Processor Allocation and Task Scheduling of Matrix Chain Products on Parallel Systems†

Heejo Lee*, Jong Kim*, Sungje Hong*, and Sunggu Lee**

*Dept. of Computer Science and Engineering
**Dept. of Electrical Engineering
Pohang University of Science and Technology
San 31 Hyoja Dong, Pohang 790-784, KOREA
E-mail : {heejo,jkim}@postech.ac.kr

Abstract

The problem of finding an optimal product sequence for sequential multiplication of a chain of matrices (the matrix chain ordering problem, MCOP) is well-known and has been studied for a long time. In this paper, we consider the problem of finding an optimal product schedule for evaluating a chain of matrix products on a parallel computer (the matrix chain scheduling problem, MCSP). The difference between the MCSP and the MCOP is that the MCOP pertains to a product sequence for single processor systems and the MCSP pertains to a sequence of concurrent matrix products for parallel systems. The approach of parallelizing each matrix product after finding an optimal product sequence for single processor systems does not always guarantee the minimum evaluation time on parallel systems since each parallelized matrix product may use processors inefficiently. We introduce a new processor scheduling algorithm for the MCSP which reduces the evaluation time of a chain of matrix products on a parallel computer, even at the expense of a slight increase in the total number of operations. Given a chain of n matrices and a matrix product utilizing at most P/k processors in a P-processor system, the proposed algorithm approaches $k(n-1)/(n+k\log(k)-k)$ times the performance of parallel evaluation using the optimal sequence found for the MCOP. Also, experiments performed on a Fujitsu AP1000 multicomputer show that the proposed algorithm significantly decreases the time required to evaluate a chain of matrix products in parallel systems.

Keywords – Processor allocation, task scheduling, matrix chain product, parallel matrix multiplication, matrix chain scheduling problem.

† This research was supported in part by the NON DIRECTED RESEARCH FUND, Korea Research Foundation.

1 Introduction

Matrix multiplication is a computation intensive part of many commonly used scientific computing applications. Many algorithms in numerical and non-numerical problems are solved efficiently using matrix-matrix multiplications [1]. Also, in case of parallel algorithms, some problems that are normally not solved by sequential algorithms turn out to be efficiently solved using matrix-matrix multiplications or variations of matrix-matrix multiplication [2]. Such an application uses matrix multiplication as a basic computational kernel so that a chain of matrices is frequently required and the matrices are multiplied consecutively [2, 3, 4, 5].

In the evaluation of a chain of matrix products with n matrices, $\mathcal{M} = M_1 \times M_2 \times \cdots \times M_n$, where M_i is an $m_i \times m_{i+1} (m_i \geq 1)$ matrix, the product sequence greatly affects the total number of operations required to evaluate $\mathcal{M}$, even though the final result is the same for all product sequences due to the associative law of matrix multiplication. In the worst case, an arbitrary product sequence of matrices is $\Omega(T_{opt}^3)$ where T_{opt} is the minimum number of operations required to evaluate a chain of matrix products [6]. The matrix chain ordering problem (MCOP) focuses on finding a product sequence for a set of matrices such that the total number of operations is minimized.

An exhaustive search to find an optimal solution for the MCOP is not a good strategy since the number of possible product sequences of a chain of matrix products with n matrices is $\Theta(4^n/n^{3/2})$, which is known as the *Catalan number* [7]. Therefore, determining the optimal sequence by this method is very time consuming. There has been much work reported for solving the MCOP. The MCOP was first studied by Godbole [8] and solved using dynamic programming in $O(n^3)$ time. Chin [9] suggested an approximation algorithm, which runs in $O(n)$ time and finds a near-optimal sequence. An optimal sequential algorithm, which runs in $O(n \log(n))$ time, was given by Hu and Shing [10, 11]. This algorithm solves the MCOP by solving the equivalent problem of finding the optimal triangulation of a convex polygon. Ramanan [12] presented a simpler algorithm for the MCOP, and obtained the tight lower bound of $\Omega(n \log(n))$ for a related problem.

Let us refer to the time required to find an optimal product sequence for a chain of matrices as the *ordering time* and the time required to execute the product sequence as the *evaluation time*. Many parallel algorithms aimed at reducing the *ordering time* have been studied using the dynamic programming method [13, 14, 15, 16] and the convex polygon triangulation method [17, 18, 19]. Bradford *et al.* [16] proposed a parallel algorithm based on dynamic programming, which runs in $O(\log^3(n))$ time with $n/\log(n)$ processors on the

CRCW PRAM model. Czumaj [17] proposed an $O(\log^3(n))$ time algorithm based on the triangulation of a convex polygon, which runs with $n^2/\log^3(n)$ processors on the CREW PRAM. Also, in [18] he proposed a faster approximation algorithm that finds a near-optimal solution in $O(\log(n))$ time on the CREW PRAM, and in $O(\log\log(n))$ time on the CRCW PRAM. Ramanan [19] gives an optimal algorithm that runs in $O(\log^4(n))$ time using n processors on a CREW PRAM.

Now let us consider the *evaluation time* of a chain of matrix products. In a single processor system, the evaluation of a chain of matrix products using the optimal product sequence for the MCOP guarantees the minimum evaluation time since the sequence guarantees the minimum number of operations. However, in parallel systems, parallel computation of each matrix product using the product sequence found for the minimum number of operations does not guarantee the minimum evaluation time. This is because the evaluation time in parallel systems is affected by various factors such as dependencies among tasks, communication delays, and processor efficiency. To this date, there has been no research reported in the open literature on ways to reduce the matrix chain evaluation time in a parallel system.

In this paper, we formally present the problem of finding the matrix product schedule for parallel systems (MCSP) and analyze the complexity of the MCSP. We propose an algorithm which finds a matrix product schedule that, while possibly increasing the total number of required operations, decreases the evaluation time of a chain of matrix products by finding sets of matrix products that can be executed concurrently.

This paper is organized as follows. Section 2 presents a formal description of the processor scheduling problem for a chain of matrix products and shows that the given problem is NP-hard. In Section 3, we present a processor allocation method for multiplying a number of independent matrix products concurrently. In Section 4, we propose a matrix chain scheduling algorithm that dramatically reduces the evaluation time of a chain of matrix products by using processors efficiently in parallel systems. In Section 5, we analyze the evaluation performance of sequences found with the proposed method and sequences found for the MCOP. We also compare the proposed method with various other evaluation methods through experiments on a Fujitsu AP1000 parallel system. In Section 6, many practical issues and further extensions of the MCSP are discussed. Finally, in Section 7, we summarize and conclude the paper.

2 Matrix Chain Scheduling Problem

2.1 Notation

- P: the number of processors in a parallel system.
- $\mathcal{M}$: a matrix chain product of n matrices, i.e., $\mathcal{M} = M_1 \times M_2 \times \cdots \times M_n$.
- M_i: an $m_i \times m_{i+1}$ matrix ($m_i \geq 1$, $1 \leq i \leq n$).
- L: a product sequence tree for a matrix chain $\mathcal{M}$.
- $L_{i,j}$: the sequence subtree of L for $(M_i \times \cdots \times M_j)$.
- C: the minimum number of computations for evaluating $\mathcal{M}$.
- ΔC: the amount of increased computation incurred by modifying the current sequence tree.
- $p_{i,j}$: the number of processors assigned for evaluating $(M_i \times \cdots \times M_j)$.
- $T_{i,j}(p_{i,j})$: the execution time for evaluating $(M_i \times \cdots \times M_j)$ on $p_{i,j}$ processors.
- (m_i, m_j, m_k): a single matrix product for multiplying an $m_i \times m_j$ matrix by an $m_j \times m_k$ matrix.
- $\Phi(m_i, m_j, m_k, p)$: the execution time of a single matrix product (m_i, m_j, m_k) using p processors.
- $D(x)$: the set of divisors of x, i.e., $D(x) = \{d | d \text{ divides } x\}$.
- $LD(x, y)$: the largest divisor in $D(x)$ that is not larger than y.
- $SD(x, y)$: if $x > y$, then $SD(x, y)$ is the smallest divisor in $D(x)$ that is larger than y. Otherwise, $SD(x, y) = x$.
- m: the largest dimension among all of the matrices, i.e., $m = \max_{1 \leq i \leq n+1}(m_i)$.

2.2 Problem Description

We consider the problem of finding the schedule with minimum evaluation time for $\mathcal{M}$ on a P processor parallel system. The number of operations needed to multiply a matrix A of size $m_i \times m_j$, by a matrix B of size $m_j \times m_k$, is $m_i m_j m_k$. [1] Many parallel algorithms for matrix multiplication have been developed for various parallel architectures [23, 24],

[1] Even though Strassen's algorithm [20, 21] and its variants perform fewer than n^3 operations for $n \times n$ matrix multiplication, these faster algorithms are regarded as inappropriate methods for matrix chain products due to more erroneous results caused by round-off errors and larger storage requirements than the usual inner-product type algorithm [22]. Therefore, we assume that a simple algorithm is used so that n^3 operations are required for $n \times n$ matrix multiplication. This is also the assumption made in other research work on the MCOP.

and the execution time of matrix multiplication depends on the algorithm used and the architecture on which the algorithm runs. However, for a broader discussion, we assume that a simple parallel algorithm [24] is used and the execution time of matrix multiplication is proportional to the number of operations on a processor. Therefore, for multiplying A by B using p processors, the execution time, $\Phi(m_i, m_j, m_k, p)$, can be approximated as follows:

$$\Phi(m_i, m_j, m_k, p) \approx \begin{cases} \frac{m_i m_j m_k}{p} & \text{if } 1 \leq p \leq \frac{m_i m_j m_k}{\log(m_j)}, \\ \frac{m_i m_j m_k}{p} \log(\frac{p}{m_i m_k}) & \text{if } \frac{m_i m_j m_k}{\log(m_j)} < p \leq m_i m_j m_k. \end{cases}$$

When p_{ij} processors are allocated for evaluating $(M_i \times \cdots \times M_j)$, the evaluation time consists of two parts: the partial matrix chain evaluation time and the single matrix product execution time. The two partial matrix chains are $(M_i \cdots M_k)$ and $(M_{k+1} \cdots M_j)$ for any k $(i \leq k < j)$. The evaluation time of the two matrix product chains is dependent on the evaluation method. One method is to evaluate sequentially $(M_i \cdots M_k)$ and $(M_{k+1} \cdots M_j)$ using all available processors in p_{ij}. The other method is to evaluate both $(M_i \cdots M_k)$ and $(M_{k+1} \cdots M_j)$ concurrently by partitioning p_{ij} into $p_{i,k}$ and $p_{k+1,j}$ such that $p_{i,k} + p_{k+1,j} \leq p_{ij}$. The minimum evaluation time $T_{i,j}(p_{i,j})$ of $(M_i \cdots M_j)$ on $p_{i,j}$ processors is found from the following recurrence relation:

$$\begin{aligned} T_{i,j}(p_{i,j}) \quad \approx \quad & \min_{i \leq k < j} \Big(T_{i,k}(p_{i,j}) + T_{k+1,j}(p_{i,j}) + \Phi(m_i, m_{k+1}, m_{j+1}, p_{i,j}), \\ & \min_{p_{i,k} + p_{k+1,j} \leq p_{i,j}} \Big(\max\Big(T_{i,k}(p_{i,k}), T_{k+1,j}(p_{k+1,j}) \Big) \Big) + \Phi(m_i, m_{k+1}, m_{j+1}, p_{i,j}) \Big) . \end{aligned}$$

The problem of finding the schedule that results in the minimum evaluation time, $T_{1,n}(P)$, is equivalent to finding the best schedule, $k_{i,j}$, for $(M_i \cdots M_j)$ with the processor allocation p_{ij} to $L_{i,j}$. Therefore, the MCSP is defined as follows.

MCSP: *Given $\mathcal{M}$ and P, find the product sequence for evaluating $\mathcal{M}$ and the processor schedule for the sequence such that the evaluation time is minimized.*

2.3 MCSP Complexity

The complexity of the MCSP depends on the number of processors available for the MCSP. Consider the case in which there are sufficient processors for multiplying any number of matrices concurrently. For a matrix product (m_i, m_{k+1}, m_j), we can allocate the number of processors that guarantees the minimum execution time for the product, i.e, $\log(m_{k+1})$. Then, using dynamic programming, the problem can be solved in polynomial time according

to the following recurrence relation:

$$T_{i,j} = \begin{cases} \min_{i \le k < j} \left(\max(T_{i,k}, T_{k+1,j}) + \log(m_{k+1}) \right) & \text{if } 1 \le i < j \le n, \\ 0 & \text{if } i = j, 1 \le i \le n. \end{cases}$$

Therefore, in the case of an infinite number of processors, the problem of finding the schedule for evaluating $\mathcal{M}$ with the minimum time has a polynomial time algorithm. However, in general, the number of available processors is fixed and not sufficient to be able to allocate the maximum number of processors for each product.

Now, let us consider the case when there are P processors in a system. We show that the MCSP is NP-complete using a reduction from the processor partitioning problem which is known to be NP-complete [25].

Theorem 1: The *MCSP* is NP-complete.

Proof: The decision version of the MCSP is obviously in NP. There is a nondeterministic algorithm that generates a processor schedule with a product sequence of $\mathcal{M}$. Given the schedule, it can be decided in polynomial time whether the schedule length is less than a certain value by finding the longest path in a tree graph.

The processor partitioning problem [25], denoted by *PPP*, is to find the schedule with the minimum completion time of n tasks on a partitionable P processor system ($n \le P$). In the *PPP*, a task j is characterized as δ_j and $t_j(p_j)$ where δ_j denotes the maximum number of allocable processors for task j and $t_j(p_j)$ denotes the execution time of task j on p_i processors ($1 \le p_i \le \delta_j$). The *PPP* of deciding whether there exists a schedule whose completion time is less than D is NP-complete [25].

An instance of the *PPP* can be transformed to an instance of the MCSP. For the *PPP*, $t_j(p_j)$ can be a linear function of p_j, i.e., $t_j(p_j) = t_j(1)/p_j$. Let M_{2j-1} be a $1 \times t_j(1)$ matrix and M_{2j} be a $t_j(1) \times 1$ matrix for $1 \le j \le n$. Now define Φ for the MCSP as follows:

$$\Phi(1, t_j(1), 1, p_j) = t_j(p_j) \quad \text{for } 1 \le p_j \le \delta_j.$$

Thus the *PPP* with n tasks is transformed to an MCSP with $2n$ matrices in polynomial time.

Next let us show that if there is a feasible solution for the MCSP, there exists a solution for the *PPP*. In the transformed MCSP, an optimal sequence for the MCSP has the products $(M_1 \times M_2)$, $(M_3 \times M_4)$, $\cdots$, $(M_{2n-1} \times M_{2n})$ because these n products satisfy the optimal sequence property in Theorem 1 of [9], and because any other sequence not only reduces the degree of concurrency but also increases the computation. Therefore, an optimal schedule

for the transformed MCSP is found in a product sequence tree having these independent n products as leaves. Since execution of $n-1$ non-leaf products with single operations requires at least $\lceil \log(n-1) \rceil$ time, finding a schedule of the transformed MCSP whose completion time is less than $D + \lceil \log(n-1) \rceil$ solves the *PPP* whose schedule length is less than D. Therefore, the *PPP* is transformed to a special case of the MCSP so that the MCSP is NP-complete. □

Since the problem of finding an optimal schedule for the MCSP is an NP-hard optimization problem, we propose a heuristic algorithm in Section 4. The algorithm enhances the evaluation performance of an n-matrix product chain on a parallel system by partitioning the parallel system and executing several matrix products simultaneously; this also enhances the overall efficiency of the system. A processor allocation method for executing multiple matrix products with minimum completion time is discussed in the next section.

3 Processor Allocation for Matrix Products

In this section, we discuss how many processors should be allocated for a single matrix product and determine the optimal processor allocation for executing two matrix products simultaneously. The optimal processor allocation for two matrix products is used for scheduling matrix chain products to run multiple matrix products concurrently.

3.1 Processor Allocation for a Single Matrix Product

Many parallel matrix multiplication algorithms have been developed with various parallel architectures [21, 24]. A multiplication of two $n \times n$ matrices requires at most n^3 processors. In the best case, it takes $O(\log(n))$ time with n^3 processors by using n processors to get one element of the result matrix.[2] Each set of n processors executes a multiplication in one step and sums n elements within $O(\log(n))$ steps. However, in this case we expect low utilization of processors. While summing n data for $\log(n)$ steps, some processors stay idle. Moreover, these $\log(n)$ steps are communication steps, not computation steps. Then, how many processors should be allocated for multiplying two matrices?

One distinctive feature of parallel matrix multiplication is the jerky behavior affecting

[2] On a CRCW PRAM, the matrix multiplication runs in constant time with the assumption that when there are write conflicts, when several processors attempt to write numbers in the same location, the sum of the numbers is written in that location [26]. Since the CRCW PRAM with an arbitrary number of processors is not realistic in practice, we are not considering that case.

the execution time, speedup and efficiency of the computation. This is mainly due to the load imbalance, which is referred to as the "integer effect" in [27]. It is known that many parallel algorithms are based on allocating processors for computing a part, especially as a sub-block, of the result matrix [5]. For executing a product (m_i, m_j, m_k) on p processors, let us consider the case where the operations for computing $m_i m_k$ elements are distributed among p processors. Then, the execution time $\Phi(m_i, m_j, m_k, p)$, the speedup $S(p)$ and the efficiency $E(p)$ are estimated as follows:

$$\Phi(m_i, m_j, m_k, p) = \left\lceil \frac{m_i m_k}{p} \right\rceil m_j,$$

$$S(p) = \frac{\Phi(m_i, m_j, m_k, 1)}{\Phi(m_i, m_j, m_k, p)} = m_i m_k / \left\lceil \frac{m_i m_k}{p} \right\rceil \quad \text{and}$$

$$E(p) = \frac{S(p)}{p} = m_i m_k / \left(p \left\lceil \frac{m_i m_k}{p} \right\rceil \right).$$

Ideally, the speedup is expected to be $S(p) = p$ so that $E(p) = 1$. However, in the worst case, the integer effect causes a speedup $S(p)$ close to $p/2$, which is a factor of 2 away from what we would normally hope for. This means that half of the processors do not have any effect on the execution time. Thus, there is no need to allocate almost half of the processors in this case. Therefore, for efficient execution, the number of processors allocated to a single matrix product should not be an arbitrary number. When multiplying an $m_i \times m_j$ matrix by an $m_j \times m_k$ matrix, the possible number of allocable processors is the number of divisors of $m_i m_k$. For example, when multiplying a 2×4 matrix by a 4×3 matrix, the number of allocable processors is $1, 2, 3$, and 6, the divisors of $2 \times 3 = 6$. In this case, the load of each processor is balanced such that each processor has the same amount of computation.

Let $D(x)$ denote the set of divisors of x, i.e., $D(x) = \{d | d \text{ divides } x\}$. The number of processors allocated to a given matrix product (m_i, m_j, m_k) should be an element in $D(m_i m_k)$. Thus,

$$\Phi(m_i, m_j, m_k, p) = \frac{m_i m_j m_k}{p} \quad \text{for } p \in D(m_i m_k).$$

Even though there may exist an algorithm using $m_i m_j m_k$ processors, we assume that at most $m_i m_k$ processors are allocated to (m_i, m_j, m_k) for efficiency.

3.2 Processor Allocation for the Concurrent Computation of Multiple Matrix Products

In this subsection, we discuss a processor allocation method for independently computing multiple matrix products. When executing multiple parallel tasks concurrently, one

good heuristic for allocating processors to each task is "proportional allocation" [28, 29]. The proportional allocation algorithm allocates a number of processors proportional to the computation amount of each task. This algorithm tries to minimize the completion time of all tasks by balancing the execution times of the tasks. However, it assumes that the execution time of a task decreases if more processors are allocated to it and that any number of processors may be allocated. These assumptions, however, do not hold for the MCSP.

Proportional allocation can be applied to allocate processors for multiple matrix products. However, since we cannot allocate an arbitrary number of processors to a single matrix product, proportional allocation is not a proper processor allocation scheme for the computation of matrix products. For example, consider the case of computing two matrix products $(2, 3, 8)$ and $(4, 4, 3)$ on 20 processors. Both products involve the same amount of work, $2 \times 3 \times 8 = 4 \times 4 \times 3$. By proportional allocation, 10 processors are allocated for each matrix product. Since the number of allocable processors for the two matrix products should be numbers in $D(16)$ and $D(12)$ respectively, $LD(16, 10) = 8$ and $LD(12, 10) = 6$ processors are allocated for each product respectively. Then the completion time of the two matrix products is $\max(2 \times 3 \times 8/8, 4 \times 4 \times 3/6) = 8$ units of time. However, if we allocate 8 processors to the first product and 12 processors to the second product, the completion time is only $\max(6, 4) = 6$ units of time. Since the completion time is bounded by the longer execution time, we can reduce the completion time by allocating unused processors to the matrix product that requires a longer execution time.

The following algorithm describes the processor allocation algorithm for two independent matrix products. Given two matrix products $X = (m_x, m_{x+1}, m_{x+2})$ and $Y = (m_y, m_{y+1}, m_{y+2})$, we let $\Phi(X, p)$ and $\Phi(Y, p)$ be shorthand notation for $\Phi(m_x, m_{x+1}, m_{x+2}, p)$ and $\Phi(m_y, m_{y+1}, m_{y+2}, p)$, respectively.

Discrete Processor Allocation for Two Matrix Products (DPA)

Input: Two matrix products $X = (m_x, m_{x+1}, m_{x+2})$ and $Y = (m_y, m_{y+1}, m_{y+2})$ and a set of P processors.

Output: The number of processors allocated to the matrix products X and Y, denoted as P_x and P_y, which satisfy $1 \leq P_x, P_y \leq P$ and $P_x + P_y \leq P$.

1. $P_{prop} = \frac{m_x m_{x+1} m_{x+2}}{m_x m_{x+1} m_{x+2} + m_y m_{y+1} m_{y+2}} P$
2. $d_i = LD(m_x m_{x+2}, P_{prop})$
3. $d_{i+1} = SD(m_x m_{x+2}, P_{prop})$
4. $d_j = LD(m_y m_{y+2}, P - P_{prop})$
5. $d_{j+1} = SD(m_y m_{y+2}, P - P_{prop})$
6. if $\Phi(X, d_i) \geq \Phi(Y, d_j)$ then

7. if $\max(\Phi(X, d_{i+1}), \Phi(Y, LD(P - d_{i+1}))) < \Phi(X, d_i)$ then
8. $P_x = d_{i+1},\ P_y = LD(m_y m_{y+2}, P - d_{i+1})$
9. else
10. $P_x = d_i,\ P_y = LD(m_y m_{y+2}, P - d_i)$
11. endif
12. elseif $\max(\Phi(X, LD(P - d_{j+1})), \Phi(Y, d_{j+1})) < \Phi(Y, d_j)$ then
13. $P_x = LD(m_x m_{x+2}, P - d_{j+1}),\ P_y = d_{j+1}$
14. else
15. $P_x = LD(m_x m_{x+2}, P - d_j),\ P_y = d_j$
16. endif
17. endif

Even if we use a naive search algorithm for finding divisors, it will take $O(\max(m_i, m_j))$ time for finding both $LD(m_i m_j, p)$ and $SD(m_i m_j, p)$. We let m denote the largest dimension among all of the matrices, i.e., $m = \max_{1 \le i \le n+1}(m_i)$. Then the time complexity of the discrete processor allocation (DPA) algorithm is $O(m)$. Also, DPA guarantees the minimum completion time for two matrix products, as shown formally by the following lemma and theorem.

Lemma 1: Given two matrix products $X = (m_x, m_{x+1}, m_{x+2})$ and $Y = (m_y, m_{y+1}, m_{y+2})$ on P processors, an optimal processor allocation has at least one of four assignments: $P_x = d_i$, $P_x = d_{i+1}$, $P_y = d_j$, or $P_y = d_{j+1}$.

Proof: If an allocation does not have any of the above four assignments, then $P_x < d_i$ or $P_x > d_{i+1}$, and $P_y < d_j$ or $P_y > d_{j+1}$. In this allocation, P_x and P_y are of the following four cases: the first case is $P_x < d_i$ and $P_y < d_j$, the second is $P_x < d_i$ and $P_y > d_{j+1}$, the third is $P_x > d_{i+1}$ and $P_y < d_i$, and the fourth is $P_x > d_{i+1}$ and $P_y > d_{j+1}$. Without loss of generality, let us assume $\Phi(X, d_i) \ge \Phi(Y, d_j)$.

Case i) $P_x < d_i$ and $P_y < d_j$
Since $\max(\Phi(X, P_x), \Phi(Y, P_y)) > \Phi(X, d_i)$, the allocation of P_x and P_y does not guarantee the minimum completion time of X and Y. This means that the completion time by this allocation is longer than that by the allocation of d_i and d_j to X and Y.

Case ii) $P_x < d_i$ and $P_y > d_{j+1}$
As in *Case i)*, since $\max(\Phi(X, P_x), \Phi(Y, P_y)) = \Phi(X, P_x) > \Phi(X, d_i)$, the allocation of P_x and P_y does not result in the minimum completion time.

Case iii) $P_x > d_{i+1}$ and $P_y < d_i$
In this case, the completion time with the allocation of P_x and P_y is $\max(\Phi(X, P_x), \Phi(Y, P_y))$ $= \Phi(Y, P_y)$. Let us compare this allocation with $P'_x = d_{i+1},\ P'_y = LD(Y, P - P'_x)$. In the

allocation of P'_x and P'_y, the completion time is $\max(\Phi(X, P'_x), \Phi(Y, P'_y)) = \Phi(Y, P'_y)$. Also, since $P'_x < P_x$, it is the case that $P'_y \geq P_y$. In the case of $P'_y > P_y$, the completion time $\Phi(Y, P'_y)$ with the allocation of P'_x and P'_y is shorter than the completion time $\Phi(Y, P_y)$ with the allocation of P_x and P_y so that the allocation of P_x and P_y does not guarantee the minimum completion time of X and Y. In the case of $P'_y = P_y$, the allocation of P_x and P_y does not take less time than the allocation of P'_x and P'_y, and the optimal allocation with the minimum completion time can be found with $P'_x = d_{i+1}$.

Case iv) $P_x > d_{i+1}$ and $P_y > d_{j+1}$
In this case, we cannot allocate P_x and P_y since $P_x + P_y > P$.

For all the four cases, the allocation of P_x and P_y does not take less time than allocation with one of the original four assignments. Therefore, the optimal allocation with the minimum completion time is found with one of the original four assignments. □

Theorem 2: *DPA* guarantees the minimum completion time for two matrix products.

Proof: Since DPA finds the allocation with the minimum completion time among the allocations having one of the four assignments in Lemma 1, DPA guarantees the minimum completion time for computing two matrix products simultaneously. □

In the next section, DPA is used for two-partitioning the processors allocated to a leaf product for running another candidate product simultaneously. Thus, we can compute multiple matrix products independently with an increased degree of concurrency.

4 Matrix Chain Scheduling Algorithm

The proposed scheduling algorithm consists of three stages. First, the algorithm finds the optimal product sequence for the MCOP. Next is the top-down processor assignment stage. In this stage, processors are partitioned and proportionally assigned to each subtree according to their computation amount in order to balance the evaluation time of both the left and right partial matrix product chains. The third stage is the bottom-up execution stage that executes products independently from the leaf and tries to modify the product sequence to enhance concurrency so as to reduce the evaluation time of $\mathcal{M}$. This is done by finding the points that change the product sequence but do not excessively increase the total number of operations.

4.1 Optimal Product Sequence by the MCOP

The product sequence of $\mathcal{M}$ determines the number of operations to be executed in single processor systems. In parallel systems, the number of operations is not the sole factor determining the evaluation time, but it affects the evaluation time greatly nonetheless. Hence, our scheduling algorithm begins with the optimal product sequence found for the MCOP. There have been many works reported for finding the optimal product sequence that guarantees the minimum number of operations for any chain of matrix products. The optimal product sequence can be found in $O(n \log(n))$ time using a sequential algorithm [10, 11]. In addition, many parallel algorithms that run in polylog time have been studied [16, 17, 19]. Thus, by using these parallel algorithms, it is possible to find the optimal product sequence within polylog time on P processor systems.

Let us assume that the sequence and the number of operations found for the MCOP is stored in two tables named $S[n, n]$ and $W[n, n]$, respectively. $W[i, j]$ has the minimum number of operations for evaluating $L_{i,j}$, and $S[i, j]$ has the matrix index for partitioning the matrix chain $(M_i \times \cdots \times M_j)$. Note that the algorithm for the MCOP may not have computed $L_{i,j}, S[i, j]$ or $W[i, j]$ for all i, j.

4.2 Top-Down Processor Assignment

In the top-down processor assignment stage, the number of processors assigned to two partial matrix chains is set to be proportional to the computation amount of a subtree. If $p_{i,j}$ processors have been assigned to $L_{i,j}$, then $p_{i,j} \times \frac{W[i,S[i,j]]}{(W[i,S[i,j]]+W[S[i,j]+1,j])}$ processors are assigned to subtree $L_{i,S[i,j]}$ and $p_{i,j} \times \frac{W[S[i,j]+1,j]}{(W[i,S[i,j]]+W[S[i,j]+1,j])}$ processors are assigned to subtree $L_{S[i,j]+1,j}$.

For example, given a chain of 8 matrices with dimensions $\{3, 8, 9, 5, 8, 3, 3, 3, 4\}$ and a 64 processor system, processors are assigned as in Fig. 1.

4.3 Bottom-Up Concurrent Execution

After assigning processors to each subtree, the matrix products are executed concurrently and independently, starting from the leaf products. However, there are cases in which some processors stay idle. When there are idle processors in the execution of $L_{i,j}$, we try to modify the product sequence to use these idle processors by tracing the ancestors of the leaf node of $L_{i,j}$ in order to find a candidate for concurrent execution. This upward

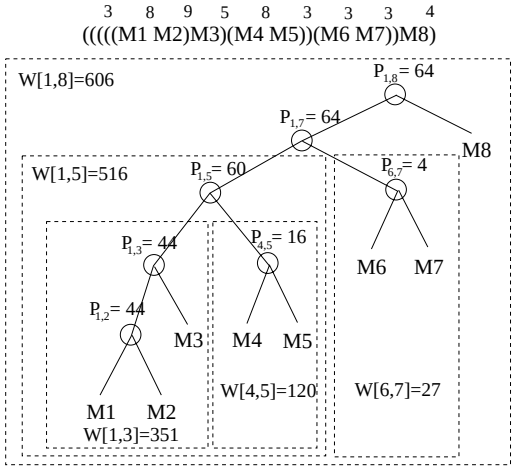

Figure 1: Top-down processor assignment.

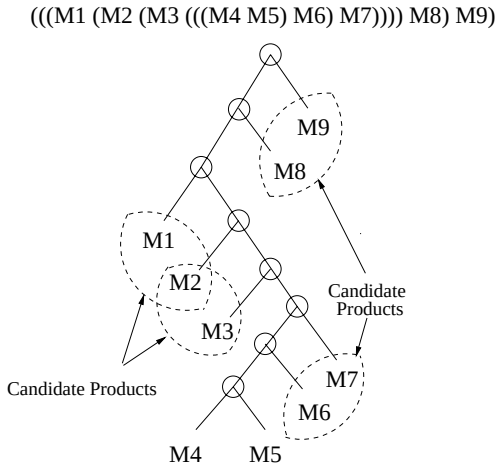

Figure 2: Candidate products on a sequence tree.

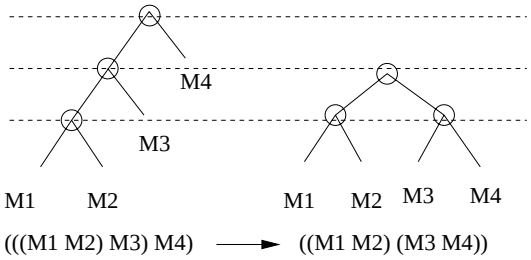

Figure 3: A sequence modification.

trace continues until a suitable candidate or a sibling which is not a leaf node is found. For example, let us consider executing the sequence tree $L_{1,9}$ shown in Fig. 2, which represents $(((M_1(M_2(M_3(((M_4M_5)M_6)M_7))))M_8)M_9)$. In the execution of (M_4M_5), the possible candidates for concurrent execution are $(M_1M_2), (M_2M_3), (M_6M_7)$ and (M_8M_9). There are other types of candidate products, e.g., (M_7M_8), which are not considered in this paper because such cases result in more modifications to the optimal sequence with no obvious benefit over other candidates.

The matrix product (M_yM_{y+1}) is a candidate of the leaf product (M_xM_{x+1}) if one of the following two conditions is satisfied.

Condition–1 $y > x + 1$ and the node associated with M_{y+1} has the left child node associated with M_y in the sequence tree;

Condition–2 $y < x-1$ and the node associated with M_y has the right child node associated with M_{y+1} in the sequence tree.

When we modify the product sequence to execute candidate products simultaneously in the current execution phase, there is some loss due to an increase in the total number of operations. Therefore, we have to check whether the modification is beneficial or not.

Consider a matrix chain with four matrices that needs three matrix products to get the final result. Assume that the optimal product sequence of this matrix chain for the MCOP is $(((M_1M_2)M_3)M_4)$, as shown in Fig. 3. When the sequence is modified to $((M_1M_2)(M_3M_4))$, the total number of operations changes from $C = m_1m_2m_3 + m_1m_3m_4 + m_1m_4m_5$ to $C' = m_1m_2m_3 + m_3m_4m_5 + m_1m_3m_5$. Therefore, the amount of increased computation ΔC is as follows:

$$\Delta C = m_3m_4m_5 + m_1m_3m_5 - m_1m_3m_4 - m_1m_4m_5.$$

In general, when we have a product sequence such as in Fig. 4, the amount of increased computation for multiplying $M_y \times M_{y+1}$ concurrently with $M_x \times M_{x+1}$ is as follows:

$$\Delta C = m_{y+1}m_{y+2}(m_y - m_z) + m_zm_y(m_{y+2} - m_{y+1}).$$

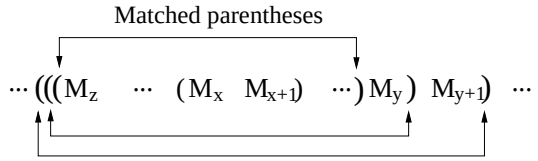

Figure 4: A snapshot of a product sequence.

In this equation, m_z represents the row of the intermediate matrix (or matrix M_z itself) that is going to be multiplied with the result of $M_y \times M_{y+1}$. In other words, there is a left parenthesis to the left of matrix M_z that matches the right parenthesis on the right side of matrix M_{y+1}. In the case of $y < z$, i.e., $(M_y(M_{y+1}(\cdots(M_xM_{x+1})\cdots M_z)))$, the amount of increased computation is

$$\Delta C = m_{y+1}m_{y+2}(m_y - m_{z+1}) + m_{z+1}m_y(m_{y+2} - m_{y+1}).$$

Finding m_z (or m_{z+1}), which is very important for the analysis of ΔC, can be done by traversing sequence tree L. If both M_y and M_{y+1} are right children, then M_z is searched by traversing the left child recursively from the parent node of M_y. Similarly, if both are left children, then M_z is searched by traversing the right child from the parent node of M_y.

In the case where $p_{i,j}$ processors are allocated to the matrix product (M_xM_{x+1}) but all $p_{i,j}$ processors cannot be utilized by the matrix product (i.e., $m_xm_{x+2} < p_{i,j}$), then we try to modify the product sequence L. At that time, we can decide to modify the current sequence by the following lemma.

Lemma 2: If a leaf product (M_xM_{x+1}) has a candidate product (M_yM_{y+1}) and the DPA algorithm will allocate p_x and p_y processors to the two matrix products respectively, then evaluation using the modified sequence reduces the evaluation time when $\Delta C < \min(\Phi(m_x, m_{x+1}, m_{x+2},\ m_xm_{x+2}) \times (p_x + p_y - m_xm_{x+2}),\ m_ym_{y+1}m_{y+2})$.

Proof: There are two necessary conditions for modifying a product sequence to have better performance. The first condition is that the utilization of idle processors (i.e., $p_x + p_y - m_xm_{x+2}$) should be greater than the computation increase resulting from modifying the product sequence tree. The work of idle processors can be estimated as the product of the number of utilized processors and the available time for these processors. Hence, the following condition should be satisfied:

$$\Delta C < \Phi(m_x, m_{x+1}, m_{x+2}, m_xm_{x+2}) \times (p_x + p_y - m_xm_{x+2}).$$

Also, the amount of computation given to idle processors, which is the time for multiplying (M_yM_{y+1}), should be more than ΔC. Therefore, the other condition to be satisfied is

$$\Delta C < m_ym_{y+1}m_{y+2}.$$

Thus, the lemma is satisfied. □

If a candidate product $(M_y M_{y+1})$ satisfies Lemma 2, then it would be better to change product sequence $L_{i,j}$ to multiply the candidate product concurrently with $(M_x M_{x+1})$. This means that the unallocated idle processors can do more work than the increased computation required by the change in the product sequence.

When the candidate product is found, the subtree $L_{i,j}$ is modified and the processors $p_{i,j}$ are redistributed among the products in $L_{i,j}$ (including the candidate product). Also, processors are allocated proportionally to each product. This results in an enhancement of the overall system performance due to an increase in processor efficiency.

4.4 The Proposed Scheduling Algorithm

The proposed scheduling algorithm for evaluating a matrix chain product is formulated as follows.

Two-Pass Matrix Chain Scheduling Algorithm

Stage–1 *MCOP*

1. Find the optimal product sequence for the MCOP by using a parallel algorithm.
2. Generate the sequence tree L.

Stage–2 *Top-Down Processor Assignment*

1. Initialize $i = 1$, $j = n$, $p_{i,j} = P$.
2. If i is not $S[i,j]$, then allocate $p_{i,j} \times W[i, S[i,j]]/(W[i, S[i,j]] + W[S[i,j]+1, j])$ processors to $L_{i,S[i,j]}$.
3. If j is not $S[i,j]+1$, then allocate $p_{i,j} \times W[S[i,j]+1, j]/(W[i, S[i,j]] + W[S[i,j]+1, j])$ to $L_{S[i,j]+1,j}$.
4. If i is $j+1$ or j, then finish this stage; otherwise, call this algorithm recursively, once with $i = i, j = S[i,j]$ and once with $i = S[i,j]+1, j = j$.

Stage–3 *Bottom-Up Concurrent Execution*

For all leaf products, execute the following steps until there are no more unscheduled leaf products.

1. Let $(M_k M_{k+1})$ be a leaf product and $p_{k,k+1}$ be the number of processors allocated to the leaf product. If $p_{k,k+1} < m_k m_{k+2}$, then go to 5.
2. Find a candidate product by tracing ancestors of the leaf product using postorder traversal. If there is no such candidate product, go to 5.
3. Let the product $(M_l M_{l+1})$ be a candidate product found by tracing ancestors of the leaf product $(M_k M_{k+1})$. Check whether the candidate product satisfies Lemma 2. If not, go to 2.
4. Modify the sequence tree such that the candidate product $(M_l M_{l+1})$ can be executed concurrently with $(M_k M_{k+1})$. Reallocate processors $p_{k,k+1}$ using the DPA algorithm and go to 1 for each leaf product of the two split subtrees.
5. Schedule the leaf product on $\min(p_{i,j},\ m_k m_{k+2})$ processors. Set the parent of the leaf product as a new leaf product.

The scheduling algorithm starts from the sequence for the MCOP and then tries to modify the sequence to increase the degree of concurrency. The evaluation time of a matrix chain product is affected by the amount of computation required and the degree of concurrency. The amount of computation required is minimized by using the MCOP sequence, and the degree of concurrency is maximized with a complete binary tree. The optimal product sequence with the minimum evaluation time has a form that is somewhere in between that of the MCOP sequence and the complete binary tree. The proposed scheduling algorithm moves from the MCOP sequence to a near-optimal sequence.

For purposes of efficiency, the scheduling algorithm modifies the current product sequence when the candidate product satisfies Lemma 2. Even though we can select the most suitable candidate among a number of candidates satisfying Lemma 2 by traversing the sequence tree, the scheduling algorithm uses the first candidate that satisfies Lemma 2 in order to minimize the scheduling time.

4.5 Algorithm Complexity

The time complexity of the proposed algorithm is analyzed as follows. Stage-1 and Stage-2 can be done within $O(n)$ time. In Stage-3, to reduce the time for checking Lemma 2, we pass the information of the skewed point (M_z for ΔC) to the next parent product when we are tracing the ancestors from a leaf product, as shown in Fig. 5. Then we do not need to traverse the children of a candidate product to find M_z since M_z is passed from the previously traced child. This allows Step 3 of Stage-3 to be completed in constant time. The maximum number of products traced to check concurrent execution is $(n-3)$. The

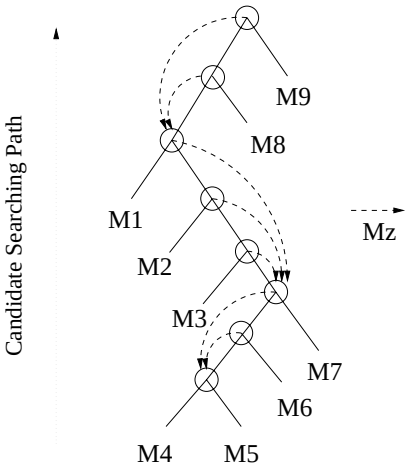

Figure 5: Candidate searching and passing the information of the skewed point M_z.

total number of products that may be traced in Stage-3 is $(n-3)+(n-4)+\cdots+1 = (n-2)(n-3)/2 = O(n^2)$. Also, in Step 4 of Stage-3, the number of sequence modifications is at most $(n-4)$. Since the DPA algorithm for two matrix products takes $O(m)$, the time complexity for Step 4 of Stage-3 is $O((n-4)m)$. Therefore, the time complexity of the proposed algorithm is $O(n^2+nm)$.

4.6 Scheduling Example

The following simple example illustrates the proposed scheduling algorithm, and compares the expected evaluation time of the product sequence by the proposed algorithm with that of the optimal product sequence for the MCOP.

In a system with 50 processors, let us consider the case of evaluating a chain of matrix products with 5 matrices. Given 5 matrices, M_1: 6×2, M_2: 2×7, M_3: 7×5, M_4: 5×7, M_5: 7×8, Stage-1 finds the product sequence with the minimum number of operations for the MCOP as $(M_1(((M_2M_3)M_4)M_5))$. The MCOP sequence tree is represented as the left tree of Fig. 6. In Stage-2, we assign 50 processors to each matrix product. In Stage-3, since the leaf product (M_2M_3) cannot utilize the 50 allocated processors, we try to modify the product sequence. The product (M_4M_5) is found to be a candidate product. By checking Lemma 2, we get $p_x = 10$, $p_y = 40$ using the DPA algorithm, $\Delta C = 7\times 8(5-2)+2\times 5(8-7) = 178$, and $\Phi(2,7,5,10) = 7$. Since $\Delta C = 178 < \min(7\times(10+40-10), 5\times 7\times 8) = 280$, the product sequence is modified to $(M_1((M_2M_3)(M_4M_5)))$ as shown in the right tree of Fig. 6.

Let us compare the evaluation time of the optimal MCOP sequence with that of the product sequence found by the proposed scheduling algorithm. When we evaluate the

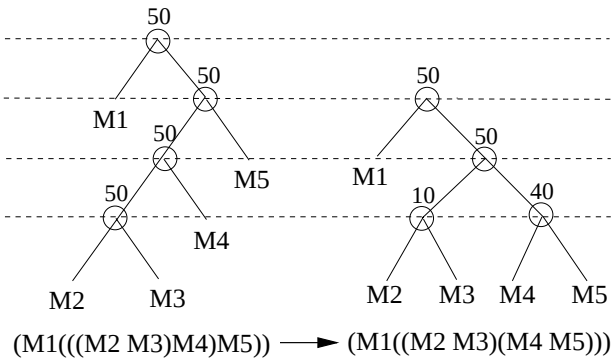

Figure 6: The sequence trees for the MCOP sequence (left), and the proposed scheduling algorithm (right).

matrix chain by the optimal MCOP sequence, it takes $(2 \times 7 \times 5)/\min(50, 2 \times 5) + (2 \times 5 \times 7)/\min(50, 2 \times 7) + (2 \times 7 \times 8)/\min(50, 2 \times 8) + (6 \times 2 \times 8)/\min(50, 6 \times 8) = 21$ units of time. The evaluation time of the product sequence by the proposed scheduling algorithm is $\max(2 \times 7 \times 5/\min(10, 2 \times 5), 5 \times 7 \times 8/\min(40, 5 \times 8)) + 2 \times 5 \times 8/\min(50, 2 \times 8) + 6 \times 2 \times 8/\min(50, 6 \times 8) = 7 + 5 + 2 = 14$ units of time. The product sequence by the proposed scheduling algorithm requires 526 operations, which is 178 operations more than the MCOP sequence having the minimum number of operations (348). However, the proposed algorithm requires less time than the MCOP sequence. This is due to the concurrent execution of multiple matrix products which increases system efficiency and reduces the total evaluation time.

5 Performance Comparison

5.1 Expected Performance

In this subsection, the performance of the evaluation sequence found by the proposed algorithm is analyzed and compared with the optimal sequence for the MCOP. Evaluation time, speedup, and efficiency are used as the performance metrics. The upper bound and the lower bound of evaluation using the sequences found by the proposed method are compared with evaluation by the MCOP sequence in terms of these metrics.

If a schedule is given for $\mathcal{M}$, the evaluation sequence of $\mathcal{M}$ can be represented as a tree with $n-1$ tasks. Let us denote task i as v_i, the number of operations in v_i as w_i, and the maximum number of processors allocated to v_i as p_i^{max}. From the previous assumption, if v_i is $(m_{i_1}, m_{i_2}, m_{i_3})$, then $w_i = m_{i_1} m_{i_2} m_{i_3}$, and $p_i^{max} = m_{i_1} m_{i_3}$.

The evaluation time of $\mathcal{M}$ in a single processor system is the same as the number of operations. Therefore, the evaluation time, represented as $T_{SEQ}(1)$, is bound by the minimum number of operations of a MCOP sequence. If the number of operations of the MCOP sequence is W, $T_{SEQ}(1)$ is found as follows:

$$T_{SEQ}(1) = \sum_{i=1}^{n-1} \Phi(m_{i_1}, m_{i_2}, m_{i_3}, 1) = \sum_{i=1}^{n-1} w_i = W.$$

On a P-processor parallel system, the evaluation time of the MCOP sequence in which each matrix product is parallelized one at a time, denoted by $T_{MCOP}(P)$, is calculated as follows:

$$T_{MCOP}(P) = \sum_{i=1}^{n-1} \Phi(m_{i_1}, m_{i_2}, m_{i_3}, \min(P, p_i^{max})) = \sum_{i=1}^{n-1} \frac{w_i}{\min(P, p_i^{max})}.$$

In the case of $p_i^{max} \geq P$ for all $1 \leq i \leq n-1$, $T_{MCOP}(P)$ becomes

$$T_{MCOP}(P) = \frac{W}{P}.$$

However, in general, there exist cases in which $p_i^{max} \leq P$ for some i and $T_{MCOP}(P)$ is larger than W/P.

The proposed two-pass scheduling method for the MCSP allocates processors to the MCOP sequence tree by top-down proportional assignment and modifies the MCOP sequence when it results in a reduction of the evaluation time. Therefore, the evaluation time of the proposed method, represented as $T_{MCSP}(P)$, is not more than the evaluation time of the MCOP sequence with the top-down processor assignment. When $p_{i,j}$ processors are allocated to an $L_{i,j}$ subtree in the MCOP sequence tree L, the evaluation time $T_{MCSP}(P)$ is found as $T_{1,n}(P)$ in the following recurrence relation:

$$T_{i,j}(P_{i,j}) = \max_{L_{i,k} \in L} (T_{i,k}(p_{i,k}), T_{k+1,j}(p_{k+1,j})) + \Phi(m_i, m_{k+1}, m_{j+1}, p_{i,j}).$$

Without loss of generality, let us assume that $p_{i,j}^{max} = m_i m_{j+1}$. In the case of $p_{i,j}^{max} \geq p_{i,j}$ for all $L_{i,j}$ in L,

$$T_{MCSP}(P) = \frac{W}{P}.$$

Let the root node task of subtree $L_{i,j}$ be v_r such that $v_r = (m_i, m_{k+1}, m_{j+1})$ and $p_r^{max} = p_{i,j}^{max} = m_i m_{j+1}$. Usually, $p_{i,j}^{max} \leq p_{i,j}$ is a sufficient condition to make $p_r^{max} \leq P$ remain

true. Thus, in the worst case, the upper bound of $T_{MCSP}(P)$ is $T_{MCOP}(P)$ so that the following relation is satisfied:

$$T_{MCSP}(P) \leq T_{MCOP}(P).$$

Now let us compare the best case performance of the proposed method with the MCOP sequence. Assume that each task v_i for $1 \leq i \leq n-1$ can utilize a maximum of P/k processors in a P-processor system, i.e., $p_i^{max} = P/k$. Then the performance using the MCOP sequence is as follows:

$$\begin{aligned} T_{MCOP}(P) &= \frac{kW}{P}, \\ S_{MCOP}(P) &= \frac{P}{k}, \\ E_{MCOP}(P) &= \frac{1}{k}. \end{aligned}$$

The best case for the proposed MCSP schedule is when the sequence tree becomes a complete binary tree after two-pass scheduling. The height of a complete binary tree with n nodes is $\log(n)$. It is well known that the completion time of a parallel task is bound by the critical path in a given task graph. Let the tasks on the critical path of the MCSP sequence be $v_1, v_2, .., v_{\log(n)}$. If we assume that the scheduled tree is a well-balanced binary tree, then each half of the processors is allocated to its left and right child tasks, respectively. Therefore, the tasks whose depths are not more than $\log(k)$ will run on P/k processors. Other tasks with depth i ($\log(k) < i \leq \log(n)$) will run on $(P/k)(1/2)^{i-\log(k)}$ processors. Therefore, $T_{MCSP}(P)$ is transformed as follows:

$$T_{MCSP}(P) = \sum_{i=1}^{\log(k)} \frac{w_i}{P/k} + \sum_{i=\log(k)+1}^{\log(n)} \frac{w_i}{(P/k) \cdot 1/2^{i-\log(k)}}.$$

Let us denote the number of operations scheduled by the proposed method as W_{MCSP}, and assume that all tasks have the same amount of work, $w_i = W_{MCSP}/(n-1)$. Then the following equations are satisfied.

$$\begin{aligned} T_{MCSP}(P) &= \frac{k}{P} \cdot \frac{\log(k) W_{MCSP} k}{(n-1)} + \frac{W_{MCSP}}{(n-1)P} \left(2^{\log(n)-\log(k)} - 1\right) \\ &= \frac{k \log(k) W_{MCSP}}{(n-1)P} + \frac{k(\frac{n}{k} - 1) W_{MCSP}}{(n-1)P} \\ &= \frac{(n + k\log(k) - k) W_{MCSP}}{(n-1)P}. \end{aligned}$$

$$S_{MCSP}(P) = \frac{(n-1)P}{(n+k\log(k)-k)}.$$

$$E_{MCSP}(P) = \frac{(n-1)}{(n+k\log(k)-k)}.$$

The best case is when the number of increased operations, with respect to the MCOP sequence, is equal to zero, e.g., $\Delta C = 0$, so that W_{MCSP} is equal to the minimum W. Therefore, the upper bound of the performance compared with the MCOP sequence is as follows:

$$\frac{T_{MCOP}(P)}{T_{MCSP}(P)} = \frac{kW/P}{(n+k\log(k)-k)\,W/(n-1)P} = \frac{k(n-1)}{n+k\log(k)-k}. \tag{5.1}$$

For example, if each task can only utilize 25% of the processors in a system ($k = 4$) and the number of matrices in a chain is 100 ($n = 100$), then $T_{MCOP}(P)/T_{MCSP}(P) = 3.8$. The proposed method evaluates matrix chain products 3.8 times faster than the method obtained by parallelizing the MCOP sequence.

From Eq. (5.1), when $k = 1$, the performance of the proposed method is bound by the evaluation of the MCOP sequence. As k or n increases, the upper bound of Eq. (5.1) increases. As n increases to infinity, the proposed scheme can run almost k times faster than the method using the MCOP sequence.

From the above discussion, when the number of matrices becomes larger and larger, and/or when the size of the matrices becomes smaller and smaller so that only part of the system is required to execute one product, the proposed method greatly outperforms the simple MCOP-based method.

5.2 Experimental Results

In this subsection, we compare the performance of various evaluation methods through experiments on a parallel system. The methods are as follows:

- **Linear:** evaluate by parallelizing each matrix product from the first product (M_1M_2) to the last one sequentially.
- **MCOP-Seq:** evaluate by parallelizing each matrix product using the MCOP sequence sequentially.
- **MCOP-Con:** evaluate by parallelizing each matrix product using the MCOP sequence, but execute independent matrix products concurrently by allocating the maximum number of processors.
- **MCSP-BT:** evaluate by the MCOP-Con method, but when there are idle processors during execution, try to modify the sequence using Lemma 2.

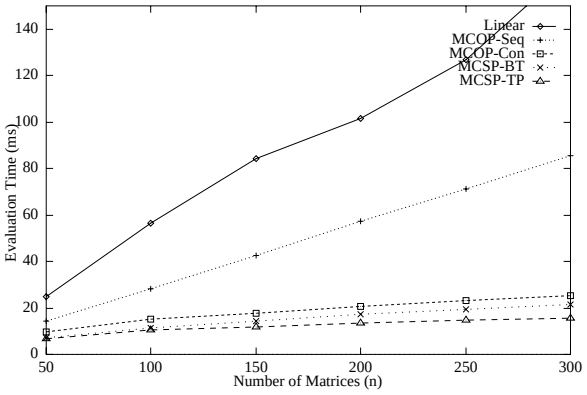

Figure 7: Evaluation time comparison by varying the number of matrices with $m = 50,\ P = 512$.

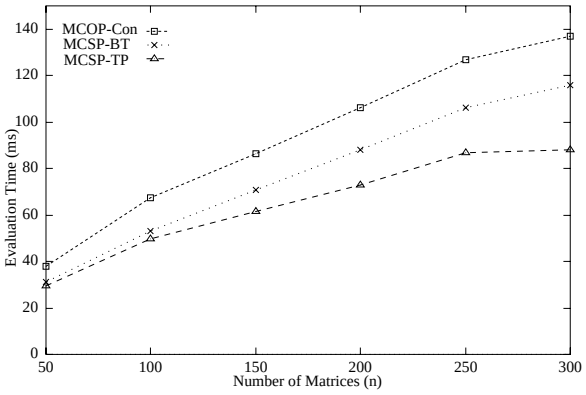

Figure 8: Evaluation time comparison by varying the number of matrices with $m = 100,\ P = 512$.

- **MCSP-TP:** evaluate by the proposed scheduling algorithm.

We experimented on the Fujitsu AP1000 parallel system, which is a distributed-memory MIMD machine with 512 cells. Each cell is a SPARC processor with 16MB of memory. The AP1000 system has three independent networks: the B-net for broadcasting, the T-net for point-to-point communication, and the S-net for synchronization. The processors are interconnected by the T-net, a two dimensional torus network with 25Mbytes/sec/port. The host computer and the processors are connected by the B-net, a broadcasting network with 50Mbytes/sec, and by the S-net for synchronization.

The evaluation times of randomly generated matrix product chains are measured for each scheduling method. Since the initial matrix loading times are highly dependent on system characteristics such as the communication link speed and the interconnection network, the loading times are excluded in the statistics of the evaluation time. In fact, the proposed algorithm can spend less time than the sequential evaluation methods for distributing matrices to processors since several matrices can be loaded at the same time.[3] The results shown in this section are the averages of 100 experiments.

Fig. 7 and Fig. 8 show the evaluation time as a function of the number of matrices being multiplied. In Fig. 7, a chain of matrix products is generated randomly for matrices varying in size from 1 to 50 ($m = 50$), and executed on a system with 512 processors ($P = 512$). The upper two lines represent the evaluation times of sequential evaluation using Linear and MCOP-Seq, and the lower three lines represent the evaluation times of the schedule

[3]Since some parallel computers such as the Fujitsu AP1000 support collective communication schemes including scatter and gather, we can reduce the matrix loading time by using these schemes.

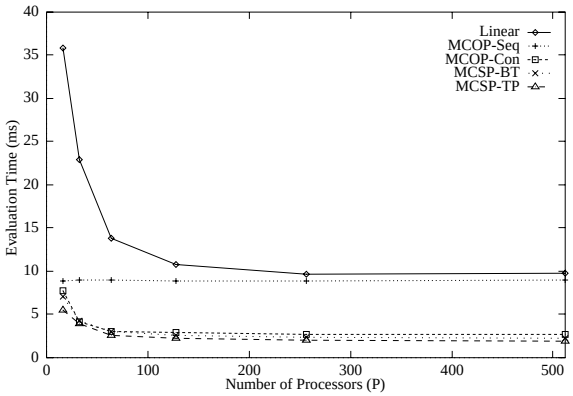

Figure 9: Evaluation time comparison for different numbers of processors with $n = 100,\ m = 20$.

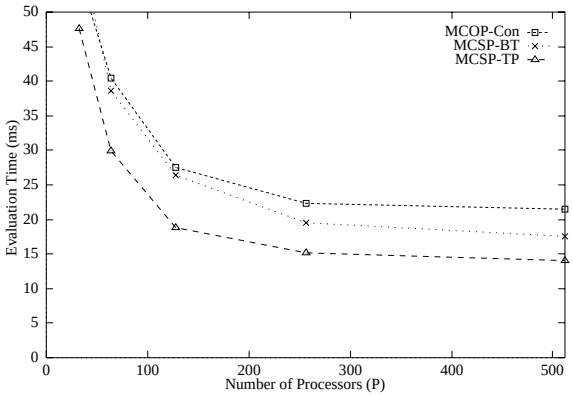

Figure 10: Evaluation time comparison for different numbers of processors with $n = 200,\ m = 50$.

sequences found by using MCOP-Con, MCSP-BT, and MCSP-TP. From the comparison of the execution time of MCOP-Seq with that of Linear, it can be seen that simply reducing the amount of computation does not greatly decrease the evaluation time. However, when we allow concurrent execution, we can improve performance further. Therefore, we confirm that the evaluation time of a chain of matrix products is greatly affected by task scheduling. In Fig. 8, we experimented with larger matrices ($m = 100$). The upper line is the evaluation by MCOP-Con, the middle line is by MCSP-BT, and the lower line is by MCSP-TP. As the number of matrices in a matrix chain increases, the proposed MCSP-TP algorithm shows a greater performance gain.

In Fig. 9 and Fig. 10, the evaluation time between different numbers of processors in a system is compared. In Fig. 9, the matrix chain consists of 100 matrices ($n = 100$) whose sizes vary from 1 to 20 ($m = 20$). The upper two lines represent Linear and MCOP-Seq, which are the sequential evaluation methods. The lower three lines represent MCOP-Con, MCSP-BT, and MCSP-TP, respectively, which are the concurrent evaluation methods. As the number of processors increases, the evaluation time by Linear decreases and becomes close to MCOP-Seq. This implies that the reduction of computation has a limited effect on reducing the evaluation time. In Fig. 10, we measured the evaluation times of three concurrent execution methods with $n = 200,\ m = 50$. The upper line represents the evaluation by MCOP-Con, the middle line represents the evaluation by MCSP-BT and the lower line represents the evaluation by MCSP-TP. As the number of processors increases, the evaluation times of MCSP-BT and MCSP-TP decrease more than that of MCOP-Con. This implies that sequence modification to increase the degree of concurrency improves the performance by utilizing processors efficiently. Another aspect we can see from this result

is that the number of processors does not affect the performance significantly, but that the processor scheduling policy is much more important than computational reduction in improving the performance of evaluating a chain of matrix products.

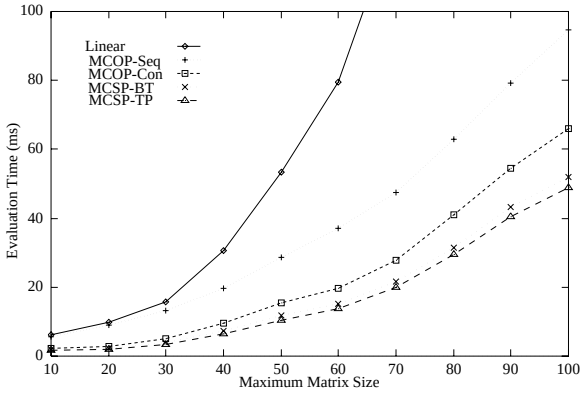

Figure 11: Evaluation time comparison by varying the largest dimension of matrices with $n = 100,\ P = 512$.

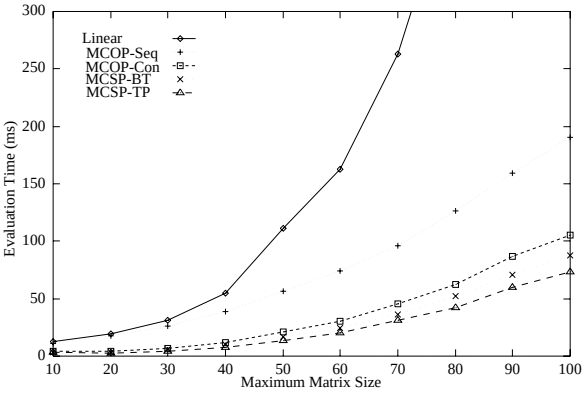

Figure 12: Evaluation time comparison by varying the largest dimension of matrices with $n = 200,\ P = 512$.

The experimental results with varying distributions of matrix sizes (m) is shown in Fig. 11 and Fig. 12. The evaluation times are measured with $P = 512$ for a chain of $n = 100$ or $n = 200$ matrices, respectively. As shown in the figures, as the variance in the matrix size gets larger, MCOP-Seq has better performance than Linear. This is caused by the fact that the amount of computation is reduced greatly by the MCOP sequence, and that there are small numbers of idle processors during their execution when evaluating a chain of large matrices. However, we notice that the proposed MCSP-TP method still outperforms all other methods. These experiments imply that the proposed MCSP-TP method is still effective for larger matrices due to performance improvement through concurrent execution (with more concurrency than the other methods), even though there are rare exceptions.

In Table 1 and Table 2, we measured the amount of computation, system utilization, and evaluation time for each evaluation method. Table 1 shows the results when $n = 100$, $m = 50$ and $P = 512$. Even though the amount of computation using MCOP-Seq is almost 22 times less than that of Linear, the evaluation time is reduced by only 44%. We observe that the system utilization of MCOP-Seq (5.59%) is significantly lower than that of Linear (58.16%). Evaluation by MCSP-BT and MCSP-TP requires more computation than that of MCOP-Con, but the evaluation time decreases due to the efficient use of processors. Table 2 shows the results with $n = 100$, $m = 100$ and $P = 512$. We observed behavior similar to that in Table 1. However, note that MCSP-BT utilizes even more processors than MCSP-TP, and still has an evaluation time that is larger than that of MCSP-TP. This

Table 1: Comparison with respect to computation amounts, system utilization, and evaluation time when $n = 100$, $m = 50$ and $P = 512$.

Scheduling Algorithm	Computation Amount	System Utilization	Evaluation Time
Linear	1235145	58.16	50.03
MCOP-Seq	57239	5.59	28.07
MCOP-Con	57239	11.16	14.96
MCSP-BT	124506	27.75	11.39
MCSP-TP	83083	21.46	10.12

Table 2: Comparison with respect to computation amounts, system utilization, and evaluation time when $n = 100$, $m = 100$ and $P = 512$.

Scheduling Algorithm	Computation Amount	System Utilization	Evaluation Time
Linear	11237091	75.39	454.63
MCOP-Seq	293206	14.75	96.77
MCOP-Con	293206	21.71	67.22
MCSP-BT	513483	36.91	53.29
MCSP-TP	349948	28.04	49.88

result implies that the proposed MCSP-TP algorithm uses processors more efficiently and effectively than MCSP-BT.

From the above experiments, we can conclude the following.

- When evaluating a chain of matrices on a parallel system, simply reducing the number of required operations does not greatly decrease the evaluation time.
- Concurrent execution of independent multiple matrix products compensates for inefficient processor usage with parallel processing, and increases the system efficiency so that performance improves greatly.
- When the number of processors becomes larger or when the number of matrices in a matrix chain increases, evaluation by the proposed scheduling algorithm MCSP-TP progressively outperforms other methods such as Linear, MCOP-Seq, MCOP-Con and MCSP-BT.
- Even when the size of the matrices is quite large so that there are no idle processors during their evaluation, MCSP-TP is still effective due to the concurrent execution of independent matrix products with the top-down processor assignment.
- When evaluating a chain with small matrices on many processors, sequence modification to increase system efficiency greatly reduces the evaluation time.
- For matrix chain products, efficient scheduling is better than increasing the number of processors.

6 Extension of the MCSP

Throughout the previous sections, a few assumptions have been made for simplicity. In this section, we discuss how to adjust these assumptions for better performance on a specific system. Also, we show a few problems that are manageable by the MCSP approach.

6.1 Practical Considerations of the MCSP

Depending on the computing environment for evaluating $\mathcal{M}$, the parameters used in the proposed method are easily adjustable. The parameters are the execution time function, the maximum number of processors allocated to a product, and the minimum number of processors allocated to a product. For ease of discussion, let us denote these parameters as $\Phi(v_i, p_i)$, p_i^{max}, and p_i^{min}, respectively, for a given product v_i.

- **Matrix multiplication algorithm:** Many parallel matrix multiplication algorithms have been suggested for various architectures [24, 23]. Depending on which algorithm is used for the MCSP, the execution time function $\Phi(v_i, p_i)$ can be specified in detail. Thus, $\Phi(v_i, p_i)$ can be changed for taking the parallel processing overheads into account. The parallel processing overheads include load imbalance, communication cost, and synchronization.

- **Parallel system architecture:** The parallel system used for evaluating $\mathcal{M}$ also affects the parameters. For example, on a shared memory multiprocessor system, the communication cost is negligible, but on a message-passing distributed system, the communication cost should be considered in $\Phi(v_i, p_i)$. Consequently the parameter p_i^{max} will be more restricted in a distributed system.

- **Input matrix size:** One limitation in using the proposed algorithm is that the method is not very effective with a chain of large matrices. However, a more elaborate description of $\Phi(v_i, p_i)$ from the above parallel algorithm-architecture parameters will compensate for this limitation by using a reduced value of p_i^{max}. Also, even in the case where any one product is large enough to utilize the whole system, the top-down processor assignment leads to better performance than sequential evaluation.

  Another parameter to be considered in the MCSP is the minimum number of processors (p_i^{min}) to be allocated to a product v_i. Basically, we assumed that any product can be executable on one processor, i.e., $p_i^{min} = 1$. However, depending on the size of the matrices, the minimum number of processors to be allocated to a product v_i can be restricted to be $p_i^{min} > 1$. This is mainly due to the memory capacity of a single processor. We can consider the parameter p_i^{min} in the top-down processor assignment stage.

- **Matrix distribution cost:** In the proposed method, the cost for the initial distribution of matrices among the processors was excluded even though this cost is not negligible in a distributed-memory system.[4] Of course, the cost of the proposed method is definitely less than sequential evaluation by the MCOP sequence. This is mainly due to the fact that the proposed method runs on fewer processors than sequential evaluation. The effect of this cost for scheduling the MCSP sequence is considered for future work.

[4] The cost is negligible in a shared-memory multiprocessor system.

6.2 MCSP Applications

We can extend the MCSP to a few sparse matrix problems. The MCSP has characteristics such that it can be represented as a tree precedence task graph with many equivalent tree graphs, and the task graph determines the number of required operations and the degree of concurrency. The proposed method for the MCSP can be applied to evaluate a chain of square matrices with sparsity and to factorize a large sparse matrix in parallel systems.

- **Sparse matrix chain products:** Sparse matrix multiplications require a different number of operations with respect to the sparsity structure. For multiplying an $n \times n$ matrix A with non-zero density d_1 by an $n \times n$ matrix B with non-zero density d_2, the number of required operations is $d_1 d_2 n^3$ [30]. Also, for multiplying a tridiagonal matrix with any type of matrix, the required operations vary depending on the type of matrices [31]. Such a sparse matrix multiplication has the same characteristics as the MCSP. A different evaluation sequence results in a different number of required operations. We can extend the MCSP to the evaluation of a chain of square matrices with sparsity.

- **Sparse matrix factorization:** Elimination trees are used extensively in sparse matrix factorization because they present the sequence of computation and parallelism [32]. Since there are many equivalent elimination trees with different parallelization structures [33], we can follow the MCSP approach for this sparse factorization problem.

The above items have been studied extensively for a long time, and each problem can be approached as a separate research subject.

7 Summary and Conclusion

In this paper, we introduced the matrix chain scheduling problem (MCSP) and proposed a heuristic scheduling algorithm for the MCSP. The proposed algorithm schedules matrix products to processors with the objective of enhancing concurrency at the expense of a slight increase in the required number of operations when compared to the optimal product sequence found for the matrix chain ordering problem (MCOP). We have shown that performance is significantly enhanced by the proposed algorithm using experiments on the Fujitsu AP1000 parallel system. As a result, we can confirm that efficient processor scheduling is much more important than simply reducing the total number of operations

when evaluating a matrix chain product in parallel systems. Given a parallel system with a large number of processors or a matrix product chain involving many matrices, evaluation by the proposed method greatly outperforms the parallel evaluation method which uses the optimal product sequence found for the MCOP. The main contribution of this work is the formalization of the MCSP and the introduction of a processor allocation and task scheduling algorithm that results in a significant performance improvement when evaluating matrix chain products in parallel systems. We are currently working on extending this algorithm to evaluate a chain of square matrices in the form of sparse matrices or band matrices, and to scheduling of parallel matrix factorization using elimination trees. Also, we plan to study generalizing the MCSP to scalable task scheduling on parallel systems.

References

[1] E. Horowitz and S. Sahni, *Fundamentals of Data Structures.* Potomac, MD: Computer Science Press, 1976.

[2] E. Dekel, D. Nassimi, and S. Sahni, "Parallel matrix and graph algorithms," *SIAM J. Comput.*, vol. 10, pp. 657–675, Nov. 1981.

[3] S.-T. Yau and Y. Y. Lu, "Reducing the symmetric matrix eigenvalue problem to matrix multiplications," *SIAM J. Sci. Comput.*, vol. 14, no. 1, pp. 121–136, 1993.

[4] S.-S. Lin, "A chained-matrices approach for parallel computation of continued fractions and its applications," *Journal of Scientific Computing*, vol. 9, no. 1, pp. 65–80, 1994.

[5] V. Kumar, A. Grama, A. Gupta, and G. Karypis, *Introduction to Parallel Computing: design and analysis of parallel algorithms.* The Benjamin/Cummings Publishing Company, Inc., 1994.

[6] A. Chandra, "Computing matrix chain products in near-optimal time," tech. rep., IBM T.J. Watson Res. Ctr., Yorktown Heights, N.Y., 1975. IBM Research Report RC 5625(#24393).

[7] H. Gould, *Bell and Catalan numbers.* Combinatorial Research Institute, Morgantown, WV., June 1977.

[8] S. Godbole, "An efficient computation of matrix chain products," *IEEE Trans. on Computers*, pp. 864–866, Sept. 1973.

[9] F. Chin, "An $O(n)$ algorithm for determining a near-optimal computation order of matrix chain product," *Comm. ACM*, pp. 544–549, 1978.

[10] T. Hu and M. Shing, "Computation of matrix chain products. part I," *SIAM J. Comput.*, vol. 11, pp. 362–373, May 1982.

[11] T. Hu and M. Shing, "Computation of matrix chain products. part II," *SIAM J. Comput.*, vol. 13, pp. 228–251, May 1984.

[12] P. Ramanan, "A new lower bound technique and its application: Tight lower bound for a polygon triangulation problem," *SIAM J. Comput.*, vol. 23, pp. 834–851, Aug. 1994.

[13] L. Valiant, S. Skyum, S. Berkowitz, and C. Rackoff, "Fast parallel computation of polynomials using few processors," *SIAM J. Comput.*, vol. 12, pp. 641–644, 1983.

[14] W. Rytter, "Note on efficient parallel computations for some dynamic programming problems," *Theoret. Comp. Sci.*, vol. 59, pp. 297–307, 1988.

[15] S.-H. S. Huang, H. Liu, and V. Viswanathan, "Parallel dynamic programming," *IEEE Trans. on Parallel and Distributed Systems*, vol. 5, pp. 326–328, Mar. 1994.

[16] P. G. Bradford, G. J. Rawlins, and G. E. Shannon, "Efficient matrix chain ordering in polylog time," *SIAM J. Comput.*, vol. 27, no. 2, pp. 466–490, 1998.

[17] A. Czumaj, "Parallel algorithm for the matrix chain product and the optimal triangulation problems," in *Proc. of Symp. on Theoret. Aspects of Computer Science*, pp. 294–305, Springer Verlag, 1993.

[18] A. Czumaj, "Very fast approximation of the matrix chain product problem," *J. Algorithms*, vol. 21, no. 1, pp. 71–79, 1996.

[19] P. Ramanan, "An efficient parallel algorithm for the matrix chain product problem," *SIAM J. Comput.*, vol. 25, pp. 874–893, Aug. 1996.

[20] V. Strassen, "Gaussian elimination is not optimal," *Numer. Math.*, vol. 13, pp. 354–356, 1969.

[21] G. H. Golub and C. F. V. Loan, *Matrix Computations*. Baltimore: The Johns Hopkins University Press, 2 ed., 1989.

[22] N.-K. Tsao, "Error complexity analysis of algorithms for matrix multiplication and matrix chain product," *IEEE Trans. on Computers*, vol. C-30, pp. 758–771, Oct. 1981.

[23] C. Puglisi, "Parallel algorithms and architectures for matrix multiplication," *Comput. Math. Appl.*, vol. 17, no. 12, pp. 1567–1572, 1989.

[24] A. Gupta and V. Kumar, "Scalability of parallel algorithms for matrix multiplication," in *Proc. of Int. Conf. on Parallel Processing*, pp. 115–123, 1993.

[25] R. Krishnamurti and E. Ma, "An approximation algorithm for scheduling tasks on varying partition sizes in partitionable multiprocessor systems," *IEEE Trans. on Computers*, vol. 41, pp. 1572–1579, Dec. 1992.

[26] S. G. Akl, *The Design and Analysis of Parallel Algorithms.* Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1989.

[27] V. Sarkar, *Partitioning and Scheduling Parallel Programs for Multiprocessors.* Cambridge (MA): The MIT Press, 1989.

[28] C. D. Polychronopoulos and U. Banerjee, "Speedup bounds and processor allocation for parallel programs on multiprocessors," in *Proc. of Int. Conf. on Parallel Processing*, pp. 961–968, 1986.

[29] C. D. Polychronopoulos and U. Banerjee, "Processor allocation for horizontal and vertical parallelism and related speedup bounds," *IEEE Trans. on Computers*, vol. C-36, pp. 410–420, Apr. 1987.

[30] A. Schoor, "Fast algorithm for sparse matrix multiplication," *Information Processing Letters*, vol. 15, no. 2, pp. 87–89, 1982.

[31] J. Takche, "Complexities of special matrix multiplication problems," *Comput. Math. Applic.*, vol. 15, no. 12, pp. 977–989, 1988.

[32] J. W. H. Liu, "The role of elimination trees in sparse factorization," *SIAM J. Matrix Anal. Appl.*, vol. 11, pp. 134–172, Jan. 1990.

[33] J. W. H. Liu, "Equivalent sparse matrix reordering by elimination tree rotations," *SIAM J. Sci. Stat. Comput.*, vol. 9, pp. 424–444, May 1988.