Critical path estimation in heterogeneous scheduling heuristics

Thomas McSweeney[*]

3rd August 2020

1 Introduction

Recall from Chapter X that the *Heterogeneous Earliest Finish Time* (HEFT) heuristic prioritizes all tasks t_i, $i = 1, \dots, n$, by recursively computing a corresponding sequence of numbers u_i which are intended to represent *critical path* lengths from each task to the end. However, as noted previously, the concept of the *critical path* is not clearly defined: DAG weights are not fixed at this stage so there are multiple 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; more 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. What is the longest path through this graph?

Clearly there are multiple possible ways this can be defined. The HEFT approach, as described in previous chapters, is to use mean values over all sets of possible task and edge costs to transform the DAG into a fixed-cost version, as in Figure Y, and then compute the critical path lengths in a standard dynamic programming way to find the task ranks. For the simple graph, we find that $u_4 = 5$, $u_3 = 19/2$, $u_2 = 21/2$ and $u_1 = 16$. Now, it should be clear that the optimal schedule for the example graph is to simply schedule all tasks on processor $P1$, which gives us a schedule of length 6. Although HEFT finds the optimal schedule, the u_i values are not bounds on the schedule makespan... Conceptually, the HEFT critical path isn't a lower bound on any schedule makespan but a lower bound on what the schedule is likely to be...

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

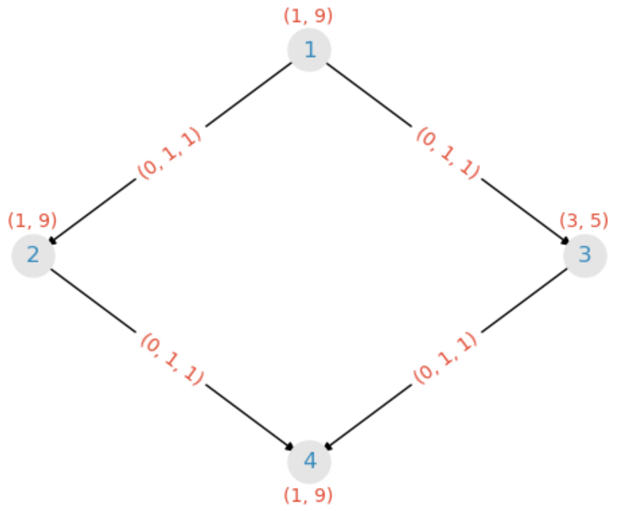

Figure 1: Simple task graph with costs on a two-processor target platform.

Given this ambiguity, alternative ways to definite the critical path in HEFT have been considered before, most notably by Zhao and Sakellariou [5], 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.

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

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

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 u_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 $u_i^a = W_i^a \ \forall a$ for all exit tasks then moving up the DAG and setting

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

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

$$u_i = \min_{a=1,\dots,q} u_i^a \tag{2}$$

gives a true lower bound on the remaining cost of any schedule once the execution of task t_i begins. These u_i values could be useful as alternative task priorities in HEFT, especially since the cost of computing all of the u_i^a in this manner is only $O(m+n) \approx O(n^2)$ so in particular is no more expensive than the usual HEFT prioritization phase. Of course, it should be emphasized here that there is absolutely no mathematically valid reason to suppose that doing so will actually lead to superior performance. Still, it seems worthwhile to investigate this empirically using our simulator, which we duly do in Section 4.

Although taking the minimum over the set of processors as in (2) is the only choice that gives a true lower bound, we could use any other average over the set of processors, such as the mean or maximum, in order to compute task priorities. Since we also consider those two alternatives in Section 4, we refer to the ranking phase defined by using the minimum, maximum and mean as OPT-MIN, OPT-MAX and OPT-MEAN respectively.

(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 Alternative rankings

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 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 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

$$m_i(W_i^a) \coloneqq \mathbb{P}[w_i = W_i^a] = \frac{1}{n_p} \quad \forall a$$

and

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

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

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

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

In particular, this means that $\mathbb{E}[w_i] = \overline{w_i}$ and $\mathbb{E}[w_{ik}] = \overline{w_{ik}}$ so that the computation of the upward ranks u_i 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.

To sum up, 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 Sharper bounds on 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 and the critical path of the stochastic graph G_s? Computing the distribution of the critical path length, or even just the moments, for a general stochastic DAG is known to be a difficult problem and is discussed in much more detail in Chapter X. However, suffice it to say here, it has long been known in the context of *Program Evaluation and Review Technique* (PERT) network analysis that the numbers u_i are in fact *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 [3], who referred to it as already being widely-known and gave a simple proof. Furthermore, he also showed how a sequences of numbers which give tighter bounds can easily be constructed.

3.1.1 Fulkerson's bound

First we explain how the DAG as shown in Figure **??** can be expressed in an equivalent formulation with only edge weights. This step is not strictly necessary but simply makes the elucidation cleaner; we should emphasize that all of the following still holds, with only minor adjustments, if this is not done.

Suppose (t_i, t_k) is any edge of the DAG and its cost w_{ik} represents the computation cost of the parent task t_i plus the communication cost of the edge, unless t_k is an exit task, in which case the edge cost also includes the computation cost of t_k. Let $\tilde{c}_{ik} = c_i + \delta_k c_k$ where $\delta_k = 1$ if t_k is an exit task and 0 otherwise. Likewise, let $\tilde{g}_{ik} = g_i + \delta_k g_k$. For $s \in \{c, g\}$, define

$$\tilde{C}^s_{ik} = C^s_{ik} + \tilde{c}_{ik} \quad \text{and} \quad \tilde{G}^s_{ik} = G^s_{ik} + \tilde{g}_{ik}.$$

Then the edge can take one of six possible values, as illustrated in Figure **??**, so that the simple DAG from Figure **??** can be represented instead as in Figure **??**. Let $\tilde{L}_{ik} = \{\tilde{c}_{ik}, \tilde{C}^c_{ik}, \tilde{C}^g_{ik}, \tilde{g}_{ik}, \tilde{G}^g_{ik}, \tilde{G}^c_{ik}\}$ be the set of all possible costs the edge may represent.

To compute the upward ranks of all tasks, the estimated edge pmfs m_{ik} need to be replaced by very similar functions $\tilde{m}_{ik}$ defined by

$$\tilde{m}_{ik}(\tilde{c}_{ik}) = \frac{n_c}{n_p^2}, \qquad \tilde{m}_{ik}(\tilde{C}^c_{ik}) = \frac{n_c(n_c-1)}{n_p^2},$$
$$\tilde{w}_{ik}(\tilde{g}_{ik}) = \frac{n_g}{n_p^2}, \qquad \tilde{m}_{ik}(\tilde{G}^g_{ik}) = \frac{n_g(n_g-1)}{n_p^2},$$
$$\tilde{m}_{ik}(\tilde{C}^g_{ik}) = \tilde{m}_{ik}(\tilde{G}^c_{ik}) = \frac{n_c n_g}{n_p^2}.$$

The ranks themselves are then given by working up from the leaves of the DAG and recursively computing

$$\tilde{u}_i = \begin{cases} 0, \quad \text{if } t_i \text{ is an exit task,} \\ \max_{k \in S_i} \{\tilde{u}_k + \mathbb{E}[w_{ik}]\}, \qquad \text{otherwise,} \end{cases} \tag{6}$$

where the expectation is computed using the pmfs $\tilde{m}_{ik}$. It can readily be verified that this sequence of numbers is identical to the u_i, with the exception of those corresponding to exit tasks which are now equal to zero. Note that here we have used the default HEFT pmfs m as the basis for $\tilde{m}$, when we could just as easily have used the *biased* version $\hat{m}$ defined in Section **??**. We will use the default throughout this section but the alternative is also considered when results are presented in Section **??**.

Let $W_i \coloneqq \{w_{ik}\}_{k \in S_i}$ be the set of all the weight RVs corresponding to edges which connect task t_i to its children. Assume that all tasks are indexed in a topological order, so that in particular if t_k is a child of t_i then $i > k$, and define

$$Z_i \coloneqq W_i \cup \{W_k\}_{k \in S_i}.$$

Let R_i be the set of all possible *realizations* of the RVs in Z_i and suppose that $z_i \in R_i$ can be expressed in the form

$$z_i = z_i^i \cup \{z_i^k\}_{k \in S_i}.$$

where z_i^k contains the realizations of the RVs in W_k. Given a realization $z_i \in R_i$, let $\ell(z_i)$ be the critical path length from task t_i to the end. Define e_i to be the expected downward critical path length from, and including, task t_i. Then by definition we have

$$e_i = \sum_{z_i \in R_i} \mathbb{P}[Z_i = z_i] \ell(z_i). \tag{7}$$

By taking average values over all processors for each task weight, HEFT effectively assumes that they are all independent of one another, although of course this may not in fact be the case. Likewise the edge weights are also implicitly regarded as being independent both of each other and the task weights, which is definitely not true since they are entirely determined once the task weights w_i are realized. Disregarding this modeling inaccuracy for the moment, for the edge weight-only version of the stochastic DAG that we are using here, all of the RVs $w_{ik} \in W_i$ incorporate the task weight RV w_i so that they are no longer independent of one another—but they are still assumed to be independent of the edge weights in the other sets W_j, $j \neq i$. In particular, this implies that

$$\mathbb{P}[Z_i = z_i] = \mathbb{P}[W_i = z_i^i] \prod_{k \in S_i} \mathbb{P}[W_k = z_i^k]. \tag{8}$$

With this we can rewrite equation (7) as

$$\begin{aligned}e_i &= \sum_{z_i \in R_i} \mathbb{P}[Z_i = z_i]\ell(z_i) \\ &= \sum_{z_i^i} \sum_{\substack{z_k \in R_k, \\ k \in S_i}} \mathbb{P}[W_i = z_i^i]\mathbb{P}[Z_k = z_k] \max_{k \in S_i}\{\ell(z_k) + z_{ik}\}, \qquad (9)\end{aligned}$$

where z_{ik} is the realization of the edge weight RV w_{ik} defined by the set of realizations z_k. It is now relatively straightforward to show that the identity $\tilde{u}_i \leq e_i$ (and therefore $u_i \leq f_i$) holds for all tasks t_i by manipulating this expression; the reader is directed to Fulkerson's original paper for full details [3]. Moreover, suppose we define a sequence of numbers by $f_i = 0$, if t_i is an exit task, and

$$\begin{aligned}f_i &= \sum_{z_i \in R_i} \mathbb{P}[Z_i = z_i] \max_{k \in S_i}\{f_k + z_{ik}\} \\ &= \sum_{z_i^i} \sum_{\substack{z_k \in R_k, \\ k \in S_i}} \mathbb{P}[W_i = z_i^i]\mathbb{P}[Z_k = z_k] \max_{k \in S_i}\{f_k + z_{ik}\}, \qquad (10)\end{aligned}$$

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

Although the bound is tighter, for each of the f_i 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. Even in this simple example for which $|\tilde{L}_{ik}| \equiv 6$ this quickly becomes impractical, as suggested by our (deliberately) tedious breakdown of the calculations above. In more general heterogeneous environments, $|\tilde{L}_{ik}|$ can be $O(p^2)$ so computing the f_i is even more daunting. Fortunately, a much more efficient method of computing the f_i was given by Clingen [1] 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 [2].

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 *unique* 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

$$\alpha_i = \max_{k \in S_i}(f_k + \min(\tilde{L}_{ik})). \qquad (11)$$

Then, with the cost independence assumptions we have already made, we can rewrite equation (10) as

$$f_i = \sum_{z \geq \alpha_i} z \left(\prod_{k \in S_i} M_{ik}(z - f_k) - \prod_{k \in S_i} M_{ik}^*(z - f_k) \right). \qquad (12)$$

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 $O(|\tilde{L}_{ik}| \cdot K)$, where K is the number of child tasks, rather than the first term being exponential in the second as before. Of course, this procedure is *still* more expensive than computing $\tilde{u}_i$ using (6), but in our experience it was practical even for large, dense DAGs and target platforms comprising numerous processing resources (albeit so far we have only consider accelerated platforms that follow the model described in Chapter **??**).

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

```
for $i = n, \dots, 1$ 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
            $g = 1$, $q = 1$
            for $k \in S_i$ do
                $g \leftarrow g \times M_{ik}(z - f_k)$
                $q \leftarrow q \times M^*_{ik}(z - f_k)$
            end
            $f_i \leftarrow f_i + z \times (g - q)$
        end
    end
end
```

Once all of the f_i have been computed, they can be taken as alternative task ranks in HEFT (or any listing heuristic). However, it should be emphasized here that 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 the full heuristic. After all, G_s itself is only a model of how we expect the processor selection phase to proceed—one that we know for a fact 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 [3]. 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 the next section.

Two refinements of Fulkerson's method were proposed by Elmaghraby [2]. 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 [4]. 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 bound at all does in fact lead to better performance in HEFT.

3.1.2 Monte Carlo

Monte Carlo simulation can also be used...

3.2 Adjusting the pmfs

Ultimately 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., $m_i(c_i)$ should represent the probability that task t_i is assigned to a CPU, and $m_i(g_i)$ likewise for a GPU. 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.

A simple idea that seems sensible, especially for accelerated platforms, is to simply *bias* the processor selection probabilities according to their relative power: if, say, a task's GPU cost is 10 times smaller than its CPU cost, 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, let r_i be the acceleration ratio of task t_i and define an alternative series of pmfs for all nodes and edges by

$$\hat{m}_i(c_i) = \frac{n_c}{n_c + r_i n_g}, \qquad \hat{m}_i(g_i) = \frac{r_i n_g}{n_c + r_i n_g},$$
$$\hat{m}_{ik}(C_{ik}^c) = \frac{n_c(n_c - 1)}{(n_c + r_i n_g)(n_c + r_k n_g)}, \qquad \hat{m}_{ik}(G_{ik}^g) = \frac{r_i r_k n_g(n_g - 1)}{(n_c + r_i n_g)(n_c + r_k n_g)},$$

$$\hat{m}_{ik}(C_{ik}^g) = \frac{r_k n_c n_g}{(n_c + r_i n_g)(n_c + r_k n_g)}, \qquad \hat{m}_{ik}(G_{ik}^c) = \frac{r_i n_c n_g}{(n_c + r_i n_g)(n_c + r_k n_g)}$$

$$\hat{m}_{ik}(0) = \frac{n_c + r_i r_k n_g}{(n_c + r_i n_g)(n_c + r_k n_g)}.$$

The expected values of the weights then become

$$\mathbb{E}[w_i] = \sum_{\ell \in L_i} \ell \hat{m}_i(\ell) = \frac{c_i n_c + g_i r_i n_g}{n_c + r_i n_g}, \tag{13}$$

$$\begin{aligned} \mathbb{E}[w_{ik}] &= \sum_{\ell \in L_{ik}} \ell \hat{m}_{ik}(\ell) \\ &= \frac{n_c(n_c - 1)C_{ik}^c + n_c n_g (r_k C_{ik}^g + r_i G_{ik}^c) + n_g(n_g - 1) r_i r_k G_{ik}^g}{(n_c + r_i n_g)(n_c + r_k n_g)}, \end{aligned} \tag{14}$$

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. While there is no mathematically valid reason to suppose that the modified pmf is truly any more useful than the original, we evaluate its performance empirically using our simulation model at the end of this section.

4 Experimental comparison of rankings

Baseline comparison with random sort.

5 Processor selection

References

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

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

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

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

[5] 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.