

A Performance-Efficient Task Duplication-Based Scheduling Algorithm for Heterogeneous Computing

Yang-Ping Cheng, Jiun-Hung Ding, Shih-Hsiang Lo, and Yeh-Ching Chung
Department of Computer Science, National Tsing Hua University, Taiwan

Abstract - Efficient task scheduling algorithm is critical for application programs to achieve high performance in heterogeneous computing systems. Although a large number of scheduling heuristics have been presented in the literature, most of them are mainly for the systems with homogeneous processors. In this paper, we present a novel task scheduling algorithm, heterogeneous task duplication scheduling (HTDS), for a bounded number of heterogeneous processors with an objective to meet high performance. The HTDS algorithm uses task duplication method to decrease the communication overhead and to minimize the schedule length of application programs. To evaluate the performance of the proposed task scheduling algorithm, we have developed a simulator that contains a parametric graph generator for generating weighted directed acyclic graphs with various characteristics. We have implemented the HTDS along with three task scheduling algorithms, HEFT, LDBS1, and LDBS2, on the simulator. The simulation results show that our task scheduling algorithm outperforms other algorithms in terms of speedup.

Keywords: Parallel processing, scheduling, heterogeneous systems.

1.0 Introduction

With the recent developments in network technology, a new computing platform called heterogeneous computing system has emerged. A heterogeneous computing system can be defined as a range of diverse computing resources, which can be local to one another or geographically distributed, utilized to execute computationally intensive applications. For example, clusters of workstations connected by high-speed networks are an instance in local area, and the Grid computing environment is another instance in distributed area. Because of the characteristics of heterogeneity, efficient task scheduling is critical for achieving high performance in heterogeneous computing systems.

When the characteristics of an application program such as the execution time of tasks, the data size of communication between tasks, and task dependencies are known a priori, the application program in general can be represented as a directed acyclic graph (DAG), in which nodes represent the tasks and edges represent the communication costs as well as the dependencies among the tasks. The task scheduling can be accomplished statically during the compile time. The objective of the task scheduling is to map tasks onto processors, order their executions so that precedence requirements are satisfied, and minimize the overall completion time of an application program.

Since the task scheduling problem has been shown to be NP-complete, it is hard to find out the real optimal solution for every case. Therefore heuristics are often used to find a sub-optimal schedule. Many heuristics have been proposed in the literature. They are mainly for homogeneous computing systems. These heuristics can be roughly classified into a

variety of categories, such as list-base algorithms [1], [5], [8], [11], [13], [19], [21], clustering-base algorithms [6], [22], and duplication-base algorithms [2], [3], [14].

Recently, many task scheduling algorithms for the heterogeneous computing systems have been proposed in the literature [4], [5], [7], [9], [15], [16], [17], and [20]. The task scheduling in heterogeneous computing systems is more complicated than that in the homogeneous computing systems. This is because the task scheduling algorithm for the heterogeneous computing systems has to take into account the different execution rate of different processors and communication rate between different processors when computing the potential start-time of tasks on processors.

In this paper, we propose a static task scheduling algorithm for a bounded number of fully connected heterogeneous resources called heterogeneous task duplication scheduling (HTDS) algorithm. The HTDS algorithm consists of two phases. In the first phase, it selects the task with the highest upward rank as used in HEFT [20] for scheduling. In the second phase, it uses a task duplication method to assign the task to the processor that has the minimal finish time. These two phases are repeated for each task until all tasks are scheduled. To evaluate the performance of the proposed task scheduling algorithm, we have developed a simulator that contains a parametric graph generator for generating weighted directed acyclic graphs with various characteristics. We have implemented the HTDS along with three task scheduling algorithms, HEFT, LDBS1 [4], and LDBS2 [4], on the simulator. We have generated DAGs with different parameter settings as test samples. The simulation results show that our task scheduling algorithm outperforms other algorithms in terms of speedup for all test samples.

This paper is organized as follows. In Section 2, we give a brief introduction of related work. In Section 3, we depict the scheduling system model which consists of a target computing environment, an application program, and performance criteria for scheduling. Section 4 introduces our scheduling algorithm HTDS in details. Section 5 presents an experimental study of our algorithm with others. Finally, we give our conclusions in Section 6.

2.0 Related Work

The task scheduling problem has been studied by some research groups for the heterogeneous computing systems. Several scheduling algorithms for heterogeneous systems are proposed, such as Dynamic Level Scheduling (DLS) [19], Heterogeneous Earliest-Finish-Time (HEFT) [20], Hybrid Heuristic [17], Levelized-Min Time (LMT) [9], Task Duplication Scheduling (TDS) [16], Heterogeneous Critical Parents with Fast Duplicator (HCPFD) [7], Fast Critical Path (FCP) [15] and Fast Load Balancing (FLB) [15], and Levelized Duplication Based Scheduling (LDBS) [4]. In this section, we briefly introduce two heuristic scheduling algorithms (HEFT and LDBS) used for experimental study in Section 5. The reason why we choose HEFT for performance comparison is that HEFT outperforms several other high performance scheduling algorithms, including DLS, LMT, FCP, etc. On the other hand, since our algorithm belongs to the same category of scheduling algorithms as LDBS, it is quite nature to compare HTDS with LDBS. For TDS, since it is limited to heterogeneous computing systems with homogeneous completely connected networks, we do not compare HTDS with TDS.

2.1 The Heterogeneous Earliest Finish Time (HEFT) Algorithm

The HEFT algorithm [20] has two major phases: a task prioritizing phase and a processor selection phase. In task prioritizing phase, HEFT computes the priority of each task according to the upward rank ($rank_u$), which is the length of the critical path from task to the exit task. The task list is generated by sorting the tasks in decreasing order of $rank_u$. Tie-breaking is done randomly. In processor selection phase, tasks are selected according to their priorities and each selected task is scheduled to the best processor that minimizes the finish time of task with an insertion based policy. The time complexity of the HEFT algorithm is $O(pv^2)$, where p is the number of processors and v is the number of tasks.

2.2 The Levelized Duplication Based Scheduling (LDBS) Algorithm

The LDBS algorithm [4] utilizes level sorting technique to group tasks that are independent of each other into a level. After level sorting process, tasks in the same level are grouped as a List set and one of the two duplication heuristics (LDBS1 and LDBS2) is used to perform the scheduling. The insertion based policy is also used.

In the LDBS1 scheduling heuristic, the task-processor pair (v_i, p_j) with the minimum finish time across all possible task-processor pairs in the considering level is selected for scheduling first. In estimating the finish time of task v_i on p_j , LDBS recursively duplicates the Critical Immediate Predecessor (CIP) of v_i on processor p_i , where each processor p_i in processor set P will be considered, until the start time of task v_i cannot be improved further. The time complexity of LDBS1 is $O(|V|^3|E||P|^3)$, where V is the set of tasks of a DAG, E is the set edges of a DAG, and P is the set of processors used for scheduling.

In the LDBS2 scheduling heuristic, tasks in List set are sorted in decreasing order of their ranks (The rank is defined the same as the rank in HEFT). After that, tasks are scheduled level by level according to the sorted order. The duplication policy is similar to that of LDBS1. The difference between LDBS1 and LDBS2 is that LDBS2 does not search all possible task-processor pairs in a List set. LDBS2 just selects the task in a List set with the highest rank and schedules the task to the processor that minimizes its finish time. The time complexity of LDBS2 is $O(|V|^3|E||P|^2)$, where V is the set of tasks of a DAG, E is the set edges of a DAG, and P is the set of processors used for scheduling.

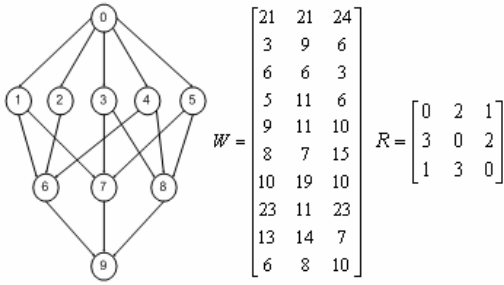
3.0 The Computational and the Architecture Model

The scheduling system model used in this paper consists of a target computing environment, an application program, and performance criteria for scheduling. The heterogeneous computing environment model is a set P of p heterogeneous computing processors connected in a fully connected topology. In our model, it is also assumed that

1. Computation can be overlapped with communication, that is, any computing resource can execute the task and communicate with other computing resources at the same time.
2. The schedule must be non-preemptive. Once a processor has started task execution, it continues without interruption. After the completion of the execution, it immediately sends the output data to all dependent children tasks simultaneously.

An application program can be represented by a directed acyclic graph (DAG) $G(V, E, C, W)$ as shown in Figure 1, where

- V is a set of v nodes, and each node $v_i \in V$ represents a task which contains instructions that are executed sequentially without preemption. In this paper, the terms of node and task are interchangeable.
- E is a set of edges between the tasks. Each edge corresponds to task dependences (communication messages or precedence constrains).
- C is the set of communication cost. Each edge has a communication cost $c_{i,j} \in C$.
- W is a $v \times p$ computation costs matrix in which each $w_{i,j}$ gives the estimated time to execute task v_i on processor p_j .



$d_{0,1}=9, d_{0,2}=3, d_{0,3}=4, d_{0,4}=6, d_{0,5}=4, d_{1,6}=5, d_{1,7}=8, d_{2,6}=2,$
 $d_{3,7}=4, d_{3,8}=5, d_{4,6}=7, d_{4,8}=3, d_{5,7}=6, d_{5,8}=7, d_{6,9}=3, d_{7,9}=5, d_{8,9}=8.$

Fig. 1. The application graph model of a heterogeneous system.

Before scheduling, the tasks are labeled with the average execution costs. The average execution cost can be computed as follows:

$$\overline{\omega}_i = \sum_{j=1}^p \frac{\omega_{i,j}}{p}. \quad (1)$$

The communication costs per transferred byte between any two machines are stored in matrix R of size $p \times p$. The communication cost of edge $e_{i,j}$ for transferring d bytes data from task v_i (scheduled on p_m) to task v_j (scheduled on p_n), is defined by

$$c_{i,j} = R_{m,n} \times d_{i,j}. \quad (2)$$

where $d_{i,j}$ is the amount of data transmitted from task v_i to task v_j (in bytes) and $R_{m,n}$ is the communication cost per transferred byte from p_m to p_n (in sec/byte). The average communication cost can be computed as follows:

$$\overline{c}_{i,j} = \overline{R} \times d_{i,j}. \quad (3)$$

Where, \overline{R} is the average communication cost per transferred byte over all processors. If two tasks are scheduled to the same processor, the communication cost between them is assumed to be zero.

The main objective of the task scheduling is to determine the assignment of tasks of a given application to processors such that the scheduling length (makespan) is minimized and all precedence constrains are also satisfied.

4.0 The HTDS Algorithm

In this section, we first introduce some definitions and basic principles used in the proposed algorithm, followed by the description of our algorithm and an example of using HTDS to schedule tasks.

4.1 Definitions and Rule used by HTDS

Given a DAG, a task cannot start its execution before it gathers all necessary data from its parent tasks. The term $DRT(n_i, p_j)$ represents the data ready time of task n_i on processor p_j . When a task n_i is scheduled on a processor p_j , we define $ST(n_i, p_j)$ and $FT(n_i, p_j)$ as the start time and the finish time of n_i on processor p_j , respectively. The start time of a task on a processor depends on the data arrival time from its parent tasks and the time when the processor is available. The *critical immediate predecessor* (CIP) of task n_i is defined as the parent task of n_i from which the sent data is the last one received by task n_i . The term $CIP(n_i, p_j)$ represents the critical immediate predecessor of task n_i on processor p_j .

To schedule a task on a processor, our method is to scan through the whole time span of the processor to find the earliest time slot that is larger enough to accommodate the schedulable task and the precedence constrains are not violated. We use Rule 1 below to control the selection of a suitable time slot for a task on a processor.

Rule 1: A task n_r can be scheduled to a processor p , on which a set of m tasks $\{n_1, n_2, \dots, n_m\}$ have been scheduled, if there exists some values of k such that

$$ST(n_{k+1}, p) - \max\{FT(n_k, p), DRT(n_r, p)\} \geq w(n_r, p),$$

where $k = 0, 1, \dots, m$, $ST(n_{m+1}, p) = \infty$, and $FT(n_0, p) = 0$.

The earliest start time of n_r on p is defined by $\max\{FT(n_i, p), DRT(n_r, p)\}$, where i is the minimal value of k satisfying the inequality. In our algorithm, tasks are ordered by their scheduling priorities. The static priority *rank* of a task n_i is recursively defined by

$$rank(n_i) = \overline{w}_i + \max_{n_j \in succ(n_i)} (\overline{c}_{i,j} + rank(n_j)), \quad (4)$$

where $succ(n_i)$ is the set of immediate successor of task n_i , $\overline{c}_{i,j}$ is the average communication cost of edge (i, j) , and \overline{w}_i is the average computation cost of task n_i . This rank is used by the HEFT algorithm [20] as well.

4.2 HTDS Algorithm

The HTDS algorithm is a task scheduling algorithm for a bounded number of heterogeneous processors. It is given in Figure 2. In Figure 2, at step 1, the computation costs of tasks and communication costs of edges are set to their mean values. Step 2 computes

the *rank* values of all tasks. At step 3, scheduling list is generated by sorting the tasks in decreasing order of *rank* values. Tie-breaking is done randomly. It can be easily shown that the decreasing order of *rank* values forms a topological order of tasks, which is a linear order that preserves the precedence constraints. Steps 4 to 21 form a loop. This loop performs the assignment of tasks to processors until all tasks in scheduling list are scheduled. While considering a task for scheduling, we will estimate the earliest start time on every processor by calling function `min_start_time`, and the task is scheduled onto the processor that minimizes its finish time. This process is performed from step 7 to step 21. The second *for* loop (steps between 13 and 18) performs task duplication process. While estimating the start time of task n_i on processor p_j , the algorithm will try to duplicate possible ancestors of n_i on each processor by calling function `min_start_time` and record the best solution. After the best processor is found, the algorithm does the necessary duplication of ancestors and schedules the task to the best processor. In the algorithm, when scheduling a task or duplicating a task on a processor, the task insertion policy (Rule 1) is applied.

```

algorithm HTDS
1. Set the computation costs of tasks and
   communication costs of edges to their mean values.
2. Compute rank values for all tasks
3. Sort the tasks in a scheduling list
   in decreasing order of rank values.
4. while (there are unscheduled tasks in the scheduling
   list)
5. do  $n_i \leftarrow$  select first task from the scheduling list
6.  $f_{time} \leftarrow \infty$ 
7. for each processor  $p_j \in$  processor set  $P$ 
8. do compute earliest start time  $ST \leftarrow ST(n_i, p_j)$  and
   finish time  $FT \leftarrow FT(n_i, p_j)$ 
9. if  $FT < f_{time}$ 
10. then  $v \leftarrow n_i, m \leftarrow p_j, v_{CIP} \leftarrow \phi, m_{CIP} \leftarrow \phi$ ;
11.  $f_{time} \leftarrow FT$ 
12.  $n_k \leftarrow CIP(n_i, p_j)$ 
13. for each processor  $p_l \in$  processor set  $P$ 
14. do  $stime \leftarrow \min\_start\_time(n_i, p_j, n_k, p_l, ST)$ 
15. Undo all duplications generated
   by min_start_time() function on  $p_l$ 
16. if  $stime + w(n_i, p_j) < f_{time}$ 
17. then  $v \leftarrow n_i, m \leftarrow p_j, v_{CIP} \leftarrow n_k, m_{CIP} \leftarrow p_l$ ;
18.  $f_{time} \leftarrow stime + w(n_i, p_j)$ 
19.  $\min\_start\_time(v, m, v_{CIP}, m_{CIP}, \infty)$ 
20. Schedule task  $v$  onto processor  $m$ 
21. Remove  $v$  from scheduling list
end of algorithm HTDS

```

Fig. 2. Algorithm HTDS

Note that HTDS is based on the duplication concept of LDBS. The difference of duplication method between LDBS and HTDS is that LDBS just tries to duplicate critical immediate parents, but HTDS tries to duplicate all possible ancestors.

The function `min_start_time` is showed in Figure 3. In function `min_start_time`, if CIP of task n_i on p_j is n_k , we will first try to duplicate n_k onto p_l . If this can reduce the start time of n_i on p_j , we then try to

minimize the start time of n_k by recursively calling `min_start_time($n_k, p_l, CIP(n_k, p_l), p_l, ST(n_k)$)` function to duplicate possible ancestors of n_k . This process is performed from step 3 to step 8. After exploring the ancestor tasks reachable from n_k , we attempt to further reduce the start time of n_i by considering if there is another CIP of n_i that dominates the start time of n_i . This process is performed by Step 13.

The time complexity of HTDS is $O(|V|^3|E||P|^2)$ which is the same as LDBS2. But actually, the execution time of HTDS is larger than LDBS2, because HTDS do more duplication trials than LDBS2.

```

algorithm min_start_time( $n_i, p_j, n_k, p_l, stime$ )
1. if  $n_k \neq \phi$ 
2. then if  $n_k$  is not already scheduled on  $p_l$ 
3. then Duplicate  $n_k$  in a time slot on  $p_l$ 
   that minimize its start time
4.  $ST_{n_k} \leftarrow ST(n_k, p_l)$ 
5. Recompute  $ST \leftarrow ST(n_i, p_j)$ 
6. if  $ST < stime$ 
7. then  $CIP_{n_k} \leftarrow CIP(n_k, p_l)$ 
8.  $ST_{n_k\_new} \leftarrow \min\_start\_time(n_k, p_l, CIP_{n_k}, p_l, ST_{n_k})$ 
9. if  $ST_{n_k\_new} < ST_{n_k}$ 
10. then update the start time
   and finish time of  $n_k$  on  $p_l$ 
11. Recompute  $ST \leftarrow ST(n_i, p_j)$ 
12.  $n_k \leftarrow CIP(n_i, p_j)$ 
13.  $stime \leftarrow \min\_start\_time(n_i, p_j, n_k, p_l, ST)$ 
14. if  $stime < ST$ 
15. then  $ST \leftarrow stime$ 
16. else undo_duplication( $n_k, p_l$ )
17. return  $ST$ 
18. else return  $stime$ 
19. else return  $stime$ 
end of algorithm min_start_time

```

Fig. 3. Algorithm `min_start_time`

4.3 Example

We now give an example to show the behavior of algorithm HTDS. Given the application graph model of heterogeneous system shown in Figure 1, at first, HTDS computes the *rank* values of all tasks. We represent the *rank* values as a set $\{99, 59, 36, 52, 51, 59, 27, 37, 35, 8\}$. According to the *rank* values, the scheduling order of the tasks with respect to the HTDS is $\{n_0, n_5, n_1, n_3, n_4, n_7, n_2, n_8, n_6, n_9\}$. Then, HTDS selects tasks according to the order of their priorities and schedules each selected task on the best processor with task duplication method. In Figure 4, we show the scheduling results of scheduling algorithms of HTDS, LDBS1, LDBS2 and HEFT for the DAG shown in Figure 1. The scheduling lengths of HTDS, LDBS2, LDBS1 and HEFT are 68, 71, 77, and 79, respectively.

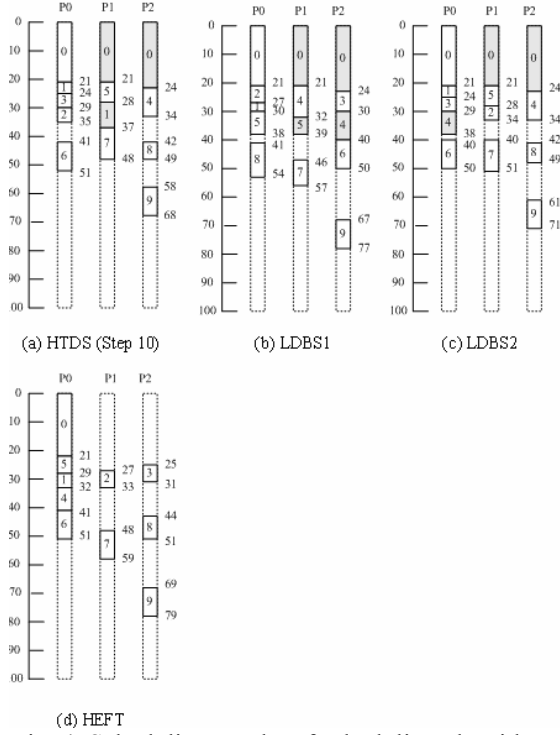


Fig. 4. Scheduling results of scheduling algorithms HTDS, LDBS1, LDBS2 and HEFT for the DAG shown in Fig. 1.

5.0 Performance Evaluation and Simulation Results

In this section, we use simulation approach to do the performance evaluation. In order to evaluate the performance of the proposed scheduling algorithm, we compared the performance of HTDS with those of HEFT, LDBS1 and LDBS2. We have implemented a random graph generator to produce weighted DAGs of applications. Our random graph generator can allow assigning sets of values to the parameters which are used to generate the application graphs for the heterogeneous environment simulator.

Our random graph generator requires the following input parameters to generate weighted DAGs.

- Number of tasks in the graph, (v).
- Graph parallelism (GP) [18]. The GP is defined as the ratio of total average computation time (w_i) of a DAG to the total average computation time of tasks on the critical path of a DAG.
- Out degree base of a task, (out_degree_base).
- Range percentage of computation costs on processors, (β). The β is basically the heterogeneity factor for processor speeds.
- Communication to computation ratio, (CCR). It is the ratio of the average communication to the average computation cost.
- Number of Processors, (PN). This parameter indicates the bounded number of processors.

In our study, we will focus on the study of the relation of CCR, GP, and PN to the scheduling speedup. For experimental study, we set the values of $v = \{300\}$, the values of $CCR = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$, the values of $GP = \{8, 16, 32\}$, the values of $\beta = \{0.5, 1, 1.5\}$, the values of $out_degree_base = \{1, 3, 6\}$, and the values of $PN = \{4, 8, 16, 32\}$. For each tuple ($v, CCR, GP, \beta, out_degree_base, PN$), we randomly generate 10 DAGs as the test samples.

Let GPP [3] be the ratio of GP to PN, i.e., $GPP = GP / PN$. Since the GP is the ratio of total average computation time (w_i) of a DAG to the total average computation time of tasks on the critical path of a DAG, it implies the maximal speedup that can be achieved by a scheduling algorithm for a given DAG. In the following we will analyze the performance of four scheduling algorithms, HTDS, HEFT, LDBS1, and LDBS2, based on the values of GPP.

5.1 GPP < 1

$GPP < 1$ implies that the PN used in a scheduling algorithm is greater than the GP of a DAG. Speedups for scheduling algorithms with $GPP = 0.5$ and $GPP = 0.25$ are shown in Figure 5(a) and Figure 5(b), respectively. From Figure 5, we can see that HTDS outperforms other algorithms for $CCR \geq 1$. It can be noted that when the value of CCR increased, the difference of speedups between HTDS and other algorithms is increased as well. The reason is that when the value of CCR increases, the possibility of duplicating more ancestors of a task also increases. HTDS tries to duplicate all possible ancestors, and this is better than only duplicating immediate parents. On the average, the rank of speedup is $HTDS \geq LDBS2 \geq LDBS1 \geq HEFT$.

Since $GPP < 1$ implies that the PN used in a scheduling algorithm is greater than the GP of a DAG, it is of interest to see if the speedup is proportional to the number of processors used. From Figure 6, we can see that, in general, the more processors we used, the better speedup we can expect. However the gain is not proportional to the number of processors used. For example, in Figure 6(d), the speedups of HTDS for DAGs with $CCR = 1$ and $GPP = 0.25, 0.5$ and 1 are 6.61, 6.10 and 5.23, respectively. The ratio of processors used is $32:16:8 = 4:2:1$ and the ratio of speedups is $6.61:6.10:5.23 = 1.26:1.17:1$.

5.2 GPP = 1

$GPP = 1$ implies that the PN used in a scheduling algorithm is equal to the GP of a DAG. In Figure 7, speedups for scheduling algorithms with $GPP = 1$ are shown. From Figure 7, we can see that the speedup ranking is $HTDS \geq LDBS2 \geq LDBS1 \geq HEFT$.

5.3 GPP>1

$GPP > 1$ implies that the PN used in a scheduling algorithm is less than the GP of a DAG. In Figure 8, speedups for scheduling algorithms with $GPP = \{2, 4\}$ are shown. From Figure 8(a), we can see that the speedups of HTDS are almost the same as LDBS2 when $CCR \leq 10$. This seems to imply that when GPP is large enough, the speedups of HTDS is the same as LDBS2. Figure 8 shows that in most case, the speedup ranking is $HTDS \geq LDBS2 \geq LDBS1 \geq HEFT$.

5.4 Comparisons of Execution Time of Scheduling Algorithms

Since the execution time of scheduling algorithms depends on the number of tasks and the number of processors, we measure the execution time with respect to the number of tasks and the number of processors, respectively. In Figure 9(a), we show the execution time of scheduling algorithms by setting the number of processors to thirty-six and the number of tasks form 500 to 1000. In Figure 9(b), we show the execution time of scheduling algorithms by setting the number of tasks to 800 and the number of processors from 20 to 60. From Figure 9, we can see that the execution time ranking is $LDBS1 \geq HTDS \geq LDBS2 \geq HEFT$. The execution time of LDBS1 is much higher than others. From Figure 9(b), we can see that the execution time of LDBS1 is very sensitive to the number of processors. As the number of processors increases, the execution time of LDBS1 increases rapidly.

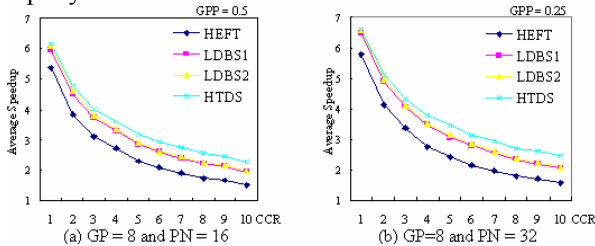


Fig. 5. The speedups of scheduling algorithms with $GPP < 1$.

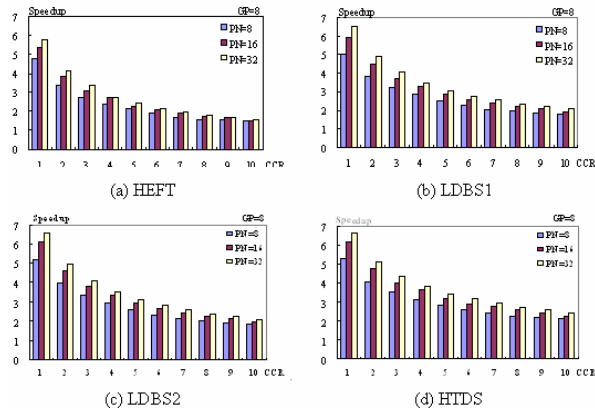


Fig. 6. The speedups of scheduling algorithms, where $CCR=1, \dots, 10$ and $GP \leq 1$.

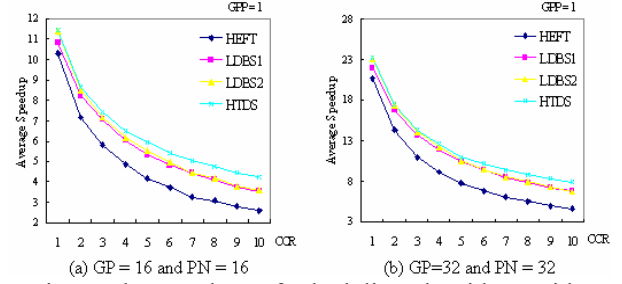


Fig. 7. The speedups of scheduling algorithms with $GPP = 1$.

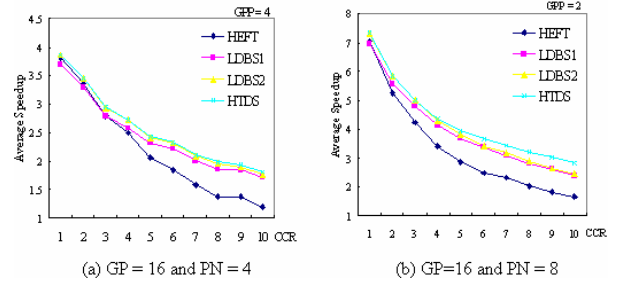


Fig. 8. The speedups of scheduling algorithms with $GPP > 1$.

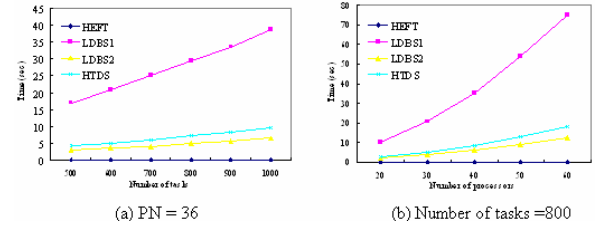


Fig. 9. The execution time (in second) of scheduling algorithms.

6.0 Conclusions

In this paper, we have presented a new algorithm, called the HTDS algorithm, for scheduling application graphs onto a heterogeneous computing system. In the experimental study, we use randomly generated application graphs with various characteristics to test HTDS along with HEFT, LDBS1, and LDBS2 scheduling algorithms. From the simulations results we have the following conclusions. (1) In terms of speedups of scheduling algorithms, on the average, the speedup ranking is $HTDS \geq LDBS2 \geq LDBS1 \geq HEFT$. (2) In terms of the execution time of scheduling algorithms, in general, the execution time ranking is $LDBS1 \geq HTDS \geq LDBS2 \geq HEFT$. Since the main objective of our algorithm is to minimize the schedule length and the scheduling is done off-line, a slightly longer time in generating a considerably improved solution should be acceptable. In conclusion, our algorithm outperforms other algorithms in term of speedup (makespan).

7.0 Acknowledgement

The work of this paper was partially supported by

National Science Council and Ministry of Economic Affairs of the Republic of China under contract NSC-94-2213-E-007-037, NSC-94-2213-E-007-080, NSC-94-2752-E-007-004-PAE, and MOEA 94-EC-17-A-04-S1-044.

8.0 References

- [1] T.L. Adam, K.M. Chandy, and J.R. Diskson, "A Comparison of List Schedules for Processing Systems," *Communication of ACM*, Vol.17, No. 12, pp. 685-690, 1974.
- [2] I. Ahmad and Y.-K. Kwok, "On Exploiting Task Duplication in Parallel Program Scheduling," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 9, No. 9 September 1998.
- [3] Y.C. Chung and S. Ranka, "Applications and Performance Analysis of a Compile-Time Optimization Approach for List Scheduling Algorithms on Distributed Memory Multiprocessors," *Proc. Supercomputing*, pp. 512-521, Nov. 1992.
- [4] Atakan Dogan and Fusun Ozguner, "LDBS: A Duplication Based Scheduling Algorithm for Heterogeneous Computing Systems," *Proceedings of the International Conference on Parallel Processing (ICPP'02)*, 2002.
- [5] H. El-Rewini and T.G. Lewis, "Scheduling Parallel Program Tasks onto Arbitrary Target Machines," *Journal of Parallel and Distributed Computing*, Vol. 9, pp.138-153, 1990.
- [6] A. Gerasoulis and T. Yang, "A Comparison of Clustering Heuristics for Scheduling DAGs on Multiprocessor," *J. Parallel and Distributed Computing*, Vol. 16, No. 4, pp. 276-291, December 1992.
- [7] T. Hagra, J. Janecek, "A High Performance Low Complexity Algorithm for Compile-Time Task Scheduling in Heterogeneous Systems," *Proceedings of the 18th International Parallel and Distributed Processing Symposium (IPDPS'04)*, 2004.
- [8] J.J. Hwang Y.C Chou, F.D. Anger, and C.Y. Lee, "Scheduling Precedence Graphs in Systems with Interprocessor Communication Times," *SIAM Journal of Computing*, Vol. 18. pp. 244-257, 1989.
- [9] M. Iverson, F. Ozguner, and G. Follen, "Parallelizing Existing Applications in a Distributed Heterogeneous Environment," *Proc. Heterogeneous Computing Workshop*, pp. 93-100, 1995.
- [10] S.J. Kim and J.C. Browne, "A General Approach to Mapping of Parallel Computation upon Multiprocessor Architectures," *Proc. Int'l Conf. Parallel Processing*, Vol. 2, pp. 1-8,1998.
- [11] B. Kruatrachue and T.G. Lewis, "Grain Size Determination for Parallel Processing," *IEEE Software*, pp. 23-32, Jan. 1988.
- [12] Y.-K. Kwok and I. Ahmad, "Static Scheduling Algorithms for Allocating Directed Task Graphs to Multiprocessors," *ACM Computing Surveys*, Vol. 31, No. 4, December 1999.
- [13] Y.-K. Kwok and I. Ahmad, "Dynamic Critical-Path Scheduling: An Effective Technique for Allocating Task Graphs to Multiprocessors," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 7, No. 5, pp. 506-521, May 1996.
- [14] G.-L. Park, B. Shirazi, and J. Marquis, "DFRN: A new approach for duplication based scheduling for distributed memory multiprocessor systems," *Proc. Int'l Conf. Parallel Processing Symposium*, pp. 157-166, 1997.
- [15] A. Radulescu and A.J.C. Van Gemund, "Fast and Effective Task Scheduling in Heterogeneous Systems," *IPDPS Workshop on Heterogeneous Computing*, pp. 229-238, MAY 2000.
- [16] S. Ranaweera and D.P. Agrawal, "A Task Duplication based Algorithm for Heterogeneous Systems," *Proc. of IPDPS*, pp. 445-450, May 2000.
- [17] R. Sakellariou, H. Zhao, "A Hybrid Heuristic for DAG Scheduling on Heterogeneous Systems," *Proceedings of the 18th International Parallel and Distributed Processing Symposium (IPDPS'04)*, 2004.
- [18] G.C. Sih and E.A. Lee, "Scheduling to Account for Interprocessor Communication within Interconnection-Constrained Processor Networks". *ICPP*, Vol. 1, pp. 9-16, 1990.
- [19] G.C. Sih and E.A. Lee, "A Compile-Time Scheduling Heuristic for Interconnection-Constrained Heterogeneous Processor Architectures," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 4, No. 2, pp. 175-186, Feb. 1993.
- [20] H. Topcuoglu, S. Hariri, and M.-Y. Wu, "Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Computing," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 13, No. 3, March 2002.
- [21] M. Wu and D. Gajski, "Hypertool: A Programming Aid for Message Passing Systems," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 1, pp. 330-343, July 1990.
- [22] T. Yang and A. Gerasoulis, "DSC: Scheduling Parallel Tasks on an unbounded Number of Processors," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 5, No. 9, pp.951-967, Sept. 1994.