Parallelizing matrix chain products

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

Dept. of Computer Science and Engineering
Pohang University of Science and Technology
Pohang 790-784, KOREA

Technical Report CS-HPC-97-003
December, 1997

Parallelizing matrix chain products†

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

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

Technical Report CS-HPC-97-003

Abstract

The problem of finding an optimal product sequence for sequential multiplication 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 MCSP and MCOP is that MCOP considers a product sequence for single processor systems and MCSP considers 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 a minimal evaluation time since each parallelized matrix product may use processors inefficiently. We introduce a processor scheduling algorithm for MCSP which attempts to minimize 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. Experiments on a Fujitsu AP1000 multicomputer show that the proposed algorithm significantly decreases the time required to evaluate a chain of matrix products in actual parallel systems.

Keywords – Processor scheduling, matrix chain product, dynamic programming, parallel matrix multiplication, matrix chain scheduling problem.

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

1 Introduction

Matrix multiplication is a computation intensive part of many commonly used scientific computing applications. In many applications, a chain of matrices is multiplied consecutively [1, 2]. 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 by the associative law of matrix multiplication. An arbitrary product sequence of matrices may be as bad as $\Omega(T_{opt}^3)$ where T_{opt} is the minimum number of operations required to evaluate a chain of matrix products [3]. The matrix chain ordering problem (MCOP) is the problem of 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 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* [4]. Let us refer to the time required to find a product sequence for a chain of matrices as the *ordering time* and the time required to execute the product sequence as the *evaluation time*. There are many works reported for solving MCOP and for reducing the ordering time. MCOP was first reported by Godbole [5] and solved using dynamic programming in $O(n^3)$ time. Chin [6] suggested an approximation algorithm which runs in $O(n)$ time for finding a near-optimal sequence. The optimal sequential algorithm, which runs in $O(n \log(n))$ time, was given by Hu and Shing [7, 8]. This algorithm solves MCOP by solving the equivalent problem of finding an optimal triangulation of a convex polygon. Ramanan [9] presented a simpler algorithm and obtained the tight lower bound of the problem as $\Omega(n \log(n))$.

Many parallel algorithms to reduce the *ordering time* have been studied using the dynamic programming method [10, 11, 12, 13] and the convex polygon triangulation method [14, 15, 16]. Bradford *et al.* [13] 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 [14] proposed a $O(\log^3(n))$ 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 [15], he proposed a faster approximation algorithm which 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 [16] gives an optimal algorithm which runs in $O(\log^4(n))$ time using n processors. Most of the recently proposed parallel algorithms run in polylog time with a linear number of processors.

Now let us consider the *evaluation time* of a chain of matrix products. In a single pro-

cessor system, the evaluation of a chain of matrix products by the optimal product sequence guarantees the minimum evaluation time since the sequence guarantees the minimum number of operations. However, in parallel systems, parallel computation of each matrix product with 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.

When computing a matrix product in a parallel system, having more processors does not always lead to a better performance. The computation time of each processor may be reduced by using a larger number of processors. However, this may cause some processors to be stay idle, which in turn results in degraded performance of the parallel system. To have better performance, a proper number of processors must be allocated to the problem at hand. The proper number of processors for a given matrix product is dependent on various system and matrix characteristics such as the interconnection network used, the communication speed, the matrix size, and the computation amount. This leads us to consider whether multiplying matrices sequentially based on the optimal product sequence using a parallel multiplication algorithm is a good strategy in parallel systems or not. In many applications, it is asserted that the performance of parallel systems decreases as more processors are allocated due to the communication overhead and inefficient use of processors.

In this paper, we formally present the problem of finding the matrix product schedule for parallel systems (MCSP) and analyze the problem complexity of MCSP. We propose an algorithm which finds a matrix product schedule that, while slightly increasing the number of 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 the formal description of the processor scheduling problem for a chain of matrix products and shows that the given problem is NP-Hard. We present a processor allocation method for executing a number of independent matrix products concurrently in Section 3. In Section 4, we propose a matrix chain scheduling algorithm which dramatically reduces the evaluation time of a chain of matrix products by using processors efficiently in parallel systems, and analyze the algorithm. In Section 5, we present the performance of the proposed method and compare with that of sequential matrix products ordered by the optimal product sequence for MCOP using experiments on the Fujitsu AP1000 parallel system. Finally, in Section 6, 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 with 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 amount of computation for evaluating $\mathcal{M}$.
- ΔC : the amount of increased computation 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): 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 single matrix product (m_i, m_j, m_k) when p processors are allocated.
- $D(x)$: the set of divisors of x, i.e., $D(x) = \{d | d \text{ divides } x\}$.

2.2 Problem Description

We consider the problem of finding the optimal schedule with minimum evaluation time of $\mathcal{M} = M_1 \times M_2 \times \cdots \times M_n$ on a P processor parallel system. The number of operations for multiplying 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 in various parallel architectures [20, 21]. The execution time of matrix multiplication depends on the algorithm used and the architecture on which the algorithm runs. However, for more discussion, we assume that the simple parallel algorithm [21] 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 matrices with p processors, the execution time $\Phi(m_i, m_j, m_k, p)$ can be

[1] Even though Strassen's algorithm [17, 18] and its variants perform fewer than n^3 operations for $n \times n$ matrix multiplication, these faster algorithms are not suitable for evaluating matrix chain products due to more erroneous results and larger storage requirements than the usual inner-product type algorithm [19]. Therefore, we assume that the simple algorithm is used so that n^3 operations are required for $n \times n$ matrix multiplication (this was also assumed in other research work on MCOP).

approximated as follows.

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

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. Two partial matrix chains are $(M_i \cdots M_k)$ and $(M_{k+1} \cdots M_j)$ for some k $(i \le k < j)$. The evaluation time of 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 $p_{i,k} + p_{k+1,j} \le p_{ij}$. 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.

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

The problem of finding the schedule which results in the minimum evaluation time $T_{1,n}(P)$ is a problem of 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 problem is defined as follows.

MCSP: *find the product sequence for evaluating a chain of matrix products and the processor schedule for the sequence such that the evaluation time is minimized on a parallel system.*

2.3 MCSP Complexity

Consider the case in which there are sufficient processors for multiplying any number of matrices concurrently. We assume that, for each matrix product which multiplies an $m_i \times m_j$ matrix by an $m_j \times m_k$ matrix, $m_i m_j m_k$ processors are allocated and the computation time is $\log(m_j)$. The problem can be represented as the following recurrence relation and solved in polynomial time using dynamic programming.

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

Therefore, in the case of infinitely many available processors, the problem of finding the schedule for evaluating $\mathcal{M}$ with the minimum time is a polynomial time algorithm. However,

in general, the number of available processors is fixed and not sufficient to allocate the requested number of processors for each product.

Now, let us consider the case when there are P processors in a system. The complexity of MCSP is discussed in the following theorem.

Theorem 1: *MCSP* is NP-hard.

Proof: The nonpreemptive scheduling for precedence-constrained parallel tasks is known as an NP-complete problem [22]. We reduce the precedence-constrained parallel task scheduling problem to MCSP. MCSP finds the product sequence and processor schedule simultaneously which minimizes the evaluation time of a chain of matrix products. When a product sequence of a chain of matrix products is fixed, the problem of finding a processor schedule for the fixed product sequence is equivalent to the precedence-constrained parallel task scheduling problem.

In MCSP, there are matrix chain products whose optimal product sequence is fixed. In other words, the product sequence which minimizes the evaluation time is unique. One obvious case is that in which the optimal sequence for MCOP is a complete binary tree. In such a case, the optimal sequence for MCSP is also a complete binary tree. Then the problem of finding the processor schedule is equivalent to the precedence-constrained parallel task scheduling problem. Therefore, the MCSP includes the precedence-constrained parallel task scheduling problem. Thus, MCSP is NP-hard. □

Since the problem of finding an optimal schedule is NP-Hard, we propose an approximation algorithm in Section 4. The algorithm enhances the performance of evaluating an n-matrix product chain on a parallel system by partitioning the parallel system and concurrently executing several matrix products; 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 what is an efficient processor allocation for executing multiple matrix products concurrently.

3.1 Processor Allocation for a Single Matrix Product

Many parallel matrix multiplication algorithms have been developed in various parallel architectures [18, 21]. 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 an element of the result matrix.[2] Each of n processors executes a multiply 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 to have better efficiency when multiplying two matrices?

Efficiency decreases as the number of processors allocated to a single matrix product increases. Until the number of processors allocated reaches n^2, a speedup of the single matrix product is obtained without much efficiency degradation [24]. However, allocating more than n^2 processors is not cost-effective due to greatly increased overhead and much degraded efficiency. Therefore, we consider allocating at most n^2 processors when multiplying two $n \times n$ matrices. By extension, we also allocate at most $m_i m_k$ processors when multiplying an $m_i \times m_j$ matrix by an $m_j \times m_k$ matrix.

The number of processors allocable to a single matrix product cannot be an arbitrary number for efficient execution. When multiplying an $m_i \times m_j$ matrix by an $m_j \times m_k$ matrix, the number of maximally allocable processors is $m_i m_k$, and the possible number of allocable processors are the divisors of $m_i m_k$. For example, when multiplying a 2×4 matrix by a 4×3 matrix, the number of allocable processors are $1, 2, 3$, and 6. In this case, the load of each processor is balanced such that each processor has the same amount of computation. These numbers are the divisors of 6, which is the maximal number of allocable processors for the given matrix product.

We use the notation $D(x)$ to denote the set of divisors of x, i.e., $D(x) = \{d | d \text{ divides } x\}$. The number of processors allocable to a given matrix product (m_i, m_j, m_k) should be an element in $D(m_i m_k)$, and these elements are not continuous numbers.

[2] On a CRCW PRAM, the matrix multiplication runs in constant time with the assumption on the write conflicts that, when several processors attempt to write numbers in the same location, the sum of the numbers is written in that location [23]. Since the CRCW PRAM with arbitrary number of processors is not realistic in practice, we are not considering that case.

3.2 Processor Allocation for the Concurrent Computation of Multiple Matrix Products

In general, the performance of parallel systems is maximized by executing as many tasks as possible concurrently while using a few processors for each task [25]. This is mainly due to the fact that the concurrent execution compensates for the performance loss that normally results from parallelizing a task. 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" [26, 27]. The proportional allocation algorithm allocates processors proportional to the computation amount of each task. This algorithm tries to minimize the completion time of all tasks by balancing the execution time of each task. However, it assumes that the execution time of a task decreases if more tasks are allocated to it and that the number of allocable processors are continuous numbers, both of which are not true for the MCSP problem.

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. By proportional allocation, 10 processors are allocated for each matrix product. Since the number of allocable processors for each matrix product should be numbers in $D(16)$ and $D(12)$, 8 and 6 processors are allocated for each product respectively. The completion time of two matrix products becomes 8 unit time because $\max(2 \times 3 \times 8/8, 4 \times 4 \times 3/6) = 8$. However, if we allocate 8 processors to the first product and 12 processors to the second product, the completion time becomes only $\max(6, 4) = 6$ unit 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 which requires longer execution time.

The completion time of two matrix products by proportional allocation is shown in Fig. 1. Let $P1$ be the number of processors allocated to the first product. As $P1$ increases, the execution time $T1$ of the first product decreases, and that of second product, $T2$, increases. Proportional allocation allocates 10 processors to each product, but there is 6 unused processors. We can reduce the completion time using the unused processors. After proportional allocation, we can find the number of unused processors and the product which has the longer execution time. In the example, the second matrix product takes longer. Thus, the unused 6 processors are allocated to the second product so that the

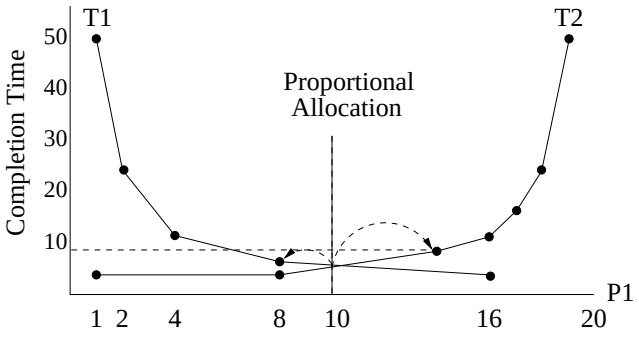


Figure 1: Proportional allocation for two matrix products.

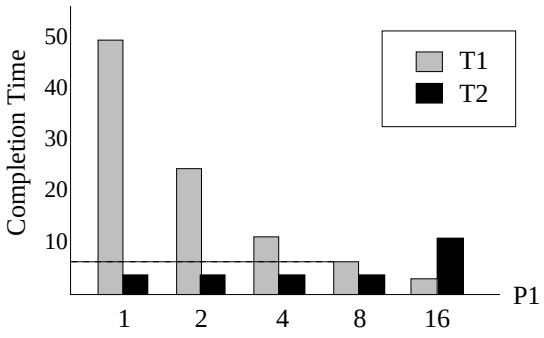


Figure 2: Optimal allocation for two matrix products.

second product runs on 12 processors. Then the completion time of the second product becomes 4. Hence, the overall completion time becomes 6. This allocation is optimal, as shown in Fig. 2. The following algorithm describes the processor allocation algorithm for two independent matrix products.

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. Set $P_{prop} = \frac{m_x m_{x+1} m_{x+2}}{m_x m_{x+1} m_{x+2} + m_y m_{y+1} m_{y+2}} P$.
2. Find $d_{x,i}$ in $D(m_x m_{x+2})$ which satisfies $d_{x,i} \leq P_{prop}$.
3. Find $d_{y,j}$ in $D(m_y m_{y+2})$ which satisfies $d_{y,j} \leq P - P_{prop}$.
4. If $\Phi(m_x, m_{x+1}, m_{x+2}, d_{x,i}) < \Phi(m_y, m_{y+1}, m_{y+2}, d_{y,j})$, then $P_x = d_{x,i}, P_y = P - d_{x,i}$. Otherwise, $P_x = P - d_{y,j}, P_y = d_{y,j}$.

Even if we use a naive search algorithm for finding divisors, Step 2 and Step 3 take $O(P)$ time. Therefore, since the remaining steps require constant time, the time complexity of discrete processor allocation (DPA) is $O(P)$. Also, DPA guarantees the minimum completion time for two matrix products, as shown formally by the following lemma and theorem.

Lemma 1: The completion time of two matrix products when processors are allocated by the DPA algorithm is shorter than or equal to that by the proportional allocation.

Proof: The proportional allocation algorithm allocates P_{prop} processors to the product X, and $P - P_{prop}$ processors to the product Y. When $d_{x,i}$ is the largest divisor in $D(m_x m_{x+2})$ which is less than P_{prop}, the product X utilizes $d_{x,i}$ processors. And when $d_{y,j}$ is the largest divisor in $D(m_y m_{y+2})$ which is less than $P - P_{prop}$, the product Y utilizes $d_{y,j}$. Therefore, the completion time by proportional allocation is bounded by $\max(\Phi(m_x, m_{x+1}, m_{x+2}, d_{x,i}), \Phi(m_y, m_{y+1}, m_{y+2}, d_{y,j}))$.

In the case of $\Phi(m_x, m_{x+1}, m_{x+2}, d_{x,i}) < \Phi(m_y, m_{y+1}, m_{y+2}, d_{y,j})$, the DPA allocates $d_{x,i}$ and $P - d_{x,i}$ respectively. Let $d'_{y,j}$ be the largest divisor in $D(m_y m_{y+2})$ which is less than $P - d_{x,i}$. Since $P - P_{prop} \leq P - d_{x,i}$ and $d'_{y,j} \geq d_{y,j}$, the completion time by the DPA, i.e., $\max(\Phi(m_x, m_{x+1}, m_{x+2}, d_{x,i}),\ \Phi(m_y, m_{y+1}, m_{y+2}, d'_{y,j}))$, is shorter than or equal to the completion time by the proportional allocation, i.e., $\max(\Phi(m_x, m_{x+1}, m_{x+2}, d_{x,i}),$ $\Phi(m_y, m_{y+1}, m_{y+2}, d_{y,j}))$.

Also in the case of $\Phi(m_x, m_{x+1}, m_{x+2}, d_{x,i}) \geq \Phi(m_y, m_{y+1}, m_{y+2}, d_{y,j})$, the DPA allocates $P - d_{y,j}$ and $d_{y,j}$ respectively. In the same manner, the completion time by the DPA is less than or equal to the completion time by proportional allocation. Thus the lemma is satisfied. □

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

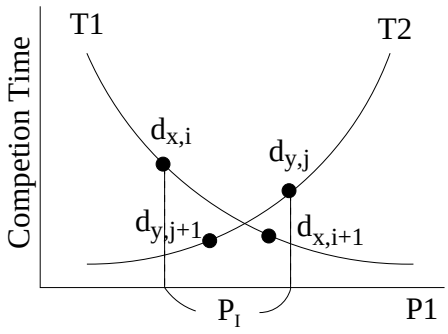
Proof: Given two matrix products $X = (m_x, m_{x+1}, m_{x+2})$ and $Y = (m_y, m_{y+1}, m_{y+2})$ on P processors, let P_{prop} be the number of processors allocated to the product X by proportional allocation.

i) $P_{prop} \geq m_x m_{x+2}$ or $P - P_{prop} \geq m_y m_{y+2}$

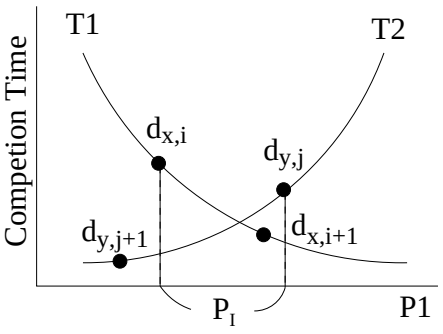
In the case of $P_{prop} \geq m_x m_{x+2}$, the completion time of two matrix products is bounded by the execution time of the matrix product X. Since DPA allocates $m_x m_{x+2}$ processors to the product X, the completion time becomes m_{x+1} which is the minimum execution time of the product X. Therefore, DPA guarantees the minimum completion time of two matrix products. In the case of $P - P_{prop} \geq m_y m_{y+2}$, the completion time is bounded by the execution time of the product Y, and again the DPA guarantees the minimum completion time.

ii) $P_{prop} < m_x m_{x+2}$ and $P_{prop} < m_y m_{y+2}$

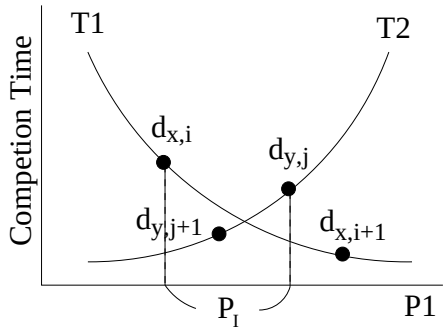
Let $d_{x,i}$ be the largest divisor in $D(m_x m_{x+2})$ where $d_{x,i} \leq P_{prop}$, and $d_{y,j}$ the largest divisor in $D(m_y m_{y+2})$ where $d_{y,j} \leq P - P_{prop}$. Then the number of unused processors, denoted as P_I, is $P - d_{x,i} - d_{y,j}$. Let $d_{x,i+1}$ denote the next divisor of $d_{x,i}$ in $D(m_x m_{x+2})$ such



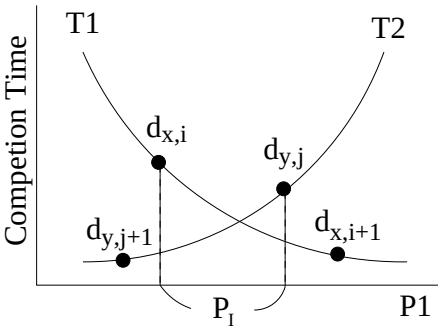
a) The case of $d_{x,i+1} - d_{x,i} \leq P_I$ and $d_{y,j+1} - d_{y,j} \leq P_I$.



b) The case of $d_{x,i+1} - d_{x,i} \leq P_I$ and $d_{y,j+1} - d_{y,j} > P_I$.



c) The case of $d_{x,i+1} - d_{x,i} > P_I$ and $d_{y,j+1} - d_{y,j} \leq P_I$.



d) The case of $d_{x,i+1} - d_{x,i} > P_I$ and $d_{y,j+1} - d_{y,j} > P_I$.

Figure 3: Four cases in the relation of $d_{x,i}, d_{x,i+1}, d_{y,j}, d_{y,j+1}$, and P_I.

that $d_{x,i} \leq P_{prop} < d_{x,i+1}$, and $d_{y,j+1}$ the next divisor of $d_{y,j}$ in $D(m_y m_{y+2})$. There are four cases in the relation of $d_{x,i}$, $d_{x,i+1}$, $d_{y,j}$, $d_{y,j+1}$, P_I as shown in Fig. 3. They are a) $d_{x,i+1} - d_{x,i} \leq P_I$ and $d_{y,j+1} - d_{y,j} \leq P_I$, b) $d_{x,i+1} - d_{x,i} \leq P_I$ and $d_{y,j+1} - d_{y,j} > P_I$, c) $d_{x,i+1} - d_{x,i} > P_I$ and $d_{y,j+1} - d_{y,j} \leq P_I$, and d) $d_{x,i+1} - d_{x,i} > P_I$ and $d_{y,j+1} - d_{y,j} > P_I$.

Without loss of generality, let us assume that $\Phi(m_x, m_{x+1}, m_{x+2}, d_{x,i}) \geq \Phi(m_y, m_{y+1}, m_{y+2}, d_{y,j})$. In cases a) and b) of Fig. 3, the minimum completion time is $\Phi(m_y, m_{y+1}, m_{y+2}, d_{y,j})$ by allocating $d_{x,i+1}$ for X and $d_{y,j}$ for Y. In these cases, DPA allocates $P - d_{y,j}$ for X and $d_{y,j}$ for Y. Since $P - d_{y,j} \geq d_{x,i+1}$ and $d_{x,i+1}$ is the largest divisor in $D(m_x m_{x+2})$ which is less than $P - d_{y,j}$, the DPA guarantees the minimum completion time. In cases of c) and d) in Fig. 3, since we cannot allocate $d_{x,i+1}$ processors to X, the minimum completion time is bounded by $\Phi(m_x, m_{x+1}, m_{x+2}, d_{x,i})$. Since $P - d_{y,j}$ and $d_{y,j}$ are allocated by DPA and $d_{x,i}$ is the largest divisor in $D(m_x m_{x+2})$ which satisfies $d_{x,i} \leq P - d_{y,j}$, the completion time by DPA is the same as the minimum. Therefore, DPA guarantees the minimum completion time of two matrix products in all cases. □

DPA can be easily extended for k independent matrix products. After allocating by proportional allocation, the unused processors are allocated to the product with the maximum execution time to reduce the completion time of all products. However, if there are still unused processors, this reallocation is continued until the maximum execution time cannot be reduced. We present the processor allocation algorithm DPA-k for k independent matrix products as follows.

Discrete Processor Allocation for k Matrix Products (DPA-k)

Input: k matrix products $X_i = (m_{i,1}, m_{i,2}, m_{i,3})$ for $1 \leq i \leq k$ are given on P processors ($k \leq P$).

Output: The number of allocated processors P_i for each matrix product X_i which satisfies $\sum_{i=1}^{k} P_i \leq P$.

1. For $i = 1$ to k do
   (a) $P_{prop,i} = \frac{m_{i,1}m_{i,2}m_{i,3}}{\sum_{j=1}^{k} m_{j,1}m_{j,2}m_{j,3}} P$.
   (b) Find the maximum $d_{i,j}$ in $D(m_{i,1}m_{i,3})$ which satisfies $d_{i,j} \leq P_{prop,i}$.
   (c) Let $P_i = d_{i,j}$.
2. While $(P - \sum_{i=1}^{k} P_i > 0)$ do
   (a) Find the product X_i with the maximum $\Phi(m_{i,1}, m_{i,2}, m_{i,3}, P_i)$.
   (b) If $(P_i < m_{i,1}m_{i,3})$ then
      i. Find the minimum $d_{i,j}$ in $D(m_{i,1}m_{i,3})$ which satisfies $d_{i,j} > P_i$.
      ii. If $(P - \sum_{i=1}^{k} P_i - (d_{i,j} - P_i) > 0)$ then $P_i = d_{i,j}$. Otherwise, stop the algorithm.

   else stop the algorithm.

Corollary 1: DPA-k guarantees the minimum completion time of k independent matrix products.

4 Matrix Chain Scheduling Algorithm

The proposed scheduling algorithm consists of three stages. First, the algorithm finds the optimal product sequence for MCOP. Next is the top-down processor assignment stage. In this stage, processors are partitioned and assigned to each subtree proportionally according to their computation amount to balance the evaluation time of both 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 number of operations.

4.1 Optimal Product Sequence by 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 deciding factor of the evaluation time, but affects it greatly nonetheless. Hence, our scheduling algorithm begins with the optimal product sequence found for MCOP. There were many works reported for finding the optimal product sequence which guarantee 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 [7, 8]. Many parallel algorithms have been studied [13, 14] which run in polylog time. Therefore, 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 computation amount found by MCOP is stored in two tables, $S[n][n]$ and $W[n][n]$, respectively. $W[i][j]$ has the minimum number of operations for evaluating $L_{i,j}$ and $S[i][j]$ the matrix index for partitioning the matrix chain $(M_i \times \cdots \times M_j)$.

4.2 Top-Down Processor Assignment

In the top-down processor assignment stage, a number of processors proportional to the computation amount of a subtree is assigned to minimize the completion time of two partial matrix chains. If $p_{i,j}$ processors were 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 the 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 the subtree $L_{S[i][j]+1,j}$.

For example, given a chain of 8 matrices with dimensions $\{3, 8, 9, 5, 8, 3, 3, 3, 4\}$ on a 64

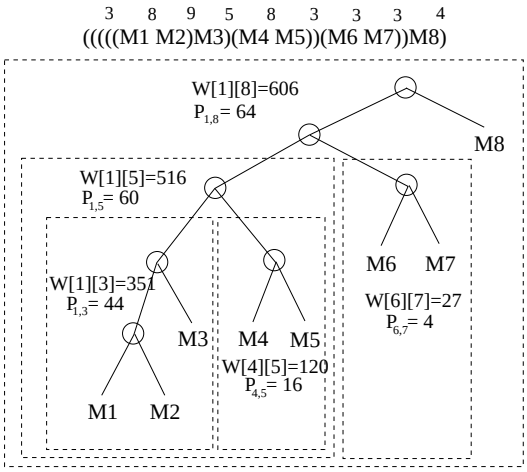


Figure 4: Top-down processor assignment.

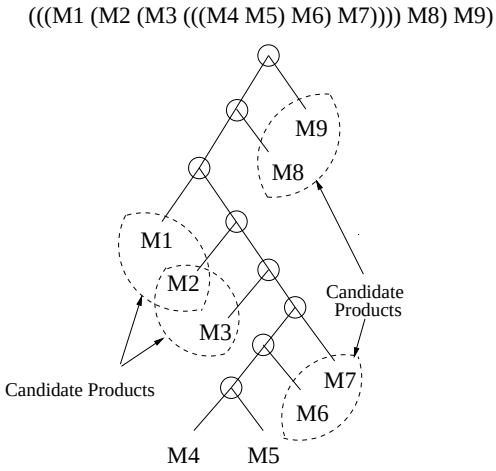


Figure 5: Candidate products on a sequence tree.

processor system, processors are assigned as in Fig. 4.

4.3 Bottom-Up Concurrent Execution

After assigning processors to each subtree, the matrix products are executed concurrently and independently from the leaf products. Thus, the performance is improved by more concurrent execution, which results in improved processor efficiency. 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.

When there are idle processors in the execution of $L_{i,j}$, ancestors of the leaf node of

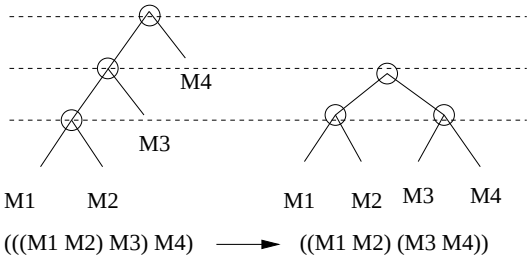


Figure 6: A sequence modification.

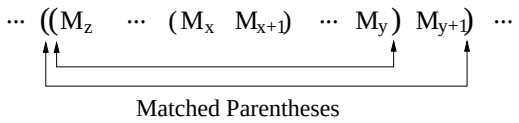


Figure 7: A snapshot of product sequence.

$L_{i,j}$ are traced in order to find a candidate for concurrent execution. This upward trace continues until a suitable candidate or a sibling which is not a leaf node is found. For example, let us consider the executing sequence tree $L_{1,9}$ shown in Fig. 5. The figure 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 $(M_1M_2), (M_2M_3), (M_6M_7), (M_8M_9)$. There are other types of candidate products which are not considered in this paper like (M_7M_8). Since such cases result in more modifications to the optimal sequence with no obvious benefit over other candidates, we do not consider these kinds of products.

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

Consider a matrix chain with four matrices which needs three matrix products to get the final result. Assume that the optimal product sequence of this matrix chain for MCOP is $(((M_1M_2)M_3)M_4)$, as shown in Fig. 6. When the sequence is modified to $((M_1M_2)(M_3M_4))$, the number of computations required 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 Fig. 7, the amount of increased computation for multiplying $M_y \times M_{y+1}$ concurrently with $M_x \times M_{x+1}$ is as follows:

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

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 $y < z$, the amount of increased computation is as follows:

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

Finding m_z (or m_{z+1}), which is very important for the analysis of ΔC, can be done by traversing the 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.

Lemma 2: In the case that $p_{i,j}$ processors are allocated to the matrix product $(M_x M_{x+1})$ but all $p_{i,j}$ processors cannot be utilized by the matrix product (i.e., $m_x m_{x+2} < p_{i,j}$), we try to modify the product sequence L. If the candidate product $(M_y M_{y+1})$ is found, and the DPA allocates p_x and p_y processors to two matrix products $(M_x M_{x+1})$ and $(M_y M_{y+1})$ respectively, then the evaluation time is reduced by modifying the sequence tree L when the candidate product $(M_y M_{y+1})$ satisfies $\Delta C < \min(\Phi(m_x, m_{x+1}, m_{x+2}, m_x m_{x+2}) \times (p_x + p_y - m_x m_{x+2}), m_y m_{y+1} m_{y+2})$.

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., $p_x + p_y - m_x m_{x+2}$) should be more 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.

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

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

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

Thus, the lemma is satisfied. □

If a candidate product $(M_y M_{y+1})$ satisfies Lemma 2, then it would be better to change the 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 by a parallel algorithm for MCOP.
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, else call this algorithm recursively for both $i = i, j = S[i][j]$ and $i = S[i][j]+1, j = j$.

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. 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. Check whether the candidate satisfies Lemma 2. If not, go to 2.

4. Modify the sequence tree such that the candidate can run 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 $\min(p_{i,j},\ m_k m_{k+1} m_{k+2})$ processors. Set the parent of the leaf product as a new leaf product.

The scheduling algorithm starts from the MCOP sequence, and tries to modify the sequence by increasing the concurrency level. The evaluation time of a matrix chain product is affected by the amount of computations and the concurrency level. The amount of computations is minimized at the MCOP sequence, and the concurrency level is maximized at the sequence of the complete binary tree. The optimal product sequence with the minimum evaluation time may be formed in the middle of the MCOP sequence and the complete binary tree. This also means that when the MCOP sequence is a complete binary tree, the MCOP sequence is the optimal product sequence. The proposed scheduling algorithm moves from the MCOP sequence to the 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 satisfying candidate for the purpose of minimizing the schedule 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. 8. Then we do not need to traverse down the children of a candidate product to find M_z since M_z is passed from the traced child. This allows Step 3 of Stage-3 to be done in constant time. The number of products being traced to check concurrent execution is $(n-3)$ at the most. The 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 $O(P)$, the time complexity for Step 4 of Stage-3 is $O((n-4)P)$. Therefore, the time complexity of the proposed algorithm is $O(n^2+nP)$.

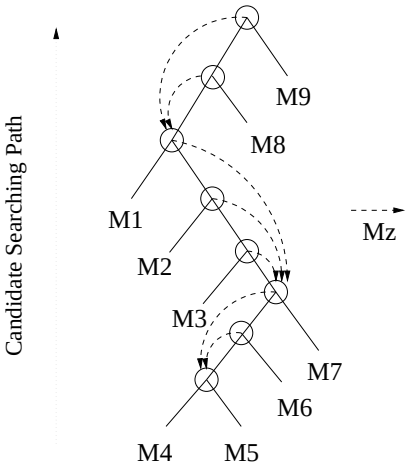


Figure 8: Candidate searching and passing the information of the skewed point M_z.

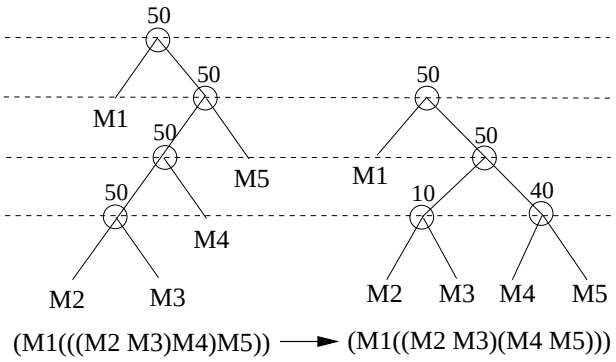


Figure 9: The sequence trees for MCOP sequence and the proposed scheduling algorithm.

4.6 Example

The following simple example illustrates the proposed scheduling algorithm. We also compare the expected evaluation time of the product sequence by the proposed algorithm with that of the product sequence for MCOP.

In a system with 50 processors, let us consider a 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 minimum operations for MCOP as $(M_1(((M_2M_3)M_4)M_5))$. The MCOP sequence tree is represented as the left tree of Fig. 9. 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 as 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. 9.

Let us compare the evaluation time of the MCOP sequence with that of the product sequence found by the proposed scheduling algorithm. When we evaluate the matrix chain by the MCOP sequence, it takes $(2\times7\times5)/\min(50, 2\times5)+(2\times5\times7)/\min(50, 2\times7)+(2\times7\times 8)/\min(50, 2\times8)+(6\times2\times8)/\min(50, 6\times8) = 21$ units of time. The evaluation time of the product sequence by the proposed scheduling algorithm is $\max(2\times7\times5/\min(10, 2\times5), 5\times 7\times8/\min(40, 5\times8))+2\times5\times8/\min(50, 2\times8)+6\times2\times8/\min(50, 6\times8) = 7+5+2 = 14$ units of time. The product sequence by the proposed scheduling algorithm requires 526 operations, which is 178 operations more than the MCOP sequence with the minimum number of operations (348). However, the proposed algorithm requires less time than the MCOP sequence with the minimum number of computations. This is due to the concurrent execution of multiple matrix products which increases system efficiency and reduces the total evaluation time.

5 Performance Analysis

In this section, we compare the performance of the proposed scheduling algorithm with various evaluation methods.

- **Linear:** evaluate from the first product (M_1M_2) to the last one sequentially.
- **MCOP-Seq:** evaluate by the MCOP sequence sequentially.
- **MCOP-Con:** evaluate by the MCOP sequence, but execute independent matrix products concurrently by allocating the maximum number of processors.
- **MCSP-BT:** evaluate by the MCOP sequence with concurrent execution, but when there are idle processors during execution, try to modify the sequence by checking Lemma 2.
- **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 processor is a SPARC processor with 16MB. The AP1000 system has three independent networks: B-net for broadcasting, T-net for torus interconnection, and S-net for synchronization. The processors are connected as a two dimensional torus and the T-net link speed between processors is 25Mbytes/sec/port. The host computer and the processors are connected by the broadcasting network (B-net) with 50Mbyte/sec and by the S-net for synchronization.

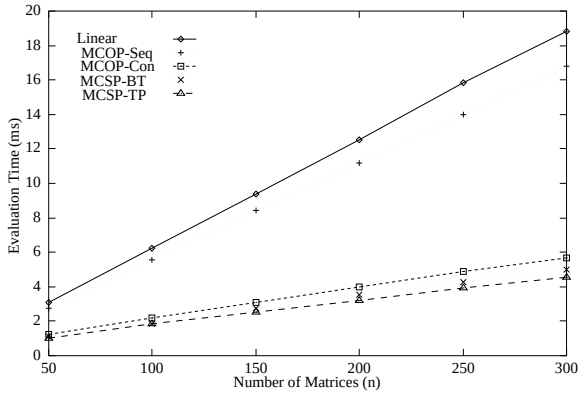


Figure 10: Evaluation time comparison by varying the number of matrices when the matrix sizes distributed from $1 \sim 10$.

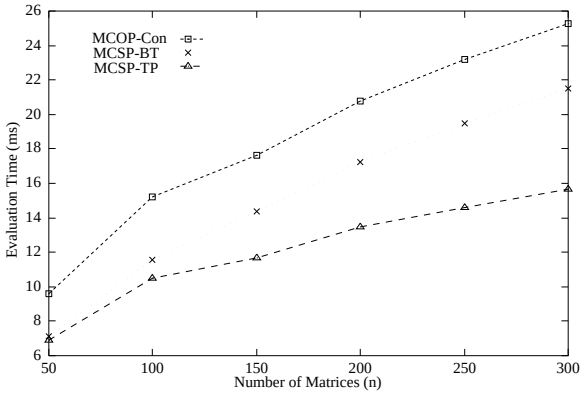


Figure 11: Evaluation time comparison by varying the number of matrices when the matrix sizes distributed from $1 \sim 50$.

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 the system characteristics such as the communication link speed and 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 by allowing several matrices to be loaded together.[3] The results shown in this section is the average of 100 experiments.

Fig. 10 and Fig. 11 show the evaluation time as a function of the number of matrices being multiplied. In Fig. 10, a chain of matrix products is generated with size varying randomly from 1 to 10, and executed on a system with 512 processors. The upper two lines represent the evaluation times of the sequential evaluation by Linear and MCOP-Seq, and the lower three lines represent the evaluation times of the scheduling sequence found by 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 only reducing the amount of computation does not greatly decrease the evaluation time. But when we allow concurrent execution, we can get further performance improvement. Therefore, we confirm that the evaluation time of a chain of matrix products is greatly affected by task scheduling. In Fig. 11, we experimented with larger matrices whose sizes varied from 1 to 50. 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 larger performance gain.

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

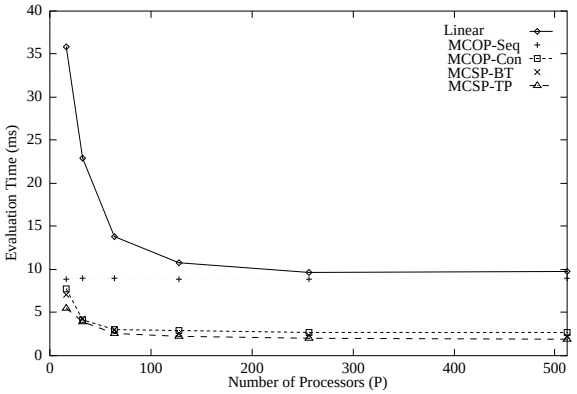


Figure 12: Evaluation time comparison for different number of processors when the matrix sizes distributed from $1 \sim 20$.

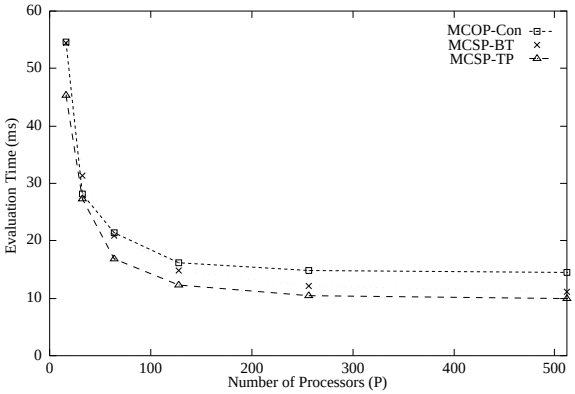


Figure 13: Evaluation time comparison for different number of processors when the matrix sizes distributed from $1 \sim 50$.

In Fig. 12 and Fig. 13, the evaluation time is compared for different numbers of processors in a system. In Fig. 12, a matrix chain consists of 100 matrices ($n = 100$) whose sizes are varied from 1 to 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 amount has a limitation in reducing the evaluation time. In Fig. 13, we measured the evaluation times of three concurrent execution methods for the matrices whose sizes are varied from 1 to 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 the sequence modification to increase the concurrency level 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 the processor scheduling policy is much more important in improving the performance of evaluating a chain of matrix products than computation reduction.

The result of experiments with varying the distribution of matrix sizes is shown in Fig. 14 and Fig. 15. The evaluation times are measured on 512 processors for a chain of 100 or 200 matrices respectively. When the maximum matrix size is set at a value such as MAX, the size of matrices is distributed randomly from 1 to MAX. As shown in the figures, when the variance in matrix size gets larger, MCOP-Seq has more of a performance gain than Linear. This is caused by the fact that the computation amount is reduced greatly by

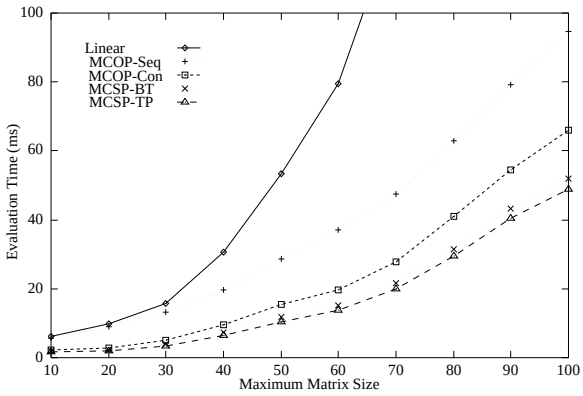


Figure 14: Evaluation time comparison by varying the matrix size distribution for a chain with 100 matrices.

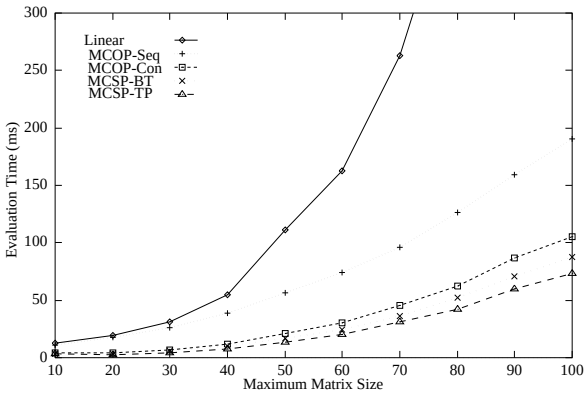


Figure 15: Evaluation time comparison by varying the matrix size distribution for a chain with 200 matrices.

Table 1: Comparison with respect to computation amounts, system utilization, and evaluation time when $P = 512, n = 100$, and the matrix size is varied from $1 \sim 10$.

Scheduling Algorithm	Computation Amount	System Utilization	Evaluation Time
Linear	6136	3.20	6.27
MCOP-Seq	1430	0.62	5.59
MCOP-Con	1430	1.63	2.15
MCSP-BT	1618	2.32	1.89
MCSP-TP	2253	3.07	1.81

the MCOP sequence, and 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 outperforms all other methods. These experiments imply that the proposed MCSP-TP 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 computation amount, system utilization, and evaluation time for each evaluation method. In Table 1, we evaluated a chain of 100 matrices whose sizes varied from 1 to 10. Even though the computation amount of MCOP-Seq is just a quarter of that of Linear, the evaluation time is similar (such as 6.27 and 5.59). We observe that the system utilization of MCOP-Seq (0.62) is significantly lower than that of Linear (3.20). Evaluation by MCSP-BT and MCSP-TP requires more computation than that by MCOP-Con; the evaluation time decreases due to the efficient use of processors. In

Table 2: Comparison with respect to computation amounts, system utilization, and evaluation time when $P = 512, n = 100$, and the matrix size is varied from $1 \sim 50$.

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, we measured the execution time for a chain with 100 matrices whose sizes varied from 1 to 50. We observed behavior similar to that in Table 1. But note that MCSP-BT utilizes even more processors than MCSP-TP, and the evaluation time of MCSP-BT is larger than that of MCSP-TP. This result implies that the proposed MCSP-TP algorithm uses processors more efficiently and effectively than MCSP-BT.

From the above experiments, we get the following results.

- When evaluating a chain of matrices on a parallel system, reducing the amount of computations does not greatly decrease the evaluation time.
- Concurrent execution of independent multiple matrix products compensates the performance loss by parallel processing and increases the system efficiency so that the 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, MCSP-BT.
- Even when the size of matrices is quite large so that there are no idle processors during their evaluation, the proposed scheduling algorithm MCSP-TP is still effective due to execution using the maximum level of concurrency.
- When evaluating a chain with small matrices on many processors, sequence modification to increase system efficiency greatly reduces the evaluation time.
- In matrix chain products, efficient scheduling is better than increasing the number of processors.

6 Summary and Conclusion

In this paper, we introduced the matrix chain scheduling problem (MCSP) and proposed a heuristic scheduling algorithm for MCSP. The proposed algorithm schedules processors to matrix products to increase concurrency at the expense of a slight increase in the required amount of computation 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 the processor scheduling is much more important than reducing the amount of computation for evaluating a matrix chain product in parallel systems. In a system with a large number of processors or a matrix product chain with many matrices, evaluation by the proposed method greatly outperforms the sequential evaluation method using the optimal product sequence found for MCOP. The main contribution of this work is the introduction of MCSP and the proposal of a scheduling algorithm which results in a significant performance improvement when evaluating matrix chain products in parallel systems. We are currently working on applying this algorithm to evaluate a chain of square matrices with the form of sparse matrices or band matrices. Also, we plan to study generalizing MCSP to scalable task scheduling on parallel systems.

Acknowledgment

We would like to thank Prof. Chul E. Kim for many fruitful suggestions on the complexity analysis. Also we give special thanks to Mr. Min-Ho Kyung for very helpful discussions on this work. Also we give special thanks to FUJITSU Laboratories Ltd. for allowing us to use their facilities, and especially to Takao Saito for arranging our urgent requests for use.

References

[1] 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.

[2] 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.

[3] 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).

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

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

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

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

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

[9] 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.

[10] 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.

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

[12] 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.

[13] P. G. Bradford, G. J. Rawlins, and G. E. Shannon, "Efficient matrix chain ordering in polylog time," in *Proc. of Int'l Parallel Processing Symp.*, pp. 234–241, 1994.

[14] 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.

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

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

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

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

[19] 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.

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

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

[22] J. Du and J. Y.-T. Leung, "Complexity of scheduling parallel task systems," *SIAM J. on Discrete Math.*, vol. 2, no. 4, pp. 473–487, 1989.

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

[24] V. Kumar and A. Gupta, "Analyzing scalability of parallel algorithms and architectures," *Journal of Parallel and Distributed Computing*, vol. 22, pp. 379–391, 1994.

[25] G. N. S. Prasanna and B. R. Musicus, "Generalized multiprocessor scheduling and applications to matrix computations," *IEEE Trans. on Parallel and Distributed Systems*, vol. 7, pp. 650–664, June 1996.

[26] 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.

[27] 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.