equal to 0 if job j at stage s of the original job is being executed at the start time of the rescheduling point, while element $z_{js}(k)$ is equal to C if the operation is accomplished before it. The procedure to determine elements' value of matrix Z(k) is displayed in Algorithm 1 and all these values are reserved on the GPU

global memory. When the power's peak is met, the later assigned operation needs to be delayed as shown by the decoding rule in Algorithm 2.

$$Z(k)=\begin{bmatrix} z_{00}(k) & z_{01}(k) & \cdots & z_{0(g-1)}(k) \\ z_{10}(k) & z_{11}(k) & \cdots & z_{1(g-1)}(k) \\ \vdots & \vdots & z_{js}(k) & \vdots \\ z_{(n+n'-1)0}(k) & z_{(n+n'-1)1}(k) & \cdots & z_{(n+n'-1)(g-1)}(k) \end{bmatrix} \quad (4.13)$$

Where $z_{js}(k)\epsilon[1, g\times(n+n')-r]\cup\{0, C\}, j\in J\cup J', s\in S$.

Algorithm 1The procedure for determining elements' value of matrix Z(k)

For $s, s', s''\epsilon S,\ s\neq s'', j, i\in J\cup J', j\neq i, m\in M$

if $x_{js}(k)=-1$ **then**

  if $S_{js}<RS<S_{js}+P_{jsm}$ **then**

    $z_{js}(k)=0$, machine m continues to process job j at stage s before executing a rescheduling plan.

  else

    $z_{js}(k)=C$.

  end if

else

  if $y_{js}(k)>y_{is'}(k)$ **then**

    $z_{js}(k)<z_{is'}(k)$, job j at stage s is assigned to its target machine earlier than job i at stage s'.

  end if

  if $s<s''$ **then**

    $z_{js}(k)<z_{js''}(k)$, job j at stage s is assigned to its target machine earlier than job j at stage s''.

  end if

end if

Algorithm 2 The decoding rule of the EDFFS

For $s, s' \epsilon S,\ j, i, i' \in J \cup J', j \neq i \neq i', z_{js}(k) < z_{is}(k),\ z_{i's'}(k) < z_{js}(k)$ && job i' at stage s' is the earliest finished one among all the processing operations at period t, $m \in M, t \in T$

if $Q_{max} \geq Q_t + Q_{jsM_{js}}$ **then**

  if $M_{js} == M_{is}$ **then**

    job j at stage s is assigned to machine m earlier than job i at stage s.

  else

    jobs are assigned to each machine in terms of matrix X(k).

  end if

else

  job j at stage s needs be delayed to be assigned to its target machine until finishing job i' at stage s'.

end if

An example of EDFFSP is presented in Table 15. There are 6 original jobs. Each job consists of 3 stages and there are two machines at each stage. Jobs are available to be assigned to machines after the release time (R_j). Each operation is processed on the target machine (M_{js}) after the start time (S_{js}). To make it simple, the processing time is set as 1, 2 and 3 for the three stages respectively. The average power consumption Q_{jsm} is defined as 1 for any operation on any machine while the value of the power's peak Q_{max} is equal to 3. Finally, we assign a priority to the total tardiness over the makespan in the objective function by setting the WT as 100. Figure 31 shows the Gantt chart of this scheduling. Regarding new arrival jobs, job 6 and job 7 need to be considered after starting the plan. In the traditional static environment, they could only be scheduled after completing operations of the original schedule at each stage as illustrated in Figure 32. However, the periodic complete rescheduling approach in a dynamic environment reschedules new arrival jobs at the beginning of the rescheduling point (RS=7) with remaining operations of original jobs simultaneously as in Figure 33.

Table 15 An example of EDFFS

	Original jobs						New arrival jobs	
	job 0	job 1	job 2	job 3	job 4	job 5	job 6	job 7
R_j	0.80	1.42	3.54	3.77	4.91	2.45	7.77	7.49
M_{j0}, M_{j1}, M_{j2}	1, 1, 0	0, 0, 1	0, 1, 1	0, 1, 0	1, 1, 1	0, 0, 0		
S_{j0}, S_{j1}, S_{j2}	0.80,1.80, 3.80	1.42, 2.42, 4.42	6.80, 9.91, 11.91	3.77, 7.91, 12.42	4.91, 5.91, 7.91	2.45, 7.42, 9.42		

The following matrices show the EDFFSP decoding result for the example. Each row of these matrices represents a job and each column represents a stage. A chromosome consists of the target machine matrix X(k) and the priority matrix Y(k) generated randomly to obtain the order matrix Z(k).

$$X(k)=\begin{bmatrix} -1 & -1 & -1 \\ -1 & -1 & -1 \\ -1 & 1 & 0 \\ -1 & 1 & 1 \\ -1 & -1 & 0 \\ -1 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 0 \end{bmatrix},\ Y(k)=\begin{bmatrix} -1 & -1 & -1 \\ -1 & -1 & -1 \\ -1 & 4 & 2 \\ -1 & 10 & 8 \\ -1 & -1 & 12 \\ -1 & 9 & 3 \\ 6 & 13 & 11 \\ 1 & 5 & 7 \end{bmatrix} \rightarrow Z(k)=\begin{bmatrix} C & C & C \\ C & C & 0 \\ 0 & 8 & 10 \\ C & 2 & 4 \\ C & 0 & 1 \\ C & 3 & 9 \\ 5 & 6 & 7 \\ 11 & 12 & 13 \end{bmatrix}$$

Following the description, the later assigned operation needs be delayed when the power's peak is met. For instance, job 7 at stage 0 was supposed to be processed after the completion of job 2 at stage 0 on machine 0 as in Figure 33. Moreover, at the same moment machine 2, 3 and 4 are busy with job 5 at stage 1, job 3 at stage 1 and job 4 at stage 2 respectively. But due to the power limitation, this scenario is not possible. As $z_{70}(k)$ is equal to 11, $z_{51}(k)$ to 3, $z_{31}(k)$ to 2, $z_{42}(k)$ to 1, job 7 at stage 0 is the newest allocated one among all of them. Thus, it is delayed until the completion of job 5 at stage 2 on machine 4.

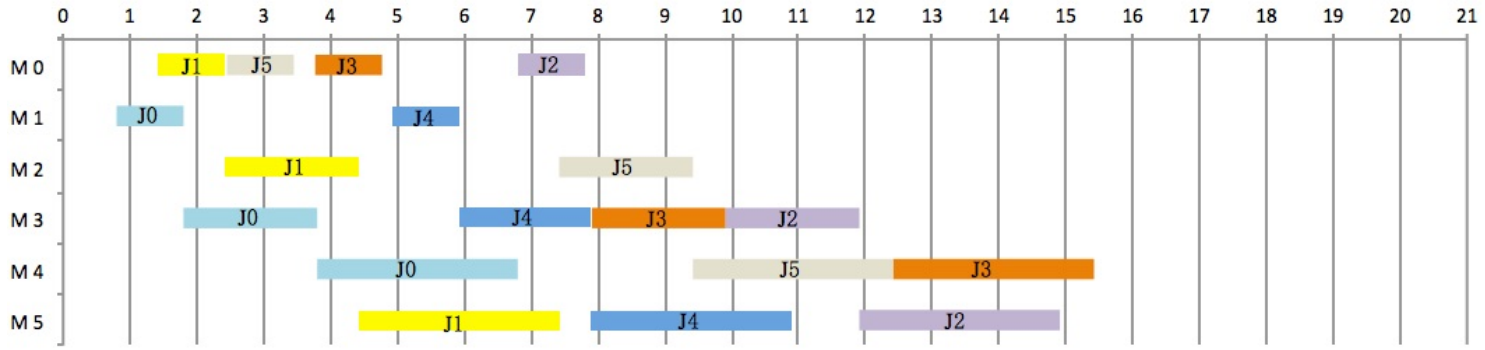

Figure 31 The original schedule of an optimized solution

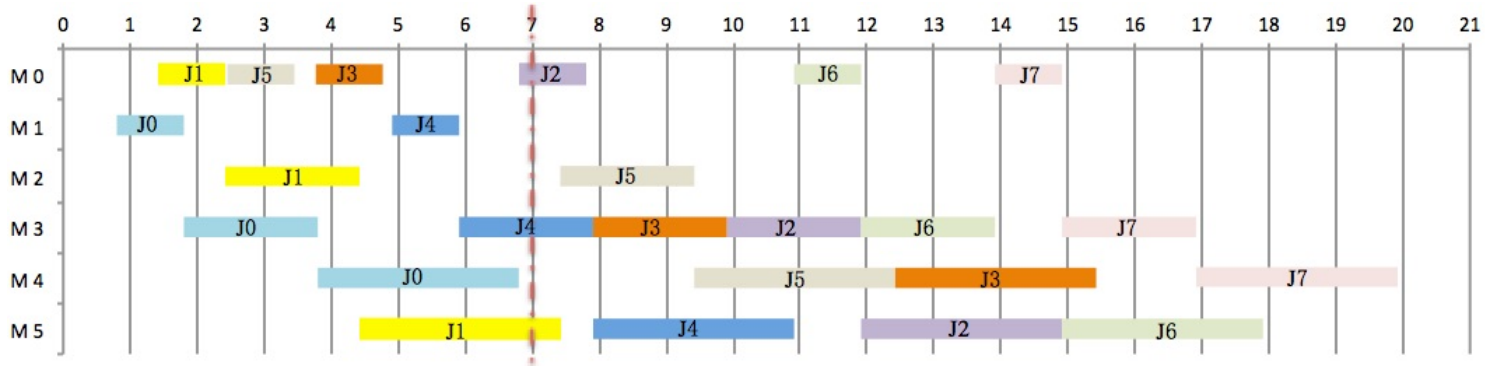

Figure 32 The updated schedule of an optimized solution in a static environment

($\sum_{j\in JUJ'} T_j = 1.64,\ C_{max} = 19.91$, Value of the Objective Function=183.91)

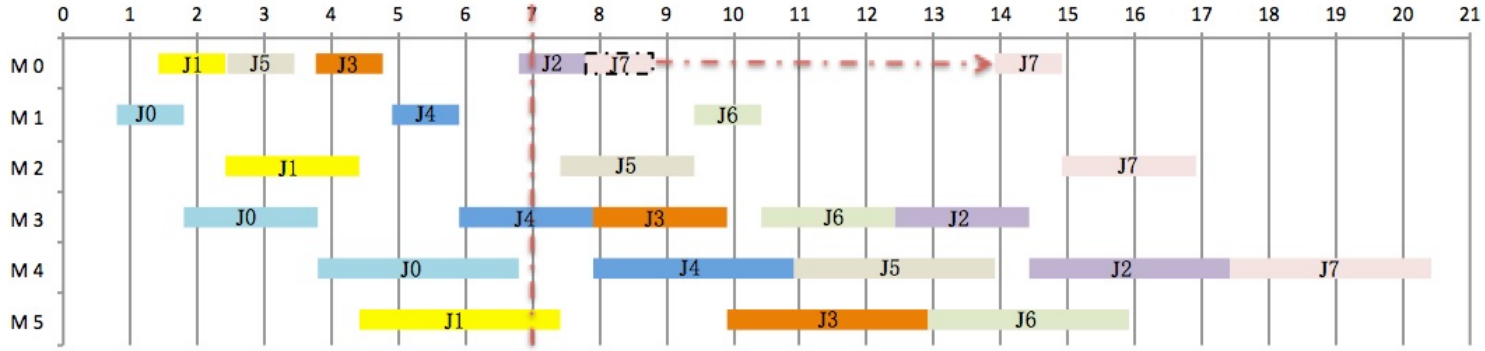

Figure 33 The updated schedule of an optimized solution obtained by the proposed approach in a dynamic environment

($\sum_{j\in JUJ'} T_j = 0.96, C_{max} = 20.42$, Value of the Objective Function=116.42)

IV.3.3 CUDA-Based Hybrid Genetic Algorithm

As the complete rescheduling strategy is rarely achievable in practice due to the prohibitive computation time [12], the parallel GA I is used in this Chapter for the

EDFFSP. This design is highly consistent with the CUDA framework in order to get the maximum speedup without compromising to solutions' quality. The main structure keeps the same as in Chapter III.3 while some modifications are made to suit the priority-based encoding and to meet the additional requirements of EDFFSP.

- The fitness function: Since the EDFFSP is also a minimization problem, the fitness function FIT(ρ) of an individual ρ is transferred from the objective function as

$$\mathrm{FIT}(\rho) = \max\left(\mathrm{E_{max}} - (\mathrm{WT} * \sum_{j \in J} \mathrm{T_j} + \mathrm{C_{max}}), 0\right) \quad (4.14)$$

  where $\mathrm{E_{max}}$ is the estimated maximum value of the objective function.

- Selection: Because the texture memory allows a CUDA thread likely to read from an address "near" the address that nearby threads [13] and the short response time is required by the EDFFSP, we modify the selection area of parallel GA I from Figure 15 to Figure 24. Thus, the memory management is similar with the cellular GA where the neighbors' information are stored on the texture memory and a tournament selection is implemented via the global memory. Finally, the individual with the largest fitness value is the winner of each tournament and is selected to replace the considered individual.

- Crossover: Individuals are still paired with neighbors as in Figure 16 while a 2D single point crossover is executed for the target machine matrix and the priority matrix respectively if a specified probability is satisfied. As the randomly generated values in the priority matrix is unique, a correction step is required to replace the duplicate values by the missing values in ascending order. All steps are executed through the global memory and an example shows the procedure in Figure 34.

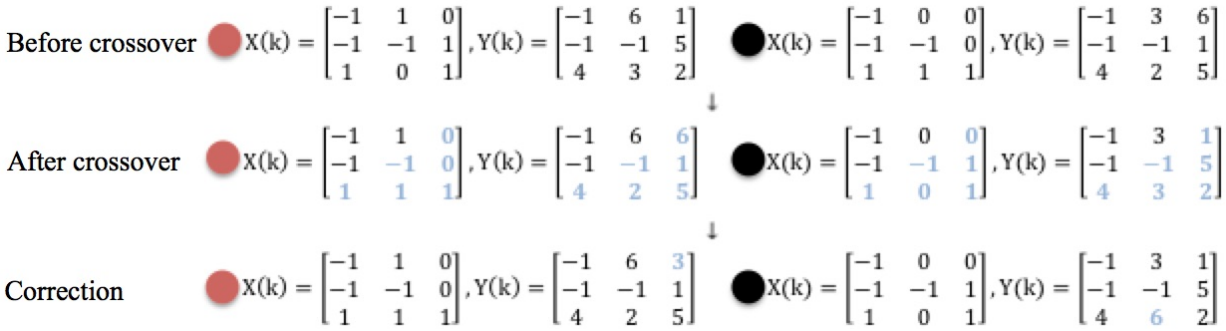

Figure 34 An example of the neighboring paired crossover

- Mutation: When the mutation is executed using the global memory, the non-negative elements of the target machine matrix of this individual are replaced by random values in the range, apart from the original ones. Regarding the priority matrix, two non-negative elements are chosen randomly to exchange the values. An example is given in Figure 35.

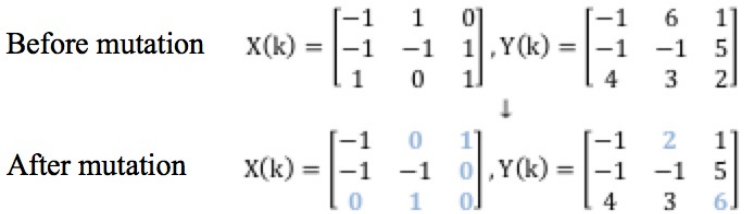

Figure 35 An example of the mutation

- Replacement: No modification.
- Migration: No modification.

IV. 4 Numerical Experiments

In order to assess the implementation of parallel GA I to the EDFFSP, 4 tests are conducted. Test 1 and test 2 are performed in terms of an energy efficient FFS without considering new arrival jobs. Test 1 configures the parameters of the proposed hybrid GA, while test 2 shows its efficiency and effectiveness compared to the simple GA [14], the cellular GA [15] and the OpenMP based master-slave GA. New arrival jobs are included in test 3 to evaluate the performance of the EDFFSP. A small size instance is considered in those 3 tests. There are 10 original jobs with 3 production stages. Each stage includes 2 parallel machines. The power's peak is imposed through a bound equal to 4. Test 4 examines the convergence trend of EDFFSP with 3 different size problems.

The instances are characterized by the different numbers of jobs (n = 10, 50, 80) with the different numbers of stages (g = 3, 4, 4), the different numbers of machines (o_s= 2, 2, 3) in each stage and the different numbers of power's peak (Q_{max} =4, 5, 10). The rescheduling point is randomly generated in test 3 and test 4. The number of new arrival jobs is decided by the ratio of the rescheduling point to the makespan in the original schedule times the amount of original jobs. This is designed to keep the total amount of jobs waiting to be scheduled roughly consistent. Other experimental relative data are defined in Table 16 and there is no update for the experimental platform.

Table 16 The experimental relative data of the EDFFS

WT	100
P_{jsm}	U[1, 5], where $P_{0sm}= P_{1sm}= \cdots = P_{(n+n'-1)sm}$
R_j	U[0, $\overline{P}$], where $\overline{P} = \sum_s(\sum_m P_{jsm}/o_s)$
D_j	$R_j+ \overline{P}(1 + \sigma)$, where σ=U[0,2]
Q_{jsm}	1

IV.4.1 Parameters Configuration Test of Parallel GA I

As the maximum threads amount per block on the CUDA framework is 1024 and they are organized in a grid, the maximum island size for the hybrid GA is 1024 (32×32). In order to have more than one island in all cases, the population size is kept as 4096 (64×64). Since small size islands with the migration lead to premature convergence while the algorithm with large size islands converges slower [16], we set there are 64 (8×8) individuals in one island. Considering the existing experiences, the most appropriate crossover rate ranges between 0.75 and 0.9 [17] and the mutation rate should be much lower than the crossover rate [18]. Therefore, the values of crossover rate and mutation rate are given as 0.9 and 0.1 respectively.

In order to ensure the performance of our GA parameters, we applied the parallel hybrid GA on the tested instance with three groups of crossover rates and three groups of mutation rates as in Table 17. According to the average results of 100 iterations, we could find the crossover rate and the mutation rate do have some influence on the algorithm performance. Moreover, when crossover rate=0.9 and mutation rate=0.1, the parallel hybrid GA could obtain satisfying results for solutions' quality and execution

time. To achieve the fairness of comparison, we set the crossover rate and the mutation rate as 0.9 and 0.1 for all GAs in the following tests of this Chapter.

Table 17 Results of the parallel hybrid GA on GPUs with different settings of crossover rate and mutation rate (Generations =100)

Crossover Rate	Mutation Rate	Solution Quality	Execution Time (s)
0.75	0.05	216.39	8.21
0.75	0.1	219.98	8.34
0.75	0.15	211.70	8.47
0.825	0.05	220.39	8.30
0.825	0.1	214.03	8.43
0.825	0.15	210.90	8.53
0.9	0.05	216.56	8.36
0.9	0.1	209.81	8.50
0.9	0.15	215.09	8.58

Due to the influence from the island size, the trend of the probability obtaining adequate solutions and the execution time with different island sizes is illustrated in Figure 36 and Table 18 respectively. Each value denotes an average result over 100 runs. Regarding the values of the objective function got by different settings of crossover rate and mutation rate are approaching 200, we set the adequate solution level as 200 for the tested instance. When a value of the objective function is less than 200 after the specified generations, it is considered as an adequate solution. From Figure 36 and Table 18, we could observe a great influence from the island size on the solutions' quality of the hybrid parallel GA on GPUs but a few difference on the execution time. The islands with 64 individuals (8×8 threads) perform best. In terms of the 2D population size 4096 (64×64), there are 64 islands (8×8 blocks).

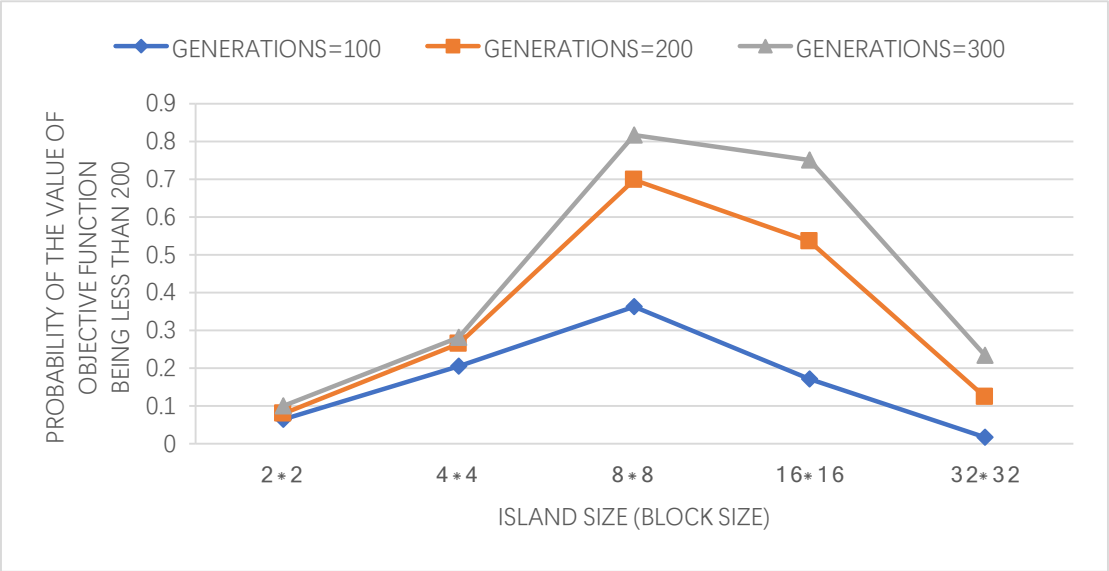

Figure 36 The trend of the probability obtaining adequate solutions with different island sizes (block sizes) on GPUs

Table 18 Execution time with different island sizes (block sizes) on GPUs (s)

Generations \ Island Size	4 (2×2)	16 (4×4)	64 (8×8)	256 (16×16)	1024 (32×32)
100	7.65	7.71	9.11	9.14	12.30

IV.4.2 Performance Evaluation Test of Parallel GA I

Firstly, we try to compare the solutions obtained from the parallel GA I, the simple GA and the cellular GA. The designs of simple GA and cellular GA keep the same as in Chapter III. For fair comparison, a master-slave GA on multi-core CPU with or without vectorization is also taken into consideration. The master-slave model exploits parallelism in the simple GA by distributing the most time consuming part, fitness function evaluation, to slaves. As it does not affect the behavior of the algorithm, the master-slave GA is only included for the execution time comparison. Furthermore, we run the hybrid parallel GA and the cellular GA on GPUs, the simple GA on single core CPU, the master-slave GA on four cores CPU. Each of them is generated 100 times respectively.

Table 19 Solutions' quality comparison

Generations	Hybrid Parallel GA			Simple GA			Cellular GA		
	Avg.	Best	Variance	Avg.	Best	Variance	Avg.	Best	Variance
100	209.81	153.45	152.22	410.72	236.55	5208.84	258.39	158.86	1635.95
200	183.16	151.67	149.47	354.64	214.31	3834.04	228.65	155.26	1549.97
300	181.80	151.67	150.01	339.09	198.69	3565.65	221.51	154.51	1073.24
400	178.32	149.83	151.67	331.57	170.60	4010.57	217.42	153.24	1322.16
500	177.93	149.47	150.63	327.46	156.41	4779.69	216.99	151.74	1073.99

From the results in Table 19, we discover that the parallel GA I always gains a better performance for solving the energy efficient FFS than the simple GA and the cellular GA with the average value, the best value and the variance of the objective function. As a result, the efficiency of parallel GA I gets confirmed by dealing with different shop scheduling problems. Moreover, the cellular GA overcomes the simple GA as it allows a better exploration of the search space with respect to the decentralized population [15].

Table 20 Execution time comparison (Generations=100)

Population size	Hybrid Parallel GA on GPUs	Cellular GA on GPUs	Simple GA on single core CPU	Master-Slave GA on 4 cores CPU	
				without vectorization	with vectorization
64×64	8.77 s	8.14 s	129.16 s	39.50 s	5.60 s
128×128	30.71 s	31.13 s	554.01 s	182.27 s	33.07 s
256×256	105.73 s	108.07 s	2651.61 s	1127.78 s	298.96 s

Since the hybrid parallel GA and the cellular GA are designed specially for 2D grid architectures, they could maximize the benefits from the CUDA framework and almost take the same execution time when dealing with different population sizes as illustrated in Table 20. On the opposite, the simple GA on single core CPU takes from 14.73 to 25.08 times the execution time of the hybrid parallel GA when the population size is increased from 64×64 to 256×256. As far as the available experiment platform, we firstly parallelized the master-slave GA using OpenMP [20] on 4 cores CPU. Afterwards, the SIMD vectorization was executed simultaneously via SSE2 [20]. The code was compiled by the command as follows and the vectorization report showed that all loops for the fitness function evaluation were well vectorized.

gcc -fopenmp -O3 -ftree-vectorize -msse2 mycode.c -ftree-vectorizer-verbose=1 -o mycode.o

With the development of multi-cores CPU and SIMD vectorization, the performance of master-slave GA has been improved a lot by distributing the fitness function evaluation to slaves and executing them concurrently. It even overcomes the GAs on GPUs with small population size. However, the GAs working on GPUs always win with less execution time when the amount of individuals is increased, due to the limited amount of cores and the limited SIMD width in our case.

IV.4.3 Sensitive Analysis Test of the EDFFSP

As the number of new arrival jobs is decided by the ratio of the RS to the makespan in the original schedule times the amount of original jobs, we change the amount of new arrival jobs by varying the ratio of the RS to the makespan in the original schedule. The influence with different ratios to the periodic complete rescheduling approach and the traditional static approach are displayed in Table 21. The iteration number is kept as 100 like the last two tests. The periodic complete rescheduling approach is more flexible in a dynamic environment as it reschedules the new arrival jobs at the beginning of the rescheduling point. However, those jobs could only be scheduled after completing operations of the original schedule at each stage by the traditional static approach. This impact is even more evident when the ratio of the RS to the makespan in the original schedule is small. And it is decreasing and almost disappears when the RS takes place near the end of the original schedule. Therefore, we strongly suggest using the periodic complete rescheduling approach with the assistance of parallel GA I when the RS is arranged at the first half part of the original schedule. Meanwhile, the traditional static approach may have similar performance if the RS is considered at the later half part.

Table 21 Comparison between the periodic complete rescheduling approach and the traditional static approach with different ratios of the RS to the makespan in the original schedule (Generations=100)

Ratio of the RS to the makespan in the original schedule	Traditional static approach	Periodic complete rescheduling approach	Improvement Ratio
20%	4108.41	2142.90	1.9172
40%	11131.51	9209.40	1.2087
60%	17892.24	16941.56	1.0561
80%	26595.63	26520.96	1.0028

As tardy jobs typically cause penalty costs [21] and have a great influence on customers' satisfaction, the weight WT indicates the priority of the total tardiness in the objective function. However, we consider the relationship between two objectives with different WT settings due to the importance of makespan in manufacturing practice and Table 22 shows the average results of 100 iterations. According to the values of total tardiness and makespan, we could find the makespan is less sensitive to the weight WT than the total tardiness as the variance of makespan is 0.61 while the variance of total tardiness is 78.17. Moreover, once the value of WT is increased to reach a very large constant, the total tardiness is approaching its minimum value. Thus, manufacturers should take the chance to optimize the value of total tardiness while limiting the makespan in a reasonable range.

Table 22 Relationship between two objectives with different WT settings (Generations=100)

WT	Total Tardiness	Makespan	Objective Function Value
0.0001	39.48	40.55	40.56
0.001	39.95	40.55	40.59
0.01	35.23	40.54	40.89
0.1	23.43	40.78	43.12
0.4	19.04	41.14	48.76
0.7	18.61	41.23	54.26
1	18.29	41.44	59.73
4	17.83	42.15	113.46
7	17.69	42.18	166.00
10	17.57	42.12	217.86
100	17.58	42.39	1800.43
1000	17.60	42.51	17645.71
10000	17.58	42.41	175831.12
Variance	78.17	0.61	

IV.4.4 Convergence trend test of the EDFFSP

As a GA converges when most of the population is identical or the diversity is minimal [22], there is no need to execute the algorithm for more generations after the convergence point. For the EDFFSP, it is important to identify the convergence point and its corresponding execution time for different size problems. Three different size problems are considered in this test. The convergence trends of the small size, the medium size and the large size problem instances are described in Figure 37, Figure 38 and Figure 39 separately. Each point in figures displays a value of 30 runs.

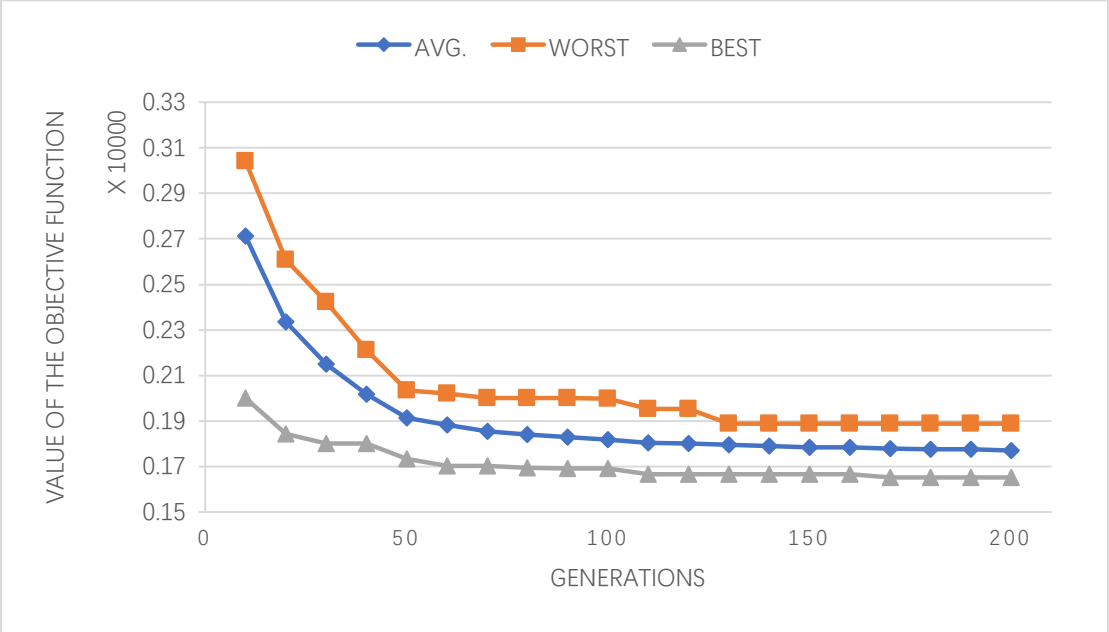

Figure 37 The convergence trend of small size problem

Figure 38 The convergence trend of medium size problem

Figure 39 The convergence trend of large size problem

With regard to the small size problem, it converges approximately at the level of 50 generations, while the values for the medium size and the large size problems are around 400 and 600. As the complexity increases when we raise the size of the problem, the execution time per 10 generations for these problems is about 1.24s, 223.37s and 4256.14s respectively. Therefore, to get solutions after the convergence for the small size problem, it takes 6.2s whereas the medium size and the large size problems need much longer time as 8934.8s and 255368.4s. Due to the dramatically increasing execution time for large-scale problems, the hybrid parallel GA may get a feasible solution before achieving the convergence based on decision-makers' consideration, namely a trade-off between the solutions' quality and the time consumption.

IV.5 Conclusion

In this Chapter, we have first studied an energy efficient dynamic flexible flow shop scheduling model using the peak power value with consideration of new arrival jobs. To solve this NP-hard problem in a short response time, a priority based hybrid parallel GA with a periodic complete rescheduling approach was developed. In the first test, we

configured the parameters of the hybrid parallel GA and obtained a reasonable island size for the tested instances to inhibit the premature convergence with a faster convergence speed. Afterwards, the designed GA in test 2 showed that it could gain better results than the simple GA, the cellular GA through the combination of merits from two levels. Meanwhile, it reduced the time requirements dramatically by optimizing the benefits from the CUDA framework. As seen in test 3, the periodic complete rescheduling approach was flexible to solve the EDFFSP, particularly when the rescheduling point was considered at the first half part of the original schedule. Moreover, the total tardiness was more sensitive in this two objectives optimization problem and its value was approaching the minimum once the weight WT was increased to a very large constant. Finally, test 4 demonstrated the response time to achieve the convergence point for large-scale EDFFSP. We suggested as well in this case decision-makers to obtain a feasible scheduling by making a trade-off between the solutions' quality and the time consumption.

Reference

[1].EIA (2009) International energy outlook 2009. May 2009 2.

[2].EIA (2010) Annual energy review 2009. Report no. DOE/EIA0384(2009); August 2010

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

[4].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.

[5].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.

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

[7].Ouelhadj, D., & Petrovic, S. (2009). A survey of dynamic scheduling in manufacturing systems. Journal of scheduling, 12(4), 417.

[8].Tang, D., Dai, M., Salido, M. A., & Giret, A. (2016). Energy-efficient dynamic scheduling for a flexible flow shop using an improved particle swarm optimization. Computers in Industry, 81, 82-95.

[9].Zhang, L., Li, X., Gao, L., & Zhang, G. (2016). Dynamic rescheduling in FMS that is simultaneously considering energy consumption and schedule efficiency. The International Journal of Advanced Manufacturing Technology, 87(5-8), 1387-1399.

[10].Melab, N., Chakroun, I., Mezmaz, M., & Tuyttens, D. (2012, September). A GPU-accelerated branch-and-bound algorithm for the flow-shop scheduling problem. In Cluster Computing (CLUSTER), 2012 IEEE International Conference on (pp. 10-17). IEEE.

[11].Czapiński, M., & Barnes, S. (2011). Tabu Search with two approaches to parallel flowshop evaluation on CUDA platform. Journal of Parallel and Distributed Computing, 71(6), 802-811.

[12].Gholami, M., Zandieh, M., & Alem-Tabriz, A. (2009). Scheduling hybrid flow shop with sequence-dependent setup times and machines with random breakdowns. The International Journal of Advanced Manufacturing Technology, 42(1-2), 189-201.

[13].Sanders, J., & Kandrot, E. (2010). CUDA by example: an introduction to general-purpose GPU programming. Addison-Wesley Professional.

[14]. Zhong, J., Hu, X., Zhang, J., & Gu, M. (2005, November). Comparison of performance between different selection strategies on simple genetic algorithms. In Computational Intelligence for Modelling, Control and Automation, 2005 and International Conference on Intelligent Agents, Web Technologies and Internet Commerce, International Conference on (Vol. 2, pp. 1115-1121). IEEE.

[15] Alba, E., & Dorronsoro, B. (2009). Cellular genetic algorithms (Vol. 42). Springer Science & Business Media.

[16]. Cantú-Paz, E. (1998). A survey of parallel genetic algorithms. Calculateurs paralleles, reseaux et systems repartis, 10(2), 141-171.

[17]. Schaffer, J. D. A. R. (1989). A study of control parameters affecting online performance of genetic algorithms for function optimization. San Meteo, California.

[18]. Cabrera, J. A., Simon, A., & Prado, M. (2002). Optimal synthesis of mechanisms with genetic algorithms. Mechanism and Machine theory, 37(10), 1165-1177.
[19]. http://www.openmp.org/
[20]. https://en.wikipedia.org/wiki/Streaming_SIMD_Extensions
[21]. Parmee, I. C. (2009). Adaptive Computing in Design and Manufacture.
[22]. Louis, S. J., & Rawlins, G. J. (1993). Predicting convergence time for genetic algorithms. Foundations of Genetic Algorithms, 2, 141-161.

Chapter V. Parallel GA II with Event-Driven Schedule Repair for Solving a JSP with Minimization of Total Tardiness and Total Energy Consumption

V.1 Introduction

Energy costs due to production have been traditionally treated as externalities that must be incurred and that cannot be reduced by production planning and scheduling [1]. With an increasing interest in industrial sustainability, integrating energy efficiency into production efficiency is concerned as one essential factor in factory practice. In the literature, there are two kinds of approaches studying energy saving in manufacturing systems [2]: avoiding peak power consumption and reducing the overall energy consumption. The first one shifts load at energy peaks when the maximum available energy is limited. The efficiency and the effectiveness of this method have been discussed in Chapter IV. However, 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]. The second one aims at reducing the total energy consumption of the manufacturing system by subdividing it and switching among the different types and levels of energy consumption. Liu et al. [3] developed a model for the bi-objective problem that minimized the total electricity consumption and the total weighted tardiness, where a non-dominant sorting genetic algorithm was used to obtain the Pareto front. Similarly, an emission-aware multi-machine job shop scheduling model was addressed in [4] and was solved through a modified multi-

objective genetic algorithm. Dai et al. [5] reported an energy efficient model for the flexible flow shop scheduling problem and utilized a genetic-simulated annealing algorithm to make a significant tradeoff between the makespan and the total energy consumption. In one word, numerous efforts have been given to combine the traditional shop scheduling efficiency with the overall energy consumption. However, the models used in these researches are deterministic in which the number of jobs is a fixed value [3]. As ongoing reactive process where the presence of a variety of unexpected disruptions is usually inevitable [6], the static scheduling obviously cannot meet the requirements in most real-world environments.

The inevitable unpredictable new arrival jobs may lead changes in the original schedule. Literature on dynamic scheduling has considered a significant number of works dealing with new arrival jobs and their effects in various manufacturing systems [6]. Nevertheless, limited researches along this domain focus on dynamic energy aware shop scheduling problems and they were generally solved by the predictive reactive approach with complete rescheduling [7, 8]. Complete rescheduling and schedule repair are the most two common used strategies in the dynamic environment. As in Chapter IV, complete rescheduling provides the optimal solutions. But it can result in instability and disruption in manufacturing flows, leading to tremendous production costs [9]. Schedule repair only attempts to revise part of the originally created schedule for responding to the production environment changes. Pach et al. [2] set up a potential fields based reactive scheduling approach for flexible manufacturing systems in which resources were able to switch to the standby mode to avoid useless energy consumption and to emit fields to attract products. Zeng et al. [10] presented the particle swarm optimization algorithm to solve the dynamic scheduling problem of multi-task for hybrid flow-shop with the objective of minimizing energy consumption by introducing idle time windows of machines. To sum up, a few works have tried to solve dynamic energy aware shop scheduling problems with schedule repair. Even schedule repair does not require prohibitive computation as complete rescheduling and has potential saving in CPU times [11], to obtain a renewed adequate scheduling plan within a short response time is still highly desired in the dynamic environment, especially for large scale or complex problems.

The research on parallel GAs is dominated by island GAs [12]. As the discussion in Chapter II, the main efforts pay attention to the homogeneous islands even the same genetic operator configurations and the migration mechanism may lead to premature convergence [13]. Moreover, with the huge evolution of multi-core CPUs and GPUs, some works have considered the cooperation between them to maximally utilize their compute capability. Dabah et al. [14] proposed five parallel approaches to accelerate the branch and bound algorithm for solving the blocking job shop scheduling problem and two of them represented a hybridization between the multi-core CPU approach and the GPUs-based parallelization approach. Benner et al [15] discussed a hybrid Lyapunov solver based on the matrix sign function where the intensive parts of the computation were accelerated using GPUs while executing the remaining operations on a multi-core CPU. In [16], Bilel et al. introduced a CPU-GPU co-simulation framework where synchronization and experiment design were performed on CPU and node's processes are executed in parallel on GPU according to the master slave model. These cases have confirmed the efficiency to design a scheme that exploits the different hardware architectures simultaneously. However, this strategy is not yet implemented for island GAs, particular for heterogeneous island GAs to solve dynamic energy aware shop scheduling problems, as far our knowledge is concerned.

Considering the above-mentioned requirements, an investigation into minimizing total tardiness and total energy consumption in the job shop with new urgent arrival jobs is concerned in this Chapter. Afterwards, an adequate renewed scheduling plan is provided in a short response by the dual heterogeneous GA executed simutaniously on different parallel platforms with the event driven schedule repair approach. Finally, the efficiency and the effectiveness to implement the proposed method for solving dynamic energy aware shop scheduling problems are validated through computational tests.

V.2 Problem Definition

The Job Shop scheduling Problem (JSP) is a NP-hard problem [17] 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 [3]. As a layout shown in Figure 40, 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.

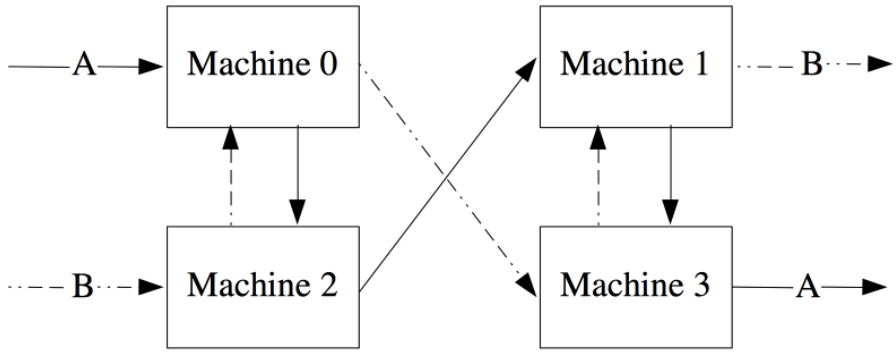

Figure 40 A job shop layout

The Energy efficient Dynamic Job Shop scheduling Problem (EDJSP) is an extension of the JSP with machine speed scaling [18] in which machines are available to be set at different speed levels when dealing with different jobs. The processing time and the energy consumption 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 consumption increase. Not like the EDFFSP presented in Chapter IV, a set of new urgent jobs may arrive after the start time of original schedule and these jobs need to be processed as soon as possible. Therefore, the production line should conduct them immediately. The operations being processed are terminated and need be rescheduled with the remaining uncompleted operations of original jobs based on the insertion of new urgent arrival jobs. The updated schedule refers some local adjustment of the original one for the stability of manufacturing system. There are two possible measures for the impact caused by the schedule changes [19]: (1) the deviation from the original jobs starting times, (2) the deviation from the original sequence. In this Chapter, a measure modified from the first one is taken into consideration where each 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 consumption is required to make the original job be completed as in the original schedule. Clearly, there is a conflict among the minimization of total tardiness, the

minimization of total energy consumption and the minimization of disruption to the original schedule as displayed in Figure 41. Thus, a trade-off must be made among of them. Because of the NP hardness of the JSP, the EDJSP is a NP-hard combinatorial optimization problem and more complex than the JSP.

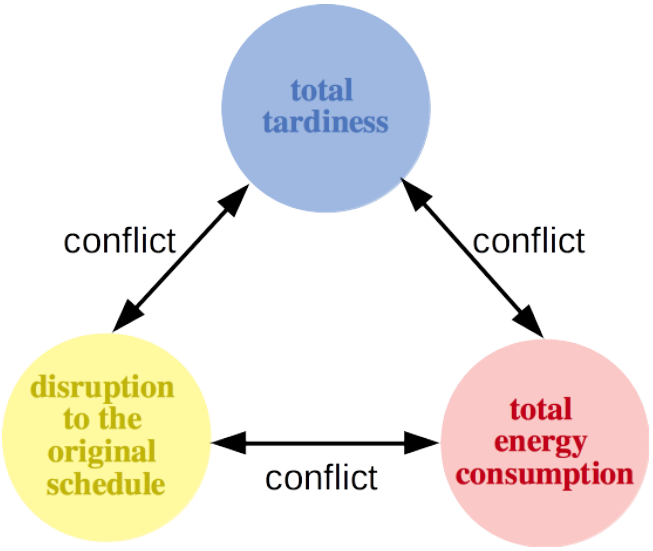

Figure 41 Conflicts among total tardiness, total energy consumption and disruption to the original schedule

For an easy presentation, we summarize the notations used along the rest of this Chapter in Table 23.

Table 23 A description of notations used in Chapter V

Notation	Description
j, i, l, x	Job indices
s, t, y	Operation indices
m	Machine index
p, q, w	Speed level indices
n	Number of original jobs
n′	Number of new arrival jobs
n″	Number of uncompleted original jobs at the rescheduling point
r1	Number 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	Number of operations of job j
g	Number of machines
h	Number of speed levels
J	Set of original jobs, $J = \{0,1,2,\dots,n-1\}$
J'	Set of new arrival jobs, $J' = \{0,1,2,\dots,n'-1\}$
J''	Set of uncompleted original jobs at the rescheduling point, $J'' = \{0,1,2,\dots,n''-1\}$
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, $j \in J \cup J'$
M_{js}	Target machine handling operation s of job j, $j \in J \cup J', s \in O_j$
S_{jsm}	Original start time of operation s of job j on machine m, $j \in J \cup J', s \in O_j$, $m \in M$
S'_{jsm}	New start time of operation s of job j on machine m, $j \in J \cup J', s \in O_j$, $m \in M$
T_j	Tardiness of job j, $j \in J \cup J'$
RS	Rescheduling point
P_{jsmp}	Processing time when operation s of job j is to be processed on machine m at speed level p, $j \in J \cup J', s \in O_j, m \in M, p \in L$
Q_{jsmp}	Energy consumption when operation s of job j is to be processed on machine m at speed level p, $j \in J \cup J', s \in O_j, m \in M, p \in L$
Z_{jsmp}	Boolean variable, it is equal to 1 if operation s of job j is processed on machine m at speed level p, otherwise, it equals to 0, $j \in J \cup J', s \in O_j, m \in M, p \in L$
TT	Total tardiness of all jobs
ET_{max}	Estimated maximum value of TT
ET_{min}	Estimated minimum value of TT
TE	Total energy consumption

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.
a, b, c, f, z	Gene indices in a chromosome
v_j	Index of occurrence time of a job number
u_j	Occurrence time of a job number
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
Z(k)	Completed status 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
n_{js}	Number of operations on machine m before operation s of job j is assigned on it.
o'_m	Number of operations on machine m
O'_m	Set of operations on machine m, $O_m = \{0,1,2, \dots, o_m - 1\}$
ρ	Individual index
μ	Index used to generate odd or even indexed individuals
$\pi_{2\mu,a}$	the a th gene in $X'(k)$ from the 2μ th individual
$\pi_{2\mu-1,a}$	the a th gene in $X'(k)$ from the (2μ-1) th individual
$\pi'_{2\mu,a}$	the a th gene in $Y'(k)$ from the 2μ th individual
$\pi'_{2\mu-1,a}$	the a th gene in $Y'(k)$ from the (2μ-1) th individual
$\bar{f}$	The average fitness value of the population
f_{max}	The maximum fitness value of the population

f'	The larger fitness value of the two selected parents which are executed crossover
$\varphi_1,\ \varphi_2,\ \varphi_3,\ \varphi_4$	Modified coefficients for the crossover rate and the mutation rate

To minimize the total tardiness, the total energy consumption and the delay caused by the schedule changes, the formal mathematical model of the EDJPS is derived from the mathematical models presented in [19, 20]. The formalization is given as follows.

Objective Function:

$$\text{Min: } \alpha\times\frac{TT-ET_{min}}{ET_{max}-ET_{min}}+\beta\times\frac{TE-EE_{min}}{EE_{max}-EE_{min}}+\gamma\times\frac{DEV-ED_{min}}{ED_{max}-ED_{min}} \tag{5.1}$$

Constraints:

$$TT=\sum_{j\in J\cup J'}T_j=\sum_{j\in J\cup J'}\max\left(S_{j(o_j-1)M_{j(o_j-1)}}+P_{j(o_j-1)M_{j(o_j-1)}p}\times Z_{j(o_j-1)M_{j(o_j-1)}p}-D_j,0\right)\quad p\in L \tag{5.2}$$

$$S_{j0M_{j0}}\geq R_j\quad j\in J\cup J' \tag{5.3}$$

$$S_{j(s+1)M_{j(s+1)}}\geq S_{jsM_{js}}+P_{jsM_{js}p}\times Z_{jsM_{js}p}\quad j\in J\cup J', s\in O_j, s>0, p\in L \tag{5.4}$$

$$S_{jsM_{js}}+P_{jsM_{js}p}\times Z_{jsM_{js}p}\leq S_{itM_{it}}$$
$$j\in J\cup J', i\in J\cup J', j\neq i, s\in O_j, t\in O_i,\ M_{js}==M_{it},\ p\in L, S_{jsm}\leq S_{itm} \tag{5.5}$$

$$\sum_{p\in L}Z_{jsM_{js}p}=1\quad j\in J\cup J', s\in O_j \tag{5.6}$$

$$TE=\sum_{j\in J\cup J'}\sum_{s\in O_j}Q_{jsM_{js}p}\times Z_{jsM_{js}p}\quad p\in L \tag{5.7}$$

$$RS\leq S_{jsM_{js}}\quad j\in J\cup J'', s\in O_j \tag{5.8}$$

$$DEV=\sum_{j\in J}wt_j\times\max\left((S'_{j(o_j-1)M_{j(o_j-1)}}+P_{j(o_j-1)M_{j(o_j-1)}p}\times Z_{j(o_j-1)M_{j(o_j-1)}p}\right)$$
$$-\left(S_{j(o_j-1)M_{j(o_j-1)}}+P_{j(o_j-1)M_{j(o_j-1)}q}\times Z_{j(o_j-1)M_{j(o_j-1)}q}\right),0)\quad p\in L, q\in L \tag{5.9}$$

The decision variables in this mathematical model are S_{jsm} and Z_{jsml}. A weighted additive utility function with three normalized objectives is described as (5.1) where all objectives can be assessed on the same scale. Constraints (5.2) defines the tardiness of jobs. The precedence among operations due to the jobs' processing cycles is presented by constraints (5.3) and (5.4), while constraint (5.5) establishes the precedence caused by the sequencing on machines. As far as the energy consumption, constraint (5.6)

states each operation can only be processed on one machine with one fixed speed level whereas the total energy consumption is given by constraint (5.7). Finally, constraint (5.8) imposes the definition of rescheduling and constraint (5.9) indicates the weighted finishing time deviation of the updated schedule from the original one.

V.3 Solving approach

V.3.1 Event-driven schedule repair strategy

With the event-driven policy, rescheduling is triggered in response to an unexpected event that alters the current system status [6]. In the case of EDJSP, urgent jobs may arrive after the starting time of the original schedule and need be processed immediately and quickly. Operations that are being executed need be terminated and uncompleted operations of original jobs must be rearranged in order to leave the machines available to firstly handle these urgent jobs. Thus, new urgent arrival jobs are assigned to machines with the highest speed levels at the beginning when the rescheduling is triggered. Uncompleted operations of original jobs are considered at the next step according to the remaining spaces on machines. The parallel GA II is chosen to generate an adequate schedule for them with the schedule repair strategy in a limited time. The flow of the event-driven schedule repair process is summarized as in Figure 42.

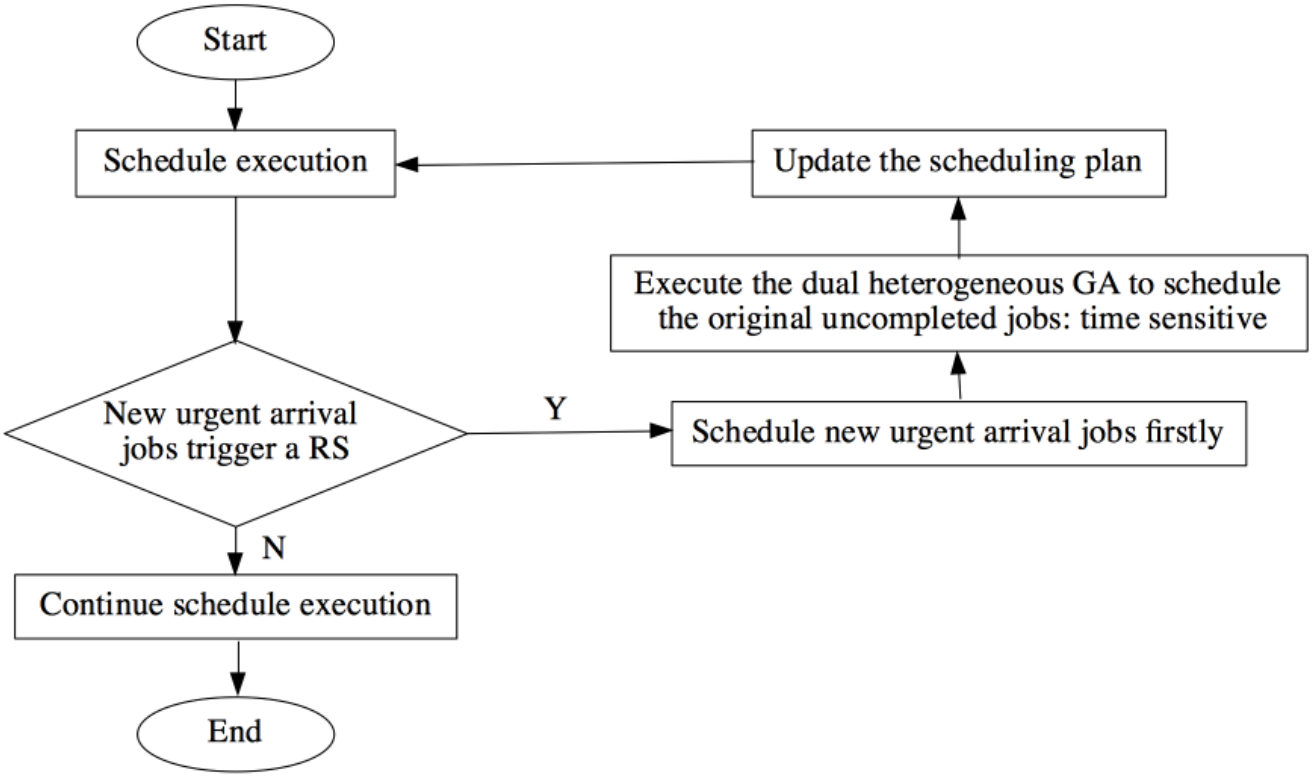

Figure 42 The flow of event-driven schedule repair process for the EDJSP

V.3.2 Hybrid Encoding Representation

In solving the EDJSP by the cellular GA, two modified operation-based encodings are adopted for representing the chromosomes. In terms of the schedule of original jobs and the schedule of new urgent arrival jobs, the chromosome contains two permutations: operation permutation X(k) (5.10) and speed level permutation Y(k) (5.11). 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 [21]. 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 [22]. According to the example provided in [23], 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 implies the operation 1 of job 2 as it is the 1st occurrence (after 0th occurrence) of number 2. Thus, X(k) can be translated to a list of ordered operations as $[o_{20}, o_{10}, o_{00}, o_{01}, o_{11}, o_{21}, o_{22}, o_{12}, o_{02}]$. On the other hand, each element $y_a(k)$ indicates the selected speed level of its related element

$x_a(k)$ on the target machine. Concerning new urgent arrival jobs, all the values in Y(k) are always kept as the highest speed level.

$$X[k]=[\, x_0(k),\ x_1(k),\ \cdots, x_a(k), \cdots, x_{(\sum_{j\in J} o_j - 1)}(k)\,] \tag{5.10}$$

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

$$Y[k]=[\, y_0(k),\ y_1(k),\ \cdots, y_a(k), \cdots, y_{(\sum_{j\in J} o_j - 1)}(k)\,] \tag{5.11}$$

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

To leave machines available to conduct new urgent arrival jobs firstly and rearrange uncompleted operations of original jobs, the chromosome of updated schedule also includes an operation permutation $X'(k)$ (5.12) and a speed level permutation $Y'(k)$ (5.13). The initialization rule for both are shown in Algorithm 3.

$$X'(k)=[\, x'_0(k),\ x'_1(k),\ \cdots, x'_a(k), \cdots, x'_{(\sum_{j\in J\cup J'} o_j - 1)}(k)\,] \tag{5.12}$$

where $x'_a(k) \epsilon [0, n + n' - 2], u_j == o_j$.

$$Y'(k)=[\, y'_0(k),\ y'_1(k),\ \cdots, y'_a(k), \cdots, y'_{(\sum_{j\in J\cup J'} o_j - 1)}(k)\,] \tag{5.13}$$

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

Regarding the pseudo GA, the complementary initialization strategy [24] is complemented by negating all alleles in a binary chromosome. To implement it with the EDJSP, even-indexed individuals are initialized by the modified operation-based encoding. On the other hand, genes in the range $[0, r2-1]$ of odd-indexed individuals keep same values with its paired parent while values of genes in the range $[r2, \sum_{j\in J\cup J'} o_j - 1]$ of $X'(k)$ is initialized with

$$\pi_{2\mu,a} = (n\text{-}1) - \pi_{2\mu-1,a} \tag{5.14}$$

and values of genes in the range $[r2, \sum_{j\in J\cup J'} o_j - 1]$ of $Y'(k)$ is initialized with

$$\pi'_{2\mu,a} = (h\text{-}1) - \pi'_{2\mu-1,a} \tag{5.15}$$

In this case, the $X'(k)$ of the $(2\mu - 1)$ th individual may be infeasible as the there is a risk that $u_j \neq o_j$. Therefore, an inspection step is carried out to replace the latest redundant values by the missing values in the ascending order. Moreover, the decoding rule is displayed in Algorithm 4.

Algorithm 3 The initialization rule of permutations $X'(k)$ and $Y'(k)$ for all individuals of the cellular GA

$a \leftarrow 0$;

for $b \leftarrow 0$ **to** $\sum_{j \in J} o_j - 1$ **do**

  $j \leftarrow x_b(0)$;

  $s \leftarrow v_j$;

  $p \leftarrow y_b(0)$;

  if $S_{jsM_{js}} + P_{jsM_{js}p} \times Z_{jsM_{js}p} > RS$ **then**

    $x'_a(0) \leftarrow x_b(0)$;

    $y'_a(0) \leftarrow y_b(0)$;

    $a \leftarrow a+1$;

  end if

end for

$r1 \leftarrow a$;

for $c \leftarrow 0$ **to** $\sum_{i \in J'} o_i - 1$ **do**

  $x'_a(0) = x_c(0)$;

  $y'_a(0) = y_c(0) =$ highest speed level;

  $a \leftarrow a+1$;

end for

$r2 \leftarrow a$;

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 4 The decoding rule of the EDJSP

for $a \leftarrow r2$ **to** $\sum_{j \in J \cup J'} o_j - 1$ **do**
  $j = x'_a(k)$;
  $s = v_j$;
  if $s == 0$ **then**
    if $R_j > RS$ **then**
      $S_{js} = R_j$;
    else
      $S_{js} = RS$;
    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**
        $w = y'_b(k)$;
        break;
      end if
    end for
    if $S_{j(s-1)M_{j(s-1)}} + P_{j(s-1)M_{j(s-1)}w} \times Z_{j(s-1)M_{j(s-1)}w} > RS$ **then**
      $S_{js} = S_{j(s-1)M_{j(s-1)}} + P_{j(s-1)M_{j(s-1)}w} \times Z_{j(s-1)M_{j(s-1)}w}$;
    else
      $S_{js} = RS$
    end if
  end if
  d← 0;
  for $c \leftarrow r1$ **to** a **do**
    $l \leftarrow x'_c(k)$;
    $t \leftarrow v_l$;
    if $M_{js} == M_{lt}$ **then**
      $O'_{M_{js}}[d] \leftarrow o_{lt}$;
      $d \leftarrow d+1$;
    end if
  end for

$n_{js} \leftarrow d$;
Sort elements in $O'_{M_{js}}[d]$ in ascending order by the starting time;
for $e \leftarrow 0$ **to** n_{js} **do**
x = job number in $O'_{M_{js}}[e]$;
y = operation number of job f in 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**
$q = y'_f(k)$;
break;
end if
end for
$p = y'_a(k)$;
if $[S_{js},\ S_{jsM_{js}} + P_{jsM_{js}p} \times Z_{jsM_{js}p}) \cap [S_{xy},\ S_{xyM_{xy}} + P_{xyM_{xyq}} \times Z_{xyM_{xyq}}) \neq \emptyset$ **then**
$S_{js} = S_{xy} + P_{xyM_{xyq}} \times Z_{xyM_{xyq}}$;
end if
end for
end for

V.3.3 Dual Heterogeneous Island GA on GPUs and multi-core CPU

- The fitness function: As an extension of the classic JSP, the EDJPS is also a minimization problem. Thus, we transfer the objective function from (Eq. (5.1)) to get the fitness function FIT(ρ) of an individual ρ as

$$FIT(\rho) = \max\left(E_{max} - \left(\alpha \times \frac{TT - ET_{min}}{ET_{max} - ET_{min}} + \beta \times \frac{TE - EE_{min}}{EE_{max} - EE_{min}} + \gamma \times \frac{DEV - ED_{min}}{ED_{max} - ED_{min}}\right), 0\right) \quad (5.14)$$

where E_{max} is the estimated maximum value of the objective function.

- Selection: Only the cellular GA has selection operator and it has no modification when it is implemented to the EDJSP.

- Crossover: To work with the modified operation-based encoding of the cellular GA, the operation-based order crossover [3] is utilized as the crossover operator

and works for genes in the chromosome within the range $[r2, \sum_{j \in J \cup J'} o_j - 1]$. Firstly, it randomly chooses the same operations from two selected 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 Figure 43 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 $[r2, \sum_{j \in J \cup J'} o_j - 1]$ 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.43. An example of the operation-based order crossover

The pseudo GA initializes every pair of parents with complementary chromosomes and the crossover is executed between the offspring from the same parents. Therefore, we take the one-point precedence preservative crossover [22] to work with genes within the range $[r2, \sum_{j \in J \cup J'} o_j - 1]$ for the pseudo GA to keep the complementary chromosomes as much as possible. An example of the same job shop instance in the operation-based crossover is presented in Figure 44 which shows this process in details. The genes out of the range are marked in red. Firstly, a crossover point for the paired parents is selected randomly. The genes within the range and before the cross point are kept for their own off-springs (in blue) while the same genes are deleted from the paired parents (back ground color in yellow). Finally, the genes left in the paired parent are appended to fill the empty positions after the cross point in the original parents.

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

Fig.44. An example of the one-point precedence preservative crossover

The crossover is executed if a specified probability is satisfied. In this Chapter, a fixed crossover rate is taken for the pseudo GA where the value is set as 1. On the other hand, an adaptive crossover rate [25] is adopted for the cellular GA and its expression is given as

$$\text{Crossover rate} = \varphi_1(f_{max} - f')/(f_{max} - \bar{f}),\ f' \geq \bar{f} \quad (5.15)$$

$$\text{Crossover rate} = \varphi_2,\ f' < \bar{f} \quad (5.16)$$

Where $\varphi_1,\ \varphi_2 \leq 1.0$.

- Mutation: The swap mutation is employed for $X'(k)$ of the cellular GA where different arbitrary genes within the range $[r2, \sum_{j \in J \cup J'} o_j - 1]$ are chosen and exchange values. Concerning $Y'(k)$ of the cellular GA, unfixed amount of genes are replaced by random values in the range, apart from the original ones. Following the above example, the procedure is illustrated in Figure 45 where genes in green illustrate the execution of mutation. Moreover, a mutation is imposed to the pseudo GA. One of its paired parents is designed to execute the same mutation operation as the cellular GA while the rest parent is mutated by the complementary initialization strategy to keep complementary chromosomes.

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.45. An example of the crossover

As same as the crossover rate, the mutation rate for the pseudo GA is fixed and is kept as 0.1 while the mutation rate of the cellular GA is adaptive in response to individuals' fitness values [25] and take the form as

$$\text{Mutation rate} = \varphi_3(f_{max} - f)/(f_{max} - \bar{f}),\ f \geq \bar{f} \quad (5.17)$$

$$\text{Mutation rate} = \varphi_4,\ f < \bar{f} \quad (5.18)$$

Where $\varphi_3,\ \varphi_4 \leq 1.0$.

- Replacement: No modification.
- Migration: No modification.

VI. 4 Numerical Experiments

Test 1 checks the efficiency and the effectiveness of parallel GA II for solving the energy efficient JSP while test 2 evaluates the performance of EDJPS by a case study. The values of ET_{max}, ET_{min}, EE_{max}, EE_{min}, ED_{max}, ED_{min} are found through the simple GA stated in Chapter III and Chapter IV by solving each of them as a single objective in the following implementations. To prevent the GA from getting stuck at a local optimum, solutions need to be completely disrupted. For this reason, we set a value of 1.0 for φ_1 and φ_2, a value of 0.5 for φ_3 and φ_4 as in [25]. Moreover, the experimental platform is kept the same as in the previous two Chapters.

VI.4.1 Evaluation

In addition to parallel GA II, two kinds of dual homogeneous island GAs are taken as comparisons. The first one utilizes the cellular GA on GPUs for its two islands while the pseudo GA is adopted for the two islands of the second one. Their other operators

keep the same settings as parallel GA II. Experiments are conducted in terms of the energy efficient JSP without considering new urgent arrival jobs. In this case, six tested problems are generated as in [26]. These instances are referred to as "easy problems" or "hard problems" with names EASY 6×4, EASY 10×8, EASY 20×10, HARD 6×4, HARD 10×8 and HARD 20×10. EASY 6×4 and HARD 6×4 are 6-job, 4-machine problems; EASY 10×8 and HARD 10×8 are 10-job, 8-machine problems; EASY 20×10 and HARD 20×10 are 20-job, 10-machine problems. Every job consists of the same amount of operations as the amount of machines, while one operation is always performed on a single machine. Moreover, each 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. Other experimental relative data are defined in Table 24.

Table 24 The experimental relative data of energy efficient JSP

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

Concerning the solutions' quality comparison, it is shown in Table 25 that the dual heterogeneous island GA and the dual cellular island GA have similar performance while the former one displays better the best value and the latter one illustrates better the average value. On the other hand, the results of the dual pseudo island GA are relatively poor. Since the EDJSP works with the operation-based encoding, the order of genes in one chromosome has great influence to the results. Unlike a simple FFSP discussed in Chapter 3, the efficiency of the complementary parent strategy get decreased unfortunately.

Table 25 Solutions' quality comparison (Population Size=2048)

Problems	Dual Heterogeneous Island GA		Dual Cellular Island GA		Dual Pesudo Island GA	
	Avg.	Best	Avg.	Best	Avg.	Best
EASY 6×4	0.013307	0.013307	0.013363	0.013307	0.037983	0.021057
EASY 10×8	0.090601	0.074350	0.086318	0.075833	0.266073	0.204708
EASY 20×10	0.037718	0.022688	0.032669	0.014034	0.359445	0.292384
HARD 6×4	0.013307	0.013307	0.013307	0.013307	0.044449	0.022629
HARD 10×8	0.079581	0.066118	0.076593	0.066203	0.269521	0.208583
HARD 20×10	0.032360	0.014858	0.029107	0.015666	0.281563	0.253130

As the cellular GA on GPUs performs best in Chapter 3 by its twice threads occupancy, we try to enhance the computation capability on the multi-core CPU further in this Chapter. Therefore, the pseudo GA on a multi-core CPU is parallelized not only using OpenMP [27] but also the SIMD vectorization via SSE2 [28]. As the execution time comparison shown in Table 26, the dual heterogeneous island GA on GPUs and a multi-core CPU overcomes the dual cellular island GA on GPUs because of the simultaneous execution on both sides. Moreover, 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 dual pseudo GA only deals only with integers whose storage size is small, the contribution of SIMD vectorization is impressive and the dual pseudo GA on a 4 core CPU with vectorization takes the least execution time in most instances.

Table 26 Execution time comparison

Problems	Population Size	Dual Heterogeneous Island GA on GPUs and a Multi-core CPU	Dual Cellular Island GA on GPUs	Dual Pesudo Island GA on a Multi-core CPU
EASY 6×4	2048	10.28 s	19.67 s	5.53 s
	8192	28.21 s	50.99 s	31.14 s
	32768	122.62 s	154.03 s	150.85 s
EASY 10×8	2048	62.12 s	120.11 s	24.87 s
	8192	169.63 s	334.24 s	108.42 s
	32768	631.04 s	1208.30 s	464.10 s
EASY 20×10	2048	377.85 s	692.13 s	224.78 s
	8192	1089.00 s	1849.96 s	896.22 s
	32768	4078.54 s	6771.42 s	3560.37 s
HARD 6×4	2048	10.53 s	19.21 s	4.87 s
	8192	27.46 s	50.48 s	31.02 s
	32768	120.01 s	152.41 s	150.96 s
HARD 10×8	2048	60.78 s	115.00 s	24.97 s
	8192	174.78 s	342.73 s	109.47 s
	32768	627.20 s	1222.74 s	489.49 s
HARD 20×10	2048	370.06 s	691.85 s	290.73 s
	8192	1074.19 s	1843.71 s	879.43 s
	32768	3946.17 s	6735.54 s	3609.78 s

VI.4.2 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 [29] (MT 10×10) 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 at the point that equals 30% of the makespan of original schedule. The operation sequence of original jobs and their processing times on target machine at speed level 0 are collected from MT10×10. On the other hand, these values for new urgent arrival jobs are generated following the rule of "hard problems". The values of energy consumption at level 0 $\left(Q_{jsm0}\right)$ and due time (D_j) are set as in Table 24 while the

value of release time (R_j) is fixed as 0. 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 27. Moreover, the processing time and energy consumption when operation s of job j processed on machine m at different level 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 priority of the schedule repair strategy.

An optimal solution of the original schedule is shown by Gantt chart in Figure 43. 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 and the uncompleted operations of original jobs are rearranged to make use of machines only when they are idle. By implementing the schedule repair strategy, an optimal solution illustrated by the Gantt chart of the updated schedule in Figure 44 presents that the processing time of some operations are obviously decreased. As a result, most original jobs' finishing time are delayed slightly which is confirmed by the details displayed in Table 28.

Table 27 The case data of an 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 28 Original jobs' finishing time comparison between an optimal solution of the original schedule and an optimal solution of the update schedule

	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

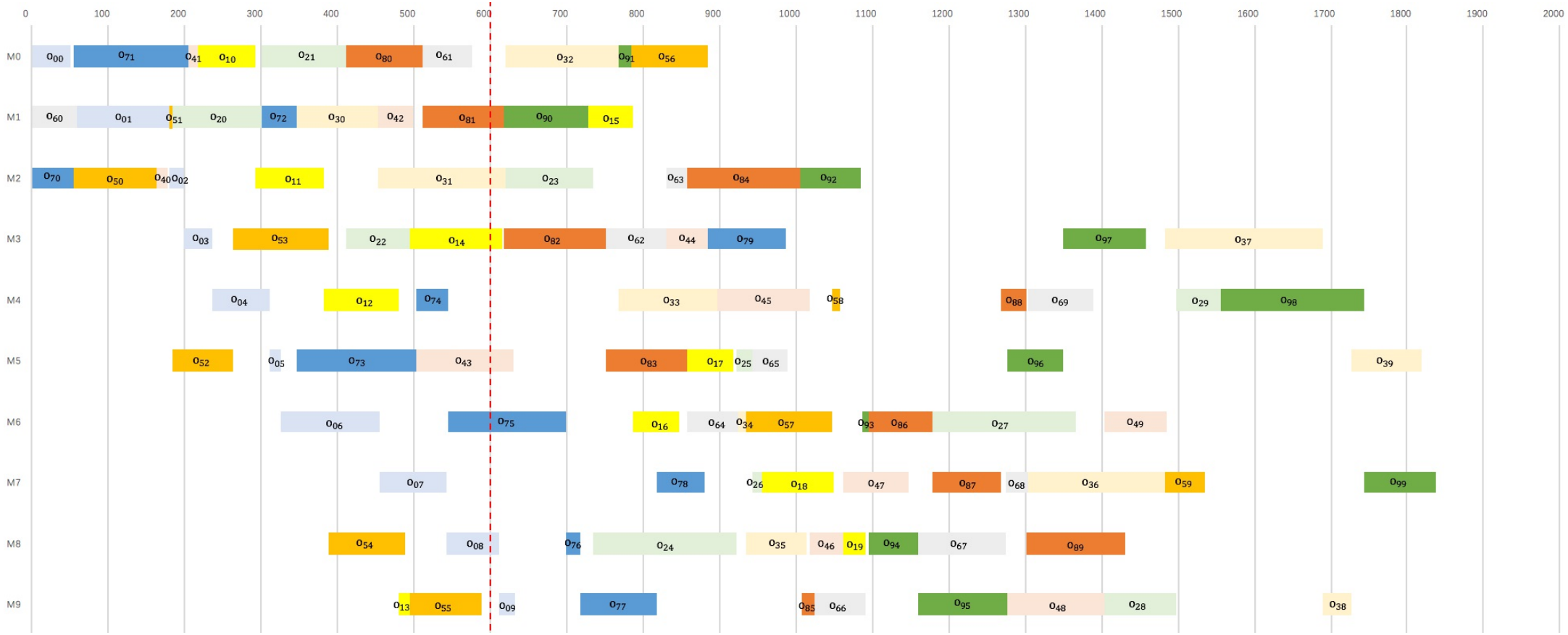

Figure 43 Gantt chart of the original schedule of an optimal solution

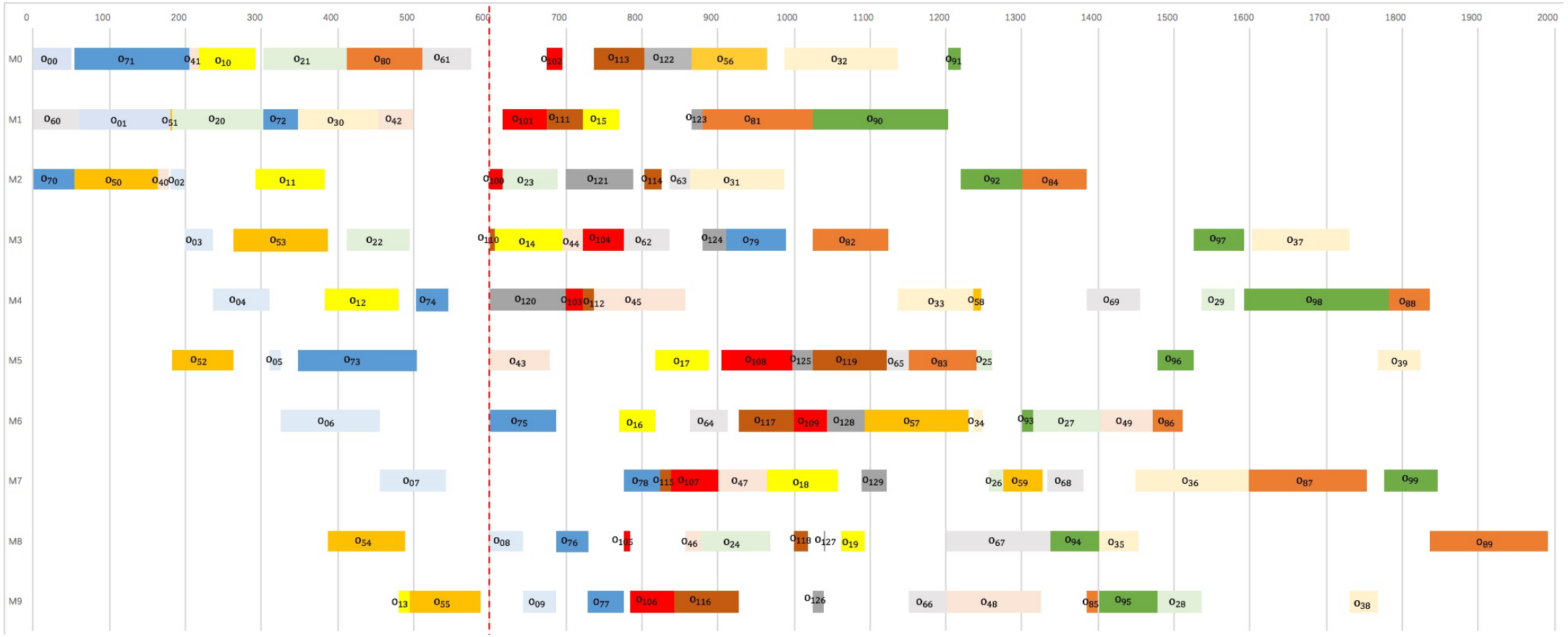

Figure 44 Gantt chart of the updated schedule of an optimal solution

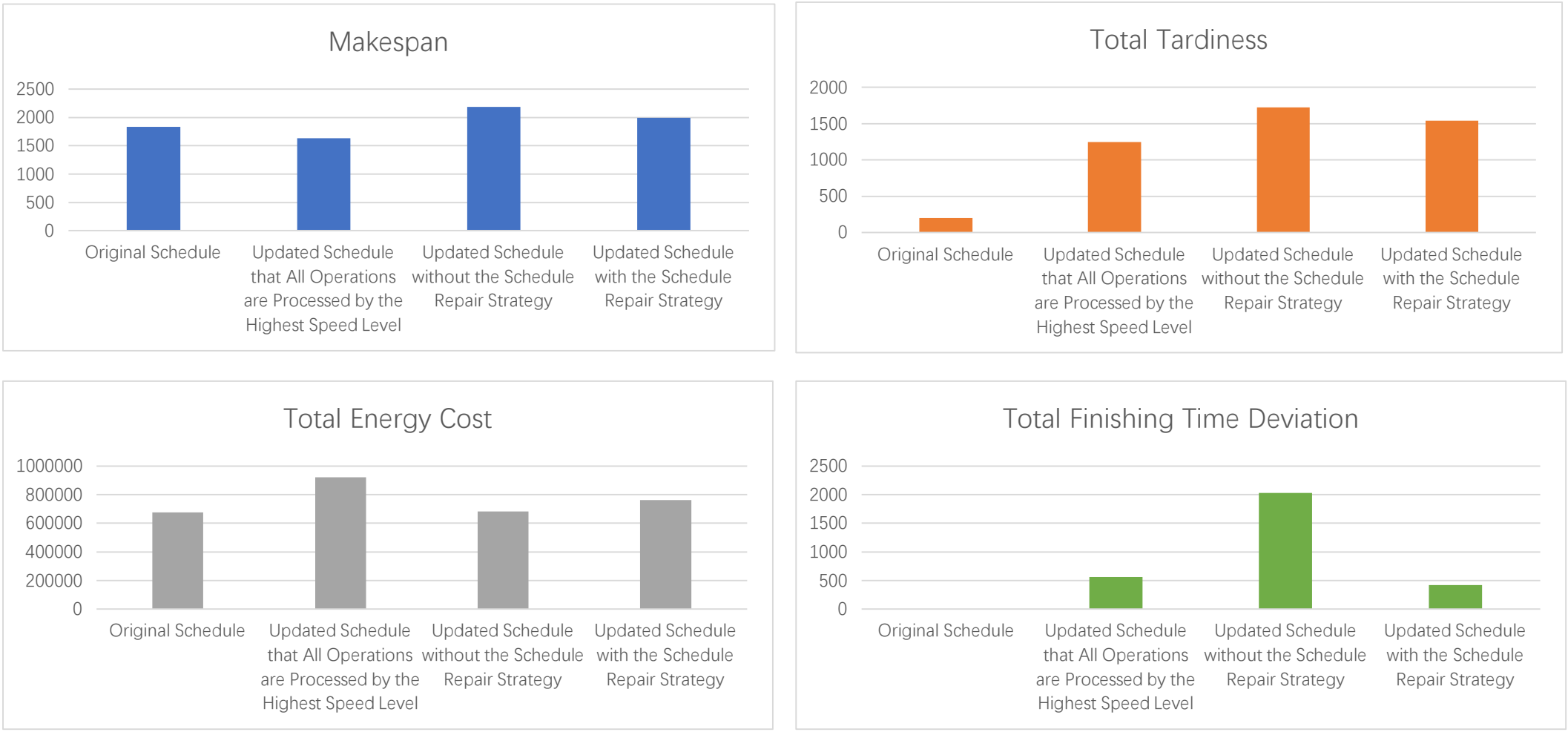

Figure 45 Comparison between the original schedule and three updated schedules

Moreover, a comparison concerning the makespan, the total tardiness, the total energy cost and the total finishing time deviation between the original schedule and three updated schedules is demonstrated in Figure 45. The updated schedule that all operations are processed by the highest speed level can decrease the makespan and the total tardiness maximum. However, the total energy cost is increased and its impact to minimize the total finishing time deviation is limited. The updated schedule without the schedule repair strategy keeps the total energy cost almost the same level as the original schedule while the other three parameters all go up, particularly the total finishing time deviation. Finally, the updated schedule with the schedule repair strategy minimize the total finishing time deviation maximum by holding the makespan, the total tardiness and the total energy cost within a reasonable level.

VI. 5 Conclusion

In this Chapter, an investigation into minimizing total tardiness and total energy consumption in the job shop with new urgent arrival jobs was firstly studied. In order to provide an adequate renewed scheduling plan in a short response, the dual heterogeneous GA executed simultaneously on different parallel platforms with the event driven schedule repair approach was updated. When dealing with six kinds of energy efficient JSP in the evaluation, the designed method verified its performance by obtaining competitive results as the dual cellular GA on GPUs while decreasing the execution time significantly. Moreover, it pointed out that the balance of computation capability between the host and the device had a great influence on its overall effectiveness. Concerning the EDJSP in the case study, an optimal solution of the updated schedule is shown by Gantt chart. Compared with the original schedule, the processing time of some operations are obviously decreased. Furthermore, we confirmed the efficiency of the event driven schedule repair strategy by minimizing the total finishing time deviation while holding the makespan, the total tardiness and the total energy cost within a reasonable level.

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].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.

[4].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.

[5].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.

[6].Ouelhadj, D., & Petrovic, S. (2009). A survey of dynamic scheduling in manufacturing systems. Journal of scheduling, 12(4), 417.

[7].Tang, D., Dai, M., Salido, M. A., & Giret, A. (2016). Energy-efficient dynamic scheduling for a flexible flow shop using an improved particle swarm optimization. Computers in Industry, 81, 82-95.

[8].Zhang, L., Li, X., Gao, L., & Zhang, G. (2016). Dynamic rescheduling in FMS that is simultaneously considering energy consumption and schedule efficiency. The International Journal of Advanced Manufacturing Technology, 87(5-8), 1387-1399.

[9].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.

[10].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.

[11].Gholami, M., Zandieh, M., & Alem-Tabriz, A. (2009). Scheduling hybrid flow shop with sequence-dependent setup times and machines with random breakdowns. The International Journal of Advanced Manufacturing Technology, 42(1-2), 189-201.

[12].Cantú-Paz, E. (1998). A survey of parallel genetic algorithms. Calculateurs paralleles, reseaux et systems repartis, 10(2), 141-171.

[13].Gozali, A. A., & Fujimura, S. (2017, May). Localization strategy for island model genetic algorithm to preserve population diversity. In International Conference on Computer and Information Science (pp. 149-161). Springer, Cham.

[14].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.

[15].Benner, P., Ezzatti, P., Kressner, D., Quintana-Ortı, E. S., & Remón, A. (2011). A mixed-precision algorithm for the solution of Lyapunov equations on hybrid CPU–GPU platforms. Parallel Computing, 37(8), 439-450.

[16].Bilel, B. R., Navid, N., & Bouksiaa, M. S. M. (2012, October). 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 (pp. 44-53). IEEE Computer Society.

[17].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.

[18].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.

[19].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.

[20]. 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.

[21]. 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.
[22]. 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.
[23]. Liu, M., & Wu, C. (2008). Intelligent optimization scheduling algorithms for manufacturing process and their applications. National Defense Industry Press, 334.
[24].Chen, Q., Zhong, Y., & Zhang, X. (2010). A pseudo genetic algorithm. Neural Computing and Applications, 19(1), 77-83.
[25].Srinivas, M., & Patnaik, L. M. (1994). Adaptive probabilities of crossover and mutation in genetic algorithms. IEEE Transactions on Systems, Man, and Cybernetics, 24(4), 656-667.
[26].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.
[27]. http://www.openmp.org/
[28]. https://en.wikipedia.org/wiki/Streaming_SIMD_Extensions
[29].Muth, J. (1963). Probabilistic learning combinations of local job-shop scheduling rules. Industrial Scheduling.

Chapter VI. Conclusion and Future Works

This thesis focuses on solving energy efficient dynamic shop scheduling problems by parallel GAs. Because of a growing economical competitive landscape and higher environmental norms, it is now vital for manufacturing companies to consider energy efficiency when dealing with traditional shop scheduling problems. Meanwhile, scheduling problems are dynamic in the real world with unexpected events after the start time. The dynamic scheduling is one familiar problem that cannot be ignored in manufacturing practices. Two energy efficient dynamic shop scheduling problems are studied in this thesis.

The Energy efficient Dynamic Flexible Flow Shop scheduling Problem (EDFFSP) takes the way to limit the peak power as electricity consumption and operating costs of manufacturing plants are usually charged based on the peak power demand from electricity providers. In this case, the later assigned operation needs to be delayed when the power's peak is met. The total tardiness and the makespan are considered as two objectives in a flexible flow shop scheduling problem. A set of new jobs may arrive after the start of the original plan. With the complete rescheduling strategy, they are processed from the beginning of the rescheduling point with the remaining uncompleted operations of original jobs. The rescheduling points always occur with regular intervals and gather all new arrival jobs' information.

The Energy efficient Dynamic Job Shop scheduling Problem (EDJSP) focuses on minimizing the total energy consumption within a job shop scheduling problem. The processing time and the energy consumption 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 consumption increase. The new arrival jobs trigger the rescheduling and are treated as urgent tasks in which the production

line should conduct them immediately. The operations being processed are terminated and need to be rescheduled with the remaining uncompleted operations of original jobs based on the insertion of new urgent arrival jobs. The updated schedule refers some local adjustment of the original one. Thus, three objectives are included in this problem, the total tardiness, the total energy consumption and the total weighted finishing time deviation of the updated schedule from the original one.

The Genetic Algorithm (GA) is considered as one of the most efficient method to solve shop scheduling problems. However, there is an increase in the required time to find adequate solutions when GAs are applied to complex and large problems. On the other hand, new integrated energy requirements in a dynamic environment lead to the complexity of the considered problem to be higher and ask even longer execution time to get acceptable solutions. In order to find adequate solutions for energy aware shop scheduling problems efficiently and achieve a speedup to meet the short response in the dynamic environment, parallel implementation is considered as one of the most promising choices and two parallel GAs are developed in this thesis. The first one is taken for solving the EDFFSP. It is a hybrid model consisting of an island GA at the upper level and a fine-grained GA at the lower level. Since the fine-grained model obtains good population diversity when dealing with high-dimensional variable spaces and the island model converges faster by subpopulations, this design combines metrics from two levels to gain competitive results. Meanwhile, the hybrid structure achieves the maximum speedup through its high consistence with the CUDA framework. The second one is implemented to solve the EDJSP that is composed of a cellular GA and a pseudo GA. The 2D variable spaces of the cellular model and the complementary parent strategy of the pseudo model keep the population diversity while a penetration inspired migration policy shares information between them. Furthermore, this heterogeneous structure can be well parallelized on GPUs simultaneously with multi-core CPU and enjoys parallel computing resources from two sides.

The main work of Chapter 2 has been organized as a survey and is published in 2018 IEEE International Parallel and Distributed Processing Symposium Workshops. Concerning the two parallel GAs discussed in Chapter 3, the parallel GA I and its application to the FFSP is published in 2018 19th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed

Computing. Meanwhile, the work of parallel GA II and its performance analysis has been submitted to the journal entitled Mathematical Problems in Engineering. Moreover, the EDFFSP solved by the CUDA-based hybrid GA with the predictive reactive complete rescheduling strategy in Chapter 4 has been accepted by the Journal of Parallel and Distributed Computing.

In the future, we will try to improve the performance of the dual heterogeneous island GA when it is implemented to solve the EDJSP as in Chapter 5. Because the efficiency of the complementary parent strategy is decreased in this case, some modifications are required for the pseudo GA on a multi-core CPU to enhance its searching ability. On the other hand, the overall execution time will not be increased by controlling the work load balance between GPUs and a multi-core CPU with SIMD vectorization. This implementation is also planned to be executed on a computing node with V100 GPUs which is a new generation architecture. Since the cost of using computing accelerators like GPUs or Intel Xeon Phi is not expensive nowadays, these devices are strongly suggested to be utilized for solving hard optimization problems in manufacturing practice. Since only the linear combination is used in this thesis to deal with multi-objective problems, the Pareto approach is considered as the nest step with a parallel version of the non-dominated sorting genetic algorithm II.

List of Publications

[1]. 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. (IF:1.815)

[2]. Luo, J., & El Baz, D. (2018, May). A Survey on Parallel Genetic Algorithms for Shop Scheduling Problems. In 2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW) (pp. 629-636). IEEE.

[3]. Luo, J., El Baz, D., & Hu, J. (2018, June). Acceleration of a CUDA-Based Hybrid Genetic Algorithm and its Application to a Flexible Flow Shop Scheduling Problem. In 2018 19th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD) (pp. 117-122). IEEE.