An Experimental Investigation into the Rank Function of the Heterogeneous Earliest Finish Time Scheduling Algorithm

Henan Zhao and Rizos Sakellariou

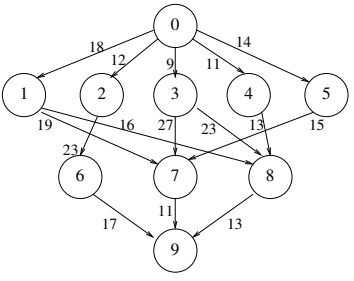
Department of Computer Science, University of Manchester, U.K.

Abstract. This paper considers the Heterogeneous Earliest Finish Time (HEFT) algorithm for scheduling the tasks of an application, represented by a directed acyclic graph, onto a bounded number of heterogeneous machines. We focus on the appropriate selection of the weight for the nodes and edges of the graph, and experiment with a number of different schemes for computing these weights. Our findings indicate that the length of the schedule produced may be affected significantly by the scheme used, and suggest that the mean value based approach used by HEFT may not be a particularly good choice.

1 Introduction

Among the scheduling algorithms for heterogeneous machines, the *Heterogeneous Earliest Finish Time* (HEFT) algorithm [3], a natural extension of the classical list scheduling algorithm for homogeneous systems to cope with heterogeneity, has been shown to produce shorter schedule lengths more often than other comparable algorithms. In classical list scheduling algorithms an application is viewed as a *directed acyclic graph* (DAG), where nodes (or tasks) represent computation and edges represent communication. By assigning a weight to each node (typically, the corresponding computation cost) and edge (typically, the corresponding communication cost), those algorithms prioritize the nodes to be scheduled on the basis of a value computed by a *rank function*; this is typically a function of the weights assigned to the nodes and edges of the graph. However, in a heterogeneous setting, the values typically used for the weights cannot be considered as constant any more: the computation cost of a task may vary, depending on the machine that this would run on; same, the communication cost may vary, depending on which machines are communicating. Although it has long been known, in homogeneous environments, that the choice of the rank function (and the values it returns) may affect the quality of the schedule produced, the proposers of HEFT have not examined the different (and additional with regard to homogeneous environments) possibilities that exist for computing the weights in a heterogeneous environment; they opted for the intuitively appealing mean values. In this paper, we consider a number of different options for computing the weights in HEFT. We find that, on average, the mean value option is not necessarily the most efficient choice, but, more importantly, the length of the schedules produced may differ significantly from one option to another.

H. Kosch, L. Böszörményi, H. Hellwagner (Eds.): Euro-Par 2003, LNCS 2790, pp. 189–194, 2003.



task	M0	M1	M2	task	M0	M1	M2
0	37	39	27	5	29	37	20
1	30	20	24	6	22	24	30
2	21	21	28	7	37	26	37
3	35	38	31	8	35	31	26
4	27	24	29	9	33	37	21

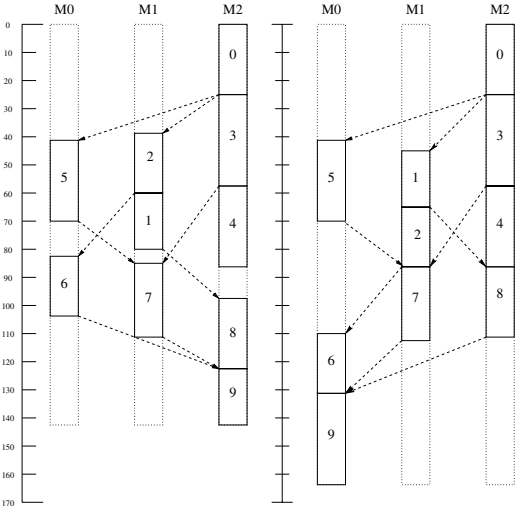


Fig. 1. A Motivating Example: Two different schedules for two different rank schemes.

2 Background and Motivation

The HEFT algorithm [3] works as follows. First, a weight is assigned to each node and edge of the graph, based on the average computation and communication, respectively. Then, the graph is traversed upwards and a rank value is assigned to each node; this is based on the sum of the weight of the node with the maximum value resulting from all possible summations that add the weight of an edge to an immediate successor node with the rank value of that successor node. Tasks are then scheduled, in order of their rank value, on the machine which gives the earliest finish time. A task may be scheduled in an idle slot between two already scheduled tasks on a machine as long as precedence constraints are preserved.

The use of the average computation and communication as weights in the graph has been an *ad hoc* choice in the HEFT algorithm. However, there are cases where the average value may not produce a good schedule. To illustrate this consider the graph shown in Figure 1 (example adopted from [3]). The communication cost between two nodes in the graph is given by the number next to each edge, except when the two nodes will run on the same processor, in which case it is zero. The computation cost of each task on three different machines is given by the table. If the nodes are prioritized using as a weight the mean values of the computation cost over all three machines (as in the original HEFT), then the schedule produced has a length equal to 164 (right-hand side of the figure). If, on the other hand, nodes are prioritized using as a weight the worst (i.e., maximum) values of the computation cost (over all three machines on which each node may run), the schedule produced has a length equal to 143. This significant improvement is the result of only a small difference in the priority order of the nodes; in the first case, the order is $\{0, 3, 5, 1, 2, 4, 7, 8, 6, 9\}$, whereas, in the second case, the order is $\{0, 3, 5, 2, 1, 4, 7, 8, 6, 9\}$. The question that motivated

this work is what is the behaviour of the mean value as an option to compute the weights over a large number of cases, and whether different schemes to compute the weights can lead to significant variations in the performance of HEFT.

3 Methodology

We have developed a program that implements the HEFT algorithm, allowing for different options for task prioritization. Its inputs are: a DAG representing an application; the number of machines of the heterogeneous platform, M; the computation cost to execute each task of the DAG on each machine; the data transfer rate between each machine; and, the amount of data required to be transmitted from one task to another if these two tasks run on different machines.

When prioritizing tasks, we have considered both upward and downward ranking. Thus, the upward rank, $r_u(i)$, of a task i is recursively defined by

$$r_u(i) = f_1(w_i^0, ..., w_i^m, ..., w_i^{M-1}) + \max_{\forall j \in S_i} (f_2(c_{ij}^{00}, ..., c_{ij}^{mm'}, ..., c_{ij}^{M-1,M-1}) + r_u(j)),$$

where w_i^m is the computation cost of task i on machine m, $0 \leq m < M$, S_i is the set of the immediate successors of task i, and $c_{ij}^{mm'}$ is the communication cost between nodes i and j when i is executed by machine m and j by machine m', $0 \leq m, m' < M$. The communication cost is derived by dividing the amount of data required to be transmitted between the tasks i and j by the data transfer rate between the two machines m and m'. It is assumed that when i and j are executed by the same machine (i.e., $m = m'$), the communication cost is zero. Furthermore, the function f_1 returns a value which is dependent on the computation cost of a given task on every machine, and the function f_2 returns a value which is dependent on the communication cost between two given tasks considering every combination of machines where the two given tasks may execute. Similarly, the downward rank, $r_d(i)$, of a task i is recursively defined by

$$r_d(i) = \max_{\forall j \in P_i} (f_1(w_j^0, ..., w_j^m, ..., w_j^{M-1}) + f_2(c_{ji}^{00}, ..., c_{ji}^{mm'}, ..., c_{ji}^{M-1,M-1}) + r_d(j)),$$

where P_i is the set of the immediate predecessors of task i.

Six approaches are used to compute a value for the functions f_1, f_2: (i) *Mean value* (denoted by the shorthand M) returns the average over all the input arguments; that is, f_1 returns the average computation cost of a task and f_2 returns the average communication cost between two tasks (this is the approach used in the original HEFT). (ii) *Median value* (ME) returns the median value over all the input arguments for both f_1 and f_2. (iii) *Worst value* (W) returns the worst value (i.e., maximum computation cost) for f_1 and, for f_2, the communication cost between the two machines on which each of the two communicating tasks has its highest computation cost. For example, in Figure 1, task 3 has its highest computation cost on machine M1 and task 8 has its highest computation cost on machine M0; then, the value returned by f_2 is specified by the amount of data transmitted between nodes 3 and 8 and the data transfer rate between machines

M0 and M1, i.e., c_{38}^{10} using the notation of the previous paragraph. (iv) *Best value* (B) returns the best value (i.e., minimum computation cost) for f_1 and, for f_2, the communication cost as determined by the procedure described previously. (v)*Simple Worst value* (SW) returns the worst value for both f_1 and f_2 (i.e., maximum computation cost and maximum communication cost, respectively). (Recall that this approach was used to prioritize the nodes when the schedule of length equal to 143 was produced in the motivating example of Section 2) (vi) *Simple Best value* (SB) returns the best value for both f_1 and f_2 (i.e., minimum computation cost and minimum communication cost, respectively).

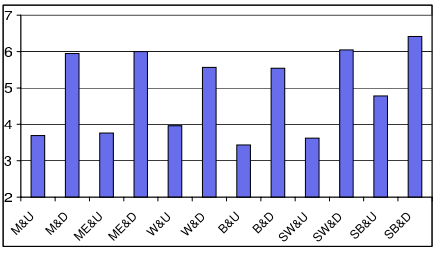
Since we use both upward and downward ranking with each of the above approaches, a total of $2 \times 6 = 12$ different schemes are considered.

4 Experimental Results and Discussion

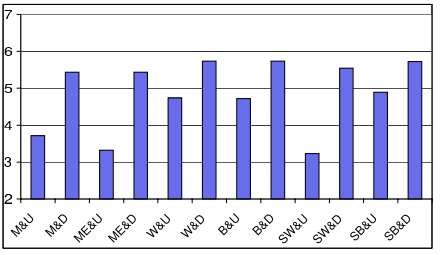
We compared the performance of the 12 different schemes in four sets of experiments using four metrics [1]. The *Average percentage degradation from the best* (*APD*) is the average (over all cases of a set of experiments) of the percentage of degradation of the schedule length of a particular scheme from the scheme returning the best solution. The *Number of best solutions* (*NB*) refers to the number of times a particular scheme was the *only one* that produced the shortest schedule. The *number of best solutions equal with another scheme* (*NEB*) counts those cases where a scheme produced the shortest schedule length but at least one scheme more achieved the same length. Finally, the *worst percentage degradation from the best* (*WPD*), is the maximum percentage degradation from the best of a given scheme over all cases of a particular set of experiments.

Two classes of DAGs were used in the experiments. In the first class, we randomly generate graphs having between 25 and 100 nodes as follows. Each graph has a single entry and a single exit node; all other nodes are divided into levels. Each level is created progressively and has a random number of nodes, which varies from two to half the number of the remaining (to be generated) nodes. In the second class of graphs, we used task graphs representing a Laplace equation solver, forms of which have been used in other related studies [2]. The versions considered in our experiments have between 25 and 400 nodes.

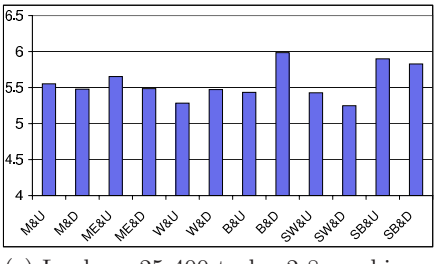
Two different approaches were used to specify the degree of heterogeneity of the machines. The first, referred by the name *random computation cost*, is similar to the approach used in [3], in that the computation cost for a given task and machine is selected randomly from a uniform distribution within a given range. The second approach attempts to take account of the observation that, in a heterogeneous environment, the factors affecting the execution of a given task on a given machine are related primarily to particular characteristics (e.g., CPU power) of the machine concerned. In this approach, a random number between 0.5 and 1 is first generated as a factor indicating the computational power of each machine. Then, the cost for a given task on a given machine is within 5% of the product of this number and a baseline computation cost for each task (selected randomly). This approach is referred by the name *proportional computation cost*.



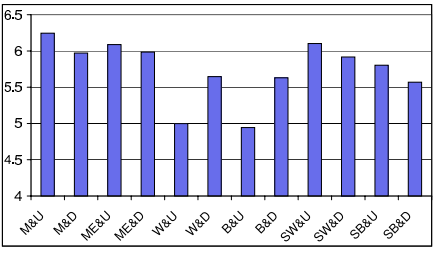
(a) Random DAGs, 25-100 tasks, 2-8 machines, random comp. cost



(b) Random DAGs, 25-100 tasks, 2-8 machines, proportional comp. cost



(c) Laplace, 25-400 tasks, 2-8 machines, random comp. cost



(d) Laplace, 25-400 tasks, 2-8 machines, proportional comp. cost

Fig. 2. Average percentage degradation from the best; 4 different sets of experiments.

All combinations above (2 families of graphs × 2 approaches for heterogeneity) lead to 4 different sets of experiments. For each set, we vary the task graph granularity so that a communication to computation ratio [1] between 0 and 4 is achieved; the number of available machines is also varied between 2 and 8. We consider a total of 2000 cases in each set of experiments.

The *APD* for each of the 12 schemes and for each set of experiments is shown in Figure 2. The shorthand denoting each scheme is a concatenation of: the shorthand of the approach used to compute the weights (see Section 3) and the suffix '&U' or '&D' depending on whether ranking is performed upward or downward, respectively. From the figure, the best 6 schemes in each set (starting with the best) are: (a) {B&U, SW&U, M&U, ME&U, W&U, SB&U}; (b) {SW&U, ME&U, M&U, B&U, W&U, SB&U}; (c) {SW&D, W&U, SW&U, B&U, W&D, M&D}; (d) {B&U, W&U, SB&D, B&D, W&D, SB&U}. Mean value schemes are outperformed in every experiment. In (a), M&U is worse than B&U by 7.6%; in (b), M&U is worse than SW&U by 14.8%; in (c), M&D is worse than SW&D by 4.4%; and, in (d), M&D is worse than B&U by 20.8%, while M&U is worst amongst all schemes. There is no clear pattern favouring one scheme; however, we notice that B&U is always among the best 4, and W&U is always among the best 5. Another observation is that upward schemes always outperform downward schemes for randomly generated DAGs, but not for Laplace.

Table 1 shows the values of the remaining three metrics, *NB*, *NEB*, *WPD*. On the basis of the value $NB + NEB$, the only scheme which is consistently

Table 1. *NB*, *NEB*, and *WPD* for each set of experiments in Figure 1.

	Experiment (a)			Experiment (b)			Experiment (c)			Experiment (d)		
	NB	*NEB*	*WPD*	*NB*	*NEB*	*WPD*	*NB*	*NEB*	*WPD*	*NB*	*NEB*	*WPD*
M&U	236	10	25.0	233	97	36.0	157	21	34.8	112	27	47.2
M&D	64	22	27.7	149	79	33.7	139	30	33.4	87	31	35.6
ME&U	186	72	26.4	200	185	21.1	133	13	34.8	121	31	32.4
ME&D	35	47	26.2	65	144	33.7	138	28	34.0	94	34	41.4
W&U	242	5	25.8	161	67	39.0	207	3	29.8	263	2	38.9
W&D	116	3	23.5	69	131	36.6	180	2	30.4	181	0	42.1
B&U	369	1	27.6	59	183	39.0	165	1	36.8	273	2	33.8
B&D	109	0	22.0	21	178	31.0	145	4	36.5	171	1	30.3
SW&U	242	72	26.4	225	193	21.1	167	17	34.8	117	24	32.0
SW&D	37	43	26.2	40	152	33.7	153	22	31.0	117	31	35.6
SB&U	159	5	29.2	19	189	39.0	190	0	34.4	191	1	42.2
SB&D	68	1	32.5	6	183	31.0	164	1	36.3	195	2	34.3

among the best 5 is W&U. Although the *WPD* metric, refers to a single bad case (one out of 2000), it is interesting to notice that the performance of any scheme may deviate from that of the best scheme significantly. At one extreme, M&U performed 47.2% worse than the best scheme in the case.

The main observations from the results presented can be summarized as follows: (i) There are significant differences between the performance of HEFT depending on the scheme used for computing the weights. (ii) There is no evidence to suggest that the use of mean values should be preferred; instead, our results seem to indicate the opposite. (iii) Upward ranking appears to perform in many cases, but not always, better than downward ranking. (iv) Based on the relative ranking of the performance of B&U and W&U with respect to other schemes, there is some evidence in favour of these two schemes.

Thus, the performance of HEFT can be improved by checking the schedule produced by each scheme (or some of them) and taking the best. This would increase the cost of the algorithm, but it may be a trade-off worth making. Future research could focus on ways to improve the scheduling process by making it less sensitive to the different schemes for ranking nodes.

References

1. Y.-K. Kwok and I. Ahmad. Benchmarking and Comparison of the Task Graph Scheduling Algorithms. *Journal of Parallel and Distributed Computing*, 59, pp. 381–422, 1999.
2. A. Radulescu and A.J.C. van Gemund. Fast and Effective Task Scheduling in Heterogeneous Systems. *9th Heterogeneous Computing Workshop*, pp. 229–238, 2000.
3. H. Topcuoglu, S. Hariri, and M.-Y. Wu. Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Computing. *IEEE Transactions on Parallel and Distributed Systems*, 13(3), pp. 260–274, March 2002.