Critical path estimation in heterogeneous scheduling heuristics

Thomas McSweeney[*]

20th August 2020

1 Introduction

Recall from previous chapters that the *Heterogeneous Earliest Finish Time* (HEFT) heuristic prioritizes all tasks t_i, $i = 1, \ldots, n$, by recursively computing a corresponding sequence of numbers u_i which are intended to represent *critical path* lengths from each task to the end. As noted previously, however, the concept of the critical path is not clearly defined for heterogeneous target platforms: there are multiple possible values that each of the DAG weights may take so there are also many different ways we could define a *longest* (i.e., costliest) path. Consider for example the simple DAG shown in Figure 1, where the labels represent all the possible weights each task/edge may take on a two-processor target platform; specifically, the labels (W_i^1, W_i^2) near the nodes represent the computation costs on processors $P1$ and $P2$, respectively, while the edge labels $(0 = W_{ik}^{11} = W_{ik}^{22}, W_{ik}^{12}, W_{ik}^{21})$ represent the possible communication costs.

How should the longest path through a graph like that in Figure 1 be defined? The HEFT approach, as described in previous chapters, is to use mean values over all sets of possible costs to fix the DAG weights and then compute the critical path lengths in a standard dynamic programming manner. For the example, this gives

$$u_6 = 2.5,\ u_5 = 7,\ u_4 = 7.75,\ u_3 = 16,$$
$$u_2 = 5.75,\ u_1 = 16.25,\ \text{and}\ u_0 = 24.75,$$

giving a scheduling priority list of $\{0, 1, 3, 4, 5, 2, 6\}$. The schedule length obtained by HEFT with these priorities is 22, so note in particular that u_0, the rank of the single entry task, is not a lower bound on this value, unlike in the homogeneous case. The question then is, what quantity do the u_i values actually represent?

[*]School of Mathematics, University of Manchester, Manchester, M13 9PL, England (`thomas.mcsweeney@postgrad.manchester.ac.uk`).

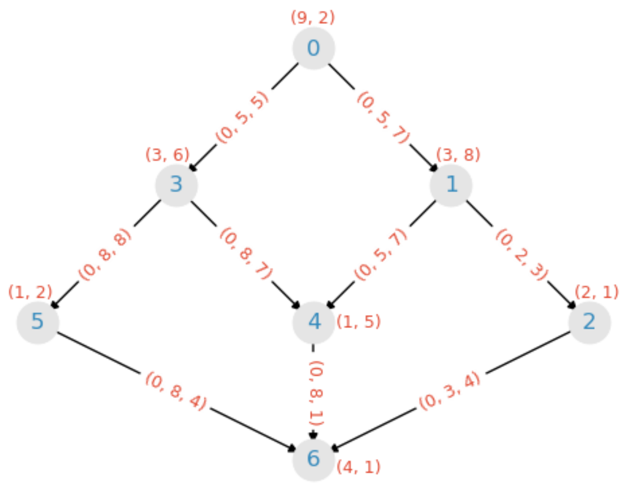


Figure 1: Simple task graph with costs on a two-processor target platform.

The most intuitive interpretation is perhaps that the HEFT ranks are estimates of what the critical paths are *likely* to be but there is no robust mathematical justification for believing that such a definition is truly the most useful.

Given this ambiguity, alternative ways to define the critical path in HEFT have been considered before, most notably by Zhao and Sakellariou [11], who empirically compared the performance of HEFT when averages other than the mean (e.g., median, maximum, minimum) are used to compute upward (or downward) ranks. Their conclusions were that using the mean is not clearly superior to other averages, although none of the other options were consistently better. Indeed, perhaps the biggest takeaway from their investigation was that HEFT is very sensitive to how priorities are computed, with significant variation being seen for different graphs and target platforms. In this chapter we undertake a similar investigation with the aim of establishing if there are choices which do consistently outperform the standard HEFT task ranking phase.

This will be an empirically-driven study, as is common in this area. To facilitate this investigation we created a software package that simulates heterogeneous scheduling problems, much like that described in the previous chapter, although not restricted to accelerated target platforms. As before, the (`Python`) source code for this simulator can be found on Github[1] and all of the results presented here can be re-run from scripts contained therein.

[1]https://github.com/mcsweeney90/critical-path-estimation

2 Optimistic bounds

Functionally, the critical path is used in HEFT as a lower bound on the makespan, so that minimizing the critical path gives us the most scope to minimize the makespan (assuming we make good use of our parallel resources). With this in mind, there are many different ways we can define the critical path so that it gives a lower bound on the makespan of any possible schedule. The most straightforward approach would be to just set all weights to their minimal values but a tighter bound can be computed in the following manner. First, define ℓ_i^a for all tasks t_i and processors p_a to be the critical path length from t_i to the end (inclusive), assuming that it is scheduled on processor p_a. These values can easily be computed recursively by setting $\ell_i^a = W_i^a$, $a = 1, \ldots, q$, for all exit tasks then moving up the DAG and setting

$$\ell_i^a = W_i^a + \max_{k \in S_i} \left(\min_{b=1,\ldots,q} \left(\ell_k^b + W_{ik}^{ab} \right) \right) \quad \forall a \tag{1}$$

for all other tasks. Then for each $i = 1, \ldots, n$,

$$\ell_i = \min_{a=1,\ldots,q} \ell_i^a \tag{2}$$

gives a true lower bound on the remaining cost of any schedule once the execution of task t_i begins. These ℓ_i values could be useful as alternative task priorities in HEFT, especially since the cost of computing all of the ℓ_i^a in this manner is only $O((m+n)q) \approx O(n^2 q)$ so in particular is the same order as the usual HEFT prioritization phase. For example, for the simple graph shown in Figure 1, we find that the ℓ_i values are as given in Table 1 (with the u_i included for comparison). Interestingly, we see that tasks 1 and 3 have the same optimistic rank (8 units) and the performance of the alternative ranks relative to the standard u_i sequence in HEFT depends on which is chosen to be scheduled first; if task 1, the priority list does not change so the schedule makespan is 22 units, but if task 3 is selected instead, the final schedule makespan is 20 units—i.e., smaller than the original.

Of course, this is only one example: it should be emphasized here that there is absolutely no mathematically valid reason to suppose that using the ℓ_i sequence instead of u_i as the task ranks in HEFT will actually lead to superior performance in general. Still, it seems worthwhile to investigate this empirically using our simulator, which we do in Section 4.

(Note that the optimistic critical path defined here is extremely similar to the optimistic cost used in the PEFT heuristic; this will be discussed further in Section 5.)

3 Stochastic interpretation

In this section we propose a family of alternative task ranking phases in HEFT based on the following interpretation of the standard ranking phase. First, note

Table 1: Upward and optimistic ranks for the DAG in Figure 1.

Task	Upward rank	Optimistic rank
0	24.75	16
1	16.25	8
2	5.75	2
3	16	8
4	7.75	5
5	7	3
6	2.5	1

that by using average values over all sets of possible task and edge costs, HEFT is implicitly assuming that any member of any set is just as likely to be incurred as any other; conceptually, HEFT is attempting to account for the uncertainty of the processor selection phase by assuming that for any given task all processors are equally likely to ultimately be chosen. So, effectively, at the prioritization phase HEFT views the node and edge weights as independent discrete random variables (RVs) with associated probability mass functions (pmfs) given by the aforementioned assumption. More precisely, let m_i be the pmf corresponding to the task weight variable w_i and m_{ik} that for the edge weight w_{ik}, then

$$m_i(W_i^a) := \mathbb{P}[w_i = W_i^a] = \frac{1}{q}, \quad a = 1, \dots, q,$$

and

$$\begin{aligned} m_{ik}(W_{ik}^{ab}) &= m_i(W_i^a) \cdot m_k(W_k^b) \\ &= \frac{1}{q^2}, \quad \forall a, b. \end{aligned}$$

Note that the expected values of the node and edge weights are therefore given by

$$\mathbb{E}[w_i] = \sum_{\ell \in L_i} \ell m_i(\ell) = \frac{1}{q} \sum_a W_i^a \tag{3}$$

$$\mathbb{E}[w_{ik}] = \sum_{\ell \in L_{ik}} \ell m_{ik}(\ell) = \frac{1}{q^2} \sum_{a,b} W_{ik}^{ab}. \tag{4}$$

In particular, this means that $\mathbb{E}[w_i] = \overline{w_i}$ and $\mathbb{E}[w_{ik}] = \overline{w_{ik}}$. So the computation of the upward ranks u_i in HEFT can instead be done by setting $u_i = \mathbb{E}[w_i]$ for all exit tasks, then moving up the DAG and recursively computing

$$u_i = \mathbb{E}[w_i] + \max_{k \in S_i} \big(u_k + \mathbb{E}[w_{ik}]\big) \tag{5}$$

for all other tasks.

In summary, since all possible node and edge weights are known but their actual values at runtime aren't (at least without restricting the processor selection phase), HEFT estimates critical path lengths from all tasks in a task graph G through a two-step process:

1. An associated graph G_s—referred to as *stochastic* because all of its weights are RVs—is implicitly constructed with node and edge pmfs m_i and m_{ik} as defined above.

2. The numbers u_i are recursively computed for all tasks in G_s using (5), and taken as the critical path lengths from the corresponding tasks in G.

In the following two sections, we propose modifications of both steps so as to obtain different critical path estimates that may be used as task ranks in HEFT. The performance of these will then be evaluated through extensive numerical simulations in Section 4.

3.1 The critical path of G_s

A natural question arises from the interpretation outlined in the previous section: what is the relationship between the sequence of numbers u_i as defined by (5) and the critical path of the stochastic graph G_s? (Of course, since all of the weights are RVs, the critical path of G_s is itself stochastic.) In fact, it has long been known in the context of *Program Evaluation and Review Technique* (PERT) network analysis that the numbers u_i are *lower bounds* on the expected value of the critical path lengths of the stochastic DAG. This result dates back at least as far as Fulkerson [5], who referred to it as already being widely-known and gave a simple proof. This prompts another question: does using the actual expected values of the critical path lengths as the task priories in HEFT lead to superior performance?

Unfortunately, computing the moments of the critical path length of a graph whose weights are discrete RVs was shown to be a $\#P$-complete problem by Hagstrom [6]. This means that it is generally impractical to compute the true expected values. However, efficient methods which yield better approximations than the u_i numbers are known; we discuss examples in the following two sections.

3.1.1 Monte Carlo sampling

Monte Carlo (MC) methods have a long history as a means of approximating the longest path distribution for PERT networks, dating back to at least the early 1960s [10]. The idea is to simulate the realization of all RVs (according to their pmfs) and then evaluate the critical path of the resulting deterministic graph. This is done repeatedly, giving a set of critical path instances whose

Table 2: Monte Carlo estimates of critical path lengths for example graph.

Task	u_i	$MC1$	$MC10$	$MC100$	$MC1000$
0	24.75	29	26.8	29.9	28.9
1	16.25	22	17.0	17.4	17.0
2	5.75	3	5.4	6.0	5.8
3	16	20	18.9	19.0	18.6
4	7.75	14	7.2	7.9	7.8
5	7	6	5.3	7.1	7.0
6	2.5	1	1.9	2.5	2.5

empirical distribution function is guaranteed to converge to the true distribution by the Glivenko-Cantelli Theorem [2]. Furthermore, analytical results allow us to quantify the approximation error for any given the number of realizations—and therefore the number of realizations needed to reach a desired accuracy.

Table 2 illustrates how our estimate of the expected critical path lengths of the stochastic graph in Figure 1 evolve as the number of realizations increases; the corresponding u_i numbers are included as well to show that they do indeed appear to be a lower bound on the values that the Monte Carlo method appear to be converging toward. Note that the critical path length estimate for task 3 eventually exceeds that of task 1 so it has a higher priority if these estimates are taken as task ranks in HEFT; as noted in the previous section, interchanging these two tasks—and keeping the same ordering for all others—leads to a smaller schedule makespan (20 units) compared to the standard ranking (22 units). Of course, this is only one example, but it does serve to illustrate that in some cases a tighter bound on the critical path of G_s can be useful.

The downside of Monte Carlo sampling is its cost. While modern architectures are well-suited to this approach because of their parallelism, this approach still may be impractical in the context of a scheduling heuristic; we often found this to be the case in the examples discussed in Section 4. Hence in this report we typically only use the Monte Carlo method as a means of obtaining a reference solution; see, for example, the following section.

3.1.2 Fulkerson's bound

For all $i = 1, \dots, n$, let c_i be the critical path length from task t_i to the end and let $e_i = \mathbb{E}[c_i]$ be its expected value. In addition to proving that the u_i sequence defines lower bounds on the critical path lengths, i.e., $u_i \leq e_i \; \forall i$, Fulkerson also showed how an alternative sequence of numbers which give tighter bounds can easily be constructed.

First we assume that G_s is expressed in an equivalent formulation without node weights. The most straightforward way to do this is to simply redefine the edge weights so that they also include the computation cost of the parent task and, if the child task is an exit, the computation cost of the child as well. More precisely, we define a new set of possible edge weights $\tilde{L}_{ik}$ by

$$\tilde{L}_{ik} = \{\tilde{W}_{ik}^{ab} := W_i^a + \delta_k W_k^b + W_{ik}^{ab}\}_{a,b=1,\dots,q},$$

where $\delta_k = 1$ if t_k is an exit task and zero otherwise. We also define a new edge weight variable $\tilde{w}_{ik} \in \tilde{L}_{ik}$ and new edge pmfs $\tilde{m}_{ik}$ for which $\tilde{m}(\tilde{W}_{ik}^{ab}) \equiv m(W_{ik}^{ab})$. Note that removing the node weights is not strictly necessary but simply makes the elucidation cleaner so we should emphasize that all of the following still holds, with only minor adjustments, if this is not done.

Now, for $i = 1, \dots, n$, define Z_i to be the set of all weight RVs corresponding to edges downward of t_i (i.e., the remainder of the graph). Let R_i be the set of all possible *realizations* of the RVs in Z_i. Given a realization $z_i \in R_i$, let $\ell(z_i)$ be the critical path length from task t_i to the end. Then by the definition of the expected value we have

$$e_i = \sum_{z_i \in R_i} \mathbb{P}[Z_i = z_i]\ell(z_i). \tag{6}$$

Let $C_i := \{w_{ik}\}_{k \in S_i}$ be the set of all the weight RVs corresponding to edges which connect task t_i to its children. Note that

$$Z_i = C_i \cup_{k \in S_i} Z_k$$

and any realization $z_i \in R_i$ can therefore be expressed as $z_i = c_i \cup_{k \in S_i} z_k^i$, where $z_k^i \in R_k$ and $c_i = \{z_{ik}^i\}_{k \in S_i}$ is the set of realizations of the edge weight RVs in C_i. In particular, this means that we can write

$$\ell(z_i) = \max_{k \in S_i}\{\ell(z_k^i) + z_{ik}^i\}.$$

Furthermore, by the independence assumptions made, we have that

$$\mathbb{P}[Z_i = z_i] = \mathbb{P}[C_i = c_i] \prod_{k \in S_i} \mathbb{P}[Z_k = z_i^k].$$

This means that we can rewrite equation (6) as

$$\begin{aligned} e_i &= \sum_{z_i \in R_i} \mathbb{P}[Z_i = z_i]\ell(z_i) \\ &= \sum_{c_i} \sum_{\substack{z_k \in R_k, \\ k \in S_i}} \mathbb{P}[C_i = c_i]\mathbb{P}[Z_k = z_k] \max_{k \in S_i}\{\ell(z_k) + z_{ik}\}, \end{aligned} \tag{7}$$

where z_{ik} is the realization of the edge weight RV w_{ik} defined by the set of realizations z_k.

It is relatively straightforward to show that the identity $u_i \leq e_i$ holds by manipulating equation (7); the reader is directed to Fulkerson's original paper for details [5]. Moreover, suppose we define a sequence of numbers by $f_i = 0$, if t_i is an exit task, and

$$\begin{aligned} f_i &= \sum_{z_i \in R_i} \mathbb{P}[Z_i = z_i] \max_{k \in S_i} \{f_k + z_{ik}\} \\ &= \sum_{c_i} \sum_{\substack{z_k \in R_k, \\ k \in S_i}} \mathbb{P}[C_i = c_i] \mathbb{P}[Z_k = z_k] \max_{k \in S_i} \{f_k + z_{ik}\}, \end{aligned} \tag{8}$$

for all other tasks, then Fulkerson showed that $u_i \leq f_i \leq e_i$ also holds—i.e, the f_i give a tighter bound on the expected values of the critical path lengths.

To compute each of the f_i using (8) we need to do an awful lot of work: suppose t_i has K children, then for any one of them t_k we need to do $O(|\tilde{L}_{ik}|)$ operations, i.e., $O(|\tilde{L}_{ik}|^K)$ in total. In general, $|\tilde{L}_{ik}|$ can be $O(p^2)$ and K can be $O(n^2)$ so computing the f_i in this manner is not always practical. Fortunately, a more efficient method was given by Clingen [3] in the context of extending Fulkerson's method to the case where edge weights are modeled as continuous random variables, although here we follow the slightly more compact notation of Elmaghraby [4].

It is well-known that the cumulative probability mass function of the maximum of a finite set of discrete RVs is equal to the product of the individual cumulative pmfs of the RVs. Let M_{ik} be the cumulative pmf along edge (t_i, t_k), so that $M_{ik}(x) = \mathbb{P}[\tilde{w}_{ik} \leq x]$. Define the related function $M^*_{ik}(x) = \mathbb{P}[\tilde{w}_{ik} < x]$. Let Z_i be the set of all possible values of $f_k + \tilde{w}_{ik}$, for $k \in S_i$, and let z run over all elements of Z_i. For $i = 1, \ldots, n$, define

$$\alpha_i = \max_{k \in S_i} (f_k + \min(\tilde{L}_{ik})). \tag{9}$$

Then, with the cost independence assumptions we have already made, we can rewrite equation (8) as

$$f_i = \sum_{z \geq \alpha_i} z \left(\prod_{k \in S_i} M_{ik}(z - f_k) - \prod_{k \in S_i} M^*_{ik}(z - f_k) \right). \tag{10}$$

A complete description of a practical procedure for computing the Fulkerson numbers f_i is given in Algorithm 1. At first blush this may not appear to be any simpler than before but, crucially, the number of operations required to compute each of the f_i is now $O(|\tilde{L}_{ik}| \cdot K)$, where K is the number of child tasks, rather than the first term being exponential in the second as before. Of course, it should also be noted that this procedure is still more expensive than computing the u_i sequence.

Table 3: Fulkerson numbers for example graph.

Task	u_i	f_i	$MC1000$
0	24.75	28.2	
1	16.25	16.9	
2	5.75	5.75	
3	16	18.1	
4	7.75	7.75	
5	7	7	
6	2.5	0	

Once all of the f_i have been computed, they can be taken as alternative task ranks in HEFT (or indeed any other listing heuristic). In Table 3 we compare the f_i values to the standard u_i for the small DAG from Figure 1; also included is *MC1000*, the critical path estimates based on 1000 Monte Carlo realizations (see Section 3.1.1), as a reference for the true critical path of the stochastic graph. We see that the f_i values do indeed give tighter bounds than the u_i and, more pertinently, if we use the former as task ranks then task 3 is scheduled before task 1 and we obtain the smaller schedule makespan of 20 rather than the standard HEFT makespan of 22.

Again, this is a single example, deliberately chosen for this behavior: although the f_i give tighter bounds on the critical path lengths of the stochastic DAG G_s there is absolutely no guarantee that this will lead to superior performance in general. After all, G_s itself is only a model of how we expect the processor selection phase to proceed—one that we know is inaccurate since, for example, it implicitly assumes that all task and edge weights are independent. Indeed, without this independence assumption it is well-known that the relation $u_i \leq f_i$ does not necessarily hold even for G_s; Fulkerson himself presented examples [5]. Still, we think this is a reasonable enough basis for an alternative ranking method in HEFT, so we investigate its performance compared to the usual u_i ranks via numerical simulation in Section 4.

Two refinements of Fulkerson's method were proposed by Elmaghraby [4]. The first involves computing each of the f_i numbers in the aforementioned manner and then reversing the direction of the remaining subgraph in order to calculate an intermediate result which can be used to improve the quality of the bound. The second is a more general approach based on using two or more *point estimates* of e_i, rather than just f_i, a method that was later generalized by Robillard and Trahan [7]. In both cases Elmagharaby proved that the new number sequences achieve tighter bounds on e_i than the Fulkerson numbers f_i. However, small-scale experimentation suggested that the improvement of Elmaghraby's new bounds over Fulkerson's were typically minor compared to the improvement of the latter over the standard HEFT u_i sequence so we chose to only evaluate here whether tightening the bounds at all is useful.

Algorithm 1: Computing the Fulkerson numbers using Clingen's method.

```
for $i = n, \ldots, 1$ do
    $f_i = 0$, $\alpha_i = 0$, $Z_i = \{\}$
    for $k \in S_i$ do
        $\ell_m = \infty$
        for $\ell \in \tilde{L}_{ik}$ do
            $\ell_m \leftarrow \min(\ell_m, \ell)$
            if $f_k + \ell \notin Z_i$ then
                $Z_i \leftarrow Z_i \cup \{f_k + \ell\}$
            end
        end
        $\alpha_i \leftarrow \max(\alpha_i, f_k + \ell_m)$
    end
    for $z \in Z_i$ do
        if $z \geq \alpha_i$ then
            $g = 1$, $d = 1$
            for $k \in S_i$ do
                $g \leftarrow g \times M_{ik}(z - f_k)$
                $d \leftarrow d \times M^*_{ik}(z - f_k)$
            end
            $f_i \leftarrow f_i + z \times (g - d)$
        end
    end
end
```

3.2 Adjusting the pmfs

In some sense, the purpose of the node and edge pmfs m_i and m_{ik} is to simulate the dynamics of the processor selection phase of HEFT—i.e., $m_i(W_i^a)$ should represent the probability that task t_i is scheduled on processor p_a, and so on. In HEFT, tasks are assigned to the processor that is estimated to complete their execution at the earliest time and attempting to model this accurately beforehand can quickly get messy and expensive—especially given the interaction between the two phases of the algorithm. However, a sensible idea may be to simply *weight* the processor selection probabilities according to their respective task computation costs: if, say, a task is 10 times faster on one processor than another then it seems more likely it will be scheduled on the former than the latter, even once the effect of contention is taken into account. More precisely, for all tasks t_i let

$$s_i = \sum_a \frac{1}{W_i^a}$$

and define a new set of pmfs by

$$\hat{m}_i(W_i^a) = \frac{1}{W_i^a s_i} \quad \forall i, a$$

and

$$\begin{aligned} \hat{m}_{ik}(W_{ik}^{ab}) &= \hat{m}_i(W_i^a) \cdot \hat{m}_k(W_k^b) \\ &= \frac{1}{W_i^a W_k^b s_i s_k} \quad \forall i, k, a, b. \end{aligned}$$

Note that we take the reciprocal of the costs in order to reflect the idea that processors with smaller costs are more likely to be chosen than larger ones.

These modified pmfs can be used in conjunction with either upward ranking, as defined by equation (5), Fulkerson's bound, or Monte Carlo methods. For example, in the first instance, the expectations simply become

$$\mathbb{E}[w_i] = \sum_{\ell \in L_i} \ell \hat{m}_i(\ell) = \frac{q}{s_i}, \tag{11}$$

$$\mathbb{E}[w_{ik}] = \sum_{\ell \in L_{ik}} \ell \hat{m}_{ik}(\ell) = \frac{1}{s_i s_k} \sum_{a,b} \frac{W_{ik}^{ab}}{W_i^a W_k^b}, \tag{12}$$

and these can be used with equation (5) to compute an alternative sequence of task ranks $\hat{u}_i$. Of course, this is slightly more computationally expensive than computing the standard u_i ranks but only by a constant factor. Similarly, by using the modified pmfs in conjunction with equation (10) we can define alternative Fulkerson numbers $\hat{f}_i$, and by sampling realizations according to $\hat{m}$ rather than m we can incorporate this idea into the Monte Carlo approach for finding longest paths. All three of these possibilities are evaluated as alternative task ranks for HEFT in the following section. Note that since equations (11) and (12) are simply weighted averages we do not believe that computing $\hat{u}_i$ instead of u_i is a new idea: although we could not find any explicit references in the literature, we suspect this has been done before in practice.

4 Experimental comparison of rankings

In order to evaluate the alternative task prioritization phases in HEFT that we have outlined so far, we make use of our software simulator. However, first we need to define a suitably large and diverse set of graphs to compare them on.

4.1 Testing environment

We decided to evaluate the new rankings using the same two types of graphs as in the previous chapter: a set of Cholesky graphs, with real costs based on timings from an accelerated machine that we have access to, and several sets of randomly-generated graphs based on topologies provided by the STG [8]. The former are identical to those described in Section X so note that in particular they target only the two accelerated platforms detailed there: one comprising 1 GPU and 7 CPU cores and the other 4 GPUs and 28 CPU cores.

Since we are now interested in more general heterogeneous platforms we consider multiple sets of graphs based on randomly-generated graph topologies from the STG which differ from those in the previous chapter. In particular, a set is defined by the following parameters:

- $n \in \{100, 1000\}$, the number of non-entry/exit tasks in each of the graphs. Each graph also has a single entry and exit task so that altogether it has e.g., 102 tasks rather than 100. Once this parameter has been specified, we use the topologies of the corresponding set of that size from the STG.
- $q \in \{2, 4, 8\}$, the number of processors in the target platform.
- $\beta \in \{0.1, 1, 10\}$, the computation-to-communication ratio (CCR). Defined as before, although the manner used to generate costs that give the target CCR differs (see below).
- $h \in \{1.0, 2.0\}$, the *heterogeneity factor* of the processors. This basically determines how similar costs on different processors are to one another (again, see below for more detail).
- $m \in \{\mathrm{R}, \mathrm{UR}\}$, the method used to generate the costs.

Once all of the other parameters are chosen, if $m = \mathrm{UR}$ (for *unrelated*), then we use the same method as in the original HEFT paper [9] or [1] to determine all of the task computation costs on each of the processors. To wit, first an average computation cost for the entire graph $\overline{w_G}$ is chosen randomly (in our case an integer in the interval $[1, 100]$). Then, for all $i = 1, \dots, n$, the average computation cost $\overline{w_i}$ of task t_i, is chosen uniformly at random from the interval $[0, \overline{w_G}]$. Finally, for all $a = 1, \dots, q$, W_i^a is also chosen uniformly at random but from the interval

$$\left[\overline{w_i} \times (1 - h/2),\ \ \overline{w_i} \times (1 + h/2)\right].$$

This method is perhaps somewhat unrealistic since task costs are generated independently of whichever processor they represent; typically costs are determined at least in part by the relative processor *powers*. With this in mind, if $m = \mathrm{R}$ (for *related*), then the method proceeds by first selecting an average power $\overline{p}$ across the set of processors uniformly at random from the interval $[1, 100]$. Then for each processor p_a, $a = 1, \dots, q$, its power r_a is in turn chosen uniformly at random from the interval

$$\left[\overline{p} \times (1 - h/2),\ \ \overline{p} \times (1 + h/2)\right].$$

Now an average task cost $\bar{t}$ is also chosen uniformly at random from the interval $[1, 100]$. For each task t_i, $i = 1, \dots, n$, choose $x_i \in [0, 2\bar{t}]$ uniformly at random and for all $a = 1, \dots, q$, realize $g_a \sim \Gamma(1, r_a)$ (i.e., choose g_a from a Gamma distribution with mean and variance r_a). This is done to ensure that the computation costs are not entirely determined by the power; a Gamma distribution was chosen because it is always positive and heavy-tailed, so has roughly the shape we're after. Finally, let $W_i^a = x_i g_a$ be the computation cost of task t_i on processor p_a.

Communication costs are generated in the same manner for both choices of m. First, an average edge cost $\overline{e}$ is computed such that the specified CCR is (approximately) achieved. Then for all edges (t_i, t_k), we choose $\overline{w_{ik}}$ uniformly at random from the interval $[0, 2\overline{e}]$. Then specific communication costs between the tasks on all possible pairs of distinct processors are chosen uniformly at random from the interval

$$\left[\overline{w_{ik}} \times (1 - h/2),\ \ \overline{w_{ik}} \times (1 + h/2)\right].$$

(Recall that costs are assumed to be zero when both tasks are scheduled on the same processor.) Note that the randomness here means that sometimes the target CCR is not precisely achieved, although it is usually acceptably close.

4.2 Benchmarking

First we want to establish how effective the HEFT task prioritization phase actually is for the sets of graphs described above. More so than the speedup and schedule length ratio, which were described in Section X, the metric that we perhaps really want to use here is how well the task list computed by HEFT compares to other *topological sorts*—orderings which respects precedence constraints—of the tasks. Now, the `Networkx` function `topological_sort` returns a topologically sorted list of nodes in an input graph. Furthermore, the algorithm which this implements does not consider any objective other than meeting the precedence constraints, so we can in some sense regard this as a random sample from the set of all possible topological orderings for the graph. Hence to gauge the effectiveness of the HEFT task prioritization phase, we compare the makespan obtained with the standard ranking with that which would be obtained when using the task priority list returned by `topological_sort` instead. Figure 2 shows the average makespan reduction, as a percentage, for all of the subsets of DAGs from the STG with 100 tasks.

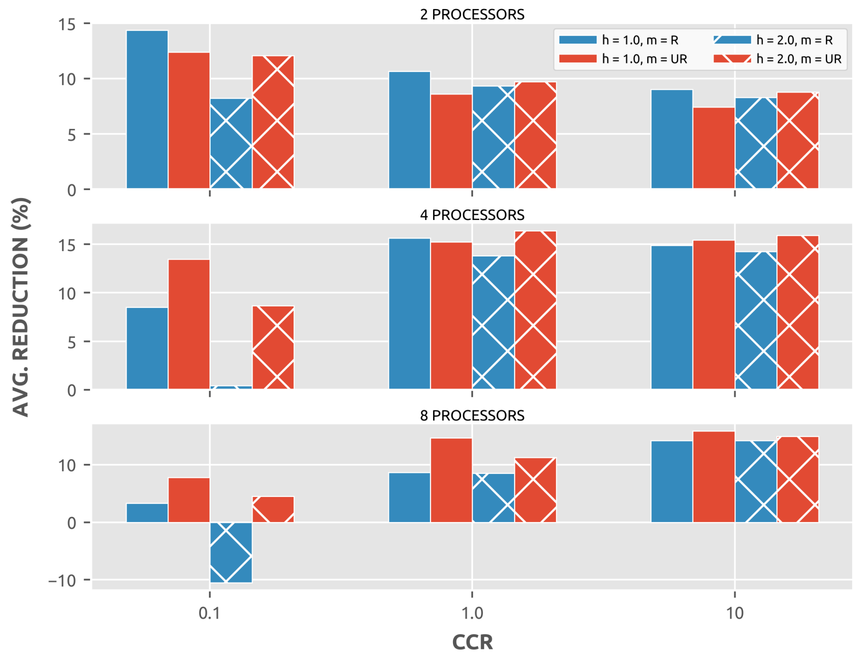


Figure 2: Average makespan reduction (%) for standard HEFT ranking phase vs random topological sort.

Clearly the most interesting takeaway from the figure is the negative reduction that we see for one of the DAG sets in the bottom-left corner—i.e., the random sort did better on average for those DAGs with CCR 0.1 and $h = 2.0$ for which costs were generated using the *related* method. Interestingly, the effect appears to become more pronounced as the ratio of tasks to processors decreases: the reduction is always positive for the larger DAG sets ($n = 1000$) with the same parameters, although the raw number of instances for which the random sort outperformed standard HEFT is high for those as well.

It is not obvious why this should be so but we suspect it is related to the similar phenomenon remarked upon in the previous chapter, where HEFT failed altogether due to difficulty managing communication costs because of its greedy processor selection phase. In fact, both standard HEFT and the random ranking alternative failed altogether in the vast majority of instances for which the former was worse than the latter; the average reduction is much smaller for certain subsets because they are precisely the ones for which HEFT is most likely to struggle in general. Disregarding those instances in which both failed, the percentage of graphs for which the random sort outperformed the standard ranking was roughly 0–2% for all subsets (of both sizes). Given this, we conclude that HEFT's task prioritization phase is clearly useful except for those circumstances when the algorithm itself struggles because of its greedy selection phase.

4.3 Results

In this section we compare the performance of HEFT with the following task prioritization phases:

- the standard upward ranks u_i,
- LB, the optimistic ranks ℓ_i as defined by (2),
- F, the Fulkerson ranks f_i as defined by (10),
- W, the weighted means $\hat{u}_i$ as defined in Section 3.2,
- WF, the weighted Fulkerson numbers $\hat{f}_i$ as defined in Section 3.2.

First, we consider the sets of DAGs based on the STG with 100 tasks as an exploratory example. Ultimately, when deciding whether to use an alternative ranking we want to know whether it's more likely to help rather than harm performance, so Figure 3 shows the difference between the percentage of DAGs for which each of the alternative rankings obtained a schedule makespan better than the standard u_i ranks and the percentage for which they were worse. Note that we combine all four of the subsets ($h = 1.0$ or 2.0, $m =$ R or UR) for each CCR and processor number combination, so that each of the bars represents a percentage over $4 \times 180 = 720$ DAGs.

Of course, the figure does not tell the whole story: what about the relative magnitudes of the makespan reductions and increases? In general, both tended to be fairly small, with makespan changes greater than 5% either way being rare. Overall, across the entire set, the F and WF rankings obtained an average reduction of around 1%, while the LB and F rankings on average did no better than the standard ranking. Of course, a 1% reduction might not seem significant, but in certain situations—such as when the same application has to be executed repeatedly—this may be worthwhile. Bear in mind also that this is an average value: more significant reductions can be seen, for example, as the number of processors increases. Note that while large negative average reductions can be seen for some of the rankings on certain graph sets with $\beta = 0.1$ (i.e., high communication), this is in fact down to only a handful of graphs. The issue is related to the previously-discussed problem of HEFT failing altogether in some cases. Typically, all of the rankings tend to fail for the same instances, but sometimes one or more of the rankings fails spectacularly whereas the HEFT ranking doesn't fail at all; this leads to a very large percentage difference in makespan which somewhat distorts the average. These cases are clearly important so should still be taken into account but they were very infrequent; overall, the number of failures was similar for all of the rankings. All of the data from the experiments described here can be found in the Github repository[2] associated with this chapter.

[2] `critical-path-estimation/scripts`

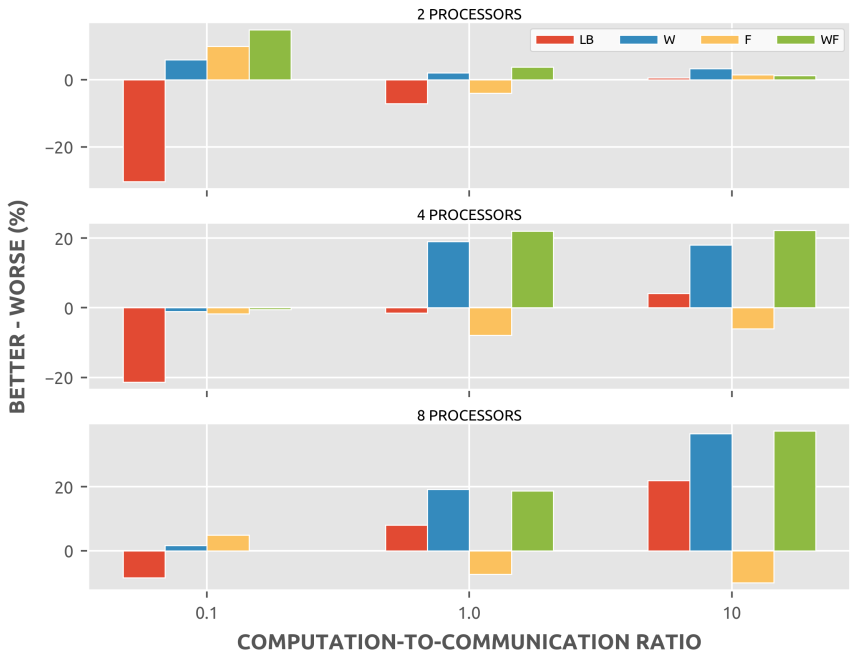
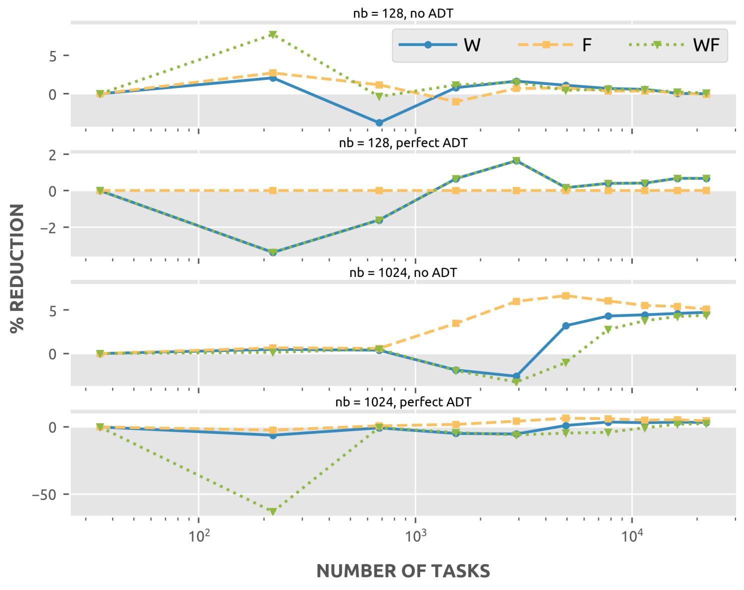


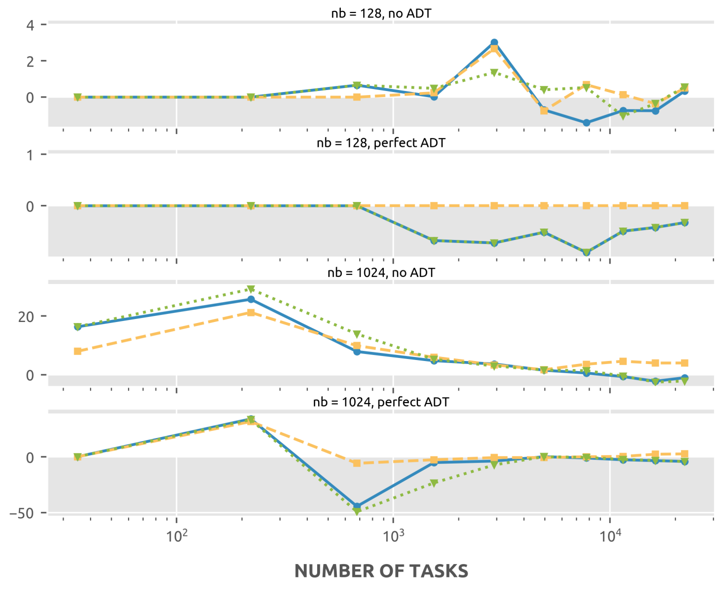
Figure 3: Percentage of instances for which the alternative rankings are better than the default, minus the percentage for which they were worse.

Results so far seem to suggest that the Fulkerson ranking in particular does not seem to be an improvement on the standard ranking. Given how much more expensive it can be, this is a big problem: only large makespan reductions can really justify the extra cost. Indeed, although the weighted Fulkerson ranking WF did much better, both compared to F and the standard ranking, it was not a significant improvement over the W ranking. This suggests that there is little advantage if any to obtaining a tighter bound on the associated stochastic graph (see Section 3.1). To confirm this, we repeated the experiments described above when the Monte Carlo approach (with a sufficient number of samples) is used to compute even tighter bounds on the critical path lengths (which are then used as ranks in HEFT). We used both the original pmfs m and the weighted pmfs $\hat{m}_i$ for this. Our results can be found in full at the Github repository but overall they supported our conclusions from the example above: tightening the bound alone led to no consistent performance gains, whereas weighting the pmfs was more promising, with a roughly 2% average makespan reduction across all of the graphs we considered.

Despite this, there are situations for which the Fulkerson ranking is competitive with the standard ranking in terms of both cost and performance. In particular, for accelerated environments, such as in the previous chapter, the set of possible values each weight may take (i.e., $\tilde{L}_{ik}$) is much smaller, which significantly improves efficiency. Furthermore, the Fulkerson ranking appears to perform



(a) 1 GPU, 7 CPU.



(b) 4 GPU, 28 CPU.

Figure 4: Schedule makespan reduction of HEFT with alternative ranking phases for Cholesky DAGs.

well for certain real application task graphs. For example, Figure 4 shows the makespan reductions of the three rankings W, F and WF compared to the default for the set of Cholesky graphs and (accelerated) target platforms defined in Section X [earlier chapter]. Although there is significant variation, we see that the Fulkerson ranking is superior to the standard ranking phase overall—typically only slightly but occasionally more significantly.

While it may be worthwhile to investigate in future if the performance of the F and WF rankings improves for larger DAGs, given that the results so far suggest they offer no advantage compared to much cheaper alternatives, we decided to omit those two rankings when we repeated our comparison for sets of larger DAGs ($n = 1000$) from the STG. The metric we choose to use now is the average percentage degradation (APD), as defined in the previous chapter. Figure 5 shows the APDs for the standard HEFT ranking and the alternative LB and W rankings. We see roughly the same pattern as for the smaller DAGs: the W ranking consistently outperforms the other two, although the margins are even smaller this time than before.

4.4 Conclusions

To summarize, our main conclusions from all of the experiments described in this section are as follows.

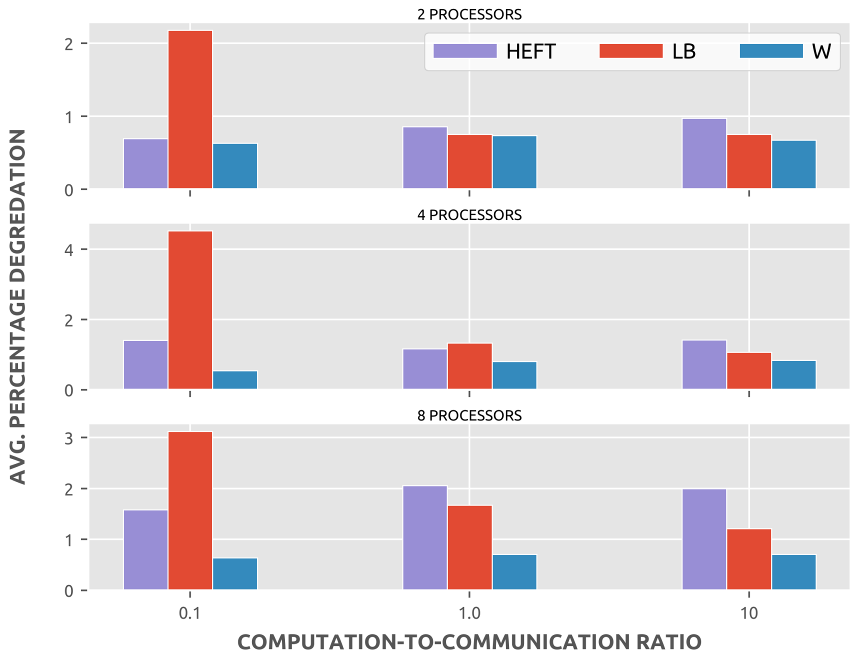


Figure 5: Average percentage degradation for select task prioritization phases in HEFT, size 1000 DAGS from the STG.

- Largely supporting previous investigations along these lines [11], our biggest takeaway is perhaps that how the critical path lengths are estimated seems to make relatively little difference to the schedule makespan overall, with improvements (if any) relative to the default approach typically being fairly minor on average.

- Having said that, the W ranking was consistently more likely to result in a smaller makespan across the majority of the sets considered. The usual caveats about the limitations of our experimental framework hold of course but overall we would softly recommend it be used instead of the default, particularly since it is conceptually simple (just a weighted mean) and not significantly more expensive.

- There is no clear benefit in obtaining tighter bounds on the associated stochastic graph (i.e., Fulkerson's bound). Given the higher time complexity in the general case, this does not therefore seem to be a worthwhile alternative.

- Similarly, there appears to be little advantage in using the LB ranking, although it also wasn't clearly inferior to the standard ranking in terms of performance or cost.

5 Processor selection

Critical path estimates are used in HEFT—and many similar listing heuristics—only at the task prioritization phase. This begs the question, can they also be useful for processor selection? Arabnejad and Barbosa's Predict Earliest Finish Time (PEFT) heuristic [1] represents one sensible way this can be done. Recall from Section X that before scheduling begins PEFT computes a table of *optimistic costs* C_i^a for all task and processor combinations in the following manner. First, set $C_i^a = 0$, $a = 1, \ldots, q$, for all exit tasks, then move up the DAG and recursively compute

$$C_i^a = \max_{k \in S_i} \left(\min_{b=1,\ldots,q} \left(\delta_{ab} \overline{w_{ik}} + W_k^b + C_k^b \right) \right) \tag{13}$$

for all other tasks, where $\delta_{ab} = 1$ if $a = b$ and 0 otherwise. The C_i^a values are referred to in PEFT as optimistic costs but can be interpreted as *conditional critical paths* in that they represent some estimate of future schedule costs given a processor selection. When scheduling, say, task t_i, in PEFT we choose the processor p_{opt} defined by

$$p_{opt} \coloneqq \min_a \left(F_i^a + C_i^a \right)$$

where, as in previous chapters, F_i^a is the estimated schedule makespan when t_i is completed by p_a. This is a nice extension of the dynamic programming principles underlying HEFT: rather than optimizing the schedule makespan up to the current task (i.e., F_i^a), we extend the horizon and optimize an estimate of the complete schedule makespan. Now, computing the full optimistic cost table is only $O(n^2 q)$—i.e., the same as HEFT—but since the values within are similar in nature to the upward ranks u_i it is sensible (and more efficient) to make use of them for prioritizing tasks, rather than going to the effort of computing the upward ranks as well. Hence PEFT defines task priorities C_i through

$$C_i = \frac{1}{q} \sum_a C_i^a. \tag{14}$$

It is important to note here that the task priorities as computed by (14) do not necessarily respect precedence constraints since the equality $C_i^a \leq C_k^a$ for $k \in S_i$ does not hold (because of the internal minimization over all processors). However this is easily remedied by selecting tasks for scheduling from the pool of currently "ready" tasks (i.e., those for which all of their parents have been scheduled) according to their priorities. Note also that although it is arguably more natural that task ranks include the cost of the task itself, which these do not, it is suggested that the savings made through the alternative selection step are more beneficial overall. Certainly, numerical experiments described by the original authors [1] suggest that PEFT is at least competitive with HEFT, especially when there are a large number of processing resources.

The structure of PEFT follows a more general heuristic framework which we refer to as *Heterogeneous Smallest Makespan* (HSM) and is defined by the following procedure:

1. Compute a table of conditional critical path estimates C_i^a for all $i = 1, \dots, n$ and $a = 1, \dots, q$.

2. Compute all task ranks C_i as some function of the C_i^a.

3. At the processor selection phase, schedule task t_i on the processor which minimizes $F_i^a + C_i^a$.

PEFT is defined by using equations (13) and (14) for the first and second parts of this framework, respectively. A natural question is, are there any better choices?

5.1 Alternative conditional critical paths

All of the methods for estimating critical path lengths at the prioritization phase which were introduced previously can be modified to give conditional critical path estimates (which also disregard the cost of the task itself). No matter which, we first let $C_i^a = 0$ for $a = 1, \dots, q$ and all exit tasks, then move up the DAG and recursively compute the other values, so from now on we focus only on the latter. The most straightforward method to extend is the optimistic lower bound, for which we now compute

$$C_i^a = \max_{k \in S_i} \left(\min_{b=1,\dots,q} \left(C_k^b + W_k^b + W_{ik}^{ab} \right) \right), \quad a = 1, \dots, q, \tag{15}$$

for all non-exit tasks. Note that these values are extremely similar to the optimistic costs (13) as used in PEFT, with the exception that the specific communication cost W_{ik}^{ab} is used in the minimization rather than the average $\overline{w_{ik}}$. Indeed, this is arguably the most intuitive way to define the conditional critical path since the value C_i^a is a true lower bound on the remaining makespan of any schedule which executes task t_i on processor p_a; locally-optimal processor selections are overruled if the best possible final makespan we can hope to achieve given that selection is inferior to other choices.

Alternatively, we could take a similar tack to the standard HEFT upward ranks and use an estimate of what we expect the conditional critical paths to be. More specifically, we move up the DAG and recursively compute

$$C_i^a = \max_{k \in S_i} \left(\frac{1}{q} \sum_{b=1}^{q} (C_k^b + W_k^b + W_{ik}^{ab}) \right), \quad a = 1, \dots, q. \tag{16}$$

In fact, we could just as easily use a weighted average inside the maximization, which is particularly attractive since the corresponding ranking performed well in the previous section. In particular, for all non-exit tasks we recursively compute

$$C_i^a = \max_{k \in S_i} \left(\frac{1}{\hat{s}_k} \sum_{b=1}^{q} \frac{C_k^b + W_k^b + W_{ik}^{ab}}{W_k^b + C_k^b} \right), \quad a = 1, \ldots, q, \tag{17}$$

where

$$\hat{s}_k = \sum_{b=1}^{q} \frac{1}{W_k^b + C_k^b}. \tag{18}$$

This is conceptually similar to the weighted pmf $\hat{m}$ as defined in Section 3.2, except that we also include the conditional critical path lengths in the weighting. The motivation for the change is that the probability a task t_i will ultimately be scheduled on a given processor p_a at the selection phase now also depends on the C_i^a value.

5.2 Computing task priorities

There are many different ways we can use the conditional critical path estimates C_i^a to derive task priorities C_i. However, given the superior performance of the weighted average task ranking in Section 4, a similar approach here would seem to be most sensible default choice. In particular, no matter how the conditional critical paths C_i^a are estimated, we compute the task priorities recursively, starting from the leaves, through

$$C_i = \frac{q}{\hat{s}_i} + \max_{k \in S_i} C_k, \tag{19}$$

where the maximization is taken to be zero if S_i is empty (i.e., for exit tasks) and $\hat{s}$ is as defined in (18). All of the results presented in the following section are based on heuristics for which the task priorities are computed in this manner, although it should again be emphasized that others are possible and several were considered in preliminary testing.

5.3 Empirical comparison

In order to evaluate the performance of the different choices in the HSM framework that have been described so far, we used the same software simulation environment as in Section 4. As before, all of the scripts used to generate the results presented here are available at the Github repository[3] for this chapter. We consider the following three variants of the HSM framework, defined by how they compute the conditional critical paths since all use the task prioritization given by (19).

[3]`critical-path-estimation/scripts`

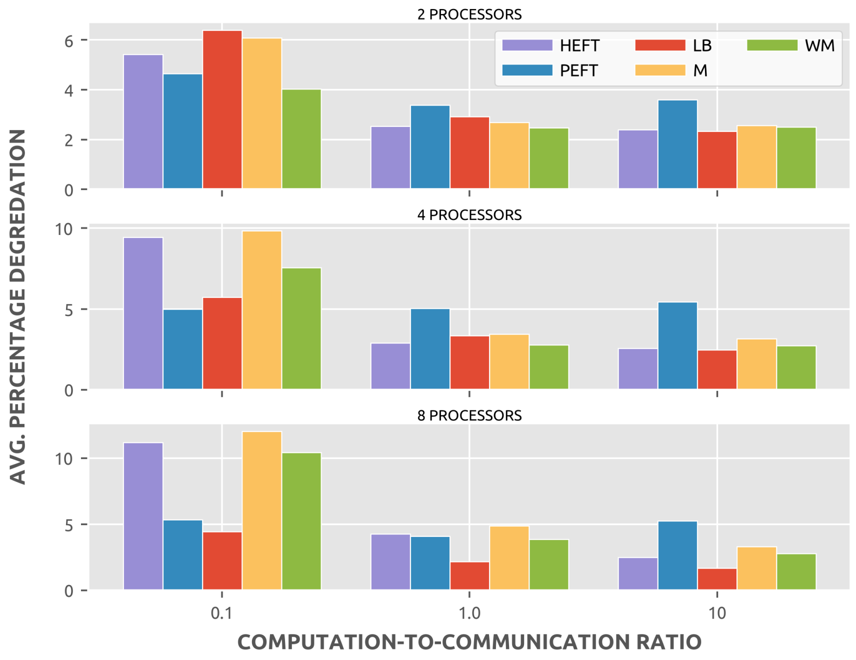


Figure 6: Average percentage degradation for HSM variants, HEFT and PEFT, size 100 DAGS from the STG.

- LB, using (15).
- M, using (16).
- WM, using (17).

Figure 6 shows the average percentage degradation of these three heuristics, as well as HEFT and PEFT, for the set of DAGs based on topologies from the STG with 100 tasks (plus single entry and exit tasks). As before, we combine all four of the subsets ($h = 1.0$ or 2.0, $m =$ R or UR) so that each CCR and processor number combination represents a larger subset comprising 720 DAGs. The most immediate takeaway from the figure is that, with the exception of the subsets with the smallest CCR, PEFT actually does relatively poorly compared to both HEFT and the HSM variants. On the surface, this might seem to contradict results published elsewhere, but it should be noted that those experiments were typically for much larger and more diverse graph sets, so some local variation is to be expected. Note that when communication predominates (i.e., $\beta = 0.1$), PEFT and HSM-LB are superior to the others for both 4 and 8 processors. This is again related to the problem of listing heuristics failing altogether for DAGs with high communication: PEFT and HSM-LB recorded only around half as many failures as the others for these sets.

One obvious question we have not so far addressed is how useful the PEFT-like processor selection phase actually is compared to the simple HEFT-like earliest finish time alternative. In fact, this was far from clear in our experiments: with a few exceptions, there was usually no statistically significant difference on average between the two. The exceptions were all for the smallest CCR sets, for which the processor selection clearly improved ($\approx 5\%$ average makespan reduction) on the ranking-only version for the LB variant and was even more significantly worse for the M and WM variants ($> 5\%$ average makespan increase).

References

[1] H. Arabnejad and J. 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.

[2] Louis-Claude Canon and Emmanuel Jeannot. Correlation-aware heuristics for evaluating the distribution of the longest path length of a DAG with random weights. *IEEE Transactions on Parallel and Distributed Systems*, 27(11):3158–3171, 2016.

[3] C. T. Clingen. A modification of Fulkerson's PERT algorithm. *Operations Research*, 12(4):629–632, 1964.

[4] Salah E. Elmaghraby. On the expected duration of PERT type networks. *Management Science*, 13(5):299–306, 1967.

[5] D. R. Fulkerson. Expected critical path lengths in PERT networks. *Operations Research*, 10(6):808–817, 1962.

[6] Jane N. Hagstrom. Computational complexity of PERT problems. *Networks*, 18(2):139–147.

[7] Pierre Robillard and Michel Trahan. Technical note—expected completion time in PERT networks. *Operations Research*, 24(1):177–182, 1976.

[8] Takao Tobita and Hironori Kasahara. A standard task graph set for fair evaluation of multiprocessor scheduling algorithms. *Journal of Scheduling*, 5(5):379–394.

[9] 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.

[10] Richard M. Van Slyke. Letter to the editor—Monte Carlo methods and the PERT problem. *Operations Research*, 11(5):839–860, 1963.

[11] Henan Zhao and Rizos Sakellariou. An experimental investigation into the rank function of the Heterogeneous Earliest Finish Time scheduling algorithm. In *Euro-Par 2003 Parallel Processing*, Harald Kosch, László Böszörményi, and Hermann Hellwagner, editors, Berlin, Heidelberg, 2003, pages 189–194. Springer Berlin Heidelberg.