The Task Scheduling with Adaptive Branch and Bound *

Chih-Kai Wu[a] and Chang-Biau Yang[a†] and Kuo-Tsung Tseng[b]
[a]Department of Computer Science and Engineering
National Sun Yat-sen University, Kaohsiung, Taiwan
[b]Department of Shipping and Transportation Management
National Kaohsiung University of Science and Technology, Kaohsiung, Taiwan

Abstract

Most versions of tasking scheduling problems are NP-hard. The list-based task scheduling method has a good balance among many types, so it is widely used in different task scheduling structures. HEFT and PEFT are very representative and classic list-based task scheduling methods. Both of them find the task ordering in the first stage. And then, in the second stage, they allocate the tasks to processors. In this paper, we apply the heuristic branch and bound method to perform the task allocation for a given task ordering (given by HEFT or PEFT) under the heterogeneous environment. We design a good estimation function for evaluating the makespan in the future. With the estimated scores of the branches, we prune away those branches that do not unlikely lead into an optimal solution. As the experimental results show, we get better solutions than the previous heuristic algorithms. Our algorithm is adaptive with an adjustable parameter which is a trade-off between the required time and the quality of the solution. Therefore, our algorithm can adapt the task scheduling in different situations to efficiently obtain good answers under different criteria.

Keywords: task scheduling, task ordering, heterogeneous, list-based scheduling, branch and bound

1 Introduction

The task scheduling problem [1–3] is an important computation issue in the field of parallel processing. Most versions of the task scheduling problems are NP-hard [4]. A task scheduling problem can usually be represented by a *directed acyclic graph* (DAG), which describes the calculation flow between tasks.

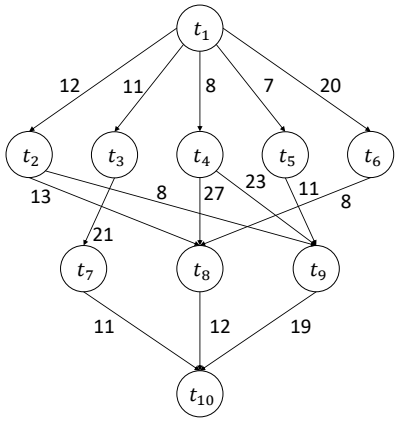

Figure 1: A DAG for representing a task scheduling problem.

Definition 1. (heterogeneous task scheduling)
Let $t_1, t_2, \cdots, t_v$ denote a set of tasks, and let m be the number of the heterogeneous processors. A DAG can be used to represent the heterogeneous task scheduling problem, where each node represents a task t_i and an edge $e_{i,j}$ from task t_i to t_j means that task t_i has to be finished before task t_j. t_i is called the parent (predecessor) *of t_j, and t_j is called the* child (successor) *of t_i. A task without a parent is called an* entry task, *and a task without children is called an* exit task. *Each task may require different execution time on different processors. If two tasks are assigned to the same processor, the* data transfer time (communication time) *is zero. The time between the first starting entry task and the final finishing exit task is defined as* makespan. *The goal of the task scheduling problem is to minimize the makespan under the given constraints.*

An example of the task scheduling on a heterogeneous network is shown in Figure 1 and Table

*This research work was partially supported by the Ministry of Science and Technology of Taiwan under contract MOST 109-2221-E-110-040-MY2.
†Corresponding author. E-mail: cbyang@cse.nsysu.edu.tw (Chang-Biau Yang).

Table 1: The computation cost for the DAG of Figure 1 in the heterogeneous environment.

task \ processor	p_1	p_2	p_3
t_1	11	12	15
t_2	13	18	14
t_3	9	12	10
t_4	10	13	16
t_5	11	17	8
t_6	12	15	7
t_7	8	11	15
t_8	14	8	13
t_9	19	17	20
t_{10}	23	13	18

1. Note that for a general case, the network may not be a layer-by-layer graph (multi-stage graph).

List-based task scheduling algorithms are generally composed of two stages: (1) *task ranking stage*: to calculate the priority of each task and to determine the assigned order; (2) *task allocation stage*: to select the most appropriate processor to perform each task, with the goal of makespan minimization. *Heterogeneous earliest finish time* (HEFT) [5, 6] and *predict earliest finish time* (PEFT) [7] are two of the most important and well-known list-based task scheduling algorithms, from which some later algorithms were developed [8]. Table 2 shows some list-based scheduling algorithms for the heterogeneous network. These algorithms usually have the advantage of low computation complexity, but the scheduling results may not be so good.

In this paper, we deal with the static task scheduling problem in the heterogeneous environment, aiming to minimize the makespan of all tasks. In our algorithm, we first use HEFT or PEFT task ranking methods to get the task order, and then we apply the heuristic branch and bound approach to the task allocation on the processors. We design an estimated function for evaluating the goodness of an unexpanded branch in the tree searching. If the estimated value of one branch exceeds the current minimal makespan, we will discard this branch. Through the adjustment of parameters, the estimated value is adaptive to find the solution quickly, or to spend more time for finding better solutions.

As the experimental results show, the makesapns obtained by our algorithm improve those found by the previous heuristic algorithms. In the case of large CCR (*communication-to-computation ratio*), our algorithm can shorten about 20% of the makespan compared with HEFT and PEFT.

The rest of this paper is organized as follows. Section 2 presents some related works of the heterogeneous task scheduling problem. In Section 3, we propose our heuristic branch and bound algorithm for assigning processors to tasks. Section 4 evaluates the proposed algorithm with experiments. Section 5 gives the conclusion of this paper.

2 Preliminaries

A common goal of task scheduling [1–3] is to minimize the makespan. Most task scheduling problems are NP-hard [4], so many heuristics algorithms were proposed.

The levelized minimal time (LMT) algorithm [10] divides the problem into two stages. The first stage is to group the tasks of different layers in DAGs, and then to sort the tasks of each layer separately. The second stage uses a greedy method to assign each task to the "fastest" available processor. Tasks in lower levels have higher scheduling priority than those in higher levels.

The critical path on a processor (CPOP) algorithm [5, 6] determines the critical path in two directions: upwards and downwards. In the calculation of the critical path, the average calculation cost is used as the cost of one task. After the critical path and ranking are decided, the tasks on the critical path is assigned to the same processor for decreasing the communication cost.

Heterogeneous earliest finish time (HEFT) [5, 6] solves the problem in two stages: ranking stage and allocation stage. In the ranking stage, tasks are ordered according to the rank values from the highest to the lowest. HEFT then determines the start time and completion time of each task on different processors, and assigns the task to the processor that completes the task earliest. HEFT uses the insertion strategy to schedule the task to the earliest free time slot on the allocated processor.

The lookahead scheduling algorithm [8] is a heuristic algorithm that improves HEFT. The local optimal decision made by this algorithm is not only through the estimation of a single task, but also looks ahead to the future schedule, and considers the information about the impact of the decision. In most cases, it can effectively reduce the makespan without too much execution time.

The *predict earliest finish time* (PEFT) [7] is an algorithm with the same time complexity of HEFT. It may achieve better performance than HEFT under the same CCR. PEFT obtains forward-looking information by establishing an optimistic cost table in the task ranking stage.

Table 2: List-based scheduling algorithms, where v and m denote the numbers of tasks and processors, respectively.

Year	Algorithm	Author(s)	Time complexity	Note
1990	MH [9]	El-Rewini and Lewis	$O(v^2m)$	Task continuously
1995	LMT [10]	Iverson *et al.*	$O(v^2m^2)$	Greedy method
1999	CPOP [5, 6]	Topcuoglu *et al.*	$O(v^2m)$	Critical path
1999	HEFT [5, 6]	Topcuoglu *et al.*	$O(v^2m)$	Task ranking
2005	HPS [11]	Ilavarasan *et al.*	$O(v^2m \log v)$	Three-stage scheduling
2007	PETS [12]	Ilavarasan and Thambidurai	$O(v^2m \log v)$	Three-stage scheduling
2010	Lookahead [8]	Bittencourt *et al.*	$O(v^4m^3)$	Improving HEFT
2013	PEFT [7]	Aravnejad and Barbosa	$O(v^2m)$	Optimistic tables

Venugopalan and Sinnen [13] proposed an *integer linear programming* (ILP) method for solving the homogeneous task scheduling problem, in which the computation costs of a task on different processors are the same. However, their ILP method can obtain the optimal solutions only for the data instances with small sizes. For large data instances, the running time of the ILP method is unacceptable.

3 Our Algorithm

For given a task ordering (such as HEFT ranking or PEFT ranking), we propose a heuristic branch and bound approach to assign tasks to processors in the task allocation stage. To balance the tradeoff between the solution quality (makespan) and execution time of our algorithm, a parameter α, $0 \leq \alpha \leq 1$, is used to control the size of the branches in the searching tree.

Suppose that m processors $p_1, p_2, \cdots, p_m$ and v tasks $t_1, t_2, \cdots, t_v$ with ordering $\pi = \langle \pi_1, \pi_2, \cdots, \pi_v \rangle$ are given. For example, the HEFT ranking (ordering) for Table 1 is $\pi = \langle 1, 4, 2, 5, 3, 6, 9, 8, 7, 10 \rangle$. Our algorithm assigns the tasks to the processors with the order $\langle t_{\pi_1}, t_{\pi_2}, \cdots, t_{\pi_v} \rangle$. For simplicity, t_{π_j} is denoted by t'_j. Let $\tau_{j,i}$ be the computation cost for task t'_j on processor i, and $c_{i,j}$ be the communication cost from task t'_i to task t'_j if they are assigned to different processors. And let δ_j be the maximum of the communication costs for the parents to task t'_j. That is, $\delta_j = \max_{t'_i \in pred(t'_j)}\{c_{i,j}\}$, where $pred(t'_j)$ denotes the set of parent tasks of t'_j.

Definition 2. (Estimation fucntion) *Let $S_k = \langle q_1, q_2, \cdots, q_k \rangle$ represent a list of processors, where $1 \leq q_i \leq m$, $1 \leq k \leq v$, m and v denote the numbers of processors and tasks, respectively. And let $M(S_{k-1}, q_k)$ denote the makespan that each t'_i is assigned to processor q_i, for $1 \leq i \leq k$. The* estimation function *of S_k with the future assignment is given as follows.*

$$E(S_{k-1}, q_k, \alpha) = \max_{1 \leq i \leq m} \{ M(S_{k-1}, q_k) + \frac{1}{m} \times \sum_{j=k+1}^{v} (\frac{\tau_{j,i}}{2-\alpha} + \frac{\alpha \times \delta_j}{m-\alpha}) \}. \quad (1)$$

It is easy to see that $E(S_{v-1}, q_v, \alpha) = M(S_{v-1}, q_v)$, since all tasks have been assigned. In Equation 1, the left part means that the makespan obtained from the existing assignment S_k. The right part is the estimated value for the assignment of task t'_j, $k+1 \leq j \leq v$ in the future, including computation cost and communication cost. The value of α, $0 \leq \alpha \leq 1$, is used to control the weights of the left and right parts.

When $\alpha = 1$, the weight of the right part is large. In our experience, if communication cost is estimated by the average of all communication costs to t'_j, it is usually overestimated. Therefore, $\frac{\alpha \times \delta_j}{m-\alpha}$ is used as the estimated communication cost so that the communication cost is decreased for the estimation. Besides, the tasks should be equally distributed to all processors, the primitive estimated value is divided by the number m of processors. If α is close to 1, our algorithm cuts more branches by overestimating the value. Thus, the required time is reduced, but possible better solutions may be lost. On the other hand, when $\alpha = 0$ (extreme situation), the communication cost is ignored, thus the estimated value is small. In this situation, more branches are kept and the better, even optimal, solutions could be found. But, the required time is increased greatly.

Let $FT(t'_k, q_k)$ denote the finish time of a task t'_k executed on a processor q_k; $T(q_k)$ be the earliest time when processor q_k is ready; $AFT(t'_i)$ denote the finish time for task t'_i. Then $\max_{t'_i \in pred(t'_k)}\{AFT(t'_i) + c_{i,k}\}$ represents the time when all parents of t'_k are ready for t'_k executed on processor q_k. Note that $c_{i,k} = 0$ if $q_i = q_k$ (they are executed on the same processor). Then we

have

$$FT(t'_k, q_k) = \max\{T(q_k), \max_{t'_i \in pred(t'_k)}\{AFT(t'_i) + c_{t'_i,t'_k}\}\} + \tau_{k,q_k}. \quad (2)$$

$$M(S_{k-1}, q_k) = \max_{1 \le i \le k}\{FT(t'_i, q_i)\}. \quad (3)$$

Thus, $M(S_{k-1}, q_k)$ can be calculated incrementally as follows.

$$M(S_{k-1}, q_k) = \max\{M(S_{k-2}, q_{k-1}), FT(t'_k, q_k)\}. \quad (4)$$

The right part of Equation 1 can be defined incrementally as follows.

$$A(k+1, i, \alpha) = \sum_{j=k+1}^{v}(\frac{\tau_{j,i}}{2-\alpha} + \frac{\alpha \times \delta_j}{m-\alpha}) = A(k, i, \alpha) - \frac{\tau_{k,i}}{2-\alpha} - \frac{\alpha \times \delta_k}{m-\alpha}. \quad (5)$$

Thus, $E(\cdot)$ can also be computed incrementally as follows.

$$E(S_{k-1}, q_k, \alpha) = \max_{1 \le i \le m}\{M(S_{k-1}, q_k) + \frac{1}{m} \times A(k+1, i, \alpha)\}. \quad (6)$$

The pseudo code of our algorithm is presented in Algorithm 1, in which the task ranking is done by other list-based task scheduling algorithms. The algorithm is implemented recursively with the hill climbing scheme (depth-first search with estimation).

4 Experimental Results

We first compare the required time of our algorithm and obtained makespan with previous heuristic algorithms, HEFT and PEFT, for various numbers of tasks, various number of processor and various CCR values. The DAGs of our experiments are multi-stage graphs, in which the tasks in each stage (layer) are only connected to the tasks in the next stage (layer).

In our experiments, the parameters are set $v \in \{10, 20, 30, 40, 60, 80, 100\}$, $m \in \{2, 3, 5, 7, 10\}$, $CCR \in \{0.1, 0.5, 1, 5, 10\}$. The parameter of our algorithm is set $\alpha \in \{0, 0.2, 0.4, 0.6, 0.8, 1\}$. For each combination of parameters, we generate 10 DAGs. These algorithms are tested on several computers with 64-bit Windows 10 OS, CPU clock rate of 3.00GHZ (Intel(R) Core(TM) i5-9500 CPU) and RAM with 8 GB.

Our algorithm uses the HEFT ranking and the PEFT ranking as our task ranking. For a given instance of task schedule, suppose the makespans obtained by ours with HEFT ranking, ours with PEFT ranking, HEFT heuristic and PEFT heuristic are M_{oh}, M_{op}, M_h and M_p, respectively. Then the ratio to HEFT is $r_{oh} = \frac{M_{oh}}{M_h}$, the ratio to PEFT $r_{op} = \frac{M_{op}}{M_p}$, and average ratio to HEFT and PEFT is $r_{oa} = \frac{r_{oh}+r_{op}}{2}$. The makespan ratio and time ratio use the same notation.

Table 3: The average makepsan ratio on $v \in \{10, 20, 30, 40, 60\}$ and $m = 3$ for various CCR values.

CCR	Average makesapn ratio to HEFT & PEFT (r_{oa}), $m = 3$					
	$\alpha = 1$	$\alpha = 0.8$	$\alpha = 0.6$	$\alpha = 0.4$	$\alpha = 0.2$	$\alpha = 0$
0.1	0.944	0.937	0.932	0.924	0.921	0.921
0.5	0.952	0.934	0.924	0.918	0.913	0.913
1	0.947	0.922	0.901	0.886	0.880	0.880
5	0.946	0.904	0.856	0.830	0.820	0.819
10	0.965	0.928	0.871	0.828	0.821	0.820

Table 4: The average makepsan ratio on $v \in \{10, 20, 30\}$ and $m = 7$ for various CCR values.

CCR	Average makesapn ratio to HEFT & PEFT (r_{oa}), $m = 7$					
	$\alpha = 1$	$\alpha = 0.8$	$\alpha = 0.6$	$\alpha = 0.4$	$\alpha = 0.2$	$\alpha = 0$
0.1	0.974	0.965	0.950	0.942	0.939	0.939
0.5	0.962	0.953	0.939	0.930	0.927	0.927
1	0.944	0.932	0.912	0.899	0.893	0.893
5	0.908	0.883	0.864	0.852	0.849	0.849
10	0.897	0.881	0.861	0.849	0.843	0.843

Tables 3 and 4 show the average makespan ratio for $m = 3$ and $m = 7$, respectively. The average calculation in different m is obtained from different set of possible v values. As these tables show, when CCR is larger, our algorithm can reduce more makespan.

In Figures 2, 3 and 4, when the number of processors is small, the time required for our algorithm is within an acceptable range. With more processors, there are more branches in the searching tree, and it becomes more difficult to find the optimal solution, since the required time increases greatly. Cutting branches is more meaningful when there are more branches with more

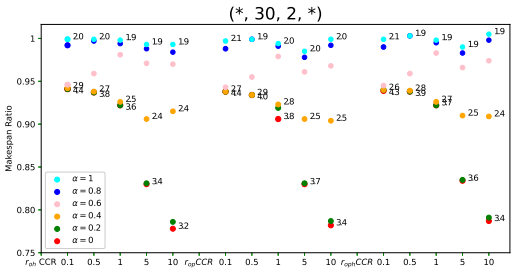

Figure 2: The experimental results for $v = 30$ and $m = 2$, where the number aside each circle represents the time ratio.

Algorithm 1 Heuristic branch and bound with hill climbing

Input: A set of tasks with ordering $T = t'_1, t'_2, \ldots, t'_v$, the DAG, the task cost table, an adjustment weight α.
Output: The near minimal makespan CM.

1: $k \leftarrow 0, M(S_0) \leftarrow 0, CM \leftarrow \infty$
2: $HILL(t'_k, k, S_{k-1}, q_k)\{$ ▷ At this moment, for $t'_1, t'_2, \ldots, t'_{k-1}$, each t'_h is assigned to processor q_h for $1 \leq h \leq k-1$ and $S_{k-1} = \langle q_1, q_2, \cdots, q_{k-1} \rangle$.
3: $S_k \leftarrow$ append q_k to S_{k-1} ▷ Assign task t'_k to processor q_k
4: **for** $i = 1$ to m **do** ▷ Evaluate next lower level $k+1$
5: $B[i].E \leftarrow E(S_k, i, \alpha)$ ▷ by Equation 6
6: $B[i].proc \leftarrow i$ ▷ Try to assign task t'_{k+1} to processor i
7: **end for**
8: Sort $B[\,]$ with E values increasingly
9: **if** $k+1 = v$ **then** ▷ The next lower level is a leaf
10: $CM \leftarrow B[1].E$ ▷ All tasks have been assigned, and $B[1]$ stores the minimum
11: **return**
12: **end if**
13: **for** $i = 1$ to m **do**
14: **if** $B[i].E < CM$ **then** ▷ May have a better solution in next lower level
15: $HILL(t'_{k+1}, k+1, S_k, B[i].proc)$ ▷ Expand next lower level
16: **else**
17: **break** ▷ This and the remaining branches are pruned away
18: **end if**
19: **end for**
20: **return** ▷ All branches have been expanded or pruned away
21: }

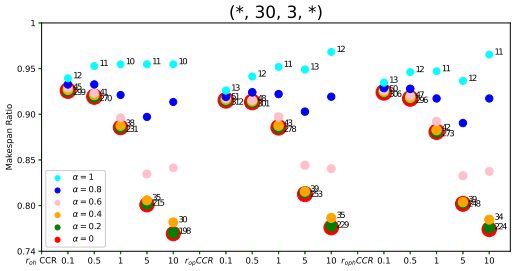

Figure 3: The experimental results for $v = 30$ and $m = 3$.

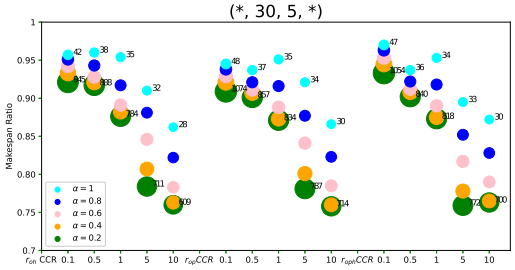

Figure 4: The experimental results for $v = 30$ and $m = 5$.

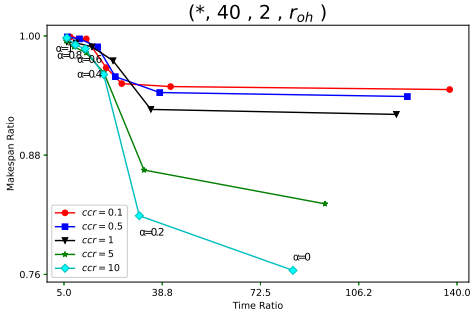

Figure 5: The experimental results for $v = 40$, $m = 2$ and $\alpha \in \{0, 0.2, 0.4, 0.6, 0.8, 1\}$.

processors.

In Figures 5, the makespan is significantly reduced when α is changed from 1 to smaller values. When α is close to 0, it takes much more time to shorten only a little makespan. When CCR is large (large communication cost), our algorithm can get more improvement, since the communication cost cannot be ignored and our algorithm tries more possible solutions than aforementioned heuristic algorithms. If the number of tasks or processors is large, there will be more choices or branches in the searching tree. The parameter α can be adapted to find the solution with the desired quality. As shown in Figures 5, the makespan is reduced about 20% for large CCR.

5 Conclusion

In this paper, for given a task ordering (provided by HEFT or PEFT ranking), we propose a heuristic branch and bound method for assigning tasks to processors in the task allocation stage. Then, we compare the makespans obtained by HEFT and PEFT with our algorithm.

As the experimental results show, our algorithm is able to offer solutions with different quality based on parameter setting, and to quickly find the optimal solution if the number of tasks is small. According to different $CCRs$ and different numbers of processors, our algorithm also illustrates its advantages. In the case of large CCR (such as 10), as in Figure 5, our algorithm can shorten about 20% of the makespan compared with HEFT and PEFT.

In the future, we hope to find all possible task execution sequences and to put them in the different processor to find better solutions. We may also try to modify our equation for estimating the communication time, which may also be helpful for the choice of branches.

References

[1] Ewa Deelman, Gurmeet Singh, Mei-Hui Su, James Blythe, Yolanda Gil, Carl Kesselman, Gaurang Mehta, Karan Vahi, G Bruce Berriman, John Good, et al. Pegasus: A framework for mapping complex scientific workflows onto distributed systems. *Scientific Programming*, 13(3):219–237, 2005.

[2] Arun Ramakrishnan, Gurmeet Singh, Henan Zhao, Ewa Deelman, Rizos Sakellariou, Karan Vahi, Kent Blackburn, David Meyers, and Michael Samidi. Scheduling data-intensiveworkflows onto storage-constrained distributed resources. In *Proceedings of the 7th IEEE International Symposium on Cluster Computing and the Grid (CCGrid'07)*, pages 401–409. Rio de Janeiro, Brazil, 2007.

[3] Philip Maechling, Ewa Deelman, Li Zhao, Robert Graves, Gaurang Mehta, Nitin Gupta, John Mehringer, Carl Kesselman, Scott Callaghan, and David Okaya. SCEC cybershake workflow-automating probabilistic seismic hazard analysis calculations. In *Workflows for e-Science*, pages 143–163. Springer, 2007.

[4] Oliver Sinnen. *Task scheduling for parallel systems*, volume 60. John Wiley & Sons, 2007.

[5] Haluk Topcuoglu, Salim Hariri, and Min-You Wu. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Transactions on Parallel and Distributed Systems*, 13(3):260–274, 2002.

[6] Haluk Topcuoglu, Salim Hariri, and Min-You Wu. Task scheduling algorithms for heterogeneous processors. In *Proceedings of the 8th Heterogeneous Computing Workshop (HCW'99)*, pages 3–14. San Juan, PR, USA, 1999.

[7] Hamid Arabnejad and Jorge G Barbosa. List scheduling algorithm for heterogeneous systems by an optimistic cost table. *IEEE Transactions on Parallel and Distributed Systems*, 25(3):682–694, 2014.

[8] Luiz F Bittencourt, Rizos Sakellariou, and Edmundo RM Madeira. Dag scheduling using a lookahead variant of the heterogeneous earliest finish time algorithm. In *Proceedings of the 18th Euromicro conference on parallel, distributed and network-based processing*, pages 27–34. Pisa, Italy, 2010.

[9] Hesham El-Rewini and Ted G. Lewis. Scheduling parallel program tasks onto arbitrary target machines. *Journal of Parallel and Distributed Computing*, 9(2):138–153, 1990.

[10] Michael A Iverson, Füsun Özgüner, and Gregory J Follen. Parallelizing existing applications in a distributed heterogeneous environment. In *Proceedings of the 4th Heterogeneous Computing Workshop (HCW'95)*, pages 1–8, 1995.

[11] E Ilavarasan, P Thambidurai, and R Mahilmannan. High performance task scheduling algorithm for heterogeneous computing system. In *Proceedings of the International Conference on Algorithms and Architectures for Parallel Processing*, pages 193–203. Berlin, Germany, 2005.

[12] E Ilavarasan and P Thambidurai. Low complexity performance effective task scheduling algorithm for heterogeneous computing environments. *Journal of Computer Sciences*, 3(2):94–103, 2007.

[13] Sarad Venugopalan and Oliver Sinnen. ILP formulations for optimal task scheduling with communication delays on parallel systems. *IEEE Transactions on Parallel and Distributed Systems*, 26(1):142–151, 2015.