Proceedings of the 5th WSEAS International Conference on Telecommunications and Informatics, Istanbul, Turkey, May 27-29, 2006 (pp497-502)

A Critical Predecessor Duplication Scheduling Algorithm for Distributed Heterogeneous Computing Environments

KUAN-CHOU LAI[a], CHUN-MEI LEE[b] and JYWE-FEI FANG[c]
Department of {Computer and Information Science[a], Information Management[b], Digital Content and Technology[c]}
National Taichung University[ac], TungNan Institute of Technology[b]
No. 140, Min-Sheng Rd., Taichung[ac]; No.152, Sec.3, PeiShen Rd., ShengKeng, Taipei[b]
Taiwan, Republic of China

Abstract: - This paper proposes a new duplication-based scheduling algorithm, called the Critical Predecessor Duplication scheduling algorithm, to exploit the potential of parallel processing in distributed heterogeneous computing systems. In such systems, designing efficient scheduling algorithm may lead to a significant variation in the generated schedule. The proposed scheduling algorithm could improve system utilization, avoid the redundant resource consumption, and then get the better schedules. Experimental results show the superiority of this algorithm to those presented in previous literature.

Key-Words: - directed acyclic graph, heterogeneity, duplication, task scheduling, parallel processing

1 Introduction

In the recent past, the efficiency of the distributed heterogeneous computing (DHC) system has been extensively explored for high performance computation. Distributed heterogeneous systems built with off-the-shelf components not only are easy to build but also decrease the lead time to market. However, their incremental expansion capability usually makes the heterogeneity of DHC systems more obvious. Distributed heterogeneous computing systems are gaining popularity over supercomputers due to the collective processing capabilities for diverse computational requirements. At the same time, however, the effective use of heterogeneous computing resources usually remains a major challenge.

A general technique adopted in distributed heterogeneous computing systems is to partition an application into a set of parallel tasks and to schedule them separately onto different processing elements (PEs). Task scheduling is to assign the tasks to a computing system and to arrange the sequence of tasks' executions in consideration of diverse resource requirements, such that the precedence relationships between tasks are not violated and the diverse resource contentions are circumvented to minimize the final completion time [9]. However, the scheduling problem is NP-complete [3] and has been studied extensively in literature.

In the static scheduling approach, these proposed heuristics could be broadly classified into a variety of categories, such as the list-based [10], clustering-based [4] and duplication-based [6]. The duplication-based approach is an attractive technique, which tries to duplicate the parents of the candidate task to more than one PE to reduce the candidate task's starting or finishing time by decreasing the communication overhead. In general, list or clustering algorithms with duplication tend to perform better than non-duplication algorithms. However, the redundant duplications without contributing performance are also observed in many duplication algorithms.

In this study, a heuristic algorithm, called the Critical Predecessor Duplication (CPD) scheduling algorithm, is proposed to schedule tasks by taking into account the system heterogeneity. The condition of strict reduction of the schedule length obtained by the CPD scheduling algorithm is also defined. It ensures that the completion time will be strictly curtailed after applying the CPD scheduling algorithm. Therefore, the CPD scheduling algorithm could produce better schedules than those obtained by existing algorithms. The preliminary experimental results show the superiority of the CPD scheduling algorithm.

The rest of this paper is organized as follows. Section 2 introduces the scheduling problem. Section 3 presents the proposed algorithm. Experimental results and performance analyses are provided in Sections 4. Conclusions are then offered.

2 Problem Formulation

As a rule, the classical scheduling algorithms schedule parallel tasks to attain the minimal completion time for a program based on the macro-dataflow model [13]. This model assumes

that the bandwidth of communications is not limited [6].

A program is presented as a directed acyclic graph (DAG) G = (N, E, T, C), where N is the set of task nodes, T is the set of node computation volumes, E is the set of communication edges that define a partial order or precedence constraints on N, and C is the set of communication volumes. The value $\tau_i \in$T is the computation volume for $n_i \in$ N. The value of $c_{ij} \in$ C is the communication volume occurring along the edge $e_{ij} \in$ E, where n_i, $n_j \in$ N.

In this model, we assume that a task is indivisible, and that a PE executes one task at a time. Task's execution would be triggered after satisfying precedence constraints and removing resource contentions. Precedence constraints occur only when the execution of one task is postponed until all the data from its immediate predecessors arrive. Removing resource contentions includes removing the computational resource's contention, in which a task's execution must be deferred until the completion of all the tasks scheduled before it within the same PE.

Suppose that a DHC system is represented as M = (P, Q, A, B), where P=$\{p_i \mid p_i \in$ P, i = 1, ... ,|P|$\}$ is the set of heterogeneous PEs, A = $\{\alpha(p_i) \mid \alpha(p_i) \in$ A, i = 1,...,|P|$\}$ is the set of the execution rates for heterogeneous PEs, $\alpha(p_i)$ is the execution rate for p_i, Q = $\{q(p_i,p_j) | q(p_i,p_j) \in$Q, i, j= 1..|P|$\}$ is the set of communication channels, $q(p_i,p_j)$ is the communication channel from p_i to p_j, and B = $\{\beta(p_i,p_j) | \beta(p_i,p_j) \in$B, i, j=1..|P|$\}$ is the set of the transferring rates for communication channels from p_i to p_j. Our study assumes that each PE has a coprocessor to deal with communications, which allow computations and communications that are independent of each other to be overlapped. Formally, let $\tau_i \times \alpha(p_k)$ be the computation cost when task n_i is allocated to p_k, and $c_{ij} \times \beta(p_k,p_l)$ be the communication cost from task n_i to task n_j, where n_i is allocated to p_k and task n_j is allocated to p_l.

Given a DAG and a system model as described above, the scheduling problem is to obtain the minimal-length, non-preemptive schedule of the task graph in the DHC system.

3 The Proposed Algorithm

Formally, the data ready time, $drt(n_i)$ of task n_i, is defined as the latest arrival time of communication data from its predecessors. After satisfying the precedence constraints and removing resource contentions, the earliest starting time of node n_i, $est(n_i)$ is the earliest time when node n_i can start execution. Let the earliest completion time of node n_i, $ect(n_i)$ be defined as follows:

$$ect(n_i) = est(n_i) + \tau_i \times \alpha(p(n_i)), \quad (1)$$

where $p(n_i) \in$ P is the PE that executes n_i. Let $pred(n_i)$ be the set of immediate predecessor nodes of n_i. Initially, $\forall n_i \in$N and $pred(n_i)=\varnothing$, let $est(n_i)= 0$.

Assume that $succ(n_i)$ is the set of immediate successor nodes of n_i. The b_level [1] of node n_i, $b\_level(n_i)$ is the length of the longest path from this node to the sink node. Formally, $\forall\, n_i \in$ N,

$$b\_level(n_i)=max(c_{ij} \times \overline{\beta} + \tau_j \times \overline{\alpha}(n_j) + b\_level(n_j)), \quad (2)$$

where $n_j \in succ(n_i)$, $\overline{\alpha}(n_j)$ is the average computation rate of task n_j, and $\overline{\beta}$ is the average communication rate.

In the scheduling process, a task is called *examined* after it is scheduled to some PE and *unexamined* before it is scheduled to some PE. There are three types of unexamined tasks, including (a) the ready tasks whose immediate predecessors are all examined, (b) the partially ready tasks which have at least one examined immediate predecessor and at least one unexamined immediate predecessor, and (c) the unready tasks whose immediate predecessors are all unexamined.

Here, we define the u_level(n_i) of an unexamined ready or partially ready node n_i as the shortest length from entry nodes to it. Formally, $\forall n_j \in pred(n_i) \cap$ examined set, and $\forall n_k \in pred(n_i) \cap$ unexamined set,

$$u\_level(n_i)=min(ect(n_j)+ c_{ji} \times \overline{\beta}(p(n_j)) , u\_level(n_k) + \tau_k \times \overline{\alpha} + c_{ki} \times \overline{\beta}(p(n_k))), \quad (3)$$

where $\overline{\beta}(p(n_j))$ is the average communication rate from $p(n_j)$ to other PEs respectively.

Due to that the CPD algorithm tries to strictly reduce the schedule length, the max-min parallel processing anomaly [2] is introduced next. This anomaly occurs when the completion time obtained after parallelizing is worse than that obtained by sequential execution, caused by a tradeoff between maximizing parallelism and minimizing inter-task communication. To overcome the max-min anomaly, the intermediate schedule length should be reduced monotonically. Then, the critical task must be identified at each scheduling step; and the challenge is to find the selection priority that could well reflect the importance of the critical tasks in the context of scheduling. Due to that the critical path may change in each scheduling step, only the task on the critical path with an immediate predecessor that is also on the critical path and is examined should be considered. Such a task is called the *Critical Task* (CT). To identify the critical task, all the immediate successors of any examined tasks are considered, and the task n_{ct}

with the maximal value of u_level(n_{ct}) + $\tau_{ct} \times \overline{\alpha}(n_{ct})$ +b_level(n_{ct}) is identified as the critical task.

Usually, most of the proposed algorithms fail to consider the schedule holes [7] problem. The CPD scheduling algorithm employs these schedule holes by inserting the duplicated parent tasks of a candidate task for advancing its earliest starting time.

In this study, the *critical predecessor* (CP) is a predecessor of a task; when one task's CP finishes executing, the task will satisfy its precedence constraints and remove its resource contentions. In another word, the advance of the earliest starting time of a task would be restricted by its critical predecessor.

In the distributed heterogeneous computing environment, a PE that allows the earliest start time is not necessarily the PE that allows the earliest completion time for a node; therefore, the CPD scheduling algorithm schedules the candidate node to the PE that permits its earliest completion time. The candidate selection depends on $priority(n_{mp}) = max(\ b\_level(n_{mp}) - \tau_{mp} \times \overline{\alpha}(n_{mp}) - u\_level(n_{mp}))$, where n_{mp} is the task with the maximal priority value. The priority function is derived from three factors:

(a) when two tasks are ready and their values of b_level minus execution cost are equal, the task that could be issued earlier should be scheduled first,

(b) when two ready tasks have the same values of execution cost plus u_level, the one having the larger remaining work should also be scheduled first, and

(c) when two ready tasks have the same values of b_level minus u_level, the task that could be scheduled to the PE with the smaller execution rate (i.e., higher speed) should be scheduled first to shorten the task's execution time.

The CPD scheduling algorithm is presented below. First, the CPD scheduling algorithm finds the average execution rates and the average communication rates. Each task is initially assigned to one virtual PE, and the communication overhead between tasks is assumed to be the average communication rate times the communication volume between tasks. And then the CPD scheduling algorithm estimates b_levels for all tasks by using the average execution rates for all heterogeneous PEs bottom-up. Let the tasks without predecessors be in the ready set. When the ready set is not empty, the CPD scheduling algorithm repeats the following steps:

1 Calculate $priority(n_i)$ of tasks in the unexamined ready set.
2 Find the candidate, n_{mp}, with maximal $priority(n_{mp})$.
3 Let n_{ct} be the task with the max. priority in the set of all immediate successors of any examined tasks.
4 Find $p(n_{mp})$ where the $ect(n_{mp})$ could be obtained.
5 Find out n_{mp}'s CP, n_{cp}, and its $drt(n_{cp})$.
6 If there is an enough free time slot from $drt(n_{cp})$ to $est(n_{mp})$ to schedule n_{cp} to $p(n_{mp})$ then
6.1 If $n_{ct} = n_{mp}$ then
6.1.1 Schedule the duplicated n_{cp} to $p(n_{mp})$ to advance $est(n_{mp})$ by the insertion policy.
6.2 Else (if $n_{ct} \neq n_{mp}$)
6.2.1 Test whether duplicated n_{cp} to $p(n_{mp})$ would affect $est(n_{ct})$
6.2.2 If it won't affect, then schedule the duplicated n_{cp} to $p(n_{mp})$ to advance the $est(n_{mp})$ by the insertion policy
6.3 End if
7 End if
8 Allocate n_{mp} to $p(n_{mp})$, and make n_{mp} examined.
9 Re-check the ready set, and all u_levels.
10 Repeat steps 1 to 9 until all tasks are examined.

Fig. 1 is an example to show the superiority of the proposed CPD scheduling algorithm. In order to simplify the analyzing process, we assume that the computation cost of all tasks in all PEs are estimated in the Table 1, and that the communication rates of all communication channels are all one (i.e., $\overline{\beta}$ =1 for all pair inter-PE communications); therefore, the numbers listed along edges are communication volume and also could be considered as the communication costs when the parent node and its child node are scheduled in different PEs. The scheduling results obtained from the HCPFD and CPD scheduling algorithms are shown in Fig. 2.

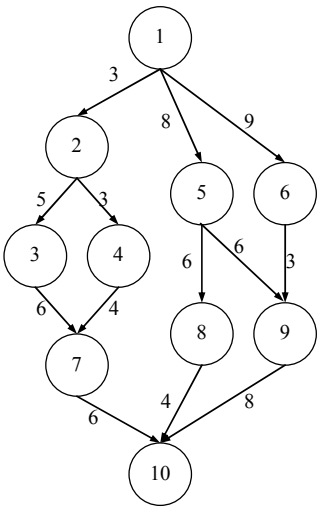

Fig. 1. An example of DAG.

In the CPD scheduling algorithm, the time complexity for calculating the b_level is O((|N|+|E|)|P|). The steps 1 to 9 are executed in

O(|N|), and finding n_{mp} and its related PE is O(|N|+|P|). Finding out the n_{cp} needs O(|N|) steps. Therefore, the time complexity of the CPD scheduling algorithm is O((|N|+|E|)(|P||N|)). Consequently, in practical applications, the complexity of the CPD scheduling algorithm is reasonable.

Table 1. Tasks' computation costs in different PEs.

Task n_i \ PEs	PE1	PE2	PE3
1	5	3	4
2	3	2	7
3	2	4	9
4	7	5	3
5	8	6	7
6	3	4	8
7	4	6	2
8	2	3	4
9	5	5	2
10	6	4	5

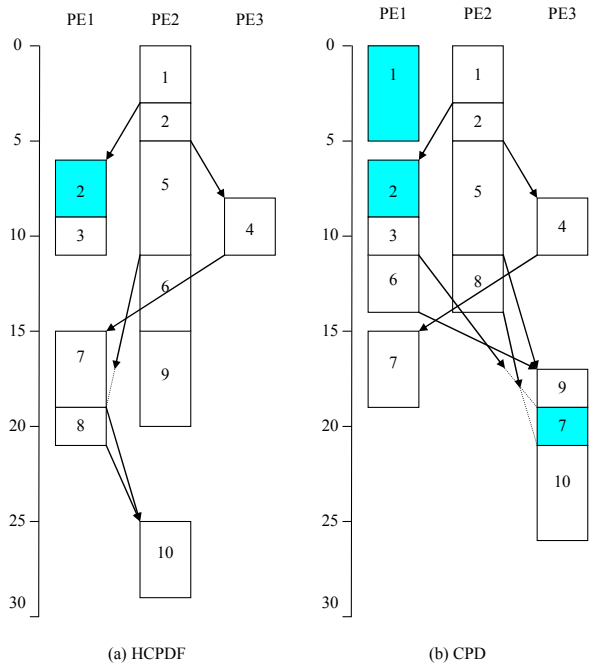

Fig. 2. Scheduling results for different algorithms.

4 Experimental Results

In this section, the simulation environment is built to demostrate the performance of the CPD scheduling algorithm by evaluating five practical applications, which include the Gauss–Jordan Elimination, the Fast Fourier Transformation, LU factoring, Fork trees, and Join trees. Their graph sizes vary from a minimal of 378 nodes to a maximum of 511 nodes. The computation/communication volumes required for each task are pre-determined for different applications. The number of PEs is set to 1, 2, 4, 8, 16, or 32. In this study, the distribution of execution rates represents the heterogeneity of computational power in a DHC system; therefore, we assume that the distribution of the execution rate is normally distributed: its mean value is 10, and the variance may be 0.2, 0.4, 0.6, 0.8, 1, 2, 4, or 8. The heterogeneity of the communication mechanisms between PEs is represented by the distribution of transfering rates. We assume that the distribution of the transfering rate is also normally distributed: its mean value is 10, and the variance may be 0.2, 0.4, 0.6, 0.8, 1, 2, 4, or 8. As these variances increase, the system heterogeneities become more obvious.

Three algorithms (Critical Nodes Parent Trees, CNPT [10]; Task duplication-based scheduling Algorithm for Network of Heterogeneous systems, TANH [6]; and Heterogeneous Critical Parents with Fast Duplicator, HCPFD [11]) are applied to comparatively demonstrate the proposed algorithm's effectiveness. In general, the HCPFD scheduling algorithm outperforms the CNPT and TANH algorithms. Therefore, only the differences between the HCPFD scheduling algorithm and the CPD one are presented as follows. First, due to that the HCPFD scheduling algorithm determines the scheduling sequence in the listing phase, it doesn't consider the effect that the critical path may change in each scheduling process. Second, the HCPFD scheduling algorithm tries to duplicate the candidate's critical parent at the idle time slot between the start time of the candidate's critical parent and the completion time of the last previous task assigned in the same PE. However, the CPD scheduling algorithm duplicate the candidate's critical parent at the idle time slot from the data ready time of the candidate's critical parent to the start time of the candidate node. It could increase the possibility of finding the free time slot for these duplicated parents. Third, owing to that the CPD scheduling algorithm initially assigns each task to one virtual PE, and then tries to reduce the time complexity by avoiding from re-evaluating the ready-task-PE pairs in every scheduling step, the time complexity is reduced.

Table 2. Algorithm Comparison in terms of Scheduling Lengths.

		CNPT	TANH	HCPFD
CPD	better than	90.67%	83.14%	76.46%
	equal to	0.12%	2.58%	3.67%
	worse than	9.21%	14.28%	19.87%

Table 2 shows that the CPD scheduling algorithm

outperforms the CNPT, TANH and HCPFD ones in terms of the schedule lengths for practical applications. The average schedule lengths obtained by the CPD scheduling algorithm are shorter than those obtained by the HCPFD scheduling algorithm in 76.46% cases, by the TANH scheduling algorithm in 83.14% cases, and by the CNPT scheduling algorithm in 90.67% cases. The experimental results also indicate that the CPD scheduling algorithm performs better than the CNPT, TANH or HCPFD scheduling ones in terms of handling the system heterogeneity.

Fig. 3 shows the average scheduling lengths for different applications by varying the number of PEs. In Fig. 3(a), the schedule lengths obtained by the CPD scheduling algorithm are shorter than that obtained by the other three algorithms. It implies that the scheduling performance of the CPD scheduling algorithm is better than that of others. The main reason is that the CPD scheduling algorithm dynamically decides the critical node in each scheduling step; however, the HCPFD algorithm decides the scheduling sequence in the listing phase, and then lacks enough information to reduce the schedule length strictly at each scheduling step. Fig. 3(b) ~ (e) also shows the same experimental results. In general, the CPD scheduling algorithm monotonically reduces the schedule length as the number of PEs increases. It implies that the strict reduction condition can indeed avoid the max-min anomaly.

Fig. 4 shows that the average system utilizations obtained by the CNPT, TANH, HCPFD and CPD scheduling algorithms reduce monotonically as the number of PEs increases. It is reasonable that the system utilization reduces as the number of PEs increases. The experimental results show that the CPD scheduling algorithm outperforms the other three algorithms in terms of the system utilization. The main reason is that the CPD scheduling algorithm dynamically duplicates the candidate's critical parent at the idle time slot from the data ready time of the candidate's critical parent to the start time of the candidate node. Therefore, there is more possibility of exploiting resource utilization.

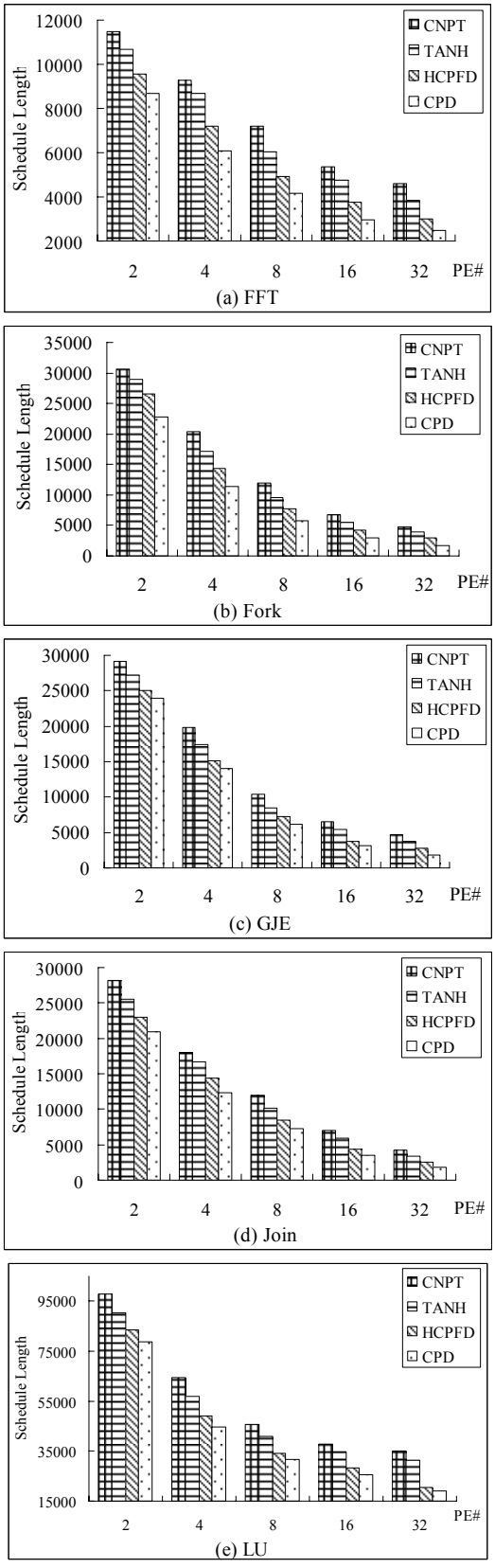

Fig. 3 Scheduling lengths by varying # of PEs.

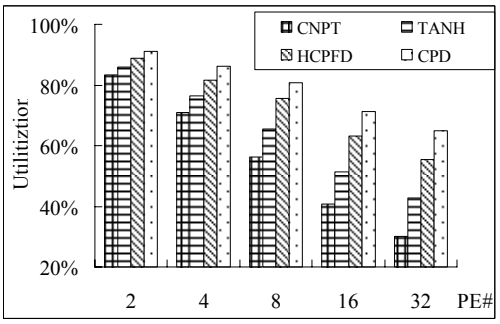

Fig. 4. Average utilization by varying # of PEs.

5 Conclusions

In this paper, we have discussed a scheduling algorithm for heterogeneous computing environments called the CPD scheduling algorithm with time complexity O((|N|+|E|)(|P||N|)). In practical applications, the complexity of the CPD scheduling algorithm is reasonable. The CPD scheduling algorithm is proposed with higher scalability and lower redundant resource consumption in DHC systems. This study examined five practical applications, but other parallel applications should be also examined. Experimental results show the superiority of our proposed algorithm to those presented in previous literature, and also show that the scheduling performance is affected by the heterogeneity of computational power, the heterogeneity of communication mechanisms and the program structure of applications. The experimental results of this study are still preliminary and improvements are ongoing.

Acknowledgement

This work was partially supported by the National Science Council of the Republic of China under the contract number: NSC94-2213-E-018-014, as well as a grant from the Science Promotion Foundation in National Taichung University under the grant number: NTCU-94014.

References:

[1] A. Gerasoulis, T. Yang, On the Granularity and Clustering of Directed Acyclic Task Graphs, *IEEE Transactions on Parallel and Distributed Systems*, Vol.4, No.6, 1993, pp.686-701.

[2] B. Kruatrachue, T. Lewis, Grain Size Determination for Parallel Processing, *IEEE Software*, Vol.5, No.1, 1988, pp. 23-32.

[3] J.D. Ullman, NP-Complete Scheduling Problems, *Journal of Computing System Science*, Vol.10, 1975 pp.384-393.

[4] M.A. Palis, J.-C. Liou, D.S.L. Wei, Task Clustering and Scheduling for Distributed Memory Parallel Architectures, *IEEE Transactions on Parallel and Distributed Systems*, Vol.7, No.1, 1996, pp.46-55.

[5] O. Sinnen, L. Sousa, List Scheduling: Extension for Contention Awareness and Evaluation of Node Priorities for Heterogeneous Cluster Architectures, *Parallel Computing*, Vol.30, No.1, 2004, pp.81-101.

[6] Rashmi Bajaj, Dharma P. Agrawal, Improving Scheduling of Tasks in a Heterogeneous Environment, *IEEE Transactions on Parallel and Distributed Systems*, Vol.15, No.2, 2004, pp.107-118.

[7] S. Selvakumar, C. Siva Ram Murthy, Scheduling Precedence Constrained Task Graphs with Non-Negligible Intertask Communication onto Multiprocessors, *IEEE Transactions on Parallel and Distributed Systems*, Vol.5, No.3, 1994, pp.328-336.

[8] Savina Bansal, Padam Kumar, Kuldip Singh, An Improved Duplication Strategy for Scheduling Precedence Constrained Graphs in Multiprocessor Systems, *IEEE Transactions on Parallel and Distributed Systems*, Vol.14, No.6, 2003, pp.533-544.

[9] T. Braun, H.J. Siegel, N. Beck, L.L. Boloni, M. Maheswaran, A.I. Reuther, et al, A Comparison Study of Static Mapping Heuristics for a Classes of Meta-Tasks on Heterogeneous Computing Systems, *Proc. Heterogeneous Computing Workshop*, 1999, pp.15-29.

[10] Tarek Hagras, Jan Janecek, A High Performance, Low Complexity Algorithm for Compile-Time Job Scheduling in Homogeneous Computing Environment, *IEEE Int. Conf. on Parallel Processing Workshops* 2003.

[11] Tarek Hagras, Jan Janecek, A High Performance, Low Complexity Algorithm for Compile-Time Task Scheduling in Heterogeneous Systems, *IEEE Int. Parallel and Distributed Processing Symposium*, 2004.

[12] Tarek Hagras, Jan Janecek, A Near Lower-Bound Complexity Algorithm for Compile-Time Task-Scheduling in Heterogeneous Computing Systems, *IEEE Proceedings of the ISPDC/HeteroPar* 2004.

[13] V. Sarkar, *Partitionning and Scheduling Parallel Programs for Execution on Multiprocessors*, MIT Press; 1989.

[14] Y.-K. Kwok, I. Ahmad, Link Contention-Constrained Scheduling and Mapping of Tasks and Messages to a Network of Heterogeneous Processors, *Cluster Computing: Journal of Networks, Software Tools, and Applications*, Vol.3, No.2, 2000, pp. 113-124.