Parallel Implementation and Performance Evaluation of Cannon's Algorithm

Biyang Fu
School of Computing
Clemson University
Clemson, US
Biyangf@clemson.edu

Haotian Deng
School of Computing
Clemson University
Clemson, US
hdeng@clemson.edu

General Comments:

1) The paper is well structured with good introduction of the problem, sufficient background of the approaches and proposed new methods, and good analysis on the results.

2) The write-up is in a very good shape, with least typos, one of the best among the 9 papers.

3) Please do pay attention to the page limit whenever you are preparing a technical paper in the future. It's a critical requirement that a submitted proposal or paper could simply be rejected if it exceeds the limit specified by the journal or conference. You have to manage to keep your literature within the page limit. In this minipaper, for example, you could reduce the length on describing the state-of-the-art algorithm of the Background and shrink a little bit on Conclusion and future work, etc.

***Abstract*—In computer science, Cannon's algorithm is an algorithm that optimizes traditional matrix multiplication and is a storage-efficient algorithm. Unlike the traditional matrix multiplication, the cannon's algorithm greatly reduces the storage capacity of matrix multiplication. This paper first introduces the principle of the Cannon's algorithm, then makes an experiment for Cannon's algorithm, and finally evaluates the performance of the implementation.**

***Index Terms*—parallel, matrix multiplication, Cannon's algorithm, performance**

I. Introduction

In the past three decades, matrix multiplication [9] [6] is one of the most important basic operations in computer science, and a number of different algorithms have been proposed for implementation of parallel matrix multiplication [7]. The Cannon's algorithm [5] is one of the most efficient and representative parallel matrix multiplication calculations on distributed memory architectures. This article explains the principle of Cannon's algorithm in detail, analyze and evaluate the performance of the algorithm through its parallel computing implementation using OpenMP [3]. Section II provides a background of the matrix multiplication; Section III describes the proposed methods; Section IV describes our experimental results and analysis ; Section V describes the conclusion and future work.

II. Background

Parallel matrix multiplication has been explored and investigated extensively in the previous two decades. There are diverse approaches to optimize the matrix multiplication [10].

A. Serial matrix multiplication

Several approaches have been proposed to optimize matrix multiplication by improving spatial and temporal locality. Blocking or tiling is one such fundamental techniques. Regardless of its generalization, blocking is architecture dependent whereas cache oblivious algorithms are an architecture independent substitute to the blocked algorithms. The divide and conquer paradigm is used by cache oblivious algorithms. Chatterjee et al. bestowed a proposal regarding recursive array layouts and fast matrix multiplication [11]. According to them cache oblivious method is to utilizing a recursive structure for the matrices. Conversely, conventional implementations of the Basic Linear Algebra Subroutines (BLAS) [12] libraries are primarily based on the blocking approach and hence require optimization on a particular hardware platform. The BLAS routines provide standard building blocks to perform basic vector and matrix operations. As the BLAS are proficient, convenient, and extensively accessible, they are generally used in the development of high quality linear algebra software, e.g. LAPACK.

B. Parallel matrix multiplication

Several parallel matrix multiplication algorithms have been proposed for distributed memory. Different layouts such as 1D Layout and 2D Layout were used for the optimization purpose. 1D Layout was found to be much slower than serial. 2D Layout includes Cannon's matrix, scalable universal matrix multiply and recursive layouts. The Scalable Universal Matrix Multiplication Algorithm (SUMMA) [6] [13] is a very useful algorithm which needs less workspace and overcomes the necessity of a square 2D grid. Cannon [5] introduced the first efficient distributed algorithm for two dimensional meshes parallel matrix multiplication providing theoretically optimal communication cost. Constant storage requirements and independency of number of processors are the major advantage of the algorithm.

III. Proposed Methods

A. Cannon's Algorithm

Cannon's algorithm is a distributed algorithm [1] for matrix multiplication for two-dimensional meshes. It is especially suitable for computers laid out in an N × N mesh. While Cannon's algorithm works well in homogeneous 2D grids [2], extending it to heterogeneous 2D grids has been shown to be difficult.

The main advantage of the algorithm is that its storage requirements remain constant and are independent of the number of processors.

B. Algorithm Details

- Consider two n × n matrices A and B partitioned into p blocks.
- A(i , j) and B(i , j) (0 i,j p) of size (n p)×(n p) each.
- Process P(i , j) initially stores A(i , j) and B(i , j) computes block C(i , j) of the result matrix.
- The initial step of the algorithm regards the alignment of the matrixes.
- Align the blocks of A and B in such a way that each process can independently start multiplying its local sub-matrices.
- This is done by shifting all submatrices A(i , j) to the left (with wraparound) by i steps and all submatrices B(i , j) up (with wraparound) by j steps.
- Perform local block multiplication.
- Each block of A moves one step left and each block of B moves one step up (again with wraparound).
- Perform next block multiplication, add to partial result, repeat until all blocks have been multiplied.

C. OpenMP

The OpenMP API [4] covers only user-directed parallelization, wherein the programmer explicitly specifies the actions to be taken by the compiler and runtime system in order to execute the program in parallel. In addition, compliant implementations are not required to check for code sequences that cause a program to be classified as none conforming. Application developers are responsible for correctly using the OpenMP API to produce a conforming program [8].

IV. Results and Analysis

A. Environment Setting

In the environment, we use Palmetto Cluster [14] as computing platform. And Computer configurations are Processor: Intel Xeon E5-4627v4 with 40 cores, RAM: 1.0TB, System Type: 64 bit Linux operating system. System also was loaded with the GCC module with a 4.8.1 version and an Openmpi module with 1.8.4 version.

TABLE I
Performance of Cannon's Algorithm

Matrix Size	Number of Threads			
	1	4	16	25
500	2.27	1.39	1.04	0.60
1000	15.68	7.71	4.80	2.05
1500	59.03	21.90	12.11	4.86
2000	134.79	65.89	25.75	9.30

B. Results

After continuous experimental testing, we have obtained the experimental results shown in TABLE 1.

We tested the combination of different matrix sizes and different number of threads to obtain and record the time spent. The table shows the changes in the number of elapsed time with the number of threads from 1 to 25. The numbers of first column indicates the dimensions of the matrices, e.g., 500 means 500x500 matrix. The numbers of the second row show the numbers of threads. The numbers of the second to the fifth column shows the elapsed time in different threads and matrix size settings. When number of thread is 1, program is executed sequentially, so this elapsed time can be used as a baseline.

C. Analysis

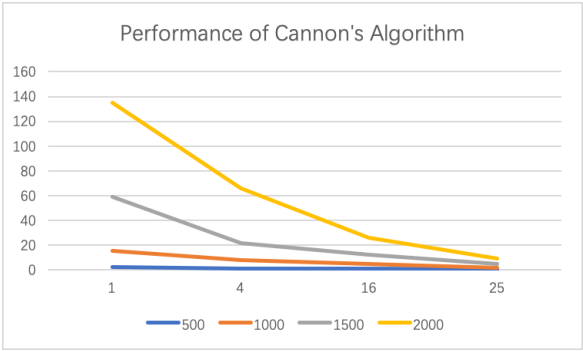

Fig. 1. Performance of Cannon's Algorithm

As shown in Fig. 1, The different colors represent different matrix sizes. For example, the yellow line represents the 2000 dimensions matrix. The x axis is the number of threads, the y axis are elapsed times. We found that when the matrix size is too small like 500x500, Cannon's algorithm cannot reflect the advantages of the parallel algorithm because the calculation time is relatively small compared to the initialization of threads and communication costs. As the size of the matrix increases, the parallelization advantage brought by the Cannon's algorithm becomes more and more obvious. In addition, we found that as the number of threads continues to increase, the time required for operations does not decrease proportionally. This is because the cost of thread communication between threads is high, and the matrix multiplication operation is nonlinear.

V. Conclusion and Future Work

The main contribution of this project is implementing Cannon's algorithm. For multi-core computers, Cannon's algorithm can be efficiently implemented. Using OpenMP as our APIs and platform to implement Cannon's algorithm, it is very meaningful. Matrix multiplication is one of the most important basic operations in computer science, Cannon's algorithm also is one of the most efficient parallel matrix multiplication calculations that is widely used. Studying on Cannon's algorithm can help us understand the parallel or high-performance computing more profoundly, and knowledge can be applied to real life. In future work, we will further optimize Cannon's algorithm by reducing communication costs, pre-processing the calculated matrix, and optimizing for the sparse matrix. Also Cannon's algorithm have a limitation which is suitable for homogenous 2D grids but its extension is difficult to heterogeneous 2D grids.

References

[1] Y. Li and H. Li, "Optimization of Parallel I/O for Cannon's Algorithm Based on Lustre," 2012 11th International Symposium on Distributed Computing and Applications to Business, Engineering Science, Guilin, 2012, pp. 31-35.

[2] L.E. Cannon, "A cellular computer to implement the Kalman filter algorithm," Ph.D. dissertation, Montana State Univ., Bozeman, MT, 1969.

[3] OpenMP Application Program Interface, Version 4.0. http://www.openmp.org/mp-documents/OpenMP4.0.0.pdf.

[4] Guide into OpenMP: Easy multithreading programming for C++. https://bisqwit.iki.fi/story/howto/openmp/

[5] Hyuk-Jae Lee, James P. Robertson, José A. B. Fortes, "Generalized Cannon's algorithm for parallel matrix multiplication" Published in ICS '97 1997 Computer Science

[6] Manojkumar Krishnan and J. Nieplocha, "SRUMMA: a matrix multiplication algorithm suitable for clusters and scalable shared memory systems," 18th International Parallel and Distributed Processing Symposium, 2004. Proceedings., Santa Fe, NM, USA, 2004, pp. 70-.

[7] B. S. Samantray and D. Kanhar, "Implementation of dense matrix multiplication on 2D mesh," 2014 International Conference on High Performance Computing and Applications (ICHPCA), Bhubaneswar, 2014, pp. 1-5.

[8] P. Gorlani, T. Kenter and C. Plessl, "OpenCL Implementation of Cannon's Matrix Multiplication Algorithm on Intel Stratix 10 FPGAs," 2019 International Conference on Field-Programmable Technology (ICFPT), Tianjin, China, 2019, pp. 99-107.

[9] P. Rathod, A. Vartak and N. Kunte, "Optimizing the complexity of matrix multiplication algorithm," 2017 International Conference on Intelligent Computing and Control (I2C2), Coimbatore, 2017, pp. 1-4.

[10] S. Afroz, M. Tahaseen, F. Ahmed, K. S. Farshee and M. N. Huda, "Survey on matrix multiplication algorithms," 2016 5th International Conference on Informatics, Electronics and Vision (ICIEV), Dhaka, 2016, pp. 151-155.

[11] S. Chatterjee, A. R. Lebeck, P. K. Patnala and M. Thottethodi, "Recursive array layouts and fast matrix multiplication," in IEEE Transactions on Parallel and Distributed Systems, vol. 13, no. 11, pp. 1105-1123, Nov. 2002.

[12] Basic Linear Algebra Subprograms (BLAS) Available online at http://www.netlib.org/blas/.

[13] van de Geijn RA, Jerrell W (1997), SUMMA: scalable universal matrix multiplication algorithm. Concurr Pract Exp 9(4), pp.255–274

[14] Palmetto Cluster Documentation Available online at https://www.palmetto.clemson.edu/palmetto/index.html