J Supercomput (2008) 44: 126–145
DOI 10.1007/s11227-007-0152-2

A dominant predecessor duplication scheduling algorithm for heterogeneous systems

Kuan-Chou Lai · Chao-Tung Yang

Published online: 29 September 2007
© Springer Science+Business Media, LLC 2007

Abstract This paper proposes a new duplication-based task scheduling algorithm for distributed heterogeneous computing (DHC) systems. For such systems, many researchers have focused on solving the NP-complete problem of scheduling directed acyclic task graphs to minimize the makespan. However, the heterogeneity of computational resources and communication mechanisms poses some major obstacles to achieving high parallel efficiency. This paper proposes a heuristic strategy called the Dominant Predecessor Duplication (DPD) scheduling algorithm, which allows for system heterogeneities and communication bandwidth to exploit the potential of parallel processing. This algorithm can improve system utilization and avoid redundant resource consumption, resulting in better schedules. Experimental results show that the system heterogeneities and program structures of applications affect scheduling performance, and that our presented algorithm is better able to avoid these problems than those presented in previous literature. Here, we show that our algorithm can be applied to design efficient distributed systems to overcome performance bottlenecks caused by system heterogeneities.

Keywords Directed acyclic graph · Heterogeneity · Duplication · Task scheduling · Distributed computing

K.-C. Lai (✉)
Department of Computer and Information Science, National Taichung University, Taichung 40302, Taiwan
e-mail: kclai@mail.ntcu.edu.tw

C.-T. Yang
Department of Computer Science and Information Engineering, Tunghai University, Taichung 40704, Taiwan
e-mail: ctyang@thu.edu.tw

1 Introduction

With advances in high-speed networks and CPU hardware technology, distributed heterogeneous computing (DHC) is applied in terms of performance and cost-effectiveness for applications with diverse computational requirements. Generally, a DHC system consists of high-speed networks, a middleware, and a suite of distributed off-the-shelf heterogeneous processing elements (PEs) [1–5]. There are at least two salient features in DHC systems: the heterogeneity of computational resources (e.g., the different execution times of tasks run on different PEs), and the heterogeneity of the network performance between PEs. Traditional parallel applications distribute computations evenly across PEs and cannot balance the load of heterogeneous systems. Consequently, faster PEs stall at the synchronization points while waiting for slower PEs to perform their tasks. In particular, latency and bandwidth limitations induced by a network can seriously degrade parallel application performance [5–8]. These limitations remain a major challenge to using heterogeneous computation and communication resources effectively, and they call for needed improvement.

When the application's characteristics (which include task execution times, task dependencies and amounts of communicated data) are known a priori, an application can be modeled as a directed acyclic graph (DAG), where the nodes represent the computational tasks and the edges represent the intertask communications (or data dependence). The static scheduling problem is to assign tasks to a distributed system and arrange the tasks' executions by the diverse resource requirements, such that precedence relationships between tasks are not violated, diverse resource contentions are circumvented, and the data transferred between the tasks is orchestrated to obtain a minimum overall completion time [3, 9] at the compile time.

There are many well-known scheduling problems in operations research and computer science, most of which are NP-complete [10, 11] and have been studied extensively in the literature. To exploit heterogeneous computational resources, many researchers have focused on obtaining near optimal solutions within an acceptable time complexity. In general, the scheduling strategies perform poorly in DHC systems because of the heterogeneity of computational resources and communication mechanisms. If the scheduling problem is not properly handled, then any gain obtained from the parallelization of the application would be diminished.

In the static scheduling approach, these proposed heuristics are classified into a variety of categories, such as the list-based approach, the clustering-based approach, and the duplication-based approach. The simple and attractive list-based scheduling algorithm assigns priorities to tasks statically or dynamically, and then repeatedly allocates the highest priority ready task to its most suitable PE to minimize the schedule length. However, because each task is scheduled without regard for subsequent tasks, the priority assignment does not always lead to optimized scheduling. This adverse effect is particularly severe when communication message scheduling is considered in DHC systems. A later task may be delayed because its incoming messages are affected by the inefficient scheduling of previous messages and it, therefore, cannot occupy an earlier time slot. The performance of list-based approaches suffers due to the trade-off between maximizing parallelism and minimizing communication, and worsens for fine grain tasks with a high communication to computation cost ratio

(CCR) [8]. There have been several list-based algorithms proposed for DHC systems, e.g., the Iso-Level Heterogeneous Allocation (ILHA) algorithm [12], Critical Nodes Parent Trees (CNPT) [13], Dynamic Level Scheduling (DLS) [14], Bubble Scheduling and Allocation (BSA) [5], the Heterogeneous Earliest Finish Time (HEFT), and Critical Path on a Processor (CROP) technique [1].

The clustering-based approach [15–18] tries to allocate heavily communicating tasks onto the same PE to reduce the overall communication cost, and includes two phases. In the first phase, tasks are grouped into an unbounded number of clusters using clustering heuristics, and then these clusters are mapped onto the available PEs by load-balancing heuristics. Although the complexity of the clustering-based approach is generally lower than that of other approaches, the performance of the clustering-based approach is still worse than that of other approaches [2].

Another attractive technique is the duplication-based approach, which aims to duplicate the parents of the candidate task to more than one PE to reduce the candidate task's start or finish time by decreasing the communication overhead. Generally, list or clustering algorithms with duplication tend to perform better than nonduplication algorithms; however, the performance improvement, which originates from the exhausted backward search in the duplication step, usually leads to a higher degree of complexity and causes redundant duplications without contributing to performance. There are several duplication-based algorithms, including the Critical Path based Full Duplication algorithm (CPFD) [19], the Heterogeneous Critical Parents with Fast Duplicator (HCPFD) [4, 20], the Bottom-up Top-down Duplication Heuristic (BTDH) [21], the Task Duplication-based Scheduling Algorithm for Network of Heterogeneous Systems (TANH) [2], and the Heterogeneous Critical Tasks Reverse Duplicator (HCTRD) [22].

Most of these algorithms only consider computational aspects, and adopt the macro-dataflow model. They assume that target systems include fully-connected PEs and have dedicated communication subsystems without any communication contention. In fact, communication contentions are important factors to consider in the production of accurate and efficient schedules [5–8]. If a communication resource is occupied by one task, then any other tasks requiring the same resource have to wait until it becomes available. As the computational power increases, communication mechanisms may cause bottlenecks in the system, particularly when executing applications with large communication requirements. Therefore, it is clear that the scheduling problem must be studied from both the computation and communication points of view.

In our study, a heuristic algorithm, called the Dominant Predecessor Duplication (DPD) scheduling algorithm, is proposed to schedule tasks according to system heterogeneity, network bandwidth, and the communication requirements of applications. The strict reduction of schedule lengths obtained by the DPD algorithm is defined to reduce the intermediate schedule length monotonically at each scheduling step. This condition shows that the completion time of a directed acyclic task graph can be strictly curtailed through the application of the DPD algorithm. In order to improve system utilization, the proposed algorithm also uses limited-length time slot lists to exploit schedule holes [8]. The DPD algorithm also takes care of the dynamically-changing critical path as the scheduling progress proceeds to accurately capture the critical tasks.

Here, we show that our DPD algorithm can avoid redundant resource consumption and produce better schedules than those obtained through existing algorithms. Experimental results are presented to verify the improvement in performance, and we have also applied 3 algorithms proposed in other studies to demonstrate the comparative effectiveness of our algorithm.

In Sect. 2 of this paper, we introduce the scheduling problem. In Sect. 3, we present our proposed algorithm, and the experimental results and performance analyses are provided in Sect. 4. Our conclusions are offered at the end of the paper.

2 Scheduling problem

In this section, we define the program and system models and present the definition of the scheduling problem.

2.1 Program model

As a rule, classical scheduling algorithms schedule parallel tasks to attain a program's minimum completion time based on the macro-dataflow model [22]. This model assumes that the bandwidth of communications performed at the same time is unlimited [2] (i.e., instantaneous contention-free transmission). However, in real systems, contention among communication mechanisms must be taken into account. Our program model enlarges the macro-dataflow program description to allow for scheduling heuristics to be used in DHC systems.

A program is presented as a directed acyclic graph (DAG). The DAG is defined as G = (N, E, T, C), where N is the set of tasks, T is the set of computation volumes (i.e., one unit of computation volume is one million instructions), E is the set of communication edges that define precedence constraints on N, and C is the set of communication volumes (i.e., one unit of communication volume is a kbyte). The value of $\tau_i \in$ 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.

As only static heuristics are discussed here, we assume that the number of tasks, the number of PEs, and the accurate estimations of the expected execution and communication volumes for each task are static and known beforehand. A task is an indivisible unit of computation (i.e., nonpreemptive), and satisfying precedence constraints and removing resource contentions trigger the execution of a task. Precedence constraints only occur when the task's execution is postponed until all the data from its immediate predecessors arrives. The removal of resource contentions includes: (a) removing computational resource contentions, in which case a task's execution is deferred until all the tasks scheduled before it within the same PE are complete, and (b) removing communication resource contentions, in which case data is received sequentially from the same communication channel.

2.2 System model

In this study, system resources only include computational resources and communication mechanisms. However, our proposed system model can also be extended to include diverse system resources.

Suppose that a DHC system is represented by M = (P, Q, A, B), where P = $\{p_i \mid p_i \in \text{P}, i = 1, \ldots, |\text{P}|\}$ is the set of PEs, A = $\{\alpha(p_i) \mid \alpha(p_i) \in \text{A}, i = 1, \ldots, |\text{P}|\}$ is the set of execution rates, $\alpha(p_i)$ is the execution rate for p_i (i.e., the unit is time/instruction counts, such as seconds/one million instructions), Q = $\{q(p_i, p_j) \mid q(p_i, p_j) \in \text{Q}, i, j = 1, \ldots, |\text{P}|\}$ is the set of communication channels, $q(p_i, p_j)$ is the communication channel from p_i to p_j, and B = $\{\beta(p_i, p_j) \mid \beta(p_i, p_j) \in \text{B}, i, j = 1, \ldots, |\text{P}|\}$ is the set of transfer rates for the communication channels from p_i to p_j (i.e., the unit is time/volume, such as seconds/kbytes). We assume that each PE has a coprocessor to deal with communications, which allows for computations and communications that are independent of each other to overlap [7, 23]. Formally, let $\tau_i \times \alpha(p_k)$ be the computation cost when n_i is allocated to p_k, and let $c_{ij} \times \beta(p_k, p_l)$ be the communication cost from n_i to n_j, where n_i is allocated to p_k and n_j is allocated to p_l.

In general, communication contention [7] includes end-point contention [24] and network contention [14]. To manifest the probable resource contentions, we assume that when a resource is occupied by one communication and is insufficient for other requirements, any other communication requiring the same resource has to wait until it becomes available.

2.3 Problem definition

Considering the DAG and the system model described above, the goal of the scheduling problem is obtaining the minimum-length nonpreemptive schedule for task graphs. To avoid a high degree of complexity, we only consider the nonbacktracking algorithm; therefore, the number of scheduling steps remains polynomial-bounded with respect to DAG size. To further simplify the analysis, we neglect the additional overhead of transforming a serial algorithm into a parallel form, and assume that no additional processing cost is required to execute programs in DHC systems.

3 The proposed algorithm

Before introducing our DPD algorithm, it is useful to discuss the related terminology. Our notations are summarized in Table 1.

3.1 Related terminology

Formally, let $pred(n_i)$ be the set of immediate predecessors of n_i. The data ready time, $drt(n_i)$, of n_i is defined as the latest arrival time of communication data from its predecessors (i.e., the data ready time of n_i is the earliest time when n_i has satisfied precedence constraints and received all its input data). However, even after receiving all the input data, the execution of a task might not be triggered because of unremoved resource contentions. After satisfying precedence constraints and removing any resource contentions, the earliest starting time of n_i, $est(n_i)$, is the earliest time when n_i can begin its execution. Let the earliest completion time of n_i, $ect(n_i)$, be defined as:

$$ect(n_i) = est(n_i) + \tau_i \times \alpha\big(p(n_i)\big), \tag{1}$$

Table 1 Summary of notations

Notation	Description
p_i	the ith processing element
n_i	the ith task node
$p(n_i)$	the processing element which n_i is allocated
τ_i	the computation volume for $n_i \in \mathrm{N}$
c_{ij}	the communication volume occurring along the edge e_{ij}
$\alpha(p_i)$	the execution rate for p_i
$\overline{\alpha}(p(n_i))$	the average execution rate for processing elements to which n_i is allocated
$\beta(p_i, p_j)$	the transferring rates for communication channels from p_i to p_j
$\overline{\beta}(p_i)$	the average transferring rates for communication channels from p_i to other PEs
$\overline{\beta}$	the average transferring rates for all communication channels
$pred(n_i)$	the set of immediate predecessors of n_i
$succ(n_i)$	the set of immediate successors of n_i
$drt(n_i)$	the latest arrival time of communication data from its predecessors
$est(n_i)$	the earliest time when node n_i can start execution
$ect(n_i)$	the earliest completion time of node n_i
$b\text{-}level(n_i)$	the length of the longest path from n_i to an exit node
$u\text{-}level(n_i)$	the longest possible length from the entry nodes to n_i
n_{mp}	the task with the maximal priority value
n_{dt}	the dominant task
n_{dp}	the dominant predecessor

where $p(n_i) \in \mathrm{P}$ is the PE that executes n_i. Assume that $succ(n_i)$ is the set of immediate successors of n_i. Initially, $\forall n_i \in \mathrm{N}$ and $pred(n_i) = \varnothing$, let $est(n_i) = 0$.

The *b-level* [9, 25] of n_i, $b\text{-}level(n_i)$, is the length of the longest path from n_i to an exit node. We elaborate the formula by taking system heterogeneity into consideration. Formally, $\forall n_i \in \mathrm{N}$,

$$b\text{-}level(n_i) = \max\bigl(c_{ij} \times \overline{\beta} + \tau_j \times \overline{\alpha}\bigl(p(n_j)\bigr) + b\text{-}level(n_j)\bigr), \qquad (2)$$

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

In the scheduling process, a task is denoted as being *examined* after it is scheduled to a PE, or as being *unexamined* before it is scheduled to a PE. Unexamined tasks are classified into three sets: (a) a ready set of tasks where the immediate predecessors of any given ready task are all examined (i.e., precedence constraints are satisfied and resource contentions are removed); (b) a partially ready set of tasks where at least one of the immediate predecessors is examined and at least one of its immediate predecessors is unexamined; and (c) an unready set of tasks in which none of the

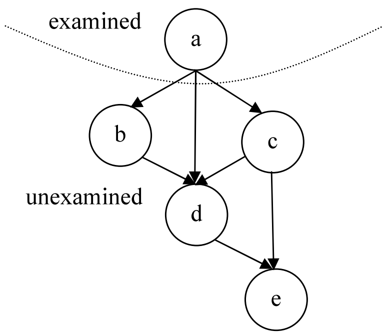

Fig. 1 Example of task classification

immediate predecessors are examined. Figure 1 shows these situations. After n_a is scheduled to a PE, it is examined. The unexamined set includes $\{n_b, n_c, n_d, n_e\}$, in which the set $\{n_b, n_c\}$ is the ready set, the set $\{n_d\}$ is the partially ready set, and the set $\{n_e\}$ is the unready set.

Here, we define the *u-level*(n_i) of an unexamined ready or partially ready node, n_i, as being the longest possible length from the entry nodes to n_i. For example, Fig. 1 shows,

$$\textit{u-level}(n_b) = ect(n_a) + c_{ab} \times \overline{\beta}\big(p(n_a)\big),$$

where $\overline{\beta}(p(n_a))$ is the average communication rate from $p(n_a)$ to other PEs. However, after n_b and n_c are examined, then

$$\begin{aligned}\textit{u-level}(n_d) = \max\big(&ect(n_a) + c_{ad} \times \overline{\beta}\big(p(n_a)\big), ect(n_b) + c_{bd} \times \overline{\beta}\big(p(n_b)\big),\\ &ect(n_c) + c_{cd} \times \overline{\beta}\big(p(n_c)\big)\big).\end{aligned}$$

Formally, $\forall n_j \in pred(n_i) \cap$ examined set, and $\forall n_k \in pred(n_i) \cap$ unexamined set,

$$\begin{aligned}\textit{u-level}(n_i) = \max\big(&ect(n_j) + c_{ji} \times \overline{\beta}\big(p(n_j)\big),\\ &\textit{u-level}(n_k) + \tau_k \times \overline{\alpha}\big(p(n_k)\big) + c_{ki} \times \overline{\beta}\big(p(n_k)\big)\big).\end{aligned} \tag{3}$$

The max–min parallel processing anomaly arises when the completion time obtained after parallelizing is greater than that obtained by sequential execution, and is caused by a trade-off between maximizing parallelism and minimizing the intertask communication. This anomaly has previously been discussed in the literature [26].

To overcome the max–min anomaly and reduce the intermediate schedule length monotonically at each scheduling step, the critical task must be identified at each step so the final schedule length can be strictly reduced. Finding the selection priority that accurately reflects the critical tasks in the context of scheduling is a challenge. However, the critical path may change dynamically as the scheduling progress proceeds (i.e., a node on a critical path at a given step may not be on the critical path at the next step). Therefore, it is not necessary to identify the entire critical path at each scheduling step. We must only consider the tasks on the critical path with immediate predecessors that are also on the critical path and *examined*. Such a task is called a *Dominant Task* (DT).

To identify a dominant task in the DPD algorithm, all immediate successors of the examined tasks are considered, and the task, n_{dt}, with the maximum value of $u\text{-}level(n_{dt}) + \tau_{dt} \times \overline{\alpha}(n_{dt}) + b\text{-}level(n_{dt})$ is identified as being the dominant task. However, a DT is not necessarily a ready task.

In general, most of the proposed algorithms fail to exploit schedule holes [8, 27]. These primarily occur when tasks are scheduled after other tasks with higher scheduling priorities. However, these tasks could be scheduled before the higher-priority tasks without affecting their earliest starting times. Therefore, the DPD algorithm duplicates the dominant tasks of a candidate task into these schedule holes to advance its earliest starting time. Then, the utilization of the PE could be raised, and the earliest starting time of a given task could be advanced, shortening the final schedule length. In this study, we use an idle time slot list, with a limited length m, to keep a list of the last m idle time slots for each PE; and in each scheduling step the DPD algorithm tries to find the schedule holes in the idle time slot list for each PE.

In this study, a *Dominant Predecessor*, n_{dp}, of n_i is an *examined* task, which is scheduled before n_i, with precedence constraints or resource contentions between n_{dp} and n_i. When a given task's dominant predecessor finishes executing, the task's execution can be triggered by satisfying precedence constraints and removing resource contentions. In other words, a task's dominant predecessor restricts the advance of the earliest starting time. The difference between a dominant predecessor in the DPD algorithm and a critical parent [4, 22] in the HCPFD algorithm is that the dominant predecessor takes both precedence constraints and resource contentions into account, whereas the critical parent only considers precedence constraints.

3.2 Dominant predecessor duplication scheduling algorithm

In the DPD algorithm, candidate selection depends on the priority function,

$$priority(n_{mp}) = \max\bigl(b\text{-}level(n_{mp}) - \tau_{mp} \times \overline{\alpha}(n_{mp}) - u\text{-}level(n_{mp})\bigr), \tag{4}$$

where n_{mp} is the task with the maximum priority value. The priority function is derived as follows: (a) when two tasks (e.g., n_a and n_b) are ready and $(b\text{-}level(n_a) - \tau_a \times \overline{\alpha}(n_a)) = (b\text{-}level(n_b) - \tau_b \times \overline{\alpha}(n_b))$, then the task that can be issued earlier (i.e., the one with a smaller *u-level*) should be scheduled first; (b) when two tasks (e.g., n_a and n_b) are ready and $(\tau_a \times \overline{\alpha}(n_a) + u\text{-}level(n_a)) = (\tau_b \times \overline{\alpha}(n_b) + u\text{-}level(n_b))$, then the one having the larger remaining work should be scheduled first; and (c) when two tasks (e.g., n_a and n_b) are ready and $(b\text{-}level(n_a) - u\text{-}level(n_a)) = (b\text{-}level(n_b) - u\text{-}level(n_b))$, then the task that can be scheduled to the PE with the smaller execution rate (i.e., a higher speed) should be scheduled first to shorten the task's execution time. Therefore, we use a composition of these three situations to obtain the priority function.

First, the DPD algorithm initializes the variables and finds the average execution and communication rates for all heterogeneous PEs. Initially, the DPD algorithm assumes that each task is assigned to a single virtual PE, and that the communication overhead between the tasks is the average communication rate multiplied by the communication volume between tasks. Then, the DPD algorithm estimates the *b-levels* for all tasks using the average execution rates of the heterogeneous PEs in a bottom-up fashion. Let the tasks without predecessors be in the ready set. When the ready set is not empty, the DPD algorithm repeats the following steps. First, the DPD algorithm calculates the *u-levels* of unexamined ready and partial ready tasks, and then finds the dominant task, n_{dt}. Second, the DPD algorithm schedules the candidate task to the PE that permits its earliest completion time, taking into consideration that, due to system heterogeneity, the PE that allows the earliest starting time does not necessarily allow the earliest completion time for a task. The DPD algorithm identifies the ready task, n_{mp}, with maximal $priority(n_{mp})$, and its corresponding PE, $p(n_{mp})$, where the $ect(n_{mp})$ could be obtained. Third, the DPD algorithm finds the n_{mp}'s dominant predecessor, n_{dp}, and its data ready time, $drt(n_{dp})$. Fourth, if $n_{dt} \neq n_{mp}$, then the DPD algorithm tries to duplicate n_{dp} to $p(n_{mp})$ without affecting the *u-level* of n_{dt}; otherwise, the DPD algorithm tries to duplicate n_{dp} to $p(n_{mp})$ in order to advance the $est(n_{mp})$. Last, the DPD algorithm allocates n_{mp} to $p(n_{mp})$ and makes n_{mp} *examined*. The repeated steps are listed as follows.

```
    1  Calculate the u-levels of unexamined ready and partial ready tasks.
    2  Find the dominant task, $n_{dt}$, in the ready and partial ready tasks.
    3  Find the ready task, $n_{mp}$, and $p(n_{mp})$, where the $ect(n_{mp})$ could be obtained.
    4  Find the $n_{mp}$'s dominant predecessor, $n_{dp}$, and its data ready time, $drt(n_{dp})$.
    5  If there is a suitable idle time slot in the idle time slot list from $drt(n_{dp})$ to
       $est(n_{mp})$ to duplicate $n_{dp}$ to $p(n_{mp})$ then
  5.1     if $n_{dt} = n_{mp}$ then
5.1.1        try to duplicate $n_{dp}$ to $p(n_{mp})$ to advance the $est(n_{mp})$.
  5.2     Else (if $n_{dt} \neq n_{mp}$)
5.2.1        if duplicating $n_{dp}$ to $p(n_{mp})$ won't affect the u-level of $n_{dt}$ then try to
             duplicate $n_{dp}$ to $p(n_{mp})$ to advance the $est(n_{mp})$.
  5.3     End if
    6  End if
    7  Allocate $n_{mp}$ to $p(n_{mp})$, and make $n_{mp}$ examined.
```

The sample DAG is shown in Fig. 2. To simplify the analysis, we assume that the computation cost (i.e., $\tau_i \times \alpha(p(n_i))$) of all the tasks in all the PEs can be estimated as shown in Table 2 (e.g., the computation costs of n_1 are 5, 3 and 4 in PE1, PE2 and PE3, respectively), and that the communication rates of all the communication channels are all unity (i.e., $\overline{\beta} = 1$ for all paired inter-PE communications); therefore, the numbers listed along the edges are the communication volume; these can also be considered as being the communication costs when the parent task and its child task are scheduled in different PEs. For example, when n_1 and n_2 are scheduled in different PEs, the communication cost between them has a value of three.

Next, we calculate the average computation costs and *b-levels* for all the tasks, as listed in Table 3. When the set of unexamined tasks is not empty, the DPD algorithm selects an unexamined ready task, n_{mp}, to be scheduled to its corresponding PE. For example, in the fourth step shown in Table 4, there are four ready nodes: n_3, n_4, n_6 and n_8. Nodes n_3 and n_4 have the maximum priority, and the DPD algorithm randomly selects n_3 as n_{mp}, and allocates it to PE1 to obtain the minimum earliest completion time. Because n_2 is n_{dp} in this step, n_2 can be duplicated to PE1 to

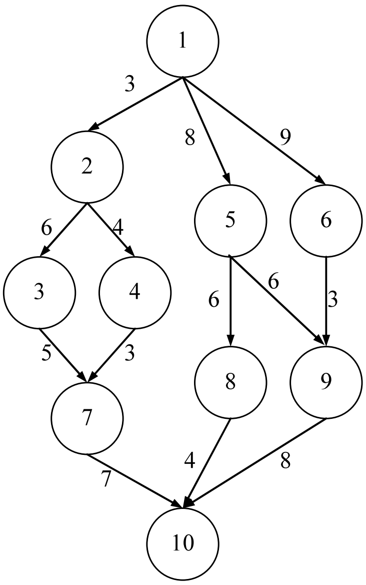
Fig. 2 Sample DAG

Table 2 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	3	2
8	2	3	4
9	4	5	3
10	6	4	5

decrease the value of $ect(n_3)$ from $ect(n_3) = 13$ to $ect(n_3) = 11$. The full list of steps is presented in Table 4. For comparison, the scheduling results obtained from the TANH, HCPFD and DPD algorithms are shown in Fig. 3.

The time complexities of the HCPFD, CNPT and TANH scheduling algorithms are O(|P||N|^2), O(|N|^2) and O(|N|^2), respectively. In the DPD algorithm, the time complexity for calculating the *b-level* is O((|N| + |E|)|P|). Steps 1 to 7 are executed in O(|N|) steps, and the time complexity in calculating the *u-levels* of unexamined ready and partial ready tasks is O(|N|), in finding the dominant task, n_{dt}, is O(|N|), in finding n_{mp} is O(|N|), and in finding n_{mp}'s corresponding PE is O(|P|). The time complexity of finding n_{dp} requires O(|N|) steps. Checking the idle time slot list to find the schedule hole requires O($|m|$) steps, and testing whether duplicating n_{dp} to

Table 3 Pre-scheduling information

Task n_i	Average computation costs	$b\text{-}level(n_i)$
1	4	38
2	4	31
3	5	20
4	5	18
5	7	23
6	5	20
7	3	12
8	3	9
9	4	13
10	5	0

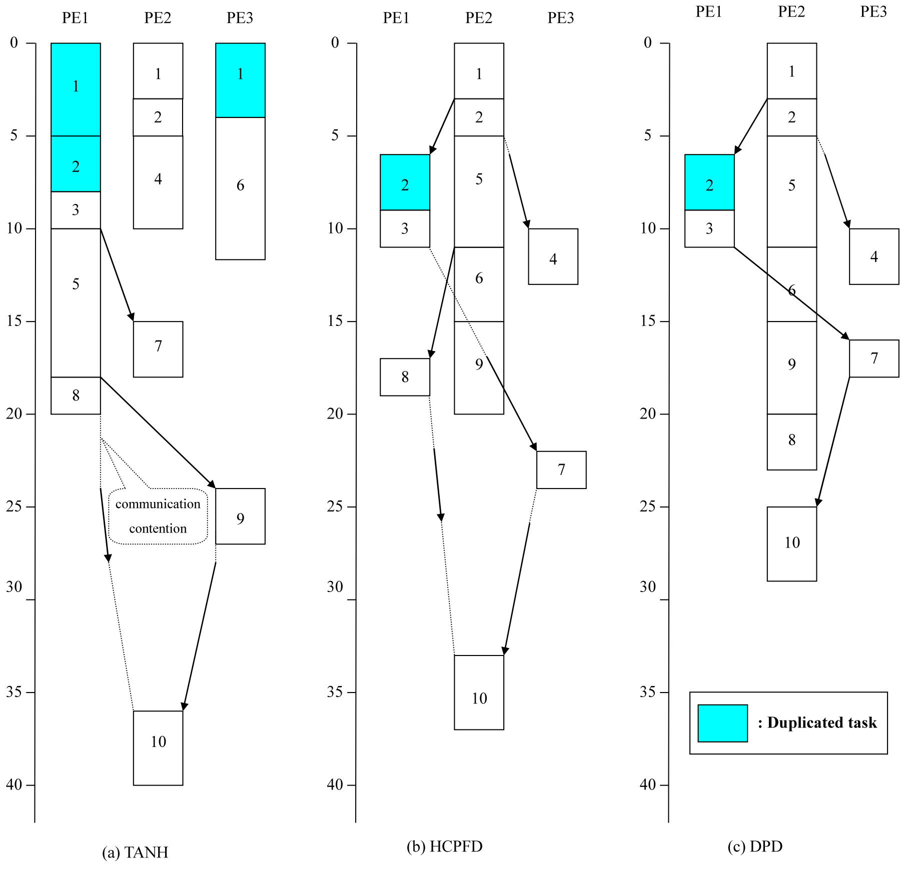

Fig. 3 Scheduling results for the sample DAG given in Fig. 2

Table 4 DPD scheduling steps

Steps	Candidate tasks	*u-level*(n_i)	Priority (n_i)	n_{mp}	$p(n_{mp})$	$ect(n_{mp})$	n_{dp}	$drt(n_{dp})$	n_{dt}	Duplicated task
1	1	0	34	1	2	3	–	–	1	–
2	2	6	21	2	2	5	1	0	2 or 5	–
	5	11	5							
	6	12	3							
3	3	11	4	5	2	11	2	3	5	–
	4	9	4							
	5	11	5							
	6	12	3							
4	3	11	4	3	1	13 $\downarrow$ 11	2	3	6 or 9	2
	4	9	4							
	6	12	3							
	8	17	-11							
	9	20	not ready							
5	4	9	4	4	3	12	2	3	6 or 9	–
	6	12	3							
	7	17	not ready							
	8	17	-11							
	9	20	not ready							
6	6	12	3	6	2	15	5	5	6 or 9	–
	7	16	-7							
	8	17	-11							
	9	20	not ready							
7	7	16	-7	7	3	18	3	9	9	–
	8	17	-11							
	9	18	-9							
8	8	17	-11	9	2	20	6	11	9 or 10	–
	9	18	-9							
	10	30	not ready							
9	8	17	-11	8	2	23	9	15	10	–
	10	28	not ready							
10	10	28	-33	10	2	29	7	16	10	–

$p(n_{mp})$ would affect the *u-level* of n_{dt} requires O(|N|) steps. Therefore, the time complexity of the DPD algorithm is $O((|N| + |E|)|P| + |N|(|N| + |P| + |m||N|)) = O(|P||E| + |m||N|^2)$. Consequently, in practical applications, the complexity of the DPD algorithm is reasonable with an additional space complexity $O(|m||P|)$.

4 Experimental results

The feasibility of the DPD algorithm is assessed by evaluating several practical applications (e.g., the Gauss–Jordan Elimination, Fast Fourier Transformation (FFT), LU factoring, Fork trees, Join trees, and randomly generated program graphs whose graph sizes vary from 378 to 511 nodes). The computation/communication volumes required for each task are predetermined for each of the different applications.

4.1 Simulation environment

In our simulation environment, the number of PEs is set to 1, 2, 4, 8, 16 or 32. The communication to computation ratio (CCR) is the ratio of the average communication rate to the average computation rate. In this study, the CCR value is set to be 0.1, 0.3, 0.5, 0.7, 0.9, 1.0, 2.0, 4.0, 6.0, 8.0 or 10.0. It is well known that system heterogeneity may affect the quality of the schedule. The variance in the execution rates, $\alpha(p_i)$, of the heterogeneous PEs is defined as the heterogeneity of computational resources. We assume that the distribution of execution rates is a normal distribution, with a mean value of 10 and a variance of 0.2, 0.4, 0.6, 0.8, 1, 2, 4 or 8. As the variance increases, the heterogeneity of computational resources becomes more obvious. The variance of the transfer rate, $\beta(p_i, p_j)$, for the communication channels is defined as the heterogeneity of the communication mechanisms. We assume that transfer rates also follow a normal distribution, with a mean value obtained from the CCR multiplied by ten, and a variance of 0.2, 0.4, 0.6, 0.8, 1, 2, 4 or 8. As the variance increases, the system heterogeneity also becomes more obvious.

All the simulation programs are coded by C and run on IBM Xserver 206 with Intel Pentium IV 3.0 GHz, 1 G DDRII RAM and two SCSI 36 G Harddisks. The simulation environment is built on the Linux operating system, Fedora Core 4 version.

4.2 Related works

Here we apply three algorithms to demonstrate the comparative effectiveness of our own. The first is a list scheduling algorithm, CNPT [13], which schedules tasks according to their priority, and is a task scheduling heuristic that supports heterogeneous processors.

The second is the TANH algorithm [2], which is based on the clustering and duplication-based approaches. This algorithm generates initial clusters by arranging nodes in an ascending order of levels, then either performs a duplication procedure when the number of available PEs is larger than the number of clusters, or carries out a processor reduction procedure when the number of available PEs is smaller than the number of clusters. Message scheduling is achieved in the final step. Because the

TANH algorithm directly merges tasks in different clusters by their levels to determine their relative positions in the processor reduction procedure, it fails to consider the situation where two tasks initially in the same generated cluster may not stay in the same PE to obtain a better schedule later.

The third, the HCPFD algorithm [4, 20], is based on the list and duplication-based approaches for a bounded number of fully connected heterogeneous PEs. This algorithm consists of two phases: the listing phase, where the algorithm divides the DAG into a set of unlisted parent-trees to obtain the sequence of the task assignment, and the machine assignment phase, where the HCPFD algorithm assigns the candidate task to the PE that minimizes its completion time, and then tries to duplicate its critical parent to the idle time slot between the critical parent's starting time and the last task assigned in the same PE.

The DPD algorithm differs from the HCPFD algorithm in the following three ways. First, because the HCPFD algorithm determines the scheduling sequence in the listing phase, it does not consider the case where the critical path may dynamically change in the scheduling process. Second, instead of duplicating the candidate's critical parent at the idle time slot between the critical parent's starting time and the last task assigned in the same PE, the DPD algorithm duplicates the candidate's dominant predecessor to the idle time slot from its data ready time to the starting time of the candidate node. The DPD algorithm can increase the possibility of finding an idle time slot for these duplicated dominant predecessors. Third, the DPD algorithm initially assigns each task to a virtual PE, and then tries to reduce the time complexity by avoiding reevaluating the ready-task-PE pairs in each scheduling step.

4.3 Performance analyses

Comparatively, the DPD algorithm outperforms the CNPT, TANH and HCPFD algorithms in schedule lengths for practical applications, as shown in Table 5. These preliminary experimental results show the superiority of the DPD algorithm. The schedule lengths obtained by the DPD algorithm are shorter than 81.92% of the studied cases by the HCPFD algorithm, 87.65% of the studied cases by the TANH algorithm, and 95.73% of the studied cases by the CNPT algorithm.

The TANH algorithm has a poor performance in the data listed in Table 5 because it adopts the clustering and duplication approaches. When the number of PEs is not large enough, the TANH algorithm produces virtual PEs to bring forth scheduling, and then merges these virtual PEs to fit in with the actual PEs using the level sorting approach. For this reason, the TANH algorithm could not achieve an optimum schedule.

As the performance of the CNPT algorithm was shown to be worse than that of the HCPFD algorithm in the literature [4, 20], we omit the performance analysis of

Table 5 Comparison in terms of scheduling lengths

	CNPT	TANH	HCPFD	
DPD	better than	95.73%	87.65%	81.92%
	equal to	0.04%	1.04%	1.86%
	worse than	4.23%	11.31%	16.22%

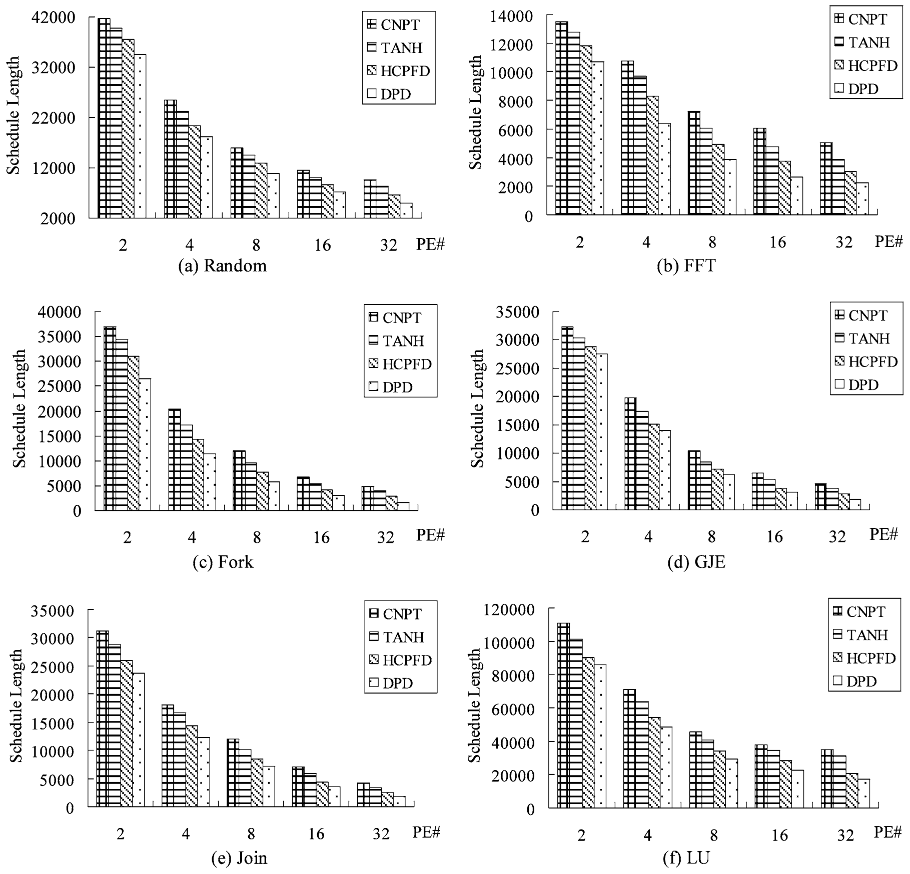

Fig. 4 Scheduling lengths by varying the number of PEs

the CNPT algorithm, and focus on a comparison of the TANH, HCPFD and DPD algorithms. Figure 4 shows the average scheduling lengths for different applications by varying the number of PEs. As shown in Fig. 4a, the schedule length obtained by the DPD algorithm is shorter than that obtained by the other three. This indicates that because the DPD algorithm dynamically decides the dominant task in each scheduling step, its scheduling performance is better than that of the others. The HCPFD algorithm decides the scheduling sequence in the listing phase, and lacks enough information to reduce the schedule length at each scheduling step. The same experimental results are also shown in Figs. 4b–f. In general, the DPD algorithm monotonically reduces the schedule length as the number of PEs increases. This shows that the strict reduction condition can indeed avoid the max–min anomaly.

Our experimental results also indicate that the DPD algorithm performs better than the CNPT, TANH or HCPFD algorithms in terms of handling the heterogeneity of computational resources and communication mechanisms. Figure 5 shows the average scheduling lengths for different applications over a range of variances of computing rates, α, where the variance of α denotes the heterogeneity of the PEs. As

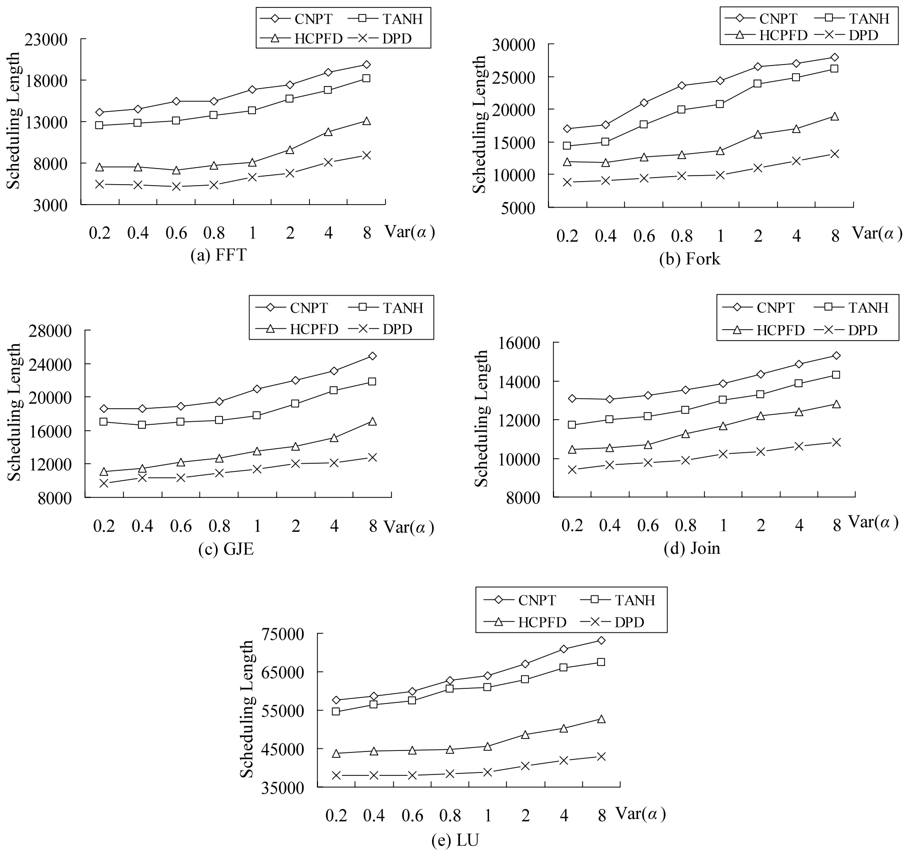

Fig. 5 Scheduling lengths by varying the variance of computing rates

the variance of α increases, so do the scheduling lengths obtained by the four algorithms. These results clearly indicate that computation heterogeneity affects scheduling performance. The data in Fig. 5 also show that as the variance of α increases, the variations of the scheduling lengths obtained by the DPD algorithm are less than those obtained by other algorithms.

Figure 6 shows the average scheduling lengths for different applications over a range of variances of communication rates, β, where the variance of β denotes the heterogeneity of the communication mechanism. As the variance of β increases, the scheduling lengths obtained by the four algorithms also increase. Just as with computation, our experimental results indicate that communication heterogeneity also affects scheduling performance. In addition, the data in Fig. 6 shows that, as the variance of β increases, the variations of the scheduling lengths obtained by the DPD algorithm are less than those obtained by the other algorithms. Figures 5 and 6 show that the system heterogeneity affects the schedule length, and that it is possible to generate a parallelized schedule length worse than that obtained by scheduling sequentially. This arises from a scheduling anomaly, in which the completion time of a DAG on a parallel processing system is worse than that on a single PE. This anom-

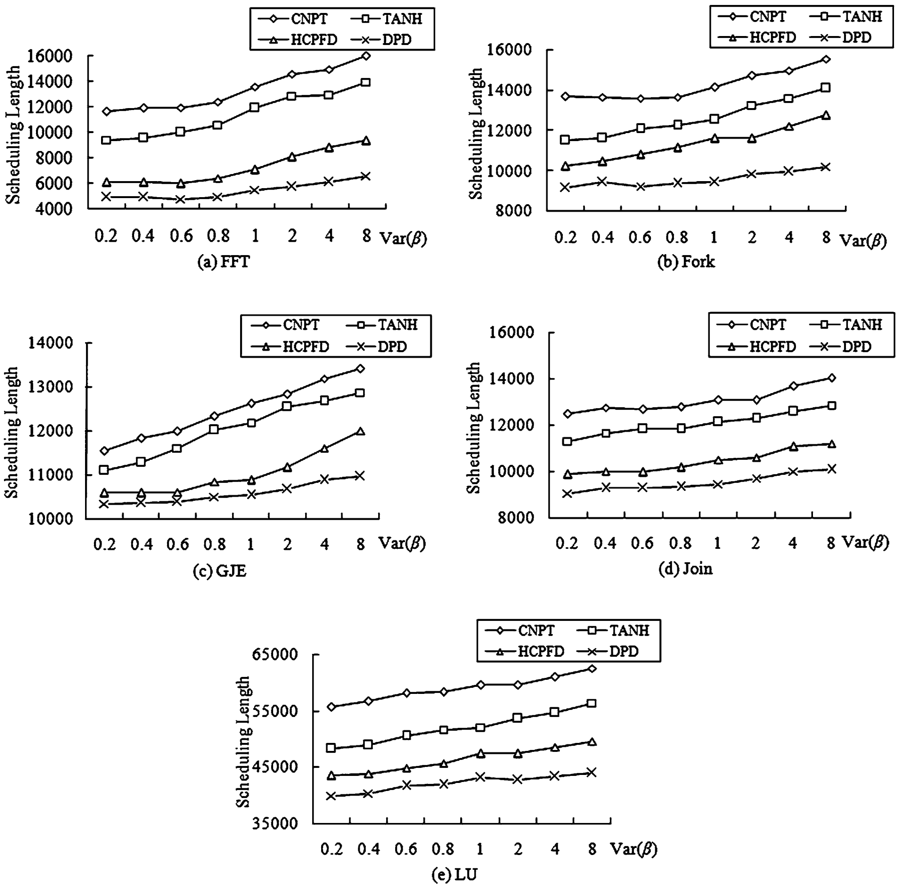

Fig. 6 Scheduling lengths by varying the variance of communication rates

aly results from the max–min problem in parallel processing, and is caused by the trade-off between maximizing the parallelism and minimizing the intertask communication.

We define utilization as $\sum \tau_i \cdot \alpha(p(n_i))/(|\mathrm{P}| \cdot \max(ect(n_i)))$, where n_i includes the duplicated tasks. In Fig. 7, the average system utilization obtained by the CNPT, TANH, HCPFD and DPD algorithms reduces monotonically as the number of PEs increases. The system utilization reasonably reduces as the number of PEs increases under the same computing volume conditions. Our experimental results show that the system utilization achieved by the DPD algorithm outperforms the other three algorithms. The main reason for this is that the DPD algorithm dynamically duplicates the candidate's dominant predecessor to the idle time slot from the dominant predecessor's data ready time to the starting time of the candidate node. Therefore, the DPD algorithm can exploit more resource utilization.

As shown in Fig. 8, the average utilization increases as the communication to computation cost ratio (CCR) increases. Because the DHC system allows overlapping of

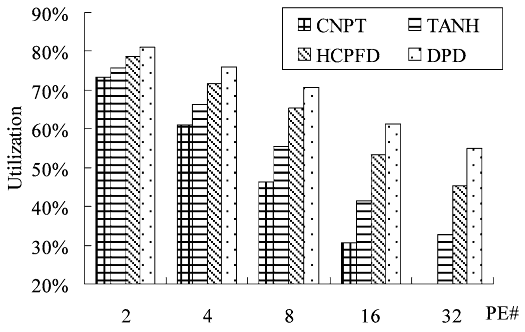

Fig. 7 Utilization by varying the number of PEs

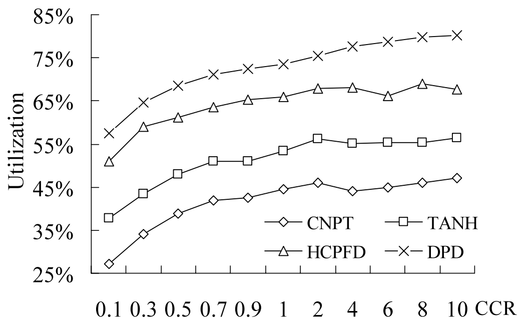

Fig. 8 Utilization by varying CCRs

computations and communications that are independent of each other, the utilization is promoted by duplicating tasks for CCR ≥ 1.

5 Conclusions

Recent advances in high-speed networks make DHC appealing in terms of its performance and cost-effectiveness. We have presented a scheduling algorithm, named DPD, for heterogeneous computing environments, which has a higher scalability and lower redundant resource consumption in DHC systems. This algorithm could avoid the max–min anomaly and exploit schedule holes. Our experimental results show that scheduling performance is affected by the heterogeneity of computational resources and communication mechanisms, and by the program structure of applications. The performance of our DPD algorithm is demonstrated by evaluating practical application benchmarks, and our results also show the superiority of our proposed algorithm to those discussed in the literature. In more than 80% of the studied cases, the DPD algorithm out performs other compared algorithms in terms of the scheduling length; and, as the system heterogeneity increases, the variations of the scheduling lengths obtained by the DPD algorithm are less than those obtained by other algorithms. The time complexity of the DPD algorithm is $\mathrm{O}(|\mathrm{P}||\mathrm{E}| + |m||\mathrm{N}|^2)$ with an additional space complexity $\mathrm{O}(|m||\mathrm{P}|)$, which is reasonable in practical applications. Therefore, our proposed scheduling algorithm can be adopted and can work efficiently in designing scheduling strategies for those situations where system heterogeneities cause performance bottlenecks. [28, 29]

Acknowledgement This work was sponsored in part by the National Science Council, Taiwan, Republic of China under the contract number: NSC 95-2221-E-142-004-.

References

1. Topcuoglu H, Hariri S, Wu M-Y (2002) Performance-effective and low-complexity task scheduling for heterogeneous computing. IEEE Trans Parallel Distrib Syst 13(3):260–274
2. Bajaj R, Agrawal DP (2004) Improving scheduling of tasks in a heterogeneous environment. IEEE Trans Parallel Distrib Syst 15(2):107–118
3. Braun T, Siegel HJ, Beck N, Boloni LL, Maheswaran M, Reuther AI et al. (1999) A comparison study of static mapping heuristics for a class of meta-tasks on heterogeneous computing systems. In: Proceedings of heterogeneous computing workshop, pp 15–29
4. Hagras T, Janecek J (2005) A high performance, low complexity algorithm for compile-time task scheduling in heterogeneous systems. Parallel Comput 31:653–670
5. Kwok Y-K, Ahmad I (2000) Link contention-constrained scheduling and mapping of tasks and messages to a network of heterogeneous processors. Clust Comput J Netw Softw Tools Appl 3(2):113–124
6. Sinnen O, Sousa L (2004) List scheduling: extension for contention awareness and evaluation of node priorities for heterogeneous cluster architectures. Parallel Comput 30(1):81–101
7. Sinnen O, Sousa LA (2005) Communication contention in task scheduling. IEEE Trans Parallel Distrib Syst 16(6):503–515
8. Selvakumar S, Siva Ram Murthy C (1994) Scheduling precedence constrained task graphs with non-negligible intertask communication onto multiprocessors. IEEE Trans Parallel Distrib Syst 5(4):328–336
9. Kwok Y-K, Ahmad I (1999) Static scheduling algorithms for allocating directed task graphs to multiprocessors. ACM Comput Surv 31(4):406–471
10. Ullman JD (1975) NP-complete scheduling problems. J Comput Syst Sci 10:384–393
11. Garey MR, Johnson DS (1979) Computers and intractability: a guide to the theory of NP-completeness. Freeman, New York
12. Olivier B, Vincent B, Yves R (2002) The iso-level scheduling heuristic for heterogeneous processors. In: Proceedings of 10th Euromicro workshop on parallel, distributed and network-based processing, pp 335–350
13. Hagras T, Janecek J (2003) A high performance, low complexity algorithm for compile-time job scheduling in homogeneous computing environments. In: IEEE 2003 international conference on parallel processing workshops (ICPPW'03), October 2003, pp 149–155
14. Sih GC, Lee EA (1993) A compiler-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures. IEEE Trans Parallel Distrib Syst 4(2):175–186
15. Gerasoulis A, Yang T (1992) A comparison of clustering heuristics for scheduling directed acyclic graphs onto multiprocessors. J Parallel Distrib Comput 16(4):276–291
16. Palis MA, Liou J-C, Wei DSL (1996) Task clustering and scheduling for distributed memory parallel architectures. IEEE Trans Parallel Distrib Syst 7(1):46–55
17. Pande SS, Agrawal DP, Mauney J (1994) A new threshold scheduling strategy for sisal programs on distributed memory systems. J Parallel Distrib Comput 21(4):223–236
18. Pande SS, Agrawal DP, Mauney J (1995) A scalable scheduling method for functional parallelism on distributed memory multiprocessors. IEEE Trans Parallel Distrib Syst 6(4):388–399
19. Ahmad I, Kwok Y-K (1998) On exploiting task duplication in parallel program scheduling. IEEE Trans Parallel Distrib Syst 9(8):872–892
20. Hagras T, Janecek J (2004) A high performance, low complexity algorithm for compile-time task scheduling in heterogeneous systems. In: IEEE proceedings of 18th international parallel and distributed processing symposium, April 2004, pp 107–115
21. Chung YC, Ranka S (1992) Application and performance analysis of a compile-time optimization approach for list scheduling algorithms on distributed-memory multiprocessors. In: Proceedings of supercomputing, pp 512–521
22. Hagras T, Janecek J (2004) A near lower-bound complexity algorithm for compile-time task-scheduling in heterogeneous computing systems. In: IEEE proceedings of third international symposium on parallel and distributed computing/third international workshop on algorithms, models and tools for parallel computing on heterogeneous networks, pp 106–113

23. Danalis A, Kim K-Y, Pollock L, Swany M (2005) Transformations to parallel codes for communication-computation overlap. In: Proceedings of the ACM/IEEE supercomputing, 12–18 November 2005, pp 58–70
24. Beaumont O, Boudet V, Robert Y (2002) A realistic model and an efficient heuristic for scheduling with heterogeneous processors. In: Proceedings of 11th heterogeneous computing workshop
25. Gerasoulis A, Yang T (1993) On the granularity and clustering of directed acyclic task graphs. IEEE Trans Parallel Distrib Syst 4(6):686–701
26. Kruatrachue B, Lewis T (1988) Grain size determination for parallel processing. IEEE Softw 5(1):23–32
27. Lai GJ, Fang JF, Sung PS, Pean DL (2003) Scheduling parallel tasks onto NUMA multiprocessors with inter-processor communication overhead. In: Proceedings of international symposium on parallel and distributed processing and applications, pp 65–75
28. Bansal S, Kumar P, Singh K (2003) An improved duplication strategy for scheduling precedence constrained graphs in multiprocessor systems. IEEE Trans Parallel Distrib Syst 14(6):533–544
29. Sarkar V (1989) Partitioning and scheduling parallel programs for execution on multiprocessors. MIT Press, Cambridge

Kuan-Chou Lai received his MS degree in computer science and information engineering from the National Cheng Kung University in 1991, and the PhD degree in computer science and information engineering from the National Chiao Tung University in 1996. Currently, he is an assistant professor in the Department of Computer and Information Science at the National Taichung University. His research interests include parallel processing, heterogeneous computing, system architecture, grid computing, and multimedia systems. He is a member of the IEEE and the IEEE Computer Society.

Chao-Tung Yang received a BS degree in computer science and information engineering from Tunghai University, Taichung, Taiwan in 1990, and the MS degree in computer and information science from National Chiao Tung University, Hsinchu, Taiwan in 1992. He received the PhD degree in computer and information science from National Chiao Tung University in July 1996. He won the 1996 Acer Dragon Award for outstanding PhD Dissertation. He has worked as an associate researcher for ground operations in the ROCSAT Ground System Section (RGS) of the National Space Program Office (NSPO) in Hsinchu Science-based Industrial Park since 1996. In August 2001, he joined the faculty of the Department of Computer Science and Information Engineering at Tunghai University, where he is currently an associate professor. His researches have been sponsored by Taiwan agencies National Science Council (NSC), National Center for High Performance Computing (NCHC), and Ministry of Education. His present research interests are in grid and cluster computing, parallel and high-performance computing, and internet-based applications. He is both member of the IEEE Computer Society and ACM.

Copyright of Journal of Supercomputing is the property of Springer Science & Business Media B.V. and its content may not be copied or emailed to multiple sites or posted to a listserv without the copyright holder's express written permission. However, users may print, download, or email articles for individual use.