Posterior Task Scheduling Algorithms for Heterogeneous Computing Systems

Linshan Shen[1], Tae-Young Choe[1]

[1]School of Computer Engineering, Kumoh National Institute of Technology, Yangho-dong, 730-701 Kumi city, KyungPook, Korea

smart781005@hotmail.com

choety@kumoh.ac.kr

Abstract. The task scheduling problem in heterogeneous system is known as NP-hard. Recently, Bajaj and Agrawal proposed an algorithm TANH (Task duplication-based scheduling Algorithm for Network of Heterogeneous systems) with optimality conditions, which are wider than previous optimality conditions. TANH algorithm combines the clustering technique with task duplication. We propose two postprocessing algorithms, HPSA1 (Heterogeneous Posterior Scheduling Algorithm) and HPSA2, to reduce the schedule length for DAGs which don't satisfy the optimality conditions of TANH algorithm. Our algorithms reduce the schedule length by exchanging task clusters in which its parent tasks reside. We compare with HCNF (Heterogeneous Critical Node First) algorithm by illustrating an example to show how our algorithms operate.

Keywords: Heterogeneous system, DAG, task scheduling, postprocessing, clustering.

1 Introduction

For high-speed computation purposes, parallel processing has been extensively explored. Some applications like fluid flow, image processing, weather modeling, and distributed database systems get a great deal of parallelism. A general methodology adopted in parallel processing is to partition an application into a set of cohesive tasks and to run them separately on different processors. The partitioned application can be modeled as a directed acyclic graph (DAG). In DAGs, a forward edge means that the predecessor task transmits the data to the successor task.

The task scheduling problem is to allocate tasks to processors in order to minimize the completion time of given application which can be expressed as a DAG. The task

scheduling problem is known as NP-hard [1,2]. Therefore, heuristic task scheduling algorithms are used to tackle the problem. The scheduling algorithms have been extensively studied [3-19]. These heuristics are classified into a variety of categories such as, list scheduling algorithms, clustering algorithms, duplication-based algorithms and guided random search methods. Most of them are designed for homogeneous computing systems.

In the classical list scheduling algorithms [3,4,5,19], tasks are assigned priorities statically or dynamically. The priorities are assigned based on computation and communication costs in the task graph. The next chosen is the highest priority task among the ready tasks whose precedence have been met and are ready for scheduling. The following step is to select most suitable processor to accommodate the chosen task. Performances of list scheduling algorithms tend to suffer due to min-max problem and deteriorate substantially for fine grain task graphs having high communication to computation cost ratio (CCR) [18].

Clustering-based algorithms [7,8,9] try to schedule heavily communicating tasks onto the same processor. It is also known as three phase scheduling. In the first phase, heavily communicating tasks are grouped into a set of clusters (unbounded) using linear or nonlinear clustering heuristics. In the second phase, clusters are mapped onto the set of available processors using communication sensitive or insensitive heuristics. In the third phase cluster merging is done based on the available number of processors.

Duplication-based algorithms allow tasks to be duplicated on one or more than processors, in order to reduce the start time of its successor tasks. Now, the duplication-based algorithms have been blended with both list and clustering-based techniques by other researchers. List or clustering-based algorithms [5,6,10-13] with task duplication tend to perform better than no duplication algorithms. Genetic algorithms [14,15] are of the most widely studied guided random search techniques for the task scheduling problem. Although they provide good quality of schedules, their execution times are significantly higher than other alternatives.

The processors in heterogeneous systems have different processing powers, so scheduling them is more complex. In recent years, many scheduling algorithms for the heterogeneous system are proposed, such as Levelized Duplication Based Scheduling (LDBS) [10], Dynamic Level Scheduling (DLS) [4], Task Duplication-based Scheduling Algorithm for Network of Heterogeneous System (TANH) [13], Fast Critical Path (FCP) and Fast Load Balancing (FLB) [5]，Task Duplication Scheduling (TDS-1) [16], Heterogeneous Earliest Finish Time First (HEFT) [17].

In this paper, we propose two posterior algorithms, HPSA1 (Heterogeneous Posterior Scheduling Algorithm) and HPSA2 for heterogeneous computing system.

The main motivation is to improve the quality of the scheduling length, while DAGs don't satisfy TANH algorithm's optimality conditions. The algorithms run at the hind of TANH algorithm.

In the next section, we define the parameters and data structures served for our algorithms. In section 3, we briefly introduce TANH and HCNF algorithms. In section 4, we propose our two algorithms. In section 5, we illustrate an example to show that our algorithms how to work. In section 6, we discuss the experimental results. In final part, we present conclusions.

2 Problem Definition

We consider any application that is represented as a directed acyclic graph (DAG). In DAG, each node represents a task and each directed edge represents communication cost between tasks. A task is assumed to be nonpreemptive. A tuple G = (V, E, P, w, c) is used to define a DAG, where V is a set of nodes, |V| is the number of nodes in V; E is a set of edges; P is a set of processors, |P| is the number of available processors; $w = w(n_i, p_i)$ indicates the computation cost of task n_i on processor p_i, where $n_i \in V$ and $p_i \in P$; $c = c(n_i, n_j)$ indicates the communication cost between task n_i and n_j, where $n_i, n_j \in V$. If both n_i and n_j are scheduled onto the same processor, $c(n_i, n_j)$ is assumed to be zero. In the other hand, the network bandwidth is assumed to be wide enough to provide contention-free transmission. Given a task n_j, $pred(n_j)$ is a set of predecessor tasks which have the outgoing edge into n_j; $succ(n_j)$ consists of the tasks which receives the data from n_j. And $n_i = fpred(n_j)$ denotes the favorite predecessor, which means among all the predecessor tasks of n_j, n_i has the highest value of the earliest finish time. The earliest start/finish time indicates when a task could be started/finished at the earliest possible time. Arrival time of task n_j equals the sum of the earliest finish time of the parent tasks which are not in the same processor with n_j and the communication time between the parent task and n_j.

A set of tasks assigned to a processor is called a cluster. Each task in a cluster has its start time and finish time in the corresponding processor. A task n_j at processor p_k has its start time $st(n_j, p_k)$ and finish time $ft(n_j, p_k)$. In order to compute $st(n_j, p_k)$ and $ft(n_j, p_k)$ for task n_j at processor p_k, we need to introduce several equations:

$$st(n_j, p_k) = \max[ft(p_k), rdy(p_k, n_j)] \quad (1)$$

$$ft(n_j, p_k) = st(n_j, p_k) + w(n_j, p_k) \quad (2)$$

$$rdy(p_k, n_j) = \max_{n_i \in pred(n_j)} [\min_{n_i \in p_i, p_i \neq p_j} [ft(n_i, p_i)] + c(n_i, n_j)] \quad (3)$$

where ft(p_k) represents the current finish time of processor k, and rdy(p_k ,n_j) represents the largest ready time from parent tasks allocated in other processors.

3 Related Works

In recent past, some algorithms combined clustering technique with duplication have been proposed. The TANH algorithm [13] just belongs to them, its time complexity is $|V|^2$. The algorithm firstly generates the initial clusters. If the number of required processors (RP) is larger than the number of available processors (AP), then using compaction procedure selects two processors and merges them to one processor until RP equals AP. Bajaj and Agrawal proposed TANH algorithm and presented a set of optimal conditions for join nodes. Unfortunately, when an application DAG doesn't satisfy the optimal conditions, the expected result is difficult to be gained.

HCNF [19] is list scheduling algorithm, it proceeds by identifying the critical path in the DAG. The critical path is a path which has the largest sum of average task computation costs and inter-task communication costs among all the paths from the entry task to the exit task. Task is free when its predecessor tasks have been assigned to processors. A list of free tasks is constructed. Within the list, the highest priorities are assigned to tasks which fall in the critical path, followed by those with the highest computation cost. A task n_i is scheduled onto processor P_i, which gives the lowest eft(n_i , P_i). Its earliest finish time eft(n_i , P_i) = w(n_i , P_i)+est(n_i , P_i), where the earliest start time est(n_i , P_i) is the maximum of time at which processor P_i becomes available and the time at which the last message arrives from any predecessor of n_i, and w(n_i , P_i) is the execution time of n_i on P_i . In order to reduce the communication time, the favorite predecessor is considered for duplication.

4 Proposed Algorithms

The main motivation of our algorithms is to reduce the finish time of tasks which determine the scheduling length by exchanging its parent clusters. If the finish time of the task can be reduced, the scheduling length may be reduced. Thus, two posterior processes, HPSA1 and HPSA2, are proposed to improve the scheduling length.

The proposed two algorithms execute at the hind of the general scheduling algorithms, as shown in Fig. 1.

4.1 Selection of Scheduling Algorithm

In this paper, we select TANH algorithm as the scheduling algorithm instead of HCNF.

The researchers of TANH algorithm proposed a set of optimal conditions, if DAGs satisfied the optimal conditions, TANH algorithm got the optimal schedule, when our algorithms executed at the hind of TANH algorithm, the optimal schedule was also got, but it was not sure to get optimal schedule when HCNF algorithm was used. If DAGs didn't satisfy the optimal conditions, it was not sure to get optimal schedule when TANH was used to schedule, but combining our algorithms to TANH can make the schedule length less than or equal to that of TANH, and HCNF also got the unsure optimal schedule. So the combination of our algorithms and TANH is used in this paper. The comparisons are shown in Table 1.

```
Main( )
 {
    ...
    Scheduling algorithm ( );
    Posterior scheduling algorithm;
 }
```

Fig. 1. The position of the posterior algorithm.

Table 1. The comparisons of TANH, TANH+HPSA, and HCNF

Task scheduling algorithm	DAG (directed a-cyclic graph)	
	It satisfies the optimal conditions of TANH	It doesn't satisfy the optimal conditions of TANH
TANH	Optimal schedule	Not sure
TANH+HPSA	Optimal schedule	The schedule length<=the schedule length by TANH
HCNF	Not sure	Not sure

Our algorithms execute at the hind of TANH algorithm, only when exchanging operation can make the schedule length reduce, the operation is done, if no improvement is gained, the operation will be canceled. So the combination of our algorithms and TANH algorithm can reduce the schedule length given by TANH algorithm. Otherwise the schedule length is preserved.

4.2 HPSA1 Algorithm

The main idea of this algorithm is to reduce the start time of target task at its processor, and reduce the scheduling length. Firstly, the exit node is target task, and its predecessors are checked, which predecessor determines the start time of the target

task, and exchange the cluster including the decided task and the other parent tasks' clusters of the target task, respectively. Which exchange made the start time of the target reduce most is selected as the right exchange. For the current target, next passes are done, until no improvement is gained. Checking the current schedule, the predecessor task that determines the start time of the target task is selected as the next target task, repeat the above steps, until no improvement is gained. The pseudocode for HPSA1 algorithm is described in the Fig. 2. The time complexity of HPSA1 is $|V|^2|P|^2 \max_{n_i \in V} |succ(n_i)|$.

```
Void HPSA1( )
{
  n_t =the exit task;
  do{
      do{
           n_p = parent task that determines the start time
                 of n_t;
           n_q = ParentTask1(n_t ,n_p);
           exchange clusters of n_p and n_q;
         } while (start time of n_t reduces);
         n_t = ParentTask2(n_t);
    }while(there is an improvement);
}
```

Fig. 2. The psedocode of HPSA1. Function *ParentTask1*(n_t, n_p) is responsible for finding another parent task n_q of n_t that minimizes the start time of n_t by exchanging clusters of n_p and n_q each other. Function *ParentTask2*(n_t,) is responsible for finding the parent task of n_t which determines the start time of n_t

4.3 HPSA2 Algorithm

For n_i and n_j in some processor p_i, an empty slot exits if the finish time of n_i in p_i is smaller than the start time of n_j in p_i. After a schedule, it is highly probable that the empty slots exist. If an empty slot not only has the largest size among all the empty slots, but also affects the schedule length, then that is more improvement on the scheduling length when we try to exchange the clusters that the predecessors accommodate and the cluster above the slot. The biggest reduction cluster that its exchange makes the scheduling length reduce most is selected as the right exchange. After the operation to the slot which has biggest size, then consider the slot with the

second biggest size and affecting the scheduling length.

```
Void  HPSA2 ( )
{
   Assign every task as -1;
   do {
        Find all empty slots;
        Generate the path;
        n_t= Findtail();
        for each predecessor n_p of n_t
            n_p =  FindTask(n_t);
        exchange clusters of n_t and n_p ;
        Assign n_t  as 0;
       } while (there is an improvement)
}
```

Fig. 3. The psedocode of HPSA2. Function *FindTask*(n_t) is responsible for finding a task which is neither at the same cluster with n_t , nor with the exit task, and maximum reduction for the slot is obtained by exchanging this cluster and the cluster that n_t resides. Function *Findtail*() is responsible for finding the tail task of an empty slot that has the biggest size and affects the scheduling length

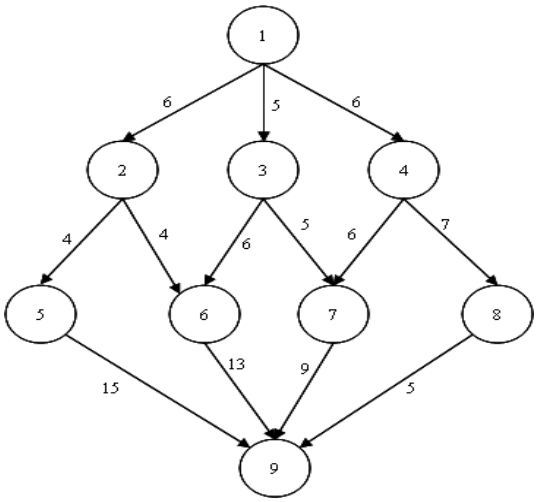
Each slot has its information: head task, tail task, and slot size. In order to know which slot affects the scheduling length, we need to check the tail task whether affects the scheduling length. We use a queue path to save tasks that determine the scheduling length from the exit task, and write down the slot size related to the tasks. If a task in path has only one predecessor, then also adds this predecessor task to the path, and the slot size is zero.

The psedocode of HPSA2 is illustrated in the Fig. 3. It marks all the tasks. If the slot related with this task was operated, this task is unmarked. Next, it memories the current scheduling length, which can be used to compare the scheduling length after exchanging the clusters that make the slot reduce most. The time complexity of HPSA2 is $|V|^2|P|^2$.

5 Illustration of an Example

In this part, we illustrate an example to show combining our two posterior processes

to TANH is efficient for improving the schedule of TANH algorithm, and their performances are better than HCNF algorithm. The example DAG is illustrated in the Fig. 4a, and the computation cost of tasks at every processor is illustrated in the Fig. 4b. We get the initial schedule as the Fig. 5 after TANH algorithm. We get the schedule as the Fig. 6 after the HCNF algorithm.



(a)

tasks	P_1	P_2	P_3	P_4
1	5	6	9	7
2	7	7	9	8
3	11	7	9	9
4	8	7	10	13
5	10	3	6	9
6	7	10	4	12
7	11	9	8	12
8	10	11	7	6
9	6	4	9	3

(b)

Fig. 4. (a) An example DAG G_1. (b) Runtime of tasks for G_1.

5.1 Using HPSA1 to Post-process

In the Fig. 5, firstly, the target is task 9, task 7 determines the start time of task 9. By exchanging cluster {1, 4, 7} and cluster {1, 3, 6}, {1, 4, 7} and {1, 4, 8}, {1, 4, 7} and {1, 2, 5}, we know that the exchange of cluster {1, 4, 7} and cluster {1, 3, 6} makes the start time of task 9 reduce most. We get a schedule of Fig. 7. When the start time of task 9 was reduced, next passes will be done, but no improvement is gained. For the current schedule, task 6 and task 7 are considered for exchanging, because they determine the start time of the task 9. However, neither of them as the target improves the schedule length.

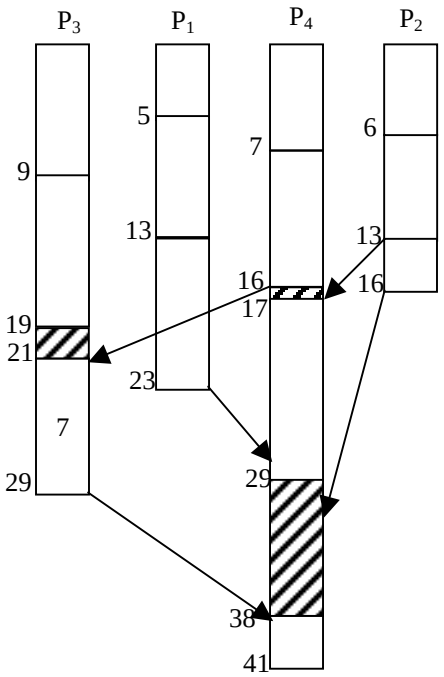


Fig. 5. The schedule S_1 of DAG G_1 by TANH algorithm.

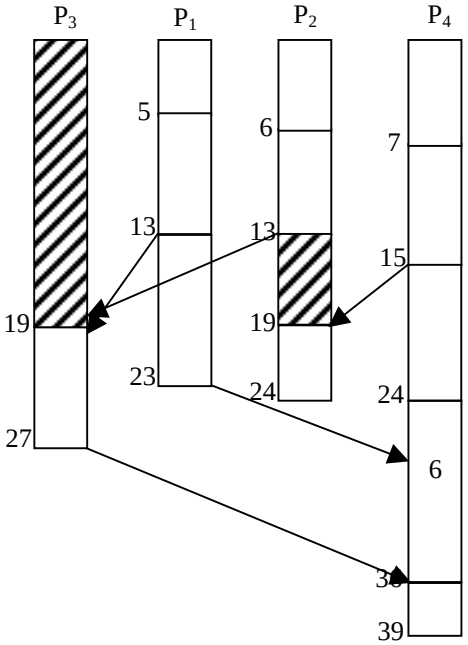


Fig. 6. The schedule S_2 of DAG G_1 by HCNF algorithm.

5.2 Using HPSA2 to Post-process

Firstly, a path is created, in which contain the tasks determine the scheduling length of the schedule in Fig. 5. We illustrate tasks in the path 9-7-3-1 in accordance with the size of slots which related to them in Table 2. From the table, the empty slot related to task 9 has the biggest size and affects the schedule length. Exchanges are executed, cluster {1, 3, 6} and cluster {1, 4, 7}, {1, 3, 6} and {1, 4, 8}, {1, 3, 6} and {1, 2, 5}, exchanging {1, 3, 6} and {1, 4, 7} has most reduction for the slot above task 9, the schedule as the Fig. 7. Next, only one slot for task 7, we operate an exchange: {1, 4}⇔{1, 3}, but no improvement is gained, HPSA2 ends.

Table 2. The Path of affecting the scheduling length of S_1.

task	9	7	3	1
Size of slot	9	2	0	0

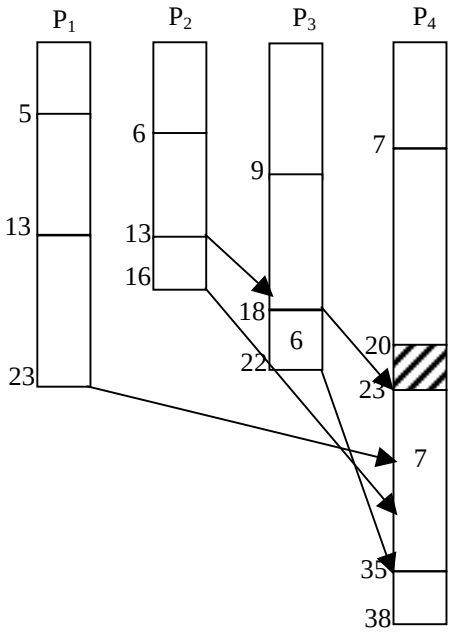


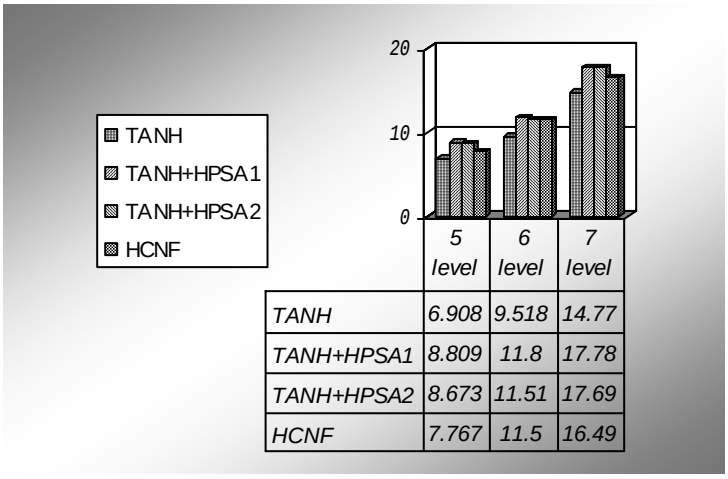
Fig. 7. First exchange of S_1 when HPSA1 is applied and first exchange of S_1 when HPSA2 is applied.

6 Experimental Results

To study the performances of our algorithms in execution time, a random DAG generator is designed by ourselves. The generator requires the following input parameters: i) graph level GL, ii) the number of processors NoP, iii) the maximal outdegree of task is 3. Without loss of generality, all DAGs are required a single entry node and an exit node. The computation time is selected randomly from 3 to 7, and the communication time is selected randomly from 2 to 6.

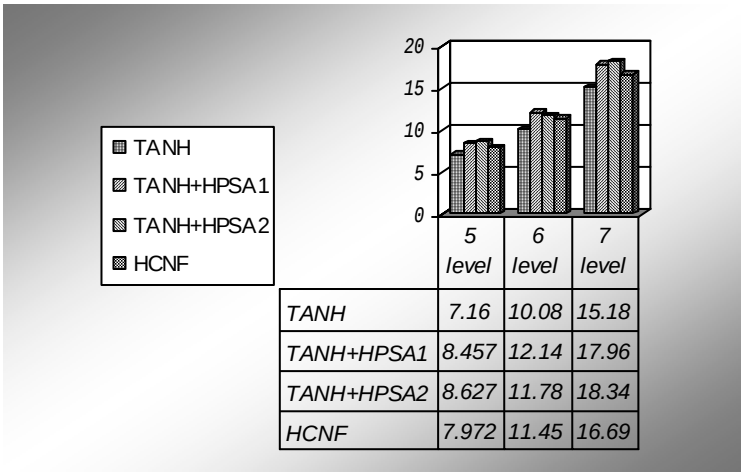
We did two sets of experiments: i) fork tasks in DAG have the same computation cost, ii) fork tasks in DAG have -1~1 different computation cost. Every set of experiments use 10 random generated DAGs when GL=5, 6, 7 in accordance with NoP=6, 8, 9, respectively. We compare the average execution times (msec) of TANH, TANH+HPSA, and HCNF. The experimental results are shown in Fig. 8 and Fig. 9, respectively.

For the first set of experiments, the sum of schedule length of TANH+HPSA1 has 3.82%, 1.79%, and 2.98% decrements than that of TANH when GL=5, 6, 7 in accordance with NoP=6, 8, 9. The sum of schedule length of TANH+HPSA2 has 4.36%, 1.79%, and 2.98% decrements than that of TANH when GL=5, 6, 7 in accordance with NoP=6, 8, 9. To the same DAGs, in the case of HCNF, the sum of schedule length of TANH+HPSA1 has 1.67%, 4.14%, and -1.68% decrements than that of HCNF. The sum of schedule length of TANH+HPSA2 has 2.23%, 4.14%, and -1.68% decrements than that of HCNF.



	5 level	6 level	7 level
TANH	6.908	9.518	14.77
TANH+HPSA1	8.809	11.8	17.78
TANH+HPSA2	8.673	11.51	17.69
HCNF	7.767	11.5	16.49

Fig. 8. The execution time comparison for fork tasks with the same computation time in DAG.



	5 level	6 level	7 level
TANH	7.16	10.08	15.18
TANH+HPSA1	8.457	12.14	17.96
TANH+HPSA2	8.627	11.78	18.34
HCNF	7.972	11.45	16.69

Fig. 9. The execution time comparison for fork tasks with -1~1 different computation time in DAG.

For the second set of experiments, the sum of schedule length of TANH+HPSA1 has 2.96%, 0.83%, and 1.73% decrements than that of TANH when GL=5, 6, 7 in accordance with NoP=6, 8, 9. The sum of schedule length of TANH+HPSA2 has 3.5%, 0.66%, and 2.25% decrements than that of TANH when GL=5, 6, 7 in accordance with NoP=6, 8, 9. To the same DAGs, in the case of HCNF, the sum of schedule length of TANH+HPSA1 has -5.56%, 2.18%, and -6.98% decrements than that of HCNF. The sum of schedule length of TANH+HPSA2 has -4.97%, 1.96%, and -6.42% decrements than that of HCNF.

The execution times of TANH+HPSA1 are 11%~74% increments than those of TANH, and 4%~40% increments than those of TANH in case of TANH+HPSA2. So our algorithms used short execution times to improve the schedule length of the previous algorithms.

7 Conclusions

In this paper, we present two postprocessing algorithms for previous algorithms in heterogeneous computing systems. HPSA1 and HPSA2 are executed at the hind of TANH algorithm. If DAG does not satisfy the optimality conditions of TANH, TANH+HPSA can reduce the schedule length given by TANH. Otherwise it preserves the schedule length. Our algorithms spend very short execution time. Thus they can be efficiently added to other scheduling algorithms in heterogeneous systems.

Acknowledgements

This paper was supported by Research Fund, Kumoh National Institute of Technology.

References

[1] J.D. Ullman. NP-complete scheduling problems. *Journal of Computing System Science*, 10:384-393, 1975.

[2] J. Liou and M. Palis, "A Comparison of General Approaches to Multiprocessor Scheduling," *Proc. Int'l Parallel Processing Symp.*, pp. 152-156, 1997.

[3] Y. Kwok and I. Ahmed, "Dynamic Critical-Path Scheduling: An Effective Technique for Allocating Task Graphs to Multiprocessors," *IEEE Trans. Parallel and Distributed Systems*, Vol. 7, no. 5, pp. 506-521, May 1996.

[4] G.C. Sih and E.A. Lee, "A Compile-Time Scheduling Heuristic for Interconnection-Constrained Heterogeneous Processor Architectures," *IEEE Trans. Parallel and Distributed Systems*, Vol. 4, no. 2, pp. 175-186, Feb. 1993.

[5] A. Radulescu and A.J.C. Van Gemund, "Fast and Effective Task Scheduling in Heterogeneous Systems," *Proc. of HCW*, pp. 229-238, May 2000.

[6] I. Ahmad and Y. Kwok, "On exploiting Task Duplication in Parallel Program Scheduling," *IEEE Trans. Parallel and Distributed Systems*, 9(9):872-892, Sept. 1998.

[7] T. Yang and A. Gerasoulis, "DSC: Scheduling Parallel Tasks on an Unbounded Number of processors," *IEEE Trans. Parallel and Distributed Systems*, vol. 5, no. 9, pp. 951-967, Sept. 1994.

[8] S.J. Kim and J.C. Browne, "A General Approach to Mapping of Parallel Computation upon Multiprocessor Architecture.," *Proc. Int'l Conf. Parallel Proc.*, vol. 2, pp. 23-32, Jan. 1988.

[9] J. Liou and M.A. Palis, "An Efficient Clustering Heuristics for Scheduling DAGs on Multiprocessors," *Proc. of Parallel and Distributed processing symposium*, 1996.

[10] Atakan Dogan and Fusun Ozguner, "LDBS: A Duplication Based Scheduling Algorithm for Heterogeneous Computing Systems," *Proc. of Int'l Parallel Processing (ICPP'02)*, 2002.

[11] S. Darba and D. P. Agrawal, "Optimal Scheduling Algorithm for Distributed-Memory Machines," *IEEE Trans. Parallel and Distributed Systems*, (1):87-94, Jan. 1998.

[12] G.-L. Park, B. Shirazi, and J. Marguis, "DFRN: A New Approach for Duplication Based Scheduling for distributed memory multiprocessor systems," *Proc. Of Int'l Parallel Processing Symposium*, Geneva, Switzerland, Apr. 1997.

[13] Rashmi Bajaj and Dharma P. Agrawal, "Improving Scheduling of Tasks in a Heterogeneous Environment," *IEEE Trans. Parallel and Distributed Systems*, vol. 15, No. 2, February 2004.

[14] E.S.H. Hou, N. Ansari, and H. Ren, "A Genetic Algorithm for Multiprocessor Scheduling," *IEEE Trans. Parallel and Distributed Systems*, Vol. 5, no. 2, pp. 113-120, Feb. 1994.

[15] H. Singh and A. Youssef, "Mapping and Scheduling Heterogeneous TaskGraphs using Genetic Algorithms," *Proc. of Heterogeneous Computing Workshop*, pp. 86-97, 1996.

[16] A. Ranaweera and D.P. Agrawal, "A Task Duplication based Algorithm for Heterogeneous Systems," *Proc. of IPDPS*, pp. 445-450, May 1-5, 2000.

[17] H. Topcuoglu, S. Hariri, M-Y. Wu, "Performance-Effective and Low-complexity Task Scheduling for heterogeneous computing," *IEEE Trans. on Parallel and Distributed Systems*, vol. 13, no. 3, March 2002.

[18] E. Ilavarasan and P. Thambidurai, "Levelized Scheduling of Directed A-cyclic Precedence Constrained Task Graphs onto Heterogeneous Computing System," *Proceedings of the First International Conference on Distributed Frameworks for Multimedia Applications (DFMA'05)*, 2005.

[19] Sanjeev Baskiyar and Prashanth C. SaiRanga, "Scheduling Directed A-cyclic Task Graphs on Heterogeneous Network of Workstations to Minimize Schedule length," *Proceeding of the 2003 International Conference on Parallel Processing Workshops (ICPPW'03)*, 2003.