ISSN 2047-3338

Parallelization the Job-shop Problem on Distributed and Shared Memory Architectures

Vu Dinh Trung[1] and Tran Van Lang[2]
[1]Faculty of Information Technology, Lac Hong University
[2]Institute of Applied Mechanics and Informatics, Vietnam Academy of Science and Technology
[1]trung@lhu.edu.vn, [2]tvlang@vast-hcm.ac.vn

***Abstract*— The paper presents the parallel algorithm for solving the scheduling problem. This algorithm is implemented in the distributed memory multi-computers, and with each machine using CPU - GPU shared memory architecture, so that the time to complete the work as quickly as possible. This algorithm is based on the branching algorithm approach for searching. The experimental results for the scheduling problem were calculated with large data. From that determines the threshold of input data of the problem in order to the computation time is minimum.**

***Index Terms*— Schedule, Job-shop Problem, Parallel, Distributed System and GPU**

I. INTRODUCTION

ACCORDING to Supercomputer Site (http://www.top500.org), the world's fastest supercomputer was Titan supercomputer, a Cray XK7 system installed at Oak Ridge, achieved 17.59 Petaflop/s (quadrillions of calculations per second) on the Linpack benchmark. It has 560,640 processors, including 261,632 NVIDIA K20x accelerator cores. This is a computer network includes more powerful CPUs, and the GPUs to speed up of computations. From that, there is a supercomputer with distributed memory architecture on multiple computers, and the memory was shared between the CPU and GPU on a single computer. Not only Titan, in the most of nowadays supercomputer there are architectures like this. So that, the building of parallel algorithms running on the distributed shared memory system is indispensable problem.

In this paper, the parallel algorithm of Job Shop Problem (JSP) was established to implement in distributed shared memory architecture. The schedule is formed when human activities need to be assigned tasks, and the tendency still persists. A good schedule would help to reduce time for completing the job, thereby saving time. So spent decades, there are many researchers have continuously study and develop the scheduling method to get a best solution.

The JSP is a problem in discrete or combinatorial optimization and, it is a generalization of the famous TSP (Travelling Salesman Problem). So that, it is NP-hard problem. It could be established as follow:

Let $M = \{M_1, M_2, \dots, M_m\}$ be finite set with m elements, where $M_i, \forall i = 1,\dots,m$ is called machine. And $J = \{J_1, J_2, \dots, J_n\}$ be n-element set, where $J_j, \forall j = 1, \dots, n$ is called job. The scheudule problem is sequential assignment of jobs to machines such that:

- every job is done by every machine exactly once
- there are maximum *m* segments in every job
- if a job has started processing, then it cannot be interrupted
- each job is processed on machines in a certain sequence

Let's define all sequences of jobs to machines, such that the completion time is minimum.

The Job Shop Problem can be applied for different purposes. For example, the employee works scheduling on a company. With this requirement, it could be described as follows: the employee part of the company consists of a set of staff and a set of work is done in the specified time. Each employee has the ability to perform work depending on the preferences as well as the time to work. Scheduling is to assign work to an employee such that ensures the work must be completed on schedule. With the above example, the Job shop problem can be applied to other types of scheduling problems such as setting up a schedule for students, assignment shifts of nurses in hospitals, distributing of teaching schedule in a training center, assignment driver of a transport company...

Trest of paper was organized as follows. The section 2 describes the related work of solving the JSP. The method to solve problem was presented in section 3. In the section 4, some experimental results were presented, and the section 5 concludes and describes some future work.

II. RELATED WORK

Currently, there are two approach methods for solving the JSP: approximate method and exact method. The research papers on the world in recent years were using approximate method. Using this method would build the algorithm that runs much faster than the algorithm by exact method, but the major drawback is the resulting solution cannot be sure. Meanwhile, the computation time of exact methods is very slow, but their results are always optimal. So that, the parallelization of algorithms to reduce the execution speed are often use for building algorithm of JSP. Most of the algorithms used in this papers are taken the basis of approximation algorithms such as taboo search, simulated annealing, ... For example, in paper [6] "*A Branch and Bound and Simulated Annealing Approach for Job Shop Scheduling*" of Tan Hui Woon and Sitinah Salim was published in 2004, authors used Branch and Bound methods and Simulated Annealing Algorithm for solving Job shop scheduling. This paper also demonstrated the simulated annealing algorithm accomplished with fewer steps. However, the major disadvantage of the simulated annealing algorithm is implemented to base on random factors. For example, the algorithm starts with a random initial schedule, and then the next schedule is randomly generated. So that, using of these parameters can give better results on a specific problem, but on other problem with same parameters, its good results are not sure.

In the paper [5], "*Parallel Simulated Annealing Algorithms*", D. Janaki Ram, T. H. Sreenivas, and K. Ganapathy Subramaniam have presented the parallel-simulated annealing algorithm to overcome the drawback of slow convergence of the algorithm. This paper has proposed two algorithms to solve; the first algorithm called the clustering algorithm (CA), and second called genetic clustering algorithm (GCA). By using the parallel algorithm combining simulated annealing and genetic algorithms, so it has somewhat improved execution speed from the classical simulated annealing algorithm. However, in essence it is still speculative search algorithm; therefore results are not proven to be the best.

The paper [2] "*Using Genetic Algorithm for solving of Job Shop Problem*", and [3] "*The Hybrid Genetic Algorithm for Job Shop Problem*" N.H. Mui and V.D. Hoa, presented the new genetic algorithm for solving the JSP. In this algorithm, authors proposed a new crossover operator combining on three parents individual. With this method, children individual were born by a crossover operator is an active schedule. The convergence of this algorithm is also demonstrated based on a theorem of Banach stability. However, the experimental results of this study show that in the case input data of problem are small size, then algorithm achieves optimal results with a high rate; but the input data are larger, then obtained results were at only near optimal level. In case with 20 jobs and 5 machines, the algorithm cannot find the optimal schedule.

Towards to the approach mixed algorithms, the paper [1] also gives some positive results. But it still did not fully solve this problem.

The above works have the advantage that the completion time of the algorithm is fast, but the major drawback that the schedule was generated is not optimal schedule. Therefore, the use of an algorithm using exact approach to create a good schedule and complete in a reasonable period of time is the problem of this paper.

III. Improved and Parallel Algorithm

A. The Branch and Bound Algorithm

There are two main algorithms in the branch and bound algorithm [6]: The branch algorithm to create hierarchy tree and algorithm for determining the bound. After determination of the bounds of the tree nodes, the algorithm can be based on these bounds to identify which branch can give the optimal results.

The branch algorithm ([6]):

The algorithm given is based on the branching scheme. The nodes of the branching tree are corresponding to the partial schedules.

- Step 1: (Initial condition): The algorithm initiates a set Ω, which is the first node is expanded from the source node.
    - Ω : = { Initial segments of each job }
    - r_{ij} := 0 for all $(i,j) \in \Omega$
- Step 2: (Machine selection): Choose a machine of set Ω on which completion time $r_{ij} + P_{ij}$ is the minimum
    - Compute $t(\Omega)$ for current partial schedule.
    - $t(\Omega) := \min \{r_{ij} + P_{ij}\}, (i,j) \in \Omega$
    - i^* := machine such that $r_{i^*j} + P_{i^*j}$ is the minimum.
- Step 3: (Branching)
    - $\Omega' := \{ (i^*j) \mid r_{i^*j} < t(\Omega) \}$
    - For all $(i^*,j) \in \Omega'$, extend a partial schedule by scheduling (i^*,j) next on machine i^*
    - For each such choice,
        - find a bound of (i^*,j)
        - delete (i^*,j) from Ω.
    - Add segment successor of (i^*,j) to Ω.
    - Return to Step 2 until $\Omega = \emptyset$ with all elements of Ω′

The bound algorithm:

The goal of algorithm is to find a lower bound of the makespan, this is a NP-hard problem. The largest makespan obtained by algorithm in [6] can be used as the lower bound. In there, continuing the branch and bound procedure to obtain the makespan, which is corresponding to the minimum, lower bound.

B. The Improved Branch and Bound Algorithm

In traditional branch and bound algorithm, the set Ω′ was created from set Ω based on condition $r_{i^*j} < t(\Omega)$ to remove the branches that are not completely feasible is not enough. Hence, it need add some conditions to remove the feasible branching. In this improved algorithm, lower bound condition min LB was used to remove in the branch algorithm:

The improved branch algorithm:

- Step 1: (Initial condition): The algorithm initiates a

```
    set Ω, which is the first node is expanded from the
    source node.
        o  Ω : = { Initial segments of each job }
        o  $r_{ij}$ := 0 for all $(i,j) \in \Omega$
        o  min LB (Lower Bound) = +∞
•   Step 2: (Machine selection): Choose a machine of set
    Ω on which completion time $r_{ij} + P_{ij}$ is the minimum
        o  Compute $t(\Omega)$ for current partial schedule.
        o  $t(\Omega) := \min \{r_{ij} + P_{ij}\}, (i,j) \in \Omega$
        o  $i^*$ := machine such that $r_{i^*j} + P_{i^*j}$ is
           minimum.
•   Step 3: (Branching)
        o  Step 3.1: $\Omega' := \{ (i^*j) | r_{i^*j} < t(\Omega) \}$
        o  Step 3.2: For all $(i^*,j) \in \Omega'$, extend a partial
           schedule by scheduling $(i^*,j)$ on next
           machine $i^*$
        o  Step 3.3: For each such choice, expand the
           branch and find a bound of $(i^*,j)$:
               ▪  If min LB < bound of $(i^*,j)$ then
                      •  jump this branch
                      •  return Step 3.2
               ▪  Else
                      •  min LB = bound of $(i^*,j)$
        o  Step 3.4: Delete $(i^*,j)$ from Ω.
        o  Step 3.5: Add segment successor of $(i^*,j)$ to
           Ω.
        o  Step 3.6: Return to Step 2 until Ω = ∅ with
           all elements of Ω'
```

C. The Parallel Algorithms with Distributed-Memory Architecture

The parallel algorithm is based on expansion of the branches in the branch and bound algorithm. During the process of the branch expansion, these branches would be expanded simultaneously from different tasks. After these tasks have done, they would send results to the master. The parallel algorithm as follows:

Master

```
•   Step 1: Ω : = { Initial segments of each job }
•   Step 2: Send Ω to Slaves
•   Step 3: $r_{ij}$ := 0 for all $(i,j) \in \Omega$
•   Step 4: $t(\Omega) := \min \{r_{ij} + P_{ij}\}, (i,j) \in \Omega$
        o  Select machine $i^*$ such that $r_{ij} + P_{ij}$ is
           minimum.
        o  $\Omega' := \{ (i^*j) | r_{i^*j} < t(\Omega) \}$
•   Step 5: Partition set Ω' into subset Ω'' and send
    them to Slaves
•   Step 6: Receive results from Slaves and store the
    optimal result.
```

Slave

```
•   Step 1:
        o  Receive Ω from Master
        o  min LB = +∞
•   Step 2: Receive Ω'' from Master
        o  Step 2.1: For all $(i^*,j) \in \Omega'$, extend the
           partial schedule by scheduling $(i^*,j)$ on next
           machine $i^*$
        o  Step 2.2: For each such choice, expand the
           branch and find a bound of $(i^*,j)$ of machine
           $i^*$:
               ▪  If min LB < bound of $(i^*,j)$ then
                      •  jump this branch
                      •  return Step 2.1
               ▪  Else
                      •  min LB = bound of $(i^*,j)$
        o  Step 2.3: Delete $(i^*,j)$ from Ω
        o  Step 2.4: Add segment successor of $(i^*,j)$ to
           Ω
        o  Step 2.5: : Return to Step 2 until Ω = ∅ with
           all elements of Ω'
•   Step 3: Send result to Master
```

D. The Parallel Algorithms with Shared-Memory Architecture

In GPU-CPU environment, data were commonly used in all processes. Hence, in the improved branch and bound algorithm, some follow statements were run concurrently on GPU:

- Finding the minimum t of set Ω: $t(\Omega) = \min\{r_{ij} + P_{ij}\}$
- Finding the machine t^* such that $r_{i^*j} + P_{i^*j} = t$
- For all segments $(i^*, j) \in \Omega'$, extend partial schedule by scheduling (i^*,j) on next machine i^*

The parallel algorithm on GPU-CPU as follows:

On the CPU

```
•   Step 1: (Initial condition)
        o  Ω : = { Initial segments of each job }
        o  $r_{ij}$ := 0 for all $(i,j) \in \Omega$
        o  min LB (Lower Bound) = +∞
•   Step 2: (Machine selection)
        o  Invoke CUDA_SelectMachine (Ω, t, i*)
•   Step 3: (Branching)
        o  Step 3.1:
               ▪  Invoke CUDA_ ExtendSchedule (Ω,
                  Ω',i*)
        o  Step 3.2: For each such choice, expand the
           branch and find a bound of $(i^*,j)$:
               ▪  If min LB < bound of $(i^*,j)$ then
                      •  jump this branch
                      •  return Step 3.2
               ▪  Else
                      •  min LB = bound of $(i^*,j)$
        o  Step 3.3: Delete $(i^*,j)$ from Ω.
        o  Step 3.4: Add segment successor of $(i^*,j)$ to
           Ω.
        o  Step 3.5: Return to Step 2 until Ω = ∅ with
           all elements of Ω'
```

On the GPU

The program run on GPU would init all the processes, and each process would calculate the value $r_{ij} + p_{ij}$ concurrently.

These processes also compare the values $r_{ij} + p_{ij}$ to find minimum, from that select machine i^*.

```
• Step 2 (Fig. 1): CUDA_MachineSelection( Ω, t, i* )
    o tib := getThreadID
    o if $t > r[tib] + p[tib]$ then
        ▪ $t := r[tib] + p[tib]$
        ▪ $i^*$ := machine such that $t$ is minimum
```

With each segment on set Ω' of a job that run on machine i^*, the processes of GPU would find the rest segments to add the new path into schedule (Fig. 2).

```
• Step 3.1: CUDA_ ExtendSchedule (Ω, Ω',i*)
    o tib := getThreadID
    o if $i^*$ = machine executed operation[tib] of set Ω then
        ▪ add operation[tib] to Ω'
        ▪ extend a partial schedule by scheduling the machine executed operation[tib] of set Ω'
```

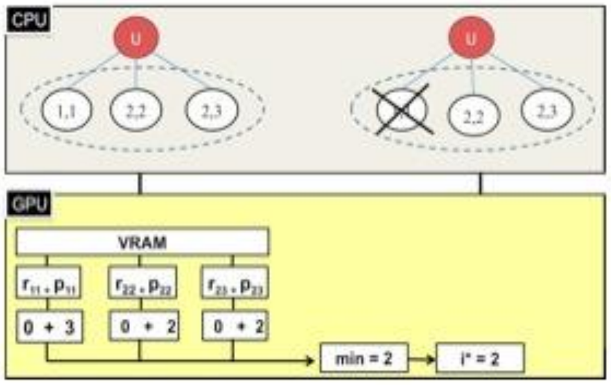

Figure 1. Finding the minimum and machine i^* on GPU

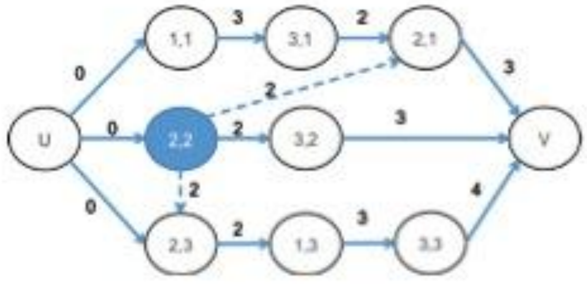

Figure 2. Extending schedule on disjunctive graph

IV. Experimental results

A. Using the Multi-Computers

Using the parallel algorithms with 3 processes in the MPI Environment on computer with 256Mb RAM. The algorithm was implemented by Language C using Libarary MPI. The experimental results are dicribled in Table 1 and Figure 3.

Table 1. Comparing time (in second) of 3 algorithms using MPI

Job/Machine	*The classical algorithm*	*Improved algorithm*	*Parallel algorithm*
3/3	0.005532	0.003456	0.439290
4/4	0.123183	0.058664	0.535645
5/5	3.471250	0.490766	0.912148
6/6	141.143496	9.916246	5.028870
7/7	*	77.56584	21.485825
8/8	*	89.629005	55,020864
9/9	*	141.547706	71.993723
10/10	*	477.874597	389.455488

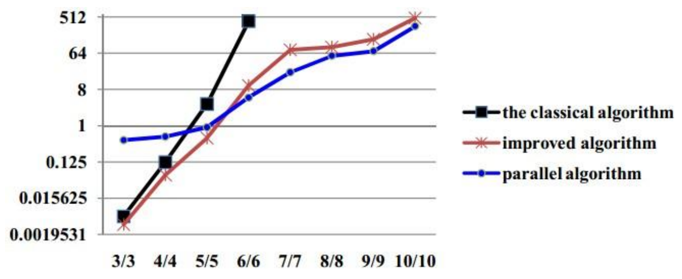

Figure 3. The graphs on computation time of algorithms

With mention results, the execution time of the improved sequence algorithm is faster than time of the traditional algorithm. Besides, in the case of large data, the parallel algorithm is always best algorithm.

B. Using GPU – CPU with nVIDIA Card

The deployment environment for testing as follows:

- Computer: Intel core 2 Dual 2.66 GHz, 2GB RAM
- Graphic Card: GeForce GTX 250

The algorithm was implemented by Language C using Library CUDA. The experimental results are dicribled in Table 2 and Figure 4.

Table 2. Comparing time (in second) of 3 algorithms using CUDA

Job/Mach.	Improved Alg.	Alg. using GPU	Alg. using MPI
3/3	0.003456	0.218000	0.779147
4/4	0.058664	0.718000	9.377534
5/5	0.490766	1.560000	37.634233
6/6	9.916246	6.833000	369.530780
7/7	77.56584	27.75200	1663.936657
8/8	89.629005	41.71400	*
9/9	141.547706	57.56400	*
10/10	477.874597	59.17100	*

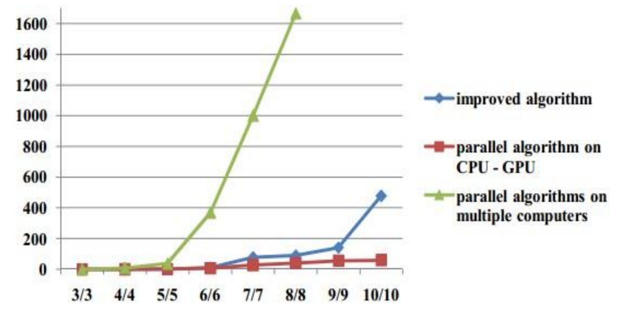

Figure 4. The graphs on computation time with CUDA

Comment:

- The improved sequential algorithm is better than the parallel algorithm with small input data. But in the case of large data input, the computation time of the improved sequential algorithm is slower than parallel algorithm's one.
- On GPU-CPU environment, this algorithm would execute parallelly at the time that needs to handle data (step 2, step 3.1), instead of processing the elements of data array in succession. These elements were run concurrently on different processes, and the value of them would be calculated many times at different periods. Thus, if the data elements are processed on a computer network environment, then they need to be divided into several parts and sent to the slaves to handle. After slaves have finished, the result would be sent back to the server computer via transmission network line. So that, the calculation time the values of data array are proportional to communication time, this would take a lot of time. Hence, the algorithm described in section *D* only effects on CPU-GPU environment, and no effect on the computer network environment (multicomputer).

V. Conclusion

The improved branch and bound algorithm in this paper is very effective for JSP problem. Especially when this problem was implemented on the shared memory architecture using GPU. In JSP problem, this improved algorithm removes many feasible branches; so finding the job schedule is faster than the classical algorithm. In this paper, the improved branch and bound algorithm also were changed to develop into 2 parallel algorithms implemented on a computer network, as well as on computer with graphic card as nVIDIA (GPU). The experimental results show that computing environments with shared memory when applied to the JSP problem using the branch and bound algorithm.

References

[1] Rui Zang, Cheng Wu (2010), *A hybrid approach to large-scale job shop scheduling*, Applied Intelligence, Volume 32, Number 1, pp. 47 – 59.

[2] Nguyen Huu Mui, Vu Dinh Hoa (2010), *Using Genetic Algorithm for solving of Job Shop Problem*, Proceeding of The 15th Vietnam National Conference "*Some Selective Problems on Information and Communication Technology*", Hung Yen, 19 - 20 Aug, 2010, Sci and Tech Pub. House, pp. 71 - 82.

[3] Nguyen Huu Mui, Vu Dinh Hoa (2011), *The Hybrid Genetic Algorithm for Job Shop Problem*, Proceeding of The 5th Vietnam National Conference on Fundamental and Applied Information Technology Research, Dong Nai, 11-12 August, 2011. Sci and Tech Pub. House, pp. .239 - 249

[4] Brian Patrick Ivers (2003), *Job shop optmization through multiple independent particle swarms*, Thesis of Bachelor of Science in Electrical Engineering, Oklahoma State University.

[5] D. Janaki Ram, T. H. Sreenivas, K. Ganapathy Subramaniam (1996), *Parallel Simulated Annealing Algorithms*", Journal of Parallel and Distributed Computing, V.37, No.0121, pp. 207–212.

[6] Tan Hui Woon, Sutinah Salim (2004), *A Branch and Bound and Simulated Annealing Approach for Job Shop Scheduling*, Matematika, Jabatan Matematik, University of Teknologi Malaysia, Jilid 20, bil.1, hlm. pp. 1–17.