Hybrid multi-core CPU and GPU-based B&B approaches for the blocking job shop scheduling problem

Adel Dabah, Ahcène Bendjoudi, Abdelhakim Aitzai, Didier El Baz, Nadia Nouali Taboudjemat

Hybrid Multi-core CPU and GPU-based B&B Approaches for the Blocking Job Shop Scheduling Problem.

Adel Dabah[a,b], Ahcène Bendjoudi[a], Abdelhakim AitZai[b], Didier El-Baz[c], Nadia Nouali Taboudjemat[a]

{adabah,abendjoudi,nnouali}@cerist.dz; {h.aitzai,adel.dabah}@usthb.dz; elbaz@laas.fr

[a] CERIST Research Center, Algiers, Algeria.
[b] University of Sciences and Technology Houari Boumedienne (USTHB) Algiers, Algeria.
[c] LAAS-CNRS, Université de Toulouse, CNRS, Toulouse, France.

Abstract

The Branch and Bound algorithm (B&B) is a well known method for solving optimally Combinatorial Optimization Problems (COPs). This method is based on intelligent enumeration of all feasible solutions which reduce considerably the search space. Nevertheless, it remains inefficient when using the sequential approach to deal with large problem instances due to its huge resolutions time. However, the execution time can be reduced considerably by using parallel computing architectures. With the huge evolution of the multi-cores CPUs and GPUs, It is quite hard to design schemes that efficiently exploit the different hardware architectures simultaneously. As a result, most of the existing works focus on exploiting one hardware architecture at a time. In this paper, we propose five parallel approaches to accelerate the B&B execution time using Multi and Many-core systems at the same time. Our goal is to solve optimally the Blocking Job Shop Scheduling problem (BJSS) which is one of the hardest scheduling problem. The first proposed approach is a *multi-search* parallelization based on Master/Worker paradigm, exploiting the Multi-Core CPU-processors. The second and the third approaches represent a GPU based parallelization schemes having different level of parallelism and GPU occupancy. The forth and fifth approaches represent a hybridization between the Muli-core approach and the GPU based parallelization approaches. The goal of this hybridization is to benefit from the power of both the CPU-cores and the GPU at the same time. This hybridization is based on concurrent kernels execution provided by Nvidia Multi process Service (MPS) which allows multiple host processes (Master and workers) to use at the same time the GPU to launch their kernels in order to accelerate the bounding of one or several nodes at a time. Experiments using the well known Taillard instances confirm the efficiency of our proposals and show a relative speedup of 160x as compared to an optimized sequential B&B algorithm.

Keywords: Job shop scheduling; Parallel Branch-and-Bound algorithm; Multi-core and Many-core computing; Nvidia MPS.

1. Introduction

The job shop scheduling problem (JSSP) consists in scheduling a set of n jobs on a set of m machines. Each job has its own sequence of crossing on machines. The execution of a job on a machine is called operation and each one uses the machine for an uninterrupted processing time. The classical JSSP assumes an infinite storage space between machines which is not realistic. The Blocking Job Shop Scheduling problem (BJSS) is a version of the classical JSSP with no storage space between machines, where a job has to wait on the current machine until the next one becomes available. Our goal is to minimize the completion time of all jobs (*Makespan*). The classical JSSP is known to be NP-hard in the strong sense and its search space is equal to $(n!)^m$ [13]. The BJSS problem wich is an extension of the JSSP, appears to be even more difficult to solve [17]. This problem has several application areas such as manufacturing systems with no storage space, train scheduling, hospital resource scheduling, *etc*.

The B&B algorithm is a well known method to solve optimally the COPs. It is based on intelligent enumeration of all feasible solutions, which reduce considerably the search space. Nevertheless, its sequential version requires a huge execution time to solve small instances and it remains inefficient when dealing with large or real-world instances. Therefore, using parallel comput-

ing architectures such as the Multi-core CPUs and the GPUs is unavoidable to improve its running time of this method.

Nowadays, most of processors are based on Multi-core architectures. They are composed of two or more independent central processing units (cores) that can run multiple instructions at the same time. In addition to the power provided by multi-core architectures, Graphics Processing Units (GPUs) have emerged as a new popular support for massively parallel computing. GPUs are many-core co-processor devices that provide a hierarchy of memories having different sizes and access latencies and providing a highly multi-threaded environment where the threads are scheduled and executed as warps (group of threads) using the SIMT model [27].

With the huge evolution of the Multi-core CPUs and GPUs, it is quite hard to design schemes that efficiently exploit different hardware architectures simultaneously. As a result, most of the proposed parallel B&B algorithms in the literature [8, 15, 2, 7, 4] exploit only the CPU-core or only the GPU which may lead to an under-utilization of these resources and a loss of a significant computing power. The major contribution of this paper is the new parallelization schemes that exploit and combine different parallelization levels of the B&B algorithm using Multi and Many-core systems simultaneously. The proposed schemes are based on the Nvidia Multi Process Service (MPS) that allows us to increase the GPU occupation over time. Therefore, achieving a high relative speedup for large instances.

The first approach (*Multi-core B&B*) is a tree based parallelization, exploiting the Multi-core CPU-processors available in all recent PCs. The proposed approach is based on Master/Worker paradigm where the workers independently explore the branches sent by the master. The performance of this approach depends on the number of used CPU-cores.

The second and the third approaches are a GPU based parallel B&B schemes. The second approach: *Parallel Evaluation of the Bound* (*PEB*) is a node-based parallelization exploiting the idea that the evaluation (bounding) of each node of the search tree can be calculated in parallel on the GPU using several GPU-threads. The proposed scheme here is very important to reduce the B&B execution time since the bounding phase consumes over 85% of the whole execution time. Therefore, at each iteration of this scheme, one node is evaluated in parallel on the GPU by several threads organized in one GPU block. Experiments validate our idea and show the benefit of accelerating the bounding process on GPU by achieving a speedup up to 18x. The third approach (*Parallel Evaluation of Several Bounds* (*PESB*)) represents a generalization of the second approach (PEB) obtained by sending at each iteration a pool of nodes to the GPU for evaluation instead of one node at a time in the PEB approach. The number of nodes sent varies according to the size of instances. This approach allows us to achieve a relative speedup of 66x.

The drawback of the previous approaches is the under-utilization of either CPU or GPU resources which represents a wast of significant computing power. To increase the GPU occupation and benefit from both the multi-core CPU and the GPU at the same time, we propose a hybridization between the Multi-core CPU approach and the two GPU based approaches. This Hybridization is based on the concurrent kernels execution provided by Nvidia MPS *i.e.* Multiple host processes (master and workers) can execute simultaneously their kernels on the GPU. The forth approach, *H-PEB* represents a hybridization between the Multi-core approach and the PEB GPU based approach which mean that several host processes (Master and workers) can use the GPU simultaneously to accelerate the bounding of each node on the GPU according to the PEB scheme. Increasing the GPU occupation by this approach allows us to reach a relative speedup of 93x as compared to an optimized sequential B&B. In the same way, the fifth parallel approach *H-PESB* represents a hybridization between the Multi-core approach and the PESB GPU based approach. Therefore, all host processes (Master and workers) can use simultaneously the GPU to evaluate several nodes on the GPU instead of one node in the H-PEB approach. The obtained results for this last approach show a relative speedup of 160x as compared to an optimized sequential B&B approach.

The remainder of this paper is organized as follows: Section 2 describes the blocking job shop scheduling problem, the *alternative graph model* and related work. Section 3 contains a brief description of the sequential B&B algorithm and its components. Section 4 presents the proposed parallelization approaches of the B&B algorithm. Section 5 discusses computational results. Finally conclusions and perspectives are presented in Section 6.

2. Blocking job shop scheduling problem

2.1. Problem Formulation

The classical JSSP can be defined by a set J of n jobs $(J1, ..., Jn)$ to be processed on a set M of m machines $(M1, ..., Mm)$. Each machine can process at most one job at a given time. The execution of a job on a machine is called operation. We note by O the set of all operations $(o_1, ..., o_{n*m})$. Each operation o_i needs to use a

machine $M(i)$ for an uninterrupted duration called processing time p_i. Each job has its own sequence of crossing on machines which creates precedence constraints between consecutive operations of the same job. A solution (schedule) for this problem consists to assign a starting and finishing times t_i and c_i for each operation o_i ($i = 1, ..., n * m$); while satisfying all constraints. Our goal is to minimize the *Makespan* (*Cmax*). The JSSP assumes an unlimited intermediate buffer capacity between consecutive operations of a job which is impossible in real manufacturing. The BJSS is a version of the classical JSSP with no intermediate buffers, where a job has to wait on the current machine until the next machine becomes available for processing. This problem can be modelled as an alternative graph representation introduced by Mascis *et al.* [1] which is a generalization of the disjunctive graph of Roy and Sussman [4]. This model can be defined as $G = (N, F, A)$. N represents a set of nodes (operations) with two additional dummy nodes (start and finish) modelling the start and the finishing of the schedule. F represents a set of fixed arcs imposed by precedence constraints between consecutive operations of the same job and f_{qp} is the length of arc $(q, p) \in F$. Finally, A is a set of alternative pairs $((i, j), (h, k))$ representing the processing order for concurrent operations on the same machine and a_{ij} is the length of alternative arc (i, j). Each alternative pair contains two alternative arcs and each one of them expresses the fact that one operation must be completed on the target machine before starting the processing of the other operation. Moreover, subsection *2.1.1.* describes the alternative arcs generation. A selection S_1 is a set of alternative arcs obtained from A by choosing at most one alternative arc from each alternative pair, and $G(S_1) = (N, F \cup S_1)$ represents the graph representation of the selection. We note that a selection S_1 is feasible if there is no positive length cycle in $G(S_1)$ and the evaluation (*Makespan*) of feasible selection S_1 is the longest path in $G(S_1)$. We say that S_1 is a complete selection if exactly one arc is chosen from each pair, therefore $|A| = |S_1|$. We define a schedule (solution of the problem) as a complete feasible selection. Finally, given a feasible selection S_1, let $l(i, j)$ be the length of the longest path from operation i to j in $G(S_1)$. We call the last operation of each job (example o_r) an ideal operation because the machine becomes immediately available after the end of its processing time p_r. If o_i is a blocking operation, we denote by $\sigma(i)$ the operation immediately following o_i in the same job.

2.2. Forming the graph of all possibilities (Search graph)

The search graph represents the graph of all possibilities which is used by the B&B algorithm as search tree root. This graph is obtained by generating all alternative pairs, knowing that each alternative pair represents the processing order between every two concurrent operations. Therefore, if a machine has four concurrent operations, we need six alternative pairs to show all the possibilities on how these operations are executed on the machine. The alternative pair between every two concurrent operations is generated as follow.

2.2.1. Alternative pair generation

Let us consider two blocking operations o_i, o_j and one ideal operation o_r, where $M(i) = M(j) = M(r)$. Since the three operations cannot be executed at the same time, we associate them with pairs of alternative arcs.

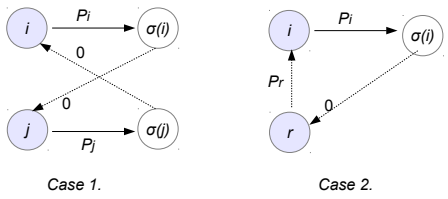

Figure 1: Alternative pairs between blocking and ideal operations.

Case 1: the alternative pair between operations o_i and o_j (Fig. 1): The first alternative arc $(\sigma(i), j)$ having length 0 represents the situation where o_i is processed before o_j. Since o_i is a blocking operation, $M(i)$ can begin the processing of o_j only after the starting time of $\sigma(i)$(when o_i leaves $M(i)$). The same method is followed for the other alternative arc$(\sigma(j), i)$ since o_j is a blocking operation.

Case2: the alternative pair between operations o_i and o_r (Fig. 1): It is the same process as in the first case for the alternative arc $(\sigma(i), r)$ since o_i is a blocking operation. The other alternative arc depends on the fact that o_r is an ideal operation therefore, we add the alternative arc(r, i) with length p_r.

2.3. Example

Table 1 represents a BJSS instance with two products (jobs) and three machines. The first product ($J1$) has 5

Table 1: BJSS instance with two jobs and three machines.

job	sequence	processing times
$J1$	$M1, M2, M3$	5, 3, 8
$J2$	$M2, M1, M3$	8, 2, 7

min processing time on machine $M1$, 3 min on $M2$ and 8 min on machine $M3$. The second product ($J2$) has 8 min processing time on machine $M2$, 2 min on $M1$ and 7 min on machine $M3$.

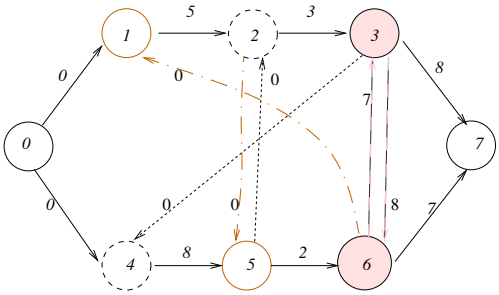

Figure 2: Alternative graph for BJSSP instance of table 1.

Figure 2 represents an alternative graph of the BJSS instance in Table 1. This graph has three alternative pairs, two between blocking operations and one between ideal operations. Both operations 2 and 4 need the same machine $M2$ and since $M2$ can not process both operations at the same time, we associate them with an alternative pair. Since operations 2 and 4 are blocking operations the first alternative arc (3, 4) represents the choice where operation 2 must be finished before the beginning of operation 4. Its mate, arc (2, 5) represents the choice whereby operation 4 must be finished before the beginning of operation 2. We use the same process to generate the alternative pair ((2,5), (6,1)) between operations 1 and 5. The alternative pair between operations 3 and 6 is ((3, 6), (6, 3)) because both operations 3 and 6 are ideal.

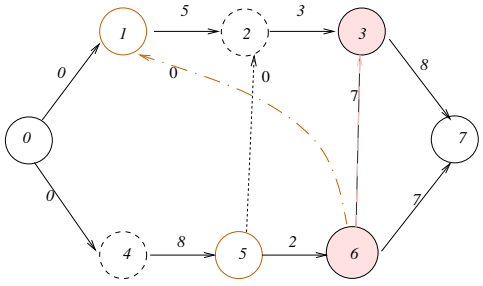

Figure 3: Schedule for BJSS Problem in table 1 whit $Cmax$=26.

Figure 3 represents a feasible schedule (solution) for the BJSS instance in Table 1, obtained by choosing one arc from each pair in the alternative graph of Figure 2. The *Makespan* ($Cmax$ = 26) of this schedule is the longest path in the obtained graph. The Gantt chart in

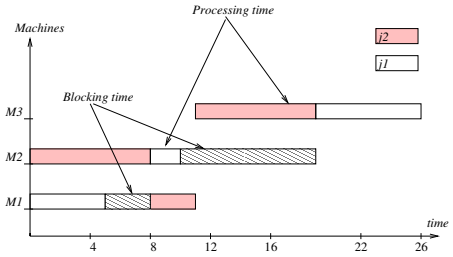

Figure 4: Gantt chart of the schedule in figure 3.

Figure 4 represents both the processing and blocking times of the solution of Figure 3. For example, after the end of its processing time the job $J1$ blocks the machine $M1$ until machine $M2$ becomes available for processing $J1$.

2.4. Related works

The BJSS problem can be solved using either exact or approximate methods. In the literature, several heuristics and metaheuristics has been proposed to solve the BJSS problem as in [3,8,10,9,5,6]. In addition, heuristics as in [30, 29] and studies on meta-heuristics based on multi-agent systems has proven theirs efficiency in solving other optimization problems, among them [31, 32, 33].

In this paper, we focus on accelerating the B&B algorithm using multi-cores CPUs and GPUs simultaneously. Most of B&B methods for the job shop problem are based on the *one machine scheduling problem* proposed by Carlier et *al*. [10].

Only a few authors tried to solve optimally the BJSS problem, the most effective one which is the base of our sequential version is proposed by Mascis *et al*. [17]. The authors formulate the problem by means of an alternative graph model which is a generalization of the disjunctive graph of Roy and Sussman [22]. Based on this model, they solve optimally the 10×10 benchmark instances of this problem.

Ait Zai *et al*. [1], proposed an original B&B method based on graph theory to solve the BJSS problem. The idea of its branching scheme relies on the implicit enumeration of all possible combinations on a given machine. The authors gave solutions for local instances only.

The B&B algorithms are not efficient when dealing with large problem instances, therefore computing accelerators like GPUs are required. Several authors have proposed to accelerate the B&B method using GPUs. This work represents an extension of our work in [6], in which we added three parallel approaches in order to fully occupy the GPU. This extension allows us to increase the performance of our approaches by 74% to reach a speedup of 164x compared to 90x in [5]. In [6], we propose a preliminary version of the B&B algorithm dedicated to treat small instances (less than 50 jobs), the concept of the parallelization is kept for the two GPU approaches. However, to treat large instances we had to redesign the code and the GPU memory management to deal with such huge generated data. The performance obtained in this paper boost the speedup to reach a 164X compared to 60x in [6].

Chakroun *et al.* [8] and [15], take the classical approach of sending nodes to be evaluated on GPU to solve the Flow shop scheduling problem since this step takes more than 98% of the global execution time. Therefore, each GPU thread supports the evaluation of a single node of the search tree. In [2, 9] the authors extend the approach below to exploit the Multi-core CPU processors in the generalization of sub-problems that are going to be sent to the GPU for evaluation.

In [7], Alami *et al.* proposed a CPU-GPU based B&B applied to the knapsack problem. In the proposed parallelization scheme the branching and bounding can be done either on the CPU or the GPU according to the size of the search tree. This approach uses less CPU-GPU communication and better management of data-structures in GPU memory.

In [4], Carneiro *et al*. apply the B&B to the traveling salesman problem where a pool of nodes is sent to the GPU for evaluation. Each GPU-thread applies the branching and bounding operators to a single node and builds its own local tree. The resulting nodes are moved back to the CPU where the promising nodes are inserted into the tree.

In [18], the authors proposed multi-core and many-core parallel B&B for big optimization problems. The authors propose two B&B implementations, the first one focuses on exploiting the traditional multi-core CPU processors while the seconds one is dedicated for Intel Xeon Phi coprocessors considering both native and ofload modes. The reported results show that the many-core approaches (native and offload) are twice faster as compared with the multi-core approach.

In [23], the authors proposed a parallel B&B algorithm exploiting the advantage of instance specific computing on Field Programmable Gate Array (FPGA) which has proven to be highly efficient in term of area, energy consumption, and performance. In addition, the proposed parallelization is based on work stealing strategies to ensure dynamic load-balancing between the parallel threads. The authors approach was applied on the reconstruction of corrupted AES keys problem, the reported results show an overall speedup of 47x.

In [25], Trong and Bilel proposed parallel B&B for large scale heterogeneous distributed platforms with several distributed CPUs and GPUs. The proposed approaches address the critical issue of how to map B&B workload with the different levels of parallelism exposed by the target compute platform. The reported results show the significant impact of the adaptive load balancing among the heterogeneous compute resources on the performance.

Most of the previously cited works focus on exploiting the GPU part and ignoring the available CPU-cores. Also, most authors use the *parallel evaluation of bounds* model in which each GPU thread supports the bounding of a single search tree node. This represents a lot of calculation and lot of resources for a single thread which may limit the performance. For this reason, we propose in this paper new parallelization schemes, that exploit and combine different parallelization levels to accelerate the B&B execution time using Nvidia MPS.

3. The Branch and Bound algorithm for BJSS

The B&B algorithms make an intelligent enumeration of all feasible solutions. They are mainly characterized by two operators: branching and bounding. The branching is a recursive process, which consists in replacing the search space of a given problem by a set of smaller sub-problems. The bounding operator contains two bounds: the Lower Bound (LB) and the Upper Bound (UB). The LB represents an estimation of the lowest evaluation of all feasible solutions in the considered sub-problem, while the UB represents the upper limitation of the evaluation of each search tree node. Any solution of the problem can be considered as initial value for the UB which is used by other operators and it is updated as soon as a new better solution is found by the B&B algorithm. The B&B algorithm is based essentially on the bounds to make an intelligent enumeration of the search space. The elimination operator uses the bounds (LB and UB) to eliminate the sub-problems that cannot lead to improve the current best solution found for the problem.

The most effective B&B algorithms, for the JSSP, are based on the disjunctive graph model [3]. Our B&B is

based on the adaptation of this approach to the blocking case (alternative graph) [17]. Our method which is based on [17] consists in fixing an order (precedence order) between every two concurrent operations, which leads to fix the corresponding alternative pair (from A) and a set of fixed arcs represents a selection.

Table 2: The description of the symbols used in our B&B algorithm.

Symbol	Description
UB, LB	Upper Bound, Lower Bound.
$LIST$	A set of nodes (sub-problems).
$s*$	The optimal solution.
R_i	The *ith* successor of node R.
$LB(R_i)$	Lower bound of node R_i.

Algorithm 1 and Table 2 describes the used B&B algorithm.

Algorithm 1 Pseudo-code of the sequential B&B algorithm

```
LIST ← {original problem};
UB ← ∞;
while LIST != ∅ do
  R ← LIST (Choose a Node R from LIST);
  Generate successors R_i from R | (i = 1, ..., n);
  for Each successour R_i do
    if LB(R_i) < UB then
      if R_i represents one solution then
        UB = LB(R_i);
        s* = solution in R_i;
      else
        LIST = LIST ∪ R_i;
      end if
    end if
  end for
end while
return s*
```

In the following, we describe the different operators of the used B&B algorithm for the BJSS problem.

3.1. Branching

The B&B algorithm can be represented by a search tree. The tree is rooted by the original problem *i.e.* no alternative pairs are fixed ($|S_0|$=0). A search tree node R is characterized by (S_R, A_R) and represented by the graph G(S_R)=$(N, F \cup S_R)$. S_R denotes the set of fixed alternative arcs and A_R represents a set of unselected alternative pairs in this node. The branching creates two immediate successors ($R1$, $R2$) of R by fixing an alternative pair $((i, j), (h, k)) \in A_R$ that has a direct impact on the longest path in the graph. The node $R1$ (resp. $R2$) is characterized by $S_{R1} = S_R \cup (i, j)$ (resp. $S_{R2} = S_R \cup (h, k)$) and $A_{Ri} = A_R - \{((i, j), (h, k))\}$. Each successor represents the sub-search space related to the fixed alternative arc. After this, each successor is handled recursively in the same way until we find a complete selection or eliminate the sub-problem and prune the tree if the lower bound value is bigger than the upper bound. Finally, the exploration strategy represents the way the search tree is explored in order to find the solution with minimum makespan (optimal solution). Several exploration strategies exist in the literature such-as, best first exploration, worst first exploration, and breadth first exploration. The exploration strategy used in our case consists to choose the after a branching process the node with the biggest Makespan (worst first). For the BJSS problem, and as compared with best first strategy, the worst first exploration strategy has more chance to reach leaf nodes (solutions) therefore more chance to improve the UB and eliminate a large number of branches.

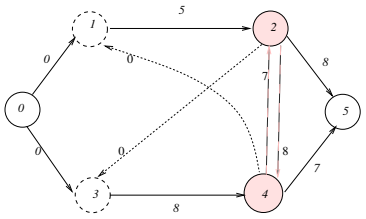

Figure 5: Alternative graph for BJSS instance with two jobs and two machines.

Figure 5 represents the alternative graph of a BJSS instance with 2 jobs and 2 machines. The Figure shows also the existence of two alternative pairs: the first pair ((2, 3), (4, 1)) is between operations 1 and 3 and the second pair ((2, 4), (4, 2)) is between operations 2 and 4.

Figure 6 represents the search tree of the BJSS instance in Figure 5 which contains two alternative pairs. At each level, one pairs is fixed which generate two sub-problems. This process is repeated until a complete feasible selection is obtained or infeasibility is detected.

Figure 7 represents the alternative graph of the optimal solution obtained from the search tree in Figure 6.

3.2. Evaluation (Bounding)

Any solution of the problem can be considered as an initial value for the Upper Bound (UB) which is updated as soon as a new better solution is found. In our case, the UB=+∞.

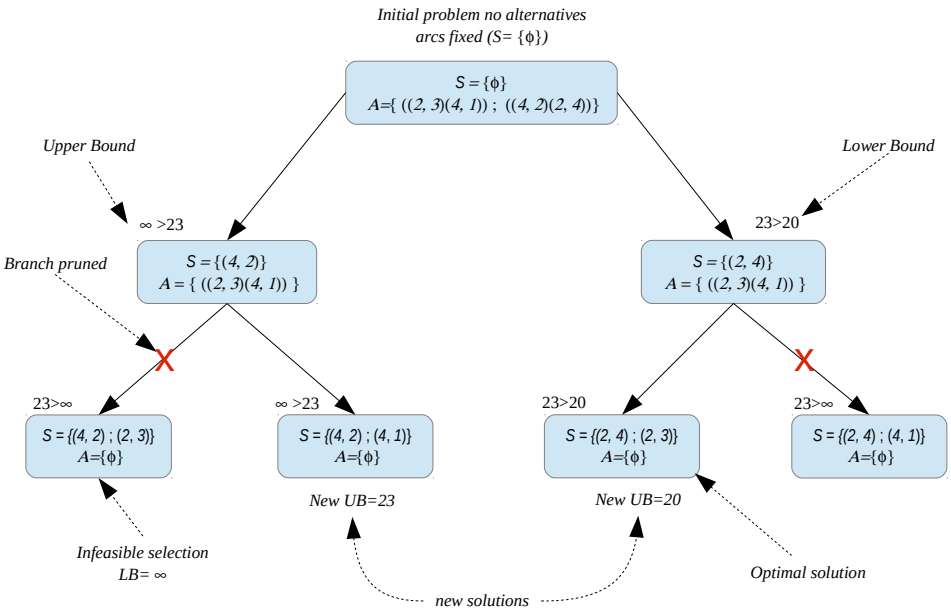

Figure 6: Search tree for the BJSS instance in Figure 5.

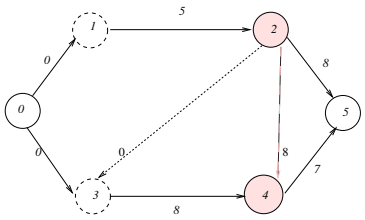

Figure 7: Alternative graph of the optimal solution (*Cmax=20*).

After the branching operation, the bounding process consists to calculate the Lower Bound (LB) for each sub problem. The LB used in our case is the one used by Carlier *et al.* [10] to solve optimally the classical JSSP, it is based on the *one machine scheduling problem*. Therefore, we calculate the lower bound for each machine l using cuncurent operations on it:

$$LB_l = \text{Min } r_i + \Sigma p_i + \text{Min } q_i \;/\; M(i) = l, l = \{1, .., m\}.$$

The LB for the subproblem is equal to the maximum value of LB_l.

$$\text{LB= Max } \{LB_l\} /, l = \{1, .., m\}.$$

To do a link with the alternative graph model, each search tree node represents an alternative graph. This lower bound can not be calculated without the new *Head* and *Tail* values for each operation affected by the branching process. $(H_i = l(0, i), T_i = l(i, n * m) \;/\; (i = 1, ..., n * m)$ The process of adjusting the *Head* and *Tail* values is very expensive and consumes 70% of global execution time. This process is done sequentially for all operations affected by the change made and can be repeated several times for the same operation if there are multiple paths leading to this operation.

3.2.1. Immediate selection

The immediate selection represents several techniques which allows to accelerate the B&B algorithm by reducing the number of branching necessary to obtain the optimal solution. This process is done sequentially and costs 18% of the global processing time since there is a large number of alternative pairs (99000 for big instances). This process uses also the head and tail values computed in the bounding process. Given a search tree node (sub-problem) R with a feasible selection S_R and a set of unselected pairs A_R. For each unselected pair $((i, j), (h, k)) \in A_R$: if $l(0, h) + a_{hk} + l(k, n) \geq UB$ then S_R=$S_R \cup (i, j)$. This rule expresses the fact that adding the arc (k, h) (resp. (i, j)) to S_R will produce a sub-problem with a lower bound greater than the upper bound. Consequently the arc (i, j) (resp. (h, k)) is added to S_R. If both alternative arcs (i, j) and (h, k) do not satisfy the condition of $LB < UB$, we prune the branch by eliminating node R, since this latter cannot contain a

solution that can improve the best solution found by the B&B algorithm.

The complexity of the evaluation process depends on the number of operations ($n \times m$) in the treated instance. Therefore, the evaluation time increases by increasing the size of the instances. The implementation of the evaluation process, requires six data structures. The matrix *Succ* ($(n * m) \times n$) contains the successors of each operation, *i.e.* row i represents the successors of operation o_i. Similarly, the matrix *Pred* ($(n * m) \times n$) contains the predecessors of each operation. Vector S contains all selected and unselected pairs. Vector H (resp. T) contains the *Head* (resp. *Tail*) of each operation. The element $H[i]$=$l(0, i)$ represents the longest path from o_0 to o_i. The same $T[i]$=$l(i, n * m)$ the longest path from o_i to the last operation in the graph o_{n*m}.

4. The proposed parallelization approaches for the B&B algorithm

The fact that each node of the B&B search-tree can be explored independently amplifies the parallelization of this algorithm. The only global information in the algorithm is the value of the upper bound.

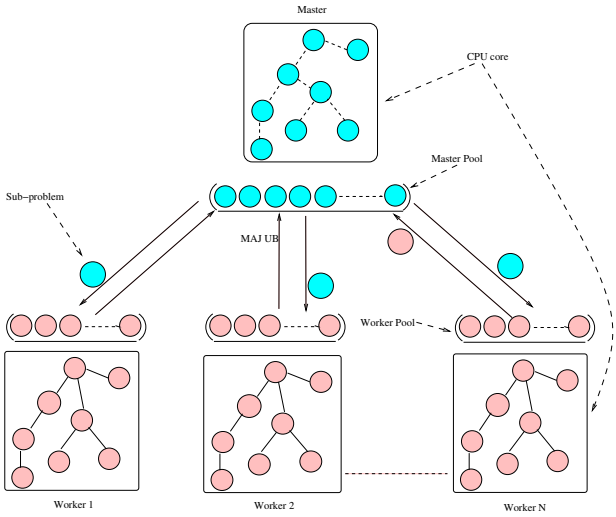

Figure 8: Global architecture of the proposed Master/Worker parallelization.

The algorithm parallelization may depend on the architecture of the processing machine, synchronization, granularity of tasks and communication between different processes. There are several classes of the B&B parallelization. For more details the reader may refer to [12]. In the following, we present our proposed parallel B&B approaches for the BJSS problem.

4.1. Multi-search parallel B&B on Multi-core CPU

In this section, we describe the proposed parallel B&B algorithm exploiting the CPU-cores available in all recent computers. The proposed approach (see Figure 8) is based on the master/worker paradigm. A work pool represents a set of active sub-problems. There is a unique master work pool owned and managed only by the master process which contains the sequential search tree and several local work pools, empty initially, owned by the different workers. Both of the master and workers work pools are managed in the same way. The exploration of the search-tree is done simultaneously by the master and workers since each one of them has its own B&B algorithm and its own work pool. The results given by a worker can influence others. Therefore, our approach can be seen as a multi-search parallelization in which the goal is to accelerate the exploration of master search tree stored in the master pool. We call blocked worker a worker with an empty work pool waiting for a sub-problem to explore. The master process initializes the search by creating the root node, launches its own B&B algorithm which generates a set of active sub-problems stored in the master work pool. After that, the master wakes up the blocked workers by sending them sub-problems from master work pool. After that, each worker launches its own B&B algorithm. During the search, the work pools evolve continuously and when they become empty, the corresponding workers send a request to the master and wait for sub-problems. When the master receives a request, it satisfies the request if the global pool is not empty. When the global work pool is empty, the master sends a request to all workers to send him back a sub-problem. Two states are then reserved to each process (blocked or active). Each time the global work pool is empty, the master checks the state of all workers. If all the workers are blocked, then the master ends the calculation and frees the workers. The workers perform a *depth-first strategy* in order to reach quickly feasible solutions or eliminate the branches if the lower bound is greater than the upper bound. A worker which finds a better solution than the current best one broadcasts the new value to all workers via the master to ensure efficient branching process. An extended version of this approach that exploits the computing power provided by cluster architectures is presented in [7].

4.2. The Proposed GPU-based B&B schemes

In this section, we describe our GPU-based parallelization schemes for the B&B algorithm, these schemes were firstly tested for small instances then adapted here

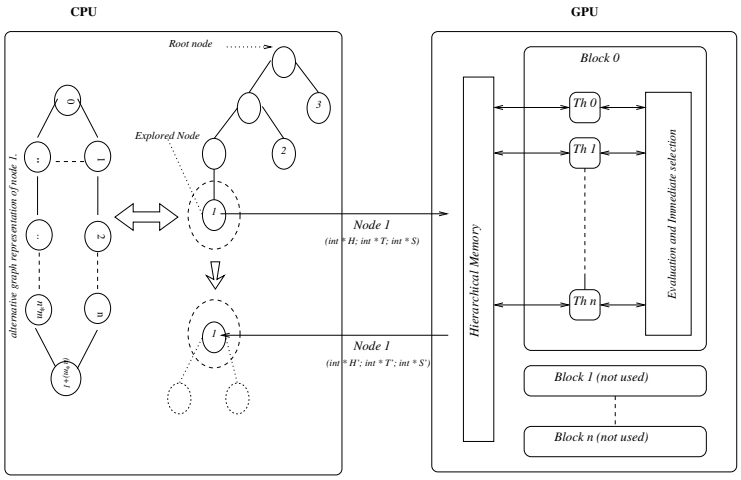

Figure 9: GPU Evaluation of a single node.

for large Taillard instances. To handle such huge instances, we had to readjust the memory allocation and the distribution of the work-units over the threads of a block since the number of the graph operations for large instances is much bigger than the maximum number of threads in GPU block.

The GPU architectures are based on SIMT (Single Instruction, Multiple Threads) paradigm. According to this paradigm, the same program called *kernel* is executed simultaneously by a set of parallel threads with different data. The threads are organized according to a grid of thread-blocks hierarchy specified in the kernel call. The grid represents a set of thread-blocks. Threads of the same block can cooperate by using a private shared memory and barrier of synchronization. Threads can access multiple memory spaces: *constant memory* and *texture memory* are read-only cached memory accessible by all threads. The *global memory* is a read-write memory, also accessible by all threads. Unlike the global memory the *shared memory* is a cached memory accessible only by threads in the same block [27].

4.2.1. Parallel Evaluation of the Bound (PEB)

We have seen in section 3 that the evaluation process and the immediate selection consume together more than 85% of the global execution time, therefore, it is crucial to accelerate this phase in order to reduce the B&B execution time.

In the following, we present our proposed node-based parallelization scheme for the B&B algorithm exploiting GPU-based architectures. The proposed scheme referred to as *Parallel Evaluation of the Bound* (PEB), exploits the idea that the evaluation and immediate-selection for each node can be done in parallel using several threads.

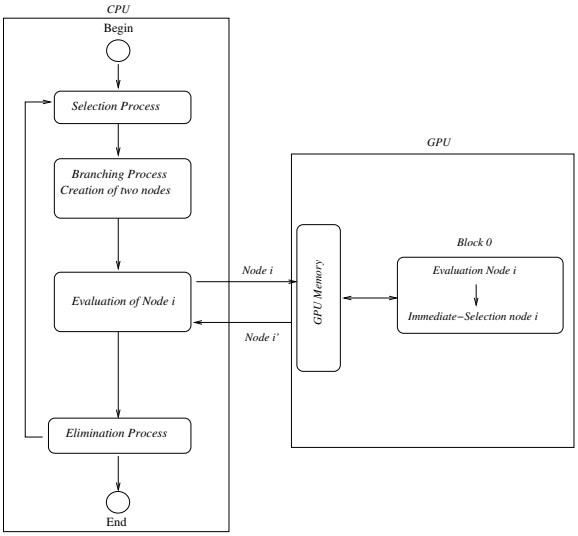

Figure 10: Parallel Evaluation of one Bound (PEB scheme).

As shown in Figures 9 and 10, this approach uses the same design as the sequential B&B algorithm except that the evaluation (bounding) of each node is done in parallel on GPU. As already presented, each node of the search tree represents a graph of $n \times m$ operations. The bounding process consists in updating the head and tail values for each operation in the graph. The parallel *PEB* scheme is based on the idea that each GPU-thread updates the head and tail values for a single operation in the graph. This scheme exploits the fact that the updating process can be done independently for each operation. Therefore, the GPU block size is equal to the

number of operations in the graph ($n \times m$). As shown in Figure 9, at each iteration, only one node is sent to the GPU for evaluation and immediate selection using one thread-block. The bounding begins by copying the head and tail vectors to the shared memory, *i.e.* each thread copies the head and tail values relative to its *id*. After that, each thread updates the head and tail values (H[i], T[i]) for the operation relative to its id (i) using respectively the head of its predecessors and the tail of its successors.

$$H[i] = \text{Max } \{H[r] + p_{ri}\} / r \in Pred[i].$$

$$T[i] = \text{Max } \{T[r] + p_{ir}\} / r \in Succ[i].$$

At the end of this computation, each thread waits for the other threads of the block using a barrier of synchronization to ensure the visibility of the new head and tail values to all threads of the block which is important to have a valid update process.

The work is repeated several times until there is no update of the head or the tail values for all threads or an infeasibility is detected.

After the end of the bounding process, each thread computes the immediate selection for a set of unselected alternative pairs using the new head and tail values. Finally, the new results are sent back to the CPU to be used by the branching and elimination process. As can be seen in Figure 9, a single block is used on the GPU to evaluate one node while the other blocks are idle.

The weakness of this solution lies in the under-utilization of the GPU capacity and thus a waste of a significant computing power. To overcome this drawback, we propose a second level of parallelization.

4.2.2. Parallel Evaluation of Several Bounds (PESB)

We propose in this section a second level of parallelization which allows to more occupy the GPU. This level represents a generalization of the first scheme (PEB) called Parallel Evaluation of Several Bounds (PESB). The goal here is to increase the GPU occupation by generalizing the idea of the first level (Bounding is faster) to exploit more efficiently the GPU computing power. Therefore, at each iteration of the B&B algorithm, a pool of nodes is sent to the GPU for evaluation and immediate-selection instead of one. *i.e.* each GPU-block supports the evaluation of a single node. Then, the new results for each node are sent back to the CPU to be used by the selection, branching and elimination process as shown in Figure 11. The nodes sent to the GPU for evaluation are chosen among the nodes recently added in the B&B work pool which allows to avoid memory saturation by exploring the search tree in depth first.

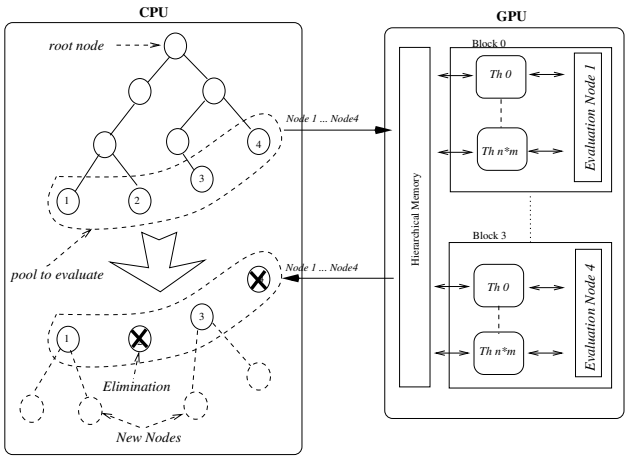

Figure 11: GPU evaluation of several nodes.

As we have already seen, five data structures are used for the bounding of each node on the GPU. The vectors Head (H [$m * n$]), Tail (T [$n * m$]) and alternative pairs (S [$nbpair$]) are sent from the CPU to the GPU. Therefore, they are stored in the global memory of the GPU. The matrices *Succ* and *Pred* are also stored in the GPU global memory. These two matrices are calculated on the GPU using the vector S as an initialization for the bounding. To accelerate this initialization phase the calculation is divided across all the threads of the block.

The access to the global memory is much longer than the shared memory, but the latter is smaller compared to the global memory. The number of blocks that can run in parallel on each Streaming Multiprocessor depends on the amount of shared memory used by each block. Therefore, we use the shared memory only for the Head and Tail vectors in order to have a large number of blocks running in parallel and since there is a high number of accesses to these vectors.

The weakness of the previous approaches lies in the under-utilization of the GPU capacity in addition to the multi-core CPU processors and thus a waste of significant computing power. To overcome this drawback, we propose a hybridization (Multi-core CPU/ GPU based) to increase the GPU occupation.

4.3. Hybrid Master-worker/GPU based parallelization

In this section, we present two hybrid Master-worker/GPU approaches based on the three approaches presented above. This hybridization is based on Nvidia Multi Processes Service (MPS), which is a client-server

runtime implementation of the CUDA API used to increase the overall GPU utilization. Without MPS, only one host process can use the GPU at a given time, therefore, it potentially my underutilize the GPU resources. To overcome this problem, Nvidia provides the MPS to enable multiple host processes like MPI processes to use the Hyper-Q capability on the Nvidia Kepler GPUs. Hyper-Q allows a single host process to process multiple CUDA kernels concurrently on the same GPU. As we can see in Figure 12, the MPS consists of several components: the Control Daemon Process is responsible for starting and stopping the MPS server, as well as coordinating connections between clients and the server [28].

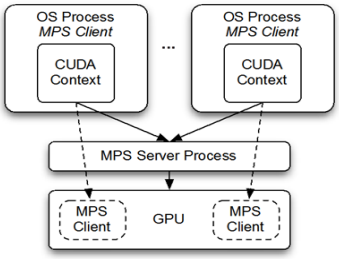

Figure 12: MPS compnents.

The server process provides the connection between clients and the GPU which allows concurrency. Each process (server, clients) has its own CUDA context for its GPU operations. When the MPS client connects to the control daemon, the later creates an MPS server if no server is active, then the client proceed to connect with the server [28]. Note that all communications between MPS clients/server and MPS control daemon is done using a named Pipe. Furthermore, Figure 13. shows how to use the Multi Processes Service (MPS) to run MPI applications.

```
mkdir /tmp/mps /tmp/mps-log
export CUDA_VISIBLE_DEVICES=0                    # SELECT GPU 0.
export CUDA_MPS_PIPE_DIRECTORY=/tmp/mps          # NAMED PIPES
export CUDA_MPS_LOG_DIRECTORY=/tmp/mps-log       # LOGFILES
nvidia-cuda-mps-control -d                       # START THE DAEMON
unset CUDA_VISIBLE_DEVICES
mpirun -x CUDA_MPS_PIPE_DIRECTORY=/tmp/mps -np 35 ./BB
export CUDA_MPS_PIPE_DIRECTORY=/tmp/mps  # SELECT THE LOCATION OF MPS DAEMON
echo quit | nvidia-cuda-mps-control              # STOP MPS DAEMON
rm -rf /tmp/mps /tmp/mps-log
```

Figure 13: Running MPI application using MPS.

4.3.1. *Hybrid Parallel Evaluation of the Bound (H-PEB)*

We propose in this section a hybridization of the first two approaches (Multi-core and GPU node based) to increase the GPU occupation and then improve the runtime. This version generalizes the idea of the PEB approach to exploit the advantages of both the Multi-core CPU processors and the GPU at the same time. The hybrid approach is based on concurrent kernels execution provided by Nvidia in devices of compute capability 2.x and higher. The maximum number of kernels that a device can execute concurrently varies between 16 and 32 according to device compute capability [27].

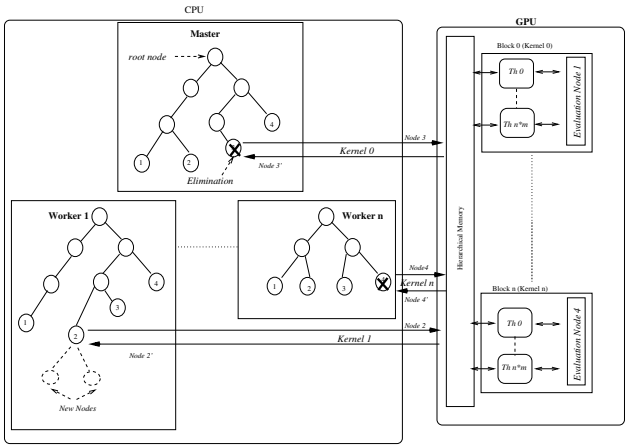

Figure 14: Hybrid Multi-core CPU/GPU approach.

Therefore, in our proposed scheme, several CPU processes from the Multi-core approach (the Master or the workers) launch their kernels simultaneously on the GPU in order to accelerate the bounding of one node at a time according to the PEB scheme. Each host (MPI) process launches its own kernel in the default stream and the MPS server manages to execute the kernels in parallel by using different CUDA-Streams. The advantage of our hybrid approach based on concurrent kernel execution is the occupation of the GPU over time. *i.e.* at each moment, our hybrid approach can have simultaneously several workers executing instructions on the GPU while others perform data-transfer from/to the GPU and yet others apply the selection and elimination operators on the CPU. Using this approach, we have been able to increase the occupation of the GPU but this later still not yet fully occupied because of the limited number of parallel processes and the number of nodes sent by each process. For this reason, we propose in the following an other hybrid parallel approach to fully occupy the GPU.

4.3.2. Hybrid Parallel Evaluation of Several Bounds (H-PESB)

In order to fully occupy the GPU, we propose in the following the last parallel approach called Hybrid Parallel Evaluation of Several Bounds (*H-PESB*). This approach represents a hybridization between the multi-core approach and the *PESB* approach using the Nvidia MPS allowing several MPI-processes to use the GPU at the same time. Each mpi-process (master, workers) has its own B&B algorithm and use the GPU to evaluate a pool of node according to the PESB scheme. Thanks to MPS-server, each MPI-process uses the GPU like it is the only one using it, therefore, the communication between the CPU and the GPU is done exactly the same way as it is in one host process case. This approach is also similar to the H-PEB scheme, except that at each iteration, each host process sends a pool of nodes to the GPU for evaluation and immediate-selection instead of one node at a time. *i.e.* several GPU-blocks are allocated for each MPI-process and each one of them supports the evaluation of a single node. Therefore, each thread of the block update the head and tail values for one or several operations.The nodes sent by each mpi-process to the GPU for evaluation are chosen among the nodes recently added in the mpi-process B&B work pool. At the end, the bounding results of nodes are sent back to the CPU to be used by the corresponding mpi-process for the selection, branching and elimination operations. As shown in Figure 15, The master ho contains the root node begins by dividing the search space between the workers. After that, each worker explores its search space independently from the others and uses the GPU to evaluate several nodes at a time. As explained previously, The master and workers use the GPU simultaneously thanks to the Nvidia MPS that allows concurrent kernels execution while respecting the availability of GPU resources.

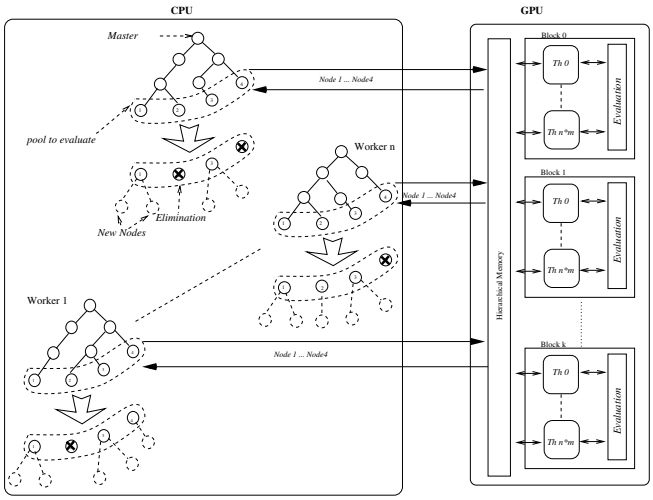

Figure 15: Hybrid Parallel Evaluation of Several Bounds (*H-PESB*).

5. Experimentations

In this section, computational results are given using benchmarks obtained from the well known classical job shop instances by replacing the infinite buffer capacity by a zero buffer capacity constraint. We tested our approaches using the benchmark instances proposed by Taillard's [24]. The different instances are denoted by $n \times m$, where n and m represent respectively the number of jobs and the number of machines. The size of the Taillard's instances for the job shop problem varies between 15x15 and 100x20. The experiments have been carried out using Intel Xeon E5640 CPU with four CPU-cores, 2.67 GHz clock speed each and Nvidia Tesla K40 with 2280 cuda cores and 12 GB GDDR5 of global memory. The approaches have been implemented using C-CUDA 7.0, C++ and MPI [26] as a communication tool between processes. All reported times in this paper represent the average time to explore an equal number of nodes for each benchmark size. In our case, this number is fixed to 700,000 nodes, which is acceptebale since an optimized sequantial B&B algorithm takes 19 hours to complete this number of nodes.

For the 100×20 benchmark instances there are 2002 operations. Since the GPU hardware limit is 1024 threads par block, we adapt the PEB approach to enable each thread to treat 2 operations instead of one which enables us to treat such big instances.

Figure 16 shows the execution time needed for the Multi-core CPU and H-PEB approaches to explore 700,000 nodes using different number of MPI-processes. For the Multi-core approach, the best time is reached for **5** MPI-processes. After that, we notice an increase in execution time when increasing the number of parallel MPI-processes. This can be explained by the limited number of CPU-cores available in our workstation (4 cores). Therefore, the workers tasks are executed sequentialy when the number of workers is above 4. For the Hybrid H-PEB approach the best time is reached for 35 processes which is the maximum supported since the Nvidia MPS support up to 35 connections to the MPS server. This hybrid version supports large number of workers compared to the Multi-core version since each worker has less than 15% of its execution time on the CPU.

For the H-PESB approach, there is no easy way

Table 3: Average execution time (in seconds) of the proposed approaches to explore 700,000 nodes. nb-pr: The number of parallel (host) processes running simultaneously. nb-nodes: The number of nodes evaluated simultaneously on the GPU by each parallel process.

Size	$B\&B_{Seq.}$	$B\&B_{Mcore}$	*PEB* (nb_pr=1)		*H-PEB* (nb_pr=35)		*PESB* (nb_pr=1)		*H-PESB* (nb_pr=5)	
			nb-nodes	time	nb-nodes	time	nb-nodes	Time	nb-nodes	Time
15×15	188	52	1	603	1	162	240	37	90	60
20×15	384	113	1	653	1	164	240	53	90	59
20×20	393	120	1	736	1	173	240	71	86	58
30×15	1076	375	1	795	1	180	240	104	54	60
30×20	1127	447	1	955	1	209	140	156	46	71
50×15	4246	1454	1	1162	1	270	80	280	34	103
50×20	10546	3728	1	1530	1	340	30	396	26	145
100×20	69300	19200	1	3760	1	741	20	1050	20	418

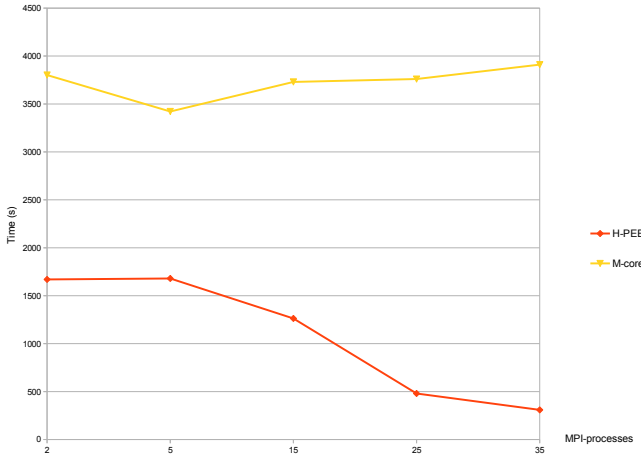

Figure 16: Impact of using different number of MPI-processes to explore 700,000 nodes for Tai61 instance.

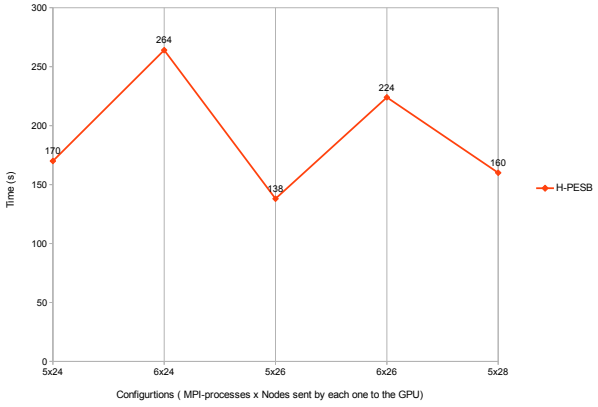

Figure 17: H-PESB execution time using diffrent configurations to explore 700,000 nodes for Tai61 instance.

to find the best configuration. For each instance, we have to test several configurations and take the best one among them. Each configuration is defined as the number of MPI-processes and the number of nodes sent to the GPU by each mpi-process at each iteration. As illustrated in Figure 17 and unlike the H-PEB approach, the best performance is reached for low number of MPI-processes, because the MPS server can manage simultaneously a limited number of physical contexts (one for each parallel process) because of the large amount of virtual memory allocated to each process matching the size of nodes sent to the GPU. For the H-PEB approach we can simultaneously manage large number of physical context (35) because of the low virtual memory allocated to each process matching the size of one node only. We notice also from Figure 17 that it is more benific to increase the number of nodes sent to the GPU instead of increasing the number of MPI-process. Therefore for each instance, we fixed the number of MPI-processes to 5 and we increase the number of nodes sent to the GPU while it reduces the execution time needed to explore 700,000 nodes.

Table 3 reports the average execution times for each approach to explore 700,000 nodes. The first column (*Size*) reports the size of the benchmark instances. Column $B\&B_{Seq}$ reports the average execution time of an optimized sequential B&B algorithm. Column $B\&B_{Mcore}$ gives the execution time obtained by our Master/worker approach exploiting only the Multi-cores CPU using 4 workers. For the all other approaches columns *Time* and *nb-nodes* report respectively the average execution time needed by each approach to explore 700,000 nodes and the number of nodes sent to the GPU at each iteration. For each column the parameter *nb_pr* indicates the num-

ber of parallel processes running simultaneously on the CPU according to the master/worker paradigm.

Column *PEB* reports the results of our GPU node-based approach obtained by sending one node at a time for parallel evaluation on the GPU. Column *PESB* reports the results of our second GPU based approach obtained by sending several nodes to the GPU to be evaluated simultaneously and each one of them is evaluated in parallel using one block of GPU threads.

Columns *H-PEB* and *H-PESB* reports the results of the hybridization between the Multi-core approach and *PEB, PESB* GPU approaches using Nvidia MPS *i.e.* both master and workers use simultaneously the GPU to accelerate they respective bounding processes.

As mentioned before, 35 MPI-processes are used in the *H-PEB* approach and each one uses the default CUDA Stream to launch its kernel to evaluate one node at a time. For the *H-PESB* approach, we fixed the number of parallel processes to 5 due to the huge amount of virtual memory matching the number of nodes sent to the GPU.

We notice from Table 3 that the complexity and the execution time increase when increasing the size of instances. Therefore, the need for parallelization is crucial.

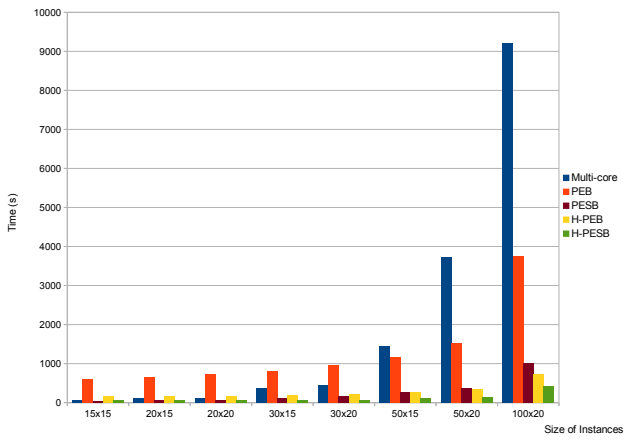

Figure 18: Execution time of the proposed approaches.

Figure 18 shows the histogram representation of the execution time for the different approaches. The first result from Table 3 and Figure 18 is the positive impact of using parallel architectures to reduce the execution time needed to solve the BJSS problem.

The improvement obtained with the Multi-core version is low which is expected since our workstation contains only four CPU-cores. Therefore, increasing the number of workers above four reduces further the obtained performances. For the PEB version, we notice a low performance for small instances (15x15-30x20) against the Multi-core and sequential approaches. This can be explained by the high ratio of communication to computing time on the GPU *i.e.* the approach passes more time in sending data and recovering results to/from the GPU. By increasing the size of instances, we notice a significant improvement in execution time as compared with the sequential and multi-core cases. In addition to the efficiency in reducing the execution time for large instances, the PEB approach does not depend on the GPU capacity since it uses a small amount of GPU resources. However, only one block is used and the other blocks remain idle. Therefore, this approach can not benefit from the entire GPU capacity.

The performence of the *PESB* approach depends on the number of nodes that the GPU can evaluate simultaneously which is determined by the amount of the shared memory used by each GPU block to evaluate a node. The number of nodes evaluated simultaneously for small instances is equal to the maximum number of block that our GPU can run simultaneously (240). By increasing the size of instances, the number of nodes evaluated simultaneously matching the number of parallel blocks that a GPU can handle decreases. This behaviour can be explained by the huge amount of shared memory needed to each block for large instances. Since the amount of shared memory is fixed, the number of parallel block decreases by increasing the shared memory used by each block matching the generated data for the handled instance used for synchronization. By sending several nodes at a time instead of one in the PEB approach, we have been able to reduce the execution time by a factor of 3 as compared with the PEB approach and a factor of 18x as compared to the multi-core version.

The hybrid approach (*H-PEB*) reduces considerably the execution time even for small instances against the sequential approach. This performance represents the results of exploiting both the CPU-cores and the GPU at the same time by using concurrent kernels execution provided by nvidia MPS which allows us to increase the GPU occupation. Furthermore, the wasted time in CPU/GPU communications is covered by the concurrent access to the GPU where several workers execute their bounding operation at the same time. This hybridization allows us to reduce the execution time by a factor of 5x as compared with the PEB approach and a factor of 26x as compared with the multi-core version. Unlike smaller instances, the *H-PEB* approach outperform the results of the *PESB* approach for large instances due to limited number of nodes handled si-

multaneously by this later.

Our last parallel approach (*H-PESB*) is also based on the Nvidia MPS, it represents a hybridization between the multi-core approach and the *PESB* approach. This approach fully occupies the GPU which explains the good obtained performance even for the smaller instances. The result of this approach is 2 times faster as compared with *PESB* approach and 46 times faster as compared with the multi-core version. The results of our two hybrid approaches show clearly the benefit of using both the multi-core CPU and the GPU at the same time as compared with approaches exploiting only the Multi-core CPU or only the GPU.

We notice for our experience of using Nvidia MPS that the best performance is obtained for large number of parallel host processes as we can notice from the H-PEB results in table 3 and Figure 16. But, This depends on the MPS server resources *i.e.* if he has enough virtual memory space for the parallel processes which is not the case in the H-PESB approach.

Table 4: The number of GPU communications needed for each approach to explore 700,000 nodes.

Approaches	#processes	#nodes sent	GPU communications
Multi-core	5	0	0
PEB	1	1	1400,000
H-PEB	35	1	40,000
H-PESB	5	20	14,000

Table 4 shows the number of GPU communications needed for each approach to explore 700,000 nodes. For each approach, column processes reports the number of used parallel processes. Column nodes reports the number of nodes sent by each process. Finally, column GPU communications reports the number communications between the CPU and the GPU. The PEB approach has huge number of GPU communications (1400,000) because at each iteration, only one node is sent to the GPU then the results are move back to the CPU. For the H-PEB approach we have 40,000 communications since at each iteration, we can have 35 connections at the same time to the GPU without blocking. The same for the H-PESB approach, the later reduces more the number of communications to the GPU which explains the obtained performances.

Figure 19 shows the relative speedup of our proposed approaches for different problem sizes. The speedup of our Multi-core version is around 4 for all sizes which is expected since it depends on the number of CPU-cores available in our workstation. The speed-up of

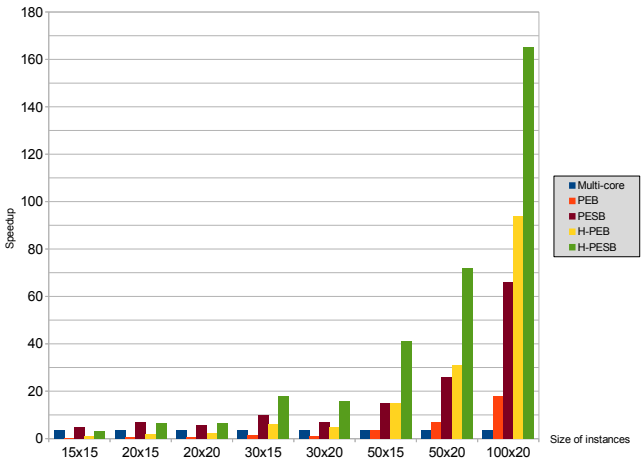

Figure 19: The speedup of the proposed approaches.

the other approaches is proportional to the size of instances. Therefore, the maximum speed-up is obtained for the 100x20 instances. This is logical because the speedup of these approaches depends on the amount of the computation on the GPU. The idea used in PEB approach to accelerate the bounding of one node at a time on GPU using several threads organized into one GPU block gave good results (18 times faster) compared to the sequential version.

The *PESB* approach gave the best performance against all approaches for small instances, however this is not the case for large instances as compared with the H-PEB and *H-PESB* approaches due to the limited number of nodes evaluated simultaneously on the GPU. This can be explained by the limited amount of shared memory available in the device and the large amount of this memory needed by each GPU block for synchronization.

The speedup obtained by the hybrid H-PEB approach is around 90 times faster which confirms the efficiency and the benefit of using both CPU-cores and GPU at the same time.

The *H-PESB* approach that fully occupy the GPU has acheived the best performances for almost all sizes agains all other approaches. It has acheived an impressive speedup, especially for the largest instances where it is up to 160 times faster than a sequential B&B algorithm. In addition, the speedup of hybrid approaches grows according to the size of instances. This is due to the ratio of computing to communication time that increases by increasing the size of instances. This proves that the hybrid approaches are scalable and can easily deal with large instances. These last two ap-

proaches are based on the concurrent kernels execution via Nvidia Multi Processes Service (MPS) which is rarely exploited in scientific computing. The performance (speedup) of the hybrid approaches is the result of:
1- Using the PEB scheme which is 18 times faster as the basis of our hybrid approaches.
2- Exploiting both the power of the CPU-cores and the GPU at the same time using Nvidia MPS.
3- The occupation of the GPU over time *i.e.* several workers run instructions on the GPU while others perform data-transfer from/to the GPU and yet others apply elimination and branching operators on the CPU.

6. Conclusion

This paper investigates the acceleration of the B&B method using Multi and Many-core systems in order to solve the NP-hard Blocking Job Shop Scheduling problem. This problem represents a version of the classical JSSP with no intermediate buffer between machines. In this paper, five approaches have been proposed. The first approach exploits only the CPU-core of our machine. The second one (PEB) is a GPU node based parallelization. Finally, the last two approaches (H-PEB and H-PESB) are hybrid, they exploit the Multi-core CPU and the GPU at the same time by combining the first two approaches using concurrent Kernels execution provided by Nvidia MPS. The obtained results confirm the efficiency of our proposals and the positive impact of using computing accelerators like GPUs to solve this problem. The results show the advantage of increasing the GPU occupation over time by using Hybrid approaches based on the cuncurent kernels execution provided by Nvidia MPS which allows us to achieve an impressive speedup of 160x for large instances as compared with an optimized sequential B&B approach.

As a future perspective, we plan to act on the granularity of tasks assigned to each thread and explore more heterogeneous architectures like Intel Xeon Phi.

Acknowledgment

We would like to thank Dr. Djamal Belazzougui for his assistance and comments that greatly improved the presentation of this manuscript.

Dr.Didier El Baz gratefully acknowledges the support of NVIDIA Corporation with the donation of the Tesla K40 GPU used for this research work.

References

[1] AitZai, Abdelhakim, Brahim Benmedjdoub, and Mourad Boudhar. "A branch and bound and parallel genetic algorithm for the job shop scheduling problem with blocking." International Journal of Operational Research 14, no. 3 (2012): 343-365.

[2] Bendjoudi, Ahcène, Mehdi Chekini, Makhlouf Gharbi, Malika Mehdi, Karima Benatchba, Fatima Sitayeb-Benbouzid, and Nouredine Melab. "Parallel B&B Algorithm for Hybrid Multicore/GPU Architectures." In High Performance Computing and Communications (HPCC), 2013 IEEE 10th International Conference on, pp. 914-921. IEEE, 2013.

[3] Brucker, Peter. Scheduling algorithms. Vol. 3. Berlin: Springer, 2007.

[4] Carneiro, Tiago, Albert Einstein Muritiba, Marcos Negreiros, and Gustavo Augusto Lima de Campos. "A new parallel schema for branch-and-bound algorithms using GPGPU." In Computer Architecture and High Performance Computing (SBAC-PAD), 2011 23rd International Symposium on, pp. 41-47. IEEE, 2011.

[5] Adel Dabah, Ahcene Bendjoudi, Abdelhakim AitZai. "Multi and Many-core Parallel B&B approaches for the Blocking Job Shop Scheduling Problem." 14 International Conference on High Performance Computing & Simulation (HPCS), 2016 : 705-713.

[6] Dabah, A., Bendjoudi, A., El-Baz, D., & Aitzai, A. (2016, May). GPU-Based Two Level Parallel B&B for the Blocking Job Shop Scheduling Problem. In Parallel and Distributed Processing Symposium Workshops, 2016 IEEE International (pp. 747-755). IEEE.

[7] Adel Dabah, Ahcene Bendjoudi, Abdelhakim AitZai. "Efficient parallel B&B method for the blocking job shop scheduling problem." 14 International Conference on High Performance Computing & Simulation (HPCS 2016)

[8] Chakroun, Imen, Mohand Mezmaz, Nouredine Melab, and Ahcene Bendjoudi. "Reducing thread divergence in a GPU-accelerated branch-and-bound algorithm." Concurrency and Computation: Practice and Experience 25, no. 8 (2013): 1121-1136.

[9] CHAKROUN, Imen, MELAB, Nordine, Mohand MEZMAZ, Daniel Tuyttens: Combining multi-core and GPU computing for solving combinatorial optimization problems. Journal of Parallel and Distributed Computing, 2013, vol. 73, no 12, p. 1563-1577.

[10] Carlier, Jacques, and Eric Pinson. "Adjustment of heads and tails for the job-shop problem." European Journal of Operational Research 78.2 (1994): 146-161.

[11] Gröflin, Heinz, and Andreas Klinkert. "A new neighborhood and tabu search for the blocking job shop." Discrete Applied Mathematics 157, no. 17 (2009): 3643-3655.

[12] Gendron, Bernard, and Teodor Gabriel Crainic. "Parallel branch-and-branch algorithms: Survey and synthesis." Operations research 42, no. 6 (1994): 1042-1066.

[13] Hall, Nicholas G., and Chelliah Sriskandarajah. "A survey of machine scheduling problems with blocking and no-wait in process." Operations research 44, no. 3 (1996): 510-525.

[14] Lalami, Mohamed Esseghir, and Didier El-Baz. "GPU implementation of the branch and bound method for knapsack problems." In Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2012 IEEE 26th International, pp. 1769-1777. IEEE, 2012.

[15] Melab, Nouredine, Imen Chakroun, and Ahcène Bendjoudi. "Graphics processing unit-accelerated bounding for branch-and-bound applied to a permutation problem using data access optimization." Concurrency and Computation: Practice and Experience 26, no. 16 (2014): 2667-2683.

[16] Mati, Yazid, Nidhal Rezg, and Xiaolan Xie. "A taboo search approach for deadlock-free scheduling of automated manufacturing systems." Journal of Intelligent Manufacturing 12, no. 5-6 (2001): 535-552.
[17] Mascis, Alessandro, and Dario Pacciarelli. "Job-shop scheduling with blocking and no-wait constraints." European Journal of Operational Research 143, no. 3 (2002): 498-517.
[18] Melab, Nouredine, et al. "Multi-core versus many-core computing for many-task Branch-and-Bound applied to big optimization problems." Future Generation Computer Systems (2017).
[19] Meloni, Carlo, Dario Pacciarelli, and Marco Pranzo. "A rollout metaheuristic for job shop scheduling problems." Annals of Operations Research 131.1-4 (2004): 215-235.
[20] Oddi, Angelo, Riccardo Rasconi, Amedeo Cesta, and Stephen F. Smith. "Iterative Improvement Algorithms for the Blocking Job Shop." In ICAPS. 2012.
[21] Pranzo, Marco, and Dario Pacciarelli. "An iterated greedy metaheuristic for the blocking job shop scheduling problem." Journal of Heuristics (2013): 1-25.
[22] Roy, Bernard, and B. Sussmann. "Les problemes d'ordonnancement avec contraintes disjonctives." Note ds 9 (1964).
[23] Riebler, Heinrich, et al. "Efficient Branch and Bound on FPGAs Using Work Stealing and Instance-Specific Designs." ACM Transactions on Reconfigurable Technology and Systems (TRETS) 10.3 (2017): 24.
[24] Taillard, Eric. Taillard's FSP benchmarks. http://mistic.heigvd.ch/taillard/problemes.dir/ordonnancement.dir/ordonnancement.html.
[25] Trong-Tuan Vu, Bilel Derbel. Parallel Branch-and-Bound in multi-core multi-CPU multi-GPU heterogeneous environments. Future Generation Computer Systems, Volume 56, March 2016, Pages 95-109.
[26] Message Passing Interface Forum. MPI: A Message Passing Interface Standard, Version 3.0, 2012.
[27] NVIDIA Corporation. CUDA C Programming Guide https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html.
[28] NVIDIA Corporation. Multi-Process Service, https://docs.nvidia.com/deploy/pdf/CUDA_Multi_Process_Service_Overview.pdf
[29] Wangdong Yang, Kenli Li, Keqin Li. A hybrid computing method of SpMV on CPU-GPU heterogeneous computing systems. Journal of Parallel and Distributed Computing, Volume 104, June 2017, Pages 49-60.
[30] Danilo S. Souza, Haroldo G. Santos, Igor M. Coelho. A Hybrid Heuristic in GPU-CPU Based on Scatter Search for the Generalized Assignment Problem. Procedia Computer Science, Volume 108, 2017, Pages 1404-1413.
[31] Leila Asadzadeh. A local search genetic algorithm for the job shop scheduling problem with intelligent agents. Computers & Industrial Engineering, Volume 85, July 2015, Pages 376-383.
[32] HE Nouri, O Belkahla Driss, K Ghédira. Hybrid metaheuristics for scheduling of machines and transport robots in job shop environment. Applied Intelligence 45 (3), 2016, 808-828.
[33] Houssem Eddine Nouri, Olfa Belkahla Driss, Khaled Ghédira. Hybrid Metaheuristics within a Holonic Multiagent Model for the Flexible Job Shop Problem. Procedia Computer Science, Volume 60, 2015, Pages 83-92.