UET SCHEDULING WITH UNIT INTERPROCESSOR COMMUNICATION DELAYS

V.J. RAYWARD-SMITH
School of Information Systems, University of East Anglia, Norwich, NR4 7TJ, UK

Received 19 November 1985
Revised 21 August 1986

We consider the problem of scheduling a partially ordered set of unit execution time (UET) tasks on m > 1 processors where there is a communication delay of unit time between any pair of distinct processors. We show that the problem of finding an optimal schedule is NP-hard. A greedy schedule is one where no processor remains idle if there is some task available which it could process. We establish that the length of an arbitrary greedy schedule, \( \omega^c \), satisfies

\[
\omega^c \leq \left( 3 - \frac{2}{m} \right) \omega^c_{opt} - \left( 1 - \frac{1}{m} \right)
\]

where \( \omega^c_{opt} \) is the length of the optimal schedule. We define a generalized list schedule (a type of greedy schedule) and discuss anomalous behaviour of such schedules with respect to speed-up. The relevance of these results to the implementation of parallel languages is discussed.

Keywords: UET scheduling, list scheduling, communication delays, multiprocessor systems, algorithm analysis, NP-completeness, speed-up anomaly.

1. Introduction

Results from unit execution time (UET) scheduling theory have proved useful in the analysis of the implementation of parallel languages [3]. In particular, we are concerned with the scheduling of a finite set, \( T \), of tasks on \( m \) identical processors. Each task is assumed to have unit execution time and the tasks are partially ordered. This partial ordering, \( < \), is usually presented in the form of a directed acyclic graph (dag). A considerable amount of research has been done into this problem [see, e.g., 5, 9]. The application of these results to the implementation of parallel languages is justified by associating the tasks with the actions of the parallel program which will have been ordered using language constructs such as semaphores [11]. If we have a synchronized multiprocessor system which allocates actions to fixed length time slices and allows preemption, if required, one can assume a model of UET tasks. It is important to realise that in this application of scheduling theory we assume a significantly finer granularity of the tasks than is found in traditional applications.

In Section 2, we briefly review relevant known results concerning the above model. In Section 3, we introduce to the model an additional constraint, viz. an in-
terprocessor communication time delay. We assume a communication delay of unit time between any two distinct processors. Our interest in this is motivated by the above application of the theory; implementors of parallel languages have observed that communication delays can significantly degrade performance [1, 8, 16, 17]. Assuming we have unit time communication delays, we obtain a worst case bound for a greedy schedule. We also establish the NP-hardness of finding an optimal schedule in Section 4. Another common observation of implementors concerns the anomalous behaviour whereby an increase in the number of processors can sometimes degrade performance. In Section 5, we consider such anomalous behaviour with and without communication delays. In Section 6, we give a conclusion discussing the practical application of the results we have achieved.

2. Scheduling partially ordered UET tasks

Let \( T \) denote a set of UET tasks partially ordered by \(<\). We will schedule \( T \) on \( m \geq 1 \) processors, \( P_1, P_2, \ldots, P_m \). The length of a schedule is the total time taken

![Fig. 1(a). A dag.](image)

<table>
<thead>
<tr>
<th>P</th>
<th>a_1</th>
<th>a_2</th>
<th>a_3</th>
<th>a_4</th>
<th>a_5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>a_1</td>
<td>a_5</td>
<td>a_6</td>
<td>a_13</td>
<td>a_17</td>
</tr>
<tr>
<td>2</td>
<td>a_2</td>
<td>a_9</td>
<td>a_7</td>
<td>a_14</td>
<td>a_18</td>
</tr>
<tr>
<td>3</td>
<td>a_3</td>
<td>a_10</td>
<td>a_8</td>
<td>a_15</td>
<td>a_19</td>
</tr>
<tr>
<td>4</td>
<td>a_4</td>
<td>a_11</td>
<td>a_12</td>
<td>a_16</td>
<td></td>
</tr>
</tbody>
</table>

Fig. 1(b). Corresponding list schedule.
before every task is processed and all processors are idle.

If we process the tasks on just one processor, then we will have defined a total ordering, \( \sqsubseteq \), containing \( < \). This total ordering can be used to implement a list schedule [17] as below.

\[
\{\text{UET list scheduler}\}
\]

\[
\begin{align*}
U &:= T; \{ U \text{ is the set of unprocessed tasks}\} \\
c &:= 0; \{ c \text{ is the clock}\} \\
\textbf{while} \ U \neq \emptyset \ \textbf{do} \\
& \quad c := c + 1; \\
& \quad R := \{ t \in T \mid \exists t' \in U \text{ such that } t' < t \}; \\
& \quad \{ R \text{ is the ready set, i.e., the set of tasks which are available for processing}\} \\
& \quad i := 1; \\
& \quad \textbf{while} \ R \neq \emptyset \ \textbf{and} \ i \leq m \ \textbf{do} \\
& \quad \quad \text{min} := \{ t' \in R \mid t' \sqsubseteq t \text{ for all } t \in R \}; \\
& \quad \quad \text{allocate to processor } P_i \text{ for time } c \text{ the task } \text{min}; \\
& \quad \quad R := R - \{ \text{min} \}; \\
& \quad \quad U := U - \{ \text{min} \}; \\
& \quad \quad i := i + 1 \\
& \quad \textbf{endwhile} \\
& \textbf{endwhile}
\end{align*}
\]

Consider \( m = 4 \) processors and the task system of Fig. 1(a) totally ordered by \( a_i \sqsubseteq a_j \iff i < j \). The UET list scheduler then produces the schedule described by the Gantt chart of Fig. 1(b).

For any fixed \( m \), there always exists some total ordering, \( \sqsubseteq \), of the tasks such that the UET list scheduler gives an optimal schedule [3]. However, this is not the case if the tasks have differing processing times [18]. The task of finding an optimal schedule for the set \( T \) of UET actions on \( m \) processors is thus that of defining the required total ordering. Unfortunately, for an arbitrary number of processors, this problem is NP-hard [19]. For \( m = 2 \), the problem is solvable using the Coffman–Graham algorithm [6] but for fixed \( m \geq 3 \), no algorithms which ensure optimal schedules are yet known. In practice, a variety of heuristics are used.

Say \( \sqsubseteq \) and \( \sqsubseteq' \) are two total orderings both containing \( < \). If \( \omega \) denotes the length of the UET list schedule using \( m \) processors and total ordering \( \sqsubseteq \) and \( \omega' \) that obtained using \( m' \) processors and \( \sqsubseteq' \), then we have the following result.

**Theorem 1**

\[
\omega' \leq \left( 1 + \frac{m-1}{m'} \right) \omega.
\] (1)

This result follows immediately from a more general result cited in [5, 9]. In par-
ticular by setting \( m' = m \), it follows that for any list schedule
\[
\omega \leq \left(2 - \frac{1}{m}\right)\omega_{\text{opt}}
\]
(2)
where \( \omega_{\text{opt}} \) denotes the length of the optimal, i.e., shortest possible, schedule.

The level of a task in a dag is the length of the longest path from the task to any descendant task. Hu [13] proposed that the total ordering, \( \sqsubseteq \), should satisfy level\((t) \geq \text{level}(t') \) \( \Rightarrow t \sqsubseteq t' \). The total ordering used to schedule the tasks of Fig. 1 satisfied this constraint. This level strategy has been shown to be optimal for in-forests or out-forests [2]. The worst case performance of the level strategy is given by
\[
\omega_{\text{HU}} \leq \begin{cases} 
\frac{4}{3} \omega_{\text{opt}} & \text{if } m = 2, \\
\left(2 - \frac{1}{m-1}\right)\omega_{\text{opt}} & \text{if } m > 2.
\end{cases}
\]
(3)
This worst case bound established in [4] is known to be tight. Thus, this algorithm is a little better in worst case performance than an arbitrary schedule.

The Coffman-Graham algorithm is a refinement of Hu’s algorithm which is optimal for \( m = 2 \) processors. Lam and Sethi [14] have established a tight worst case bound
\[
\omega_{\text{CG}} \leq \left(2 - \frac{2}{m}\right)\omega_{\text{opt}} \text{ for } m \geq 2.
\]
(4)
However, for the implementation of parallel languages, this algorithm is too sophisticated. Scheduling algorithms used are very basic. For example, implementors of functional programming languages use a depth-first approach, a generalization of the familiar depth-first tree search algorithm. The worst case bound
\[
\omega_{\text{df}} \leq \left(2 - \frac{1}{m}\right)\omega_{\text{opt}}
\]
(5)
follows immediately from (2). It can be shown to be tight by considering \( m^2 - m + 1 \) start tasks (i.e., tasks with no ancestors) the ‘rightmost’ of which heads a chain of \( m \) tasks. A breadth first strategy on the same example shows a similar worst case performance.

3. UET scheduling with interprocessor communication delays

We amend our basic scheduling model to include a time delay of \( t = 1 \) units whenever we transmit information from processor \( P_i \) to processor \( P_j \) (\( i \neq j \)). We thus have an additional constraint that if a task, \( t \), is scheduled on processor \( P_i \) at time \( c \), then no child of \( t \) can be processed on \( P_j \neq P_i \) until time \( c + 2 \). For the dag of Fig.
1(a) and four processors, the schedule of Fig. 1(b) is no longer feasible. An optimal schedule is given by the Gantt chart of Fig. 2.

We say a schedule is greedy if no processor remains idle if there is a task available which it can process. Assuming communication delays, we denote the length of an optimal schedule by $\omega_{opt}^c$. Since the tasks have unit execution time, it is easy to construct from any optimal schedule, a greedy schedule of the same length. Thus, we can restrict our attention to such greedy schedules.

**Theorem 2**

$$\omega_{opt} \leq \omega_{opt}^c \leq 2\omega_{opt} - 1.$$  \hfill (6)

**Proof.** Interprocessor communication delays add additional constraints to the scheduling problem so $\omega_{opt}^c \geq \omega_{opt}$ follows immediately. The second inequality follows from the observation that from any optimal schedule which ignores communication delays, we can obtain a valid schedule which allows for such delays by rescheduling actions processed at time $i$ to be at time $2i - 1$ and having all processors idle at time $2, 4, \ldots$.

It is easy to construct examples of $n$ actions where $\omega_{opt} = \omega_{opt}^c (< \text{ is empty})$ and where $\omega_{opt}^c = 2\omega_{opt} - 1$ (the first $m$ actions are all parents of each of the second $m$ actions which are all parents of each of the third $m$ actions, etc.). Thus, the bounds of Theorem 2 are the best possible.

Given a total ordering, $\sqsubseteq$, of the tasks, it is possible to allocate the tasks using the following (generalized) list scheduler. This should be compared with the UET list scheduler of Section 2.

```
{Generalized UET list scheduler}
U := T; \{U is the set of unprocessed tasks\}
c := 0; \{c is the clock\}
while U \neq \emptyset do
    c := c + 1;
    for i := 1 to m do
```
$R_i := \{ t \in U \mid t \text{ has no parents or each parent of } t \text{ has been scheduled either on processor } P_i \text{ at time } < c \text{ or on processor } P_j \neq P_i \text{ at time } < c - 1 \};$

$\{R_i \text{ is the ready set for processor } P_i \};$

if $R_i \neq \emptyset$ then
  \[ \min := \{ t' \in R_i \mid t' \sqsubset t \text{ for all } t \in R_i \}; \]
  allocate to processor $P_i$ for time $c$ the task $\min$;
  $U := U - \{ \min \}$
endif

endfor

endwhile

Fig. 2 can be obtained from a generalized list schedule using the total ordering
\[
a_1 \sqsubset a_2 \sqsubset a_3 \sqsubset a_5 \sqsubset a_4 \sqsubset a_{10} \sqsubset a_{11} \sqsubset a_6 \sqsubset a_9 \sqsubset a_7 \sqsubset a_{12} \sqsubset a_8 \sqsubset a_{13} \sqsubset a_{14} \sqsubset a_{15} \sqsubset a_{16} \sqsubset a_{17} \sqsubset a_{18} \sqsubset a_{19}.\]

Clearly, any generalized list schedule is an example of a greedy schedule.

Given any schedule, $s$, of length $\omega$, we define $T_s(i, j)$, $1 \leq i \leq m$, $0 \leq j < \omega$ to be the sequence of tasks processed by $P_i$ at time $\geq j$. It is then easy to establish the following results.

Fig. 3.
Lemma 1. If $P_i$ is idle at time $j$ but could process some element, $a$, currently processed at time $>j$, then a valid schedule can be constructed from $s$ by allocating $a$ to $P_i$ at time $j$.

Lemma 2. If $P_i$ is idle at time $j$ but could process a task, $a$, processed by $P_k$ at time $j$, then a valid schedule can be constructed from $s$ by interchanging $T_s(i,j)$ and $T_s(k,j)$.

Now, if $s$ is an optimal schedule, we can transform it into a greedy schedule using Lemma 1 repeatedly. Not every greedy schedule is a generalized list schedule (see Fig. 3). A greedy schedule only has to satisfy "$P_i$ idles at time $j$ implies there is no $a \in T$ scheduled at time $>j$ which $P_i$ could have processed at time $j$". A generalized list schedule has to satisfy "$P_i$ idles at time $j$ implies $R_i=\emptyset$, i.e., there is no $a \in T$ scheduled at time $>j$ which $P_i$ could have processed at time $j$ nor is there any $b \in T$ scheduled at time $j$ on $P_k$ ($k>i$) which $P_i$ could have processed at time $j$". However, an optimal greedy schedule can always be transformed into an optimal generalized list schedule by repeated use of Lemma 2.

Assuming communication delays, let $\omega_g^c$ be the length of a greedy schedule of $T$ on $m$ processors and $\omega_g^c'$ be the length of some other greedy schedule on $m'>1$ processors. We now prove the following result.

**Theorem 3**

$$\omega_g^c' \leq \left[2 + \frac{m-2}{m'}\right] \omega_g^c - \left[1 - \frac{1}{m'}\right].$$

**Proof.** To prove this result we need the concept of a *layered digraph*. A layered digraph is one where every node is either at depth 0 or is at depth $k>0$ and has all

![Layered Digraph](image)

Fig. 4. A (5,2)-layered digraph.
its parents at depth $k - 1$. A layer of the digraph then comprises all the nodes of some given depth. A layered digraph will be called an $(n, m)$-layered digraph iff (i) it has $n$ layers, (ii) all terminal nodes are in the $n$th layer, i.e., at depth $n - 1$, and (iii) $0 \leq m \leq n - 1$ layers are such that every node in the layer has more than one parent.

Fig. 4 gives an example of a $(5, 2)$-layered digraph.

The importance of an $(n, m)$-layered digraph lies in the following result easily proved by induction on $n$.

**Lemma 3.** With unit interprocessor communication delays, the optimal time to schedule a set of UET tasks ordered as an $(n, m)$-layered is $\geq n + m$.

Now, consider an arbitrary dag of UET tasks and let $\omega^c_g$ be the length of an arbitrary greedy schedule on $m'$ machines. Let $c_1 < c_2 < \cdots < c_r$ be the times in the schedule when there is some idle time on one or more processors. If $c$ is such an idling time, then either

1. $c$ is a dormant time, i.e., no processors are active. In this case, every task processed after $c$ must have at least two ancestors processed at time $c - 1$, or
2. $c$ is not a dormant time, i.e., at least one processor is active. In this case, every task processed after $c$ must have at least one ancestor processed at time $c$ or at time $c - 1$.

Let $c_{\lambda_1} < c_{\lambda_2} < \cdots < c_{\lambda_s}$ be dormant times. Now, consider the sequence of times $c_{\lambda_1} - 1 < c_{\lambda_1} < c_{\lambda_2} - 1 < c_{\lambda_2} < \cdots < c_{\lambda_s} - 1 < c_{\lambda_s} < \omega^c_g$. Let $C$ denote the times in this sequence. Since a greedy schedule cannot have two adjacent dormant times nor start nor end with a dormant time, all the times in this sequence are distinct. We can construct a layered digraph with terminal node some task scheduled at $\omega^c_g$. That task must have two ancestor tasks scheduled at time $c_{\lambda_s} - 1$. In turn, both of these tasks must have two ancestor tasks scheduled at time $c_{\lambda(s-1)} - 1$. We continue in this way to construct a subset of $T$ which will be ordered as an $(s + 1, s)$-layered digraph. Now, we consider the times $C'$ which occur in the sequence $c_1 < c_2 < \cdots < c_r$ but not in $c_{\lambda_1} - 1 < c_{\lambda_1} < c_{\lambda_2} - 1 < c_{\lambda_2} < \cdots < c_{\lambda_s} < \omega^c_g$. There must be at least $r - (2s + 1)$ of them. We can thus select $\lceil (r - (2s + 1))/2 \rceil$ times in $C'$ so that if $c$ is selected, then $c - 1$ is not selected. This new set of times is denoted by $C''$. For each $c \in C''$, we can add a new layer in the digraph since for each task scheduled at time $> c$ we know by (2) above that it must have an ancestor processed at time $c$ or $c - 1$. Thus, we can construct a subset of $T$ which will be ordered as an $(s + \lceil (r - 2s - 1)/2 \rceil + 1, s)$-layered digraph.

From Lemma 3, $\omega^c_g \geq 2s + 1 + (r - 2s - 1)/2$. So

$$2\omega^c_g - 3s - 1 \geq r - s.$$
Thus

\[ p = \frac{m'}{m} \text{ processor idle time + processor non-idle time} \]

\[ \leq m's + (m' - 1)(r - s) + m\omega_g^c \]

\[ \leq m's + (m' - 1)(2\omega_g^c - 3s - 1) + m\omega_g^c \]

\[ = s(3 - 2m') + \omega_g^c(2m' + m - 2) - m' + 1. \]

Result (7) is an a priori worst case bound; result (8) is an a posteriori bound. As a corollary to the theorem and the observation that an optimal schedule can be obtained from a greedy approach, we have the result that for any greedy schedule

\[ \omega_g^c \leq \left( 2 + \frac{m - 2}{m'} \right) \omega_g^c - \left( 2 - \frac{3}{m'} \right) s - \left( 1 - \frac{1}{m'} \right). \]

Since \( s \geq 0 \) and \( m' > 1 \), \((2 - 3/m')s \geq 0\) and hence the result of the theorem can be established.

Result (7) is an a priori worst case bound; result (8) is an a posteriori bound. As a corollary to the theorem and the observation that an optimal schedule can be obtained from a greedy approach, we have the result that for any greedy schedule

\[ \omega_g^c \leq \left( 2 + \frac{m - 2}{m'} \right) \omega_g^c - \left( 2 - \frac{3}{m'} \right) s - \left( 1 - \frac{1}{m'} \right). \]

The multiplicative constant in (9) is the best possible. Consider the system of tasks described in Fig. 5. The optimal schedule of these tasks on \( m \geq 2 \) processors will process all of the \( a_i \) on one processor and all the \( b_j \) on another. The total time taken would then be \( km \). With a poor schedule, we could alternate between \( a \)'s and \( b \)'s only using a single processor and yet still have a greedy schedule. This would take time \( 2km - 2 \). Now, assume we also have \( km^2 - 2km + 2 \) independent tasks. With
$m$ processors, the optimal schedule will still take time $km$. The worst greedy schedule will process all the independent tasks first and (assuming $m > 2$) will only embark upon task $a_i$ in time interval $km - 2k + 1$. If we then alternate between $a$'s and $b$'s, the total schedule will take $km - 2k + 2km - 2 = 3km - 2k - 2$ time. Thus, the ratio $\omega_G / \omega_{opt}$ is $3 - 2/m - 2/km$ and as $k \rightarrow \infty$ we achieve the desired ratio.

4. The complexity of the problem

The decision problem associated with scheduling partially ordered UET tasks assuming unit interprocessor communication delays can be stated as follows.

Scheduling partially ordered UET tasks with unit interprocessor communication delays (SPOUTC).

Instance. A set, $T$, of tasks partially ordered by $<$, a number $m > 0$ of processors and a time limit, $b$.

Question. Is there a valid schedule of $T$ on the $m$ processors which allows for unit interprocessor communication delays and has length $\leq b$? More formally, is there an injection $s^c : T \rightarrow \{1, 2, \ldots, m\} \times \{1, 2, \ldots, b\}$ such that $\forall a, b \in T, a < b$ implies

either $\delta_1(s^c(a)) = \delta_1(s^c(b))$ and $\delta_2(s^c(a)) < \delta_2(s^c(b))$

or $\delta_1(s^c(a)) \neq \delta_1(s^c(b))$ and $\delta_2(s^c(a)) < \delta_2(s^c(b)) - 1$?

In the above, $\delta_1, \delta_2$ are the standard projection functions. In this section we establish that the above problem is NP-complete and hence that UET scheduling with unit communication delays is an NP-hard problem. Clearly, SPOUTC $\in$ NP since we could guess a function $s : T \rightarrow \{1, 2, \ldots, m\} \times \{1, 2, \ldots, b\}$ and then in polynomial time check if it was an injection satisfying the necessary conditions.

Thus, to establish that SPOUTC is NP-complete, we need to find an NP-complete problem $\Pi$ and a polynomial transformation $\Pi \approx$ SPOUTC.

Not surprisingly, the problem we choose is the UET scheduling problem without interprocessor communication delays. The associated decision problem is

Scheduling partially ordered UET tasks (SOUT).

Instance. A set, $T$, of non-independent tasks partially ordered by $<$, a number, $m$, of processors, $0 < m \leq |T|$ and a time limit $b \leq |T|$.

Question. Is there a valid schedule of $T$ on the $m$ processors which takes time $\leq b$? More formally, is there an injection $s : T \rightarrow \{1, 2, \ldots, m\} \times \{1, 2, \ldots, b\}$ such that $\forall a, b \in T, a < b$ implies $\delta_2(s(a)) < \delta_2(s(b))$?

The NP-completeness of SOUT is proved in [19]. It is important to note that none of the tasks are independent and thus any instance will have an encoding of length order $\Omega(n)$ where $n = |T|$.
Let $I= (T, <, m, b)$ be an instance of SPOUT. We construct an instance $f(I)$ of SPOUTC as follows. The tasks of $f(I)$ comprise the tasks $T$ partially ordered by $<$ together with the set, $V$, of $(b + 1)(m + 1) + b$ tasks partially ordered as in Fig. 6. In this dag, for each $0 \leq i < b$ and for each $1 < j \leq m + 1$, $a_{ij}$ is a parent of $a_{i+1j}$ and $a_{i1}$ is a parent of each $a_{i+j}$. Also, for each $0 \leq i \leq b$, $a_{i1}$ is a parent of $c_{i+1}$ which, in turn, is a parent of $a_{i+11}$. The number of processors in $f(I)$ is set as $m + 1$ and the time limit becomes $2b + 1$. 

![Diagram](image.png)
Since any encoding of $I$ has length $\Omega(n)$ and since $f(I)$ can be encoded in length $O(n^4)$, the length of an encoding of $f(I)$ is polynomial in the length of an encoding of $I$. We claim, moreover, that $I$ is a YES-instance of SPOUT if $f(I)$ is a YES-instance of SPOUTC. Reference to Fig. 7 will aid the reader in the understanding of the following proof.

If $I \in Y_{\text{SPOUT}}$, then there exists an injection $s : T \rightarrow \{1, 2, \ldots, b\}$ such that $\forall a, b \in T$, $a < b$ implies $\delta_2(s(a)) < \delta_2(s(b))$. We can then define $s^c : T \rightarrow \{1, 2, \ldots, m + 1\} \times \{1, 2, \ldots, 2b + 1\}$ by $s^c(a) = (\delta_1(s(a)) + 1, 2\delta_2(s(a)))$. Then $s^c$ is clearly injective and $\forall a, b \in T$, $a < b$ implies $\delta_2(s^c(a)) < \delta_2(s^c(b)) - 1$. Now we extend $s^c$ to an injection $s^c : T \cup V \rightarrow \{1, 2, \ldots, m + 1\} \times \{1, 2, \ldots, 2b + 1\}$ by defining $s^c(a_{ij}) = (j, 2i + 1)$ and $s^c(c_i) = (1, 2i)$. Then, it is routine to check that $\forall a, b \in T \cup V$, $a < b$ implies either $\delta_1(s^c(a)) = \delta_1(s^c(b))$ and $\delta_2(s^c(a)) < \delta_2(s^c(b))$ or $\delta_1(s^c(a)) \neq \delta_1(s^c(b))$ and $\delta_2(s^c(a)) < \delta_2(s^c(b)) - 1$. Thus $f(I)$ is a YES-instance of SPOUTC.

If $f(I)$ is a YES-instance of SPOUTC, then there exists an injection $s^c : T \cup V \rightarrow \{1, 2, \ldots, m + 1\} \times \{1, 2, \ldots, 2b + 1\}$ satisfying the condition that $\forall a, b \in T \cup V$, $a < b$ implies either $\delta_1(s^c(a)) = \delta_1(s^c(b))$ and $\delta_2(s^c(a)) < \delta_2(s^c(b))$ or $\delta_1(s^c(a)) \neq \delta_1(s^c(b))$ and $\delta_2(s^c(a)) < \delta_2(s^c(b)) - 1$. Within $V$ we have a chain of $2b + 1$ tasks, viz. $a_{01} < a_{11} < a_{2} \cdots < a_{b-1} < a_{b} < a_{b+1}$. From this we can deduce that $\delta_1(s^c(a_{01})) = \delta_1(s^c(c_1)) = \cdots = \delta_1(s^c(c_{b+1}))$ and $\delta_2(s^c(a_{01})) = 2i + 1$, $\delta_2(s^c(c_i)) = 2i$. Thus one processor is completely devoted to processing these tasks. Without loss of generality, we can assume this is processor 1. Now, for any $0 \leq i < b - 1$ and for any $1 < j < m + 1$, we also have a chain $a_{ij} < a_{i+1,j} < a_{i+2,j}$. Task $a_{i+1,j}$ cannot be processed on processor 1. Thus we deduce $\delta_2(s^c(a_{i+1,j})) = 2i + 3$. Hence, at time $2i + 1$, $1 \leq i \leq b$, all processors are dedicated to processing tasks $a_{11}, a_{12}, \ldots, a_{m+1}$. This result also holds for $i = 0$ since tasks $a_{01}, a_{02}, \ldots, a_{0,m+1}$ must be processed at time 1. Hence, we can deduce that for all $a \in T$, $\delta_1(s^c(a)) \in \{2, 3, \ldots, m + 1\}$ and $\delta_2(s^c(a)) \in \{2, 4, \ldots, 2b\}$. We now define $s(a) = (\delta_1(s^c(a)) - 1, \delta_2(s^c(a)) / 2)$. $s : T \rightarrow \{1, \ldots, m\} \times \{1, \ldots, b\}$ is injective and satisfies $a < b$ implies $\delta_2(s(a)) < \delta_2(s(b))$ as required. Hence $I$ is a YES-instance of SPOUT.

We have established a polynomial transformation from a known NP-complete problem, SPOUT, to SPOUTC. Hence we can deduce the following.

**Theorem 4.** SPOUTC is NP-complete.

This result implies that we are unlikely to find a polynomial time algorithm to find minimal length schedules of UET tasks in the context of unit interprocessor communication delays.

5. Anomalous behaviour

Implementors of parallel languages have observed that an increase in the number of processors sometimes degrades the system’s performance. This is usually (but not
always) because of the scheduling algorithm being used. This anomalous behaviour has been well researched in scheduling theory [3, 12]. Even without communication delays, a UET list scheduler can perform slower on \( m + 1 \) processors than it does on \( m \). For example, consider the dag of Fig. 1. If, as usual, we use the list scheduler based on \( a_i \preceq a_j \) iff \( i < j \), but now on five processors, we obtain the schedule of Fig. 8 - it is of greater length than the schedule of Fig. 1(b) when we used only four processors. Moreover, the list scheduler used could have been constructed using a level strategy. It is thus possible to get anomalous behaviour using Hu's level scheduling algorithm.
or a depth-first strategy. However, a breadth-first schedule of UET tasks will never give rise to speed up anomalies [3]. Anomalous behaviour of list scheduling can be obtained even when the dag is a tree. The tree in Fig. 9 with total ordering defined by the indices gives rise to a schedule of length 4 on four processors but of length 5 of five processors. There exist dags and total orderings such that an increase from three to four processors causes a speed up anomaly (see Fig. 10) but it is known that a speed up anomaly can never occur when increasing the number of processors from two to three [3].

With unit interprocessor communication delays and even quite simple dags, a
generalized list scheduling algorithm can give rise to speed up anomalies. In Figs. 11 and 12 we give two examples to illustrate such anomalous behaviour when the number of processors is increased from two or three. The example of Fig. 11 can easily be generalized to give an example of a speed up anomaly arising from an increase from $m$ to $m+1$ processors ($m \geq 2$). The ordering used in Fig. 11 could be obtained from either a level ordering or a breadth-first ordering. That of Fig. 12 illustrates that a generalized depth-first schedule can also give rise to speed up anomalies. Finally, in Fig. 13, we show that anomalous behaviour is possible when the dag is a tree.

6. Conclusion

The model we have proposed has applications in the implementation of parallel languages. We have established that within this model optimal scheduling is NP-
hard. This means that we are unlikely to be able to produce an optimal scheduling algorithm even if the total structure of the dag is known in advance. With a parallel program, the structure of the dag only becomes apparent as the program is executed. Thus, the situation is even worse. Any implementor may as well use a greedy algorithm and will probably use a generalized list scheduling algorithm. What the implementor needs is a total ordering which gives a reasonably good (i.e., near optimal) schedule in most cases. Additionally, this total ordering cannot depend upon knowledge of the complete dag. Level ordering thus appears to be ruled out but it is possible to use some heuristic to give an approximation to it; the implementor should be able to estimate the level of an action. However, the more sophisticated Coffman–Graham ordering is certainly unrealistic. A depth-first ordering will probably be used but perhaps be slightly amended to take some account of (an estimate of) task levels.

Although an optimal schedule cannot be realistically achieved, we do know that any arbitrary greedy schedule does no worse than something less than three times the optimal. This is a reassuring result. Although communication delays have increased the ratio $\omega/\omega_{opt}$, it is still reasonably bounded. Less reassuring are the results concerning anomalous behaviour. It appears that communication delays make speed up anomalies much more likely and with any practical ordering we might use, such anomalies can arise. This phenomenon appears to be unavoidable.

Scheduling theory is well established but there is very little work which allows for interprocessor communication costs/delays. In [15], results are given concerning the assignment of non-UET, independent tasks with associated communication costs whenever two tasks are assigned to different processors. Other work in this area includes [7, 10]. The model proposed in this paper can clearly be generalized to arbitrary length tasks and arbitrary communication delays. Finding optimal schedules will remain NP-hard. Heuristics will need to be analysed, not just for worst case performance, but also for average case performance.

Acknowledgements

The author wishes to acknowledge useful discussion he has had with Professor F.W. Burton, University of Colorado at Denver and Dr. G.P. McKeown of the University of East Anglia. Comments made by the referees were particularly helpful.

References


