TELKOMNIKA Indonesian Journal of Electrical Engineering
Vol.12, No.6, June 2014, pp. 4858 ~ 4875
DOI: 10.11591/telkomnika.v12i6.5847

A Heuristic Greedy Algorithm for Scheduling Out-Tree Task Graphs

Jian Jun Zhang*, Wei Wen Hu, Mei Ni Yang
College of Science, Naval University of Engineering,
No. 717, Jiefang Avenue, Wuhan City, Hubei Province, P. R. China, +86-13871162297
*Corresponidng author, e-mail: wahh0912@163.com

Abstract

The scheduling of Out-Tree task graphs is one of the critical factors in implementing the compilers of parallel languages and improving the performance of parallel computing. Although there are a few heterogeneity based algorithms in the literature suitable for scheduling Out-Tree task graphs, they usually require significantly high scheduling costs and may not deliver good quality schedules with lower costs. This paper presents a heuristic greedy scheduling algorithm for Out-Tree task graphs with an objective to simultaneously meet high performance and fast scheduling time, which is based on list and task duplication, tries to find the best point between balancing loads and shortening the schedule length and improves the schedule performance without increasing the time complexity of the algorithm. The comparative study shows that the proposed algorithm surpasses previous approaches in terms of schedule length ratio, speedup and efficiency metrics.

Keywords: *parallel distributed computing, heterogeneous computing system, task scheduling, Out-Tree task graph, task duplication*

1. Introduction

The effective task scheduling of applications plays a significant role in the performance of parallel distributed computing. Tasks must be scheduled and assigned to processors in a way that minimizes the total execution time or the schedule length of the distributed application. One of the most important issues for high-performance computing with Heterogeneous Computing Systems (HCS) is the mapping strategies they adopt. In general, static task scheduling for HCSs is NP-complete problem [1]. Because of its key importance on performance, the heterogeneity based scheduling algorithms has been extensively studied and various heuristics were proposed in the literature [1-9]. In this paper, we consider the scheduling of Out-Tree task graphs in HCSs. Many parallel programs exhibit in the Out-Tree structure, and this type of parallel program paradigm arises in many application areas. The scheduling problem of Out-Tree task graphs plays a very important role in implementing the compilers of parallel languages and improving the performance of parallel computing.

There are several classical heuristic scheduling algorithms in HCSs [2-5], which is suitable for scheduling Out-Tree task graphs. The Task Duplication Scheduling (H_TDS) algorithm [3] schedules tasks according to their *level*s and assigns each task to a *suitable* processor while guaranteeing the shorter schedule length and reasonable time complexity, but it needs too many processors when scheduling Out-Tree task graphs. The Heterogeneous Earliest-Finish-Time (HEFT) algorithm [4], which is based on list scheduling, schedules the task according to its $Rank_u$ value which is based on its average execution cost, and uses an insertion based strategy that considers the possible insertions of a task on the most suitable processor which minimizes its earliest finish time. It ignores the load balance between processors and the economization on processors, which easily leads to an undesirable schedule length and needs too many processors when scheduling Out-Tree task graphs.

In this paper, based on the parallel computing model in HCSs and the characteristics of the Out-Tree graphs, we propose a new algorithm, called the Heuristic Greedy Algorithm, for Scheduling Out-Tree task graphs (HGAS_OT), the motivation behind which is to deliver good quality of schedules with lower costs. Combing the strategies of list scheduling and task duplication, and taking the load balances into consideration, it schedules the task according to

Received January 25, 2014; Revised March 20, 2014; Accepted April 3, 2014

the priority which is the maximal value of all its earliest completion times when scheduling it to all processors respectively, and tries to schedule each leaf task on its most suitable processor while guaranteeing the shorter schedule length and less number of used processors.

2. The Proposed Algorithm

2.1. Preliminaries.

A scheduling system model consists of an application, a target computing environment, and a performance criterion for scheduling. An application is represented by a directed acyclic graph, $G = (V, E)$, where V is the set of v tasks and E is the set of e edges between the tasks. $Data$ is a $v \times v$ matrix of communication data, where $data_{i,j}$ is the amount of data required to be transmitted from task n_i to n_j. In a given task graph, a task without any parent is called an *entry task* and a task without any child is called an *exit task*.

Out-Tree task graph is one of the basic structures for parallel processing, an example of which is shown in Figure 1. We assume that the target computing environment consists of a set Q of n heterogeneous processors connected in a fully connected topology in which all inter-processor communications are assumed to perform without contention and computation can be overlapped with communication. Task executions of a given application are assumed to be non-preemptive. W is a $m \times n$ computation cost matrix in which each w_{ij} (also denoted by $w(n_i,P_j)$) gives the estimated execution time to complete task n_i on processor P_j. Before scheduling, the tasks are labeled with the average execution costs.

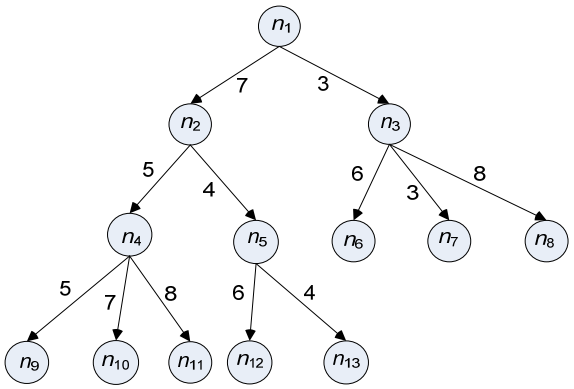

Figure 1. An example Out-Tree Task Graph with 13 Tasks

The data transfer rates between processors are stored in matrix B of size $n \times n$. The communication start-up costs of processors are given in a n-dimensional vector L. The communication cost of the edge (i, j), which is for transferring data from task n_i (scheduled on P_l) to n_j (scheduled on P_k), is defined by $c_{ij}=L_l+(Data_{ij}/B_{lk})$. Before scheduling, average communication costs are used to label the edges. The average communication cost of an edge (i, j) is defined by $\overline{c_{ij}} = \overline{L} + Data_{ij} / \overline{B}$, where $\overline{B}$ is the average transfer rate among the processors and $\overline{L}$ is the average communication start-up time.

Before presenting our algorithm, it is necessary to define a few parameters for Out-Tree task graphs.

Definition 1. Parameter $ct(n_i,P_j)$ denotes the execution time of the task n_i and its ancestors on processor P_j, which can be recursively denoted as $ct(n_i,P_j)= ct(pred(n_i),P_j)+w(n_i,P_j)$, starting from the only entry task, where $pred(n_i)$ is the only immediately predecessor task of task n_i and thus $ct(pred(n_i),P_j)$ is the time when all data needed by n_i has arrived at processor P_j; especially, for the root task n_1, $ct(n_1,P_j)=w(n_1,P_j)$. In order to compute $ct(n_i,P_j)$, the immediately predecessor task of n_i must have been scheduled.

Definition 2. Parameter $lact(n_i)$ and $ect(n_i,P_q)$ denotes the latest allowable completion time and the earliest completion time of the task n_i respectively, which can be calculated by $lact(n_i)=\max\{ct(n_i,P_j) | P_j \in Q\}$ and $ect(n_i,P_q)=\min\{ct(n_i,P_j) | P_j \in Q\}$, where P_q gives the

corresponding processor on which task n_i completes its execution in its earliest completion time, $ect(n_i,P_q)$. Additionally, *SL* denotes the current schedule length at each scheduling step.

After all tasks in a Out-Tree task graph are scheduled, the schedule length (i.e., overall completion time) will be the actual finish time of the exit task. If there are multiple exit tasks and the convention of inserting a pseudo exit task is not applied, the schedule length (which is also called *makespan*) is defined as the maximum of all the actual finish times of the exit tasks. The objective function of the task scheduling algorithm is to determine the assignment of tasks of a given application to processors such that its schedule length is minimized.

2.2. Description of the Algorithm

Our algorithm has two major phases: a task prioritizing phase for computing the priorities of all the leaf tasks and a processor selection phase for selecting the tasks in the order of their priorities and scheduling each selected task and its ancestor tasks on its "best" processor, which minimizes the task's finish time.

Task prioritizing phase: In this phase, our algorithm requires the priority of each leaf task to be set with the *lact* attributes. The task list *l_task* is generated by sorting the tasks in decreasing order of their *lact* attributes. Tie-breaking is done randomly. There can be alternative polices for tie-breaking. Since the alternative polices increase the time complexity, we prefer a random selection strategy.

Processor selection phase: In this phase, our algorithm first assigns the first task in *l_task* and all its ancestors to the processor which guarantees its earliest completion time. Then, it deploys the following greedy strategy that, for subsequent tasks in *l_task*, while guaranteeing not to increase the current *SL*, the algorithm assigns the leaf node and its ancestor nodes to used processor by avoiding duplication of task. On the other hand, if it is necessary to increase the current *SL*, while guaranteeing the increase in theschedule length is as less as possible, the algorithm schedules the leaf node and all its ancestor nodes to a new processor or a used processor. Figure 2 shows the steps involved in the proposed algorithm in detail.

Algorithm 1:
Begin
Compute parameter *lact*s of the leaf tasks, sort leaf tasks $n_1, n_2,\ldots, n_m$ in descending order by *lact*s, for convenience (sake), still denote them as $n_1, n_2,\ldots,n_m$, and put them into *l_task* in turn;
while (*l*_task $\neq \Phi$) **do**
Remove the leaf task n_i from *l_task*;
$SL=max\{SL, ect(n_i,P_q)\}$;
if there exist certain P_j such that $ct(n_i,P_j)<=SL$, where $P_j \notin Q$
Schedule the leaf task n_i and its ancestors without duplication to processor P_j;
else
Schedule the leaf task n_i and its ancestors without duplication to processor P_q;
Remove processor P_q from *Q*;
Endwhile
End

Figure 2. Overall steps of the HGAS_OT Algorithm

2.3. The Time Complexity and Effectivity of the Algorithm

In this section, we will analyze the time complexity and the effectivity of the HGAS_OT algorithm.

First of all, in the task prioritizing phase the HGAS_OT algorithm traverses all the tasks of the DAG to compute the *lact* attributes of the leaf tasks and sorts the leaf tasks in the non-increasing order of their *lact* values, the worst case time complexity of this step would be on the order of $O(vp)+O(vlogv)$, where *v* is the number of tasks in the task graph and *p* is the number of processors required. The second phase is carried out to search whether the new leaf node and its ancestor nodes could be scheduled to the suitable processor, all the used processors may be examined, with the worst case time complexity of $O(v^2p)$.

Consequently, the overall time complexity of the HGAS_OT algorithm is $O(v^2p)$.

On the other hand, Given an Out-Tree task graph and a HCS, the HGAS_OT algorithm produces an schedule, the schedule length of which is the values of *SL* when the algorithm terminates, This is just the fact we can see from the description of the HGAS_OT algorithm.

3. Performance and Comparison

3.1. An Illustrative Example

To illustrate the effectiveness of the proposed algorithm, in this section we first give the scheduling process for the example Out-Tree task graph in Figure 1. Without loss of generality, suppose n=8 and generate the computation cost matrix randomly, as is expressed in (1).

$$\begin{pmatrix} 5 & 7 & 6 & 7 & 5 & 5 & 6 & 6 \\ 3 & 2 & 3 & 3 & 3 & 2 & 4 & 3 \\ 3 & 2 & 2 & 3 & 3 & 3 & 3 & 4 \\ 7 & 8 & 9 & 8 & 9 & 8 & 7 & 9 \\ 3 & 3 & 3 & 4 & 3 & 3 & 4 & 3 \\ 6 & 7 & 7 & 6 & 6 & 6 & 7 & 7 \\ 3 & 4 & 3 & 3 & 3 & 3 & 5 & 4 \\ 7 & 6 & 9 & 7 & 8 & 7 & 6 & 8 \\ 3 & 2 & 3 & 4 & 3 & 2 & 2 & 3 \\ 3 & 4 & 3 & 3 & 4 & 3 & 4 & 3 \\ 5 & 7 & 5 & 6 & 5 & 6 & 5 & 7 \\ 7 & 8 & 6 & 6 & 7 & 8 & 7 & 7 \\ 3 & 5 & 6 & 3 & 4 & 5 & 4 & 4 \end{pmatrix} \quad (1)$$

Let us show how the HGAS_OT algorithm works in detail. Firstly, in task prioritizing phase, *lact* values for all leaf nodes are computed in a top-down fashion, starting from the only entry task n_1. The corresponding results are shown in Table 1. So the list *l_task* is n_{11}, n_9, n_{10}, n_{12}, n_8, n_{13}, n_6 and n_7.

Table 1. The Latest Allowable Completion Times of the Leaf Nodes

Task	n_6	n_7	n_8	n_9	n_{10}	n_{11}	n_{12}	n_{13}
lact	17	14	18	22	21	25	21	18

Then in processor selection phase the algorithm schedules leaf node n_{11} and its ancestors to processor P_1, and we have $ct(n_{11},P_1)$=20, SL=20; for leaf task n_9, because $ct(n_9,P_1)$=20+3=23>max{20,$ect(n_9,P_6)$}, the algorithm assigns n_9 and its ancestors to processor P_6, and we have $ct(n_9,P_6)$=17; as to node n_{10}, from $ct(n_{10},P_1)$=20+3=23>max{20,18} and $ct(n_{10},P_6)$=17+3=20≤max{20,18}, the HGAS algorithm schedules n_{10} and all its ancestors to processor P_6, and lets $ct(n_{10},P_6)$=20.

Adopting above strategy, the HGAS_OT algorithm sequentially carries out the scheduling process of the subsequent leaf nodes: it schedules task n_{12} and all its ancestors to processor P_3 and lets $ct(n_{12},P_3)$=18, schedules task n_8 and all its ancestors to processor P_2 and lets $ct(n_8,P_2)$=15, schedules task n_{13} and all its ancestors to processor P_5 and lets $ct(n_{13},P_5)$=15, and, schedules task n_7 to processor P_2 (noticing all its ancestors have already been in processor P_2) and lets $ct(n_7,P_2)=ct(n_8,P_2)+w(n_7,P_2)$=19. At last, we have *l_task*=Φ, and the algorithm terminates. The processor allocations and scheduling times produced by our algorithm are shown in Figure 3.

On the other hand, for the HEFT algorithm, in its task prioritizing phase, the upward rank value, $Rank_u$, for all nodes, which is based on mean computation and mean communication costs, are computed. The corresponding results are shown in Table 2. So the scheduling order is n_1, n_2, n_4, n_3, n_5, n_8, n_{11}, n_{12}, n_6, n_{10}, n_{13}, n_9 and n_7.

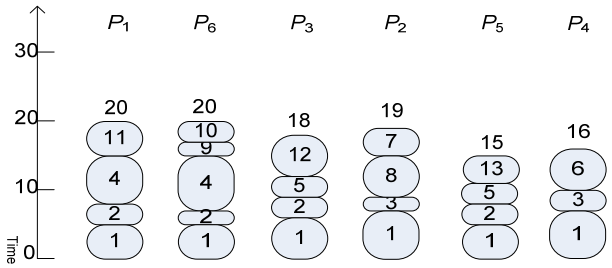

Figure 3. The Scheduling Result Produced by the HGAS_OT Algorithm

Table 2. The Upward Rank Values of All the Nodes

Task	n_1	n_2	n_3	n_4	n_5	n_6	n_7	n_8	n_9	n_{10}	n_{11}	n_{12}	n_{13}
$Rank_u$	42.625	36.75	21.125	26.875	20.25	12.5	6.5	15.25	7.75	10.375	13.75	13	8.25

Then in processor selection phase, the HEFT algorithm sequentially carries out the scheduling process: it schedules task n_1 to its best processor P_1, schedules task n_2 to its best processor P_1, schedules task n_4 to its best processor P_1, schedules task n_3 to its best processor P_2 which enables minimal execution time, schedules task n_5 to its best processor P_3 which enables minimal execution time, schedules task n_8 to its best processor P_2, schedules task n_{11} to its best processor P_1, schedules task n_{12} to its best processor P_3, schedules task n_6 to its best processor P_4 which enables minimal execution time, schedules task n_{10} to its best processor P_1, schedules task n_{13} to its best processor P_5 which enables minimal execution time, schedules task n_9 to its best processor P_6 which enables minimal execution time, and at last, schedules task n_7 to its best processor P_8 which enables minimal execution time. The processor allocations and scheduling times produced by the HEFT algorithm are shown in Figure 4.

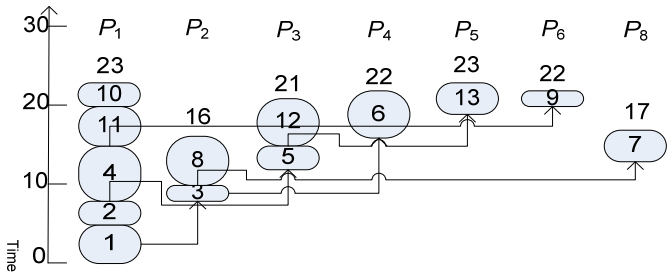

Figure 4. The Scheduling Result Produced by the HEFT Algorithm

3.2. Comparison Metrics

Except the basic metrics, i.e., the schedule length, number of used processors and time complexity, the comparisons of the algorithms are based on the following metrics:

(1) Schedule Length Ratio (SLR): The main performance measure of a scheduling algorithm on a graph is the schedule length (*makespan*) of its output schedule. Since a large set of task graphs with different properties is used, it is necessary to normalize the schedule length to a low bound, which is called the Schedule Length Ratio (SLR). The SLR value of an algorithm on a graph is defined by:

$$SLR = \frac{makespan}{\sum_{n_i \in CP_{MIN}} \min\left\{w(n_i, P_j) \mid P_j \in Q\right\}}. \quad (2)$$

The denominator is the summation of the minimum computation costs of tasks on the CP_{MIN}. (For an unscheduled DAG, if the computation cost of each node n_i is set with the

minimum value, then the critical path will be based on minimum computation costs, which is represented as $CP_{MIN.}$) The SLR of a graph (using an algorithm) cannot be less than one since the denominator is the lower bound. The task scheduling algorithm that gives the lowest SLR of a graph is the best algorithm with respect to performance.

(2) Speedup: The speedup value for a given graph is computed by dividing the sequential execution time (i.e., cumulative computation costs of the tasks in the graph) by the parallel execution time (i.e., the *makespan* of the output schedule). The sequential execution time is computed by assigning all tasks to a single processor that minimizes the cumulative of the computation costs.

$$Speedup = \frac{\min\left\{\sum_{n_i \in V} w(n_i, P_j) \mid P_j \in Q\right\}}{makespan}. \quad (3)$$

If the sum of the computation costs is maximized, it results in a higher speedup, but ends up with the same ranking of the scheduling algorithms.

(3) Efficiency: The ratio of the speedup value to the number of processors used, which is another comparison metric usually used for application graphs of real world problems.

3.3. Performance and Comparison

The example Out-Tree task graph shown in Figure 1 can be used to compare the HGAS algorithm with the H_TDS and HEFT algorithm. The H_TDS algorithm initially generates a set of clusters similar to linear clusters. Then dulplication is carried out until system resources are exhausted. The processor allocations and the scheduling times produced by the H_TDS algorithms are shown in Figure 5. Table 3 gives the comparison result.

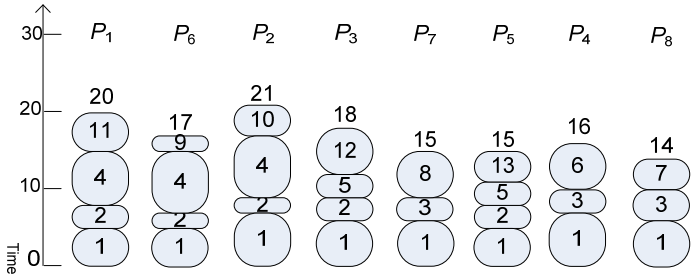

Figure 5. The Scheduling Result Produced by the H_TDS Algorithm

Table 3. Comparison with other Algorithms for DAG in Figure 1

	Schedule length	Number of used processors	Time complexity	Schedule length ratio	Speedup	Efficiency
HGAS_OT	20	6	$O(v^2p)$	1.053	2.90	0.483
H_TDS	21	8	$O(v^2p)$	1.105	2.76	0.345
HEFT	23	7	$O(v^2p)$	1.211	2.52	0.360

As shown in Figures 3-5 and Table 3, the HGAS_OT algorithm deploys an effective strategy, which effectively balances the workloads, shortens the schedule length, economizes the processors, and so improves the schedule performance. It only used six processors and its schedule length is only 20. Despite its reasonable time complexity, the H_TDS algorithm excessively uses task duplications and ignores the economization on processors, whose number of used processors is 8 and schedule length is 21. The HEFT algorithm is not duplication based and neglects the balance of the workloads, the full utilization of the computing capacity of heterogeneous processors and the reduction of the schedule length, it uses seven processors and its schedule length is, 23, more than that of the HGAS_OT algorithm. Overall, the comparison with other algorithms for DAG in Figure 1 shows that the proposed algorithm

outperforms the other two approaches in terms of schedule length ratio, speedup and efficiency metrics.

For extensive comparison, we present the comparative evaluation of the HGAS_OT, H_TDS and HEFT algorithm. We adopt the Out-Tree task graphs randomly generated by classical methods [4] as the workloads for testing these algorithms in the same HCS. Firstly 20 to 200 nodes are generated, corresponding matrices *W*, *B* and *L* are also randomly generated [4]. Visual C is used as the simulation program and the simulation is performed by the personal computer with Windows 7, 2G RAM and 2.53GHz CPU. The comparison of the number of used processors, the schedule length and the efficiency are shown in Figures 6-8, respectively.

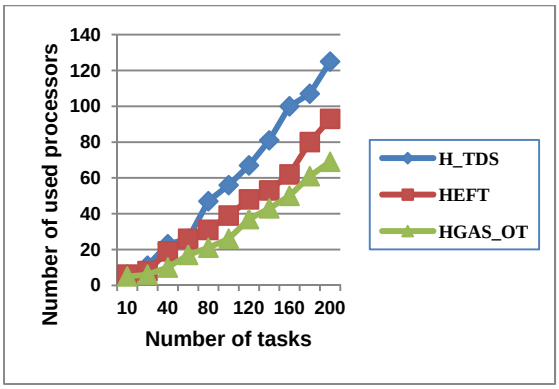

Figure 6. Comparison of the Number of Used Processors

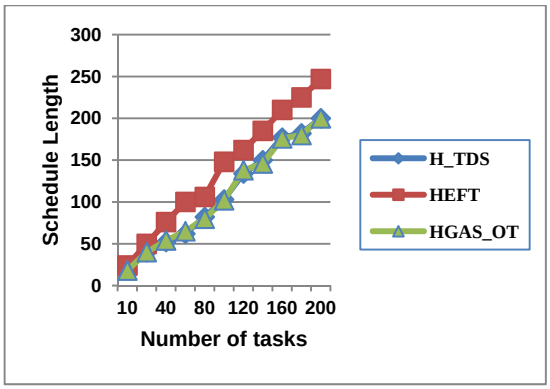

Figure 7. Comparison of the Schedule Lengths

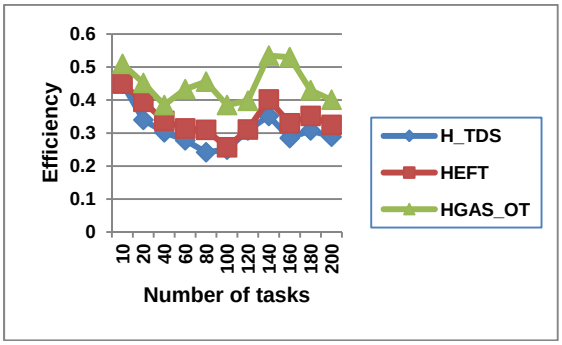

Figure 8. Comparison of the Standard Efficiencies

As shown in Figures 6-8, the HGAS algorithm has the characteristic of less number of used processors. The schedule lengths produced by the HGAS algorithm are very close to that of the H_TDS algorithm and are obviously less than that of the HEFT algorithm. Overall, as compared to other two algorithms, the HGAS_OT algorithm effectively improves the scheduling efficiency of the Out-Tree task graphs in HCSs, and the more the number of nodes in the task graph is, the more prominence the superiority in the major performances over other compared algorithms is.

4. Conclusion

Effective task scheduling is important to achieve high performance in HCSs. In this paper, we present a new greedy scheduling algorithm for scheduling Out-Tree task graphs in HCSs, called the HGAS_OT algorithm. It schedules tasks according to a new priority when scheduling each leaf node to corresponding processor, and merges the list and duplication based strategy to assign each leaf task to the most suitable processor while guaranteeing the

shorter schedule length and less number of used processors. Experimental results validate the HGAS_OT algorithm outperforms the H_TDS and HEFT algorithm when schedule length, the number of used processors, schedule length ratio and efficiency are concerned.

One planned future research is to analytically investigate the trade-off between the quality of schedules of the algorithms, i.e., average *makespan* values, and the number of processor available. This extension may come up with some bounds on the degradation of *makespan* given that the number of processors available may not be sufficient. It is also planned to extent the algorithm for more general target computing environments by considering the link contention.

Acknowledgements

This work is supported partially by the National Natural Science Foundation of China under Grant No. 71171198 to Yexin Song, the Natural Science Foundation of Naval University of Engineering under Grant No. HGDYDJJ13151 to Jianjun Zhang and under Grant No. HGDQNJJ13153 to Meini Yang, respectively.

References

[1] Foad Lotfifar, Hadi Shahriar Shahhoseini. *Complexity Task Scheduling Algorithm for Heterogeneous Computing Systems*. Third Asia International Conference on Modelling & Simulation. Bali. 2009; 5: 596-601.
[2] Jianjun Zhang, Yexin Song, Dengbin Huang. Task scheduling algorithm for Fork-Join task graghs in heterogeneous environment. *Computer Engineering & Design*. 2010; 31(3): 486-490. (In Chinese)
[3] Samantha Ranaweera, Dharma P Agrawal. *A Task Duplication Based Scheduling Algorithm for Heterogeneous Systems*. Proceedings of the 14th International Parallel and Distributed Processing Symposium. Florida. 2000; 445-450.
[4] H Topcuoglu, S Hariri, MY Wu. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Trans. Parallel and Distributed Systems*. 2002; 13(3): 260-274.
[5] T Hagras, J Janecek. *An approach to compile-time task scheduling in heterogeneous computing systems*. Proceedings of the 33rd International Conference on Parallel Processing Workshops. Canada. 2004; 182-189.
[6] Sang Cheol Kim, Sunggu Lee, Jaegyoon Hahm. Push-Pull: Deterministic Search-Based DAG Scheduling for Heterogeneous Cluster. *IEEE Trans. Parallel and Distributed Systems*. 2007; 18(11): 1489-1502.
[7] Fatma A Omara, Mona M Arafa. Genetic Algorithms for Task Scheduling Problem. *Journal of Parallel and Distributed Computing*. 2010; 70(1): 13-22.
[8] Jiadong Yang, Hua Xu, Peifa Jia. *Task Scheduling for Heterogeneous Computing based on Bayesian Optimization Algorithm*. International Conference on Computational Intelligence and Security. Beijing. 2009; 1: 112-117.
[9] B Demiroz, HR Topcuoglu. Static task scheduling with a unified objective on time and resource domains. *The Computer Journal*. 2006; 49(6): 731–743.