Proceedings of the International MultiConference of Engineers and Computer Scientists 2009 Vol II
IMECS 2009, March 18 - 20, 2009, Hong Kong

An Empirical Distributed Matrix Multiplication Algorithm to Reduce Time Complexity

Md. Nazrul Islam, Md. Shohidul Islam, M.A. Kashem, M.R. Islam, M.S. Islam

Abstract— **Matrix multiplication is an integral component of most of the systems implementing Graph theory, Numerical algorithms, Digital control, Signal and image processing (i.e robotics, computer vision, artificial intelligence e.t.c). So reduction of multiplication time can influence drastically the overall system performance. Based on the importance, this paper presents a novel distributed algorithm for matrix multiplication to lower the time complexity efficiently. For distributed processing, computational time is usually analyzed assuming that all processors are of the same type and operating at same speed. i.e., homogeneous system. A number of autonomous machines are connected by a local area network that makes a distributed computing environment where server and multiple clients exchange their data or information by using message passing technique. The result shows that an enormous amount of time can be reduced by adopting such technique by dividing the tasks on different clients, where execution time grows rapidly with the increase of data on a single machine.**

Index Terms—**Distributed Matrix Multiplication, Homogeneous System, Sequential Algorithm, Time Complexity.**

I. Introduction

The main focus of this work is to present a distributed algorithm for Matrix Multiplication. We have developed the algorithm "*Distributed control, inner-product workers, multiple vectors per message (n/k rows, n columns per worker)*". We distribute the large order of matrices into number of clients which coordinate with server. The clients perform their calculation on the specific input data and send their results back to the server. As the total computation is done by several processors, the time complexity is reduced enormously.

Manuscript received on October 30, 2008. This work was supported in part by the IAENG Hong Kong and Department of Computer Science & Engineering, Dhaka University of Engineering & Technology, Gazipur-1700, Bangladesh.

M.Nazrul Islam, M. Shohidul Islam , M.A. Kashem, M.R. Islam and M.S. Islam are with the Department of Computer Science & Engineering in Dhaka University of Engineering & Technology, Gazipur-1700, Bangladesh. (e-mail: nazrul_ruet@yahoo.com, shohidulcse@duet.ac.bd, drkasheml l@duet.ac.bd, uzzal01328@yahoo.com and msislam_80@yahoo.com).

II. Comparative Features with Existing works

For efficient matrix multiplication different distributed algorithms exist such as (i). Distributed control, inner-product workers, (ii). Distributed control, outer-product workers, multiple vectors per message, (iii). Distributed control, inner-product workers, multiple vectors per message (n/k rows, n columns per worker), (iv). Winograd's method and (v). Hybrid Winograd-Strassen. From the design view of the algorithm, we have developed "distributed control, inner-product workers, multiple vectors per message (n/k rows, n columns per worker)"- as this algorithm reduces the time complexity needed by others. But it might show that the algorithm has some problems. One crucial issue is equal portion of data were not distributed to all processors and the last one gets in addition of remainder portion, as a result some of processors was heavily loaded than that of others. Consider a 100 x 100 order matrix and 6 processors to perform distributed computing then the first 5 processors compute 80 rows (each processor compute 16 rows of final matrix) and the last processor computes rest of data i.e. 20 rows of final matrix, but this is not fully distributed and varies time complexity when large order matrix is considered. We have considered remedy of this fact into our devised algorithm.

III. Algorithm Development

The time complexity of matrix multiplication depends on the number of operations which are performed according to the algorithm for specified input domain. Basically there are two approaches for matrix multiplication: sequential approach which is implemented by a single processor and parallel approach that is implemented by multiple processors as they behave like server-client relationship.

A. Sequential approach

Sequential approach is implemented by a single processor where time complexity is related to only one factor and that is computation time. Suppose ***a*** and ***b*** are the matrices to multiply; c_{ij} is the resultant matrix; ***k*** is the no. of clients interconnected to the server; ***n*** is the order of those matrices It is counted that all matrices have the same order. ***i, j*** and ***q*** are the variable to continue the looping process. The algorithm for sequential approach is:

```
procedure matrix multiplication (a, b: matrices)
for i:=0 to n-1
  for j:=0 to n-1
begin
c_ij :=0
for q:=1 to n-1
  c_ij := c_ij+a_iq ×b_qj
end
```

B. Distributed approach

Distributed model of computation has been implemented in java using client-server based socket programming wherein each client is assigned task as implied in the below given algorithm.

Algorithm for the client end:

```
process   client [i: =0 to n-1]
  integer a[n] –row i of matrix a
  integer b[n] –all of matrix b
  integer c[n] –row i of matrix c
  receive initial values for vector a and matrix b
  for[j:=0 to n-1]{
    c[j]:=0
    for[k:=0 to n-1]{
      c[j]:=c[j]+a[k]×b[k,j]
      }
    }
  send result vector c to the server process
```

The pseudo code for server end:

```
process   server
  integer a[n] –source matrix a
  integer b[n] – source matrix b
  integer c[n] – source matrix c
  initialize a and b
identify how many clients are connected to the same port
choose how many clients (k) you want to involve
calculate how many rows you want to send each client
for[i:=0 to n-1]{
        send row i of a to client[k]
        send all of b to client[k]
        }
for[i:=0 to n-1]{
        receive row i of c from client[k]
        print the results, which are shown now in matrix c}
```

IV. How Time Varies

A. Sequential Computing

Resultant matrix has n^2 entries. To find each entry requires a total of n multiplication and $n-1$ additions. Hence, a total of n^3 multiplications and n^2 $(n-1)$ additions are used. So, time Complexity is $O(n^3)$.

B. Distributed Computing

The execution time t_d is composed of two parts: a computation part says t_{comp}, and a communication part say t_{comm}; Thus t_d = *computation time* (t_{comp}) + *communication ime*(t_{comm})

i.e. $t_d = t_{comp} + t_{comm}$.

Resultant matrix has at least **1** entry. To find each entry requires a total of n multiplication and $n-1$ additions. Hence, a total of n multiplications and $(n-1)$ additions are used. So, the computation time complexity is $O(n)$.

$t_{comm} = t_{startup} + nt_{data}$; where, $t_{startup}$ is the startup time called message latency to pack and unpack data; t_{data}, transmission time to send one data word is a constant; and n is the no. of data words. So, communication time complexity is $O(n)$, hence, overall time complexity is less than sequential computation.

V. Results and Discussion

A. Experimental Results

From table 1 we may give our attention that how the complexity varies with the increase of clients on one side and another with the increasing of matrices order.

TABLE I Matrix multiplication time statistics

Order (n)	Sequential Approach	Distribute with 4 client	Distribute with 6 client	Distribute with 8 client	Distribute with 10 client
100	2.248 sec	0.891 sec	0.621 sec	0.431 sec	0.382 sec
300	7.797 sec	3.344 sec	3.262 Sec	3.122 sec	2.941 sec
500	1.045 min	36.844 sec	34.672 sec	30.41 sec	26.42 sec
700	2.380 min	56.421 sec	48.245 sec	44.35 sec	40.48 sec
1000	5.730 min	2.31 min	2.012 Min	1.82 min	1.64 min

So it should not be avoided, rather be noted for low ordered data distributed among more clients may switched to increase the time than should be for communication complexity. However, we would like to be frankly said that the more the machine increases the less will be the computational time which can be clearly verified by large ordered data or matrix (i.e. n>=300).

VI. Conclusion

This paper discussed to reduce the computation time for processing a huge number of data. In distributed system the performance is influence by dividing the task among processors. One important issue remains unsolved, if the system is considered as distributed heterogeneous system, where data is given again and again which has performed tasks quickly. We plan to address this issue in our future research.

REFERENCES

[1]. Nicholas Comino, V. Lakshmi Narasimhan, "A Novel Data Distribution Technique for Host-Client Type Parallel Applications", *IEEE Transactions on Parallel and Distributed Systems*, vol.13 , no.2, pp.97-110, Feb 2002.

[2]. Jyh-Jang Lim, Jai Menon and David Palmer "A Distributed Development Environment for Embedded Software" *SPE,* vol.23, no.11, Nov 1993.

[3]. Stephen A. Rees, James P. Black, "An Experimental Investigation of Distributed Matrix Multiplication Techniques". SPE, vol.21, no.10, Oct 1991.

[4]. Sartaj Shani,"*Computer Algorithm*", third edition, McGraw-Hill.

[5]. Andrew S. Tanenbaum, *"Computer Networks",* fourth edition, Prentice-Hall, Inc.

[6]. H. M. Deitel, P. J. Deitel, *"Java How to Program"*, sixth edition, Prentice-Hall, Inc.

[7]. Michael Allen, Barry Wilkinson, *"Parallel Programming"*, fifth edition, Prentice-Hall, Inc, ISBN:0131405632.