Performance of Matrix Multiplication Algorithms on AP1000

Naijie GU [+], Xiaoqing CHEN [+], Guoliang CHEN [+], Hidehiko TANAKA [++]

[+] Department of computer science and technology
University of science and technology of china, Hefei, Anhui, 230026
E-mail: njgu@cs.ustc.edu.cn, glchen@cs.ustc.edu.cn

[++] University of Tokyo, Department of electrical engineering
7-3-1 Hongo, Tokyo, Japan, 113
E-mail: tanaka@mtl.t.u-tokyo.ac.jp

Abstract: In this paper we give an experiment evaluation of three matrix multiplication algorithms on AP1000. Our analysis shows that the complexity of some well-known communication scheme is not so good as it was considered. Our experiment results show that the real computational times (time used for multiply sub-matrix) of these three algorithms are different though they have same theoretical complexity. Both from analysis and experiment results, it is shown that cannon's algorithm has the best behavior.

Keywords: Parallel algorithm, communication, scalability.

I. INTRODUCTION

Dense matrix multiplication is a basic operation that is widely used in many applications. There are a lot of papers discuss about this operation on parallel computer[1,2].

In this paper, we discuss the implementation details of three different matrix multiplication algorithms on AP1000, and describe two all-to-all broadcast schemes on 2-D torus. We also analyze the performance of each algorithm in detail, and give some experiment results. Our analysis shows that the cannon's algorithm[3] has the best behavior among these three algorithms, our experiment results support this conclusion. By the analysis, we show that the time complexity of some well-known communication scheme is not so good as it was considered. By the experiment, we discover that the real computational time complexity is different from algorithm to algorithm, though their theoretical complexities are all the same.

The computing model is described in section 2; the algorithms are introduced in section 3; the performance analysis is shown in section 4; and the experiment results are given in section 5.

II. COMPUTING MODEL

AP1000 parallel machine is a distributed memory MIMD system with 64 independent SPARC processors. Each processor has a 128 Kbyte direct-mapped copyback cache, 16 Mbyte of memory, and a Weitek floating-point unit (FPU) of theoretical peak speed 8.33 MFLOP (single precision) and 5.67 MFLOP (double precision). The topology of the AP1000 is a 2-D torus, with hardware support for wormhole routing.

By wormhole routing, the time required for a processor to send a message of m words to another processor is modeled as $t_s + m \cdot t_w$, suppose that no data congestion exist. The t_s is the message start-up cost and the t_w is the data transmission time per word.

III. ALGORITHMS

In this section we present the well-known distributed algorithms for multiplying two dense matrices A and B of size $n \times n$.

When we perform matrix multiplication on a parallel machine, the most important thing we must consider is how to partition the matrix and how to distribute the data upon processors. There are various data partition and distribution methods for mesh connected parallel machine. We consider two most popular data distribution methods--striped partition and checkerboard partition.

3.1 Algorithm for striped partition

Matrices A and B are divided into groups of contiguous complete rows, and each processor is assigned one such group. Fig. 1 shows one such data distribution of matrix A on four processors. Initially each processor has $n/p \times n$ elements of each matrix.

Algorithm for striped partition:

a) Transpose matrix B to B^T, i.e., perform a all-to-all communication.

b) Each processor calculates $C_{i,j} = \sum a_{i,k} \cdot b_{k,j}$, *where* $C_{i,j}$ $(0 \le j < n/p)$ *is a* $n/p \times n/p$ *sub-matrix of* C, $a_{i,k}$ and $b_{k,j}$ are elements of A and B *respectively.*

c) if $i \neq 0$ *then processor* PE_i *sends sub-matrix of* B *to* PE_{i-1}, *else sends the sub-matrix to* PE_{p-1}.

d) Repeats step b) and c), until executes step b) p *times.*

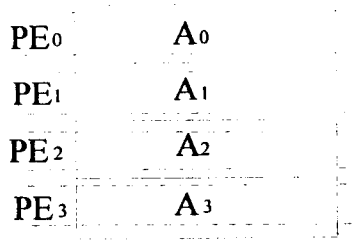

Fig. 1: Matrix A partitioned into 4 stripes.

3.2 Simple algorithm [1]

Matrices A and B are block partitioned into $\sqrt{p}$ blocks along each dimension, and each sub-matrix is naturally mapped onto each processor as shown in Fig.2. The sub-matrix $A_{i,j}$ and $B_{i,j}$ are mapped onto processor $PE_{i,j}$. Thus, initially each processor has n^2/p elements of each matrix.

Simple algorithm:

a) Broadcast sub-matrix of A *along X-direction, i.e., perform a all-to-all broadcast among processor row.*

b) Broadcast sub-matrix of B *along Y-direction, i.e., perform a all-to-all broadcast among processor column.*

c) Each processor calculates $C_{i,j} = \sum_{k=0}^{\sqrt{p}} A_{i,k} \times B_{k,j}$.

	0	1	2	3
0	A00	A01	A02	A03
1	A10	A11	A12	A13
2	A20	A21	A22	A23
3	A30	A31	A32	A33

Fig. 2: Matrix A distributed onto a 4×4 mesh.

3.3 Cannon's algorithm [1,3]

Matrices A and B are mapped naturally onto the processors as in Algorithm simple.

Cannon's algorithm

a) Skew sub-matrix of A *and* B *to an appropriate position, that are, shift* $A_{i,j}$ *left* i *steps, and shift* $B_{i,j}$ *up* j *steps.*

b) Each processor multiplies sub-matrix of A *and* B *stored in it, and adds the result with* $C_{i,j}$.

c) Shift sub-matrix of A *one step left, shift sub-matrix of* B *one step up.*

d) Repeat steps b) and c), until execute step b) $\sqrt{p}$ *times.*

IV. PERFORMANCE ANALYSIS

In this section we discuss the implementation details of above algorithms on AP1000, and analyze the performance of above algorithms.

4.1 Algorithm under striped partition

Communication complexity:

There are two communication phases in the algorithm for striped partition. One all-to-all communication to transpose matrix B, and each processor performs $p-1$ times sending sub-matrix of B one step ahead.

We use the all-to-all communication scheme described in [4], and the time required for this scheme is $\Theta((p-1)\cdot t_s + \sqrt{p}\cdot n^2/p \cdot t_w)$ [4]. The time used for send sub-matrix of B one step ahead is $t_s + n^2/p \cdot t_w$. The total communication time is $\Theta(2(p-1)\cdot t_s + (p-1+\sqrt{p})\cdot n^2/p \cdot t_w)$.

Space used:

We need to store sub-matrix of A, B, C in each processor, so the required space is at least $3 \cdot n^2/p$. Since we also need some extra space to perform the communication, the actual space used is about $4 \cdot n^2/p$.

Scalability:

The overhead for communication is:

$$T_{o1} \approx 2p^2 \cdot t_s + p^2 \cdot n^2/p \cdot t_w$$

The overhead by idling, synchronization:

$$T_{o2} \approx p^2$$

The iso-efficiency function [5] is:

$$W = c \cdot p^3$$

Where c is a constant related to efficiency E for a given system.

4.2 Simple algorithm

Communication complexity:

The algorithm consists of two communication phases. In the first phase, all processors in each row independently engage in an all-to-all broadcast of the sub-matrix of A among themselves. In the second phase, all processors in each column perform an all-to-all broadcast of sub-matrix of B. To do this, the following two broadcast schemes can be used.

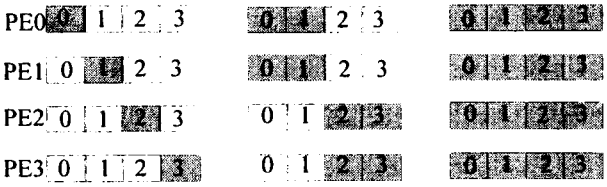

Fig. 3: Broadcast scheme 1 on four processors.

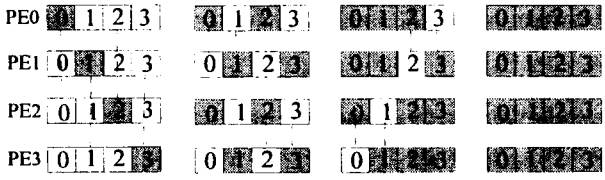

Fig. 4: Broadcast scheme 2 on four processors.

If we do not consider the data congestion during broadcasting, the time required for these two broadcast schemes is:

$$T^1_{broadcast} = \log\sqrt{p}\cdot t_s + (\sqrt{p}-1)\cdot n^2/p\cdot t_w$$

$$T^2_{broadcast} = (\sqrt{p}-1)\cdot t_s + (\sqrt{p}-1)\cdot n^2/p\cdot t_w$$

We can find above theoretical result in several papers, and it is shown that scheme 1 has a small time complexity than scheme 2. But, actually there are some data congestion exit while doing all-to-all broadcast. Now we give out the time complexity of these two schemes more precisely (assume that the parallel system is a 2-D mesh).

$$T^1_{broadcast} = \sum_{k=1}^{\log_{\sqrt{p}} p}\{t_s + 2^{k-1}(2^{k-1}\cdot n^2/p\cdot t)\}$$

$$= \log\sqrt{p}\cdot t_s + n^2/p\cdot t_w\cdot\sum_{k=1}^{\log_{\sqrt{p}} p} 4^{k-1}$$

$$= \log\sqrt{p}\cdot t_s + (p-1)/3\cdot n^2/p\cdot t_w$$

$$T^2_{broadcast} = \sum_{k=1}^{\sqrt{p}-1} t_s + \sum_{k=1}^{\log\sqrt{p}}(\sqrt{p}/2\cdot n^2/p\cdot t_w)$$

$$= (\sqrt{p}-1)t_s + (\log\sqrt{p}-1)\cdot\sqrt{p}\cdot n^2/p\cdot t_w$$

From above analysis, it is shown that the actual performance of scheme 2 is better than scheme 1. This result is different from the pure theoretical result. Our experiment results supported above analysis (Table 1, the time unit is second).

Table 1: Time used by two schemes ($p = 64$)

	$n = 1024$	$n = 2048$
Scheme 1	0,535999	2.244315
Scheme 2	0.518633	2.042738

Space used:

Since simple algorithm broadcast sub-matrix of A among processor's row, and broadcast sub-matrix of B among processor's column, the total space used in each processor is at least: $2\sqrt{p}\cdot n^2/p + n^2/p$.

Scalability:

The overhead for communication is:

$$T_{o1} \approx 2p^{3/2}\cdot t_s + \log p\cdot p^{3/2}\cdot n^2/p\cdot t_w$$

The overhead for idling and other operation is:

$$T_{o2} \approx 2p^{3/2} + 2\cdot p^{3/2}\cdot n^2/p$$

The iso-efficiency function is:

$$W = c\cdot\log^3 p\cdot p^{3/2}$$

Where c is a constant related to efficiency E for a given system.

4.3 Cannon's algorithm

Communication complexity:

The algorithm consists of two communication steps. In first communication step, skew sub-matrix of A and B to an appropriate position, that are, shift $A_{i,j}$ left i steps, and shift $B_{i,j}$ up j steps. In second communication step, each processor shifts sub-matrix of A one step left, and shifts sub-matrix of B one step up.

Since there exits data congestion while skewing sub-matrix of A and B to the appropriate position, the time required for this step is as follows:

$$T_{skew} = 2(t_s + \sqrt{p}/2\cdot n^2/p\cdot t_w)$$

The time used for shift the sub-matrices of A and B is: $T_{shift} = 2\sum_{k=1}^{\sqrt{p}-1}(t_s + n^2/p\cdot t_w)$

Space used:

From the theoretical view, the space required by cannon's algorithm is $3n^2 / p$. But when running this algorithm a parallel machine, we need some extra space for data communication. Thus the space used is at least $4n^2 / p$. To reduce the overhead, we use $5n^2 / p$ store units in our program.

Scalability:

The overhead for communication is:

$$T_{o1} \approx p^{3/2} \cdot t_s + 3p^2 \cdot n^2 / p \cdot t_w$$

The overhead for idling, synchronization:

$$T_{o2} \approx 2p^{3/2}$$

The iso-efficiency function is:

$$W = c \cdot p^{3/2}$$

Where c is a constant related to efficiency E for a given system.

V. EXPERIMENT RESULTS

In this section we give some experiment results of these three algorithms on AP1000 parallel machine.

Table 2 shows the time used for communication by these three algorithms. From the table and above analysis, we can find that cannon's algorithm has the smallest communication complexity.

Table 2: ($p = 64$, time unit: second)
Communication time used by three algorithms

	The size of matrix		
	$N = 1024$	$N = 2048$	$N = 4096$
Striped Algorithm	2.055804	8.611601	34.326796
Simple Algorithm	1.274853	5.644712	cannot perform
Cannon's Algorithm	0.510961	2.229583	8.851716

Table 3: Computation time used by three algorithms.

$p = 64$	The size of matrix		
	$N = 1024$	$N = 2048$	$N = 4096$
Striped Algorithm	67.107333	574.69463	4690.7846
Simple Algorithm	66.810016	540.15706	4425.7589
Cannon's Algorithm	65.280649	533.06957	4248.6059

Table 4: Overall time used by three algorithms.

$p = 64$	The size of matrix		
	$N = 1024$	$N = 2048$	$N = 4096$
Striped Algorithm	71.424047	596.605519	4941.47567
Simple Algorithm	68.963752	548.161738	can not perform
Cannon's Algorithm	65.802730	535.345238	4257.62345

These three algorithms have the same theoretical computational complexity n^3 / p, but in actual they are different. Our analysis and experiment results (Table 3 and Table 4) shown that cannon's algorithm has the smallest computation time and smallest overall time. Thus among these algorithms the cannon's algorithm is the best.

ACKNOWLEDGMENTS

We are grateful to Mr. Ichiro IDE and all the others in professor Tanaka's lab for supporting our research.

REFERENCES:

[1] H. Gupta, P. Sadayappn, Communication-efficient matrix multiplication on hypercubes, Parallel computing 22(1996): 75-99.

[2] A. Gupta and V. Kumar, Scalability of parallel algorithms for matrix multiplication, Proc. 1993 Int. Confer. on parallel Processing 3(1993) 115.

[3] L.E.Cannon, A cellular computer to implement the Kalman Filter Algorithm, Technical report, Ph.D. Thesis, Montana State University, 1969.

[4] Naijie GU, Guoliang CHEN, H. TANAKA, An efficient implication of PSRS algorithm on AP1000, Technical report, Department of computer science and technology, University of science and technology of china, 1996.
Also to be appeared on the Proc. of this Confer.

[5] V. Kumar, A. Grama, A. Gupta, G. Karypis, Introduction to Parallel Computing: design and analysis of algorithms, the Benjamin/Cummings Publish Company, Inc. 1994, 128-134.