ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA THÀNH PHỐ HCM
KHOA KHOA HỌC VÀ KỸ THUẬT MÁY TÍNH

TÍNH TOÁN SONG SONG (CO3067)

Project Final

Nhân ma trận trên nhiều máy tính dùng MPI One-sided Communication

Lecturer: Thoại Nam
Student performs: Bùi Hữu Đang – 1811828
Nguyễn Hải Đăng – 1811912
Lê Trung Hiếu – 1812167
Lê Thị Hà – 1812020

Hồ Chí Minh, Tháng 6/2021

Mục lục

1 Bài toán nhân ma trận

Ma trận và các phép toán liên quan tới nó là một phần rất quan trọng trong hầu hết mọi thuật toán liên quan đến số học.

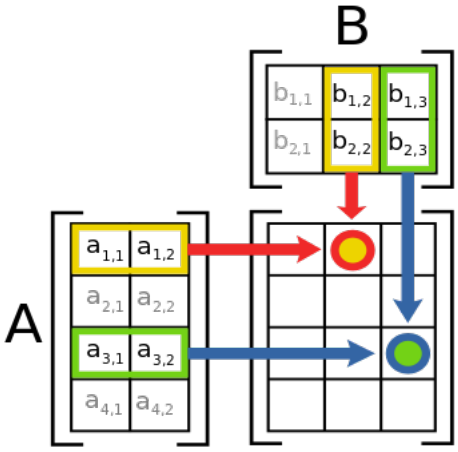
1.1 Định nghĩa phép Nhân ma trận

Nhắc lại một chút kiến thức về toán học về phương pháp nhân 2 ma trận A và B. Điều kiện đầu tiên để có thể thực hiện phép nhân này là khi số cột của ma trận A bằng với số hàng của ma trận B.

Với A là một ma trận có kích thước $n \times m$ và B là một ma trận có kích thước $m \times p$ thì tích của $A \times B$ sẽ là một ma trận $n \times p$ được tính bằng cách sau:

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} \text{ax} + \text{by} \\ \text{cx} + \text{dy} \end{pmatrix}$$

Hình sau mô tả cách tính một phần tử AB[i][j] của ma trận tích:



Hình 1: *Mô tả cách tính một phần tử AB[i][j] của ma trận tích*

Một phần tử là tổng của phép nhân các phần tử trong một hàng của ma trận A với các phần tử trong cột tương ứng trong ma trận B

$$[AB]_{i,j} = A_{i,1}B_{1,j} + A_{i,2}B_{2,j} + ... + A_{i,n}B_{n,j}$$

Hay viết cho gọn như sau:

$$[AB]_{i,j} = \sum_{r=1}^{n} A_{i,r}B_{r,j}$$

1.2 Các giải thuật nhân ma trận

1.2.1 The naive algorithm

Naive Algorithm là từ dùng để chỉ một thuật toán đơn giản nhất được suy luận một cách "ngây thơ" bằng cách xử lý thông thường, ví dụ như tìm kiếm tuần tự (sequential/linear search)

Trong trường hợp này, chúng ta thường implement thuật toán nhân ma trận bằng cách áp dụng chính xác công thức từ định nghĩa toán học của nó, sử dụng vòng lặp, như sau:

Input: Hai ma trận A kích thước $n \times m$ và B kích thước $m \times p$

1: Khởi tạo ma trận C có kích thước $n \times p$
2: For i từ $1 \rightarrow n$:
3: For j từ $1 \rightarrow p$:
4: Gán $sum = 0$
5: For r từ $1 \rightarrow m$:
6: Gán $sum = sum + A_{i,r} \times B_{r,j}$
7: Gán $C_{i,j} = sum$

Output: Ma trận C kích thước $n \times p$

Hình 2: *Thuật toán naive algorithm*

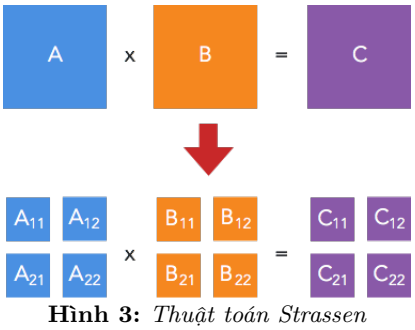
Tại sao lại gọi là naive algorithm (ngây thơ)? đó là vì nó rất dễ implement, chỉ cần đi theo lối suy nghĩ thông thường, bỏ qua hết mọi yếu tố như độ phức tạp, sự tối ưu...

Độ phức tạp của thuật toán trên là $O(nmp)$, trong trường hợp tất cả các ma trận đều là ma trận vuông $n \times n$ thì độ phức tạp của thuật toán sẽ là $O(n^3)$

1.2.2 Thuật toán Strassen

Vào năm 1969, Volker Strassen - lúc đó đang là sinh viên tại MIT - cho rằng $O(n^3)$ chưa phải là con số tối ưu cho phép nhân ma trận, và đề xuất một thuật toán mới có thời gian chạy chỉ nhanh hơn một chút nhưng về sau đã kéo theo rất nhiều nhà khoa học lao vào tiếp tục nghiên cứu và cho đến thời điểm bây giờ, đã có rất nhiều phương pháp mới được đưa ra như là thuật toán Coppersmith-Winograd (sẽ nói ở phần sau), hoặc các giải pháp tiếp cận bằng lập trình song song trên nhiều máy tính/nhiều core,... Điểm thú vị là Strassen nghĩ ra thuật toán này vì nó là bài tập trong một lớp mà ông đang học

Ý tưởng thuật toán của Strassen [3] là áp dụng chia để trị để giải quyết bài toán theo hướng của giải thuật cơ bản trên. Cụ thể là: với mỗi ma trận vuông A, B, C có kích thước $n \times n$, chúng ta chia chúng thành 4 ma trận con, và biểu diễn tích $A \times B = C$ theo các ma trận con đó



Hình 3: *Thuật toán Strassen*

Độ phức tạp của thuật toán Strassen là $O(n^{log7}) \approx O(n^2.807)$

1.3 Nhước điểm các giải thuật tuần tự

Với giải thuật nhân ma trận theo cách tiếp cận brute force chúng ta implement thuật toán nhân ma trận bằng cách áp dụng chính xác công thức từ định nghĩa toán học của nó

2 Giải pháp

Với cách tiếp cận brute force độ phức tạp là $O(n^3)$ và rất không hiệu quả khi n là rất lớn

Tính toán song song đem lại hiệu quả tính toán cao hơn hẳn. Nhiều thư viện hỗ trợ việc tính toán song song đã được giới thiệu để hỗ trợ LTV trong đó có MPI.

2.1 MPI

MPI là một thư viện hỗ trợ tạo ra chương trình song song cũng như truyền dữ liệu giữa các processes trên C, C++ hay Fortrans. Các ngôn ngữ này không hỗ trợ các cấu trúc hay thư viện hỗ trợ lập trình song song. MPI đã thiết kế để giải quyết vấn đề trên bằng cách cung cấp những API đơn giản cho việc tạo ra các processes thực thi song song và trao đổi dữ liệu giữa các processes này.MPI không được thừa nhận bởi bất kỳ tiêu chuẩn nào, tuy nhiên nó lại trở thành một chuẩn thực tế cho giao tiếp giữa các tiến trình để một chương trình song song có thể chạy trên bộ nhớ phân tán.

2.2 Giải thuật nhân ma trận sử dụng MPI Two-sided Communication

Giả sử nhân ma trận A[5 x 5] x B[5 x 5] = C[5 x 5] thực hiện song song với MPI có 6 processes

- Chia làm 1 process master và 5 process workers
- Trên process master: Chia mỗi hàng của Ma trận A cho 5 process worker để thực hiện tính toán trên process worker. Sau khi thực xong thu thập dữ liệu từ process worker và ghi lại kết quả vào ma trận C
- Trên process worker: thực hiện tính toán nhân ma trận [1 x 5] lấy từ process master với ma trận B [5 x 5] và trả kết quả một array 5 phần tử về cho process master

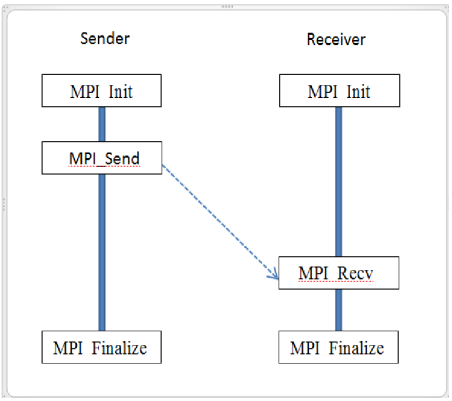
Mã giả:

```
MPI_Init(...);

// Tim process ID cua process dang chay va tong so process
MPI_Comm_rank(...);
MPI_Comm_size(...);

// Neu la process master
if (pid == 0){
    // Phan chia hang cho cac process worker
   offset = 0 // Vi tri cua hang
   for (worker = 1; worker <= numOfWorker; worker++){
        MPI_Send(&offset,...);
        MPI_Send(&matrixA[offset][0],...);
        MPI_Send(&matrixB,...);
```

```
        offset++;
    }

    // Nhan du lieu tu cac process worker
    for (worker = 1; worker <= numOfWorker; worker++){
        MPI_Recv(&offset,...);
        MPI_Recv(&matrixC[offset][0],...);
    }
}

    // Neu la process worker
   else {
        // Nhan du lieu tu process master
      MPI_Recv(&offset,...);
      MPI_Recv(&a,...);
      MPI_Recv(&b,...);

        // Thuc hien tinh toan ma tran

        // Gui du lieu nguoc lai cho process master
      MPI_Send(&offset,...);
      MPI_Send(&c,...);
}

MPI_Finalize();
```

2.3 Giải thuật nhân ma trận sử dụng MPI One-sided Communication

Việc sử dụng MPI two-sided communication (bên Send và bên Recv) như giải thuật ở phần trước có vấn đề. Đó là khi chương trình gửi gọi lệnh *MPI_Send* và chương trình nhận gọi lệnh *MPI_Recv*, dữ liệu ở vũng nhớ chương trình gửi được copy vào một buffer sau đó được gửi lên mạng chung, rồi dữ liệu này mới được copy tới vùng dữ liệu ở chương trình nhận. Vấn đề ở đây là chương trình gửi phải đợi đến khi nào chương trình nhận sẵn sàng để nhận dữ liệu thì dữ liệu mới được gửi. Vấn đề này gây ra gián đoạn trong việc gửi dữ liệu và làm giảm hiệu suất tính toán



Hình 4: *Delay khi dùng MPI Two-sided communication*

Để khắc phục nhược điểm trên, tiêu chuẩn MPI 2 đã giới thiệu Remote Memory Access (RMA) hay còn gọi là One-sided communication vì nó chỉ yêu cầu 1 bên gửi dữ liệu. Hiện nay Intel® MPI Library 5.0 đã hỗ trợ one-sided communication, một process có thể truy cập trực tiếp tới vùng nhớ của process master mà không cần sự can thiệp của process master. Với cách tiếp cận này, khi 1 process gửi dữ liệu tới process master thì process master có thể tiếp tục thi mà không cần phải đợi dữ liệu nhận và ngược lại

Để các process có thể truy cập đến vùng nhớ của một process, process đó phải khai báo rõ ràng vùng nhớ của mình với các process khác. Nó khái bào vùng nhớ chia sẻ (hay còn gọi window) của mình bằng lệnh MPI_Win_create. Việc đồng bộ hóa trong MPI On-sided communication có thể được thực hiện bằng lệnh MPI_Win_fence. Nói một cách đơn giản, giữa lệnh gọi MPI_Win_fence, tất cả lệnh RMA đã được hoàn thành.

Dựa trên những ưu điểm đó ta tối ưu thuật toán với MPI one-sided communication:

Đầu tiên ta sẽ phân 1 process làm master và 5 process còn lại làm worker. Sau đó ta sẽ tạo một cửa sổ chia sẻ vùng nhớ chung cho process master và lưu trữ ma trận B ở đó. Tiếp theo là lấy từng hàng của ma trận A phân chia cho các process worker và lưu trữ ở vùng nhớ của nó.

Ta sẽ thực hiện một hàm chung ở các process worker như sau:

- Bước 1: Tạo một array rỗng có độ dài bằng 5 như trong ví dụ này.
- Bước 2: Lấy một cột của ma trận B từ vùng nhớ chung của process master về.
- Bước 3: Thực hiện nhân hàng và cột ta được phần tử đầu tiên trong array
- Bước 4: Lặp lại bước 2, 3 cho đến khi hoàn thành array.
- Vậy là ta đã tính toán các hàng của array kết quả một cách song song.
- Bước 5: Cập nhật dữ liệu vào vùng nhớ chia sẻ của process master kết quả ma trận.

Mã giả:

```
MPI_Init();

if (rank == 0) // process master
{
    // Tao cua so chia se du lieu chung
    MPI_Win_create();
    // Phan code luu ma tran B va tao ma tran ket qua C
    // Phan code phan phat cac hang cua ma tran A vao cac process worker
}

if (rank != 0) // process worker
{
    for(){
        MPI_Get() // lay tung cot cua ma tran B ve
        //
        // Doan code nhan ma tran
        //
    }
    MPI_Put() // Ghi ket qua vao vung nho chia se o master
}

MPI_Finalize()
```

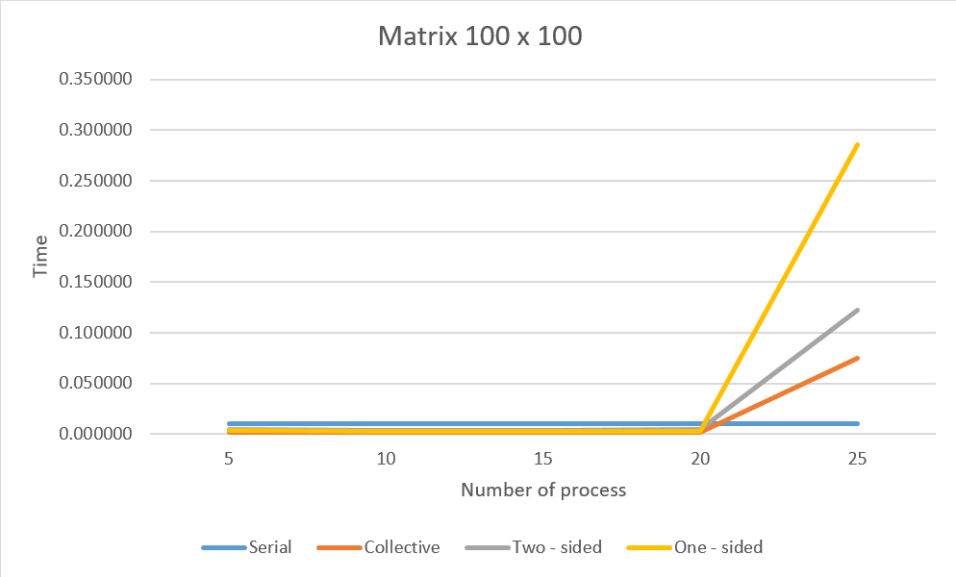
3 Môi trường thử nghiệm và kết quả

3.1 Kết quả chạy chương trình trên một máy có nhiều bộ xử lý

3.1.1 Ma trận 100 x 100

numOfProcess	5	10	15	20	25
Serial	0.010000	0.010000	0.010000	0.010000	0.010000
Collective	0.002126	0.001441	0.001316	0.001470	0.074666
Two - sided	0.004109	0.003488	0.003925	0.004773	0.122415
One - sided	0.004096	0.003044	0.003100	0.002704	0.285884

Hình 5: *số liệu khi nhân ma trận 100 x 100*

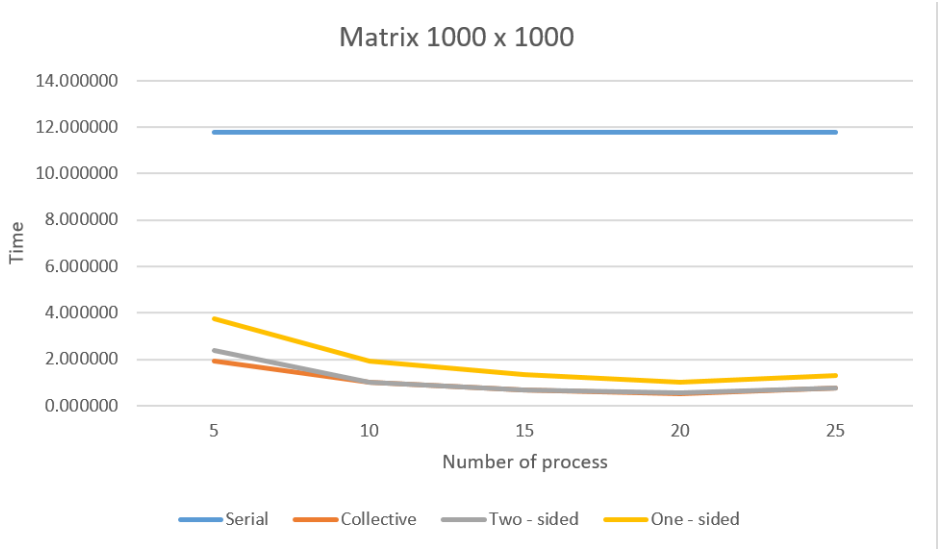


Hình 6: *Biểu đồ khi nhân ma trận 100 x 100*

3.1.2 Ma trận 1000 x 1000

numOfProcess	5	10	15	20	25
Serial	11.776000	11.776000	11.776000	11.776000	11.776000
Collective	1.930635	0.998436	0.698879	0.529267	0.758113
Two - sided	2.375532	1.018068	0.672728	0.543226	0.761338
One - sided	3.770194	1.928992	1.333671	1.031874	1.317034

Hình 7: *số liệu khi nhân ma trận 1000 x 1000*

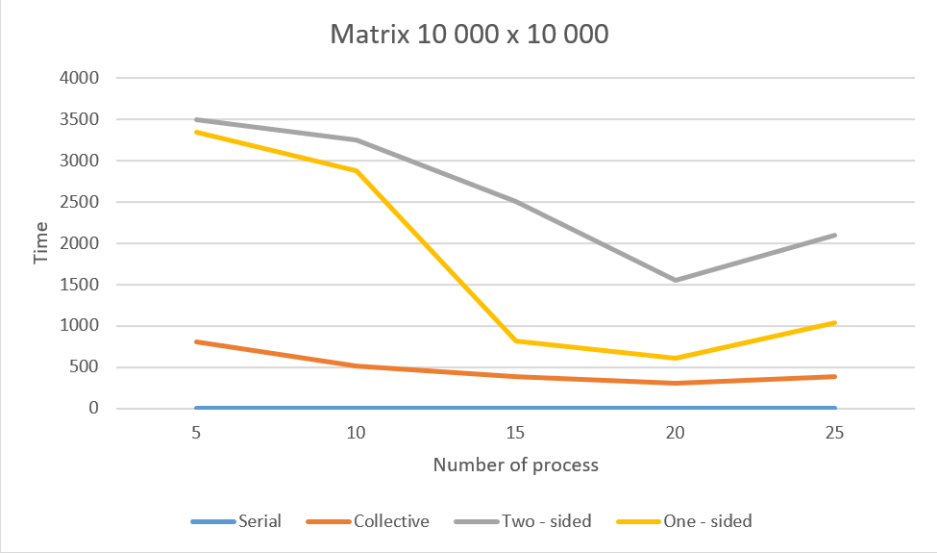


Hình 8: *Biểu đồ khi nhân ma trận 1000 x 1000*

3.1.3 Ma trận 10 000 x 10 000

numOfProcess	5	10	15	20	25
Serial	0	0	0	0	0
Collective	810.225478	515.774565	382.255644	305.589112	382.332596
Two - sided	3495.654756	3250.248895	2504.744226	1550.252112	2095.145226
One - sided	3340.512264	2879.254621	820.421955	608.333671	1042.957461

Hình 9: *số liệu khi nhân ma trận 10 000 x 10 000*



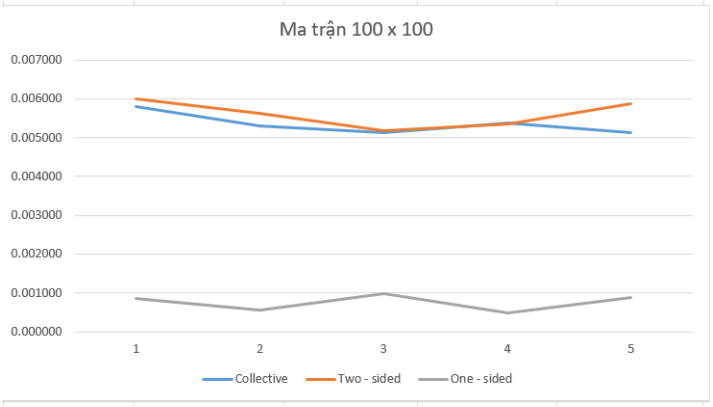
Hình 10: *Biểu đồ khi nhân ma trận 10 000 x 10 000*

3.2 Kết quả chạy chương trình trên hệ thống supernode

3.2.1 Ma trận 100 x 100

Ma trận 100 x 100			
Lần	**Collective**	**Two - sided**	**One - sided**
1	0.005796	0.005995	0.000846
2	0.005314	0.005631	0.000547
3	0.005135	0.005177	0.000986
4	0.005381	0.005345	0.000475
5	0.005135	0.005876	0.000879
Trung bình	**0.005352**	**0.005605**	**0.000747**

Hình 11: *số liệu khi nhân ma trận 100 x 100*

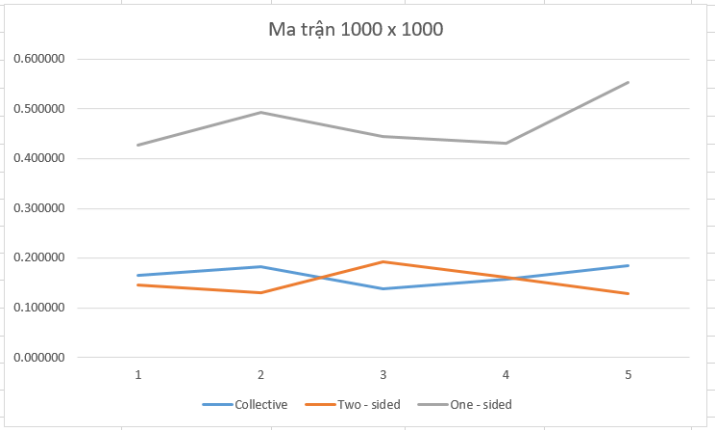


Hình 12: *Biểu đồ khi nhân ma trận 100 x 100*

3.2.2 Ma trận 1000 x 1000

Ma trận 1000 x 1000			
Lần	Collective	Two - sided	One - sided
1	0.165845	0.145683	0.426684
2	0.182369	0.130492	0.493009
3	0.137724	0.193004	0.443761
4	0.157655	0.161269	0.431034
5	0.185241	0.128067	0.553637
Trung bình	**0.165767**	**0.151703**	**0.469625**

Hình 13: *số liệu khi nhân ma trận 1000 x 1000*

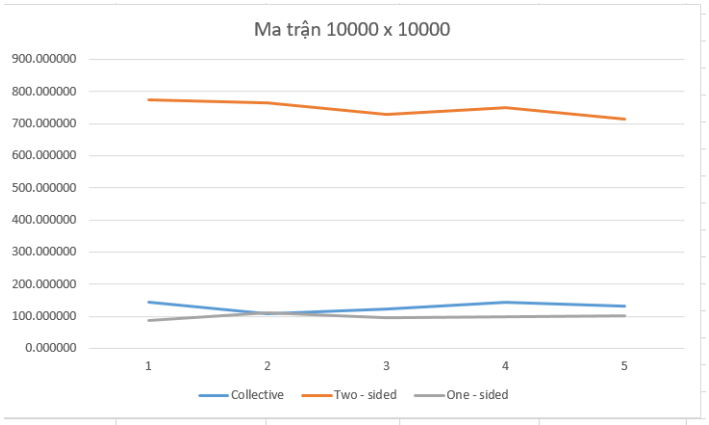


Hình 14: *Biểu đồ khi nhân ma trận 1000 x 1000*

3.2.3 Ma trận 10 000 x 10 000

Ma trận 10000 x 10000			
Lần	Collective	Two - sided	One - sided
1	144.149888	773.225751	87.791158
2	108.297620	764.213835	111.099192
3	122.746879	729.970783	96.762845
4	143.853721	750.567933	97.843322
5	133.134999	713.300923	103.099192
Trung bình	**130.436621**	**746.255845**	**99.319142**

Hình 15: *số liệu khi nhân ma trận 10 000 x 10 000*



Hình 16: *Biểu đồ khi nhân ma trận 10 000 x 10 000*

4 Phân tích đánh giá

Bài toán nhân ma trận đem lại nhiều ứng dụnng rất quan trọng trong cuộc sống, việc nghiên cứu và cải tiến phương pháp giải bài toán nhân ma trận luôn là vấn đề cấp thiết. Qua việc phân tích và thử nghiệm các phương pháp nhân ma trận theo hướng tiếp cận song song hóa cho thấy sự hiểu quả của việc ứng dụng tính toán song song. Hiệu quả của tính toán song song là rất đáng, tuy nhiên cần xem xét kỹ những tình huống thực tế để sử dụng nó, vì có thể thấy nếu ứng dụng cho những bài toán nhỏ có thể sẽ mất thời gian tính toán hơn so với cách tính tuần tự

Tài liệu

[1] thefullsnack.com. (24/05/2021), *Các thuật toán nhân ma trận.* Access from: Các thuật toán nhân ma trận.

[2] software.intel.com. (28/05/2021), *MPI One-Sided Communication.* Access from: MPI One-Sided Communication.

[3] www.mn.uio.no. (28/05/2021), *7 Setting up MPI in compute clusters.* Access from: 7 Setting up MPI in compute clusters.