Processor Allocation for Matrix Products

Heejo Lee, Jong Kim, and Sungje Hong

Dept. of Computer Science and Engineering
Pohang University of Science and Technology
San 31 Hyoja Dong, Pohang 790-784, KOREA
Phone : +(82)-562-279-2257
Fax : +(82)-562-279-2299
E-mail : {heejo,jkim}@postech.ac.kr

Abstract

In this paper, we present the problem of allocating processors for matrix products. First, we consider how many processors should be allocated for computing one matrix product on a parallel system. Then, we discuss how to allocate processors for a number of independent matrix products on the parallel system. In many cases, it is shown that the performance of parallel algorithms does not improve beyond a certain point even though more processors are allocated for more parallelism. The results from experiments on the Fujitsu AP1000 parallel system for a matrix product show that allocating more processors is not always beneficial for overall system performance. Also, when evaluating a number of independent matrix products, the concurrent execution of multiple matrix products by partitioning the system is better than the independent evaluation of matrix products sequentially by parallelizing each matrix product. Finally, we conclude that such kind of result can be applicable to many processor allocation problems on a parallel system such as the processor allocation problem for evaluating a chain of matrix products.

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

1 Introduction

Matrix operations are commonly used in many scientific computations such as linear algebra and numerical analysis. Matrix products are most commonly encountered and computation intensive part of the scientific computations. To reduce the computation time of matrix products, many parallel algorithms have been studied in various parallel architectures [1].

Parallel algorithms for matrix product has the scalability characteristic [2]. A matrix product for multiplying two $n \times n$ matrices can utilize from one processor to n^3 processors. In case of n^3 processors, the execution time can be $O(\log(n))$ time. However, we can expect low processor utilization and increased communication overhead for calculating one element of the result matrix by n processors. Then, how many processors should be allocated for multiplying two matrices to have a better efficiency?

One general problem in parallel processing is that how many processors should be allocated for a given task to have a better response time or better system utilization. In many cases, it is shown that the performance of parallel algorithms does not improve beyond a certain point even though more processors are allocated for more parallelism. This is mainly due to the increased interprocessor communication between more parallelized subtasks. In some algorithms, the execution time decreases by allocating more processors, but some processors are severely under-utilized. Matrix product shows this kind of behavior. Therefore, only the num-

ber of processors that guarantees the best efficiency should be allocated.

Until now, even though many parallel algorithms for a matrix product have been studied on a parallel system under various conditions, there are few works for finding the number of processors which guarantees the best efficiency of the system, and consequently the best overall system performance.

One related problem we also have to consider is how to schedule and allocate processors for each product when evaluating a number of independent matrix products. One of simple approaches is just parallelizing each matrix product one after the other. However, we can imagine that the approach of parallelizing each product and executing one after the other using the whole system results in inefficient use of processors and poor performance. In this paper, we show the effect of concurrent execution by partitioning a parallel system using experiments, and study the processor allocation method for independent matrix products.

This paper is organized as follows. Section 2 presents the processor allocation problem for one matrix product. In Section 3, we consider the case of evaluating a number of independent matrix products. In Section 4, we discuss how to apply the results to other processor allocation problems such as for evaluating a chain of matrix products, which is known as matrix chain ordering problem (MCOP). Finally, in Section 5, we summarize and conclude the paper.

2 Processor Allocation for One Matrix Product

In this section, we consider the problem of allocating the proper number of processors for a matrix product. Among many parallel algorithms for multiplying two matrices, the algorithms for a mesh-connected system is simple and efficient since the mesh structure is highly analogous to the matrix so that one can find a quite efficient parallel algorithm easily in the mesh structure. The parallel algorithm for a matrix product used in this section is a commonly used algorithm for a parallel system connected by mesh.

Consider a multiplication $C = A \times B$, where A and B are an $n \times n$ matrix, on a mesh-connected multiprocessor system with P processors. If $P \leq n^2$, then one way of parallelizing the computation is dividing the computation needed for C matrix into each processor as much as $\frac{n}{\sqrt{P}} \times \frac{n}{\sqrt{P}}$. In this case, each processor needs an $\frac{n}{\sqrt{P}} \times n$ sub-matrix from A, and an $n \times \frac{n}{\sqrt{P}}$ sub-matrix from B. And, $\frac{n^2}{\sqrt{P}}$ multiplications and $\frac{(n-1)n}{\sqrt{P}}$ additions are required to get a part of C matrix. The number of operations for each processor is proportional to $\Theta(n^2/\sqrt{P})$. For example, when $P = n^2$, at least $\Theta(n)$ operations are assigned to each processor.

A multiplication of two $n \times n$ matrices requires at most n^3 processors. In the best case, it takes $O(\log(n))$ time such that n processors are used to get an element of C matrix. Each of n processors multiplies in one step, and n elements are summed within $O(\log(n))$ steps if the architecture supports it. However, in the case of using n^3 processors, we can expect low utilization of processors. Each processor spends $\log(n)$ time on communication while summing n data. Moreover, these processors are not active all the time. Then, how many processors should be allocated for multiplying two matrices to have a better efficiency?

We measured the computation time of a parallel matrix multiplication on the Fujitsu AP1000 parallel system with 512 processors. The AP1000 system supports three independent network: B-net for broadcasting, T-net for Torus interconnection, S-net for synchronization. The processors are connected by two dimensional Torus (T-net) with 25Mbyte/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.

For the matrix multiplication $C = A \times B$ on a $P_1 \times P_2$ mesh, where A is an $m_i \times m_j$ matrix and B is an $m_j \times m_k$ matrix, each processor requires a part of A and B matrices. If $m_i \times m_k \geq P_1 \times P_2$, then $\frac{m_i}{P_1} \times m_j$ sub-matrix of A and $m_j \times \frac{m_k}{P2}$ sub-matrix of B are needed for each processor to compute an $\frac{m_i}{P_1} \times \frac{m_k}{P2}$ sub-matrix of C as shown in Fig. 1. In case of $m_i \times m_k < P_1 \times P_2 \leq m_i \times m_j \times m_k$, $\frac{P1 \times P2}{m_i \times m_k}$ processors are used to compute one element of C matrix.

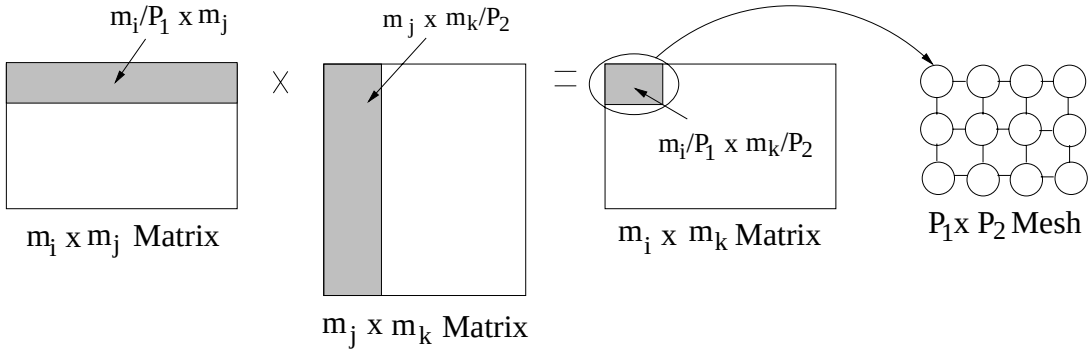

Figure 1: Multiplication of an $m_i \times m_j$ matrix and an $m_j \times m_k$ matrix on a $P_1 \times P_2$ mesh.

The process of matrix multiplication can be divided into three phases. The first phase is distributing parts of matrices A and B to each corresponding processor. The second phase is concurrent multiplications using parts of two matrices on each processor. The third phase is collecting partial results into one processor which needs the result matrix. In our experiments, the result matrix is returned to the host computer. We measured the execution time of matrix multiplication in various ways on the Fujitsu AP1000 parallel system.

In the first experiment, the total running time, i.e., the sum of three phases, is measured, which includes the time for distributing sub-matrices of A and B to corresponding processors, computing sub-matrix multiplication, and collecting results into the host computer. Fig. 2 shows the running time of a matrix multiplication for various matrices of different size on meshes of different size. For example, 128 processors in the figure represents 8×16 mesh. The (m_i, m_j, m_k) notation represents that A is an $m_i \times m_j$ matrix and B is an $m_j \times m_k$ matrix. From the result, we can see that the running time decreases as the number of processors increases until a certain point. Let us call this point as *summit point*. However, when the number of allocated processors is larger than that point, e.g., 64 processors for $(128, 128, 128)$ in Fig. 2, the execution time increases as more processors are allocated. This means that the speedup obtained using parallel processing does not exceed the linear speedup and for some cases, the more execution time is needed as the number of allocated processors increases.

In Fig. 3, the execution time of matrix products with small matrices is shown to measure

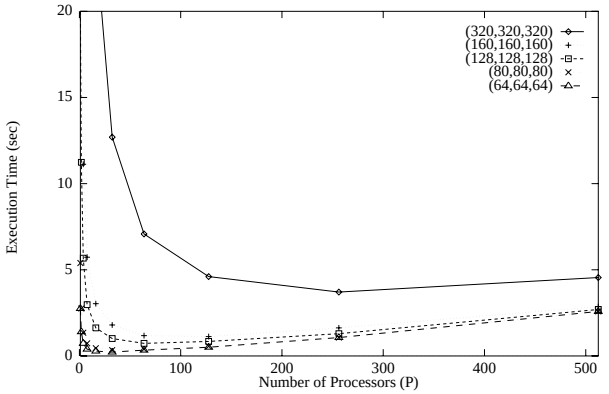

Figure 2: Execution time with varying the number of processors.

the cases of allocating more than $m_i \times m_k$ processors when multiplying an $m_i \times m_j$ matrix and an $m_j \times m_k$ matrix. From the result of this experiment, we can see that allocating more than $m_i \times m_k$ processors also does not reduce the execution time. The execution of $(8, 8, 8)$ on 512 processors takes more time than the execution on 64 processors.

In the next experiment, we have measured only the computation time, which excludes the communication time for distributing submatrices of A and B, and collecting the results. I.e., each processor already contains its data of A and B and does not need to send its result to the collecting processor. The matrix computation in a multiprocessor system with shared memory will show this kind of execution time. The computation time of a product is the maximum computation time among processors. As shown in Fig. 4, the computation time decreases by increasing the number of allocated processors. However, in Fig. 5, when allocating more than

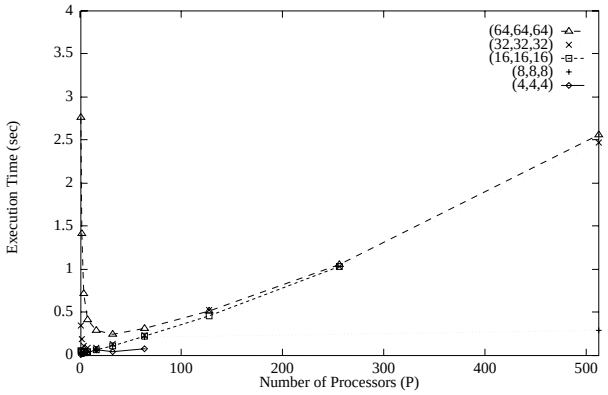

Figure 3: Execution time for small size matrices.

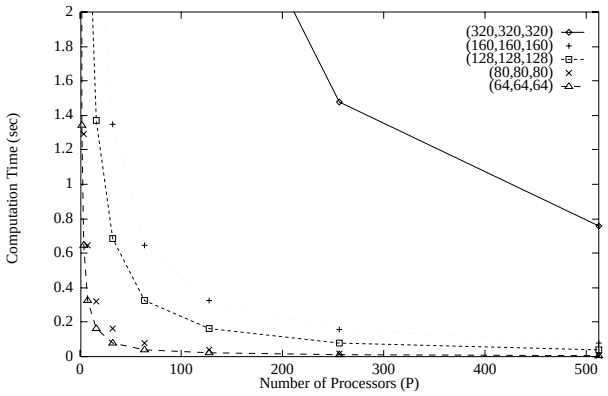

Figure 4: Computation time only for large size matrices.

16 processors for $(4,4,4)$, the computation takes more time than when 16 processors are allocated. That means the summit point of $(4,4,4)$ is 16-processor. In this experiment, the summit point of (m_i, m_j, m_k) is $m_i \times m_k$ processors.

From the above experiments for multiplying two matrices, an $m_i \times m_j$ matrix and an $m_j \times m_k$ matrix, we can see that there is no benefit when more than $m_i \times m_k$ processors are allocated.[1] When the number of allocated processors becomes smaller and smaller, we have no need to worry about overall performance degradation by the communication overhead and inefficient use of processors.

[1] In general, if more than $m_i \times m_k$ processors are allocated, the number of idle processors increases although we can reduce the execution time.

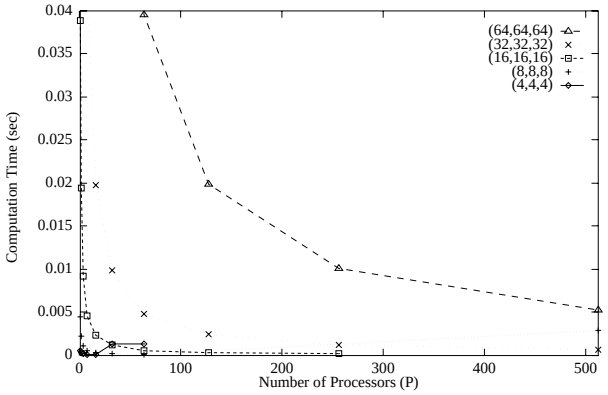

Figure 5: Computation time only for small size matrices.

3 Evaluation of Multiple Independent Matrix Products

Now, let us consider the case when a number of independent matrix products are given for execution. There could be several approaches for evaluating these matrix products. For example, one can consider evaluating each matrix product using the whole system one after the other by parallelizing each product. Also, others can consider the evaluation method by running multiple matrix products concurrently by partitioning the system. It is possible and easily implementable due to the scalability of matrix products [2]. To compare these approaches, let us estimate the required evaluation time.

For a given set of matrix products $C = \{C_1, C_2, .., C_n\}$, the evaluation time by sequential evaluation on a P processor system can be estimated as follows.

$$T_s(C) = \sum_{i=1}^{n} T(C_i, P)$$

$T(C_i, P)$ is the execution time of C_i using P processors. The total evaluation time of this method $T_s(C)$ is the summation of each matrix product execution time for all n products. If we assume all products have the same dimension, i.e., $C_1 = C_2 = .. = C_n$, the evaluation time is $n \cdot T(C_1, P)$.

Now let us assume that the products which can be executable on P processors are also can be executable on P' $(1 \leq P' \leq P)$ processors using the same matrix product algorithm, and each processor has enough memory capacity to

execute it. The total execution time of n products when d matrix products run concurrently by partitioning the system into d groups as like $P = P_1 + P_2 + ... + P_d$, is as follows.

$$T_p = \sum_{i=1}^{n/d} \max_{j=1..d}(T(C_j, P_j))$$

By the speedup limitation of parallelized matrix product, which is also shown using the experimental results in Section 2, the following relation holds for any matrix product C_j and any d $(2 \leq d \leq P)$.

$$T(C_j, P) \geq \frac{1}{d} T(C_j, P/d)$$

When we assume the system is partitioned equally like $P_i = P/d$, the following equation is derived.

$$\begin{aligned} T_p(C) &= \sum_{i=1}^{n/d} \max_{j=1..d}(T(C_j, P_j)) \\ &= \sum_{i=1}^{n/d} \max_{j=1..d}(T(C_j, P/d)) \\ &\leq \sum_{i=1}^{n} T(C_i, P) = T_s(C) \end{aligned}$$

Therefore, the execution time $T_p(C)$ by partitioning the system into d groups is less than the execution time $T_s(C)$.

We have measured the total execution time of given matrix products executed on the Fujitsu AP1000 parallel system with 512 processors. It includes the time to distribute the submatrices and to collect the results to the host computer. Fig. 6 shows the results on various number of partitions for evaluating identical 8 matrix products. For example, 2 in the x-axis represents the evaluation method that two matrix products are executed concurrently on the two-partitioned system and 4 repetitions are required for evaluating 8 matrix products. As the results indicate, the method of running multiple matrix products concurrently enhances the performance than the method of running each matrix product with the whole system. As the concurrency level increases, the total execution time decreases in this case.

Fig. 7 shows the results for 32 identical matrix products. For the $(320, 320, 320)$ product,

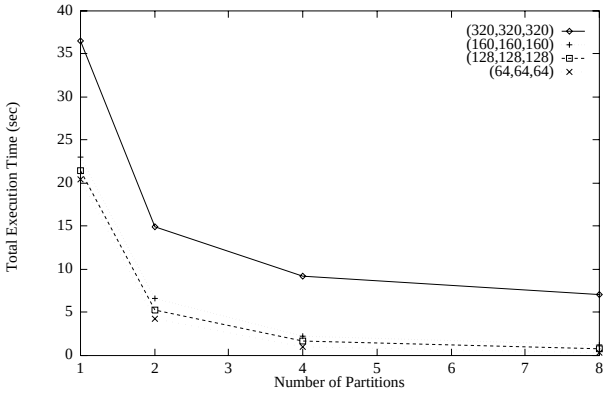

Figure 6: Total execution time of 8 matrix products over various partitioning sizes.

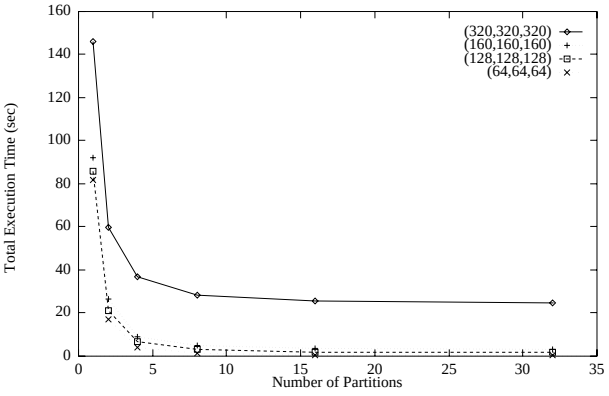

Figure 7: Total execution time of 32 matrix products over various partitioning size.

running on the two-partitioned system (60 seconds) gains almost 2.5 times speedup than running on the whole system (150 seconds). Since the speedup by doubling the allocated processors cannot exceed 2, the execution on the two-partitioned system reinforces the performance more than twice when there is enough parallel tasks to run concurrently. This simple derivation shows the effect. Speedup boundary for parallelizing a product C_j:

$$\frac{1}{2} T(C_j, P) \leq T(C_j, 2P).$$

Reinforced speedup by reducing parallelism and executing two C_j products concurrently:

$$T(C_j, P/2) \leq 2T(C_j, P)$$

However, only increasing the concurrency level by partitioning a system does not guarantee the performance enhancement. In case of

(320, 320, 320) product in Fig. 7, more than 8 partitioning does not gain performance enhancement.

From the above experiments, we can conclude that the overall system performance is enhanced by executing multiple independent matrix products concurrently. The concurrent execution by partitioning a parallel system compensates the loss of the performance by parallelizing a task and increases the overall system performance. Notice that the performance significantly improves when a matrix product is running on less than summit point of processor allocation.

4 Applying to Matrix Chain Products

The performance enhancement by reducing the dependent parallelism and increasing the independent concurrency can be applicable to many processor allocation problems with scalable tasks.

As one example, let us consider the evaluation problem of a matrix chain product, which is commonly used as an example of dynamic programming. 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 a $m_i \times m_{i+1} (m_i \geq 1)$ matrix, the product sequence of matrices greatly affects the total number of operations to evaluate $\mathcal{M}$, even though the final result is the same for all product sequences by the associative law of the 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 to find a product sequence of matrices such that the number of operations is minimized.

There are many works reported for solving MCOP and for reducing the ordering time. First of all, MCOP was reported by Godbole [4] and solved using dynamic programming in $O(n^3)$ time. The optimal sequential algorithm which runs in $O(n \log(n))$ time was given by Hu and Shing [5, 6]. This algorithm solves MCOP by solving the equivalent problem, known as the problem of finding an optimal triangulation of a convex polygon. Many parallel algorithms to reduce the ordering time have been studied using the dynamic programming method [7, 8, 9, 10] or the triangulation of a convex polygon [11, 12]. Bradford [9] proposed a parallel algorithm based on the dynamic programming which runs in $O(\log^2(n))$ time with n processors on the EREW PRAM or $O(\log^2(n) \log\log(n))$ time with $n/\log\log(n)$ processors on the CRCW PRAM. Czumaj 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 [11]. Ramanan improved the algorithm to run in $O(\log^4(n))$ time using n processors [12].

However, the evaluation of a matrix chain product by the MCOP order cannot guarantee the minimal time on a parallel system due to inefficient use of processors. In the single processor system, the evaluation of a chain of matrix products by the optimal product sequence guarantees the minimum evaluation time since the optimal product 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 that the evaluation time in parallel systems is affected by various factors such as dependency among tasks, communication delays, and efficiency of processors as we discussed in Section 2. Hence, the evaluation time of a chain of matrix products in parallel systems can be reduced by executing two or more independent matrix products concurrently even though doing so may increase the number of operations. Therefore, it is more important to find a set of independent matrix products and to increase the set by modifying the product sequence as long as we can reduce the overall evaluation time.

5 Conclusion

The processor allocation problem for matrix products is considered in this paper. From the experiments, allocating more processors does not increase the speedup after a certain point – *summit point*. And the summit point of a product to multiply an $m_i \times m_j$ matrix and an $m_j \times m_k$ matrix is smaller than $m_i \times m_k$ in any kind of

a parallel system. For evaluating a number of independent matrix products, we can get the performance enhancement by executing multiple matrix products concurrently on a partitioned system.

This kind of efficient processor allocation concept can be applicable to many processor allocation problems for parallel tasks with scalability characteristics. We showed one example with the evaluation problem for a matrix chain products which is known as matrix chain ordering problem (MCOP). We are currently studying the evaluation method for a matrix chain product on a parallel system. Also, the result of this work can be applicable to many other processor allocation problem with scalable tasks.

Acknowledgments

This research was supported in part by NON DIRECTED RESEARCH FUND, Korea Research Foundation. We are also grateful to FUJITSU Laboratories Ltd. for allowing us to use their facilities.

References

[1] F. T. Leighton, *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes*. Morgan Kaufmann, 1992.

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

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

[5] T. Hu and M. Shing, "Computation of matrix chain products. part I," *SIAM Journal on Computing*, vol. 11, pp. 362–373, May 1982.

[6] T. Hu and M. Shing, "Computation of matrix chain products. part II," *SIAM Journal on Computing*, vol. 13, pp. 228–251, May 1984.

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

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

[9] 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.

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

[11] 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, 1992.

[12] P. Ramanan, "An efficient parallel algorithm for the matrix chain product problem," *SIAM Journal on Computing*, vol. 25, Aug. 1996.