On the Scope of Applicability of the ETF Algorithm

Cristina Boeres, George Chochia, and Peter Thanisch
Department of Computer Science
The University of Edinburgh
King's Buildings, Edinburgh EH9 3JZ
Scotland
email: {cbx,gac,pt}@dcs.ed.ac.uk

Abstract

Superficially, the *Earliest Task First* (ETF) heuristic [1] is attractive because it models heterogeneous messages passing through a heterogeneous network. On closer inspection, however, this is precisely the set of circumstances that can cause ETF to produce seriously sub-optimal schedules. In this paper we analyze the scope of applicability of ETF. We show that ETF has a good performance if messages are short and the links are fast and a poor performance otherwise. For the first application we choose the Diamond DAG with unit execution time for each task and the multiprocessor system in the form of the fully connected network. We show that ETF partitions the DAG into lines each of which is scheduled on the same processor. The analysis reveals that if the communication times between pairs of adjacent tasks in a precedence relation are all less than or equal to unit then the schedule is optimal. If the communication time is equal to the processing time needed to evaluate a row then the completion time is $O(\sqrt{n})$ times more than the optimal one for an $n \times n$ Diamond DAG. For the second application, we choose the join DAG evaluated by two connected processors.

1 Introduction

The problem of scheduling an application onto a set of processors, taking into account communication delays, has been widely studied and is known to be NP-hard except for special cases [11]. One of the forms is the problem of finding an efficient static schedule of a set of partially ordered tasks on a multiprocessor system. Many scheduling heuristics have been proposed to solve this problem, and based on the technique employed, the heuristics may be classified in different groups [7]. Heuristics that manipulate the paths between the source node and sink node of the graph are classified as *critical path heuristics*. Another group of heuristics are the ones which give priorities to the tasks, depending on the precedence order and the weights associated to tasks and edges. These algorithms are classified as *list scheduling heuristics*.

In an attempt to include the information about different topologies of real multiprocessors, Hwang *et al.* [1] proposed the *Earliest Task First* (ETF), which is a simple greedy strategy, classified as a *list scheduling heuristic* [2]. ETF schedules a free task to an available processor, in which it can start as early as possible; a task is 'free' when all its immediate predecessors have already been scheduled. It is emphasised that ETF tries to hide communication with computation, leading to the establishment of its performance guarantee.

In [1], an extensive analysis of ETF is done and the makespans of schedules determined by ETF are shown. However, we wish to focus on problems that arise with ETF when certain relationships hold between the parameters that represent the application and the architecture.

In [12], the performance of ETF is compared with the Dominant Sequence Clustering (DSC) algorithm. The number of processors given to ETF is the same as the number of clusters determined by DSC algorithm. Similarities and differences between the two algorithms are described in [12] which examines the ETF property of considering the distance in choosing the best processor to allocate to tasks. Gerasoulis *et al.* point out that ETF gives poor results when scheduling a join *Dag*. In such a case, if the communication cost is sufficiently greater then the computation cost, it is better to execute the tasks in the same processor. However, as ETF schedules the tasks at the earliest time possible, it does not allocates tasks at the same processor even if the communication cost is high.

In [4] an analysis and classification of non-greedy heuristics and greedy heuristics is done. ETF is classified as a greedy heuristic and an extensive set of experiments shows that the DFBN non-greedy heuristic proposed in [3] has a smaller time complexity than greedy algorithms like ETF. However, both ETF and DFBN give bad schedules when considering a join *Dag* as described by Gerasoulis *et al.* [12].

The paper is organized as follows: in Section (2) we give a description of the ETF algorithm. Section (3) contains applications of ETF to the Diamond DAG and the join DAG; we show that the time complexity of ETF can be more than a constant factor times worse than the optimal one. In section (4) we analyze the scope of applicability of ETF and suggest a three stage heuristic which hopefully may improve the situation.

2 The ETF Algorithm

The ETF algorithm was developed for scheduling tasks on multiprocessor systems with an arbitrary topology. The interesting feature of the algorithm is that it makes a scheduling decision based on the delay associated with message transfer between any pair of nodes in the network and the amount of data to be transferred between a pair of tasks t, t' such that t is an immediate predecessor of t'.

Given a set of tasks $T = \{t_i |\ i = 1, ..., m\}$ with precedence relation $<$, i.e. where $t, t' \in T$, $t < t'$ means that the evaluation of t' cannot be initiated until the evaluation of the t is complete. $G_T = (T, <)$ is the directed acyclic graph. The function $\mu(t)$: $T \rightarrow Z_0^+$ specifies the evaluation time of the task $t \in T$. Function $\eta(t, t')$: $T \times T \rightarrow Z_0^+$ is associated with the amount of data to be transferred between t and t'. The multiprocessor system is a set of identical processors $P = \{p_i |\ i = 1, ..., n\}$. For each pair $p, p' \in P$ the function $\tau(p, p')$: $P \times P \rightarrow Z_0^+$ specifies the delay associated with passing of a message of unit size from processor p to p'. The following bound, ω_{ETF}, on the *makespan*, i.e. time to complete the evaluation of any G_T, is proved

$$\omega_{ETF} \leq \left(2 - \frac{1}{n}\right) \omega_{opt} + C, \qquad (1)$$

where ω_{opt} is the optimal completion time attainable when communication is ignored and

$$C = \max_{chains} (\tau_{max} \sum_{i=1}^{l-1} \eta(t_{c_i}, t_{c_{i+1}})),$$

where $(t_{c_1}, ..., t_{c_l})$ is a chain of l tasks from G_T, and $\tau_{max} = \max_{p,p' \in P} \tau(p, p')$. The maximum is taken over the set of all source to sink chains.

In order to formulate the algorithm it is necessary to introduce the variable *current moment (CM)*, representing the current moment of the event clock, a set of available tasks, A, which can be scheduled at time CM, and a set I of idle processors at time CM. Let D_t and S_t denote, respectively, the sets of immediate predecessors and successors of task t. Let $s(t)$ and $f(t)$ denote, respectively, the times when the evaluation of task t begins and finishes. If all immediate predecessors of t are scheduled then function $r(t, p)$

$$r(t, p) = \begin{cases} 0 & \text{if } D_t = \emptyset \\ \max_{t' \in D_t}(f(t') + \eta(t', t)\tau(p(t'), p)) & \end{cases} .$$

evaluates the time when the last message from the predecessor task arrives in p, i.e. the earliest time at which t may be scheduled on p. The function *next(CM)* returns the earliest time after CM at which a processor finishes computing a task. The ETF algorithm is given in Figure 1.

Initialization: $I=\{p_1, ..., p_k\}$, $A=\{t \mid D_t = \emptyset\}$, $q=0$;
$CM=0$, $NM = \infty$, $r(t,p) = 0$, $\forall\ t\ \in A$, $\forall\ p\ \in\ I$; $f(t) = \infty$, $\forall\ t \in T$;

0 *while* $q < \mid T \mid$ *do*
1   *while* $I \neq \emptyset$ *AND* $A \neq \emptyset$ *do*
2     $\hat{t}, \hat{p} :\ r(\hat{t}, \hat{p}) = \min_{t \in A} \min_{p \in I} r(t,p)$;
3     $e_s = \max(CM, r(\hat{t}, \hat{p}))$;
4   *if* $e_s \leq NM$ *then*
5       $p(\hat{t}) = \hat{p}; s(\hat{t}) = e_s; f(\hat{t}) = s(\hat{t}) + \mu(\hat{t})$;
6       $A = A - \hat{t}$; $I = I - \hat{p}$; $q = q + 1$;
7   *if* $f(\hat{t}) \leq NM$ *then* $NM = f(\hat{t})$ *endif*
8   *else goto 11*
9   *endif*
10   *endwhile*
11   $CM = NM; T_{new} = \{t \mid f(t) = CM, t \in T\}; NM = next(CM); A' = \emptyset$;
12   *for* $t \in T_{new}$ *do*
13     $I = I \cup p(t)$;
14   *for* $t' \in S_t$ *do*
15       $D_{t'} = D_{t'} - 1$;
16   *if* $|D_{t'}| \equiv 0$ *then* $A' = A' \cup t'$ *endif*
17   *endfor*
18   *endfor*
19   $A = A \cup A'$;
20   *for* $t' \in A$ *do*
21   *for* $p \in I$ *do*
22       $r(t', p) = \max_{t \in D_{t'}}(f(t) + \eta(t,t')\ \tau(p(t), p))$
23   *endfor*
24   *endfor*
25 *endwhile*

Figure 1: The ETF algorithm

3 Applications of ETF

In this section we consider two applications of ETF. Let $G_T = (T, <)$ be the *diamond DAG* with n^2 nodes [8]. The nodes and the edges of the graph can be associated with that of the of $n \times n$ rectangular grid. The nodes are at the grid intersection points and the edges connect nodes at neighbouring intersections. The edges have the orientation from the node with smaller cartesian coordinate to that with larger one. The Diamond DAG with $n = 6$ is show in Figure 2. The tasks from T can be identified as $t_{i,j},\ i,j = 0, ..., n-1$, where the first index is used for the row number, and the second for column number. Choose the multiprocessor

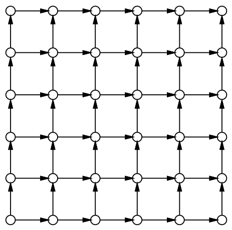

Figure 2: The Diamond DAG

system to be a completely connected network of $k \leq n$ processors. Set $\tau(p, p') = 1$ for all $p, p' \in P$ and $\eta(t, t') = \delta$ for all $t, t' \in T$ such that $t < t'$.

Theorem 1 *ETF schedules all the tasks in any given row on the same processor so that the task $t_{i,j}$ is scheduled in time $i \cdot (1+\delta)+j$. Also, at most $\left\lceil \frac{n}{1+\delta} \right\rceil$ processors may be involved in the computation simultaneously.*

Proof. Apply the induction. At time 0, the set A has one member t_{00} which has no predecessors. The set I consists of all processors in the network. Pick any of them (line 2). The condition in line 4 is true, hence t_{00} will be scheduled on this processor (line 5). The task will be removed from A (line 6) and will never become a member of A in the future. Indeed, the only new members of A are successors of t_{00} (lines from 11 to 19). Hence the theorem holds for time 0. Suppose it holds until time u. The set of tasks which were scheduled in time u satisfy the equation $u = i \cdot (1 + \delta) + j$ which has at most one solution for each row i. According to the induction hypothesis these are the only tasks which are evaluated. At time $u + 1$, which is *next(u)*, all these processors finish the evaluation of their tasks, thus they will be placed into I again (line 13). Consider any $i = (u - j)/(1 + \delta)$ for some $j = 0, ..., n - 2$, such that $t_{i,j}$ is not the last task in that row. Then the task $t_{i,j+1}$ will be placed in the set A of available tasks at time $u + 1$. This task can be scheduled at $u + 1$ on the same processor p as $t_{i,j}$ or at $u + 1 + \delta$ on any other available processor. Thus function $r(t_{i,j+1}, p)$ is minimal for p, hence this processor will be selected (line 2). Let $t_{i,j}$ be the task with maximal i satisfing $u = i \cdot (1 + \delta) + j$ at time u. Then if $u + 1 = (i + 1) \cdot (1 + \delta)$ the task t_{i+1j} will be available at time $u + 1$ and it will be scheduled on some (and only one) available processor from I. Therefore we have shown that all the tasks in the same row are evaluated by the same processor as the previous task in the row. If $t_{i,j}$ is the last task in the row, i.e. $j = n-1$ the processor is returned to I i.e. is available again. Hence there could be at most $\left\lceil \frac{n}{1+\delta} \right\rceil$ processors involved in the computation simultaneously. ■

The theorem establishes that the ETF partitions the diamond DAG into lines. It was shown in [6] that for this partitioning, the makespan ω_l is

$$\omega_l = \frac{n^2}{k} + (k-1)(1+\delta) \tag{2}$$

where k is the number of processors. Formally the minimum of (2) is attainable when $k = \frac{n}{\sqrt{1+\delta}}$. However it cannot be reached because of the restriction $k \leq \frac{n}{1+\delta}$ established above. As ω_l is a decreasing function of k before the formal minimum, the attainable minimum corresponds to maximal k, i.e. $k = \frac{n}{1+\delta}$, in which case we obtain

$$\omega_l = n + (n-1)\cdot(1+\delta) \quad . \tag{3}$$

Below we prove that (3) is the minimal possible makespan if $0 \leq \delta \leq 1$. Indeed if δ is within this region the task $t_{i+1,i+1}$ cannot be evaluated earlier than $1+\delta$ with respect to $t_{i,i}$ for all $i = 0, ..., n-2$. Thus the minimal makespan is given by $(n-1)\cdot(1+\delta)$ plus the time required to evaluate n tasks $t_{i,i},\ i = 0, ..., n-1$ i.e. (3). This is the region where ETF finds the optimal schedule. However this is not the case if δ is large. If $\delta = n-1$, then $k = 1$, which means that the DAG will be evaluated by single processor in time n^2.

Consider the partitioning of the DAG into stripes [8] of size $\sqrt{n}$. All stripes mapped on different processors. The first processor starts evaluating the first $\sqrt{n}$ tasks in the direction towards the boundary, then communicates with its neighbor proceeding at the same time to the next group of $\sqrt{n}$ tasks, and so on. It is easy to see that the makespan in that case is $2n\sqrt{n}$. Thus the time complexity of the evaluation is $O(\sqrt{n})$ times worse than the optimal one.

For the second application we choose the join DAG, i.e. two level binary tree where tasks t_1, t_2 have common successor t_3. In [12] it is stated that ETF has a poor performance for this kind of DAG. Let the multiprocessor system consists of two connected processors p_1 and p_2. Set $\eta(t_1, t_3) = \eta(t_2, t_3) = 1$, $\mu(t_i) = 1,\ i = 1, 2, 3$ and $\tau(p_1, p_2) = \tau$. It is easy to check that both t_1 and t_2 will be scheduled on p_1 and p_2 at time 0. The task t_3 became available at time $1+\tau$ and can be scheduled either on p_1 or on p_2. Hence the makespan is $2+\tau$. If $0 \leq \tau \leq 1$ then the makespan is optimal, however if $\tau \gg 1$ it is much worse than optimal (equal to 3). In the limiting case $\tau \to \infty$, the ETF makespan is infinitely larger than the optimal one.

4 Conclusion and Work in Progress

The results obtained in the previous section are in agreement with the bound stated in [1] for unit execution time (UET) and unit communication time assumptions (UCT), namely

$$\omega_{ETF} \leq \left(3 - \frac{1}{n}\right) \cdot \omega_{opt} - 1 \quad .$$

Choose the unit to be $U = min_{t \in T}\ \mu(t)$. Then the UCT assumption implies that where the tasks t and t' are scheduled on processors p and p', respectively, $max_{t,t' \in T}\ \eta(t,t') \cdot \tau(p,p')$, is of order U. This assumption does not hold in practice if the tasks are the CPU operations like additions and/or multiplication, etc. and the communication between the nodes is organized by means of a message passing interface (MPI). The MPI usually has a large start-up overhead, i.e. the time to initiate the data transfer; this can be of order 10 to 1000 times U depending on the computer and the interface. To model this situation we have to assume that either messages are long or the link is slow. As was demonstrated by the examples above, however, this may result in the inefficient schedules. To improve the situation the DAG must be reorganized in a new one where each node is a cluster of tasks. For the new DAG, the UET, and UCT assumptions must hold.

Another problem with the ETF heuristic is that a pair of functions $\eta(t,t')$ and $\tau(p,p')$ is not sufficient to specify the communication overhead precisely, i.e. there is no parameter associated with the start up overhead ρ, which in many cases [5] is dominant over the product $\eta(t,t') \cdot \tau(p,p')$. The start up overhead is the processor activity and hence the processor is busy during this time interval. In case of ETF the communication is allowed to be completely overlapped with computations. In other words ETF simulates the nonblocking message passing with zero startup overhead, whereas real message passing is a mixture of a blocking communication for the period of time ρ and a nonblocking communication for the period of $\eta(t,t') \cdot \tau(p,p')$. In order to overcome this problem we suggest the following three step heuristic. The first two steps are used to produce a new DAG. Each node of the new DAG is a cluster of tasks from T such that a sum of their execution time is approximately equal to ρ. In first step Papadimitriou and Yannakakis's heuristic [9] is applied to the original DAG. At this step the number of processors is unspecified. As the second step, we apply the heuristic suggested by Thurimella and Yesha [10]. The number of processors at this step is chosen to be equal to that in the target multiprocessor system. The output is a new DAG with the properties specified above. In the last step we apply the ETF heuristic. The analysis of the schedules is a subject of the work in progress.

References

[1] J-J. Hwang, Y-C. Chow, F.D. Anger, and C-Y. Lee. Scheduling precedence graphs in systems with interprocessor communication times. *SIAM J. Comput.*, 18(2):244–257, 1989.

[2] A.A. Khan, C.L. McCreary, and M.S. Jones A comparison of multiprocessor scheduling heuristics. Technical Report comparison-ICPP-94, Dept. of Com-

puter Science and Engineering, Auburn University, 1994. Published in the Proceeding of the 8th Int. Parallel Processing Sym. - April, 1994.

[3] S. Manoharan and P. Thanisch. Assigning dependency graphs onto processor networks. *Parallel Computing*, 17(1):63–73, 1991.

[4] S. Manoharan and N.P. Topham. An assessment of assignment schemes for dependency graphs. *Parallel Computing*, 21(1):85–107, 1995.

[5] M.G. Norman, G. Chochia, P. Thanisch, and E. Issman. Predicting the performance of the diamond dag computation. Technical Report EPCC-TR-92-07, Edinburgh Parallel Computing Centre, 1992.

[6] M. Norman, P. Thanisch, and G. Chang. Partitioning DAG Computations. In W. Joosen and E. Milgrom, editors, *Parallel Computing: From Theory to Sound Practice*, pages 360–364, Amsterdam, 1992. IOS Press.

[7] M.G. Norman and P. Thanisch. Models of machines and computations for mapping in multicomputers. *Computing Surveys*, 25(3):263–302, 1993.

[8] C.H. Papadimitriou and J.D. Ullman. A communication-time tradeoff. *SIAM J. Comput.*, 16(4):639–646, 1987.

[9] C.H. Papadimitriou and M. Yannakakis. Towards an architecture-independent analysis of parallel algorithms. *SIAM J. Comput.*, 19:322–328, 1990.

[10] R. Thurimella and Y. Yesha. A scheduling principle for precedence graphs with communication delay. *J. of Computer and Software Engineering*, 2(2):165–176, 1994.

[11] B. Veltman, B.J. Lageweg, and J.K. Lenstra. Multiprocessor scheduling with communication delays. *Parallel Computing*, 16(2-3):173–182, 1990.

[12] T. Yang and A. Gerasoulis. DSC: Scheduling parallel tasks on an unbounded number of processors. *IEEE Trans. Paral. Distr. Systems*, 5(9):951–967, Sep 1994.