Mathematical Problems of Computer Science 50, 111--118, 2018.

Performance Analysis of Matrix Multiplication Algorithms Using MPI and OpenMP

Tigran M. Galstyan

Institute for Informatics and Automation Problems of NAS RA
e-mail: tig.galstyan.96@gmail.com

Abstract

The combination of OpenMP and MPI in programming is called hybrid programming. Hybrid programming (through messages and shared memory) has gained an important role since the appearance of cluster architectures. A hybrid programming method combines the MPI and OpenMP libraries to use this hierarchical multi-core architecture. The purpose of this work is to carry out the performance analysis of matrix multiplication algorithms in a cluster system. Each node in the cluster consists of multiple core CPUs, in which memory is distributed among the nodes and shared memory. Algorithms use MPI as a message-passing mechanism and OpenMP as shared memory.

Keywords: Hybrid, OpenMP, MPI, Matrix Multiplication, Fox.

1. Introduction

Parallel algorithms play an important role in the computation of high-performance computing environment. Dividing a task into smaller tasks and assigning them to different processors for parallel execution are two key concepts in the performance of parallel algorithms. Multiprocessor machines allow simultaneous execution of different application programs on different processors. They also allow a single application program to execute faster if it can be rewritten to use multiple processors [1]. The most common way to write a parallel program is to use a sequential language and a subroutine library. The bodies of process are written in the sequential language such as C. Process creation, communication and synchronization are then programmed by calling the library function. For message passing environment, we use the MPI. The MPI functions are included in a header file called *mpi.h* [2, 3, 4]. For shared memory, we use the OpenMP. The OpenMP functions are included in a header file called *omp.h* [2, 5].

Objective

The purpose of this work is to study hybrid programming. For example, Matrix Multiplication using MPI and OpenMP. To carry out performance analysis of matrix multiplication algorithms in cluster system. Implement matrix multiplication algorithms in C using OpenMP and MPI. To create a *Hybrid algorithm* and to compare it with famous matrix multiplication algorithms, for example, Fox algorithm. To compare algorithms by increasing the size of order matrix.

2. MPI

Message Passing Interface (MPI) is a standardized and portable message-passing standard designed by a group of researchers from academia and industry to function on a wide variety of parallel computing architectures. The standard defines the syntax and semantics of a core of library routines useful to a wide range of users writing portable message-passing programs in C, C++, and Fortran. There are several well-tested and efficient implementations of MPI, many of which are open-source or in the public domain. They fostered the development of a parallel software industry, and encouraged the development of portable and scalable largescale parallel applications [6].

3. OpenMP

The OpenMP (Open Multi-Processing) API supports multi-platform shared-memory parallel programming in C/C++ and Fortran. The OpenMP API defines a portable, scalable model with a simple and flexible interface for developing parallel applications on platforms from the desktop to the supercomputer. An application built with the hybrid model of parallel programming can run on a computer cluster using both OpenMP and Message Passing Interface (MPI), such that OpenMP is used for parallelism within a (multi-core) node while MPI is used for parallelism between nodes. Attempts have also been made to run OpenMP on software distributed shared memory systems, to translate OpenMP into MPI and to extend OpenMP for non-shared memory systems [7].

4. Matrix Multiplication

In mathematics, matrix multiplication or matrix product is a binary operation that produces a matrix from two matrixes. In more detail, if A is an $n\ X\ m$ matrix and B is an $m\ x\ p$ matrix, their matrix product AB is an $n\ x\ p$ matrix, in which the m entries across a row of A are multiplied by the m entries down a column of B and summed to produce an entry of AB (Figure 1) .

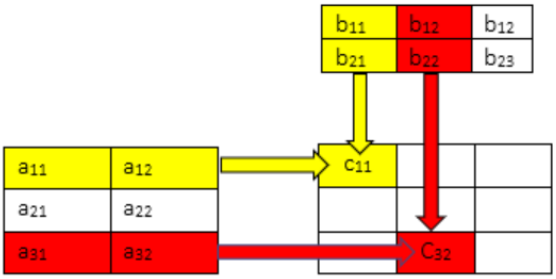


Fig. 1. Matrix Multiplication.

If A is an $n\ x\ m$ matrix and B is an $m\ x\ p$ matrix, then $C = AB$ is an $n\ x\ p$ matrix, and the value of each element in C is defined as:

$$C_{ij} = A_{i1}B_{1j} + \cdots + A_{im}B_{mj} = \sum_{k=1}^{m} A_{ik}B_{kj}.$$

Its computational complexity is $O(n^3)$ (for $n\ x\ n$ matrices). To improve performance, we have added OpenMP and MPI.

5. Hybrid Experiment

Matrix C is calculated in each node using OpenMP. To do so, each process receives a piece of the matrix A and the matrix B, sent to all processes. The number of orders C matrix is divisible by the number of processes. The algorithm uses a master/worker-type interaction, where the master works both as coordinator and as a worker. It divides the matrix A into pieces to be processed, and then generates the processing phases (Figure 2).

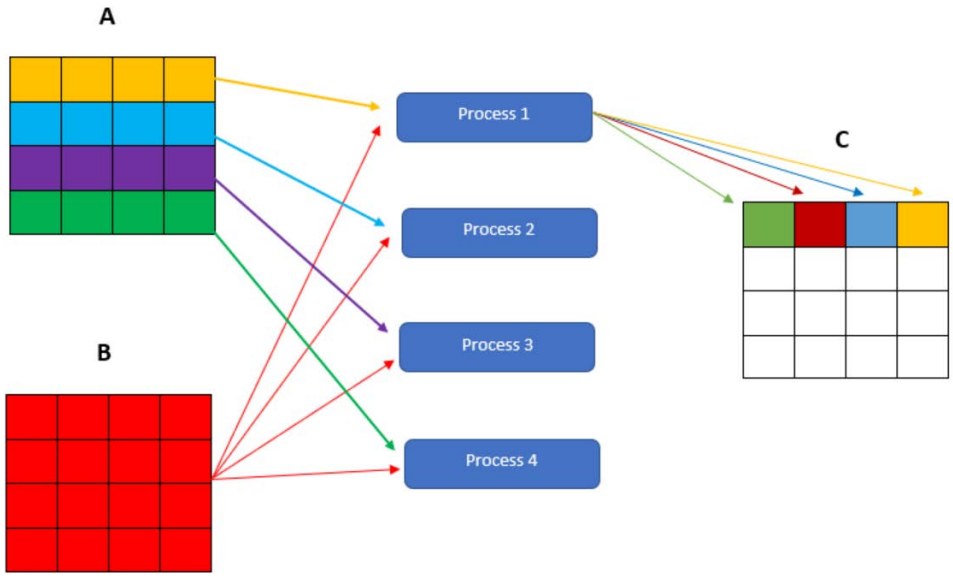


Fig. 2. Matrix multiplication by processes.

6. Fox`s Algorithm

Fox algorithm is one of the known solutions to this problem. Details of the algorithm are given below: [8]

A and B are $n\ xn$ matrixes.
Compute $C = AB$ in parallel.
Let $q = \sqrt{p}$ be an integer such that it divides n, where p is the number of processes. Create a Cartesian topology [9] with processes $(\mathrm{i},\mathrm{j}), (\mathrm{i},\mathrm{j}\ =\ 0,\ldots,\mathrm{q}\ -\ 1\ )$.
Denote $m = n/q$ Distribute A and B by blocks on p processes so that $A_{i,j}$ and $B_{i,j}$ are $m\ x\ m$ blocks stored on process (i,j).
On process (i,j),we want to compute:

$$C_{ij} = \sum_{k=0}^{q-1} A_{ik}B_{kj} = A_{i0}B_{0j} + A_{i1}B_{1j} + \cdots + A_{i,j-1}B_{i-1,j} + A_{ij}B_{ij} + A_{i,j+1}B_{i+1,j} + \cdots + A_{i,q-1}B_{q-1,j}$$

Rewrite this as:

Stage	Compute
0	$C_{ij} = A_{ii}B_{ij}$
1	$C_{ij} = C_{ij} + A_{i,i+1}B_{i+1,j}$
…	…..
	$C_{ij} = C_{ij} + A_{i,q-1}B_{q-1,,j}$
	$C_{ij} = C_{ij} + A_{i,0}B_{0,j}$
	$C_{ij} = C_{ij} + A_{i,1}B_{1,j}$
…	….
q – 1	$C_{ij} = C_{ij} + A_{i,i-1}B_{i-1,j}$

Each process computes in stages:

Stage 0

- process (i,j) has A_{ii}, B_{ij}but needs $\boldsymbol{A_{i,i}}$
- process (i,i) broadcasts $\boldsymbol{A_{i,j}}$ across processor row i
- process (i,j) computes $\boldsymbol{C_{ij} = A_{ii}B_{ij}}$

Stage 1

- process (i,j) has A_{ij}, B_{ij} but needs $\boldsymbol{A_{i,i+1}B_{i+1,j}}$
  - shift the jth block column of B by one block up (block 0 goes to block q − 1)
  - process $(\mathrm{i},\mathrm{i}+1)$ broadcasts $\mathrm{A_{i,i+1}}$ across processor row i
- process (i,j) computes $\boldsymbol{C_{ij} = C_{ij} + A_{i,i+1}B_{i+1,j}}$

Similarly, in the next stages.

7. Results

The following tables and charts show the execution times (expressed in seconds), the difference in time between the algorithms. Table 1 and Chart 1 show the difference between the Hybrid experiment, the Sequential algorithm and the Sequential algorithm with OpenMP.

Table 1:

	Sequential (sec)	Sequential +OpenMP (sec)	Hybrid (sec)
200x200	0,06	0,059	0,04
400x400	0,65	0,59	0,3
500x500	2	1.28	0.36
1000x1000	23	20	10
1500x1500	102	90	46
2000x2000	280	200	128
2500x2500	513	439	280
3000x3000	995	831	500

Chart 1:

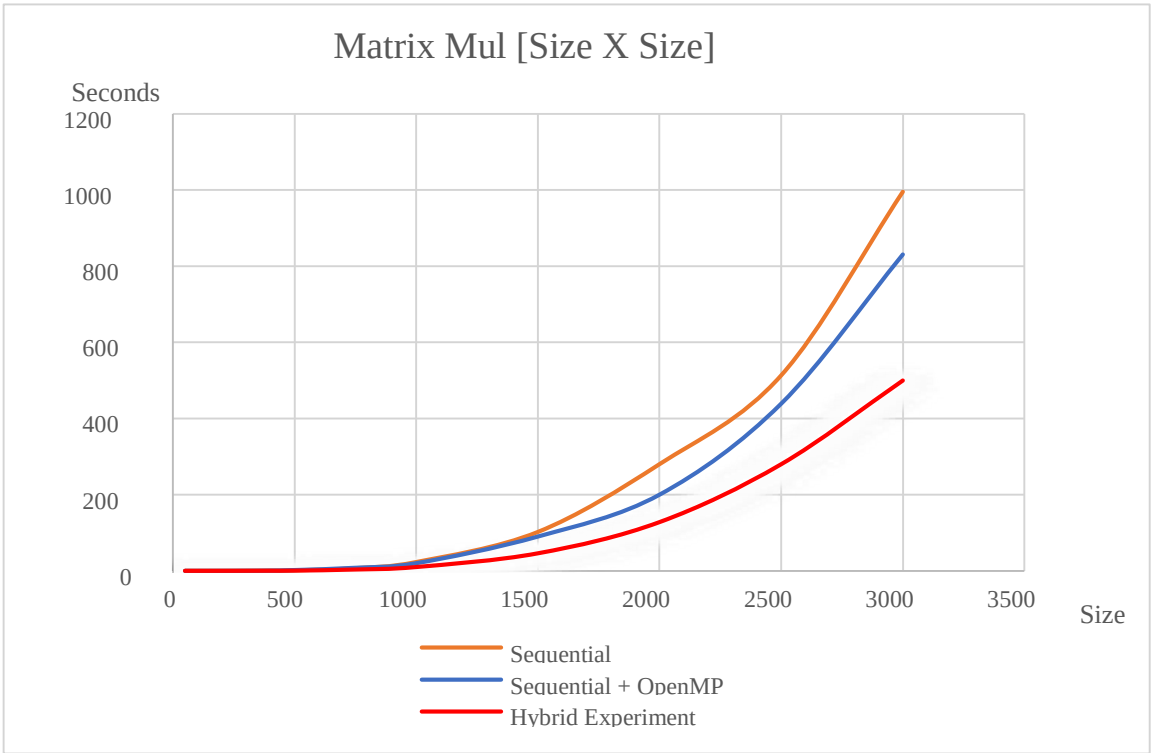
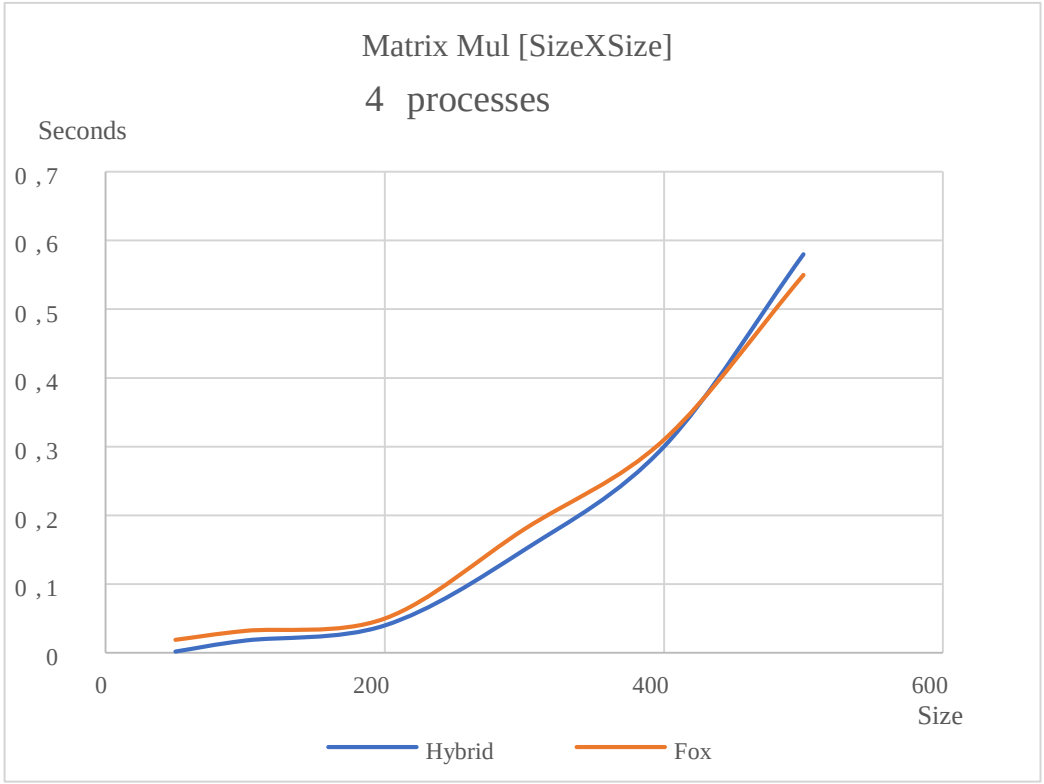


Table 2 and Chart 2 show the difference between the Hybrid experiment and the Fox algorithm on 4 processes.

Table 2:

Size	Hybrid (sec)	Fox (sec)
50x50	0,002	0,019
100x100	0,018	0,032
200x200	0,04	0,05
300x300	0,15	0,18
400x400	0,3	0,31
500x500	0,56	0,55
750x750	3,5	2,9
1000x1000	10	8
1500x1500	46	40
2000x200	128	113
2500x2500	280	250

Chart 2:



8. Conclusions and Future Work

As it can be observed, the time difference between the Hybrid experiment and the sequential algorithm with OpenMP is large.

As you can see, there is a difference between the Hybrid experiment and the Fox algorithm. The hybrid experiment runs faster up to 500x500 matrices. After 500x500, the Fox algorithm runs faster.

The hybrid experiment runs faster up to 500 x 500 matrices, because one of the matrices was shared to all processes. when the matrix size is not large, the matrix shared quickly, so time is faster than Fox algorithm time.

For the future, it is planned to improve the efficiency of the hybrid experiment, to make the code faster than the fox algorithm for matrices over 500x500.

References

[1] J. Ali and R. Zaman, *Performance Analysis of Matrix Multiplication Algorithms Using MPI*: Khan Department of Computer Science, Aligarh Muslim University, Aligarh. Parallel Programming in C with MPI and OpenMP – by Michael J. Quinn.
[2] Parallel Programming in C with MPI and OpenMP – by Michael J. Quinn.
[3] [Online]. Available: https://www.open-mpi.org/
[4] [Online]. Available: https://www.mpich.org/
[5] [Online]. Available: https://www.openmp.org/
[6] [Online]. Available: https://en.wikipedia.org/wiki/Message_P assing_Interface
[7] [Online]. Available: https://en.wikipedia.org/wiki/OpenMP
[8] Ned Nedialkov, *Communicators and Topologies: Matrix Multiplication Example*, McMaster University Canada CS/SE 4F03 March 2016.
[9] [Online]. Available: http://pages.tacc.utexas.edu/~eijkhout/pcse/html/mpi-topo.html

Submitted 22.05.2018, accepted 20.11.2018.

Մատրիցների բազմապատկման արտադրողականության վերլուծությունը օգտագործելով OpenMP և MPI

Տ. Գալստյան

Ամփոփում

OpenMP և MPI համադրությունը ծրագրավորման մեջ անվանում են հիբրիդային ծրագրավորում: Հիբրիդային ծրագրավորումը (հաղորդագրությունների և ընդհանուր հիշողությունների միջոցով) ձեռք է բերել մեծ համբավ կլաստերային ճարտարապետության հայտնվելուց հետո: Հիբրիդային ծրագրավորումը միավորում է OpenMP և MPI գրադարանները բազմամիջուկային ճարտարապետություններում

օգտագործելու համար։Այս աշխատանքի նպատակն է դուրս բերել մատրիցների բազմապատկման արտադրողականության վերլուծությունը կլաստերային համակարգում։ Յուրաքանչյուր հանգույց կլաստերում կազմված է մի քանի կենտրոնական պրոցեսորներից, որոնցում բաժանվում է հիշողությունը հանգույցների միջև և համատեղ հիշողության։ Ալգորիթմը օգտագործում է MPI-ը՝ հաղորդագրություններ ուղարկելու համար և OpenMP՝ հիշողության բաժանման համար։

Анализ производительности матричных алгоритмов умножения с использованием MPI и OpenMP

Т. Галстян

Аннотация

Комбинация OpenMP и MPI в программировании называется гибридным программированием. Гибридное программирование (посредством сообщений и разделяемой памяти) приобрело большую роль с момента появления кластерных архитектур. Метод гибридного программирования объединяет библиотеки MPI и OpenMP для использования этой иерархической многоядерной архитектуры. Цель этой работы - выполнить алгоритмы умножения матрицы анализа производительности в кластерной системе. Каждый узел кластера состоит из нескольких центральных процессоров, в которых память распределена между узлами и разделяемой памятью. Алгоритмы используют MPI в качестве механизма передачи сообщений и OpenMP в качестве совместно используемой памяти.