Journal of Parallel and Distributed Computing 133 (2019) 244–257

Contents lists available at ScienceDirect

J. Parallel Distrib. Comput.

journal homepage: www.elsevier.com/locate/jpdc

GPU based parallel genetic algorithm for solving an energy efficient dynamic flexible flow shop scheduling problem

Jia Luo [a,*], Shigeru Fujimura [b], Didier El Baz [a], Bastien Plazolles [c]

[a] LAAS-CNRS, Université de Toulouse, CNRS, Toulouse, France
[b] Graduate School of Information, Production, and Systems, Waseda University, Kitakyushu, Japan
[c] Géosciences Environnement Toulouse (CNRS UMR5563), Université de Toulouse, Toulouse, France

HIGHLIGHTS

- An energy efficient dynamic flexible flow shop scheduling (EDFFS) model is studied.
- A parallel Genetic Algorithm (GA) with a predictive reactive complete rescheduling approach is developed.
- The parallel GA is highly consistent with the hierarchy of threads and memory of CUDA.
- The parallel GA obtains competitive results and reduces time requirements dramatically.
- The proposed method is flexible to solve the EDFFS problem and overcomes the traditional static approach.

ARTICLE INFO

Article history:
Received 29 September 2017
Received in revised form 29 June 2018
Accepted 30 July 2018
Available online 16 August 2018

Keywords:
Flexible flow shop
Energy efficiency
Dynamic scheduling
Hybrid parallel genetic algorithm
GPU Computing

ABSTRACT

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. This paper proposes an energy efficient dynamic flexible flow shop scheduling model using the peak power value with consideration of new arrival jobs. As the problem is strongly NP-hard, a priority based hybrid parallel Genetic Algorithm with a predictive reactive complete rescheduling strategy is developed. In order to achieve a speedup to meet the short response in the dynamic environment, the proposed method is designed to be highly consistent with the NVIDIA CUDA software model. Finally, numerical experiments are conducted and show that our approach can not only solve the problem flexibly, but also gain competitive results and reduce time requirements dramatically.

1. Introduction

About one half of the world's total energy is currently consumed by the industrial sector [10] and its energy consumption has nearly doubled over the last 60 years [11]. 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.

The adjustment of scheduling strategies only requires modest time and cost investment, compared with the redesign methods for machines or processes [12]. Therefore, a lot of traditional scheduling strategies considering minimizing the total energy consumption have been studied [22,26,27]. Meanwhile, some efforts have been made on taking the peak power into account, because electricity consumption and operating costs of manufacturing plants are usually charged based on the peak power demand from electricity providers [39]. Most of the researches only concentrate on establishing a mathematical model for solving this optimization problem in a static environment. But in fact, scheduling problems are dynamic in the real world with uncertain new arrival jobs after

* Corresponding author.
E-mail address: jluo@laas.fr (J. Luo).

https://doi.org/10.1016/j.jpdc.2018.07.022
0743-7315/

the start time. Few works [36,42] have taken reactive approaches into consideration for supporting energy efficient dynamic systems. However, they care only about the improvement of algorithms to gain better quality solutions, while ignoring the time consumption to implement such approaches. Without a doubt, a method proposing an adequate rescheduling plan in a short response time is greatly desired in this case.

In the last decade, Graphics Processing Units (GPUs) have gained widespread popularity as computing accelerators for High Performance Computing (HPC) applications [2]. Researches on GPU-based approaches for solving scheduling problems [4,8,25,31] have won favor in recent years with the development of Compute Unified Device Architecture (CUDA) in 2006 [16]. Despite all those advances, a complex problem such as energy efficient dynamic flexible flow shop scheduling has not been considered as far our knowledge is concerned. Additionally, many parallel Genetic Algorithm (GA) implementations for solving optimization problems have shown their success [21,33,37] as seen in the literature. Therefore, the GPU based parallel GA for solving an energy efficient dynamic flexible flow shop scheduling problem remains an open research challenge based on the previous works, and the one that we seek to address in this paper.

The total tardiness and the makespan with a peak power limitation are analyzed in this paper while considering the flexible flow shop in a dynamic environment. A predictive reactive 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 the 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. Specially, the contributions of our work are summarized as followed:

1. We propose an energy efficient dynamic flexible flow shop scheduling (EDFFS) model using the peak power value with consideration of new arrival jobs.
2. A priority based hybrid parallel GA mapping to the NVIDIA CUDA software model is developed with a predictive reactive complete rescheduling approach.
3. Computational tests show that our method can not only solve the EDFFS problem flexibly, but also gain competitive results and reduce time requirements dramatically.

The remaining sections of this paper are organized as follows. Section 2 introduces related works. Section 3 describes the research problem and the mathematical model. Section 4 presents the priority based hybrid parallel GA for solving the energy efficient dynamic flexible flow shop scheduling problem. Section 5 illustrates numerical experiments and results analysis. Finally, Section 6 states conclusions.

2. Related works

Recently, there has been a growing interest in reducing energy consumption in manufacturing processes. Several works tried to limit the peak power in parallel multi-machine contexts. Fang et al. [12] presented a multi-objectives mixed-integer programming model of the flow shop scheduling problem that considered the peak power load, the energy consumption, and the associated carbon footprint in addition to the cycle time. Bruzzone et al. [3] 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 [39] to achieve a global optimal solution between the peak power and the traditional production efficiency without compromise on computing efficiency. As delaying production activities may not be acceptable in manufacturing, minimizing the total energy consumption within the traditional scheduling problem is an alternative solution. Liu et al. [22] developed a model for the bi-objectives problem that minimized the total electricity consumption and the total weighted tardiness, where a non-dominant sorting genetic algorithm is used to obtain the Pareto front. Similarly, an emission-aware multi-machine job shop scheduling model was addressed in [40] and was solved through a modified multi-objectives genetic algorithm. Dai et al. [9] 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 makespan and total energy consumption. To sum up, numerous works have focused on various energy efficient shop scheduling problems in a static perspective. But, due to frequently inevitable new arrival jobs in the production environment, a fixed preset scheduling plan could not meet the requirement.

Dynamic scheduling problems are more complex than static scheduling problems. A lot of methods have been taken to solve this kind of problems [28]. Most of them only considered the efficiency of the traditional scheduling problem without including energy efficient demand. Tang et al. [36] adopted a predictive reactive approach with an improved particle swarm optimization to search for the Pareto optimal solution of dynamic flexible flow shop scheduling problems that minimized the energy consumption and the makespan. Pach et al. [29] 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. Zhang et al. studied the dynamic rescheduling in flexible manufacturing systems considering the energy consumption and the schedule efficiency in [42] with a new goal programming math model, a rescheduling method based genetic algorithm and a period policy. In a word, some efforts to solve energy efficient dynamic 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 for large-scale manufacturing problems.

In recent years, various algorithms, like the branch and bound, the genetic algorithm, the Tabu search, using GPUs have been successfully applied to generate optimized results for scheduling problems with impressive time decrease. Melab et al. [25] 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. [7] 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. Zajicek et al. [41] studied a parallel island-based genetic algorithm for solving the flow shop scheduling problem by carrying out all computations on GPUs in order to reduce data communication costs. Pinel et al. [31] presented GPU implementations on the Min–Min heuristic and the GraphCell, an advanced parallel cellular genetic algorithm, for solving very large instances of the scheduling of independent tasks problem. An improved genetic algorithm and its implementation on GPUs to search for the optimal solution to flow shop scheduling problems with fuzzy processing times and fuzzy due dates were discussed in [19]. These cases have confirmed that the parallel GAs on GPUs have good performance in solving scheduling problems. However, it is also revealed that few studies have been conducted to integrate GPUs computing in dynamic energy efficient scheduling problems, because of the complexity that is caused.

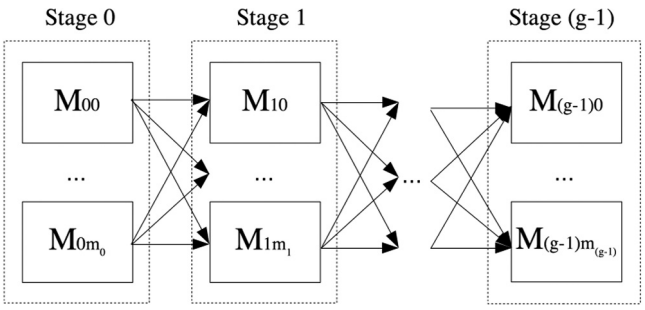

Fig. 1. A flexible flow shop layout.

Although many research works on scheduling problems have been studied in GPUs literature, none of them have so far, and to the best of our knowledge, considered the energy saving strategies and the dynamic environment completely. The above-mentioned efforts provide a starting point for exploring the GPU based hybrid parallel GA for solving an energy efficient dynamic flexible flow shop scheduling problem with competitive results and dramatical time reduction.

3. Problem statement

3.1. EDFFS problem description

The flexible flow shop scheduling problem (FFS) is a multistage production process that consists of two or more stages in series as illustrated in Fig. 1. 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.

An energy efficient dynamic flexible flow shop scheduling (EDFFS) is a further development of the FFS. 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. One instance of the EDFFS problem consists of a set J of jobs, a set S of stages and a set M_s of machines at each stage. Each job $j \in J$ on machine $m \in M_s$ has its corresponding processing time and power consumption. Furthermore, there is a power's peak limitation when the system operates. As an FFS problem is considered to be NP-hard in essence and difficult to solve [13], the EDFFS problem is a NP-hard combinatorial optimization problem and more complex than the FFS problem. Additionally, required conditions for the EDFFS are shown in Table 1.

3.2. Mathematical model of the EDFFS

For an easy presentation, we summarize the notations used along the rest of the paper in Table 2.

To achieve the power's peak limitation and to minimize the traditional makespan and the total tardiness objectives, the formal mathematical model for the EDFFS is an extension of the mathematical models presented in [3,39] to cover rescheduling. The formulation is given in the following.

Objective function:

$$\min:\ WT * \sum_{j \in J \cup J'} T_j + C_{max} \tag{1}$$

Constraints:

$$T_j = \max(S_{j(g-1)} + P_{j(g-1)M_{j(g-1)}} - D_j, 0) \quad j \in J \cup J' \tag{2}$$

$$C_{max} = \max_j(S_{j(g-1)} + P_{j(g-1)M_{j(g-1)}}) \quad j \in J \cup J' \tag{3}$$

$$S_{j0} \geq R_j \quad j \in J \cup J' \tag{4}$$

$$S_{js} \geq S_{j(s-1)} + P_{j(s-1)M_{j(s-1)}} \quad j \in J \cup J',\ s \in S,\ s > 0 \tag{5}$$

$$S_{js} + P_{jsM_{js}} \leq S_{is} \quad j \in J \cup J',\ i \in J \cup J',\ s \in S,\ j \neq i,$$
$$M_{js} = M_{is},\ S_{js} \leq S_{is} \tag{6}$$

$$Q_{max} \geq Q_t \quad t \in T \tag{7}$$

$$Q_t = \sum_{j \in J \cup J'} \sum_{s \in S} Q_{jsMjs} * u_{jst} \quad t \in T \tag{8}$$

$$u_{jst} = \begin{cases} 1 & j \in J \cup J',\ s \in S,\ S_{js} \leq t < S_{js} + P_{jsM_{js}} \\ 0 & \text{otherwise} \end{cases} \tag{9}$$

$$RS \leq S_{js} \quad j \in J \cup J',\ s \in S \tag{10}$$

The decision variables in this mathematical model are M_{js} and S_{js}. As two scheduling objectives are considered, it is formulated as a single additive objective function (1) by aggregating the total tardiness and the makespan with the weight WT. As tardy jobs typically cause penalty costs [30] and have a great influence on customers' satisfaction, the weight WT indicates the priority of the first objective. Constraints (2) and (3) define the tardiness of jobs and the makespan separately. The precedence among operations due to the jobs' processing cycles is presented by constraints (4) and (5), while constraint (6) establishes the precedence caused by the sequencing on machines. In addition, constraint (7) introduces the power's peak by an upper bound whereas the power consumption during a certain period is expressed by constraint (8). Constraint (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 (10) imposes the definition of rescheduling.

4. Solving approach

4.1. Predictive reactive complete rescheduling strategy

The predictive reactive method is the most common dynamic scheduling approach used in manufacturing systems [28]. To solve the EDFFS, 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. An hybrid parallel GA on GPUs is proposed for solving the 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 [28]. Fig. 2 summarizes the flow of the predictive reactive complete rescheduling process.

4.2. Hybrid parallel GA model

The Genetic Algorithm (GA) is a stochastic search algorithm based on the principle of natural selection and recombination [15]. However, there is an increase in the required time to find adequate solutions when GAs are applied to more complex and larger problems. Parallel implementation is considered as one of the most promising choices to make it faster. The CUDA framework is chosen to parallelize the GA on GPUs in this paper. It is a Single Instruction, Multiple Threads (SIMT) parallel programming model. The parallel threads are grouped into blocks which are organized in a grid [32] as shown in Fig. 3 using the local memory, the shared memory and the global memory respectively.

Table 1
Additional required conditions for the EDFFS.

Number	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	There is no precedence between operations of different jobs, but there is precedence among operations due to the jobs' processing cycles.
4	Preemptive operations are not allowed.
5	Each job is available for processing after the release time.
6	Machines may suffer new arrival jobs at any time after the rescheduling point.
7	Processing times and average power consumption for any operation of all jobs on any machine are known.
8	Setup times for job processing and machine assignment times between stages are not taken into consideration.
9	There is infinite intermediate storage between machines.

Table 2
A description of notations used in all formulae.

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	Number of machines at the stage s. Each stage has the same amount of machines
H	Time horizon
J	Set of original jobs, $J = \{0, 1, 2, \ldots, n-1\}$
J′	Set of new arrival jobs, $J' = \{0, 1, 2, \ldots, n'-1\}$
S	Set of stages, $S = \{0, 1, 2, \ldots, g-1\}$
M_s	Set of machines at the stage s, $s \in S$, $M_s = \{0, 1, 2, \ldots, o-1\}$
T	Set of time periods, $T = \{1, 2, 3, \ldots, H\}$
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, $C \in R_+$

There are different ways of exploiting parallelism in GAs: master–slave models, fine-grained models, island models, and hybrid models [6]. Fine-grained models can perform well due to the larger genetic diversity obtained by dividing the population into a number of subpopulations [20]. Island models are the most famous for the research on parallel GAs. Populations on islands are free to converge toward different sub-optima with a faster improvement of the average fitness [6] and a migration operator can help mix good features that emerge from different local islands. To obtain a good speedup from the CUDA framework and to combine the

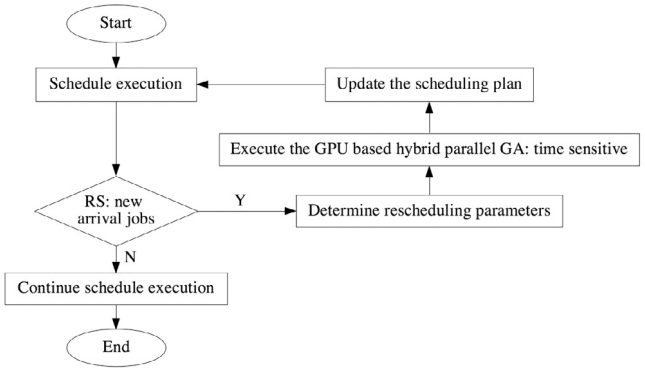

Fig. 2. The flow of predictive reactive complete rescheduling process for the EDFFS.

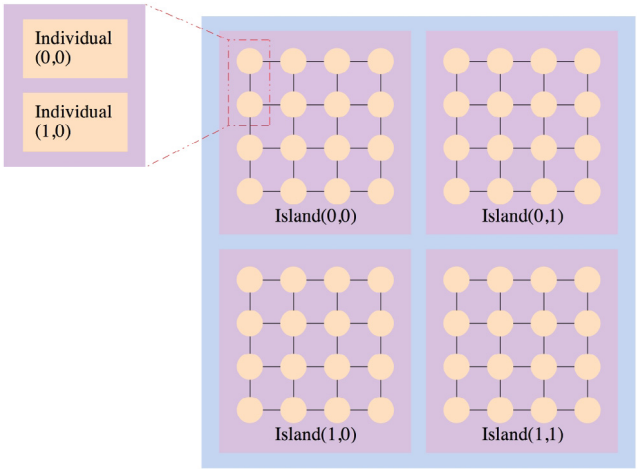

Fig. 4. The structure of hybrid parallel GA.

Table 3
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

advantages of fine-grained models and island models, we establish the hybrid model presented in Fig. 4 with a fine-grained GA at the lower level and an island GA at the upper level. A correspondence between the hybrid parallel GA components and the hierarchy of CUDA threads is displayed in Table 3. It turns out that the proposed GA is highly consistent with the hierarchy of CUDA threads and could utilize different types of memory effectively.

At the lower level, each CUDA thread processes one GA individual. Because of the 2D grid, GA individuals can get connected completely with this topology. A tournament based selection is executed with the cooperation of global memory and texture memory as the texture memory has a fast response to read information from neighbors. The crossover, the mutation and the fitness function calculation are generated using 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 after every generation inside the island and a migration among islands every 10 generations are carried. The shared memory is chosen to complete these work primarily while the overwriting is processed via the global memory synchronously. The procedure of the hybrid parallel GA with memory management is expressed in Fig. 5. More details are discussed in Section 4.4.

4.3. Priority based encoding representation

According to the problem description in Section 3, a target machine matrix X(k), stored on the GPU global memory with $n+n'$

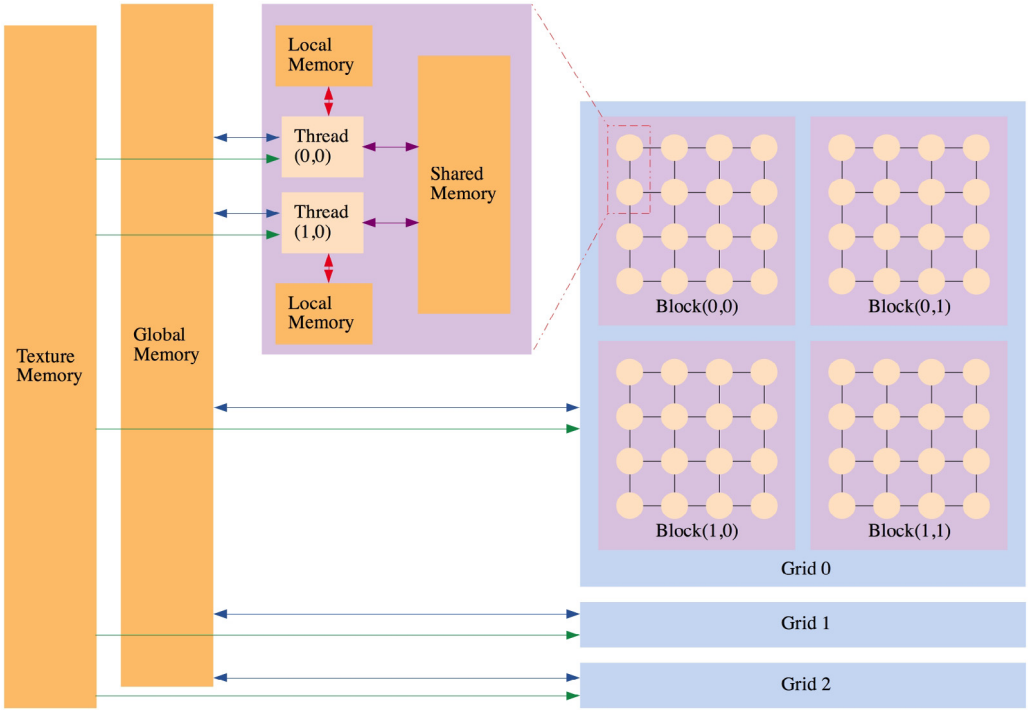

Fig. 3. The hierarchy of threads and different types of memory of CUDA framework.

rows and g columns, is presented in (11).

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

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

Moreover, (12) shows a $(n+n') \times g$ matrix placed on the GPU global memory that expresses the priority relation among operations.

$$Y(k) = \begin{bmatrix} y_{00}(k) & y_{01}(k) & \cdots & y_{0(g-1)}(k) \\ y_{10}(k) & y_{11}(k) & \cdots & y_{1(g-1)}(k) \\ \vdots & \vdots & y_{js}(k) & \vdots \\ y_{(n+n'-1)0}(k) & y_{(n+n'-1)1}(k) & \cdots & y_{(n+n'-1)(g-1)}(k) \end{bmatrix} \tag{12}$$

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

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 EDFFS 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 the 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) (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.

$$Z(k) = \begin{bmatrix} z_{00}(k) & z_{01}(k) & \cdots & z_{0(g-1)}(k) \\ z_{10}(k) & z_{11}(k) & \cdots & z_{1(g-1)}(k) \\ \vdots & \vdots & z_{js}(k) & \vdots \\ z_{(n+n'-1)0}(k) & z_{(n+n'-1)1}(k) & \cdots & z_{(n+n'-1)(g-1)}(k) \end{bmatrix} \tag{13}$$

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

Algorithm 1

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

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

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, $m \in M_s, t \in T$
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 EDFFS is presented in Table 4. 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. Fig. 6 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 Fig. 7. However, the predictive reactive 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 Fig. 8.

The following matrices show the EDFFS decoding result for the example. Each row of these matrices represents a job and

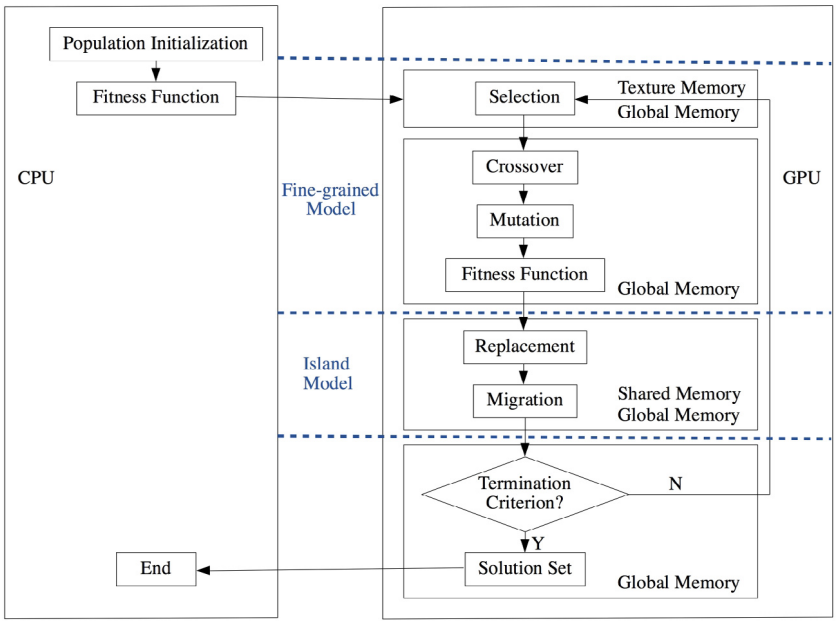

Fig. 5. The procedure of parallel hybrid GA with memory management.

Table 4
An example of the 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		

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 to 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 Fig. 8. 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.

4.4. Priority based GA operations on GPUs

- The fitness function: The hybrid parallel GA assesses the solutions based on the fitness function. In general, it is generated by the objective function to evaluate the solution domain. Since most shop scheduling problems are minimization problems [38] and the EDFFS is not an exception, the above-mentioned objective function Eq. (1) is transformed into the fitness function as

$$\text{Fitness function} = \max(E_{max} - (WT * \sum_{j \in J \cup J'} T_j + C_{max}), 0), \quad (14)$$

  where E_{max} is the estimated maximum value of the objective function.
- The selection operation: On the basis of the value of fitness function, the larger fitness an individual has, the higher the chance it has to be chosen in the next generation. Because the 2D grid is adopted as the spatial population structure where each grid point contains one individual, the local asteroid selection is taken to make the selection operation. Moreover, since GPU texture caches are designed to gain an increase in performance accelerating access patterns with a great deal of spatial locality [34], we define the neighborhood on the grid always contains 5 individuals: the considered one and neighboring individuals as displayed in Fig. 9. Among these individuals, 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.

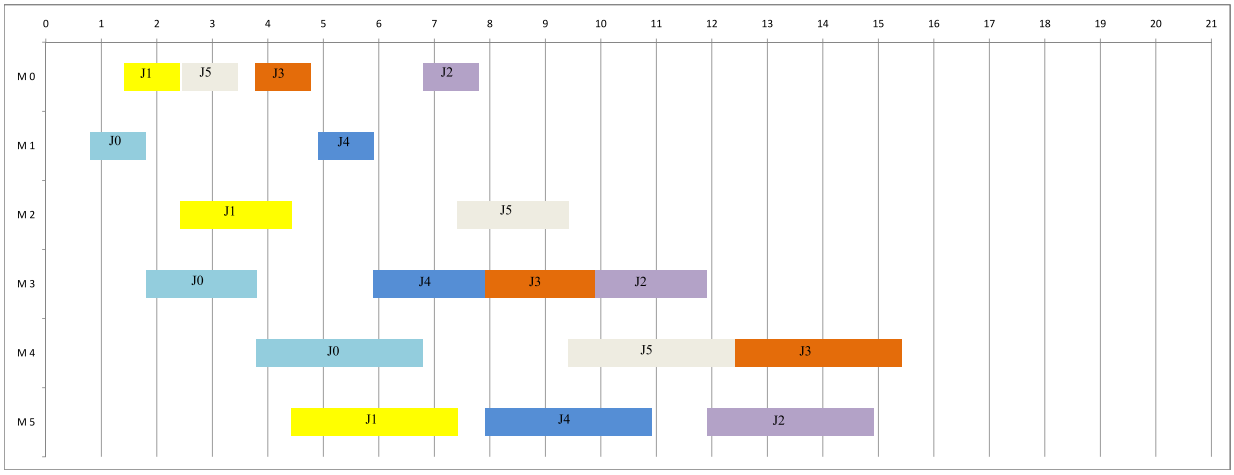

Fig. 6. The original schedule of an optimized solution.

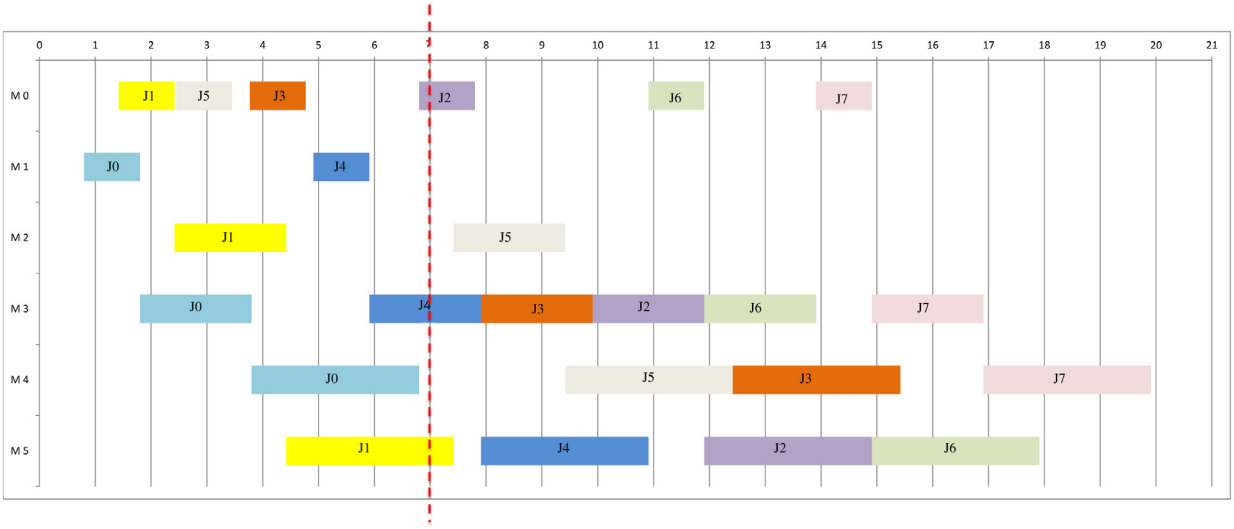

Fig. 7. The updated schedule of an optimized solution in a static environment ($\sum_{j\in J\cup J'} T_j = 1.64$, $C_{max} = 19.91$, Value of the Objective Function = 183.91).

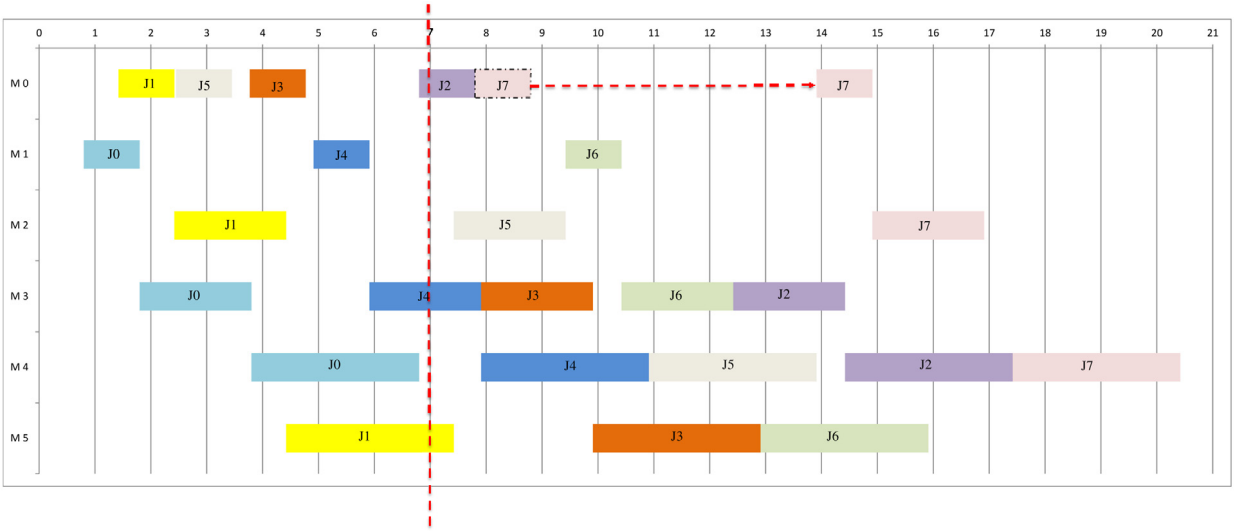

Fig. 8. The updated schedule of an optimized solution obtained by the proposed approach in a dynamic environment ($\sum_{j\in J\cup J'} T_j = 0.96$, $C_{max} = 20.42$, Value of the Objective Function = 116.42).

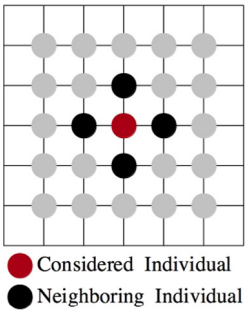

Fig. 9. The local asteroid selection.

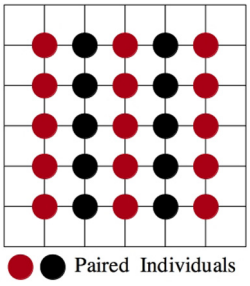

Fig. 10. The neighboring paired crossover.

- The crossover operation: We pair individuals with neighbors (See Fig. 10.) rather than selecting two from the population randomly. This strategy does not require global information sharing and is appreciated to work on the 2D grid architecture. Meanwhile, a risk that converges to the local minima can be eliminated by its cooperation with the local asteroid selection. In details, 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 are 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 Fig. 11.
- The mutation operation: Any individual in the population gets a random number generated on the interval 0–1. If it is smaller than the default mutation rate, the mutation operation is executed using the global memory in order to yield solutions with new information. 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.

$$\text{Before mutation}\quad X(k)=\begin{bmatrix}-1&1&0\\-1&-1&1\\1&0&1\end{bmatrix}, Y(k)=\begin{bmatrix}-1&6&1\\-1&-1&5\\4&3&2\end{bmatrix}$$

$$\downarrow$$

$$\text{After mutation}\quad X(k)=\begin{bmatrix}-1&0&1\\-1&-1&0\\0&1&0\end{bmatrix}, Y(k)=\begin{bmatrix}-1&2&1\\-1&-1&5\\4&3&6\end{bmatrix}$$

Fig. 12. An example of the mutation.

Regarding the priority matrix, two non-negative elements are chosen randomly to exchange the values. An example is given in Fig. 12.

- The replacement operation: 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.
- The migration operation: Islands are interconnected as a single ring as shown in Fig. 13. 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.

5. Numerical experiments

To analyze the performance of the proposed approach, test 1 and test 2 are conducted in terms of an energy efficient FFS without considering new arrival jobs. Test 1 configures the parameters of the proposed hybrid parallel GA, while test 2 shows its efficiency and effectiveness compared to the classical GA [15], the cellular GA [1] and the master–slave GA. New arrival jobs are included in test 3 to evaluate the performance of the EDFFS. A small size instance is considered in these 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 EDFFS with 3 different size problems. The instances are characterized by different numbers of jobs (n = 10, 50, 80) with different numbers of stages (g = 3, 4, 4), different numbers of machines (o = 2, 2, 3) in each stage and different numbers of power's peak (Qmax = 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. The

$$\text{Before crossover}\quad \bullet\ X(k)=\begin{bmatrix}-1&1&0\\-1&-1&1\\1&0&1\end{bmatrix}, Y(k)=\begin{bmatrix}-1&6&1\\-1&-1&5\\4&3&2\end{bmatrix} \qquad \bullet\ X(k)=\begin{bmatrix}-1&0&0\\-1&-1&0\\1&1&1\end{bmatrix}, Y(k)=\begin{bmatrix}-1&3&6\\-1&-1&1\\4&2&5\end{bmatrix}$$

$$\downarrow$$

$$\text{After crossover}\quad \bullet\ X(k)=\begin{bmatrix}-1&1&0\\-1&-1&0\\1&1&1\end{bmatrix}, Y(k)=\begin{bmatrix}-1&6&6\\-1&-1&1\\4&2&5\end{bmatrix} \qquad \bullet\ X(k)=\begin{bmatrix}-1&0&0\\-1&-1&1\\1&0&1\end{bmatrix}, Y(k)=\begin{bmatrix}-1&3&1\\-1&-1&5\\4&3&2\end{bmatrix}$$

$$\downarrow$$

$$\text{Correction}\quad \bullet\ X(k)=\begin{bmatrix}-1&1&0\\-1&-1&0\\1&1&1\end{bmatrix}, Y(k)=\begin{bmatrix}-1&6&3\\-1&-1&1\\4&2&5\end{bmatrix} \qquad \bullet\ X(k)=\begin{bmatrix}-1&0&0\\-1&-1&1\\1&0&1\end{bmatrix}, Y(k)=\begin{bmatrix}-1&3&1\\-1&-1&5\\4&6&2\end{bmatrix}$$

Fig. 11. An example of the neighboring paired crossover.

Table 5
The experimental relative data.

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 $\overline{P} = \sum_s \left(\sum_m P_{jsm}/o\right)$
D_j	$R_j + \overline{P}(1+\sigma)$, where $\sigma = U[0, 2]$
Q_{jsm}	1

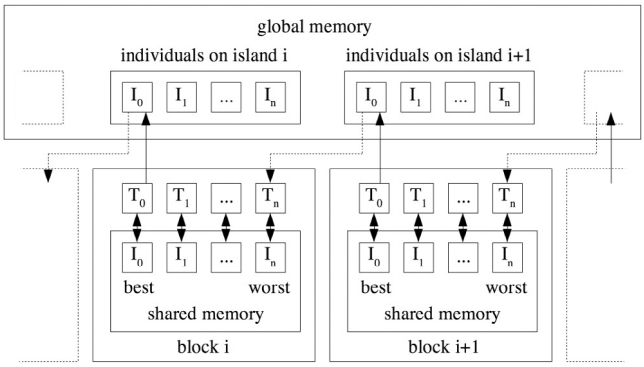

Fig. 13. The single ring migration among islands.

estimated maximum value of objective function E_{max} is set as 10^a, where $a \in N_+$. The value of a is kept increasing from 1 until all individuals' initial objective function values are smaller than the E_{max}. Other experimental relative data are defined in Table 5. The experimental platform is based on Intel Xeon E5640 CPU with 2.67 GHz clock speed. The GPU code implementation is carried out using CUDA 8.0 on the NVIDIA Tesla K40, with 2880 cores at 0.745 GHz and 12 GB GDDR5 of global memory. All programs are written in C, except for the GPU kernels in CUDA C.

5.1. Parameters configuration test of the hybrid parallel GA

As the maximum thread 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 6048 (64×64). Since small size islands with the migration lead to premature convergence while the algorithm with large size islands converges slower [6], 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 [35] and the mutation rate should be much lower than the crossover rate [5]. 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 6. 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.

Due to the influence from the island size, the trend of the probability obtaining adequate solutions and the execution time with different island sizes are illustrated in Fig. 14 and Table 7 respectively. Each value denotes an average result over 100 runs. Regarding the values of objective function got by different settings

Table 6
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

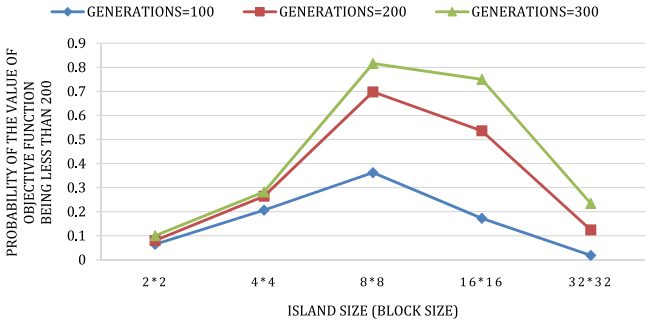

Fig. 14. The trend of the probability obtaining adequate solutions with different island sizes (block sizes) on GPUs.

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 Fig. 14 and Table 7, we could observe a great influence on the solutions' quality by varying the island size 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).

5.2. Performance evaluation test of the hybrid parallel GA

Firstly, we try to compare the solutions obtained from the hybrid parallel GA, the classical GA and the cellular GA. As the roulette wheel selection is the most frequently used selection strategy [43], we take it for the classical GA while the single point crossover is executed with randomly paired individuals. Meanwhile, the mutation operation is kept the same as the hybrid parallel GA. The cellular GA is a popular way to apply the conventional GA in a grid environment and has been implemented a lot to solve combinatorial optimization problems [14,24,31]. In this case, two individuals are selected from a similar neighborhood area as the local asteroid selection in Section 4.4. Then the single point crossover recombines the chromosomes from them to generate a new individual. Finally, this new individual executes the same mutation as other two GAs and replaces the considered individual if its solution is better. 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 classical 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 classical GA on single core CPU, the master–slave GA on four cores CPU. Each of them is generated 100 times respectively.

Table 7
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

Table 8
Solutions' quality comparison.

Generations	Hybrid Parallel GA			Classical 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

Table 9
Execution time comparison (Generations = 100).

Populationsize	Hybrid Parallel GA on GPUs	Cellular GA on GPUs	Classical 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

From the results in Table 8, we discover that the proposed hybrid parallel GA always gains a better performance with the average value, the best value and the variance of the objective function than the classical GA and the cellular GA. Since fine-grained models at the lower level could obtain good population diversity when dealing with high-dimensional variable spaces [20,21] and island models at the upper level converge faster by subpopulations [6], the hybrid parallel GA combines the merits from both. Moreover, the cellular GA overcomes the classical GA as it allows a better exploration of the search space with respect to the decentralized population [31].

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 9. On the opposite, the classical 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 [17] on 4 cores CPU. Afterwards, the SIMD vectorization was executed simultaneously via SSE2 [18]. 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. Therefore, we expect the hybrid parallel GA can achieve further acceleration for more complicated or larger-scale problems by its fully implemented parallelism on the CUDA framework.

5.3. Sensitive analysis test of the EDFFS

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 predictive reactive complete rescheduling approach and the traditional static approach are displayed in Table 10. The iteration number is kept as 100 like the last two tests. The predictive reactive 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, these 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 predictive reactive complete rescheduling approach with the assistance of GPUs 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.

As tardy jobs typically cause penalty costs [30] 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 11 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.

5.4. Convergence trend test of the EDFFS

As a GA converges when most of the population is identical or the diversity is minimal [23], there is no need to execute the algorithm for more generations after the convergence point. For the EDFFS, 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

Table 10
Comparison between the predictive reactive 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	Predictive reactive completerescheduling 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

Table 11
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
10 000	17.58	42.41	175831.12
Variance	78.17	0.61	

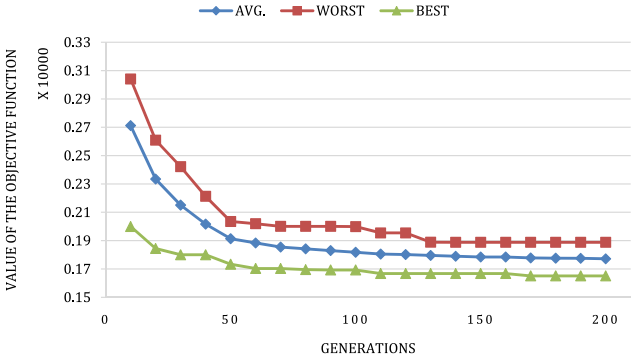

Fig. 15. The convergence trend of small size problem.

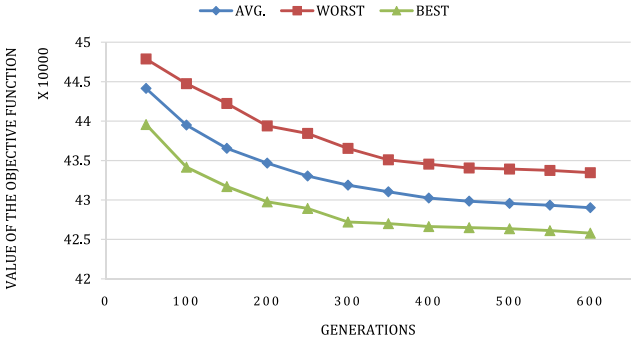

Fig. 16. The convergence trend of medium size problem.

trends of small size, medium size and large size problem instances are described in Figs. 15–17 separately. Each point in these figures displays a value of 30 runs.

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 1.24 s, 223.37 s and 4256.14 s respectively. Therefore,

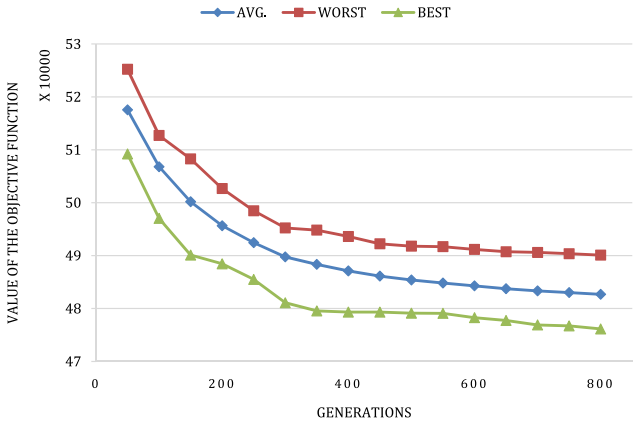

Fig. 17. The convergence trend of large size problem.

to get solutions after the convergence for the small size problem, it takes 6.2 s whereas the medium size and the large size problems need much longer time as 8934.8 s and 255368.4 s. 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.

6. Conclusions

In this paper, 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 predictive reactive complete rescheduling approach was developed. The proposed GA consisted of a fine-grained GA at the lower level and an island GA at the upper level, which was highly consistent with the hierarchy of threads and different types of memory of CUDA framework. In the first test, we configured the parameters of the hybrid parallel GA and obtained a reasonable island size for the tested instance 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 classical 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 predictive reactive complete rescheduling approach was flexible to solve the EDFFS, 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 EDFFS problems. 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.

Acknowledgments

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

References

[1] E. Alba, B. Dorronsoro, Cellular genetic algorithms, in: Operations Research/Computer Science Interfaces, Springer-Verlag, Heidelberg, 2008.
[2] V. Boyer, D. El Baz, Recent advances on GPU computing in operations research, in: Parallel and Distributed Processing Symposium Workshops & PhD Forum, IPDPSW, 2013 IEEE 27th International, IEEE, 2013, pp. 1778–1787.
[3] A. Bruzzone, D. Anghinolfi, M. Paolucci, F. Tonelli, Energy-aware scheduling for improving manufacturing process sustainability: a mathematical model for flexible flow shops, CIRP Annals-Manuf. Technol. 61 (1) (2012) 459–462.
[4] L. Bukata, P. Sucha, A GPU algorithm design for resource constrained project scheduling problem, in: Parallel, Distributed and Network-Based Processing, PDP, 2013 21st Euromicro International Conference, IEEE, 2013, pp. 367–374.
[5] J.A. Cabrera, A. Simon, M. Prado, Optimal synthesis of mechanisms with genetic algorithms, Mech. Mach. Theory 37 (10) (2002) 1165–1177.
[6] E. Cantu-Paz, A survey of parallel genetic algorithms, Calculateurs paralleles, reseaux et systems repartis, 10 (2) (1998) 141–171.
[7] M. Czapinski, S. Barnes, Tabu search with two approaches to parallel flowshop evaluation on cuda platform, J. Parallel Distrib. Comput. 71 (6) (2011) 802–811.
[8] A. Dabah, A. Bendjoudi, D. El Baz, A. AitZai, GPU-based two level parallel B & B for the blocking job shop scheduling problem, in: Parallel and Distributed Processing Symposium Workshops, 2016 IEEE International, IEEE, 2016, pp. 747–755.
[9] 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, Robot. Comput.-Integr. Manuf. 29 (5) (2013) 418–429.
[10] EIA, International energy outlook, 2009. May 2009 2.
[11] EIA, Annual energy review 2009. Report no. DOE/EIA0384(2009), August 2010.
[12] K. Fang, N. Uhan, F. Zhao, J.W. Sutherland, A new shop scheduling approach in support of sustainable manufacturing, Glocalized Solutions for Sustainability in Manufacturing, Springer, 2011, pp. 305–310.
[13] J.N.D. Gupta, Two-stage, hybrid flow shop scheduling problem, J. Oper. Res. Soc. 39 (4) (1988) 359–364.
[14] M. Guzek, J.E. Pecero, B. Dorronsoro, P. Bouvry, S.U. Khan, A cellular genetic algorithm for scheduling applications and energy-aware communication optimization, in: High Performance Computing and Simulation, HPCS, 2010 International Conference, IEEE, 2010, pp. 241–248.
[15] J.H. Holland, Genetic algorithms, Sci. Am. 267 (1) (1992) 66–73.
[16] https://developer.nvidia.com/cuda-toolkit.
[17] http://www.openmp.org/.
[18] https://en.wikipedia.org/wiki/Streaming_SIMD_Extensions.
[19] C.S. Huang, Y.C. Huang, P.J. Lai, Modified genetic algorithms for solving fuzzy flow shop scheduling problems and their implementation with CUDA, Expert Syst. Appl. 39 (5) (2012) 4999–5005.
[20] U. Kohlmorgen, H. Schmeck, K. Haase, Experiences with fine-grained parallel genetic algorithms, Ann. Oper. Res. 90 (1999) 203–219.
[21] J.M. Li, X.J. Wang, R.S. He, Z.X. Chi, An efficient fine-grained parallel genetic algorithm based on GPU-accelerated, in: Network and Parallel Computing Workshops, 2007 NPC workshops. IFIP International Conference, IEEE, 2007, pp. 855–862.
[22] Y. Liu, H. Dong, N. Lohse, S. Petrovic, N. Gindy, An investigation into minimising total energy consumption and total weighted tardiness in job shops, J. Cleaner Prod. 65 (2014) 87–96.
[23] S.J. Louis, G.J. Rawlins, Predicting convergence time for genetic algorithms, Found. Genetic Algorithms 2 (1993) 141–161.
[24] G. Luque, E. Alba, B. Dorronsoro, An asynchronous parallel implementation of a cellular genetic algorithm for combinatorial optimization, in: Proceedings of the International Genetic and Evolutionary Computation Conference, GECCO, ACM, 2009, pp. 1395–1402.
[25] N. Melab, I. Chakroun, M. Mezmaz, D. Tuyttens, A gpu-accelerated branch-and-bound algorithm for the flow-shop scheduling problem, in: 65 Cluster Computing, CLUSTER, 2012 IEEE International Conference, IEEE, 2012, pp. 10–17.
[26] G. Mouzon, M.B. Yildirim, A framework to minimise total energy consumption and total tardiness on a single machine, Int. J. Sustain. Eng. 1 (2) (2008) 105–116.
[27] J.M. Nilakantan, G.Q. Huang, S. Ponnambalam, An investigation on minimizing cycle time and total energy consumption in robotic assembly line systems, J. Cleaner Prod. 90 (2015) 311–325.
[28] D. Ouelhadj, S. Petrovic, A survey of dynamic scheduling in manufacturing systems, J. Sched. 12 (4) (2009) 417.
[29] C. Pach, T. Berger, Y. Sallez, T. Bonte, E. Adam, D. Trentesaux, Reactive and energy-aware scheduling of flexible manufacturing systems using potential fields, Comput. Ind. 65 (3) (2014) 434–448.
[30] I.C. Parmee, Adaptive computing in design and manufacture, Eng. Optim. 41 (9) (2009) 811–812.
[31] F. Pinel, B. Dorronsoro, P. Bouvry, Solving very large instances of the scheduling of independent tasks problem on the GPU, J. Parallel Distrib. Comput. 73 (1) (2013) 101–110.
[32] B. Plazolles, D. El Baz, M. Spel, V. Rivola, P. Gegout, SMID Monte-Carlo numerical simulations accelerated on GPU and Xeon Phi, Int. J. Parallel Program. (2017) 1–23.
[33] P. Pospichal, J. Jaros, J. Schwarz, Parallel genetic algorithm on the CUDA architecture, in: European Conference on the Applications of Evolutionary Computation, Springer, 2010, pp. 442–451.
[34] J. Sanders, E. Kandrot, CUDA by Example: An Introduction to General-Purpose GPU Programming, Addison-Wesley Professional, 2010.
[35] J.D. Schaffer, R.A. Caruana, L.J. Eshelman, R. Das, A study of control parameters affecting online performance of genetic algorithms for function optimization, in: Proceedings of the Third International Conference on Genetic Algorithms, Morgan Kaufmann Publishers Inc., 1989, pp. 51–60.
[36] D. Tang, M. Dai, M.A. Salido, A. Giret, Energy-efficient dynamic scheduling for a flexible flow shop using an improved particle swarm optimization, Comput. Ind. 81 (2016) 82–95.
[37] S. Tsutsui, N. Fujimoto, Solving quadratic assignment problems by genetic algorithms with gpu computation: a case study, in: Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers, ACM, 2009, pp. 2523–2530.
[38] F. Werner, Genetic algorithms for shop scheduling problems: a survey, Preprint 11 (2011) 31.
[39] F. Xu, W. Weng, S. Fujimura, Energy-efficient scheduling for flexible flow shops by using MIP, in: IIE Annual Conference. Proceedings, Institute of Industrial and Systems Engineers, IISE, 2014, p. 1040.
[40] 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.
[41] T. Zajıcek, P. Sucha, Accelerating a flow shop scheduling algorithm on the GPU in: Workshop on Models and Algorithms for Planning and Scheduling Problems, MAPSP, 2011, pp. 143–144.
[42] L. Zhang, X. Li, L. Gao, G. Zhang, Dynamic rescheduling in FMS that is simultaneously considering energy consumption and schedule efficiency, Int. J. Adv. Manuf. Technol. 87 (5–8) (2016) 1387–1399.
[43] J. Zhong, X. Hu, J. Zhang, M. Gu, Comparison of performance between different selection strategies on simple genetic algorithms, in: Computational Intelligence for Modeling, Control and Automation, 2005 International Conference on Intelligent Agents, Web Technologies and Internet Commerce, vol. 2, IEEE, 2005, pp. 1115–1121.

Jia Luo received the bachelor's degree from Shanghai University, China, in 2011 and the master's degree from Waseda University, Japan, in 2015. She is presently pursuing a Ph.D. degree at the Laboratory of Analysis and Systems Architecture (LAAS-CNRS), Toulouse, France. Her Ph.D. topic is about scheduling, parallel algorithms and GPU computing.

Shigeru Fujimura is a Professor at Graduate School of Information, Production, and Systems, Waseda University, Japan, since April 2003. He received the B.E. and the M.E. degrees from Waseda University in 1983 and 1985, respectively. He also received the Dr. Eng. from Waseda University in 1995. He joined Yokogawa Electric Corporation in 1985 and worked there until 2003. His research interests are production management, production scheduling, intelligent interface agent, object oriented modeling, and software engineering.

Didier El Baz received the Engineer degree in Electrical Engineering and Computer Science from National Institute of Applied Sciences in Toulouse, France (Institut National des Sciences Appliquées, INSA) in 1981 and the Doctor Engineer degree in Control Theory from INSA Toulouse in January 1984. Dr. El Baz was a visiting scientist in the Laboratory for Information and Decision Systems, MIT Cambridge Massachusetts, USA, in 1984. He is the founder and head of the Distributed Computing and Asynchronism team at the Laboratory of Analysis and Systems Architecture (LAAS-CNRS). Dr. El Baz is the author of 40 papers in referred international journals and 70 papers in referred international conferences. His fields of interest are in optimization, parallel and distributed computing.

Bastien Plazolles received the Ph.D. degree in High Performance Computing from University Paul Sabatier in 2017. His Ph.D. research was focused on using computing accelerators (GPUs, Intel Xeon Phi) to address real world problems such as probabilistic determination of fallout area of stratospheric balloon and satellite atmospheric reentry analysis. Now he holds a postdoctoral position at CNRS, in the Space Geodesy Team (GET/GRGS Toulouse France) working on the development of parallel seismic inversion tools.