Multi and Many-core Parallel B&B approaches for the Blocking Job Shop Scheduling Problem.

Adel Dabah[1,2], Ahcène Bendjoudi[1], Abdelhakim AitZai[2]
[1] *CERIST Research Center Algiers, Algeria*
Email:{adabah,abendjoudi}@cerist.dz
[2] *University of Sciences and Technology*
Houari Boumedienne (USTHB) Algiers, Algeria
Email:{h.aitzai,adel.dabah}@usthb.dz

Didier El-Baz[4], Nadia Nouali Taboudjemat[3]
[3] *CERIST Research Center Algiers, Algeria*
Email:nnouali@cerist.dz
[4] *LAAS-CNRS, Université de Toulouse, CNRS,*
Toulouse, France
Email: elbaz@laas.fr

***Abstract*—In this paper, we propose three approaches to accelerate the B&B execution time using Multi and Many-core systems to solve the NP-hard *Blocking Job Shop Scheduling* problem (BJSS). The first approach is based on Master/Worker paradigm where the workers independently explore the branches sent by the master. The second approach is a node-based parallelization that does not change the design of the B&B algorithm, except that the bounding process is faster since it is calculated in parallel using several threads organized in one GPU block. The third approach is a Multi-Core CPU/GPU hybridization that benefits 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) i.e. each host process (Master or Worker) launches his own kernel to accelerate the bounding process on GPU. The obtained results using Taillard instances confirm the efficiency of our proposals. The first two approaches are respectively three and eighteen times faster compared to the sequential version. The results of the hybrid approach show a relative speedup over ninety times as compared to the sequential approach and therefore prove the advantage of using both the CPU-cores and the GPU at the same time.** ***Keywords*-Job shop; blocking with swap; GPGPU; Multi-core CPU; parallel computing; Branch-and-Bound.**

I. Introduction

The job shop scheduling problem (JSSP) consists in scheduling a set of jobs on a set of machines. Each job has its own sequence of crossing on machines. The classical JSSP assumes an infinite storage space between machines which is not realistic. The BJSS is a version of the classical JSSP with no storage space between machines, where a job has to wait on a 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 [11], and the blocking extension of this problem BJSS appears to be even more difficult to solve [15]. 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 an exact method based on intelligent enumeration of all feasible solutions. Nevertheless, its sequential case takes a huge amount of time to solve small instances and remains inefficient when dealing with large instances. Therefore, the parallelization of this method is essential. In the literature, several CPU and GPU parallel B&B algorithms have been proposed [6], [13], [2], [5], [4]. However, most authors exploit only the CPU-core or only the GPU which may results in the under-utilization of these resources and a loss of a significant computing power, hence, a loss in performance.

In this paper, we propose three approaches to accelerate the B&B execution time using Multi and Many-core systems. The first approach is a tree based parallelization, exploiting 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 approach is a node-based parallelization (Parallel Evaluation of the Bound), exploiting the idea that the evaluation of each node can be calculated in parallel. Therefore, at each iteration one node will be sent for parallel evaluation on GPU by using several threads organized in one GPU block. Experiments show that this version is 18 times faster than the sequential B&B version. The drawback of the first two approaches is the underuse of the CPU and GPU resources. To overcome this drawback, we propose a hybrid CPU-core/GPU approach to benefit from both the multi-core CPU and the GPU at the same time. This approach is based on the concurrent kernels execution provided by Nvidia MPS *i.e.* each host process (master or worker) launches his own kernel to accelerate his bounding process on the GPU. The obtained results, using the Taillard instances show a relative speedup of 93x as compared to an optimized sequential B&B version which confirms the efficiency of the proposed hybridization.

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.

II. Blocking job shop scheduling problem

A. 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 arc represents the fact that one operation must be completed before starting the processing of the other operation. A selection S_1 is a set of arcs obtained from A by choosing at most one arc from each pair, and $G(S_1) = (N, F \cup S_1)$ represents the obtained graph. 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 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.

Table 1 represents a BJSS instance with two products (jobs) and three machines. The first product $(J1)$ has 5 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 $M3$.

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

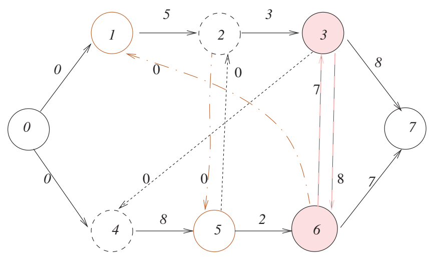


Figure 1. Alternative graph for BJSSP instance of table 1.

Figure 1 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. His 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.

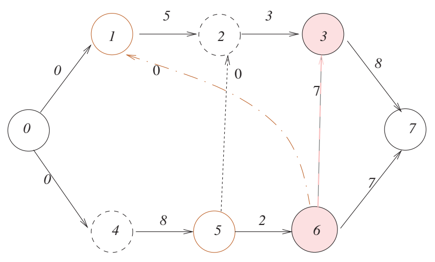


Figure 2. Schedule for BJSP in table 1 whit *Cmax*=26.

Figure 2 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 1. The *Makespan* (*Cmax* = 26) of this schedule is the longest path in the obtained graph.

The Gantt chart in Figure 3 represents both the processing and blocking times of the solution of Figure 2.
For example, after the end of its processing time the job $J1$ blocks the machine $M1$ until machine $M2$ becomes available for processing $J1$.

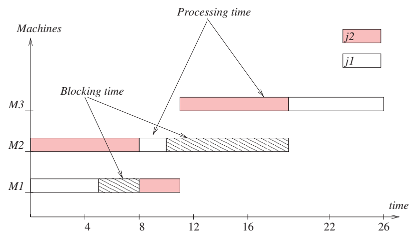


Figure 3. Gantt chart of the schedule in figure 2.

1) Alternative pairs 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.

Case 1: the alternative pair between operations o_i and o_j (Fig. 4): 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.

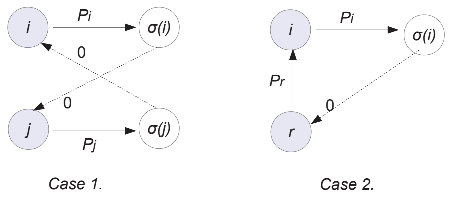


Figure 4. Alternative pairs between blocking and ideal operations.

Case2: the alternative pair between operations o_i and o_r (Fig. 4): 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.

B. Related works

Most of B&B methods, for the job shop problem, are based on the resolution of single machine problems proposed by Carlier. For solving optimally the BJSS we find the B&B method proposed by Mascis *et al*. [15]. The authors formulate the problem by means of an alternative graph model which is a generalization of the disjunctive graph of Roy and Sussman [19]. 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. The idea of his 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. In [6] and [13]. Chakroun *et al.* take the classical approach of sending nodes to be evaluated on GPU to solve the FSP 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], [7] the authors extend the approach below and propose a multi-core/GPU scheme to exploit both multi-core CPU processors and GPU accelerator to solve the same problem. In [5], 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.

Most of the previously cited works focus on exploiting the GPU part and ignoring the available CPU-cores. For this reason, we propose an original hybrid CPU-core/GPU approach based on concurrent kernels execution to exploit both CPU and GPU parts of our workstation.

III. THE BRANCH AND BOUND ALGORITHM FOR BJSS

The B&B algorithms make an intelligent enumeration of all feasible solutions. They are 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 lower bounding operator is used to compute the lower bound for the evaluation of all feasible solutions in the considered sub-problem. The elimination operator uses the bounds to eliminate the sub-problems that cannot improve the current best solution found for the problem. Algorithm 1 describes the used B&B algorithm.

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

```
$LIST \leftarrow \{original\, problem\}$;
$UB \leftarrow \infty$;
while $LIST$ != $\emptyset$ do
    $R \leftarrow LIST$ (Choose a Node $R$ from $LIST$);
    Generate successors $R_i$ from $R \mid (i = 1, ..., n)$;
    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
                $LIST = LIST \cup R_i$;
            end if
        end if
    end for
end while
return $s^*$
```

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) [15]. Our method consists in fixing an order (precedence) between every two concurrent operations, which leads to fix the corresponding alternative pair (from A).

A. Branching

The B&B algorithm can be represented by a search tree. The tree is rooted by the original problem; 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} = $A_R - \{((i,j),\ (h,k))\}$. The corresponding successors represent 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 sub-problems and prune the tree if the lower bound value of the current sub-problem is bigger than the upper bound. Finally, our exploration strategy after a branching process is to choose the node which has the bigger Makespan, which allows to reach rapidly feasible solutions and also leads to improve the UB and eliminate a large number of branches.

B. Evaluation (Bounding)

Any solution of the problem can be considered as an initial value for an upper bound (in our case UB=$+\infty$) which is updated as soon as a new better solution is found. The lower bound (Evaluation) used in our case is the one used by Carlier *et al.* [8] to solve optimally the JSSP. It is based on the *one machine scheduling problem*. To do a link with alternative graph model, each search tree node represents an alternative graph. The lower bound used is similar to the Makespan of the sub-problem obtained by adjusting the head and tail structures ($H_i = l(0,i), T_i = l(i, n*m)$) for each operation o_i (i = 1, ..., $n*m$); in the graph representation. This process is very expensive and consumes 70% of global execution time of the method. 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 that lead to this operation.

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, as illustrated below, requires six data structures. The matrix MP $(n*m) \times (n*m)$ represents the length of all alternative arcs, MP[i][j]=a_{ij} if the arc exists and -1 if not. The matrix Succ $((n*m) \times n)$ contains the successors of each operation, therefore, row i represents the successors of operation o_i. Similarly, the matrix Pred $((n*m) \times n)$ contains the predecessors of each operation.

C. 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 the head and tail values computed in the bounding process. Given a 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.

IV. The proposed parallelization approach for the B&B algorithm

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

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 parallelization strategies, according to the degree of parallelization. For more details the reader may refer to [12].

A. Multi-core parallel B&B

In this section, we describe the proposed parallel B&B algorithm, exploiting the CPU-core available in our workstation. The proposed approach (see Fig. 5) is based on

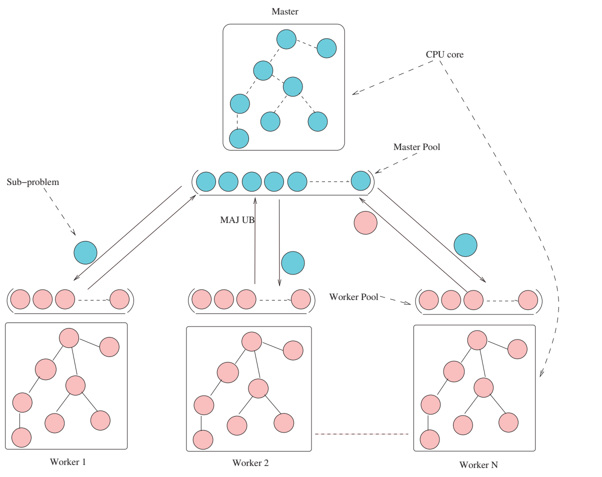


Figure 5. Global architecture of the proposed parallel B&B algorithm.

the master/worker paradigm. The exploration of the search-tree is done simultaneously by the master and workers, the results given by a worker can influence others. Therefore, our approach can be seen as a multi-search parallelization.

A work pool represents a set of active sub-problems. There are two types of work pools: a unique global work pool managed by the master and several local work pools owned by the different workers. Each worker has its own local work pool (see Fig. 5). Therefore, a collegial strategy is considered. The master initializes the search tree by creating the root, launches his own B&B algorithm which generates a set of active sub-problems stored in the global (master) work pool and wakes up the blocked workers by sending a sub-problem from the global work pool. After that, each worker launches his own B&B algorithm. During the search, the local pools evolve continuously and when they become empty, the corresponding workers send a request to the master and wait for sub-problems. The workers perform a *worst-first strategy* in order to reach feasible solutions more quickly 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 it 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 will be presented in [5].

B. The Proposed GPU-based B&B algorithm

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.

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 [22];

In the following, we present our proposed node-based parallelization scheme for the B&B algorithm, exploiting GPU-based architectures. The proposed scheme exploits the idea that the evaluation and immediate-selection steps can be done in parallel for each node.

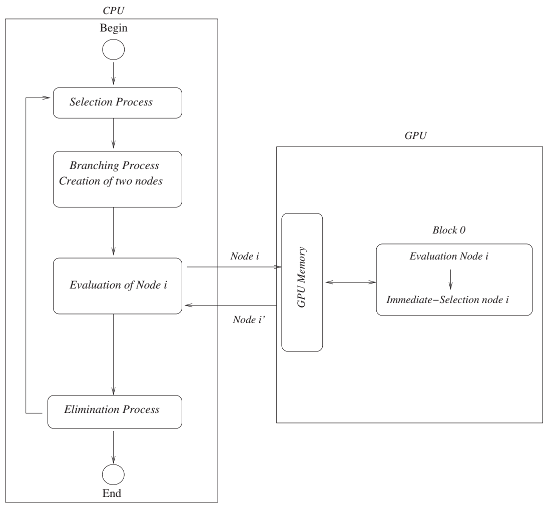
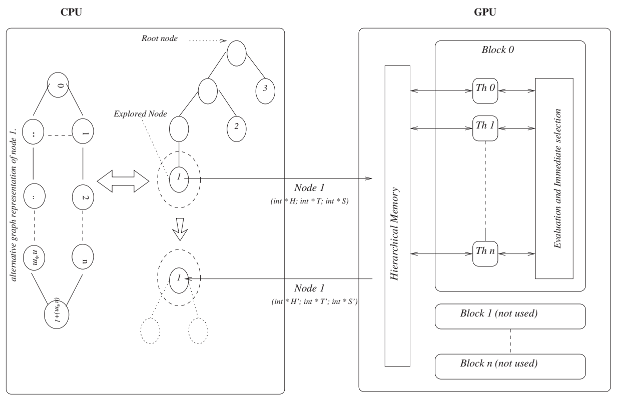


Figure 6. First level PEB scheme.

This approach uses the same design as the sequential B&B algorithm except that the evaluation (bounding) is done in parallel on GPU for each node as shown in Figure 6. As already presented, each node of the search tree represents a graph of $n \times m$ operations. The evaluation process consists in updating the head and tail values for each operation in the graph. At the PEB level, we propose a parallel evaluation scheme based on the idea that each GPU-thread supports updating head and tail values for a single operation in the graph, exploiting the fact that the updating can be done independently for each operation. Therefore, the GPU block size equals $n \times m$, the number of operations in the graph. As shown in Figure 7, at each iteration, only one node is sent to the GPU for evaluation and immediate selection using one thread-block. Each thread updates the head and tail values for one operation. The new values are sent back to the CPU to be used in the branching and elimination process. As can be seen in Figure 7, a single block is used on the GPU to evaluate one node while the others block are idle. The



CPU
GPU
Root node
Explored Node
alternative graph representation of node 1.
Block 0
Th 0
Th 1
Th n
Hierarchical Memory
Evaluation and Immediate selection
Node 1
(int * H; int * T; int * S)
Node 1
(int * H'; int * T'; int * S')
Block 1 (not used)
Block n (not used)

Figure 7. GPU evaluation of a single node.

weakness of this solution resides in the under-utilization of the GPU capacity and thus a waste of a significant computing power. To overcome this drawback, we propose a hybridization of the first two approaches (Master/Worker and the GPU based) to increase the GPU occupation.

C. Hybrid Multi-core CPU/GPU parallelization (H-PEB)

We propose in this section a hybridization of the first two approaches (Multi-core CPU and GPU) to increase the GPU occupation. This version generalizes the idea of the PEB approach to exploit the advantages of both CPU-core and 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 [22]. Therefore, each CPU process (Master or workers) launches his own kernel in the default stream to accelerate his bounding of each node on the GPU. Furthermore, Figure 8 explains the

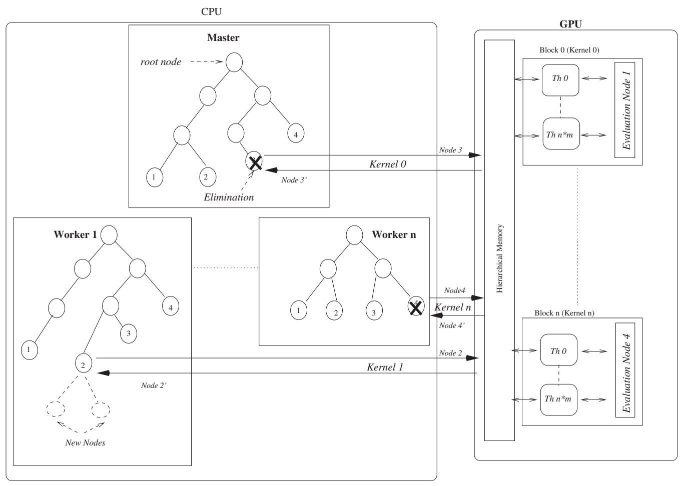


Figure 8. Hybrid Multi-core CPU/GPU approach.

hybrid approach. 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. This hybrid approach provides also a way to reduce the overhead of the CPU/GPU data-transfer.

1) Nvidia Multi Processes Service (MPS) and concurrent kernels execution:

MPS 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 Multi Processes Service 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 9 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 [23]. The

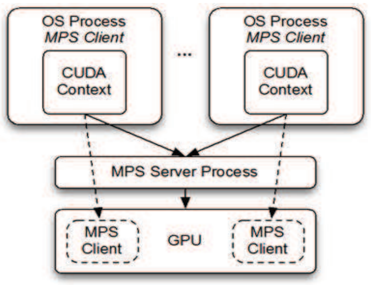


Figure 9. MPS compnents.

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 [23]. Note that all communications between MPS clients/server and MPS control daemon is done using a named Pipe. Furthermore, figure 10. shows how to use the Multi Processes Service (MPS) to run MPI applications.

V. EXPERIMENTATIONS

In this section computational results are given using benchmarks obtained from the well known classical job shop instances by dropping the infinite buffer capacity constraint, and replacing it by a zero buffer capacity.

We tested our algorithms using the large size benchmarks proposed by Taillard's [20]. The different instances aredenoted by $n \times m$, where n and m represent respectively the number of jobs and the number of machines.

```
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 10. Running MPI application using MPS.

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 approach has been implemented using C-CUDA 7.0. and MPI [21] as a communication tool between processes. All reported times in this paper represent the average time to explore 700,000 nodes for each benchmark. 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.

To find the appropriate number of workers we tested our proposed approaches (Multi-core CPU and H-PEB) using different number of workers to explore 700,000 nodes. For the Multi-core version, the best time is reached for **4** workers. After that, we notice an increase in execution time when increasing workers number. 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 workers number is above 4. For the Hybrid H-PEB version, the best time is reached for **35** workers which is the maximum supported since the Nvidia MPS support up to 35 connection 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 his execution time on the CPU.

Table 2
AVERAGE EXECUTION TIME IN SECOND OF THE PROPOSED APPROACHES TO EXPLORE 700000 NODES.

Size	$B\&B_{Seq.}$	$B\&B_{Mcore}$	$B\&B_{PEB}$	$Hybrid_{PEB}$	$speedup$
20×20	393	120	736	173	2.3
30×15	1076	375	795	180	6.0
30×20	1127	447	955	209	5.4
50×15	4246	1454	1162	270	15.7
50×20	10546	3728	1530	340	31.0
100×20	69300	19200	3760	741	93.5

Table 2 reports the average execution times of the sequential and proposed approaches. The first column (*Size*) reports the size of the benchmark instances. Column *Seq.* $B\&B$ reports the average execution time of an optimized sequential B&B algorithm. Column $B\&B_{M-core}$ gives the execution time obtained by our Master/worker approach exploiting only the CPU-cores of our workstation using 4 workers. Column $Hybrid_{PEB}$ reports the average execution time for exploring 700,000 nodes by sending one node for evaluation on GPU at each time. Column $B\&B_{H-PEB}$ reports the average execution time of our hybrid CPU-core/GPU approach using Nvidia MPS i.e both master and workers accelerate they bounding process on GPU using PEB model. As mentioned before, 35 workers are used in this hybrid approach and each one uses the default CUDA Stream. Finally, column $speedup$ reports the ratio between the sequential and parallel execution time for the Hybrid CPU-core/GPU B&B method.

We notice from Table 2 that the complexity and the execution time increase when increasing the size of instances. Therefore, the need for parallelization is crucial.

The first result from table 2 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. For the PEB version, we notice a low performance for small instances against the Multi-core and sequential approaches. This can be explained by the high ratio of communication to computing time on 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 efficiency in reducing the execution time, this approach (PEB) does not depend on GPU capacity since we use less then 5% of the GPU resources. Unlike the PEB approach, the hybrid

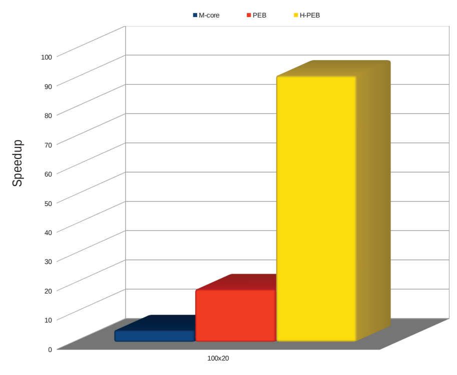


Figure 11. the speedup of the proposed approaches.

approach ($H - PEB$) provides good acceleration even for small instances. The results of this hybrid approach boost up the speedup to reach more than 90x as compared with an optimized sequential B&B method. Also the results confirm the advantage of using both CPU-core and GPU at the same time by using concurrent kernels execution provided by Nvidia MPS.

Figure 11 reports the relative speedup of our proposed three approaches for solving 100x20 problem instances. The speedup of our Multi-core version (3 times faster) is expected since it depends on the number of CPU-cores available in our workstation. The idea used in the second approach (node based) to accelerate the bounding on GPU using several threads organized in one GPU block gave good results (18 times faster) compared to the multi-core version. The speedup obtained by our proposed hybridization (H-PEB) is around 90 times faster. This result confirms the efficiency and the benefit of using both CPU-cores and GPU at the same time. This approach is based on concurrent kernels execution via Nvidia Multi Processes Service (MPS) which is rarely exploited in scientific computing. The speedup of the hybrid approach is the result of 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.

VI. 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, three approaches have been proposed. The first approach exploits only the CPU-core of our machine. The second one is a GPU node based parallelization. Finally, a third one to increase the GPU occupation by combining the first two approaches using concurrent Kernels execution provided by Nvidia MPS. The obtained results confirm the efficiency of the proposed approaches and the positive impact of using computing accelerators like GPUs to solve this problem. The performance of the Multi-core based approach is low since it depends on the number of available CPU-core which is limited. The second approach is 18 times faster and does not depend on the GPU capacity but it underutilizes the GPU resources. The third approach increases the GPU occupation which allows us to reach a speedup over 90 times faster for large instances as compared with an optimized sequential B&B version. As a future perspective, we plan to explore heterogeneous architectures like multi-core CPU, coupled with GPUs and Intel Xeon Phi.

Acknowledgment

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 Multi-core/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. "Efficient parallel B&B method for the blocking job shop scheduling problem." 14 International Conference on High Performance Computing & Simulation (HPCS 2016)

[6] 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.

[7] 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.

[8] 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.

[9] 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.

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

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] Oddi, Angelo, Riccardo Rasconi, Amedeo Cesta, and Stephen F. Smith. "Iterative Improvement Algorithms for the Blocking Job Shop." In ICAPS. 2012.

[18] Pranzo, Marco, and Dario Pacciarelli. "An iterated greedy metaheuristic for the blocking job shop scheduling problem." Journal of Heuristics (2013): 1-25.

[19] Roy, Bernard, and B. Sussmann. "Les problemes d'ordonnancement avec contraintes disjonctives." Note ds 9 (1964).

[20] Taillard, Eric. Taillard's FSP benchmarks. http://mistic.heigvd.ch/taillard/problemes.dir/ordonna /ordonnancement.html.

[21] Message Passing Interface Forum. MPI: A Message Passing Interface Standard, Version 3.0, 2012.

[22] NVIDIA Corporation. CUDA C Programming Guide https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html.

[23] NVIDIA Corporation. Multi-Process Service, https://docs.nvidia.com/deploy/pdf/CUDA_Multi_Process _Service_Overview.pdf