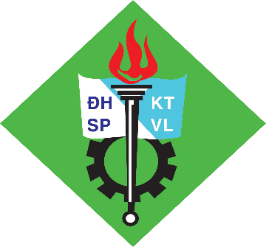
TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT VĨNH LONG

**KHOA CÔNG NGHỆ THÔNG TIN**



**BÁO CÁO ĐỀ TÀI MÔN HỌC**

**KIẾN TRÚC VÀ THUẬT TOÁN SONG SONG**

**NHÂN HAI MA TRẬN VUÔNG CẤP N**

Sinh viên thực hiện:

Nguyễn Tú Toàn 14004096

Nguyễn Bá Tùng 14004103

Trần Minh Huy 14004025

Lớp: ĐH. CÔNG NGHỆ THÔNG TIN 2014

Khóa: 2014-2018

Giáo viên hướng dẫn:Nguyễn Vạn Năng

Vĩnh Long, tháng 06 năm 2018

|  |  |
| --- | --- |
| TRƯỜNG ĐẠI HỌC SPKT VĨNH LONG  **KHOA CÔNG NGHỆ THÔNG TIN** | **CỘNG HÒA XÃ HỘI CHỦ NGHĨA VIỆT NAM**  **Độc lập – Tự do – Hạnh phúc** |

**NHẬN XÉT CỦA NGƯỜI HƯỚNG DẪN KHOA HỌC**

Tên ngành:... Mã ngành:

Trình độ: Khóa học:

Họ và tên sinh viên: MSSV:

Họ và tên sinh viên: MSSV:

Họ và tên sinh viên: MSSV:

CB hướng dẫn: Đơn vị:

Học hàm, học vị:

Nơi công tác:

**Tên đề tài**:

**NỘI DUNG NHẬN XÉT**

**I/ Đánh giá chung về đồ án:**

Hình thức trình bày bản thuyết minh; nội dung thực hiện tốt theo yêu cầu đề tài; nắm vững những vấn đề liên quan đề tài; tính ứng dụng thực tiễn (phạm vi và mức độ ứng dụng, khả năng phát triển, tính mới, tính sáng tạo, độc đáo...)

...............................................................................................................................................

...............................................................................................................................................

**II/ Tinh thần, thái độ của sinh viên:**

Thái độ làm việc và thời gian thực hiện đề tài đúng tiến độ; tự chủ trong việc thực hiện đề tài.

...............................................................................................................................................

...............................................................................................................................................

**III/ Kết luận:**

Đồng ý (hoặc Không đồng ý) cho phép sinh viên được báo cáo đồ án cơ sở ngành.

...............................................................................................................................................

...............................................................................................................................................

|  |  |  |
| --- | --- | --- |
|  |  | *Vĩnh Long, ngày ….. tháng 06 năm 2018*  **Người hướng dẫn**  Nguyễn Vạn Năng |

**TRƯỜNG ĐẠI HỌC SPKT VĨNH LONG**

**KHOA CÔNG NGHỆ THÔNG TIN**

**PHIẾU GIAO ĐỒ ÁN MÔN HỌC**

Tên đồ án: Nhân hai ma trận vuông cấp n

Phương pháp đánh giá:

🞎 Báo cáo trước hội đồng 🞎 Chấm thuyết minh

Ngày giao đồ án: ngày …. tháng 06 năm 2018

Ngày hoàn thành đồ án: ngày …. Tháng 06 năm 2018

Số lượng sinh viên thực hiện đồ án: 3

Họ và tên sinh viên: Nguyễn Tú Toàn MSSV:14004096

Họ và tên sinh viên: Nguyễn Bá Tùng MSSV:14004103

Họ và tên sinh viên: Trần Minh Huy MSSV:14004025

*Vĩnh Long, ngày ... tháng 06 năm 2018*

Trưởng Khoa/ Bộ môn Người hướng dẫn

Nguyễn Vạn Năng

**MỤC LỤC**

[**CHƯƠNG I: TỔNG QUAN VỀ ĐỀ TÀI** 1](#_Toc517917122)

[1.1 Đặt vấn đề 1](#_Toc517917123)

[1.2 Mục đích nghiên cứu 1](#_Toc517917124)

[1.3. Đối tượng nghiên cứu và phạm vi nghiên cứu 1](#_Toc517917125)

[1.3.1 Đối tượng nghiên cứu 1](#_Toc517917126)

[1.3.2 Phạm vi nghiên cứu 2](#_Toc517917127)

[1.4 Cấu trúc tiểu luận 2](#_Toc517917128)

[**CHƯƠNG II: CƠ SỞ LÝ THUYẾT** 4](#_Toc517917129)

[1.1 Giới thiệu 4](#_Toc517917130)

[2.1 Lập trình song song với OpenMP 5](#_Toc517917131)

[2.1.1 Giới thiệu 5](#_Toc517917132)

[2.1.2 Mô hình lập trình song song OpenMP 6](#_Toc517917133)

[2.1.2.1 Song song hóa dựa vào tiến trình 6](#_Toc517917134)

[2.1.2.2 Song song hóa dựa trên cơ chế luồng (Thread based parallelism) 7](#_Toc517917135)

[2.1.3 Chỉ thị trong OpenMP 8](#_Toc517917136)

[2.1.3.1 Khuôn dạng chỉ thị 8](#_Toc517917137)

[2.1.3.2 Phạm vi của chỉ thị 9](#_Toc517917138)

[2.1.4 Thư viện và các biến môi trường 10](#_Toc517917139)

[2.1.4.1 Thư viện Runtime 10](#_Toc517917140)

[2.1.4.2 Biến môi trường 11](#_Toc517917141)

[2.2 Lập trình song song với MPI 11](#_Toc517917142)

[2.2.1 Giới thiệu 11](#_Toc517917143)

[2.2.2 Khái niệm về MPI 11](#_Toc517917144)

[2.2.3 Cấu trúc cơ bản của chương trình MPI 11](#_Toc517917145)

[2.2.4 Các lệnh quản lý môi trường MPI 12](#_Toc517917146)

[**CHƯƠNG III: LẬP TRÌNH THỬ NGHIỆM** 14](#_Toc517917147)

[3.1 Mô tả bài toán 14](#_Toc517917148)

[3.2 Thử nghiệm với Open MP 15](#_Toc517917149)

[3.2.1 Thử nghiệm lập trình theo tiến trình 15](#_Toc517917150)

[3.2.1.1 Chương trình tuần tự 15](#_Toc517917151)

[3.2.1.2 Chương trình song song 18](#_Toc517917152)

[3.2.2 Thử nghiệm lập trình theo luồng 21](#_Toc517917153)

[3.2.2.1 Chương trình tuần tự 21](#_Toc517917154)

[3.2.2.2 Chương trình song song 23](#_Toc517917155)

[3.2.2.3 Biểu đồ so sánh 26](#_Toc517917156)

[3.3 Thử nghiệm với MPI 27](#_Toc517917157)

[3.3.1 Chương trình tuần tự 27](#_Toc517917158)

[3.3.2 Chương trình song song 28](#_Toc517917159)

[**CHƯƠNG IV: KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN** 37](#_Toc517917160)

[4.1 Kết luận 37](#_Toc517917161)

[4.1.1 Kết quả đạt được 37](#_Toc517917162)

[4.1.2 Khuyết điểm 37](#_Toc517917163)

[4.2 Hướng phát triển 37](#_Toc517917164)

[TÀI LIỆU THAM KHẢO 38](#_Toc517917165)

**DANH MỤC HÌNH**

[Hình 2. 1: Kiến trúc OpenMP](#_Toc517256170) 6

[Hình 2. 2: Mô hình Fork-Join](#_Toc517256171) 8

**LỜI CẢM ƠN**

Trong cuộc sống của chúng ta, nếu muốn thành công thì chúng ta phải cố gắng bằng chính nổ lực của bản thân và đôi khi cũng rất cần sự hỗ trợ, giúp đỡ dù ít hay nhiều của những người xung quanh. Trong suốt thời gian từ khi bắt đầu học tập tại trường đến nay, chúng em đã nhận được rất nhiều sự giúp đỡ của quý thầy cô. Với lòng biết ơn sâu sắc, chúng em xin chân thành cám ơn đến tất cả các thầy cô trong trường Đại học Sư Phạm Kỹ Thuật Vĩnh Long nói chung và các thầy cô trong khoa Công Nghệ Thông Tin nói riêng đã hết lòng trong công tác giảng dạy, tận tình truyền đạt kho tàng kiến thức cùng với sự hướng dẫn và giúp đỡ em hết sức nhiệt tình.

Đặc biệt, chúng em chân thành cảm ơn thầy Th.S Nguyễn Vạn Năng đã luôn tạo điều kiện tốt nhất, tận tình hướng dẫn em làm đề tài. Trong suốt thời gian thực hiện thầy đã cho nhóm những lời khuyên bổ ích, cung cấp những thông tin có giá trị, hỗ trợ cho việc lập kế hoạch, phân tích, sử dụng và lập trình song song bằng Open MP và MPI cho bài toán “*Nhân hai ma trận vuông cấp n*”. Thầy đã đề ra cho chúng em các công việc cần xử lý một cách khoa học để hoàn thành đề tài đúng tiến độ, đáp ứng được các yêu cầu đặt ra.

Cuối cùng, em xin gửi lời cảm ơn đến gia đình, bạn bè, người thân đã động viên, giúp đỡ em rất nhiều trong quá trình học tập và làm đề tài môn học “Kiến trúc và thuật toán songsong”. Mặc dù em đã cố gắng để có thể hoàn thành đề tài một cách hoàn chỉnh nhất nhưng vẫn còn nhiều thiếu sót mong nhận được sự đóng góp ý kiến của các thầy cô và các bạn để đề tài càng hoàn thiện hơn.

Em xin chân thành cảm ơn!

Vĩnh Long, ngày ..... tháng 06 năm 2018

Sinh viên thực hiện:

Nguyễn Bá Tùng

Nguyễn Tú Toàn

Trần Minh Huy

**LỜI NÓI ĐẦU**

Hiện nay, ngành khoa học máy tính đang bước vào thời kỳ công nghệ tri thức ở mức trừu tượng cao hơn, làm máy tính trở nên thông minh hơn, nhằm giúp con người giải quyết nhiều vấn đề phức tạp trong thực tiễn. Những đòi hỏi này đã đặt ngành khoa học máy tính vào giải quyết các vấn đề lớn, phức tạp, đòi hỏi dung lượng bộ nhớ phải lớn, cần nhiều thời gian tính toán để hoàn thành. Một chương trình tuần tự trên một máy có bộ xử lý đơn, mất rất nhìu thời gian thậm chí là không thể hoàn thành việc xử lý các vấn đề phức tạp này. Trong khi, chương trình song song sử dụng đồng thời nhiều bộ xử lý để giải quyết một vấn đề, đã trở thành mô hình thống trị, cho phép tiết kiệm thời gian, chi phí, để xử lý các vấn đề lớn phức tạp. Vì vậy, nên nhóm em đã chọn đề tài “*Nhân hai ma trận vuông cấp n* ” để làm bài báo cáo đồ án môn học “*Kiến trúc và thuật toán song song”*

Vì kiến thức còn hạn chế nên còn nhiều thiếu sót trong đề tài này, kính mong quý thầy cô góp ý để đề tài được hoàn thiện hơn. Em xin chân thành cám ơn.

Vĩnh Long, ngày .... tháng 06 năm 2018

**CHƯƠNG I: TỔNG QUAN VỀ ĐỀ TÀI**

**1.1 Đặt vấn đề**

Hiện nay, công nghệ thông tin đang bước vào một thời kỳ công nghệ tri thức ở mức trừu tượng cao, con người giờ đây cần máy tính thông minh hơn, giúp con người giải quyết nhiều vấn đề phức tạp hơn trong thực tiễn.

Những đòi hỏi như thế đã đặt ra một thách thức lớn, đặt ngành khoa học máy tính vào việc giải quyết các vấn đề lớn như dung lượng bộ nhớ phải lớn, thời gian tính toán phải nhiều,…

Giờ đây, một chương trình tuần tự trên một máy tính có bộ xử lý đơn gây mất rất nhiều thời gian hoặc đôi khi không thể hoàn thành được việc xử lý các công việc phúc tạp đó. Trong khi đó, chương trình song song cho phép ta xử dụng nhiều bộ xử lý đồng thời để cùng giải quyết một vấn đề phức tạp, cho phép tiết kiệm thời gian và chi phí. Mô hình này dần thay thế cho mô hình tuần tự đã lỗi thời.

Dựa trên nhu cầu thực tế đó, việc tìm hiểu và nghiên cứu về lập trình song song với hy vọng có thể sử dụng được chúng để lập trình song song bằng thư viện Open MP và thư viện MPI cho bài toán “*Nhân hai ma trận vuông cấp n*” nhằm giảm thời gian tính toán của chương trình, tận dụng các bộ xử lý ở mức tối đa nhất. Chương trình song song này sẽ là bước đệm để ta có thể thực hiện việc lập trình song song cho các bài toán phức tạp hơn trong thực tế.

**1.2 Mục đích nghiên cứu**

Mục đích của nghiên cứu này là cung cấp thêm cho người lập trình một mô hình lập trình có thể tận dụng tối ta các bộ xử lý, tiết kiệm chi phí và thời gian.

Nghiên cứu này sẽ là bước đầu để áp dụng mô hình lập trình song song vào các bài toán cụ thể trong thực tế, các chương trình khác thay vì sử dụng mô hình lập trình tuần tự như những kiến thức đã học trước đây.

**1.3. Đối tượng nghiên cứu và phạm vi nghiên cứu**

**1.3.1 Đối tượng nghiên cứu**

Đối tượng nghiên cứu ở đề tài này bao gồm: kiến trúc máy tính song song, mô hình lập trình song song, bài toán nhân hai ma trận vuông cấp n, thư viện Open Mp và thư viện MPI.

**1.3.2 Phạm vi nghiên cứu**

Phạm vi nghiên cứu đề tài bao gồm:

* Tìm hiểu kiến trúc máy tính song song.
* Tìm hiểu về các loại máy tính song song.
* Tìm hiểu về mô hình lập trình song song.
* Nghiên cứu bộ thư viện Open MP.
* Nghiên cứu giải thuật cho bài toán nhân hai ma trận vuông cấp n.
* Cài đặt và lập trình song song cho bài toán nhân hai ma trận vuông cấp n với thư viện Open MP.
* So sánh lập trình tuần tự và lập trình song song với Open MP cho bài toán nhân hai ma trận.
* Nghiên cứu bộ thư viện MPI.
* Cài đặt và lập trình song song cho bài toán nhân hai ma trận vuông cấp n với thư viện MPI.
* So sánh lập trình tuần tự và lập trình song song với MPI cho bài toán nhân hai ma trận.
  1. **Cấu trúc tiểu luận**

Tiểu luận gồm 4 phần:

* *Chương I: Tổng quan đề tài*

Giới thiệu tổng quan về đề tài “*Nhân hai ma trận vuông cấp n*” và mục đích nghiên cứu, sau đó liệt kê các đối tượng được nghiên cứu và giới hạn phạm vi nghiên cứu của đề tài.

* *Chương II: Cơ sở lý thuyết*

Lý thuyết về lập trình song song với Open MP

* Lập trình song song với Open MP theo tiến trình
* Lập trình song song với Open MP theo luồng

Lý thuyết về lập trình song song với MPI

=> trình bày các phương thức và chỉ thị được sử dụng trong đề tài nhầm làm rỏ ý nghĩa và mục đích của việc nghiên cứu đề tài.

* *Chương III: Lập trình thử nghiệm*

Mô tả việc thực hiện bài toán “*Nhân hai ma trận vuông cấp n*”

Tiến hành lập trình thử nghiệm “*Nhân hai ma trận vuông cấp n*”

* Thử nghiệm với Open MP

Chương trình tuần tự và song song theo tiến trình

Chương trình tuần tự và song song theo luồng

* Thử nghiệm với MPI
* *Chương IV: Kết luận và hướng phát triển*

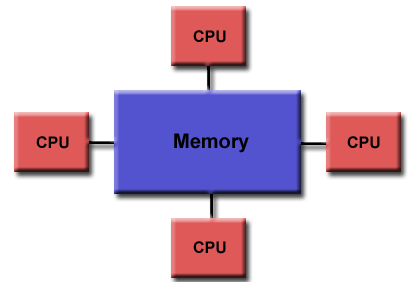
Tổng hợp lại những kết quả đạt được và chưa đạt được trong quá trình nghiên cứu và thực hiện đề tài.

Rút ra kết luận về tốc độ cũng như quá trình xử lý của hai thuật toán tuần tự và song song.

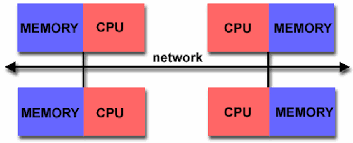
=> đề xuất hướng giải quyết và phát triển về sau của đề tài.

**CHƯƠNG II: CƠ SỞ LÝ THUYẾT**

* 1. **Giới thiệu**
* *Mô hình bộ nhớ chia sẻ*



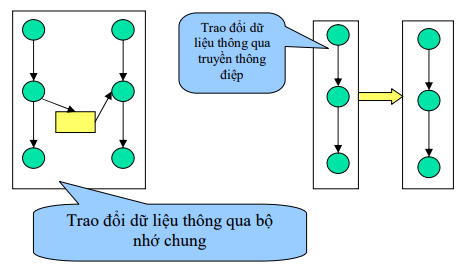
* *Mô hình bộ nhớ phân tán*



* *Các công cụ lập trình song song*

|  |  |  |
| --- | --- | --- |
|  | **Bộ nhớ chung** | **Bộ nhớ phân tán** |
| **Công cụ hệ thống** | Threads (pthread) | Sockets |
| **Công cụ chuyên biệt** | OpenMP  Pthread | MPI  PVM  Globus Toolkit 4 (GT4) |

* *Mô hình trao đổi dữ liệu*



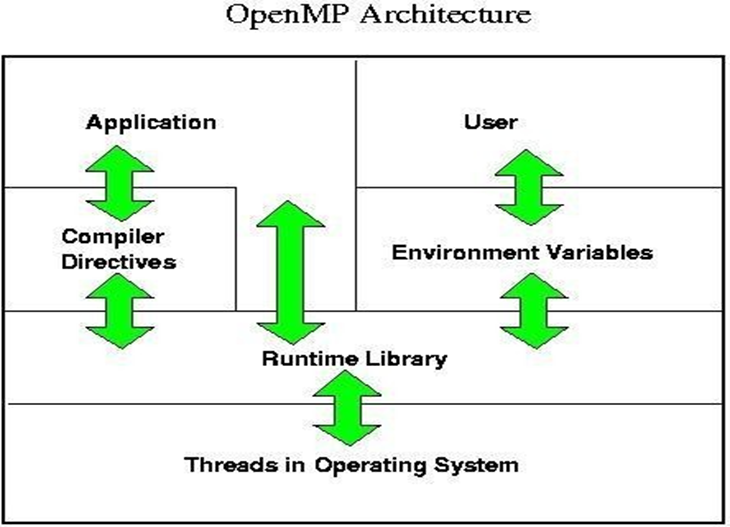
**2.1 Lập trình song song với OpenMP**

**2.1.1 Giới thiệu**

OpenMP (Open Multi-Processing) là một giao diện lập trình ứng dụng API (Application programming interface) hỗ trợ đa nền tảng dựa trên cấu trúc chia sẻ bộ nhớ chung, đa ngôn ngữ lập trình C, C++, Fortran và hầu hết các bộ kiến trúc vi xử lý và hệ điều hành Linux, Unix, Mac OS X, nền tảng Microsoft Windows.

Nó bao gồm:

* Các chỉ thị biên dịch (Compiler directives)
* Các thư viện Runtime (Library rountines)
* Các biến môi trường (Environment variables)



Hình 2.1: Kiến trúc OpenMP

**2.1.2 Mô hình lập trình song song OpenMP**

**2.1.2.1 Song song hóa dựa vào tiến trình**

Yêu cầu đầu tiên của xử lý song song là phải tạo ra được một số các tiến trình cần thiết cho bài toán và khả năng huỷ bỏ chúng khi phần việc xử lý song song kết thúc để giải phóng bộ nhớ và các thiết bị mà các tiến trình đã chiếm giữ. Việc huỷ bỏ các tiến trình phải không cản trở hoạt động của những tiến trình khác.

*VD: Cấu trúc một chương trình có N tiến trình song song*

Tạo N tiến trình:

id = create\_process(N);

=> Ta có N+1 tiến trình (một tiến trình chủ)

***Phân công nhiệm vụ cho các tiến trình:***

id = create\_process(N);

switch(id)

{

case 1: … do NhiemVu1 …(s1); break;

case 2: … do NhiemVu2 …(s2); break;

. . .

case N: … do NhiemVuN …(sn); break;

}

***Thu nhận kết quả tính toán:***

join\_process(N, 0);

// Các lệnh phải chờ

s=0;

For (i=1,n)

s=s+si;

Tiến trình chủ sẽ thu thập kết quả tính toán của các tiến trình khác và thực hiện các công việc còn lại, còn những tiến trình khác kết thúc. Khi đó chúng ta viết join\_process(N, id); id là tiến trình còn tiếp tục hoạt động.

Nếu ta đặt sau nó một số câu lệnh thì:

* Các câu lệnh này sẽ không được thực hiện cho đến khi tất cả các tiến trình đều

thực hiện join\_process().

* Sau đó chỉ còn lại một tiến trình chủ hoạt động.

***Cách thức trao đổi dữ liệu giữa các tiến trình:***

Một mặt một tiến trình có thể muốn giữ một phần dữ liệu cục bộ cho riêng mình, không cho những tiến trình khác nhìn thấy/truy cập tới những dữ liệu đó. Mặt khác, nó cũng muốn trao đổi thông tin với các tiến trình khác. Xử lý vấn đề che giấu hay chia sẻ thông tin như thế nào còn tuỳ thuộc vào mô hình mà chúng ta áp dụng, dựa vào tiến trình hay luồng.

Các tiến trình trong UNIX, WINDOWS được sử dụng như các đơn vị tính toán độc lập. Khi muốn sử dụng bộ nhớ chung, ta cần phải xin cấp phát bộ nhớ và sau khi sử dụng xong phải giải phóng chúng. Người lập trình phải có trách nhiệm giải phóng bộ nhớ chia sẻ một cách tường minh khi chúng không còn cần thiết sử dụng.

* shared(m, &id): cấp phát m byte bộ nhớ chia sẻ cho tiến trình id.

Đối với các luồng, tất cả các thông tin, theo mặc định, là nhìn thấy được. Do vậy, trong mô hình này cần phải cố gắng để che giấu thông tin.

**2.1.2.2 Song song hóa dựa trên cơ chế luồng (Thread based parallelism)**

Các luồng của một tiến trình có thể chia sẻ với nhau về không gian địa chỉ chương trình, các đoạn dữ liệu và môi trường xử lý, đồng thời cũng có vùng dữ liệu riêng để thao tác.

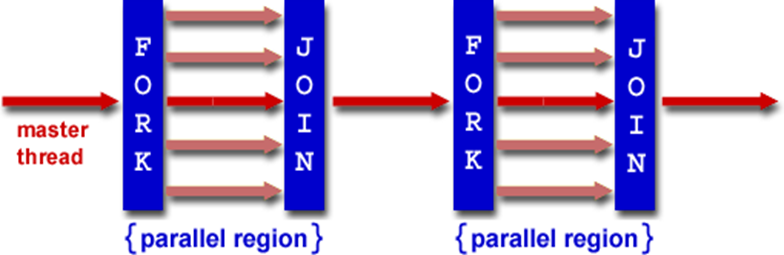
Các tiến trình và các luồng trong hệ thống song song cần phải được đồng bộ, song việc đồng bộ giữa các luồng được thực hiện hiệu quả hơn đổi với các tiến trình. Đồng bộ giữa các tiến trình đòi hỏi tốn thời gian hoạt động của hệ thống, trong khi đối với các luồng thì việc đồng bộ chủ yếu tập trung vào sự truy cập các biến chung (global) của chương trình.

Trong mô hình trên chương trình xử lý trên bộ nhớ toàn cục bao gồm nhiều luồng thực thi đồng thời. OpenMP dựa vào sự tồn tại của nhiều luồng trên một mô hình lập trình chia sẻ bộ nhớ chung.

**Mô hình Fork-Join**

Trong các mô hình trên thì OpenMP sử dụng mô hình Fork-Join để thực thi công

việc song song

****

Hình 2.2: Mô hình Fork-Join

Trong mô hình này tất cả các chương trình song song đều bắt đầu với việc xử lý đơn bởi

một luồng chủ (master thread). Luồng chủ này sẽ thực thi một cách tuần tự cho tới khi bắt gặp vùng song song (parallel region) đầu tiên.

FORK: Có nghĩa là luồng chủ sau đó sẽ tạo ra một tập các luồng song song. Và sau đó đoạn mã trong vùng song song được thực thi song song bởi tập luồng song song vừa tạo ra.

JOIN: Khi mà tập luồng song song đã hoàn thành đoạn mã trong vùng song song chúng sẽ được đồng bộ và kết thúc rồi sau đó công việc lại được thực hiện bởi luồng chủ.

**2.1.3 Chỉ thị trong OpenMP**

**2.1.3.1 Khuôn dạng chỉ thị**

Chỉ thị trong OpenMP được cho dưới dạng sau:

**# pragma omp directive-name [clause...] newline**

* # pragma omp: Yêu cầu bắt buộc đối với mọi chỉ thị OpenMP C/C++
* directive-name: Là tên của chỉ thị phải xuất hiện sau #pragma omp và đứng trước bất kì mệnh đề nào
* [clause...]: Các mệnh đề này không bắt buộc trong chỉ thị newline: Yêu cầu bắt buộc với mỗi chỉ thị nó là tập mã lệnh nằm trong khối cấu trúc được bao bọc bởi chỉ thị

**2.1.3.2 Phạm vi của chỉ thị**

Phạm vi tĩnh (Static Extent): Đó là những đoạn mã nguyên bản trong phạm vi từ đầu đến cuối khối cấu trúc cho sau mỗi chỉ thị. Phạm vi tĩnh của chỉ thị không mở rộng đến các thủ tục và các tệp chứa mã.

Chỉ thị đơn độc (Orphaned Directive): Chỉ thị đơn độc là chỉ thị xuất hiện độc lập với chỉ thị khác. Nó tồn tại ở ngoài phạm vi tĩnh của chỉ thị khác. Chỉ thị đơn độc mở rộng với các thử tục và các tệp mã nguồn

Phạm vi động (Dynamic Extent): Phạm vi động của chỉ thị bao gồm phạm vi tĩnh của của chỉ thị và phạm vi của các chỉ thị mồ côi

OpenMP có rất nhiều chỉ thị như: atomic, barrier, critical, flush, for, master, ordered, parallel, section, single, thread, private.

Các cấu trúc thường gặp:

* Cấu trúc chia sẻ
* Cấu trúc đồng bộ

*a. Cấu trúc chia sẻ*

Cấu trúc chia sẻ công việc dùng để chia việc thực hiện công việc trong vùng song song chocác luồng trong tập các luồng thực hiện công việc cho bởi vùng song song. Cấu trúc chia sẻ công việc phải được bao bọc bởi một vùng song song để có thể thực hiện song song và cấu trúc này có thể được thực hiện bởi tất cả các luồng trong tập các luồng hoặc chỉ một số luồng trong tập các luồng thực thi vùng song song. Có ba loại cấu trúc chia sẻ công việc đó là cấu trúc DO/for, cấu trúc SECTIONS và cấu trúc SINGLE

1. **Chỉ thị Do/For**

Chỉ thị DO/for chỉ ra rằng các công việc lặp đi lặp lại (interations) cho bởi vòng lặp phải được các luồng thực hiện một cách song song. Chỉ thị for trong C/C++ được cho dưới dạng sau:

#pragma omp for [*clause*...] *newline* schedule ( *type* [*,chunk\_size*] ) ordered

private ( *list* ) firstprivate ( *list* ) lastprivate ( *list* ) shared ( *list* )

reduction ( *operator : list* ) nowait

1. **Chỉ thị Sections**

Chỉ thị này dùng để chỉ ra các phần mã trong vùng song song chia cho các luồng thực hiện. Trong phạm vi của chỉ thị SECTIONS có các chỉ thị SECTION. Mỗi một SECTION sẽ được thực hiện bởi một luồng trong tập các luồng và các SECTION khác nhau sẽ được thực hiện bởi các luồng khác nhau. Trong C/C++ chi thị SECTIONS được cho dưới dạng sau

#pragma omp sections [*clause..*.] *newline*

private(*list*) firstprivate(*list*) lastprivate(*list*) reduction(*operator:list*) nowait

**2.1.4 Thư viện và các biến môi trường**

**2.1.4.1 Thư viện Runtime**

OpenMP cung cấp một thư viện với rất nhiều các hàm chức năng bao gồm các truy vấn liên quan đến số lượng và chỉ số các luồng, thiết lập số lượng các luồng sử dụng, semaphores, và các hàm thiết lập môi trường thực thi. Trong C/C++ để có thể sử dụng các hàm trên thì phải đính vào file thư viện omp.h.

* OMP\_SET\_NUM\_THREADS
* OMP\_GET\_NUM\_THREADS

**2.1.4.2 Biến môi trường**

Các biến môi trường được dùng để điều khiển sự thực hiện đoạn mã song song.

Bao gồm các biến môi trường sau:

* + OMP\_SCHEDULE
  + OMP\_NUM\_THREADS

**2.2 Lập trình song song với MPI**

**2.2.1 Giới thiệu**

*MPI là một bộ thư viện hỗ trợ cho việc lập trình kiểu truyền thông điệp*. Thư viện MPI bao gồm các thủ tục truyền tin kiểu point-to-point, và các toán hạng chuyển dữ liệu, tính toán và đồng bộ hóa.

*MPI chỉ làm việc trên các chu trình tĩnh*, tức là tất cả các chu trình cần phải được định nghĩa trước khi thực hiện và chúng sẽ thực hiện đồng thời. MPI-2 là phiên bản nâng cấp của MPI, nó có thêm các chức năng có thể đáp ứng cho các chu trình hoạt động kiểu client – server,…

**2.2.2 Khái niệm về MPI**

Thuật ngữ MPI là viết tắt của Message Passing Interface, chỉ một dạng giao thức kết nối của máy tính. Nó nằm trong chuẩn de facto cho kết nối giữa các nút chạy một chương trình song song trên bộ nhớ chia sẻ được phân phối.

Tập MPI thi hành bao gồm một thư viện các thủ tục sao cho có thể gọi được từ các chương trình Fortran, C, C++ hay Ada. Lợi thế của MPI so với các thư viện cũ là nó vừa thuận tiện (vì MPI thực thi cho hầu hết các kiến trúc bộ nhớ phân phối) vừa nhanh (vì mỗi thủ tục được tối ưu hóa cho phần cứng mà nó đang chạy). Thường được so sánh với PVM và có thể kết hợp nó để tạo thành PVMMPI.

**2.2.3 Cấu trúc cơ bản của chương trình MPI**

Cấu trúc cơ bản của một chương trình MPI như sau:



**2.2.4 Các lệnh quản lý môi trường MPI**

Các lệnh này có nhiệm vụ thiết lập môi trường cho các lệnh thực thi MPI, truy vấn chỉ số của tác vụ, các thư viện MPI,...

**MPI\_Init**: khởi động môi trường MPI.

* MPI\_Init (&argc,&argv)
* Init (argc,argv)

**MPI\_Comm\_size**: trả về tổng số tác vụ MPI đang được thực hiện trong communicator (chẳng hạn như trong MPI\_COMM\_WORLD).

* MPI\_Comm\_size (comm,&size)
* Comm::Get\_size()

**MPI\_Comm\_rank**: trả về chỉ số của tác vụ (rank). Ban đầu mỗi tác vụ sẽ được gán cho một số nguyên tố 0 đến (N−1) với N là tổng số tác vụ trong communicator MPI\_COMM\_WORLD.

* MPI\_Comm\_rank (comm,&rank)
* Comm::Get\_rank()

**MPI\_Abort**: kết thúc tất cả các tiến trình MPI.

* MPI\_Abort (comm,errorcode)
* Comm::Abort(errorcode)

**MPI\_Get\_processor\_name**: trả về tên của bộ xử lý.

* MPI\_Get\_processor\_name (&name,&resultlength)
* Get\_processor\_name(&name,resultlen)

**MPI\_Initialized**: trả về giá trị 1 nếu MPI\_Init() đã được gọi, trả về 0 trong trường hợp ngược lại.

* MPI\_Initialized (&flag)
* Initialized (&flag)

**MPI\_Wtime**: trả về thời gian chạy (tính theo giây) của bộ xử lý.

* MPI\_Wtime ()
* Wtime ()

**MPI\_Wtick**: trả về độ phân giải thời gian (tính theo giây) của MPI\_Wtime().

* MPI\_Wtick()
* Wtick ()

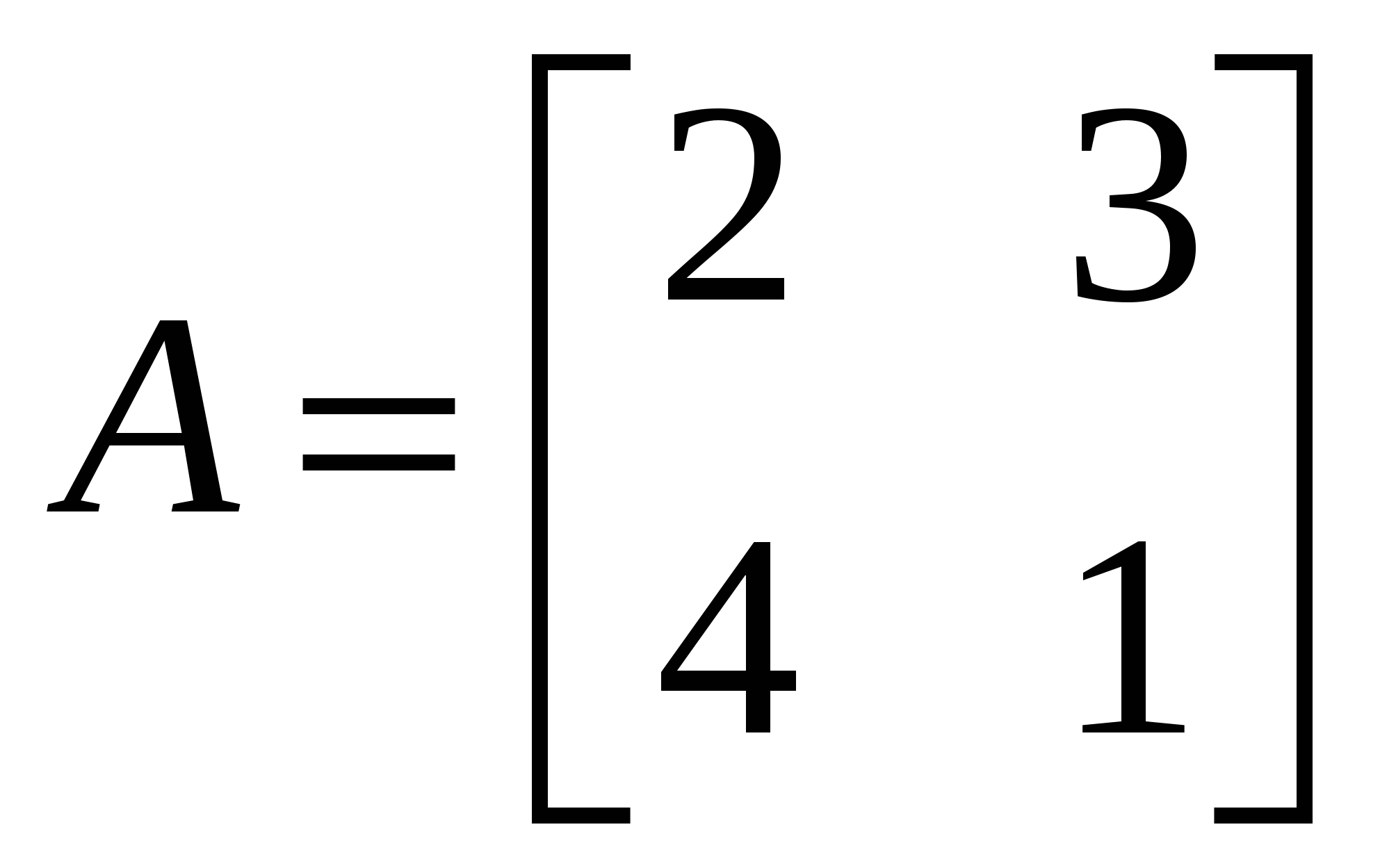
**MPI\_Finalize**: kết thúc môi trường MPI.

* MPI\_Finalize()
* Finalize()

**CHƯƠNG III: LẬP TRÌNH THỬ NGHIỆM**

**3.1 Mô tả bài toán**

Trong toán học, **ma trận** là một mảng các số, ký hiệu hoặc biểu thức, sắp xếp theo hàng và cột *hàng* và *cột*—mà mỗi ma trận tuân theo những quy tắc định trước. Từng ô trong ma trận được gọi là các *phần tử* hoặc *mục*. Ví dụ một ma trận vuông có 2 hàng và 2 cột:{\displaystyle {\begin{bmatrix}1&9&-13\\20&5&-6\end{bmatrix}}.}

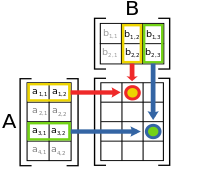


Khi các ma trận có cùng kích thước (chúng có cùng số hàng và cùng số cột), thì có thể thực hiện phép *cộng* hoặc *trừ* hai ma trận trên các phần tử tương ứng của chúng.

=> *Tuy vậy, quy tắc áp dụng cho phép nhân ma trận chỉ có thể thực hiện được khi ma trận thứ nhất có số cột bằng số hàng của ma trận thứ hai.*

**Phép nhân hai ma trận** được xác định khi và chỉ khi số cột của ma trận bên trái bằng số hàng của ma trận bên phải.

Nếu **A** là một ma trận *m*-x-*n* và **B** là một ma trận *n*-x-*p*, thì *ma trận tích* **AB** là ma trận *m*-x-*p* với các phần tử được xác định theo [tích vô hướng](https://vi.wikipedia.org/wiki/T%C3%ADch_v%C3%B4_h%C6%B0%E1%BB%9Bng) của hàng tương ứng trong **A** với cột tương ứng trong **B**.



Minh họa tích hai ma trận A và B{\displaystyle {\begin{aligned}{\begin{bmatrix}{\underline {2}}&{\underline {3}}&{\underline {4}}\\1&0&0\\\end{bmatrix}}{\begin{bmatrix}0&{\underline {1000}}\\1&{\underline {100}}\\0&{\underline {10}}\\\end{bmatrix}}&={\begin{bmatrix}3&{\underline {2340}}\\0&1000\\\end{bmatrix}}.\end{aligned}}}

Phép nhân ma trận thỏa mãn quy tắc :

(**AB**)**C** = **A**(**BC**) => ([tính chất kết hợp](https://vi.wikipedia.org/wiki/K%E1%BA%BFt_h%E1%BB%A3p))

(**A**+**B**)**C** = **AC**+**BC** cũng như  (**A**+**B**)**C** = **CA**+**CB** => ([luật phân phối](https://vi.wikipedia.org/w/index.php?title=T%C3%ADnh_ch%E1%BA%A5t_ph%C3%A2n_ph%E1%BB%91i&action=edit&redlink=1) trái và phải)

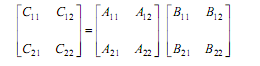
Khi kích thước của các ma trận tham gia vào phép nhân thỏa mãn yêu cầu của tích hai ma trận.[[28]](https://vi.wikipedia.org/wiki/Ma_tr%E1%BA%ADn_(to%C3%A1n_h%E1%BB%8Dc)#cite_note-28) Tích **AB** có thể xác định trong khi **BA** không nhất thiết phải xác định, tức là nếu **A** và **B** lần lượt có số chiều *m*-x-*n* và *n*-x-*k*, và *m* ≠ *k*.

Thậm chí khi cả hai tích này đều tồn tại thì chúng không nhất thiết phải bằng nhau, tức là **AB** ≠ **BA**, hay phép nhân ma trận không có tính [giao hoán](https://vi.wikipedia.org/wiki/Giao_ho%C3%A1n), một đặc điểm khác với các trường số (hữu tỉ, thực, hay phức) mà tích của các số không phụ thuộc vào thứ tự của các số thực hiện trong phép nhân. Ví dụ về nhân hai ma trận không có tính giao hoán:

{\displaystyle {\begin{bmatrix}1&2\\3&4\\\end{bmatrix}}{\begin{bmatrix}0&1\\0&0\\\end{bmatrix}}={\begin{bmatrix}0&1\\0&3\\\end{bmatrix}},}=

Trong khi đó:

{\displaystyle {\begin{bmatrix}0&1\\0&0\\\end{bmatrix}}{\begin{bmatrix}1&2\\3&4\\\end{bmatrix}}={\begin{bmatrix}3&4\\0&0\\\end{bmatrix}}.}=

Dựa vào cách thức tính ma trận như trên, ta có thể tính tích 2 ma trận như sau:

Trong đó :

C11= A11B11 + A12B21

C12= A11B12 + A12B22

C21 = A21B11 + A22B21

C22 = A21B12 + A22B22

**3.2 Thử nghiệm với Open MP**

**3.2.1 Thử nghiệm lập trình theo tiến trình**

**3.2.1.1 Chương trình tuần tự**

//Các thư viện sử dụng và các hàm sử dụng:

#include "stdafx.h"

#include<iostream>

#include<cstdlib>

#include<time.h>

using namespace std;

void initRand()

{

srand(time(NULL));

}

int Random(int min, int max)

{

if (min >= max)

return 0;

int d = max - min;

return (rand() % (d + 1) + min);

}

//Khởi tạo ngẫu nhiên hai ma trận thứ nhất và ma trận thứ hai:

void NhapMaTran(int a[][100], int n)

{

for (int i = 0; i < n; i++)

{

for (int j = 0; j < n; j++)

{

cout << "Nhap phan tu ma tran thu[" << i << "][" << j << "]: ";

cin >> a[i][j];

}

}

}

//In các ma trận:

void XuatMaTran(int a[][100], int n)

{

for (int i = 0; i < n; i++)

{

for (int j = 0; j < n; j++)

{

cout << a[i][j] << "\t";

}

cout << endl << endl;

}

}

//Nhân hai ma trận:

void TichMaTran(int a[][100], int b[][100], int c[][100], int n)

{

for (int i = 0; i < n; i++)

{

for (int j = 0; j < n; j++)

{

c[i][j] = a[i][j] \* b[i][j];

}

}

}

int main() {

clock\_t start, end; //Biến thời gian

//Khai báo các biến sử dụng trong chương trình:

int a[100][100], b[100][100], c[100][100], n;

initRand();

double dif; //Biến qui đổi thời gian ra giây

cout << "Nhap cap cua ma tran n: ";

cin >> n;

cout << "Ma Tran A" << endl;

NhapMaTran(a, n);

cout << "Cac phan tu ma tran A : " << endl;

XuatMaTran(a, n);

cout << "Ma Tran B" << endl;

NhapMaTran(b, n);

cout << "Cac phan tu ma tran B:" << endl;

XuatMaTran(b, n);

start = clock(); //Bắt đầu đếm thời

cout << "Tich Ma Hai Tran" << endl;

TichMaTran(a, b, c, n);

XuatMaTran(c, n);

end = clock(); //Thời gian kết thúc

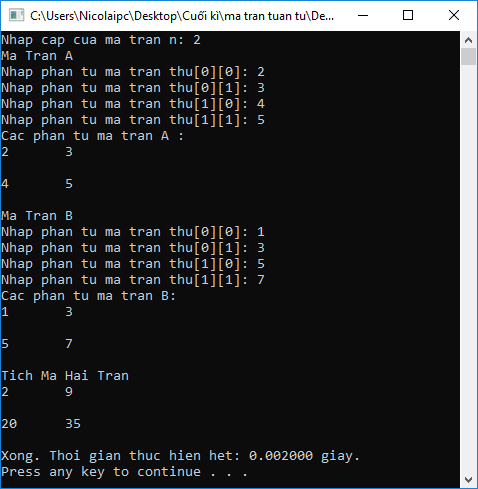
dif = ((double)(end - start)) / CLOCKS\_PER\_SEC; //Qui đổi thời gian ra giây

printf("Xong. Thoi gian thuc hien het: %f giay.\n", dif);

system("pause");

}

* **Kết quả demo**



* **Thời gian thực hiện**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Cấp ma trận** | **Lần 1** | **Lần 2** | **Lần 3** | **Lần 4** | **Lần 5** | **Thời gian trung bình** |
| 10x10 | 0.12 | 0.115 | 0.112 | 0.156 | 0.114 | 0.1234 |
| 50x50 | 6.992 | 4.673 | 3.711 | 3.862 | 3.22 | 4.4916 |
| 80x80 | 6.69 | 7.037 | 6.435 | 6.378 | 6.38 | 6.584 |
| 100x100 | 12.515 | 12.359 | 13.167 | 11.932 | 12.016 | 12.3978 |

* **Độ phức tạp thuật toán**

Độ phức tạp thuật toán sẽ được tính dựa trên đoạn code lập trình phức tạp nhất trong chương trình. Ở đây thuật toán nhân hai ma trận là phức tạp nhất nên độ phức tạp thuật toán sẽ là O(n3)

* **Đánh giá thuật toán**

Thuật toán xử lý khá ổn định đối với kích thước ma trận vừa và nhỏ, tuy nhiên khả năng xử lý của thuật toán chậm dần khi kích thước ma trận tăng, dẫn đến thời gian xử lý tăng theo, tốc độ xử lý ma trận giảm.

=> Chưa đáp ứng được nhu cầu thực tế về tốc độ và thời gian xử lý.

**3.2.1.2 Chương trình song song**

//Các thư viện sử dụng và các hàm sử dụng:

#include<iostream>

#include<cstdlib>

#include<omp.h>

#include <stdio.h>

#include <conio.h>

#include<time.h>

using namespace std;

void initRand()

{

srand(time(NULL));

}

int Random(int min, int max)

{

if (min >= max)

return 0;

int d = max - min;

return (rand() % (d + 1) + min);

}

//Khởi tạo ngẫu nhiên hai ma trận thứ nhất và ma trận thứ hai:

void NhapMaTran(int a[][100], int n)

{

#pragma omp parallel sections

{

#pragma omp section

for (int i = 0; i < n; i++)

{

for (int j = 0; j < n; j++)

{

cout << "Nhap phan tu ma tran thu:[" << i << "][" << j << "]: ";

cin >> a[i][j] ;

}

}

}

}

//In các ma trận:

void XuatMaTran(int a[][100], int n)

{

#pragma omp parallel sections

{

#pragma omp section

for (int i = 0; i < n; i++)

{

for (int j = 0; j < n; j++)

{

cout << a[i][j] << "\t";

}

cout << endl << endl;

}

}

}

//Nhân hai ma trận:

void TichMaTran(int a[][100], int b[][100], int c[][100], int n)

{

#pragma omp parallel sections

{

#pragma omp section

for (int i = 0; i < n; i++)

{

for (int j = 0; j < n; j++)

{

c[i][j] = a[i][j] \* b[i][j];

}

}

}

}

int main() {

initRand();

double dif; //Biến qui đổi thời gian ra giây

omp\_set\_num\_threads(1);

#pragma omp parallel

{

//Khai báo các biến sử dụng trong chương trình:

int a[100][100], b[100][100], c[100][100], n;

cout << "Nhap cap cua ma tran n: ";

cin >> n;

cout << "Ma tran A:" << endl;

NhapMaTran(a, n);

cout << "Cac phan tu ma tran A:" << endl;

XuatMaTran(a, n);

cout << "Ma Tran B" << endl;

NhapMaTran(b, n);

cout << "Cac phan tu ma tran B:" << endl;

XuatMaTran(b, n);

double start = omp\_get\_wtime();

cout << "Tich Hai Ma Tran:" << endl;

TichMaTran(a, b, c, n);

XuatMaTran(c, n);

double end = omp\_get\_wtime(); //Thoi gian ket thuc

dif = end - start; //Khoang thoi gian thuc hien

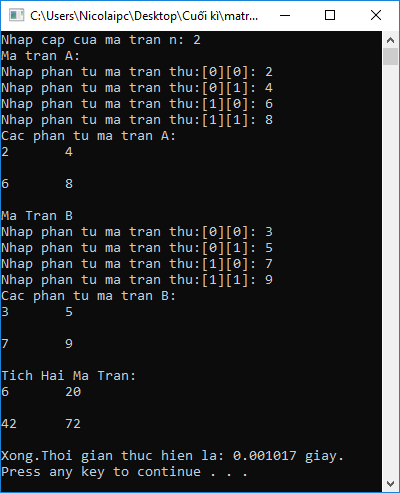
printf("Xong.Thoi gian thuc hien la: %f giay.\n", dif);

system("pause");

}

}

* **Kết quả demo**



* **Thời gian thực hiện**

*Thời gian của 1 bộ xử lý*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Cấp ma trận** | **Lần 1** | **Lần 2** | **Lần 3** | **Lần 4** | **Lần 5** | **Thời gian trung bình** |
| 10x10 | 0.134 | 0.125 | 0.121 | 0.167 | 0.123 | 0.134 |
| 50x50 | 5.67 | 4.897 | 4.711 | 4.962 | 4.557 | 4.9594 |
| 80x80 | 6.985 | 7.137 | 7.485 | 6.598 | 6.874 | 7.0158 |
| 100x100 | 13.215 | 12.654 | 12.849 | 12.932 | 12.454 | 12.8208 |

*Thời gian của 4 bộ xử lý*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Cấp ma trận** | **Lần 1** | **Lần 2** | **Lần 3** | **Lần 4** | **Lần 5** | **Thời gian trung bình** |
| 10x10 | 0.105 | 0.112 | 0.107 | 0.124 | 0.106 | 0.1108 |
| 50x50 | 3.656 | 3.432 | 3.211 | 3.435 | 2.964 | 3.3396 |
| 80x80 | 6.49 | 6.36 | 6.263 | 6.155 | 6.463 | 6.3462 |
| 100x100 | 11.854 | 11.473 | 12.042 | 11.548 | 11.684 | 11.7202 |

* **Độ phức tạp thuật toán**

Độ phức tạp thuật toán sẽ được tính dựa trên đoạn code lập trình phức tạp nhất trong chương trình. Ở đây thuật toán nhân hai ma trận là phức tạp nhất nên độ phức tạp thuật toán sẽ là O(n3)

* **Đánh giá thuật toán**

Thời gian tính toán của chương trình song song trên Open MI là nhanh hơn so với chương trình tuần tự. Khi kích thước ma trận càng lớn thì thời gian tính toán của lập trình song song càng nhanh hơn so với lập trình tuần tự. Tốc độ của lập trình song song lớn gấp 2 lần so với tính toán tuần tự.

**3.2.2 Thử nghiệm lập trình theo luồng**

**3.2.2.1 Chương trình tuần tự**

// nhan2matran.cpp : Defines the entry point for the console application.

//

#include "stdafx.h"

// Matrix Mul without omp.cpp : Defines the entry point for the console application.

//

// Nhan hai ma tran khong su dung OpenMP

#include "StdAfx.h"

#include <stdlib.h>

#include <stdio.h>

#include <conio.h>

#include <time.h>

#include <float.h>

#include <iostream>

//#define N 800 // Kich thuoc ma tran NxN

using namespace std;

int main()

{

cout << "Nhap cap cua ma tran n: ";

int N;

cin >> N;

int \*\*a, \*\*b, \*\*c;

clock\_t start, end; //Bien thoi gian

int i, j, k;

double dif; // Bien qui doi thoi gian ra giay

//Cap phat bo nho cho ma tran A

a = (int \*\*)malloc(10 \* N);

for (i = 0; i<N; i++)

{

a[i] = (int \*)malloc(10 \* N);

}

//Cap phat bo nho cho ma tran B

b = (int \*\*)malloc(10 \* N);

for (i = 0; i<N; i++)

{

b[i] = (int \*)malloc(10 \* N);

}

//Cap phat bo nho cho ma tran tong

c = (int \*\*)malloc(10 \* N);

for (i = 0; i< N; i++)

{

c[i] = (int \*)malloc(10 \* N);

}

printf("Khoi tao ma tran A...\n");

start = clock(); //Bat dau dem thoi gian

//Khoi tao ma tran A

for (i = 0; i<N; i++)

{

for (j = 0; j<N; j++)

{

a[i][j] = i + j;

cout << a[i][j] << "\t";

}

cout << endl;

}

printf("Khoi tao ma tran B...\n");

//Khoi tao ma tran B

for (i = 0; i<N; i++)

{

for (j = 0; j<N; j++)

{

b[i][j] = i \* j;

cout << b[i][j] << "\t";

}

cout << endl;

}

//Khoi tao ma tran C

for (i = 0; i<N; i++)

{

for (j = 0; j< N; j++)

{

c[i][j] = 0;

}

}

printf("Nhan hai ma tran.....\n");

for (i = 0; i<N; i++)

{

for (j = 0; j<N; j++)

{

for (k = 0; k<N; k++)

{

c[i][j] += a[i][k] \* b[k][j];

}

cout << c[i][j] << "\t";

}

cout << endl;

}

end = clock(); //Thoi gian ket thuc

dif = ((double)(end - start)) / CLOCKS\_PER\_SEC; //Qui doi thoi gian ra giay

printf("Xong. Thoi gian thuc hien het: %f giay.\n", dif);

// Tien hanh giai phong bo nho cho cac ma tran da cap phat

for (i = 0; i<N; i++)

{

free(a[i]);

}

free(a);

for (i = 0; i<N; i++)

{

free(b[i]);

}

free(b);

for (i = 0; i<N; i++)

{

free(c[i]);

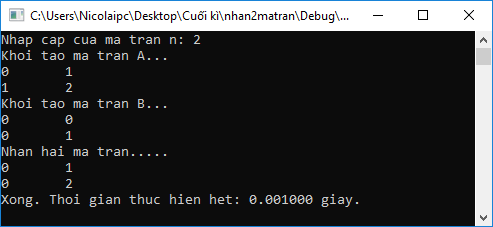
}

free(c);

\_getch();

}

* **Kết quả demo**



*Thời gian tính toán nhân hai ma trận không sử dụng OpenMP*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **TT** | **Kích thước ma trận (NxN)** | **Thời gian chạy lần 1 (Giây)** | **Thời gian chạyLần 2 (Giây)** | **Thời gian chạy lần 3 (Giây)** | **Thời gian chạy trung bình (Giây)** | **Thời gian chạy lâu nhất (Giây)** |
| 1 | 500x500 | 2.226 | 2.222 | 1.965 | 2.138 | 2.226 |
| 2 | 1000x1000 | 23.276 | 22.704 | 22.679 | 22.886 | 23.276 |
| 3 | 1500x1500 | 73.431 | 72.957 | 72.593 | 72.994 | 73.431 |
| 4 | 2000x2000 | 168.976 | 173.586 | 149.34 | 163.967 | 173.586 |
| 5 | 2500x2500 | 347.531 | 352.504 | 346.597 | 348.877 | 352.504 |
| 6 | 3000x3000 | 650.096 | 730.219 | 744.628 | 708.314 | 744.628 |
| 7 | 3500x3500 | 1196.762 | 1072.1 | 1079.67 | 1116.177 | 1196.762 |
| 8 | 4000x4000 | 1277.624 | 1250.586 | 1260.405 | 1262.872 | 1277.624 |

**3.2.2.2 Chương trình song song**

// Nhan hai ma tran su dung OpenMP

#include "stdafx.h"

#include <stdlib.h>

#include <stdio.h>

#include <conio.h>

#include <omp.h>

#include <iostream>

//#define N 200// Kich thuoc ma tran NxN

using namespace std;

int main(int argc, char \*argv[])

{

cout << "Nhap cap cua ma tran n: ";

int N;

cin >> N;

int \*\*a, \*\*b, \*\*c; // Khai báo mảng 2 chiều so nguyen

int tid, nthreads, chunk = 10; //nthreads: bien de luu so threads duoc su dung;tid: Bien de luu so thu tu cua thread

//chunk = 10: Sử dụng để chia trong vong lap For

double dif; //Bien luu thoi gian tinh toan

int i, j, k; // Bien su dung cho vong lap

/\* Cap phat bo nho cho ma tran A \*/

a = (int \*\*)malloc(10 \* N);

for (i = 0; i<N; i++)

{

a[i] = (int \*)malloc(10 \* N);

}

/\* Cap phat bo nho cho ma tran B \*/

b = (int \*\*)malloc(10 \* N);

for (i = 0; i<N; i++)

{

b[i] = (int \*)malloc(10 \* N);

}

/\* Cap phat bo nho cho ma tran C \*/

c = (int \*\*)malloc(10 \* N);

for (i = 0; i< N; i++)

{

c[i] = (int \*)malloc(10 \* N);

}

printf("Khoi tao ma tran...\n");

double start = omp\_get\_wtime(); //Bat dau dem thoi gian

/\*\*\* Tao mot vung song song voi cac bien chia se giua cac thread gom: a,b,c nthread, chunks va cac bien rieng duoc su dung cho cac thread la i,j,k \*\*\*/

#pragma omp parallel shared(a,b,c,nthreads,chunk) private(tid,i,j,k)

{

tid = omp\_get\_thread\_num();

if (tid == 0)

{

nthreads = omp\_get\_num\_threads();// Lay so thread duoc su dung

printf("Bat dau nhan ma tran voi so thread la: %d threads\n", nthreads);

}

//Khoi tao ma tran A

/\* Khai bao mot vong lap duoc thuc hien song song giua cac thread

voi lich trinh Tinh va kich thuoc moi doan la 10\*/

#pragma omp single

printf("Khoi tao ma tran A...\n");

#pragma omp for schedule (static, chunk)

for (i = 0; i<N; i++)

{

for (j = 0; j<N; j++)

{

a[i][j] = i + j;

cout << a[i][j] << "\t";

}

cout << endl;

}

//Khoi tao ma tran B

/\* Khai bao mot vong lap duoc thuc hien song song giua cac thread

voi lich trinh Tinh va kich thuoc moi doan la 10\*/

#pragma omp single

printf("Khoi tao ma tran B...\n");

#pragma omp for schedule (static, chunk)

for (i = 0; i<N; i++)

{

for (j = 0; j<N; j++)

{

b[i][j] = i \* j;

cout << b[i][j] << "\t";

}

cout << endl;

}

//Khoi tao ma tran C

/\* Khai bao mot vong lap duoc thuc hien song song giua cac thread

voi lich trinh Tinh va kich thuoc moi doan la 10\*/

#pragma omp single

printf("Nhan hai ma tran.....\n");

#pragma omp for schedule (static, chunk)

for (i = 0; i<N; i++)

{

for (j = 0; j<N; j++)

{

c[i][j] = 0;

}

}

printf("Thread %d dang tien hanh nhan ma tran...\n", tid);

#pragma omp for schedule (static, chunk)

for (i = 0; i<N; i++)

{

for (j = 0; j<N; j++)

{

for (k = 0; k<N; k++)

{

c[i][j] += a[i][k] \* b[k][j];

}

cout << c[i][j] << "\t";

}

cout << endl;

}

} /\*\*\*\*\*Ket thuc vung song song\*\*\*\*\*/

double end = omp\_get\_wtime(); //Thoi gian ket thuc

dif = end - start; //Khoang thoi gian thuc hien

printf("Xong.Thoi gian thuc hien la: %f giay.\n", dif);

/\*Giai phong bo nho\*/

for (i = 0; i<N; i++)

{

free(a[i]);

}

free(a);

for (i = 0; i<N; i++)

{

free(b[i]);

}

free(b);

for (i = 0; i<N; i++)

{

free(c[i]);

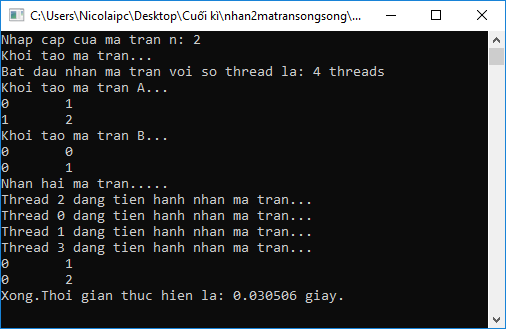
}

free(c);

\_getch();

}

* **Kết quả demo**

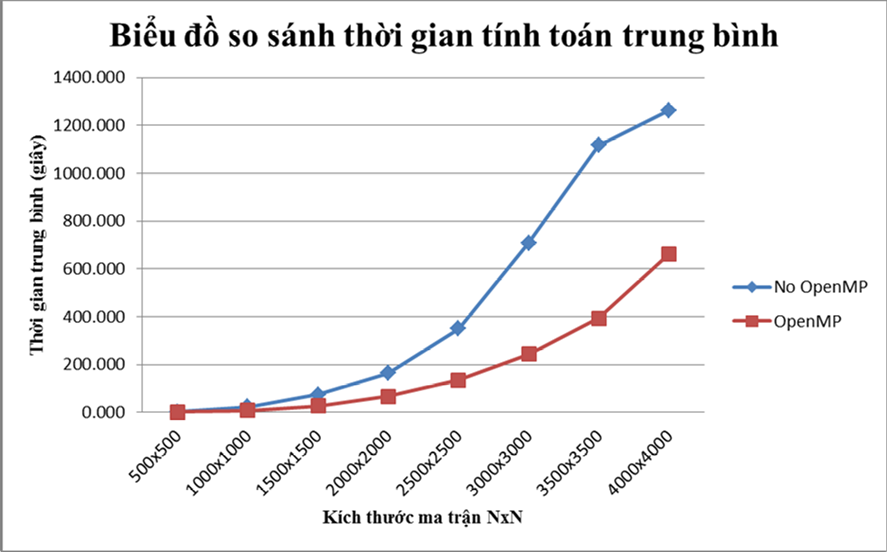


*Thời gian tính toán nhân hai ma trận sử dụng OpenMP*

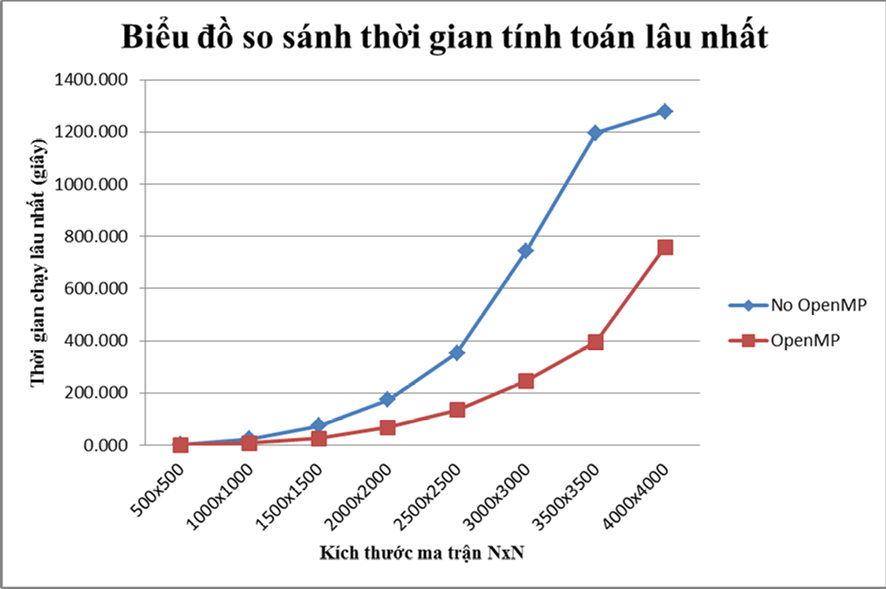
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **TT** | **Kích thước ma trận (NxN)** | **Thời gian chạy lần 1 (Giây)** | **Thời gian chạy lần 2 (Giây)** | **Thời gian chạy lần 3 (Giây)** | **Thời gian chạy trung bình (Giây)** | **Thời gian chạy lâu nhất (Giây)** |
| 1 | 500x500 | 0.905 | 0.850 | 0.819 | 0.858 | 0.905 |
| 2 | 1000x1000 | 7.448 | 7.364 | 7.455 | 7.422 | 7.455 |
| 3 | 1500x1500 | 25.370 | 25.386 | 25.973 | 25.576 | 25.973 |
| 4 | 2000x2000 | 63.488 | 66.801 | 67.333 | 65.874 | 67.333 |
| 5 | 2500x2500 | 134.567 | 134.812 | 134.185 | 134.521 | 134.812 |
| 6 | 3000x3000 | 246.558 | 240.936 | 240.891 | 242.795 | 246.558 |
| 7 | 3500x3500 | 395.057 | 393.168 | 388.070 | 392.098 | 395.057 |
| 8 | 4000x4000 | 566.025 | 665.869 | 757.783 | 663.226 | 757.783 |

**3.2.2.3 Biểu đồ so sánh**

*Biểu đồ so sánh thời gian tính toán trung bình*



*Biểu đồ so sánh thời gian tính toán lâu nhất*



**3.3 Thử nghiệm với MPI**

**3.3.1 Chương trình tuần tự**

#include <stdlib.h>

#include <stdio.h>

#include <time.h>

#include <float.h>

#define N 800 // Kich thuoc ma tran NxN

int main()

{

int \*\*a, \*\*b, \*\*c;

clock\_t start, end; //Bien thoi gian

int i, j, k;

double dif; // Bien qui doi thoi gian ra giay

//Cap phat bo nho cho ma tran A

a = (int \*\*)malloc(10 \* N);

for (i = 0; i<N; i++)

{

a[i] = (int \*)malloc(10 \* N);

}

//Cap phat bo nho cho ma tran B

b = (int \*\*)malloc(10 \* N);

for (i = 0; i<N; i++)

{

b[i] = (int \*)malloc(10 \* N);

}

//Cap phat bo nho cho ma tran tong

c = (int \*\*)malloc(10 \* N);

for (i = 0; i< N; i++)

{

c[i] = (int \*)malloc(10 \* N);

}

printf("Khoi tao ma tran...\n");

start = clock(); //Bat dau dem thoi gian

//Khoi tao ma tran A

for (i = 0; i<N; i++)

{

for (j = 0; j<N; j++)

{

a[i][j] = i + j;

}

}

//Khoi tao ma tran B

for (i = 0; i<N; i++)

{

for (j = 0; j<N; j++)

{

b[i][j] = i \* j;

}

}

//Khoi tao ma tran C

for (i = 0; i<N; i++)

{

for (j = 0; j< N; j++)

{

c[i][j] = 0;

}

}

printf("Nhan hai ma tran.....\n");

for (i = 0; i<N; i++)

{

for (j = 0; j<N; j++)

{

for (k = 0; k<N; k++)

{

c[i][j] += a[i][k] \* b[k][j];

}

}

}

end = clock(); //Thoi gian ket thuc

dif = ((double)(end - start)) / CLOCKS\_PER\_SEC; //Qui doi thoi gian ra giay

printf("Xong. Thoi gian thuc hien het: %f giay.\n", dif);

// Tien hanh giai phong bo nho cho cac ma tran da cap phat

for (i = 0; i<N; i++)

{

free(a[i]);

}

free(a);

for (i = 0; i<N; i++)

{

free(b[i]);

}

free(b);

for (i = 0; i<N; i++)

{

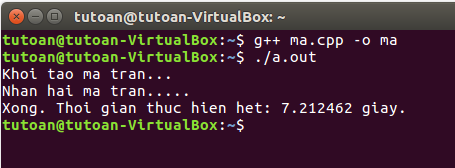
free(c[i]);

}

free(c);

}

* **Kết quả demo**



* **Đánh giá thuật toán**

Thuật toán xử lý khá ổn định đối với kích thước ma trận vừa và nhỏ, tuy nhiên khả năng xử lý của thuật toán chậm dần khi kích thước ma trận tăng, dẫn đến thời gian xử lý tăng theo, tốc độ xử lý ma trận giảm.

**3.3.2 Chương trình song song**

* **matrix\_mul.c**

#include <stdio.h>

#include <stdlib.h>

#include <mpi.h>

#include <assert.h>

#include "matrix\_mul.h"

int main(int argc, char \*argv[])

{

int N;

scanf("%d",&N);

if(argc == 2) {

N = atoi(argv[1]);

if(N <= 0) {

usage(argv[0]);

}

}

/\* The variables below represent the matrices. a and b are the matrices to

\* be multiplied, while c is the result matrix. \*/

long \*\*a = NULL, \*\*b = NULL, \*\*c = NULL;

/\* The variable below is used to initialize the matrix with values. \*/

int mul = 2;

/\* The variables below are used to calculate elapsed time. \*/

double t\_start = 0, t\_end = 0;

#ifdef BARRIERS

double p\_start = 0, p\_end = 0;

#endif

/\* MPI related variables. \*/

MPI\_Status status;

int my\_rank, nproc;

/\* Auxiliary variables. \*/

int averow, count, rows, extra, offset;

/\* Initialize MPI. \*/

MPI\_Init(&argc, &argv);

/\* Get MPI info. \*/

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &my\_rank);

MPI\_Comm\_size(MPI\_COMM\_WORLD, &nproc);

/\* Initialize matrices. \*/

a = init\_long\_matrix(N, N);

b = init\_long\_matrix(N, N);

c = init\_long\_matrix(N, N);

/\* Initializing data. \*/

if(my\_rank == MASTER) {

printf("Kich thuoc ma tran: %d\n", N);

VER("Filling matrices a and b with data.\n");

printf("cac phan tu ma tran A: \n");

for(int i = 0; i < N; ++i) {

for (int j = 0; j < N; ++j) {

a[i][j] = i \* mul;

printf("%li ", a[i][j]);

}

printf("\n");

}

printf("cac phan tu ma tran B: \n");

for(int i = 0; i < N; ++i) {

for (int j = 0; j < N; ++j) {

a[i][j] = i \* mul;

b[i][j] = i;

printf("%li ", b[i][j]);

}

printf("\n");

}

VER("Matrix A %dx%d:\n", N, N);

VER\_matrix(a, N, N);

VER("Matrix B %dx%d:\n", N, N);

VER\_matrix(b, N, N);

printf("MPI\_Wtime() precision(do chinh xac): %lf\n", MPI\_Wtick());

t\_start = MPI\_Wtime();

}

/\* End of init step. Using synchronization barrier. \*/

#ifdef BARRIERS

MPI\_Barrier(MPI\_COMM\_WORLD);

if(my\_rank == MASTER) {

p\_start = MPI\_Wtime();

}

#endif

/\* Sending work to slaves. \*/

if(my\_rank == MASTER) {

averow = N / nproc;

DBG("averow: %d\n", averow);

extra = N % nproc;

DBG("extra: %d\n", extra);

/\* We are skipping the transfer between master and master, since it's silly. \*/

offset = averow;

for(int dest = 1; dest < nproc; ++dest) {

rows = (dest <= extra) ? averow + 1 : averow;

DBG("Master sending to slave %d offset=%d and rows=%d\n", dest, rows, offset);

MPI\_Send(&offset, 1, MPI\_INT, dest, FROM\_MASTER, MPI\_COMM\_WORLD);

MPI\_Send(&rows, 1, MPI\_INT, dest, FROM\_MASTER, MPI\_COMM\_WORLD);

MPI\_Send(&a[offset][0], rows\*N, MPI\_LONG, dest, FROM\_MASTER,

MPI\_COMM\_WORLD);

MPI\_Send(&(b[0][0]), N\*N, MPI\_LONG, dest, FROM\_MASTER, MPI\_COMM\_WORLD);

offset = offset + rows;

}

/\* Afterwards, master should have rows=averows \*/

rows = averow;

} else {

/\* Receiving work from master. \*/

MPI\_Recv(&offset, 1, MPI\_INT, MASTER, FROM\_MASTER, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

MPI\_Recv(&rows, 1, MPI\_INT, MASTER, FROM\_MASTER, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

DBG("Slave %d received offset=%d and rows=%d\n", my\_rank, offset, rows);

MPI\_Recv(&a[0][0], rows\*N, MPI\_LONG, MASTER, FROM\_MASTER, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

MPI\_Get\_count(&status, MPI\_LONG, &count);

DBG("Slave %d A matrix status: MPI\_Get\_count=%d MPI\_SOURCE=%d MPI\_TAG=%d MPI\_ERROR=%d\n",

my\_rank, count, status.MPI\_SOURCE, status.MPI\_TAG, status.MPI\_ERROR);

VER("Slave %d received matrix A:\n", my\_rank);

VER\_matrix(a, N, N);

MPI\_Recv(&(b[0][0]), N\*N, MPI\_LONG, MASTER, FROM\_MASTER, MPI\_COMM\_WORLD, &status);

MPI\_Get\_count(&status, MPI\_LONG, &count);

DBG("Slave %d B matrix status: MPI\_Get\_count=%d MPI\_SOURCE=%d MPI\_TAG=%d MPI\_ERROR=%d\n",

my\_rank, count, status.MPI\_SOURCE, status.MPI\_TAG, status.MPI\_ERROR);

VER("Slave %d received matrix B:\n", my\_rank);

VER\_matrix(b, N, N);

}

/\* End of communication step. Using synchronization barrier. \*/

#ifdef BARRIERS

MPI\_Barrier(MPI\_COMM\_WORLD);

if(my\_rank == MASTER) {

p\_end = MPI\_Wtime();

printf("Thoi gian cua 1 lan xu ly: %lf\n", p\_end - p\_start);

p\_start = MPI\_Wtime();

}

#endif

#ifdef DEBUG

/\* The idea of this variable is to calculate the number of calculations

\* that each process does. \*/

long calcs = 0;

#endif

/\* Compute matrix multiplication \*/

printf("Tich 2 ma tran: \n");

for(int k = 0; k < N; ++k) {

for(int i = 0; i < rows; ++i) {

for(int j = 0; j < N; ++j) {

c[i][k] += a[i][j] \* b[j][k];

VER("Process %d calculated c[%d][%d]=%ld\n", my\_rank, i, k, c[i][k]);

#ifdef DEBUG

calcs++;

#endif

}

printf("%li ", c[i][k]);

}

printf("\n");

}

DBG("Processor %d realized %ld calculations!\n", my\_rank, calcs);

VER("Process %d computing partial C matrix:\n", my\_rank);

VER\_matrix(c, N, N);

/\* End of computation step. Using synchronization barrier. \*/

#ifdef BARRIERS

MPI\_Barrier(MPI\_COMM\_WORLD);

if(my\_rank == MASTER) {

p\_end = MPI\_Wtime();

printf("Tinh thoi gian xu ly: %lf\n", p\_end - p\_start);

p\_start = MPI\_Wtime();

}

#endif

/\* Send results to master. \*/

if(my\_rank != MASTER) {

MPI\_Send(&offset, 1, MPI\_INT, MASTER, FROM\_WORKER, MPI\_COMM\_WORLD);

MPI\_Send(&rows, 1, MPI\_INT, MASTER, FROM\_WORKER, MPI\_COMM\_WORLD);

MPI\_Send(&c[0][0], rows\*N, MPI\_LONG, MASTER, FROM\_WORKER, MPI\_COMM\_WORLD);

}

/\* Receive results from worker tasks. \*/

if (my\_rank == MASTER) {

for (int source = 1; source < nproc; ++source) {

MPI\_Recv(&offset, 1, MPI\_INT, source, FROM\_WORKER, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

MPI\_Recv(&rows, 1, MPI\_INT, source, FROM\_WORKER, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

MPI\_Recv(&c[offset][0], rows\*N, MPI\_LONG, source, FROM\_WORKER,

MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

DBG("Received results from task %d\n", source);

}

VER("Resulting matrix C:\n");

VER\_matrix(c, N, N);

}

/\* End of communication step. Using synchronization barrier. \*/

#ifdef BARRIERS

MPI\_Barrier(MPI\_COMM\_WORLD);

if(my\_rank == MASTER) {

p\_end = MPI\_Wtime();

printf("Thoi gian cua 2 lan xu ly: %lf\n", p\_end - p\_start);

p\_start = MPI\_Wtime();

}

#endif

if(my\_rank == MASTER) {

t\_end = MPI\_Wtime();

printf("Tong thoi gian xu ly: %lf\n", t\_end - t\_start);

}

#ifdef ASSERT

long col\_sum = N \* (N-1) / 2;

if(my\_rank == MASTER) {

for (int i = 0; i < N; ++i) {

for (int j = 0; j < N; ++j) {

assert(c[i][j] == i \* mul \* col\_sum);

}

}

printf("Matrix checking sucessful!\n");

}

#endif

MPI\_Finalize();

exit(EXIT\_SUCCESS);

}

long \*\*init\_long\_matrix(int rows, int cols)

{

long \*data = (long \*) calloc(rows\*cols, sizeof(long));

long \*\*array= (long \*\*) calloc(rows, sizeof(long\*));

if(data == NULL || array == NULL) {

fprintf(stderr, "Could not allocate sufficient memory!");

exit(EXIT\_FAILURE);

}

for (int i = 0; i < rows; ++i) {

array[i] = &(data[cols\*i]);

}

return array;

}

void fprintf\_matrix(FILE \*stream, long\*\* matrix, int rows, int cols)

{

for(int i = 0; i < rows; ++i) {

for(int j = 0; j < cols; ++j) {

if(j == cols - 1) {

fprintf(stream, "%4ld", matrix[i][j]);

} else {

fprintf(stream, "%4ld,", matrix[i][j]);

}

}

fprintf(stream, "\n");

}

fprintf(stream, "\n");

}

void usage(char\* program\_name)

{

fprintf(stderr, "usage: %s MATRIX\_SIZE\n", program\_name);

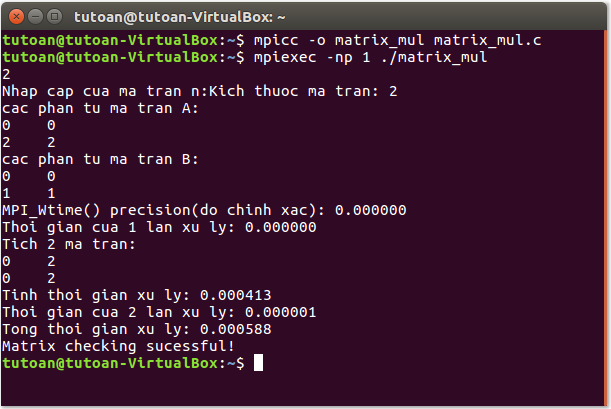
exit(EXIT\_FAILURE);

}

* **matrix\_mul.h**

|  |
| --- |
| #ifndef \_MATRIX\_MUL\_H |
|  | #define \_MATRIX\_MUL\_H |
|  | #define ASSERT |
|  | #define BARRIERS |
|  | //#define DEBUG |
|  | //#define VERBOSE |
|  | #define MASTER 0 |
|  | #define FROM\_MASTER 1 |
|  | #define FROM\_WORKER 2 |
|  |  |
|  |  |
|  | #ifdef VERBOSE |
|  | #ifndef DEBUG |
|  | #define DEBUG |
|  | #endif |
|  | #define VER(...) do { fprintf(stderr, "VERBOSE: "); \ |
|  | fprintf(stderr, \_\_VA\_ARGS\_\_); \ |
|  | fflush(stderr); } while(0) |
|  | #define VER\_matrix(...) do { fprintf(stderr, "VERBOSE: printing matrix\n"); \ |
|  | fprintf\_matrix(stderr, \_\_VA\_ARGS\_\_); \ |
|  | fflush(stderr); } while(0) |
|  | #else |
|  | #define VER(...) do {} while(0) |
|  | #define VER\_matrix(...) do {} while(0) |
|  | #endif |
|  |  |
|  | #ifdef DEBUG |
|  | #define DBG(...) do { fprintf(stderr, "DEBUG: "); \ |
|  | fprintf(stderr, \_\_VA\_ARGS\_\_); \ |
|  | fflush(stderr); } while(0) |
|  | #define DBG\_matrix(...) do { fprintf(stderr, "DEBUG: printing matrix\n");\ |
|  | fprintf\_matrix(stderr, \_\_VA\_ARGS\_\_); \ |
|  | fflush(stderr); } while(0) |
|  | #else |
|  | #define DBG(...) do {} while(0) |
|  | #define DBG\_matrix(...) do {} while(0) |
|  | #endif |
|  | #endif |
|  | long \*\*init\_long\_matrix(int rows, int cols); |
|  | void fprintf\_matrix(FILE \*stream, long\*\* matrix, int rows, int cols); |
|  | void usage(char\* program\_name); |
|  | int main(int argc, char \*argv[]) |

* **Kết quả demo**



* **Đánh giá thuật toán**

Thời gian tính toán của lập trình song song là nhanh hơn so với lập trình tuần tự. Khi kích thước ma trận càng lớn thì thời gian tính toán của lập trình song song càng nhanh hơn so với lập trình tuần tự.

Tuy nhiên code xử lý còn khá phức tạp, qui trình hoạt động khá rườm rà gây khó khăn cho người đọc.

**CHƯƠNG IV: KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN**

**4.1 Kết luận**

**4.1.1 Kết quả đạt được**

Qua kết quả thực nghiệm cho thấy: thời gian tính toán của lập trình song song là nhanh hơn so với lập trình tuần tự. Khi kích thước ma trận càng lớn thì thời gian tính toán của lập trình song song càng nhanh hơn so với lập trình tuần tự. Tốc độ của lập trình song song lớn gấp 2 lần so với tính toán tuần tự.

Tìm hiểu được các mô hình lập trình song song, tính toán song song, các mô hình bộ nhớ.

Có thể lập trình song song với Open MP và MPI cho bài toán “*Nhân hai ma trận vuông cấp n”*.

Chứng minh được rằng giải thuật song song cho bài toán “*Nhân hai ma trận vuông cấp n*” hữu hiệu hơn giải thuật tuần tự.

**4.1.2 Khuyết điểm**

Do việc đi sâu vào tiếp cận ứng dụng của OpenMP và MPI còn hạn chế chưa thể khai thác hết các chỉ thị, các hàm thư viện và các biến môi trường nhằm mục đích phân phối nhiệm vụ một cách hợp lý cho các bộ xử lý để song song hiệu quả hơn, tối ưu hơn nữa cho bài toán nhân ma trận này.

**4.2 Hướng phát triển**

Có thể mở rộng giải thuật song song cho nhiều bài toán lớn khác trong thực tế để vừa tăng tốc độ tính toán, vừa tận dụng được khả năng của những hệ thống máy tính song song mạnh mẽ hiện nay.

**TÀI LIỆU THAM KHẢO**

<http://itplus-academy.edu.vn/openmp-trong-c-1584.html>

<https://vi.wikipedia.org/wiki/T%C3%ADnh_to%C3%A1n_song_song>

<https://kipalog.com/posts/Xu-li-song-song>

<https://ctec.tvu.edu.vn/ttkhai/TCC/63_Dinh_thuc.htm>

<https://vi.scribd.com/doc/225989736/C%C6%A1-b%E1%BA%A3n-l%E1%BA%ADp-trinh-song-song-MPI-C-C>

https://tosonnguyen.wordpress.com/2013/10/27/cai-dat-mpich2-tren-linux/