A Productive Duplication-Based Scheduling Algorithm for Heterogeneous Computing Systems

Young Choon Lee and Albert Y. Zomaya

Advanced Networks Research Group, School of Information Technologies,
The University of Sydney, NSW 2006, Australia
{yclee, zomaya}@it.usyd.edu.au

Abstract. The scheduling problem has been shown to be NP-complete in general cases, and as a consequence many heuristic algorithms account for a myriad of previously proposed scheduling algorithms. Most of these algorithms are designed for homogeneous computing systems. This paper presents a novel scheduling algorithm for heterogeneous computing systems. The proposed method is known as the Productive Duplication-based Heterogeneous Earliest-Finish-Time (PDHEFT) algorithm. The PDHEFT algorithm is based on a recently proposed list-scheduling heuristic known as the Heterogeneous Earliest-Finish-Time (HEFT) algorithm which is proven to perform well with a low time complexity. However, the major performance gain of the PDHEFT algorithm is achieved through its distinctive duplication policy. The duplication policy is unique in that it takes into account the communication to computation ratio (CCR) of each task and the potential load of processors. The PDHEFT algorithm performs very competitively in terms of both resulting schedules and time complexity. In evaluating the PDHEFT algorithm a comparison is made with another two algorithms that have performed relatively well, namely, the HEFT and LDBS algorithms. It is shown that the proposed algorithm outperforms both of them with a low time complexity.

1 Introduction

Task scheduling problems have been extensively studied for many years. However, due to the NP-complete nature of the task scheduling problem in most cases [1] heuristic algorithms account for a myriad of existing scheduling algorithms. Therefore, the time complexity of a task scheduling algorithm is one of the most fundamental factors in determining its quality. In addition to time complexity, the minimization of the schedule length is another main objective of a task scheduling algorithm. Hereafter, scheduling and task scheduling are used interchangeably.

Heuristic based scheduling algorithms are normally the ones favored by a large number of researchers. Three major sub categories of heuristic based scheduling are list scheduling, clustering and task duplication. List scheduling in the heuristic based category is preferred to other scheduling techniques. This is due to the fact that list scheduling algorithms [2], [3], [4], [5], [6], [7], [8] tend to produce competitive solutions with lower time complexity compared to those of the algorithms in the other subcategories [9]. The two fundamental phases commonly found in list scheduling are task prioritization and processor selection.

This paper proposes a duplication-based scheduling algorithm known as the Productive Duplication-based Heterogeneous Earliest-Finish-Time (PDHEFT). It is based on a recent list-scheduling algorithm, HEFT. The PDHEFT algorithm schedules tasks in a task graph for heterogeneous computing systems with a distinctive duplication policy. The duplication policy is unique in that it takes the communication-to-computation cost ratio (CCR) of each task and the potential load of processors into consideration in order to avoid redundant duplications that might increase the schedule length. Despite the adoption of an additional duplication phase the time complexity of the PDHEFT algorithm still remains the same as that of the HEFT algorithm but with better quality schedules.

The target system used in this work consists of heterogeneous processors/machines that are fully interconnected. The inter-processor communications are assumed to perform with the same speed on all links without contentions. It is also assumed that a message can be transmitted from one processor to another while a task is being executed on the recipient processor which is possible in many systems.

The remainder of this paper is organized as follows. Section 2 introduces some background material on scheduling problems. The proposed algorithm is described in great detail in Section 3. In Section 4, the evaluation results are presented and explained with conclusions following in Section 5.

2 Scheduling Problem

Parallel programs, in general, can be represented by a directed acyclic graph (DAG). A DAG, $G = (V, E)$, consists of a set of v nodes, V, and e edges, E. A DAG is also called a task graph or macro-dataflow graph. In general, the nodes represent tasks partitioned from an application and the edges represent precedence constraints. An edge $(i, j) \in E$ between task n_i and task n_j also represents the inter-task communication. More specifically, the output of task n_i has to be transmitted to task n_j in order for task n_j to start its execution. A task with no predecessors is called an entry task, n_{entry}, whereas an exit task, n_{exit}, is one that does not have any successors.

A task is called a *ready task* if all of its predecessors have been completed. A *level* is associated with each task. The level of a task is defined to be:

$$level(n_i) = \left\{ \begin{array}{ll} 0\ , & if\ n_i = n_{entry} \\ \max\limits_{n_j \in imed\_pred(n_i)} \{level(n_j)\} + 1, & otherwise \end{array} \right\} \quad (1)$$

where *imed_pred(n_i)* is the set of immediate predecessor tasks of task n_i.

The weight on a task, n_i denoted as w_i represents the computation cost of the task. In addition, the computation cost of a task, n_i is on a processor, p_j is denoted as $w_{i,j}$.

The weight on an edge, denoted as $c_{i,j}$ represents the communication cost between two tasks, n_i and n_j. However, communication cost is only required when two tasks are assigned to different processors. In other words, the communication cost when they are assigned to the same processor can be ignored, i.e., 0. The average computation cost and average communication cost of a task, n_i are denoted as $\overline{w_i}$ and $\overline{c_i}$, respectively. The former is the average computation cost of task n_i over all the processors in a given system. The latter is the average communication cost between task n_i and its successor tasks.

Three frequently used task prioritization methods are *t-level*, *b-level* and *s-level.* The t-level of a task is defined as the summation of the computation and communication costs along the longest path of the node from an entry task in the task graph. The task itself is excluded from the computation. In contrast, the b-level of a task is computed by adding the computation and communication costs along the longest path of the task from an exit task in the task graph (including the task). The only distinction between the b-level and s-level is that the communication costs are not considered in the s-level.

The CCR is a measure that indicates whether a task graph is communication intensive, computation intensive or moderate. For a given task graph, it is computed by the average communication cost divided by the average computation cost on a target system. The CCR of a task n_i is defined by:

$$CCR(n_i) = \frac{\sum_{n_j \in imed\_succ(n_i)} c_{i,j}}{S_i} / \overline{w_i} \tag{2}$$

where S_i corresponds to the number of the immediate successors of task n_i.

If a set P of p processors exists for scheduling, the earliest start time and the earliest finish time of a task n_i on a processor p_j are denoted as $EST(n_i, p_j)$ and $EFT(n_i, p_j)$, respectively. The earliest start time of task n_i on processor p_j is defined to be whichever is the maximum: the earliest available time of the processor or the communication completion time of the task of its predecessors. The predecessor of a task n_i from which the communication completes at the latest time is called the most influential parent (MIP) of the task denoted as $MIP(n_i)$. The communication completion time, also called data arrival time of task n_j from task n_i is denoted as $CCT(n_i, n_j)$. The schedule length (*SL*), also called *makespan*, is defined as $max\{EFT(n_{exit})\}$ after the scheduling of v tasks in a task graph G is completed.

3 Proposed Algorithm

3.1 Algorithm Description

The PDHEFT algorithm uses the same fundamental operations used by the HEFT algorithm, such as task prioritization and processor selection. However, the major performance gain of the PDHEFT algorithm is from the distinctive duplication policy it proposes. The decision of the duplication is made based on two factors: the CCR of a task and the potential load of the processors. These factors differentiate the PDHEFT algorithm from other duplication based scheduling algorithms. The PDHEFT algorithm consists of three main phases:

- Task Prioritization Phase – assigns priorities and levels to tasks and arranges the tasks in decreasing order by b-level value.
- Processor Selection Phase – selects the processor on which the task finishes the earliest.
- Duplication Phase – determines which task to be duplicated by taking into account the CCR of each task and the load of the processors in the target system.

The workings of the PDHEFT algorithm are given in Fig. 1.

```
1. Compute average communication and computation costs of tasks
2. Compute b-level values, levels and the number of tasks in each level of all tasks
3. Sort the tasks in a scheduling list in decreasing order by b-level value
4. **while** there are unscheduled tasks in the list **do**
5.    Select the first unscheduled task, $n_i$, from the list
6.    **for** each processor $p_j$ in $P$ **do**
7.       Compute $EFT(n_i, p_j)$
8.       Add $EFT(n_i, p_j)$ to the processor reference list, $PR$ in increasing order by finish time
9.    **endfor**
10.   Assign task $n_i$ to the processor, $p*$ that minimizes its finish time
11.   Remove $p*$ from $PR$
12.   **if** task has out-degree of two or more and the number of parallelized tasks in the same level of that of task $n_i$ is not greater than $P$ **then**
13.      Duplicate task $n_i$ as many as its out-degree based on duplication criteria
14.   **endif**
15.**endwhile**
```

Fig. 1. The PDHEFT algorithm

The time complexity of the PDHEFT algorithm is $O(ep)$ which is identical to that of the HEFT algorithm. For a dense graph, the number of edges is proportional to $O(v^2)$. Thus, the time complexity of the PDHEFT algorithm is in $O(v^2p)$. This is far lower than the time complexities of the two versions of the LDBS algorithm, that are $O(v^3ep^3)$ and $O(v^3ep^2)$.

3.2 Duplication Policy

The duplication of a task is considered as soon as the task is scheduled for the first time using the processor selection procedure of the PDHEFT algorithm.

The duplicability of the task is then checked based on its CCR and out-degree. A task with an out-degree of two or more is regarded as a candidate for duplication. In order not to increase the time complexity of the PDHEFT algorithm, duplications apply only to the task currently being scheduled. In addition, the necessary information for duplication is obtained during processor selection phase.

As mentioned earlier, the duplication policy developed in this research introduces two distinct measures to determine if the duplication of a task is allowed. The first measure is the CCR of each task in a task graph. The CCR of a task rather than the CCR of a task graph is used because in the PDHEFT algorithm the duplication of a task only concerns the task and its successor tasks, i.e. a sub-graph. It was assumed that the CCR of this sub-graph, therefore, is a more significant factor than the CCR of the task graph. This leads to the fact that there might be a noticeable difference between the CCR of a task and that of the task graph. It is to be noted that duplicating tasks in a 'computation intensive' task graph is normally impractical.

Moreover, duplicating computation intensive tasks in a 'communication intensive' task graph may cause an increase of the output schedule. The CCR of a task is used to compute the *allowance* of the finish time of the task when duplicating. This means that the finish time of a task varies in a heterogeneous computing system depending on the processor that executes it. This allowance is an additional time that can be provided for the finish time of the task. If its finish time on the particular processor on which it is being duplicated is substantially larger than its minimal finish time the duplication of the task has a high possibility of being an unproductive. The allowance computation algorithm is presented in Fig. 2.

A task is classified into four types: computation intensive, moderate, communication intensive and extremely communication intensive. This classification is conducted based on the CCR of the task. The thresholds of the four types are shown in Fig. 2. These thresholds may need to be changed according to the target application model.

```
if CCR of task $n_i$ < 0.5 then          /* computation intensive */
    Let allowance = $\overline{c_i}$
else if CCR of task $n_i$ < 1.0 then     /* moderate */
    Let allowance = $\overline{w_i}$
else if CCR of task $n_i$ < 5.0 then     /* comm. intensive */
    Let allowance = $\overline{c_i}$
else                                     /* extremely communication intensive */
    Let allowance = $\overline{c_i}$ / 2
endif
```

Fig. 2. The algorithm for computing allowance

In addition to CCRs of tasks, the potential load of the processors in a given system is used to predict whether duplicating a task becomes a source of delays or interruptions for the execution of remaining unscheduled tasks.

More precisely, if there are more tasks in a level than the number of processors the tasks in that level are not considered for duplication.

The duplication algorithm is shown in Fig. 3. The `Duplicate` function is called if a task satisfies the second measure of the PDHEFT's duplication policy; that is, the function is called if the number of tasks in the same level as that of the task is no greater than the number of processors.

```
1. Duplicate( )
2. Compute the allowance of task $n_i$
3. while task $n_i$ is not duplicated as many as the number of its out-degree and there
   are unchecked processors do
4.   Select $p_j$ from $PR$
5.   Let $allowedftime = EFT(n_i, p^*) + allowance$
6.   if $EFT(n_i, p_j) < allowedftime$ then
7.     Duplicate task $n_i$ on $p_j$
8.   endif
9. endwhile
10.end
```

Fig. 3. The duplication algorithm of the PDHEFT algorithm

4 Performance Results and Comparison

The comparative evaluation of the PDHEFT algorithm is presented in this section. Comparisons have been conducted between two previously proposed scheduling algorithms, HEFT and LDBS [10], and the PDHEFT algorithm. The two former are chosen because they have been shown to deliver competitive output schedules. In addition, their target system configurations are the same as those used for the PDHEFT algorithm.

The two performance metrics used for comparison are the *normalized schedule length* (NSL) and time complexity. Typically, the schedule length of a task graph generated by a scheduling algorithm is used as the main performance measure of the algorithm. The normalized schedule length is defined as:

$$NSL = \frac{schedule\ length\ obtained\ by\ a\ particular\ algorithm}{schedule\ length\ obtained\ by\ the\ PDHEFT} \tag{3}$$

4.1 Test Parameters

The proposed algorithm and the two previously proposed algorithms, HEFT and LDBS are extensively experimented with various types of both randomly generated and well-known application task graphs. The three well-known parallel applications used for our experiments are the Laplace equation solver [11], the LU-decomposition [12] and Fast Fourier Transformation [13]. The numbers of random and well-known application task graphs are 1566 and 270, respectively. The common parameters used to populate the variations of the task graphs are:

- 9 different CCRs of 0.1, 0.2, 0.3, 0.5, 1.0, 2.0, 3.0, 5.0 and 10.0,
- 3 different processor heterogeneity values of 100, 200 and random.

The processor heterogeneity value of 100 is defined to be the percentage of the speed difference between the fastest processor and the slowest processor in a given system.

4.2 Performance Results with Random Task Graphs

The test results obtained from the random task graphs are presented in two different categories. The first category is as shown in Fig. 4 where comparisons between the three algorithms are conducted with various graph sizes on a computing system consisting of 20 heterogeneous processors. In the second category, an increasing number of processors are used as shown in Fig. 5.

Although the tests for each category are carried out with nine different CCRs as mentioned in Section 4.1, three significant test results are presented. As shown in Figs. 4 and 5, they are CCRs of 0.1, 1.0 and 10.0.

4.2.1 Comparisons with Various Graph Sizes

It is clearly shown in Fig. 4 that the PDHEFT algorithm delivers quite competitive schedule lengths irrespective of different graph sizes and CCRs. The schedule lengths obtained from communication intensive and moderate task graphs shown in Figs. 4a and 4b indicate that the PDHEFT algorithm best suits task graphs consisting of fine-grain tasks with large communication costs. This is also true for the LDBS algorithm. However, its performance drops noticeably with computation intensive task graphs as shown in Fig. 4c. The main reason for this is because LDBS does not take CCR into account. It is observed that many of the duplications tend not to contribute to shortening schedule lengths when scheduling computation intensive task graphs. The PDHEFT algorithm, however, overcomes this drastic degradation by restricting redundant duplications. It, therefore, tends to give shorter schedule lengths for computation intensive task graphs compared to those generated by LDBS.

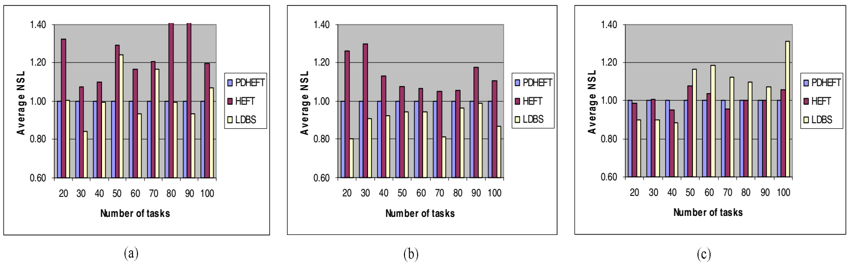


Fig. 4. Average NSL of DAGs with (a) CCR = 10/1, (b) CCR = 1/1 and (c) CCR = 1/10 under the PDHEFT, HEFT and LDBS algorithms with respect to graph size

The average schedule length of the PDHEFT algorithm computed based on the first test set shown in Fig. 4 is 11% on average and 23% at best smaller than that of the HEFT algorithm. It is observed that the LDBS algorithm delivers 6% smaller average schedule length than that of the PDHEFT algorithm for communication intensive and moderate task graphs. However, the PDHEFT generates an average schedule length that is 4% on average, 24% at best and 20% for computation intensive task graphs smaller than that of the LDBS algorithm.

4.2.2 Increasing Number of Processors

The test results shown in Fig. 5 are obtained from the second test set, 1323 task graphs. The patterns that were found in Fig. 4 are re-confirmed in Fig. 5. First, the PDHEFT algorithm performs very reliably regardless of the different characteristics of task graphs. Second, as shown in Fig. 5a the LDBS algorithm tends to deliver longer schedule lengths than that of the PDHEFT algorithm when the processors in a given system are overloaded even though task graphs are communication intensive.

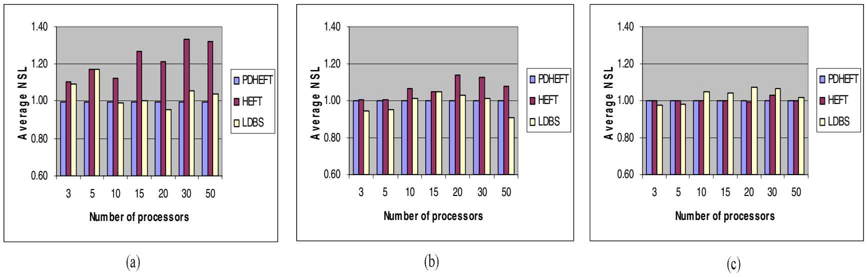


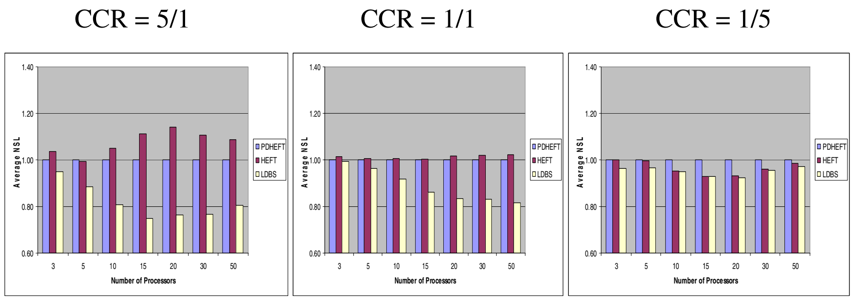
Fig. 5. Average NSL of DAGs with (a) CCR = 10/1, (b) CCR = 1/1 and (c) CCR = 1/10 under the PDHEFT, HEFT and LDBS algorithms with respect to increasing number of processors

Finally, the duplication for computation intensive task graphs does not improve the performance of the LDBS algorithm in terms of schedule length. The average schedule length of the HEFT algorithm for computation intensive task graphs as shown in Fig. 5c also proves that the LDBS algorithm performs some redundant duplications. With the second test set, the PDHEFT algorithm overall outperforms the HEFT and LDBS algorithms. The average schedule length of the PDHEFT algorithm is 8% on average and 18% at best smaller than that of the HEFT algorithm. The average schedule length of the LDBS algorithm is 2% on average longer than that of the PDHEFT algorithm.

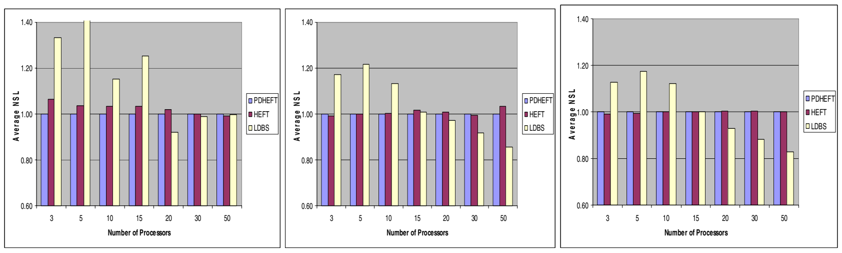
4.3 Performance Results with Well-Known Application Task Graphs

The performance results of the PDHEFT algorithm obtained from the experiments conducted with a wide range of different task graphs of the three well-known applications once again confirm its better practicability over the other two algorithms. As shown in Fig. 6 the PDHEFT algorithm achieves a consistent performance irrespective of various types of task graphs.

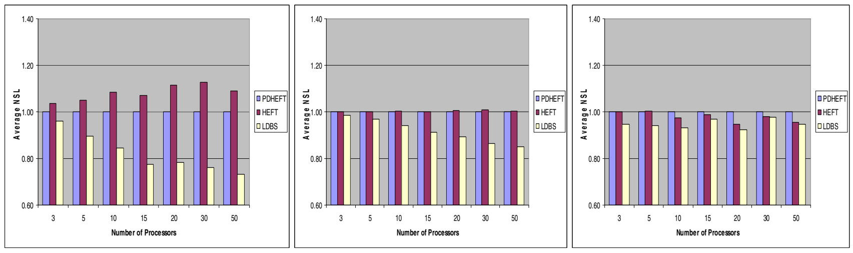
Note, that the LDBS algorithm with a large number of processors tends to outperform both the PDHEFT and HEFT algorithms. This in fact indicates the impracticability of the LDBS algorithm. More specifically, when there are relatively more tasks in a task graph than the number of processors in a given system, which is quite normal in practice, the LDBS algorithm tends to generate a longer schedule length compared to that of the PDHEFT and HEFT algorithms. Moreover, the performance of the LDBS algorithm drops noticeably when a task graph contains a number of levels on each of which many tasks have the same or similar upward rank as shown in Fig. 6b,



(a)



(b)



(c)

Fig. 6. Average NSL of Well-known Application DAGs (a) Laplace (b) LU (c) FFT

the performance results with the LU task graphs. This shortcoming occurs because the LDBS algorithm tries to duplicate even though the load of the processors is high. This leads to the necessity of predicting the load of the processors that is one of the main characteristics of PDHEFT's duplication policy.

5 Conclusion

In this paper, a new duplication based scheduling algorithm, called the PDHEFT algorithm was presented, for heterogeneous computing systems. The algorithm is based on a previously proposed and well-known algorithm, called the HEFT algorithm. A number of intensive experiments with various test configurations have been conducted. Based on the test results, the PDHEFT algorithm showed its practicability and mostly outperformed existing algorithms including HEFT and LDBS. Because of its robust duplication policy, it delivers competitive schedule lengths regardless of the characteristics of task graphs and the processor configuration in a given system. The robust duplication policy is achieved by taking two very influential factors of the scheduling process into account. They are the CCR of a given task graph and the load of the processors in a given system. In addition to the high quality of the output schedules the low time complexity of the proposed method should be highly attractive.

Acknowledgements

Professor Albert Y. Zomaya's work is supported by an Australian Research Council grant no. DP0452884.

References

1. Garey, M.R. and Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman and Co. (1979) 238–239.
2. Topcuoglu, H., Hariri, S. and Wu, M.: Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Computing. IEEE TPDS. V. 13. No. 3. (2002) 260-274.
3. Radulescu, A. and Gemund. A.J.C.: Fast and effective task scheduling in heterogeneous systems. Proc.HCW (2000) 229-238.
4. Radulescu, A. and Gemund, A.J.C.: On the complexity of list scheduling algorithms for distributed memory systems. Proc. 13th ACM Int'l Con. Supercomputing. (1999) 68–75.
5. Radulescu, A. and Gemund, A.J.C.: FLB: Fast Load Balancing for distributed-memory machines. Proc. ICPP. (1999) 534–541.
6. Sih, G.C. and Lee, E.A.: A Compile-Time Scheduling Heuristic for Interconnection-Constrained Heterogeneous Processor Architectures. IEEE TPDS. V. 4. No. 2. (1993) 175–187.
7. Kruatrachue, A. and Lewis, T.G.: Grain Size Determination for Parallel Processing. IEEE Software. (1988) 23–32.
8. Hwang, J.J., Chow, Y.C., Anger, F.D. and Lee, C.Y.: Scheduling Precedence Graphs in Systems with Interprocessor Communication Times. SIAM J. Computing. V. 18. No. 2. (1989) 244–257.
9. Kwok, Y.K. and Ahmad, I.: Benchmarking the Task Graph Scheduling Algorithms. Proc. First Merged Int'l Parallel Symp./Symp. Parallel and Distributed Processing Conf. (1998) 531–537.
10. Dogan, A. and Ozguner, R.: LDBS: A Duplication Based Scheduling Algorithm for Heterogeneous Computing Systems. Proc. ICPP. (2002) 352–359.
11. Wu, M.-Y. and Gajski, D.D.: Hypertool: A Programming Aid for Message-Passing Systems. IEEE TPDS. V. 1. No. 3. (1990) 330-343.
12. Lord, R.E., Kowalik, J.S., and Kumar, S.P.: Solving Linear Algebraic Equations on an MIMD Computer. J. ACM. V. 30. No. 1. (1983) 103-117.
13. Cormen, T.H., Leiserson, C.E., and Rivest, R.L.: Introduction to Algorithms. MIT Press. (1990).