A Study of Enhancing Computing Performance on Heterogeneous Environments

Pei-Shan Sung
National Taichung Teachers College,
Taichung, Taiwan, R.O.C
song13@cm1.hinet.net

Guan-Joe Lai
National Taichung Teachers College,
Taichung, Taiwan, R.O.C
gjlai@mail.ntctc.edu.tw

Abstract

The heterogeneous computing system could exploit the computational powers of the tasks in an application. It is the key to parallelize the tasks to several processors to reduce the total execution time. Since HC environments could meet the requirement of exploiting the computational powers, so the HC environment is studied in this paper. As a result, in this study, we consider exploiting a competent list-based heuristic algorithm, which is called the Dominant Tasks Scheduling (DTS) algorithm, for scheduling the tasks of a parallel application into HC environments.

In the systems with the high communication heterogeneity or the high computation heterogeneity, the DTS algorithm, could perform better than other proposed algorithms from the literature by considering global scheduling information and by exploiting schedule-holes. The experimental results show the superiority of the DTS algorithm.

Keywords: heterogeneous, scheduling, algorithm, parallel, network

1 Introduction

The heterogeneous computing (HC) environment consists of a distributed set of different types of personal computers or workstations (i.e., processor elements, PEs) with diverse computing resources, which are connected by high-speed transmission medias. With small extra cost, the HC environment could be constructed with the network of workstations (NOWs). HC environments could offer more powerful and commercial high performance computing systems by gathering a cluster of NOWs. An application with computationally intensive tasks could be parallel executed in the HC system. Since HC environments could meet the requirement of exploiting the computational powers, so the HC environment is studied in this paper. As a result, in this study, we consider exploiting a competent list-based heuristic algorithm, which is called the Dominant Tasks Scheduling (DTS) algorithm, for scheduling the tasks of a parallel application into HC environments.

In the systems with the high communication heterogeneity or the high computation heterogeneity, the DTS algorithm, could perform better than other proposed algorithms from the literature by considering global scheduling information and exploiting schedule-holes. The experimental results show the superiority of the DTS algorithm.

2 Related Works

In this section, three previous list-scheduling heuristic algorithms and their characteristics are described. They are the Dynamic Level Scheduling (DLS) algorithm [16], the Heterogeneous-Earliest-Finish-Time (HEFT) algorithm [17], and the Critical-Path-on-a-Processor (CPOP) [17].

2.1 The Dynamic Level Scheduling (DLS) algorithm

The main process of the Dynamic Level Scheduling (DLS) algorithm is to determine the dynamic level (DL) [16]. The DL is computed by using the static level (SL) and the starting time (ST). The $DL(n_i,J)$ is defined as $SL(n_i) - ST(n_i,J)$ of the node-processor pair (n_i,J). At each scheduling step, the DLS algorithm computes the DL value of each ready node on every processor element. The node with the largest DL value would be selected and scheduled to the corresponding processor element.

Although the DLS algorithm performs exhaustive pair matching of nodes to processor elements at each scheduling step to find the highest priority node, it doesn’t assign priority based on the CP. The DLS algorithm has some problems of the node-selection priority. It selects a node with a higher static level and a smaller starting time to be scheduled. But, sometimes, the static level of the selected node might not the highest and its starting time might not the earliest among all the ready nodes. With a large SL, a node might be scheduled first even though its starting time is not small. This could block the more important nodes to be scheduled earlier.

2.2 The Heterogeneous-Earliest-Finish-Time (HEFT) algorithm

The main process of the Heterogeneous-Earliest-Finish-Time (HEFT) algorithm is to determine the upward rank ($rank_u$) [17]. Simply, $rank_u(n_i)$ is the length of the CP from node n_i to the exit nodes, including the computation cost of node n_i. The HEFT algorithm sorts the tasks by decreasing order of $rank_u$ and then constructs a scheduling list by this order. Each task is scheduled by the order of the scheduling list onto a suitable processor element that allows the minimum earliest finish time with using the insertion-based scheduling policy. The insertion-based policy is some kind of the scheduling policies that considers the possible insertion of a task in an earliest idle time slot between two scheduled tasks on a processor element.

Although the HEFT algorithm takes care of the time-slot problem with the insertion-based scheduling policy, it still has some flaws. In fact, the data transmission through the interconnect mechanism in the HC environment would cause the communication contention. The HEFT algorithm doesn't consider the communication-contention problem. In this study, the communication-contention problem is considered.

2.3 The Critical-Path-on-a-Processor (CPOP) Algorithm

The critical-path-on-a-processor (CPOP) algorithm is similar to the HEFT algorithm. Its main process is to determine the upward rank ($rank_u$) and the downward rank ($rank_d$) [17]. The difference between the HEFT algorithm and the CPOP algorithm is that the CPOP defines the CP nodes first and schedules them onto CP processor. However, the CPOP algorithm also uses the insertion-based scheduling policy to find the EFT time of a candidate node.

Like the HEFT algorithm, the CPOP algorithm also takes care of the time-slot problem with the insertion-based scheduling policy, it still has some flaws. The CPOP algorithm doesn't take of the communication contention for data transmission through the interconnect mechanism in the HC environment.

3 The Proposed Algorithm

In this section, the proposed algorithm, the Dominant Tasks Scheduling (DTS) heuristic algorithm would be introduced.

3.1 Definitions

In this study, a heterogeneous system model is presented by M=(P, Q, A, B), where $P=\{p_i|p_i \in P, i=1,\ldots,|P|\}$ is the set of heterogeneous processor elements, $Q=\{q_{ij}|q_{ij} \in Q, i,j=1,\ldots,|P|\}$ is the set of communication channels, $A=\{\alpha_i|\alpha_i \in A, i=1,\ldots,|P|\}$ is the execution rate of processor element p_i, and $B=\{\beta_{ij}|\beta_{ij} \in B, i,j, =1,\ldots,|P|\}$ is the communication rate from processor element pi to processor element p_j. The communication channel q_{ij} means the channel from processor element p_i to processor element p_j. This study assumes that q_{ij} and q_{ji} are the same channel, and each processor element in the system has dedicated hardware to deal with communications so that communication and computation could take place simultaneously.

The computation cost of node n_i when it is allocated to processor element p_k is denoted to be $w(n_i)*\alpha_k$. The communication cost from node n_i to node n_j, where n_i is allocated to processor element p_k and task n_j is allocated to processor element p_l, is denoted to be $c_{ij}*\beta_{kl}$. Besides, let $pred(n_i)$ be the set of predecessor nodes of n_i, and $succ(n_i)$ be the set of immediate successor nodes of n_i. The $est(n_i)$ value is the earliest starting time of node n_i which satisfies the precedence constraints and considers the communication contentions. The $ect(n_i)$ value is the earliest completion time of node n_i and it is defined as follows: $est(n_i)=est(n_i)+w(n_i)*\alpha_k$. The $lct(n_i)$ value is the longest execution time from node n_i to the exit node and it is defined as follows:

$$lct(n_i)=\max\{c_{ij}*\beta_{kl}+w(n_i)*\alpha_k+lct(n_j)\},$$

where $n_j \in succ(n_i)$.

As described above, a DAG and a system model are given. The object of the proposed scheduling algorithm is to gain the minimum completion time of the task graph by the algorithm in NOWs systems.

In the DTS algorithm, only the ready nodes are qualified selected. A node is called the ready node only if it has no predecessor node or all of its predecessor nodes are scheduled already. The DTS algorithm would choose two nodes for selecting the candidate node, one is DT node, and the other one is maximum priority node.

The node n_i is defined as a DT node with $\max\{est(n_i)+w(n_i)*\alpha_k+lct(n_i)\}$, where k=1,…,|P|. Tie-breaking of the DT node selection is done by FIFO manner. The DT nodes might change dynamically at each step. Hence, the DT nodes would be identified by dynamic priority in this algorithm. Also, the algorithm uses dynamic priority to determine the maximum priority node that is for node-selection at each step. In this study, we define node-selection priority as follows:

$$Priority(n_i)=lct(n_i)-w(n_i)*\alpha_k-est(n_i)$$

Tie-breaking of the node-selection is also by FIFO manner. A node could obtain the highest priority value if its starting time and its computation cost are minimum, and its lct value is maximum. A node has the larger lct value means that it carries the loads of successor nodes. It would take a lot time to execute all of these successor nodes. With the node-selection priority, the maximum priority node would be selected. The communication cost between two nodes is zero when the nodes are scheduled to the same processor element.

To ensure that the DTS algorithm would select the most important partial node to be scheduled, the node-selection condition is defined as follows:

Condition A: Assume n_a is the DT node and n_b is the maximum priority node at schedule step i. Also assume $P(n_a)$ is the processor element that node n_a is allocated to obtain the highest DT priority value, and $P(n_b)$ is the processor element that node n_b is allocated to obtain the highest node-selection priority value. The candidate node is finally selected to scheduled at each step by the condition A, which is described in Fig. 1.

With the condition A, the reduction of schedule length is ensured. If scheduling maximum priority node would affect the DT length, the candidate node would be the DT node, not the maximum priority node.

```
If (na = = nb)
        candidate node = na ;
Else
        If min ect(nb) is on P(na)
              IF est(nb) > est(na)
                 candidate node= na;
              Else
                 candidate node= nb;
              EndIf
        Else
              candidate node = nb;
        EndIf
EndIf
```

Fig. 1. Condition A.

After a candidate node is selected, a pair of a candidate node and its corresponding processor element is considered.

The schedule-holes are the unoccupied time slices of processor elements to accommodate unscheduled nodes. Exploiting schedule-holes is a manner that schedules the low priority nodes before the higher priority nodes without affecting the earliest starting times of these higher priority nodes. In order to exploit the schedule-holes, the condition B ensures that the schedule length of a task graph would be strictly reduced.

Condition B: Assume that n_a and n_b are both ready nodes and n_a is candidate node at step i. Also, assume that $P\_Time(P(n_a))$ is the time for $P(n_a)$ ready to execute tasks for all of the ready tasks after step i-1. The node n_b could be scheduled on $P(n_a)$ before n_a at step i if n_b could satisfy the following conditions:

1) $est(n_b) \geq P\_Time(P(n_a))$ as $P(n_a) = P(n_b)$
2) $ext(n_b) \leq est(n_a)$

Tie-breaking of selecting the node n_b selection is by the minimum est. In this manner, the algorithm could exploit the schedule-holes.

3.2 The Dominant Tasks Scheduling (DTS) algorithm

In this section, we present the dominant tasks scheduling (DTS) heuristic algorithm as shown in Fig. 2.

In the line 1 in Fig. 2, the DTS algorithm computes the lct value for each node. The line 4 in Fig. 2 defines the DT node n_{DT_node} by the maximum partial DT_length. The equation of the maximum partial DT_length is described below:

$\max\{est(n_i)+w(n_i)*\alpha_k+lct(n_i)\}$, where k=1,…,|P|.

The line 5 in Fig. 2 defines the node $n_{max\_priority}$ with the highest priority. The priority for each node at each scheduling step is described below:

$$Priority(n_i)=lct(n_i)-w(n_i)*\alpha_k-est(n_i)$$

The line 6 in Fig. 2 would select the candidate node n_i to be scheduled by the condition A; as shown in Fig. 1.

The line 7 in Fig. 2 would find the suitable processor element p_k that provided the minimum $est(n_i)$ value. With the condition B, the line 8 in Fig. 2 selects the second candidate node n_j from all ready nodes except node n_i to be scheduled on the processor element p_k. After scheduling n_i and n_j (if n_j could be found in the line 8 in Fig. 2), the DTS algorithm would update the list of ready nodes. The scheduling would continue the steps from the line 3 to line 11 in Fig. 2 till all nodes are scheduled. The while loop takes O(n) operations, and the finding

n_{DT_node} step takes O(n(n+e)p) operations, where n is the number of tasks, the e is the number of edges, and the p is the number of processor elements. The finding $n_{max\_priority}$ step also takes O(n(n+e)p). The step of selecting and assigning the candidate node takes O(p) operations. Also, the step of selecting and scheduling the second candidate node takes O(p) operations, too. To update the ready node list takes O(ne) operations. Therefore, the time complexity of the DTS algorithm is $O(n^2(n+e)p)$. The time complexity of the DTS is acceptable for using in compiler time.

Algorithm DTS
Input: a system M=(P,Q,A,B); a DAG=(N,E,W,C).
Output: A schedule with minimal parallel completion time for the heterogeneous NOWs.
Begin

1. Compute lct value for each node
2. Make Ready_node_list
3. While Ready_node_list not empty
4. Find $n_{DT\_node} \in$ ready nodes, which has maximum partial DT_length.
5. Find $n_{\max\_priority} \in$ ready nodes, which has max priority.
6. Select candidate node n_i from n_{DT_node} and $n_{max\_priority}$ with the condition A.
7. Assign n_i to processor p_k that provides min. est(n_i).
8. Select second candidate node n_j from ready nodes expect n_i if satisfy condition B and breaks tie with minimum est
9. Assign n_j to processor p_k.
10. Update Ready_node_list.
11. endWhile

End

Fig. 2. The DTS algorithm.

4 Experimental Results

In this study, four proposed algorithms are experimented. They are the DTS algorithm, the HEFT algorithm, the CPOP algorithm, and the DLS algorithm. Six practical applications are applied for evaluating these algorithms. The six practical applications are the fork tasks, the join tasks, the fork-join tasks, the FFT [3], the Gaussian elimination [19], and the LU-decomposition [11]. The comparisons are based on the schedule lengths, which are generated by these algorithms.

Exploiting schedule-holes could reduce the schedule length efficiently. By the experimental results, the condition B of the DTS algorithm performs better than the insertion-policy in exploiting schedule-holes in the communication-sensitive environments.

The schedule length is the main measure for an algorithm's performance. The upper bound of schedule length is obtained by arranging all tasks on one processor element. The comparison of four algorithms' average schedule lengths is shown in Fig. 3.

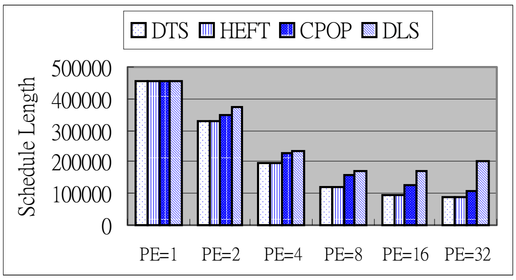


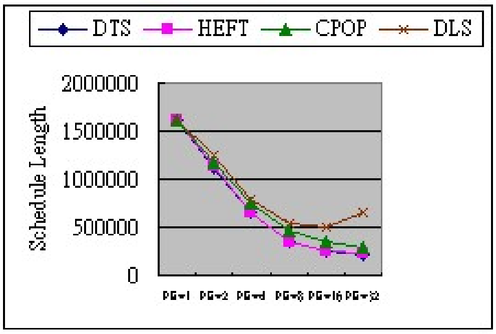
Fig. 3. Comparisons of four algorithms' average schedule length.

In Fig. 3, the schedule length generated by the DTS algorithm is shorter than those of the HEFT, the CPOP, and the DLS algorithms while the number of processor elements is 16, although, the average of the schedule lengths obtained by the DTS algorithm are similar to those obtained by the HEFT algorithm. When the number of processor elements is increasing, the schedule length of each algorithm except the DLS algorithm is decreasing. The phenomenon is the max-min anomaly in the parallel processing problems. Such phenomenon of the DLS algorithm, which generates the longer schedule length, is called max-min anomaly.

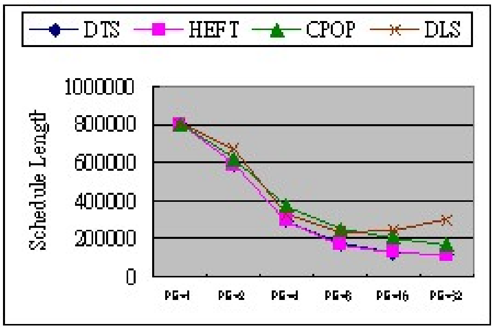
Also, this study compares the schedule length performance of four algorithms with different CCR values, where CCR is computation/communication rate. In this study, the CCR varies from 0.05, 0.1 to 1. The comparisons of average schedule length generated by every algorithm with different CCR values are shown in Fig. 4.

In Fig. 4, the performance of the DTS algorithm is similar to that of the HEFT algorithm, but it is better than that of the HEFT algorithm when CCR=0.05 and CCR=0.1. The DTS algorithm perform better than the HEFT algorithm when CCR=1.

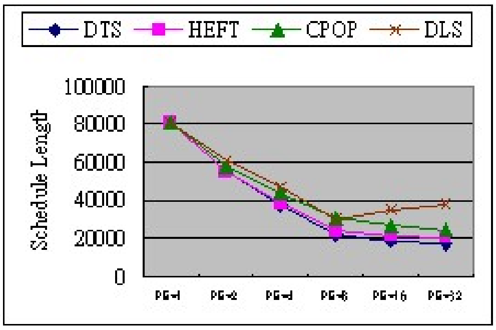
The next experiment is to evaluate the performances of four algorithms by varying the heterogeneity of the communication. The number of PE varies from 2, 4, 8, 16 to 32.



(a) CCR=0.05



(b) CCR=0.1

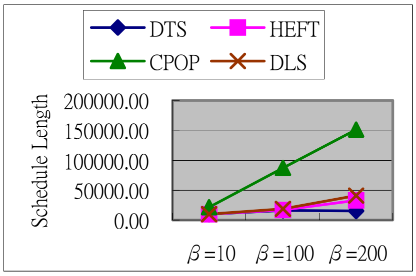


(c) CCR=1

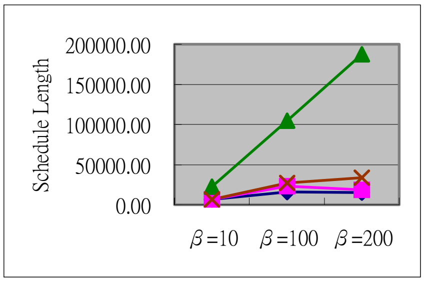
Fig. 4. Comparisons of average schedule length generated by every algorithm with different CCR values.

The mean of each processor's computation rate, α, is equal to 10. The communication rate, β, varies from 10, 100 to 200. Taking the Gaussian elimination task graph for example, the experimental results of schedule length generated by every algorithm with varying β values and the different numbers of processor elements are shown in Fig. 5.

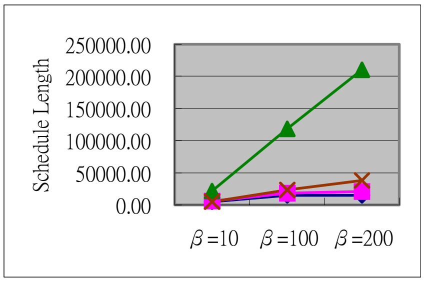
In Fig. 5, as the β increases, the DTS algorithm performs better than other algorithms. This shows that the DTS algorithm could be applied to the higher communication heterogeneity. To simplify the explanations, some examples about the experimental results generated by other types of the task graphs are shown in Fig. 6. In Fig. 6(a), the DTS algorithm performs better than other algorithms when β=100.



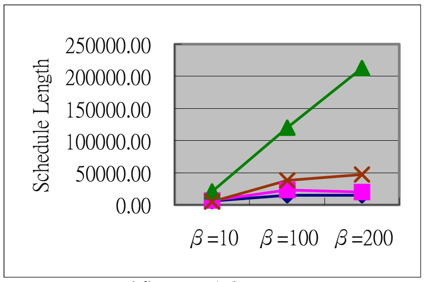
(a) PE=2.



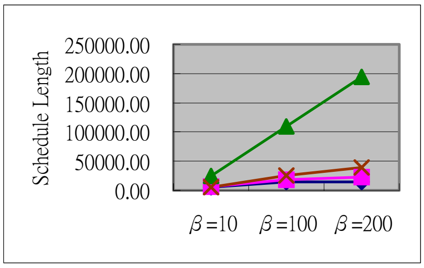
(b) PE=4.



(c) PE=8.

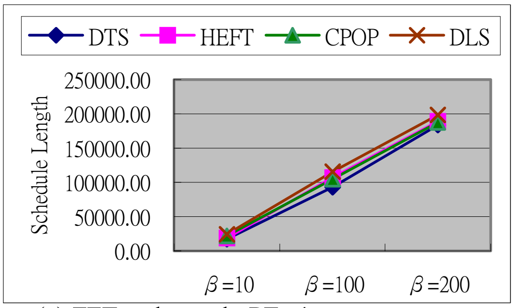


(d) PE=16.

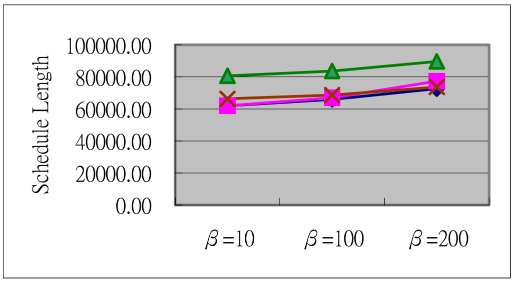


(e) PE=32.

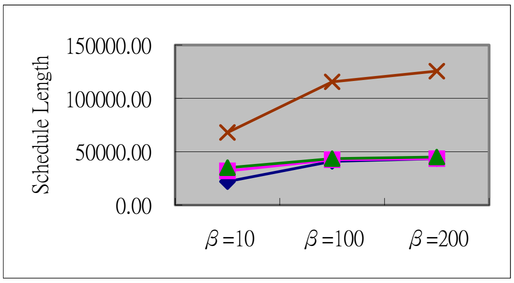
Fig. 5. Schedule length of the Gaussian elimination task graph generated by every algorithm with varying β values and PE numbers.



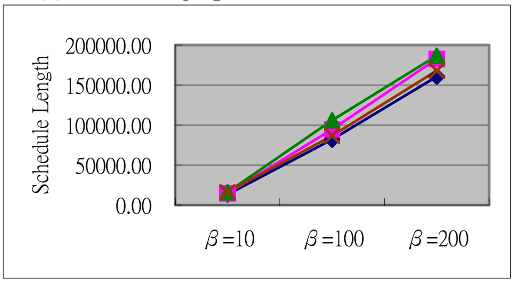
(a) FFT task graph, PE=4.



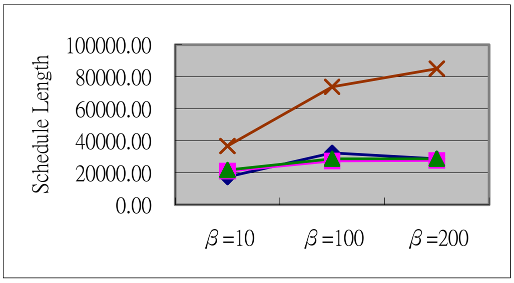
(b) LU-decomposition task graph, PE=8.



(c) Fork task graph, PE=32.



(d) Join task graph, PE=16.



(e) Fork-join task graph, PE=32.

Fig. 6. Experimental results of different types of the task graphs with varying β values and PE numbers.

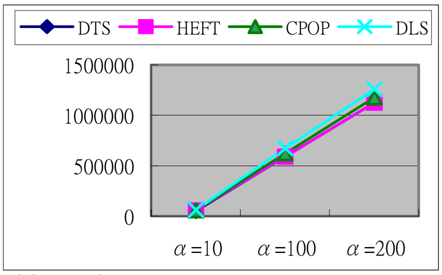
In Fig. 6(b), the DTS algorithm performs better than other algorithms when β=200. Similarly, in Fig. 6(c), the DTS algorithm performs better than other algorithms when β=10. Also, in Fig. 6(d), the DTS algorithm performs better than other algorithms when β=10, 100 to 200. This shows that the DTS algorithm could be applied to the higher communication heterogeneity. In Fig. 6(e), the HEFT algorithm and the CPOP algorithm perform better than the DTS algorithm by using insertion-policy. In the fork-join task graph, a lot of schedule-holes are caused. Although, with insertion-policy, the HEFT algorithm and the CPOP algorithm exploit schedule-holes better than the DTS algorithm in the fork-join task graph, but in other types of task graph, the DTS algorithm perform better than them.

Another experiment is to evaluate the performances of four algorithms with varying the heterogeneity of the computation. The number of PE varies from 2, 4, 8, 16 to 32. The mean of communication rate, β, is equal to 10. The computation rate, α, varies from 10, 100 to 200. The experimental results of average schedule length generated by every algorithm with varying α values and different numbers of processors are shown in Fig. 7.

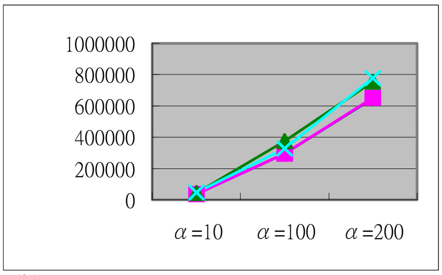
Obviously, with increasing α values, the DTS algorithm and the HEFT algorithm perform better than the CPOP algorithm and the DLS algorithm. This shows that the DTS algorithm and the HEFT algorithm could be applied to the higher heterogeneity of computation.

5 Conclusions

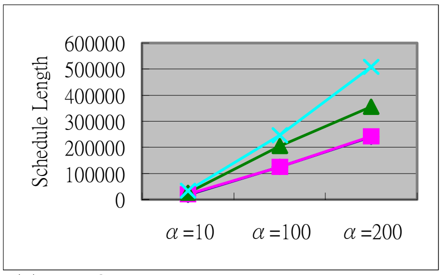
This study proposes a list-scheduling algorithm, the dominant tasks scheduling (DTS) algorithm. In the DTS algorithm, the communication contention is considered. It makes scheduling strategies more suitable for real machines. The global scheduling information could be considered by the condition A. It makes the selection of CP nodes more correct. Moreover, the DTS algorithm could exploit schedule-holes efficiently by using condition B. The experimental results show that the DTS algorithm reduces the completion time well by using condition B. The experimental results show that the DTS algorithm performs better than others algorithms in the system with the higher communication heterogeneity. Also, the DTS algorithm and the HEFT algorithm perform better than others algorithms in the system with the higher computation heterogeneity. In the systems with the high communication heterogeneity or the high computation heterogeneity, the DTS algorithm could perform better than other algorithms by considering global scheduling information and exploiting schedule-holes.



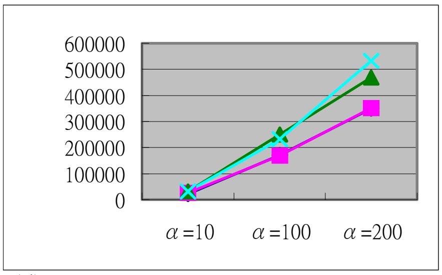
(a) PE=2.



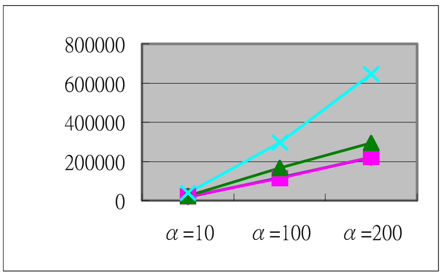
(b) PE=4.



(c) PE=8.



(d) PE=16.



(e) PE=32.

Fig. 7. Experimental results of average schedule length generated by every algorithm with varying α values and PE numbers.

Reference

[1] T. D. Braun, H.J. Siegel, N. Beck, L. L. Bölóni, A. I. Reuther, M. D. Theys, B. Yao, and R. F. Freund, "A Comparison Study of Static Mapping Heuristics for a Class of Meta-Tasks on Heterogeneous Computing Systems," in *Proc. Heterogeneous Computing Workshop*, pp. 15-29, 1999.

[2] T. D. Braun, H.J. Siegel, and A.A. Maciejewski, "Heterogeneous Computing: Goals, Methods, and Open Problems," *2001 International Conference on Parallel and Distributed Processing Technologies and Applications (PDPTA 2001)*, Vol. I, co-sponsors: CSREA, IPSJ, et al., 1-12,Las Vegas, NV, (Invited), June 2001.

[3] Y.C. Chung, C.C. Liu, and J.S. Liu, "Applications and Performance Analysis of A Compile-Time Optimization Approach for List Scheduling Algorithms on Distributed Memory Multiprocessors," *Journal of Information Science and Engineering*, vol. 11, no. 2, pp. 155-181, June 1995.

[4] R.C. Correa, A. Ferreria, and P. Rebreyend, "Integrating List Heuristics into Genetic Algorithms for Multiprocessor Scheduling," in *Proc. Eighth IEEE Symp. Parallel and Distributed Processing (SPDP '96)*, October 1996.

[5] A. Dogan and F. Özgüner, "Matching and Scheduling Algorithms for Minimizing Execution Time and Failure Probability of Applications in Heterogeneous Computing," *IEEE Trans. Parallel and Distributed Systems,* vol. 13, no. 3, pp. 308-323, 2002.

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

[7] B. Kruatrachue and T.G. Lewis, "Grain Size Determination for Parallel Processing," *IEEE software*, pp. 23-32, January 1998.

[8] Y.K. Kwok, "Parallel Program Execution on a Heterogeneous PC Cluster Using Task Duplication," *Heterogeneous Computing Workshop 2000*, pp. 364-374, 2000.

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

[10] Y.K. Kwok and I. Ahmad, "Static

Scheduling Algorithms for Allocating Directed Task Graphs to Multiprocessors," *ACM Computing Surveys, vol. 31, no. 4*, pp. 406-471, December 1999.
[11] R.E. Lord, J.S. Kowalik, and S.P. Kumar, "Solving Linear Algebraic Equations on an MIMD Computer," *Journal of the ACM*, vol. 30, no. 1, pp. 103-117, January 1983.
[12] U. Manber, *Algorithms: A creative approach*, Taipei, Taiwan: Pearson Education Taiwan, 2003.
[13] J.M. Orduña, V. Arnau, A. Ruiz, R. Valero, and J. Duato, "On the Design of Communication-Aware Task Scheduling Strategies for Heterogeneous Systems," *ICPP 2000*, pp. 391-398, 2000.
[14] A. Radulescu and J.C. Arjan, "Low-Cost Task Scheduling for Distributed-Memory Machines," *IEEE Trans. Parallel and Distributed Systems,* vol. 13, no. 6, pp. 648-658, 2002.
[15] S. Ranaweera and D.P. Agrawal, "A Scalable Task Duplication Based Scheduling Algorithm for Heterogeneous Systems," *ICPP 2000*, pp. 383-390, 2000.
[16] G.C. Shih and E.A. Lee, "A Compile-Time Scheduling Heuristic for Interconnection-Constrained Heterogeneous Processor Architectures," *IEEE Trans. Parallel and Distributed Systems,* vol. 4, no. 2, pp. 175 –187,1993.
[17] H. Topcuoglu, S. Hariri, and M.Y. Wu, "Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Computing," *IEEE Trans. Parallel and Distributed Systems, vol. 13, no.* 3, pp. 260-274, 2002.
[18] J.B. Weissman, "Scheduling Multi-Component Applications in Heterogeneous Wide-Area Networks," *Heterogeneous Computing Workshop 2000*, pp. 209-215, 2000.
[19] M.Y. Wu and D. Gajski, "Hypertool: A Programming Aid for Message-Passing Systems," *IEEE Transactions on Parallel and Distributed Systems, vol. 1, no.* 3, pp. 330-343, 1990.
[20] T. Yang and A. Gerasoulis, "DSC: Scheduling Parallel Tasks on an Unbounded Number of Processors," *IEEE Transactions on Parallel and Distributed Systems, vol. 5, no.* 9, pp. 951-967, 1993.
[21] T. Yang and A. Gerasoulis, "List Scheduling With and Without Communication Delays," *Parallel Computing,* vol. 19, no. 12, pp. 1321-1344, 1993.