A Parallel Algorithm for Matrix Fast Exponentiation Based on MPI

Ji Zhang
College of Computer Science and Technology
Southwest Minzu University
Chengdu, China
e-mail: paodekuai2@163.com

Xu-chuan Zhou*
College of Computer Science and Technology
Southwest Minzu University
Chengdu, China
e-mail: xczhou@swun.edu.cn

***Abstract*—Matrix Fast Exponentiation Algorithm (MFEA) is widely used as an algorithm for fast solving the matrix power. But its efficiency can not meet the needs of the big data application. In this paper, we analyze the parallel potential of MFEA and propose a Parallel Matrix Fast Exponentiation Algorithm (PMFEA) based on message passing interface (MPI). PMFEA uses data partitioning to improve the processing capacity for the big data. The experimental results show that PMFEA achieves about 8.41X speedup compared to MFEA on 32-core server.**

***Keywords*- Matrix Exponentiation; MFEA; parallel; multi-core**

I. INTRODUCTION

Matrix power computation is an important numerical calculation, its performance directly influences others numerical calculation[1]. Many research has been done on matrix power computation in the world: an improved algorithms for matrix power computation based on LARPBS model was proposed in [1] and a parallel algorithm for scalable matrix exponentiation was proposed in [2]. With the advent of the big data age, the computational efficiency of existing algorithms can not meet the needs of data processing capacity of various applications. Hence, a more efficient algorithm is urgently needed.

Matrix Fast Exponentiation Algorithm (MFEA) is an algorithm for fast solving matrix power based on the divide and conquer approach. It has important applications in many problems such as PageRank Algorithm, linear recurrences, graph theory and so on. In order to improve the computational efficiency of MFEA, we analyze the feasibility of parallelizing MFEA with the multi-core processor, propose a Parallel Matrix Fast Exponentiation Algorithm (PMFEA) and implement PMFEA based on message passing interface(MPI). The experimental results show the superiority of PMFEA: PMFEA achieves about 8.41X speedup compared to MFEA on 32-core server.

II. MATRIX FAST EXPONENTIATION ALGORITHM

MFEA is an efficient algorithm for computing matrix powers based on the divide and conquer approach and the associative law of matrix multiplication. The main calculation method of MFEA is as follows: It continually cuts the power of matrix in half, adding an extra matrix multiplication if the power of matrix is odd, until the power of matrix is 1, and then returns the result to compute the matrix of higher powers, and finally gets the answer. The function expression of the algorithm is shown as:

$$A^q = \begin{cases} A^{\lfloor q/2 \rfloor} \times A^{\lfloor q/2 \rfloor} \times A, & \text{if } q \text{ is odd} \\ A^{\lfloor q/2 \rfloor} \times A^{\lfloor q/2 \rfloor}, & \text{if } q \text{ is even} \end{cases} \quad (1)$$

where A is original matrix and q is the power of matrix. Algorithm 1 shows the pseudo-code for the recursive implementation of MFEA

Algorithm 1 Recursive Implementation of MFEA

input: matrix A, power q
output: A^q

```
1:  function fastExp(q)
2:      if q = 1 then
3:          return A
4:      end if
5:      result ← fastExp(q / 2)
6:      result ← result × result
7:      if q is odd then
8:          result ← result × A
9:      end if
10:     return result
11: end function
```

III. PARALLEL ALGORITHM FOR MATRIX FAST EXPONENTIATION

A. Parallelism Analysis

In this paper, we have deeply analyzed MFEA and found that the runtime complexity of decomposing q, the power of matrix, in MFEA has reached $O(\log_2 q)$ via the approach of divide and conquer. When the matrix multiplication with a runtime complexity of $O(m^3)$ is adopted, the runtime complexity of MFEA is $O(m^3 \cdot \log_2 q)$. From the runtime complexity of MFEA, we can find that the algorithm take less time to decompose the power and spend more time in the matrix multiplication which confine the capacity of MFEA facing big data. So, the optimization for matrix multiplication becomes the most important factor to improve the efficiency of MFEA.

Assuming all the matrices are $m \times m$ square matrices. As already known in basic algebra texts, an element c_{ij} in the matrix C_m, $C_m = A_m \times B_m$, is calculated as:

$$c_{ij} = \sum_{k=1}^{m} a_{ik} \times b_{kj}$$

where a_{ij}, b_{ij}, c_{ij} are the elements in the i-th row and j-th column, for matrices A, B, and C, respectively. Thus, each element in matrix C can be calculated independently. We can parallelize the matrix multiplication by dividing the original matrix into smaller matrices and calculating these matrices at different nodes. By this method, matrix multiplication can make full use of parallel computing resources and improve observably its computing efficiency as the increase in the number of nodes.

Through comparative experiments, we found that MFEA takes less time to decompose the power. If we parallelize this process, the computing efficiency of the algorithm will decrease due to the communication cost between nodes. Thus, in order to minimize the runtime complexity of PMFEA, the process of decomposing power is used as the sequential part of PMFEA to run on all nodes.

Based on the above analysis, we can obtain the proportion of the part performed via sequential way in PMFEA is 11%. According to the law of Amdahl[3], ψ, the maximum of speedup of parallel algorithm, can be calculated as:

$$\psi \leq \frac{1}{f+(1-f)/p} \quad (2)$$

where f is the proportion of the part performed by sequential way and p is the number of processor. So, according to the equation (2), the maximum of speedup of PMFEA in theory is calculated as:

$$\lim_{p \to +\infty} \frac{1}{0.11 + (1-0.11)/p} \approx 9.1$$

In conclusion, the parallelization of MFEA is feasible and will improve significantly the performance of MFEA.

B. Design and Implementation

In this paper, PMFEA is done by using MPI. As a programming language, C++ is used. We adopt some optimization tips to improve the runtime efficiency of PMFEA, during design and implementation.

The main process of PMFEA is as follows:

a) The master node reads the data and distributes it to other nodes.

b) All nodes decompose the matrix power.

c) To determine whether matrix multiplication is required and it is performed if necessary.

d) Repeat b) and c) until the matrix power is 1, the result is obtained.

As we discussed before, the process of decomposing power is executed in the following way. Firstly, we launch the calculation on all nodes. When a matrix multiplication is required, the master node divides the matrix into smaller matrices and respectively sends them to computing nodes. Then, computing nodes perform matrix multiplication and send the result to the master node. Finally, the master node merges results and all nodes continue executing the operation of decomposing. In the implementation of the process of decomposing, we optimize the strategy of divide and conquer to avoid the external cost caused by recursion. We express the power as binary and read the binary representation from right to lift, starting with the second bit from right. Start with the matrix A, and every time we read a 0 bit, square what we have got. Every time we read a 1 bit, square what we have got and multiply by A. For example, A^{13} can be computed as follow.

$$A^{13} = A^{1011_2} = A^{2^3+2^1+2^0} = A^{2^3} \times A^{2^1} \times A^{2^0}$$

This method makes the liner loop can be used to implement the process of decomposing power in the same runtime complexity. The implementation is showed in Algorithm 2.

Algorithm 2 Decomposing power

input: matrix A, power q
output: A^q

```
1: if taskid = master then \*master*\
2:     while q > 0 do
3:         if q is odd then
4:             result = Mastermatmul(result, A)
5:         end if
6:         A = Mastermatmul(A, A)
7:         q = q >> 1
8:     end while
9: else\*slave*\
10:    while q > 0 do
11:        if q is odd then
12:            Slavematmul()
13:        end if
14:        Slavematmul()
15:        q = q >> 1
16:    end while
17: end if
```

In the design of parallel matrix multiplication, we decompose the matrix by row. Assuming all matrices are $m \times m$ square matrices and the number of computing nodes is p ($p \leq m$), matrices can be decomposed as:

$$C = \begin{bmatrix} C_1 \\ \vdots \\ C_p \end{bmatrix} = \begin{bmatrix} A_1 B \\ \vdots \\ A_p B \end{bmatrix}$$

Firstly, master node divides matrix A into p number of smaller matrices that are adjacent $\lfloor m/p \rfloor$ or $\lceil m/p \rceil$ rows in matrix A and sends A_i and B to i-th computing node. Then, each computing node calculates $\lfloor m/p \rfloor$ or $\lceil m/p \rceil$ rows in the final matrix C and sends C_i to master node. Finally, master node merges all C_i into matrix C. So, the runtime complexity of matrix on each computing node is $O(m^3/p)$. In the ideal condition, the runtime complexity of parallel matrix multiplication equals the computing node's and has a

significant improvement of performance compared with sequential matrix multiplication.

According to our design and implementation, we can obtain the runtime complexity of PMFEA, in theory, is $O((m^3 \cdot log_2 q)/p)$, which is $1/p$ times of MFEA's. So, we predict the parallelization for MFEA will improve its performance and this improvement will increase with the increase in the number of computing nodes.

IV. Experimental Results and Analysis

The experimental environment is shown in Table I.

TABLE I. Experimental Environment

parameter	value
server	HP ProLiant ML350 Gen9
CPU	Intel Xeon E5-2600 v332 cores2.40GHz
memory	32GDDR4 1600MHz
operating system	redhat 7.0
compiler	GCC 5.2

In these experiments, all the matrices are $m \times m$ square matrices in which all elements are 1. We obtained the experimental results by repeating 20 times for every experiment and taking the mean of experimental results.

A. Experiments with Matrix Size

To experimentally determine the impact of the matrix size to the performance of PMFEA, we fixed q, the power of matrix, at 64 and experimented PMFEA in different m (200, 400, 600, 800, 1000, 1200), size of matrices, and p (4, 8, 16, 32), number of cores. We also compare these results to the performance of MFEA. Experimental results are shown in Table II and Fig. 1.

TABLE II. Measured Performance of Parallel Matrix Fast Exponentiation Algorithm of m * m matrices by different cores (s)

matrix size	MFEA	4-core	8-core	16-core	32-core
200	0.457	0.223	0.208	0.117	0.075
400	3.02	1.74	0.745	0.577	0.422
600	7.496	2.844	1.557	1.081	0.891
800	23.383	8.747	4.584	2.81	2.825
1000	32.4	11.746	6.102	5.147	4.145
1200	53.36	19.774	10.287	7.15	6.809

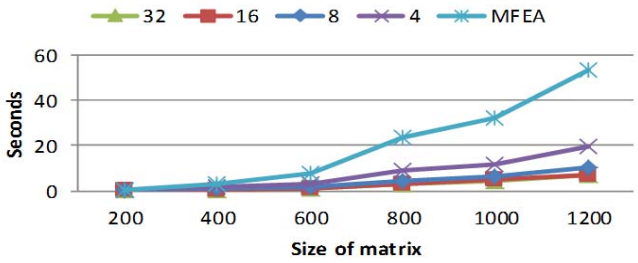

Figure 1. Measured Performance of Parallel Matrix Fast Exponentiation Algorithm of m * m matrices by different cores

From Table II and Fig. 1, we observe a substantial difference in the run-time between PMFEA and MFEA and the gap between them increases with the size of matrices and the number of cores. For m = 1200, the run-time of MFEA is 53.36s, but the run-time of PMFEA is 6.8s on 32 cores machines, which perform 7 times efficiently than MFEA. At the same time, we find that PMFEA shown good performance in a certain number of cores but it also shows the almost same results when the number of cores is increased more. In order to analyze the reason of this phenomenon, we calculated the speedup of PMFEA on different p(8, 16, 32), the number of cores. The result is shown in Fig. 2. It is seen that the speedup of PMFEA increases with the size of matrices on 8 cores and 16 cores machines and reached 8.41 with the matrices in size 600 on 32 cores machines, which is close 9.1, the maximum theoretical speedup of PFMEA, proposed in this paper and verifies the feasibility of PMFEA. But the speedup of PMFEA decreases with the matrices in size 800 on 32 cores machines. The reason of this phenomenon is that the external communication cost caused by data transmission among nodes. The external communication cost that will increase with the size of matrices leads to the actual speedup is lower than the ideal speedup. When the reduction of calculation cost is lower than the increase of communication cost, the run-time of PMFEA will increase and the speedup of PMFEA will decrease.

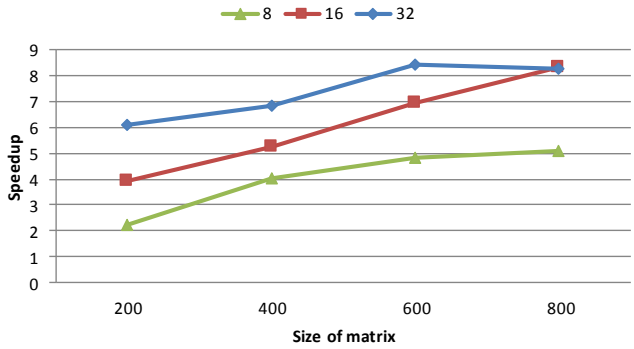

Figure. 2 Measured speedup of PMFEA of m * m matrices by different cores

B. Experiments with matrix power

To experimentally determine the impact of the power of matrix to the performance of PFMEA, we fixed m, the size of matrix, at 600 and experimented PMFEA in different q (8, 16, 32, 64), the power of matrices, and p (4, 8, 16, 32), the number of cores. We also compared these results with the performance of MFEA. Experimental results are shown in Fig. 3.

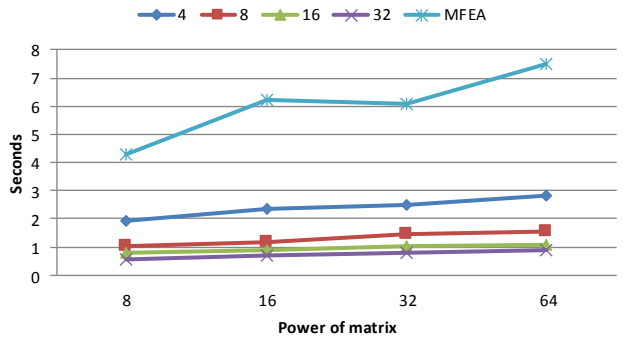

Figure 3. Measured Performance of Parallel Matrix Fast Exponentiation Algorithm of exponentiation by different cores

In Fig. 3, the run-time of PMFEA decreases rapidly as the increase in the number of cores, which is similar to the change of PMFEA's performance in different size of matrices. These results demonstrate the performance of PMFEA is superior to the performance of MFEA in both different size of matrices and different power of matrices. We can also find that when PMFEA run on the ascertain number of cores, the change of the run-time is little in different power of matrices, which verifies our analysis that the key factor impacting the performance of MFEA is the size of matrix instead of the power of matrix.

V. CONCLUSION

In this paper, based on multi-core architecture and message passing interface (MPI), a Parallel Matrix Fast Exponentiation Algorithm(PMFEA) is designed and implemented. Firstly, Matrix Fast Exponentiation Algorithm (MFEA) is introduced and the problems in the era of big data and the feasibility of parallel MFEA is deeply analyzed. Then, for the efficiency bottlenecks of MFEA, PMFEA based on MPI is designed and the strategy of divide and conquer of PMFEA is optimized in implementation. Finally, the experimental results clearly indicate that the performance of PMFEA is higher than the performance of MFEA and the speedup of PMFEA has reached 8.41 on 32-cores server. In further research, we will focus on improving algorithm design to reduce communication cost and memory overhead.

ACKNOWLEDGMENT

This research was supported by the Fundamental Research Funds for the Central Universities, Southwest University for Nationalities, (2015NYB12).

REFERENCES

[1] H. Chen and Shen, "An Improved Algorithms for Matrix Power Computation and Analysis of It's Complexity," *Computer Engineering & Applications,* 2003.

[2] H. J. Chen, K. R. Li, and J. Q. Luo, "Fast and scalable algorithms for matrix power computation and their applications," *J.yangzhou Univ.nat.sci.ed,* pp. 36-40, 2004.

[3] G. L. Chen,*"Parallel algorithm practice"*2004

[4] R. C. Agarwal, S. M. Balle, F. G. Gustavson, and M. Joshi, "A three-dimensional approach to parallel matrix multiplication," *Ibm Journal of Research & Development,* vol. 39, pp. 575-582, 1995.

[5] L. E. Cannon, "A CELLULAR COMPUTER TO IMPLEMENT THE KALMAN FILTER ALGORITHM," *A Cellular Computer to Implement the Kalman Filter Algorithm,* 1969.

[6] J. Mcnames, "A Fast Nearest-Neighbor Algorithm Based on a Principal Axis Search Tree," *IEEE Transactions on Pattern Analysis & Machine Intelligence,* vol. 23, pp. 964-976, 2001.

[7] G. Ballard, J. Demmel, O. Holtz, B. Lipshitz, and O. Schwartz, "Brief announcement: strong scaling of matrix multiplication algorithms and memory-independent communication lower bounds," in *Twenty-Fourth ACM Symposium on Parallelism in Algorithms and Architectures*, 2012, pp. 77-79.

[8] G. Ballard, J. Demmel, O. Holtz, B. Lipshitz, and O. Schwartz, "Communication-optimal parallel algorithm for strassen's matrix multiplication," in *Twenty-Fourth ACM Symposium on Parallelism in Algorithms and Architectures*, 2012, pp. 193-204.

[9] G. Ballard, J. Demmel, O. Holtz, and O. Schwartz, "Graph expansion and communication costs of fast matrix multiplication," *Journal of the Acm,* vol. 59, p. 32, 2012.

[10] H. Cheng, Y. Q. Zhang, X. Y. Zhang, and L. I. Yu-Cheng, "Implementation and Performance Analysis of CPU-GPU Parallel Matrix Multiplication," *Computer Engineering,* vol. 36, pp. 24-25, 2010.

[11] Q. K. Liu, M. A. Ming-Wei, and W. C. Yan, "Parallel matrix multiplication based on MPI+CUDA asynchronous model," *Journal of Computer Applications,* 2011.

[12] N. Taghiyev and M. Akcay, "Parallel matrix multiplication for various implementations," in *International Conference on Application of Information and Communication Technologies*, 2013, pp. 1-5