An $O(m' \log m')$ Algorithm for a Chain-like Task System

Gilbert H. Young and Chi-Lok Chan
Computer Science Department
The Chinese University of Hong Kong

Abstract

The optimal allocation of a chain-like task system on a chain-like network computers has been studied in a number of papers [1][2][3][4]. The best known algorithm has a time complexity of $O((m'-n)^2 n)$ if $m-n = O(\sqrt{\log n})$; otherwise $O(m' \log m' + m'(m'-n))$ where m' and n denote the number of un-mergeable modules and the number of processors respectively[4]. The space complexity of Hsu's algorithm and Young and Chan's algorithm are $O((m'-n)n)$ and $O(m'(m'-n))$ respectively. In this paper, we generalize the decision algorithm discussed in [4] and propose an $O(m' \log m')$ algorithm for the optimization problem based on the generalized decision test. This is a searching strategy based on a sorted matrix[5][6][7]. This algorithm is better than all existing algorithms except $m'-n = o(\sqrt{\log n})$. The space complexity is $O(m')$. Furthermore, the optimal schedules obtained use the minimal number of processors so that they are the optimally load balanced optimal schedules.

1 Introduction

The problem we investigate in this paper is the allocation of a chain-like task system on the chain-like network computers which was first presented by Bokhari [1]. A chain-like task system consists of m modules scheduled on n processors. Each module is associated with an execution time and communicates with its neighboring modules with a communication cost. The maximum rate of processing is now determined by the processor (the *bottleneck processor*) that takes the longest amount of time (*schedule length*) to perform its tasks. The optimal allocation problem is to find a schedule in which the bottleneck processor has the minimal execution time. This problem is first proposed by Bokhari and he gave an algorithm with time complexity of $O(m^3 n)$ [1]. Sheu and Chiang showed that a more general assignment problem can be solved in $O(\min\{m,n\}m^2)$[2]. Hsu presented a two phase algorithm [3]. The first phase, merge phase, is used to get an un-mergeable chain-like task system with m' modules. If $m' \leq n$, each module is assigned to one processor; otherwise, the second phase, assignment phase, can be used to find an optimal schedule with time complexity of $O(m + (m'-n)^2 n)$. The decision version, whether there exist a schedule with schedule length no greater than a given constant $\mathcal{K}$, is first presented by Young and Chan [4]. The time and space complexities are $O(m')$. Moreover, they gave an $O(m + m' \log m' + m'(m'-n))$ algorithm for solving the optimization problem. In this paper, we generalize the decision algorithm presented in [4]. The time and space complexities remain $O(m')$. Besides, we propose another algorithm for obtaining an optimal schedule based on searching an element in a sorted matrix[5][6][7]. The time and space complexities of this algorithm are $O(m + m' \log m')$ and $O(m)$ respectively. In addition, the optimal schedules obtained use the minimal number of processors.

The remainder of this paper is organized as follows: section 2 states the problem assumptions and notations used in this paper. In section 3, a generalized decision test is presented. Section 4 proposes an $O(m + m' \log m')$ algorithm based on a sorted matrix. Finally, some conclusions will be drawn in section 5.

2 Problem Assumptions and Notations Definition

- A chain-like task system consists of m modules, $t_1, t_2, \ldots, t_m$, scheduled on n processors. Execution time of module i is denoted as E_i and communication time of adjacent modules i and $i+1$ is denoted as $C_{i,i+1}$ $(0 \leq i \leq m)$. All of execution times and communication times are positive except that $C_{0,1}$ and $C_{m,m+1}$ equal zero.

- Adjacent modules must be assigned to adjacent processors or the same processor, which is called *contiguity constraint*[1]. If adjacent modules are assigned to the same processor, their communication (intra-communication) time is assumed to be zero.

- The completion time of a module i,denoted as CMP_i, is equal to $C_{i-1,i} + E_i + C_{i,i+1}$. The completion time of modules $i, i+1, \ldots, j$, denoted as $CMP_{i \oplus i+1 \oplus \cdots \oplus j}$, is equal to $C_{i-1,i} + \sum_{k=i}^{j} E_k + C_{j,j+1}$. The completion time of a processor j, $CP_j(S)$, is defined as the completion time of all modules scheduled on this processor in S.

- The *schedule length* of a schedule, $\mathcal{L}(S)$, is defined as the maximum completion time of all processors in S, ie.

  $$\mathcal{L}(S) = \max_i \{CP_i(S)\}$$

  The corresponding processor is called *bottleneck processor*.

- A schedule is said to be *feasible* if it satisfies the contiguity constraint and $\mathcal{F}$ is the set of all feasible schedules. A schedule, S, is called a *feasible length-$\mathcal{K}$ schedule*[4] if

  $$\mathcal{L}(S) = \max_{1 \leq i \leq n} \{CP_i(S)\} \leq \mathcal{K}$$

  where $\mathcal{K}$ is a given constant. $\mathcal{F}_{\mathcal{K}}$ is the set of all feasible length-$\mathcal{K}$ schedules.

- An *optimal schedule*, S°, is a feasible schedule with minimal schedule length, $\mathcal{K}^o$, ie. $\exists S^\circ \in \mathcal{F}_{\mathcal{K}^\circ}$

  $$\mathcal{L}(S^\circ) \leq \mathcal{L}(S'), \qquad \forall S' \in \mathcal{F}$$

- Two modules, t_i and t_{i+1} are *un-mergeable*[3] if

  $$C_{i,i+1} < E_i + C_{i-1,i} \quad \text{and} \quad C_{i,i+1} < E_{i+1} + C_{i+1,i+2}$$

- A chain-like task system is said to be un-mergeable if all pairs of modules are un-mergeable. Hsu presented an $O(m)$ algorithm to obtain an un-mergeable chain-like task system.

3 Generalized Decision Test

By Young and Chan's decision algorithm[4], we can decide whether there exists a feasible length-$\mathcal{K}$ schedule, where $\mathcal{K}$ is a given constant. In other words, we can check whether $\mathcal{K}$ is greater than the optimal schedule length $\mathcal{K}^o$. By modifying this algorithm, we can decide whether there exists a feasible schedule with schedule length strictly less than $\mathcal{K}$, or equivalently, check whether a constant $\mathcal{K}$ is less than $\mathcal{K}^o$. The modified algorithm is identical to Young and Chan's decision algorithm [4] except that all ">" symbols (in line 3 and 8) are replaced by "$\geq$".

Algorithm 1 *(Greedy algorithm)*

Input: *m' un-mergeable modules with their execution times and communication times, n processors and $\mathcal{K} \in R^+$*

Output: *'Yes' and gives a feasible length-$\mathcal{K}_{-\delta}$ schedule S if there exists one; otherwise, 'No'.*

(1) $j = 1, CP = 0$
(2) for $i = 1$ to m'
(3)   if $CP + E_i + C_{i,i+1} - C_{i-1,i} \geq \mathcal{K}$ // modified!
(4)   if $j = n$
(5)   return 'No'
(6)   else
(7) $CP = C_{i-1,i} + E_i + C_{i,i+1}$
(8)   if $(CP \geq \mathcal{K})$ // modified!
(9)   return 'No'
(10) $j = j + 1$
(11)   else
(12) $CP = CP + E_i + C_{i,i+1} - C_{i-1,i}$
(13) $S[i] = j$
(14) return 'Yes' and S

Theorem 1 *For m' un-mergeable modules scheduled on n processors, Algorithm 1 returns a schedule S with schedule length less than $\mathcal{K}$ iff there exists one.*

Proof: Similar to the proof of the decision algorithm in [4]. □

Theorem 2 *For m' un-mergeable modules scheduled on n processors and any constant $\mathcal{K}^o$, $\mathcal{K}^o$ is the optimal schedule length iff there exists a feasible length-$\mathcal{K}^o$ schedule and there does not exists any schedule with schedule length less than $\mathcal{K}$.*

Proof:

$\mathcal{K}^o$ is an optimal schedule length

$\Leftrightarrow$ $\exists S \in \mathcal{F}_{\mathcal{K}^o}$ s.t. $\mathcal{L}(S) = \mathcal{K}^o$ and $\nexists S' \in \mathcal{F}$ s.t. $\mathcal{L}(S') < \mathcal{K}^o$ □

As a result, we can compare any constant, $\mathcal{K}$ with the optimal schedule length, $\mathcal{K}^o$ by Algorithm 1 and Young and Chan's decision algorithm [4]. For the rest of the paper, the following generalized decision test is used for the optimization problem,

Young and Chan's decision algorithm[4]	Algorithm 1	Result
YES	YES	$\mathcal{K} > \mathcal{K}^o$
YES	NO	$\mathcal{K} = \mathcal{K}^o$
NO	NO	$\mathcal{K} < \mathcal{K}^o$

The generalized decision test has a time and space complexities of $O(m')$. If the optimal schedule length is known, we can obtain the optimal schedule by using Young and Chan's decision algorithm [4] (the optimal schedule is a feasible length-$\mathcal{K}^o$ schedule). In addition, any feasible length-$\mathcal{K}$ schedule obtained has the following properties,

Lemma 3 *By comparing the feasible length-$\mathcal{K}$ schedule S return by Young and Chan's decision algorithm [4] and any feasible length-$\mathcal{K}$ schedule S', the total number of modules have been scheduled up to p_j $(1 \le j \le n)$ of S is no less than that of S', ie.*

$$M(S,j) \ge M(S',j), \qquad \forall j, 1 \le j \le n, \forall S' \in \mathcal{F}_{\mathcal{K}}$$

where $M(S,j)$ denotes the total number of modules scheduled on $p_1, p_2, \ldots, p_j$ in S.

Proof: Suppose on the contrary, there exists a feasible schedule S' such that there exists a processor, j and $M(S,j) < M(S',j)$ where $1 \le j \le n$. Let k be the smallest index of processor in S' and $M(S,j) < M(S',j)$, n_{k-1} and n_k (also n'_{k-1} and n'_k) be the total number of modules scheduled up to p_{k-1} and p_k in S (S') respectively $(M(S,k-1) = M(S',k-1) = 0$ if $k=1)$. Then

$$n_{k-1} = M(S,k-1) \ge M(S',k-1) = n'_{k-1} \quad \text{and} \quad n'_k > n_k$$

For S, $t_{n_{k-1}+1}, \ldots, t_{n_k}$ are scheduled on p_k and t_{n_k+1} cannot be scheduled on p_k (line 3 in Algorithm 1 [4]), ie.

$$CMP_{n_{k-1}+1\oplus\cdots\oplus n_k \oplus n_k+1} > \mathrm{K}$$

$$\begin{aligned} CMP_{n'_{k-1}+1\oplus\cdots\oplus n'_k} &\ge CMP_{n_{k-1}+1\oplus\cdots\oplus n'_k} && ([4],\ n_{k-1} \ge n'_{k-1}) \\ &\ge CMP_{n_{k-1}+1\oplus\cdots\oplus n_k+1} && ([4],\ n'_k \ge n_k+1) \\ &> \mathcal{K} \end{aligned}$$

which is contradictory to the fact that S' is a feasible length-$\mathcal{K}$ schedule. □

Theorem 4 *If S is a schedule return from Young and Chan's decision algorithm [4], S is a feasible length-$\mathcal{K}$ schedule using the minimal number of processors.*

Proof: Suppose S uses n^* processors and $\exists S' \in \mathcal{F}_{\mathcal{K}}$ such that S' uses only n' processors with $n' < n^*$,

$$\begin{aligned} m' &= M(S',n') && \text{(By definition)} \\ &\le M(S,n') && \text{(By Lemma 3)} \\ &< M(S,n^*) && (n' < n^*) \\ &= m' \end{aligned}$$

It leads to contradiction. □

By Theorem4, we can get an optimal schedule which uses the minimal number of processors.

4 Searching Algorithm Based on a Sorted Matrix

In [4], Young and Chan proposed a searching based algorithm for finding an optimal schedule. All possible schedule lengths are stored in a set Λ and

$$\Lambda = \{CMP_{i\oplus\cdots\oplus j} : 1 \le i \le j \le m' \text{ and } j - i \le m' - n\}$$

The optimal schedule length, $\mathcal{K}^o$, is an element in Λ such that $\mathcal{K}^o$ is the minimal value and there exists a feasible length-$\mathcal{K}^o$ schedule. In this section, We are going to use another representation of all possible schedule lengths and give an $O(m + m' \log m')$ algorithm for obtaining an optimal schedule.

4.1 Sorted Matrix

By using the monotonic increasing property of sub-chains of un-mergeable modules ([4]), a partial order relationship of all possible schedule lengths can be established, $\forall i \leq k \leq j, j - i \leq m' - n$,

$$\begin{array}{lrcl} & CMP_{i\oplus\cdots\oplus j} & \leq & CMP_{i\oplus\cdots\oplus k} \\ \text{and} & CMP_{i\oplus\cdots\oplus j} & \leq & CMP_{k\oplus\cdots\oplus j} \end{array}$$

The schedule length can be arranged as a $m' \times m'$ matrix, S, such that

$$s_{i,j} = \begin{cases} 0 & \text{if } i > j \\ \infty & \text{if } i < j - m' + n \\ CMP_{i\oplus\cdots\oplus j} & \text{otherwise} \end{cases}$$

ie.

$$S = \begin{pmatrix} CMP_1 & \cdots & CMP_{1\oplus\cdots\oplus m'-n+1} & \infty & \cdots & \infty \\ 0 & CMP_2 & \cdots & CMP_{2\oplus\cdots\oplus m'-n+2} & \ddots & \vdots \\ 0 & 0 & \ddots & \ddots & \ddots & \infty \\ \vdots & 0 & \ddots & \ddots & \ddots & CMP_{m'-2\oplus\cdots\oplus m'} \\ 0 & \vdots & \ddots & 0 & \ddots & \vdots \\ 0 & 0 & \cdots & 0 & 0 & CMP_{m'} \end{pmatrix}$$

Each row (or column) contains a monotonic increasing (or decreasing) sequence of elements ([4]). The transpose of S, S^T, is called a *sorted matrix* [5][6][7] such that $\forall i, j, k, 1 \leq k \leq j \leq m'$ and $1 \leq i \leq m'$,

$$s^T_{i,k} \leq s^T_{i,j} \qquad \text{and} \qquad s^T_{k,i} \leq s^T_{j,i}$$

By taking advantage of this ordering of elements, we can efficiently solve the optimization problem. The simplest approach is obtaining a total order of all elements by sorting the elements in the sorted matrix, S^T, and then applying a binary search to find the optimal schedule length. Unfortunately, sorting a $m' \times m'$ sorted matrix has a lower bound of $\Omega(m'^2 \log m')$ [8] which is worse than the searching based algorithm[4]. Furthermore, we cannot directly construct the sorted matrix by calculating all the elements; otherwise the time complexity is at least the size of the sorted matrix, ie. $\Omega(m'(m'-n))$. This lower bound becomes $\Omega(m'^2)$ if $m' - n = \Omega(n)$. As a result, we needs a representation of the sorted matrix such that it is not necessary to calculate and store all the elements but we can compute any element in $O(1)$ time.

4.2 Sorted Matrix $X + Y$

It can be easily observed that every pair of elements (non-zero and non-infinity) of any two rows or columns has a common difference, ie. for all $0 < s^T_{i,j}, s^T_{i',j}, s^T_{i,k}, s^T_{i',k}$, $s^T_{i,j'}, s^T_{k,j}, s^T_{k,j'} < \infty$,

$$\begin{array}{lrcll} & s^T_{i,j} - s^T_{i',j} & = & s^T_{i,k} - s^T_{i',k} & \text{(any two rows)} \\ \text{and} & s^T_{i,j} - s^T_{i,j'} & = & s^T_{k,j} - s^T_{k,j'} & \text{(any two columns)} \end{array}$$

It is an extended form of a *sorted matrix* $X + Y$. [8][5][6]

Define X and Y be two arrays with m' elements each and

$$\begin{aligned} X[i] &= \begin{cases} CMP_1 & \text{if } i = m' \\ C_{1,2} - \sum_{j=2}^{m'-i} E_j + C_{m'-i,m'-i+1} & \text{otherwise} \end{cases} \\ Y[i] &= \sum_{j=2}^{i} E_j + C_{i,i+1} - C_{1,2} \end{aligned}$$

In addition, a function $C : \{1, \ldots, m'\} \times \{1, \ldots, m'\} \to Z^+$ is defined as follows,

$$C(i,j) = \begin{cases} 0 & \text{if } i+j < m'+1 \\ \infty & \text{if } i+j > 2m'-n+1 \\ X[i]+Y[j] & \text{otherwise} \end{cases}$$

Consider the completion time of any sub-chain of un-mergeable modules, $CMP_{i\oplus\cdots\oplus j}$, $1 \le i \le j \le m'$ and $j - i \le m' - n$,

Case 1 $i = 1$,

$$\begin{aligned} CMP_{i\oplus\cdots\oplus j} &= CMP_{1\oplus\cdots\oplus j} \\ &= C_{0,1} + \sum_{k=1}^{j} E_k + C_{j,j+1} \\ &= C_{0,1} + E_1 + C_{1,2} + \sum_{k=2}^{j} E_k + C_{j,j+1}, -C_{1,2} \\ &= X[m'] + Y[j] \\ &= X[m'-i+1] + Y[j] \end{aligned}$$

case 2 $i > 1$,

$$\begin{aligned} CMP_{i\oplus\cdots\oplus j} &= C_{i-1,i} + \sum_{k=i}^{j} E_k + C_{j,j+1} \\ &= C_{i-1,i} + \sum_{k=2}^{j} E_k - \sum_{k=2}^{i} E_k + C_{j,j+1} \qquad (j > i-1) \\ &= C_{1,2} - \sum_{k=2}^{m'-(m'-i+1)} E_k + C_{m'-(m'-i+1),m'-(m'-i+1)+1} + \sum_{k=2}^{j} E_k + C_{j,j+1} - C_{1,2} \\ &= X[m'-i+1] + Y[j] \end{aligned}$$

Since,

$$\begin{aligned} & m'-i+1+j = m'+1+j-i \ge m'+1 && (j \ge i) \\ \text{and} \quad & m'-i+1+j = m'+1+j-i \le m'+1+m'-n && (j-i \le m'-n) \\ & \qquad = 2m'-n+1 \end{aligned}$$

$\forall 1 \le i \le j \le m'$ and $j - i \le m' - n$,

$$CMP_{i\oplus\cdots\oplus j} = C(m'-n+1, j)$$

Thus, we can calculate the completion time of any sub-chain in $O(1)$ time after we have computed the two arrays X and Y. That is we can access any elements in the sorted matrix in $O(1)$ time. The two array X and Y can be represented by the following recurrence equations,

$$\begin{aligned} X[m'] &= C_{0,1} + E_1 + C_{1,2} \\ X[m'-1] &= 2C_{1,2} \\ X[m'-p] &= X[m'-p+1 - E_p + C_{p,p+1} - C_{p-1,p}, && \forall p, 2 \le p < m' \\ Y[1] &= 0 \\ Y[p] &= Y[p-1] + E_p + C_{p,p+1} - C_{p-1,p}, && \forall p, 2 \le p \le m' \end{aligned}$$

The following function can be used to compute X and Y,

Function Compute-XY

Input: m' un-mergeable modules in a chain-like task system with their execution times and communication times, n processors

Output: X and Y

(1) $X[m'] = C_{0,1} + E_1 + C_{1,2}$ //for X
(2) $X[m'-1] = 2C_{1,2}$
(3) for $p = 2$ to $m' - 1$ do
(4) $\quad X[m' - p = X[m' - p + 1] - E_p + C_{p,p+1} - C_{p-1,p}$
(5) $Y[1] = 0$ //for Y
(6) for $p = 2$ to m' do
(7) $\quad Y[p] = Y[p-1] + E_p + C_{p,p+1} - C_{p-1,p}$
(8) return X and Y

Line 1,2 and 5 take $O(1)$ time and the two loops (line 3-4 and 6-7) take $O(m')$ time. Thus, we can compute the two arrays X and Y in $O(m')$ time.

We can pre-compute the two arrays X and Y so that we can access any element in the logical sorted matrix S^T in $O(1)$ time by the following function,

Function $C(i,j)$

Input: X and Y

Output: the element $s^T_{i,j}$ in the sorted matrix S^T

if $i + j < m' + 1$
$\quad$ return 0
if $i + j > 2m' - n + 1$
$\quad$ return ∞
return $X[i] + Y[j]$

4.3 Algorithm for the Optimization Problem

This algorithm is a modified version of the p-centers problem on a tree structure [9]. Assume the number of un-mergeable modules, m', is a power of 2. The sorted matrix, S^T, is divided into sub-matrices (called *block matrices*) at the beginning of the t-th iteration and the dimension of each block matrix, $B^{i,j}(t)$, is $D(t) = \frac{m}{2^t}$ where $s^T_{i,j}$ is the upper left element of $B^{i,j}(t)$, $i, j = 1, D(t)+1, 2D(t)+1, \ldots, (2^t - 1)D(t) + 1$, ie.

$$S^T = \begin{pmatrix} B^{1,1}(t) & B^{1,D(t)+1}(t) & \cdots & B^{1,(2^t-1)D(t)+1}(t) \\ B^{D(t)+1,1}(t) & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ B^{(2^t-1)D(t)+1,1}(t) & \cdots & \cdots & B^{(2^t-1)D(t)+1,(2^t-1)D(t)+1}(t) \end{pmatrix}$$

and $\forall i, j = 1, D(t)+1, 2D(t)+1, \ldots, (2^t-1)D(t)+1$,

$$B^{i,j}(t) = \begin{pmatrix} s^T_{i,j} & \cdots & s^T_{i,j+D(t)-1} \\ \vdots & \ddots & \vdots \\ s^T_{i+D(t)-1,j} & \cdots & s^T_{i+D(t)-1,j+D(t)-1} \end{pmatrix}$$

Furthermore, let $b^{i,j}_s(t)$ and $b^{i,j}_l(t)$ denote the smallest and largest elements in $B^{i,j}(t)$ respectively,

$$
\begin{aligned}
b_s^{i,j}(t) &= s_{i,j}^T \\
b_l^{i,j}(t) &= s_{i+D(t)-1,j+D(t)-1}^T
\end{aligned}
$$

It is obvious that $B^{i,j}(t)$ contains all zero or infinity elements if $b_s(t) = b_l(t) = 0$ or ∞. At each iteration, each block matrix is divided into 4 sub-matrices with equal size and some block matrices are removed if the optimal schedule length is not included in these matrices. Let $\Gamma(t)$ denotes the set of block matrices remained after the t-th iteration and $\Gamma(0) = \{S^T\}$. At the t-iteration, every block matrix in $\Gamma(t-1)$ is divided into 4 block matrices with equal size. That is $\forall B^{i,j}(t-1) \in \Gamma(t-1)$, $B^{i,j}(t)$, $B^{i+D(t)+1,j}(t)$, $B^{i,j+D(t)+1}(t)$ and $B^{i+D(t)+1,j+D(t)+1}(t)$ are generated and stored in $\Gamma(t)$, where $D(T) = \frac{m'}{2^t}$. Two values, $b_s(t)$ and $b_l(t)$ are calculated as follows,

$$
\begin{aligned}
b_s(t) &= \text{median in } \{b_s^{i,j}(t) : B^{i,j}(t) \in \Gamma(t)\} \\
\text{and} \qquad b_l(t) &= \text{median in } \{b_l^{i,j}(t) : B^{i,j}(t) \in \Gamma(t)\}
\end{aligned}
$$

Then, the decision test is applied with $\mathcal{K} = b_s(t)$ and $b_l(t)$ respectively. If $b_s(t) = \mathcal{K}^o$ or $b_l(t) = \mathcal{K}^o$ where $\mathcal{K}^o$ is the optimal schedule, the algorithm simply returns the corresponding feasible length-$\mathcal{K}^o$ schedule. If $b_s(t) > \mathcal{K}^o$, all block matrices with the smallest element no less than $b_s(t)$ are removed from $\Gamma(t)$; otherwise, remove all block matrices with the largest element no greater than $b_s(t)$. Similarly, if $b_l(t) < \mathcal{K}^o$, remove all block matrices with the largest element no greater than $b_l(t)$; otherwise, remove all block matrices with the smallest element no less than $b_l(t)$. This completes one iteration. After $\log m'$ iterations, each block matrix in $\Gamma(\log m')$ has only one element. The searching based algorithm presented in section ??? is applied on this set, $\Gamma(\log m')$, to find the optimal schedule. The above algorithm is summarized as follows,

Algorithm 2 *(Algorithm Based on a Sorted Matrix)*

Input: *m modules in a chain-like task system with their execution times and communication times, n processors*
Output: *An optimal schedule.*

(1) merge the m modules into m' un-mergeable modules by Hsu's algorithm
(2) Compute-XY
(3) $\Gamma(0) = \{S^T\}$
(4) for $t = 1$ to $\log m'$
(5) $\Gamma(t) = \emptyset, D(t) = \frac{m'}{2^t}$
(6) $\forall B^{i,j}(t-1) \in \Gamma(t-1)$, divide $B^{i,j}(t-1)$ into 4 matrices $B^{i,j}(t)$, $B^{i+D(t)+1,j}(t)$, $B^{i,j+D(t)+1}(t)$ and $B^{i+D(t)+1,j+D(t)+1}(t)$. Add each of these block matrices, $B^{x,y}(t)$, to $\Gamma(t)$ if $b_s^{x,y}(t) \neq b_l^{x,y}(t)$
(7) $b_s(t) =$median in $\{b_s^{i,j}(t) : B^{i,j}(t) \in \Gamma(t)\}$
        $b_l(t) =$median in $\{b_l^{i,j}(t) : B^{i,j}(t) \in \Gamma(t)\}$
(8)   apply the decision test with $\mathcal{K} = b_s(t)$ and $b_l(t)$ respectively.
(9)   if $b_s(t) = \mathcal{K}^o$ or $b_l(t) = \mathcal{K}^o$
(10)   return the corresponding feasible length-$\mathcal{K}^o$ schedule
(11)   if $b_s(t) > \mathcal{K}^o$
(12) $\Gamma(t) = \Gamma(t) - \{B^{i,j}(t) : b_s^{i,j}(t) \geq b_s(t), B^{i,j}(t) \in \Gamma(t)\}$
(13)   else
(14) $\Gamma(t) = \Gamma(t) - \{B^{i,j}(t) : b_l^{i,j}(t) \leq b_s(t), B^{i,j}(t) \in \Gamma(t)\}$
(15)   if $b_l(t) < \mathcal{K}^o$

```
(16)          Γ(t) = Γ(t) − {B^{i,j}(t) : b_l^{i,j}(t) ≤ b_l(t), B^{i,j}(t) ∈ Γ(t)}
(17)      else
(18)          Γ(t) = Γ(t) − {B^{i,j}(t) : b_s^{i,j}(t) ≥ b_l(t), B^{i,j}(t) ∈ Γ(t)}
(19) M =median in Γ(t)        //each block matrix has only one element
(20) apply the decision test with K = M
(21) if M =K^o
(22)      return the corresponding length-K^o schedule
(23) else if M >K^o
(24)      Γ(t) = Γ(t) − {B^{i,j}(t) : b_s^{i,j}(t) ≥ M, B^{i,j}(t) ∈ Γ(t)}
(25) else
(26)      Γ(t) = Γ(t) − {B^{i,j}(t) : b_s^{i,j}(t) ≤ M, B^{i,j}(t) ∈ Γ(t)}
(27) repeat (19)
```

In Algorithm 2, block matrix $B^{i,j}(t)$ is removed from $\Gamma(t)$ if and only if $b_s^{i,j}(t) > \mathcal{K}^o$ or $b_l^{i,j}(t) < \mathcal{K}^o$ (line 12,14,16 and 18) so that there exists a block matrix $B^{i,j}(t) \in \Gamma(t)$ such that $B^{i,j}(t)$ includes an element which is always the optimal schedule length, $\mathcal{K}^o$. As a result, Algorithm 2 always halts and returns an optimal schedule.

4.4 Complexity Analysis

By [7], $\Gamma(t)$ can be partitioned into $2^{t+1}+1$ subsets $\Gamma_i(t)$ s.t.

$$\Gamma(t) = \bigcup_{p=1-2^t}^{2^t-1} \Gamma_p(t) \text{ and } \Gamma_p(t) = \{B^{i,j}(t) : i-j = pD(t), B^{i,j}(t) \in \Gamma(t)\}$$

$\forall \Gamma_p(t) \subseteq \Gamma(t), \forall B^{a,b}(t), B^{c,d}(t) \in \Gamma_p(t)$, either

$$b_s^{a,b}(t) \le b_l^{a,b}(t) \le b_s^{c,d}(t) \le b_l^{c,d}(t) \quad \text{or} \quad b_s^{c,d}(t) \le b_l^{c,d}(t) \le b_s^{a,b}(t) \le b_l^{a,b}(t)$$

In other words, every elements in $B^{a,b}(t)$ is no less (or greater) than all elements in $B^{c,d}(t)$ if they belong to the same subset $\Gamma_p(t)$. Each subset $\Gamma_p(t)$ is called a *incomparable chain of block matrices*[7]. In addition, $\forall B^{i,j}(t) \in \Gamma(t), b_s^{i,j}(t) < b_l^{i,j}(t)$ (line 6 in Algorithm 2), thus

$$b_s^{a,b}(t) < b_l^{a,b}(t) < b_s^{c,d}(t) < b_l^{c,d}(t) \quad \text{or} \quad b_s^{c,d}(t) < b_l^{c,d}(t) < b_s^{a,b}(t) < b_l^{a,b}(t)$$

Lemma 5 *At the t-th iteration, for any value b, the number of block matrix $B^{i,j}(t)$, such that $b_s^{i,j}(t) \le b \le b_l^{i,j}(t)$, is no greater than $2^{t+1}-1$.*

Proof: $\forall \Gamma_p(t) \subseteq \Gamma(t)$, if $\exists B^{i,j}(t) \in \Gamma_p(t)$ such that $b_s^{i,j}(t) \le b \le b_l^{i,j}(t)$, then $\forall B^{a,b}(t) \in \Gamma_p(t) - \{B^{i,j}(t)\}$, either

$$b_s^{a,b}(t) < b_l^{a,b}(t) < b_s^{i,j}(t) \le b \quad \text{or} \quad b_l^{a,b}(t) > b_s^{a,b}(t) > b_l^{i,j}(t) \ge b$$

By [7], there are $2^{t+1}-1$ incomparable chains of block matrices and there is at most one block matrix in each $\Gamma_p(t) \subseteq (t)$ such that it includes the element b. □

Lemma 6 *$\forall t = 1, 2, \ldots, \log m'$, the number of block matrices remained after the t-th iteration, $N_t = |\Gamma(t)|$, is no greater than 2^{t+2} and $N_0 = 1$.*

Proof: By induction on t.
For $t = 1$,

$$N_1 \le 4N_0 = 4 \le 2^{1+2}$$

Assume it is true for $t = k$, ie.

$$N_k \le 2^{k+2}$$

For $t = k+1$,

Case 1: If $b_s(t) > \mathcal{K}^o$ or $b_l(t) < \mathcal{K}^o$ (line 11-12 or 15-16), half of the block matrices in $\Gamma(t)$ are removed and

$$N_{k+1} \leq \frac{1}{2} \times 4N_k \leq \frac{4 \times 2^{k+2}}{2} = 2^{(k+1)+2}$$

Case 2: If $b_s(t) < \mathcal{K}^o$ and $b_l(t) > \mathcal{K}^o$

Consider $b_s(t) < \mathcal{K}^o$, by lemma 5, the number of block matrices include $b_s(t)$ is no greater than $2^{t+1} - 1 = 2^{k+2} - 1 < 2^{k+2}$. Thus, the number of block matrices $B^{i,j}(t)$ such that $B_s^{i,j}(t) \geq b_s(t)$ or $B_l^{i,j}(t) \leq b_s(t)$ is no less than $4N_k - 2^{k+2}$. As a result, the number of block removed in line 14 is no less than $4N_k - 2^{k+2} - 2N_k$ (because at least $2N_k$ block matrices have the smallest value greater than the median $b_s(t)$), that is the number of block matrices removed in line 14 is at least $2N_k - 2^{k+2}$. Similarly, the number of block removed in line 18 is at least $2N_k - 2^{k+2}$. The total number of block matrices removed is at least $4N_k - 2^{k+3}$ so that

$$N_{k+1} \leq 4N_k - (4N_k - 2^{k+3}) = 2^{k+3} = 2^{(k+1)+2}$$

so, it is true for $t = k + 1$. □

In Algorithm 2, $O(m)$ time is needed for merging in line 1. The function Compute-XY has a time and space complexities of $O(m')$. At the t-th iteration, $\Gamma(t)$ can be implemented by an $O(N - t)$ array. $b_s(t)$ and $b_l(t)$ can be computed in linear time (by function $C(i, j)$ in section 4.2 and linear time algorithm of median finding [10][11][12]) The decision test in line 8 consumes $O(m')$ time. The steps for reducing the size of $\Gamma(t)$ in line 11-18 have a time complexity of $O(N_t)$. As a consequence, the time taken for step 11-18 is

$$\begin{aligned} \sum_{t=1}^{\log m'} O(N_t + m') &= O(2^{\log m'} + m' \log m') && \text{(by Lemma 6)} \\ &= O(m' \log m') \end{aligned}$$

After the $logm'$-th iteration, $N_{\log m'} \leq 2^{\log m' + 2} = O(m')$ by Lemma6. The number of execution from line 19-27 is $O(\log m')$ and each iteration takes $O(m')$ times. As a result, the total time complexity of Algorithm 2 is

$$O(m + m' \log m')$$

The space complexity of Algorithm 2 is linear, ie.

$$O(m)$$

5 Conclusion

In this paper, an algorithm is presented for the decision problem of comparing any constant, $\mathcal{K}$, with the optimal schedule. The time and space complexities of this algorithm are $O(m')$. This is an asymptotically optimal algorithm. In addition, we can get an optimal schedule if the optimal schedule is known and the resulting optimal schedule uses the minimal number of processors among all optimal schedules. For Hsu's algorithm [3], all processors must always be utilized in any optimal schedule if $m' \geq n$.

For obtaining an optimal schedule, an $O(m' \log m')$ algorithm is presented. This is a modified version of the searching algorithm in [4]. All possible schedule lengths are organized as the elements in a sorted matrix. By using the partial ordering of a sorted matrix and the generalized decision test, we can find an optimal schedule with time and space complexities of $O(m + m' \log m')$ and $O(m)$ respectively. This algorithm is better than all existing algorithms except Hsu's one [3] when the special case of $m' - n = o(\sqrt{\log n})$ occurs.

References

[1] Shahid H. Bokhari. "Partitioning Problems in Parallel, Pipelined, and Distributed Computing". *IEEE Transactions on Computers*, vol 37:pp. 48–57, January 1988.

[2] J.P. Sheu and Z.F. Chaing. "Efficient allocation of chain-like task on chain-like network computers". *Information Processing Letters*, vol 36:pp. 241–245, 1990.

[3] C.C. Hsu. "A Two-phase Approach for the Optimal Assignment of a Chain-like Task on a Chain-like Network Computer". Technical report, National Taiwan Institute of Technology, 1993.

[4] Gilbert Young and Chan Chi-Lok. "Scheduling Algorithm for a Chain-like Task System". In *"Lecture Notes in Computer Science"*, volume vol:762, pages 496–515, 1993.

[5] Donald B. Johnson and Samuel D. Kashdan. "Lower bounds for Selection in X+Y and Other Multisets". *Journal of ACM*, vol 25:pp. 556–570, 1978.

[6] Greg N. Frederickson and Donald B. Johnson. "The Complexity of Selection and Ranking in X+Y and Matrices with Sorted Columns". *Journal of Computer and System Science*, vol 24:pp. 197–208, 1982.

[7] Greg N. Frederickson and Donald B. Johnson. "Generalized Selection and Ranking Matrices". *SIAM Journal of Computing*, vol 13:pp. 14–30, 1984.

[8] L.H. Harper, T.H. Payne, J.E. Savage, and E. Straus. "Sorting X+Y". *Communication of ACM*, vol 18:pp. 347–349, 1975.

[9] Greg N. Frederickson and Donald B. Johnson. "Finding kth Paths and p-Centers by Generating and Searching Good Data Structure". *Journal of Algorithms*, vol 4:pp. 61–80, 1983.

[10] Manuel Blum, Robert W. Floyd, Vaughan Pratt, Rivest Ronald L., and Robert E. Tarjan. "Time Bounds for Selection". *Journal of Computer and System Science*, vol 7(4):pp. 448–461, 1973.

[11] Robert W. Floyd and Ronald L. Rivest. "Expected Time Bounds of Selection". *Communication of ACM*, vol 18(3):pp. 165–172, 1975.

[12] C.A.R. Hoare. "Algorithm 63(Partition) and Algorithm 65(Find)". *Communication of ACM*, vol 4(7):pp. 321–322, 1961.