TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM

**TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG**

**KHOA TOÁN – THỐNG KÊ**



**KHÓA LUẬN TỐT NGHIỆP**

**TÌM HIỂU MỘT SỐ GIẢI THUẬT SONG SONG**

*Người hướng dẫn:* ThS**.HUỲNH VĂN KHA**

*Người thực hiện:* **VÕ NGUYỄN LOAN ANH**

*Lớp :* **130C0102**

*Khoá :* **17**

**THÀNH PHỐ HỒ CHÍ MINH, NĂM 2017**

TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM

**TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG**

**KHOA TOÁN – THỐNG KÊ**



**KHÓA LUẬN TỐT NGHIỆP**

**TÌM HIỂU MỘT SỐ GIẢI THUẬT SONG SONG**

*Người hướng dẫn:* ThS**.HUỲNH VĂN KHA**

*Người thực hiện:* **VÕ NGUYỄN LOAN ANH**

*Lớp :* **130C0102**

*Khoá :* **17**

**THÀNH PHỐ HỒ CHÍ MINH, NĂM 2017**

# LỜI CẢM ƠN

Lời đầu tiên, em xin chân thành cảm ơn các Thầy Cô trường Đại Học Tôn Đức Thắng nói chung, và Thầy Cô Khoa Toán-Thống Kê nói riêng, đã quan tâm, dạy dỗ, hướng dẫn em ngay từ những ngày đầu bỡ ngỡ vào trường. Các Thầy Cô đã hết lòng truyền đạt và chỉ bảo, cho em những kiến thức, kinh nghiệm quý báu của mình trong suốt thời gian vừa qua.

Em cũng xin gửi lời cảm ơn sâu sắc đến Thầy HUỲNH VĂN KHA trong Khoa Toán – Thống kê Trường Đại Học Tôn Đức Thắng đã luôn theo sát, tạo điều kiện giúp đỡ, chỉ bảo em cách soạn thảo và trình bày báo cáo, đưa ra những lời góp ý, cách sửa chữa. Nhờ sự chỉ bảo của thầy và sự giúp đỡ của các bạn trong lớp đã giúp em hoàn thành được đề tài này.

Em đã cố gắng tìm hiểu thêm nguồn thông tin khác nhau như là sách, giáo trình, mạng internet,…Nhưng với kiến thức và kinh nghiệm còn hạn chế thì không tránh khỏi những sai sót. Vì vậy, kính mong được sự đóng góp ý kiến của thầy để em có thể hoàn thiện hơn về báo cáo và là cơ sở vững chắc để sau này ra trường làm việc.

Em xin chân thành cảm ơn.

# CÔNG TRÌNH ĐƯỢC HOÀN THÀNH

# TẠI TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG

Tôi xin cam đoan đây là công trình nghiên cứu của riêng tôi và được sự hướng dẫn khoa học của ThS.HUỲNH VĂN KHA. Các nội dung nghiên cứu, kết quả trong đề tài này là trung thực và chưa công bố dưới bất kỳ hình thức nào trước đây. Những số liệu trong các bảng biểu phục vụ cho việc phân tích, nhận xét, đánh giá được chính tác giả thu thập từ các nguồn khác nhau có ghi rõ trong phần tài liệu tham khảo.

Ngoài ra, trong luận văn còn sử dụng một số nhận xét, đánh giá cũng như số liệu của các tác giả khác, cơ quan tổ chức khác đều có trích dẫn và chú thích nguồn gốc.

**Nếu phát hiện có bất kỳ sự gian lận nào tôi xin hoàn toàn chịu trách nhiệm về nội dung luận văn của mình.** Trường đại học Tôn Đức Thắng không liên quan đến những vi phạm tác quyền, bản quyền do tôi gây ra trong quá trình thực hiện (nếu có).

*TP. Hồ Chí Minh, ngày 23 tháng 6 năm 2017*

*Võ Nguyễn Loan Anh*

MỤC LỤC

[LỜI CẢM ƠN i](#_Toc485996453)

[CÔNG TRÌNH ĐƯỢC HOÀN THÀNH ii](#_Toc485996454)

[TẠI TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG ii](#_Toc485996455)

[MỤC LỤC iii](#_Toc485996456)

[DANH MỤC CÁC BẢNG vi](#_Toc485996457)

[DANH MỤC CÁC HÌNH vii](#_Toc485996458)

[TÓM TẮT 1](#_Toc485996459)

[**CHƯƠNG 1. 3**](#_Toc485996460)

[**Giới thiệu chung về lập trình song song. 3**](#_Toc485996461)

[1.1 Lập trình song song là gì? 3](#_Toc485996462)

[1.2 Lý do cần phải sử dụng lập trình song song? 4](#_Toc485996463)

[1.3 Cân bằng tải. 6](#_Toc485996464)

[1.4.1 Mô hình Data-Parallel (mô hình song song dữ liệu). 7](#_Toc485996465)

[1.4.2 Mô hình Task Graph (mô hình đồ thị tác vụ) 8](#_Toc485996466)

[1.4.3 Mô hình Work Pool. 9](#_Toc485996467)

[1.4.4 Mô hình Master-Slave. 11](#_Toc485996468)

[1.4.5 Mô hình pipeline (mô hình đường ống). 12](#_Toc485996470)

[1.4.6 Mô hình lai. 14](#_Toc485996471)

[**CHƯƠNG 2.** 15](#_Toc485996472)

[**Giới thiệu về thư viện lập trình song song MPI.** 15](#_Toc485996473)

[2.1 MPI là gì? 15](#_Toc485996474)

[2.2 Cấu trúc cơ bản của một chương trình MPI. 16](#_Toc485996475)

[2.3 Các cơ chế truyền thông điệp. 17](#_Toc485996476)

[2.3.1 Point-to-point (giao tiếp điểm-điểm). 17](#_Toc485996477)

[2.3.2 Collective communicationn (giao tiếp tập thể). 17](#_Toc485996478)

[**CHƯƠNG 3.** 22](#_Toc485996481)

[**Đánh giá speed up, efficiency của việc tính toán song song. Định luật Amdahl’s law.** 22](#_Toc485996482)

[3.1. Speed up (tốc độ) và Efficiency (hiệu quả). 22](#_Toc485996483)

[3.2. Định luật Amdahl’s law. 22](#_Toc485996484)

[**CHƯƠNG 4.** 25](#_Toc485996485)

[**Giới thiệu một số thuật toán song song và tuần tự.** 25](#_Toc485996486)

[4.1 Tính số Pi. 25](#_Toc485996487)

[4.1.1 Phương pháp toán học. 25](#_Toc485996488)

[4.1.2 Giải thuật tính số Pi. 25](#_Toc485996489)

[4.1.2.1 Giải thuật tuần tự. 25](#_Toc485996490)

[4.1.2.2 Mô tả giải thuật song song. 25](#_Toc485996491)

[4.1.2.3 Giải thuật song song. 26](#_Toc485996492)

[4.2 Nhân hai ma trận. 30](#_Toc485996493)

[4.2.1 Phương pháp toán học. 30](#_Toc485996494)

[4.2.2 Giải thuật nhân ma trận với ma trận. 30](#_Toc485996495)

[4.2.2.1 Giải thuật tuần tự. 30](#_Toc485996496)

[4.2.2.2 Mô tả giải thuật song song. 31](#_Toc485996497)

[4.2.2.3 Giải thuật song song. 31](#_Toc485996498)

[4.3 Nhân ma trận với véc-tơ. 36](#_Toc485996499)

[KẾT LUẬN 39](#_Toc485996500)

# DANH MỤC CÁC BẢNG

[Bảng 4‑1 So sánh kết quả chạy 30 lần thuật toán tính số Pi giữa giải thuật tuần tự và song song với đơn vị tính bằng giây. 28](#_Toc487665516)

[Bảng 4‑2 So sánh kết quả chạy 30 lần thuật toán nhân ma trận với ma trận giữa giải thuật tuần tự và song song với đơn vị tính bằng giây. 35](#_Toc487665531)

[Bảng 4‑3 So sánh kết quả chạy 30 lần thuật toán nhân ma trận với véc-tơ giữa giải thuật tuần tự và song song với đơn vị tính bằng giây. 38](#_Toc487665545)

**DANH MỤC CÁC HÌNH**

[Hình 1‑1 Đồ thị thể hiện sự gia tăng của các transistor qua các năm 4](#_Toc485809505)

[Hình 1‑2 Mô hình song song dữ liệu 8](#_Toc485809506)

[Hình 1‑3 Mô hình đồ thị tác vụ 9](#_Toc485809507)

[Hình 1‑4 Mô hình Work Pool 11](file:///C:\Users\sony\Desktop\LuanVan\TỔNG%20LIÊN%20ĐOÀN%20LAO%20ĐỘNG%20VIỆT%20NAM.docx#_Toc485809508)

[Hình 1‑5 Mô hình Master - Slave 12](file:///C:\Users\sony\Desktop\LuanVan\TỔNG%20LIÊN%20ĐOÀN%20LAO%20ĐỘNG%20VIỆT%20NAM.docx#_Toc485809509)

[Hình 1‑6 Ví dụ việc láp ráp xe đạp áp dụng mô hình Pipeline 14](#_Toc485809510)

[Hình 2‑1 Cơ chế Broadcast 18](file:///C:\Users\sony\Desktop\LuanVan\TỔNG%20LIÊN%20ĐOÀN%20LAO%20ĐỘNG%20VIỆT%20NAM.docx#_Toc485809511)

[Hình 2‑2 Cơ chế Scatter 18](file:///C:\Users\sony\Desktop\LuanVan\TỔNG%20LIÊN%20ĐOÀN%20LAO%20ĐỘNG%20VIỆT%20NAM.docx#_Toc485809512)

[Hình 2‑3 Cơ chế Gather 19](file:///C:\Users\sony\Desktop\LuanVan\TỔNG%20LIÊN%20ĐOÀN%20LAO%20ĐỘNG%20VIỆT%20NAM.docx#_Toc485809513)

[Hình 2‑4 Cơ chế Reduce 19](file:///C:\Users\sony\Desktop\LuanVan\TỔNG%20LIÊN%20ĐOÀN%20LAO%20ĐỘNG%20VIỆT%20NAM.docx#_Toc485809514)

# TÓM TẮT

Khoa học kỹ thuật ngày càng phát triển, đặt ra nhiều bài toán với khối lượng tính toán rất lớn. Trong số đó có những bài toán mà kết quả chỉ có ý nghĩa nếu được hoàn thành trong khoảng thời gian cho phép. Ví dụ như các tính toán trong thời gian thực, mô phỏng sự chuyển động của các phân tử, tính quĩ đạo chuyển động của vật thể trong không gian, dự báo thời tiết... Để giải quyết những bài toán này, người ta đã nghiên cứu tăng tốc độ tính toán bằng phương pháp tính toán song song với sự thực thi đồng thời của nhiều tài nguyên máy tính giúp giải quyết các bài toán yêu cầu thời gian xử lý nhanh, đồng thời xử lý khối lượng dữ liệu lớn.

Kỹ thuật tính toán song song cho phép giảm thời gian thực thi bài toán tùy thuộc cách phân chia và số bộ xử lý thực thi chương trình. Nguyên tắc quan trọng nhất của tính toán song song chính là tính đồng thời hay còn được hiểu là xử lý nhiều tác vụ cùng một lúc.

Mục đích của khóa luận là nghiên cứu phương pháp tính toán song song, các chuẩn hỗ trợ cho việc lập trình song song, và tiến hành cài đặt một số thuật toán theo kỹ thuật lập trình song song.

Bố cục của khóa luận được phân thành 4 chương:

Chương 1: Giới thiệu tổng quan về lập trình song song và các mô hình của nó. Cũng như lý do tại sao chúng ta cần phải sử dụng lập trình song song.

Chương 2: Giới thiệu các thư viện được sử dụng trong lập trình song song. Và cụ thể trong khóa luận này là thư viện MPI.

Chương 3: Đánh giá tốc độ và độ hiệu quả của lập trình song song. Đồng thời giới thiệu định luật Amdahl’s law.

Chương 4: Mô tả các giải thuật bằng phương pháp lập trình tuần tự và lập trình song song như: giải thuật tính số pi, ma trận nhân véc-tơ, ma trận nhân ma trận. Qua đó, so sánh kết quả thời gian thực hiện giữa 2 phương pháp lập trình.

# CHƯƠNG 1.

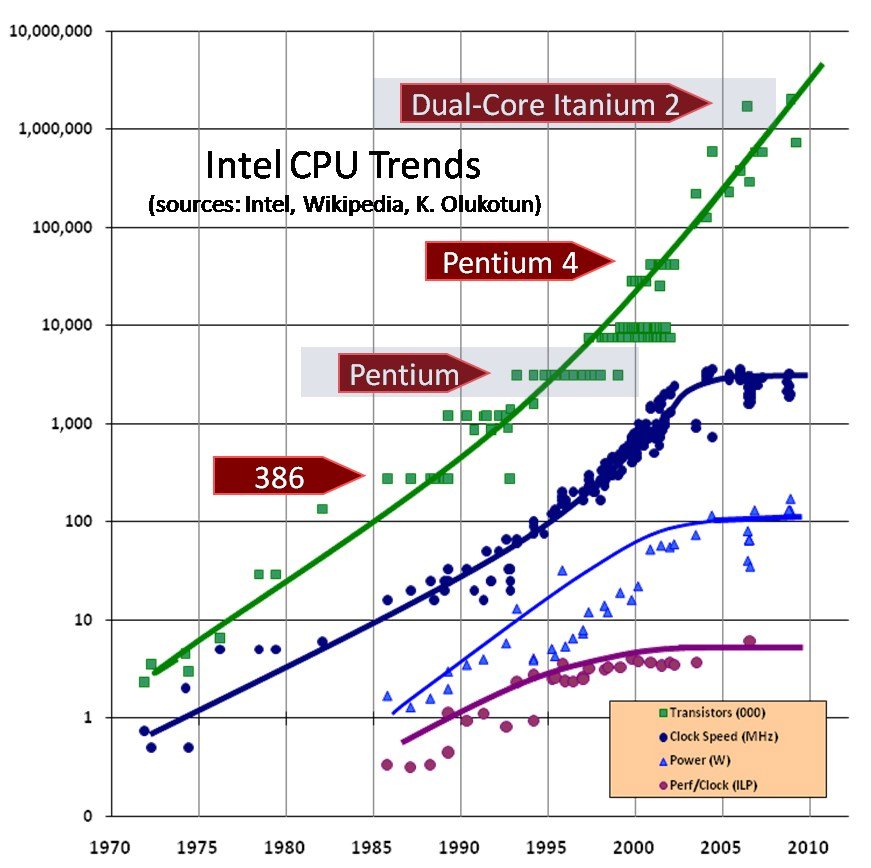
# Giới thiệu chung về lập trình song song.

# 1.1 Lập trình song song là gì?

Ngày nay khoa học kỹ thuật càng phát triển, đặt ra nhiều bài toán với khối lượng tính toán rất lớn. Trong số đó có những bài toán mà kết quả chỉ có ý nghĩa nếu chúng được hoàn thành trong khoảng thời gian cho phép. Ví dụ như để dự báo thời tiết cho ngày mai, thì chúng ta cần phải có kết quả của các ngày trước đó. Và kết quả đó phải có được chậm nhất là vào tối nay. Nhưng nếu chỉ với việc thực hiện trên một máy tính đơn như thông thường, nghĩa là một bài toán được giải quyết thông qua một chuỗi các câu lệnh tuần tự, thì chúng ta khó mà có được kết quả như mong muốn. Và để giải quyết những bài toán này, người ta đã nghiên cứu tăng tốc độ tính toán bằng hai phương pháp hay kết hợp cả hai:

Phương pháp 1: Cải tiến công nghệ, tăng tốc độ xử lý của máy tính. Vì theo định luật Moore : "*Số lượng*[*transistor*](https://vi.wikipedia.org/wiki/Transistor)*trên mỗi đơn vị*[*inch vuông*](https://vi.wikipedia.org/w/index.php?title=Inch_vu%C3%B4ng&action=edit&redlink=1)*sẽ tăng lên gấp đôi sau 18 tháng.*" (1 inch vuông xấp xỉ 6,45 cm²). Mặc dù tốc độ tính toán của các bộ xử lý tăng nhiều qua từng năm, nhưng do giới hạn vật lý nên khả năng tính toán của chúng không thể tăng mãi. Nên nếu muốn tăng khả năng tính toán thì chúng ta phải khai thác được khả năng xử lý song song. Và hình 1.1 thể hiện sự gia tăng của các transistor qua các năm được thể hiện ở bên dưới.

Phương pháp 2: Chia bài toán ra thành những công việc nhỏ có thể chạy song song trên nhiều bộ xử lý. Việc phát triển theo phương pháp này cho ra đời phương pháp tính toán song song, bằng việc sử dụng đồng thời nhiều tài nguyên tính toán để giải quyết một bài toán. Các tài nguyên tính toán có thể bao gồm một máy tính với nhiều bộ vi xử lý hoặc một tập các máy tính kết nối mạng hay là một sự kết hợp của hai dạng trên. Nó có ưu điểm là giá thành rẻ hơn nhiều so với siêu máy tính có cùng sức mạnh (do sử dụng các thiết bị thông thường) và tính linh hoạt của hệ thống.



Hình 1‑1 Đồ thị thể hiện sự gia tăng của các transistor qua các năm (nguồn internet)

# 1.2 Lý do cần phải sử dụng lập trình song song?

Nếu chỉ với phương pháp lập trình tuần tự thông thường thì không thể nào có thể tận dụng hết được sức mạnh của các hệ thống . Đồng thời cũng có thể mất rất nhiều thời gian. Phương pháp tính toán song song cho phép giảm thời gian thực thi bài toán tùy thuộc vào cách phân chia và số lượng bộ xử lý thực thi chương trình. Nguyên tắc quan trọng nhất của tính toán song song chính là tính đồng thời hay còn được hiểu là việc xử lý nhiều process (tiến trình) cùng một lúc. Tuy nhiên, lập trình song song thì phức tạp hơn so với lập trình tuần tự thông thường do gặp phải những mặt hạn chế như khó code, cân bằng tải (sẽ được trình bày ở mục 1.3)

Để có thể dễ dàng hình dung hơn về ứng dụng của lập trình song song thì ta xét các ví dụ sau:

Như muốn dự báo thời tiết cho ngày mai thì chúng ta cẩn phải có số liệu của vài ngày, vài tháng hoặc thậm chí cả vài năm trước đó. Nên nếu chỉ với sức mạnh của một máy tính đơn thông thường thì việc đáp ứng nhu cầu đó có thể sẽ diễn ra rất chậm phải cần đến 2,3 ngày hoặc cũng có thể sẽ không đáp ứng được nhu cầu đó. Nhưng để dự báo thời tiết cho ngày mai, thì kết quả chúng ta phải có được chậm nhất có thể là vào tối nay. Vì nếu dự báo thời tiết cho ngày mai mà đến 2,3 hôm sau mới có kết quả thì kết quả đó cũng chẳng còn có ý nghĩa gì. Do đó, chúng ta phải cần đến sức mạnh của nhiều máy để có thể rút ngắn thời gian thực hiện.

Hoặc muốn mua hay bán cổ phiếu thì chúng ta cần phải có giá trị cổ phiếu của những ngày trước đó, để có thể tính toán được xác suất tăng hoặc giảm của giá trị cổ phiếu. Giúp ta có thể dễ dàng hơn trong việc quyết định mua vào hay bán ra. Nhưng nếu chỉ với sức mạnh của một máy tính đơn thông thường thì kết quả sẽ chậm trễ. Và khi có được kết quả thì người ta đã đi trước chúng ta một bước. Vì vậy, chúng ta cần đến sức mạnh của nhiều máy để có thể giải quyết nhanh chóng hơn.

Tuy nhiên, dù tốc độ tính toán của các bộ xử lý tăng nhiều qua từng năm, nhưng do giới hạn vật lý nên khả năng tính toán của chúng không thể tăng mãi. Đến một lúc nào đó thì dù chúng ta có tăng việc sử dụng lên bao nhiêu máy tính đi nữa thì thời gian thực hiện công việc tính toán để đáp ứng nhu cầu của chúng ta vẫn sẽ không được rút ngắn hơn. Điều này được thể hiện rõ trong định luật Amdahl’s law (mục 3.2).

# 1.3 Cân bằng tải.

Như đã nói ở trên, khi lập trình song song sẽ gặp phải những mặt hạn chế như: khó code, cân bằng tải….

Do khi chúng ta chia một bài toán cho một số tiến trình cố định để thực hiện song song, thì mỗi tiến trình sẽ biết trước được số lượng công việc mà nó cần làm. Khi đó, một số bộ xử lý có thể hoàn thành công việc ở mỗi tiến trình của nó trước các bộ xử lý khác, và sẽ ở trạng thái rảnh rỗi do số lượng công việc được phân chia không đồng đều nhau, hoặc tốc độ tính toán giữa các bộ xử lý là khác nhau (bộ xử lý này tính toán nhanh hơn bộ xử lý khác). Câ**n bằng tải**là cách phân phối khối lượng tải trên nhiều máy tính hoặc một cụm máy tính để có thể sử dụng tối ưu các nguồn lực, tối đa hóa thông lượng, giảm thời gian đáp ứng và tránh tình trạng quá tải, một máy quá nhiều công việc, còn máy thì quá ít hoặc thậm chí không có công việc nào. Bằng cách đó, sẽ giúp cho hệ thống giảm thiểu tối đa tình trạng một máy chủ bị quá tải và ngưng hoạt động. Hoặc khi một máy chủ gặp sự cố, cân bằng tải sẽ chỉ đạo phân phối công việc của máy chủ đó cho các máy chủ còn lại, đẩy thời gian uptime của hệ thống lên cao nhất và cải thiện năng suất hoạt động tổng thể.

Cân bằng tải cũng là cơ chế rất quan trọng trong việc mở rộng quy mô của mạng máy tính. Khi lắp đặt một máy chủ mới vào hệ thống, cân bằng tải sẽ tự động cắt giảm khối lượng công việc từ các máy chủ cũ sang máy chủ mới. Và trong lập trình song song thì cân bằng tải giúp cho tiến trình chủ phân chia công việc đồng đều cho các tiến trình con, tránh trường hợp công việc phân chia không đồng đều dẫn đến việc quá tải giữa các tiến trình, giúp các tiến trình không bị gián đoạn trong việc tính toán.

**1.4 Một số mô hình của lập trình song song.**

Việc đưa ra một mô hình máy tính chung cho việc lập trình giúp cho việc thiết kế giải thuật trở nên đơn giản hơn. Mô hình lập trình song song bao gồm các ứng dụng, ngôn ngữ, bộ biên dịch, thư viện, hệ thống truyền thông và vào/ra song song. Trên thực tế, chưa có một máy tính song song nào, cũng như cách phân chia công việc cho các bộ xử lý nào có thể áp dụng hiệu quả cho mọi bài toán. Do đó, tùy vào từng bài toán mà người lập trình lựa chọn một mô hình song song hoặc pha trộn giữa các mô hình với nhau để phát triển các ứng dụng song song trên một hệ thống cụ thể. Hiện nay có rất nhiều mô hình lập trình song song như:

## 1.4.1 Mô hình Data-Parallel (mô hình song song dữ liệu).

Là một trong những mô hình thuật toán đơn giản nhất. Với mô hình này thì các tiến trình của chương trình làm việc với cùng một cấu trúc dữ liệu. Tuy nhiên mỗi tiến trình sẽ làm việc trên từng phân vùng dữ liệu khác nhau và các nhiệm vụ phải thực hiện các thao tác giống nhau.. Thông thường, các giai đoạn tính toán song song dữ liệu được xen kẽ với các tương tác để đồng bộ hóa các tiến trình hoặc để có được dữ liệu mới cho các tiến trình.

Thuật toán song song dữ liệu có thể được thực hiện trong cả hai mô hình message-passing (chuyển tiếp địa chỉ-không gian và tin nhắn). Tuy nhiên, địa chỉ được phân chia trong mô hình truyền thông điệp (tin nhắn) có thể cho phép kiểm soát tốt hơn, do đó có thể xử lý tốt hơn.

Một đặc điểm chính của các bài toán song song dữ liệu là đối với hầu hết các bài toán, mức độ song song của dữ liệu tăng tỷ lệ thuận với kích thước của nó, do đó ta có thể chia các bài toán lớn thành nhiều bài toán nhỏ rồi phân chia cho nhiều tiến trình thực hiện song song để có thể giải quyết hiệu quả hơn. Hình 1.2 thể hiện mô hình song song dữ liệu

Data Stream 1

Process 1

Process 2

Process 3

Process n

Control Unit

Memory

Unit

Data Stream 2

Data Stream 3

Instruction

Stream

Data Stream n

Hình 1‑2 Mô hình song song dữ liệu

## 1.4.2 Mô hình Task Graph (mô hình đồ thị tác vụ)

Trong mô hình đồ thị tác vụ, mối liên hệ giữa các tiến trình được sử dụng để tăng hoặc giảm chi phí tương tác. Mô hình này thường được sử dụng để giải quyết các vấn đề trong đó lượng dữ liệu liên quan đến các tiến trình lớn hơn so với số lượng tính toán. Thông thường, các công việc ở các tiến trình là có sẵn để giúp tối ưu hóa chi phí di chuyển dữ liệu giữa các tiến trình. Trong mô hình này, công việc được chia sẻ dễ dàng hơn với không gian địa chỉ toàn cục, nhưng trong không gian địa chỉ riêng thì các cơ chế có sẵn sẽ chia sẻ công việc.

Các kỹ thuật giảm tương tác điển hình áp dụng cho mô hình này bao gồm giảm khối lượng và tần suất tương tác bằng cách thúc đẩy các nhiệm vụ dựa trên mô hình tương tác của các tiến trình và sử dụng các phương pháp tương tác không đồng bộ để đan xen sự tương tác với tính toán.

Các ví dụ về các thuật toán dựa trên mô hình đồ thị tác vụ bao gồm phép phân tích song song, phân tích ma trận thưa thớt và nhiều thuật toán song song bắt nguồn từ sự phân rã và phân chia. Hình 1.3 thể hiện mô hình đồ thị tác vụ.

Hình 1‑3 Mô hình đồ thị tác vụ

## 1.4.3 Mô hình Work Pool.

Work Pool hoặc mô hình task pool được đặc trưng bởi một ánh xạ động các công việc lên các quy trình cân bằng tải, trong đó bất kỳ công việc nào cũng có thể được thực hiện bởi bất cứ tiến trình nào mà không cần yêu cầu gì. Việc trao đổi dữ liệu qua lại có thể được tập trung hoặc phân cấp. Con trỏ đến các tiến trình có thể được lưu trữ trong danh sách chia sẻ vật lý, hàng đợi ưu tiên, bảng băm hoặc cây, hoặc chúng có thể được lưu trữ trong một cấu trúc dữ liệu phân tán. Công việc có thể được có sẵn ngay từ đầu hoặc có thể được phát sinh ra trong quá trình xử lý; Tức là các tiến trình có thể tự tạo ra công việc và thêm vào “pool”. Nếu công việc được tạo ra tự động thì cần phải có một thuật toán “phát hiện chấm dứt” (thuật toán này có thể hiểu đơn giản như sau: chúng ta sẽ tạo 1 lá cờ gán cho tiến trình đầu tiên, và sẽ được truyền lần lượt đến các tiến trình kế tiếp. Lá cờ đến tiến trình nào thì tiến trình đó sẽ thực hiện công việc tính toán của mình, và khi hoàn thành xong thì nó sẽ chuyển lá cờ đến tiến trình tiếp theo. Cho đến khi lá cờ quay trở lại tiến trình đầu tiên thì tất cả các tiến trình đã thực hiện xong công việc tính toán của chúng và thuật toán kết thúc.) để tất cả các tiến trình có thể phát hiện sự hoàn thành toàn bộ chương trình, và ngừng tìm kiếm thêm công việc.

Mô hình Work Pool thường được sử dụng khi lượng dữ liệu tương ứng với các tiến trình nhỏ hơn so với việc tính toán liên quan đến dữ liệu đó (ngược lại với mô hình đồ thị tác vụ). Kết quả là các tiến trình có thể giao tiếp với nhau mà không gây ra quá nhiều xung đột dữ liệu.

Việc tìm kiếm trên cây bằng tính toán song song, khi các công việc được lưu trong một cấu trúc dữ liệu tập trung hoặc phân tán là một ví dụ về việc sử dụng mô hình Work Pool mà các nhiệm vụ tự phát sinh thêm trong quá trình xử lý.

Hình 1.4 thể hiện việc các máy tính lấy công việc từ “Pool” theo mô hình Work Pool để giải quyết.

computer

computer

computer

Hình 1‑4 Mô hình Work Pool

## 1.4.4 Mô hình Master-Slave.

#### Trong mô hình này Master đóng vai trò là máy chủ giao việc cho các máy con (Slave), hay còn được hiểu là một hoặc nhiều tiến trình tổng thể tạo ra công việc và phân bổ nó cho tiến trình khác. Ngoài ra, các tiến trình cũng có thể được phân bổ trước các công việc nếu như các Master ước tính được kích thước của các tiến trình và số công việc giữa các tiến trình phải được đảm bảo là phân chia tương đương nhau. Trong một trường hợp khác, Slave được giao những công việc nhỏ ở những thời điểm khác nhau. Điều này được thực hiện nhiều hơn vì không phải mất thời gian để Master tạo ra công việc và không làm cho tất cả Slave chờ đợi cho đến khi Master đã tạo ra tất cả các công việc. Trong một số trường hợp, công việc có thể cần phải được thực hiện theo từng giai đoạn, và việc làm trong giai đoạn tiếp theo phụ thuộc vào kết quả của giai đoạn trước đó. Ở trường hợp này, Master có thể khiến tất cả Slave đồng bộ hóa sau mỗi giai đoạn. Thông thường, không có tiến trình nào có thể làm trọn vẹn hết các công việc, nên bất kỳ Slave nào cũng có thể làm bất cứ công việc nào mà nó được giao. Mô hình Master-Slave có thể được khái quát hoá thành mô hình quản lý cấp cao, trong đó tiến trình cấp cao cung cấp một khối lượng công việc cho các tiến trình cấp thấp hơn, và những tiến trình cấp thấp này lại tiếp tục chia nhỏ nhiệm vụ cho những tiến trình cấp thấp hơn nữa, đồng thời các tiến trình cấp cao có thể đảm nhận một phần công việc. Tức là người quản lý biết rằng họ cần cung cấp công việc và nhân viên biết rằng họ cần phải làm việc từ người quản lý.

Khi sử dụng mô hình master-slave, cần phải cẩn thận để đảm bảo rằng Master không bị tình trạng thắt nút cổ chai (nghĩa là cùng lúc các Slave đều đổ dồn kết quả về cho Master, làm cho nó không kịp nhận dữ liệu dẫn đến tắt nghẽn), nó xảy ra khi công việc giữa các tiến trình được phân chia không đồng đều, hoặc khả năng thực hiện công việc giữa các Slave là nhanh chậm khác nhau. Hình 1.5 thể hiện sự trao đổi dữ liệu qua lại giữa Master và các Slave trong mô hình Master-Slave.

Master

Slave

Slave

Slave

Slave

Hình 1‑5 Mô hình Master - Slave

## 1.4.5 Mô hình pipeline (mô hình đường ống).

Trong mô hình đường ống, một luồng dữ liệu được truyền đi thông qua một loạt các giai đoạn, mỗi giai đoạn thực hiện một nhiệm vụ của nó. Thực hiện đồng thời các chương trình khác nhau trên cùng một luồng dữ liệu được gọi là stream parallelism (song song luồng). Ngoại trừ giai đoạn bắt đầu đường ống, thì sự xuất hiện của dữ liệu mới sẽ kích hoạt một nhiệm vụ mới ứng với một giai đoạn trong đường ống, và có thể hình thành các đường ống đó dưới dạng các mảng tuyến tính hoặc đa chiều, cây hoặc các đồ thị chunk có hoặc không có chu kỳ. Một đường ống là một chuỗi các “nhà sản xuất” và “người tiêu dùng”. Mỗi quá trình trong đường ống có thể được xem như là một “người tiêu dùng” của một dãy các dữ liệu thuộc quá trình trước nó, và đồng thời cũng được xem như là một “nhà sản xuất” dữ liệu cho quá trình sau nó trong đường ống.

Một ví dụ về đường ống hai chiều là thuật toán phân tích LU song song (phân tích LU là phương pháp phân tích [ma trận](https://vi.wikipedia.org/wiki/Ma_tr%E1%BA%ADn) thành tích của một ma trận tam giác dưới và một ma trận tam giác trên, ví dụ với ma trận A 3x3 thì phân tích LU sẽ là:

=

Để dễ dàng hình dung mô hình pipeline, hình 1.6 dưới đây sẽ là mô hình ví dụ về quy trình lắp ráp một chiếc xe đạp gồm 3 giai đoạn: ráp sườn xe, ráp bánh xe, và ráp thắng xe. Ở lần 1: giai đoạn 1 sẽ ráp sườn xe, giai đoạn 2 sẽ ráp bánh xe, giai đoạn 3 sẽ ráp thắng xe. Đến lần 2: trong lúc giai đoạn 1 ráp sườn xe xong giao qua cho giai đoạn 2 ráp bánh xe, thì giai đoạn 1 sẽ nhận một cái sườn xe khác để ráp. Tiếp theo lần 3, thì giai đoạn 2 đã ráp xong bánh xe giao cho giai đoạn 3 và tiếp tục nhận một sườn xe khác từ giai đoạn 1 để ráp bánh, khi đó giai đoạn 1 lại nhận tiếp một sườn xe mới để ráp. Với việc làm cuốn chiếu như thế thì bắt đầu từ lần 4 trở đi, sau mỗi lần sẽ có 1 chiếc xe đạp ra đời.

Lần 1:

|  |  |  |
| --- | --- | --- |
| Ráp sườn xe | Ráp bánh xe | Ráp thắng xe |

|  |  |  |
| --- | --- | --- |
| Ráp sườn xe | Ráp bánh xe | Ráp thắng xe |

Lần 2:

|  |  |  |
| --- | --- | --- |
| Ráp sườn xe | Ráp bánh xe | Ráp thắng xe |

Lần 3:

Hình 1‑6 Ví dụ việc láp ráp xe đạp áp dụng mô hình Pipeline

## 1.4.6 Mô hình lai.

Trong một số trường hợp, nhiều mô hình có thể được áp dụng cho một vấn đề, kết quả là một mô hình thuật toán lai. Một mô hình lai có thể được tạo ra hoặc là bởi các mô hình phân cấp hoặc là bởi nhiều mô hình được áp dụng theo các giai đoạn khác nhau của một thuật toán song song. Trong một số trường hợp, một công thức thuật toán có thể có các đặc tính của nhiều mô hình thuật toán. Ví dụ, dữ liệu có thể chảy theo kiểu đường ống bởi một đồ thị tác vụ. Trong một trường hợp khác, tính toán chính có thể được mô tả bởi đồ thị tác vụ, nhưng mỗi nút của biểu đồ có thể đại diện cho một siêu văn bản bao gồm nhiều công việc phụ có thể phù hợp với song song dữ liệu hoặc đường ống.

Thuật toán Quick sort (sắp xếp nhanh) bằng giải thuật song song là một trong những ứng dụng điển hình của mô hình lai.

# CHƯƠNG 2.

# Giới thiệu về thư viện lập trình song song MPI.

# 2.1 MPI là gì?

Hiện nay đã có nhiều công cụ phổ biến hỗ trợ cho việc lập trình song song như MPI (Message Passing Interface) hỗ trợ lập trình song song trên mô hình bộ nhớ phân tán, OpenMP (Open MultiProcessing) hỗ trợ lập trình song song trên mô hình chia sẻ bộ nhớ chung, Pthread hỗ trợ lập trình luồng ... Chúng cung cấp các hàm dùng cho việc trao đổi thông tin giữa các tiến trình tính toán trong hệ thống song song. Và trong khóa luận này, chúng ta chỉ tìm hiểu đến thư viện MPI dùng để hỗ trợ lập trình song song trên mô hình bộ nhớ phân tán.

Mô hình bộ nhớ phân tán là trong đó không tồn tại bộ nhớ chia sẻ chung mà mỗi bộ xử lý có bộ nhớ riêng của chúng. Trong máy tính song song có bộ nhớ phân tán, thì các bộ xử lý liên lạc với nhau bằng các thông điệp qua một mạng liên kết (gồm các liên kết truyền thông trực tiếp giữa một số bộ xử lý). Điều đó dẫn đến việc quan trọng là cần phải lựa chọn là các bộ xử lý nào được nối với nhau. Vì tốc độ liên lạc là tối ưu khi các bộ xử lý được nối trực tiếp với nhau. Tuy nhiên điều này thường là không khả thi do số lượng các liên kết là quá lớn dẫn đến việc tăng giá thành của hệ thống. Cách thứ hai được sử dụng là các bộ xử lý liên lạc thông qua một BUS chia sẻ. Điều này dẫn đến việc tốn nhiều thời gian hơn khi số lượng bộ xử lý lớn dẫn đến vấn đề tranh chấp BUS. Để hỗ trợ cho lập trình song song trên bộ nhớ phân tán thì chúng ta thường dùng thư viện MPI.

MPI là từ viết tắt của Message Passing Interface. Đây là một chuẩn mô tả các đặc điểm và cú pháp của một thư viện lập trình song song dựa vào phương thức gửi nhận tin bởi MPIF(Message Passing Interface Forum). Bộ công cụ này cung cấp cho ta các hàm để sử dụng cho việc trao đổi thông tin giữa các tiến trình tính toán trong hệ thống song song. MPI có thể được viết bằng C/C++,fortran.

MPI được lập trình cho các máy tính có kiến trúc có bộ nhớ phân tán. Tuy nhiên, đôi lúc nó cũng có thể dùng cho vài kiến trúc chia sẻ bộ nhớ như kiến trúc SMP/NUMA.

# 2.2 Cấu trúc cơ bản của một chương trình MPI.

Khai báo header, biến, prototype…..

Bắt đầu chương trình

…………

<đoạn code tuần tự>

..…………

Khởi tạo môi trường MPI

..…………

**<**đoạn code cần thực hiện song song**>**

..…………

Kết thúc môi trường MPI

..…………

<đoạn code tuần tự>

..…………

Kết thúc chương trình

# 2.3 Các cơ chế truyền thông điệp.

## 2.3.1 Point-to-point (giao tiếp điểm-điểm).

Đây là cơ chế giao tiếp giữa từng cặp tiến trình với nhau, trong đó một tiến trình thực hiện công việc gửi thông điệp và tiến trình còn lại có nhiệm vụ nhận thông điệp đó. Các thông điệp được phân biệt với nhau qua chỉ số tác vụ. Trong cơ chế này có nhiều kiểu giao tiếp với nhau như:

* Blocking: các lệnh gửi hoặc nhận dữ liệu sẽ kết thúc khi việc gửi hoặc nhận dữ liệu hoàn tất.
* Non-blocking: các lệnh gửi hoặc nhận dữ liệu sẽ kết thúc ngay mà không quan tâm đến việc dữ liệu đã hoàn toàn được gửi đi hay nhận về hay chưa. Mà việc đó sẽ được kiểm tra bằng các lệnh khác trong thư viện MPI.
* Synchronous: gửi dữ liệu đồng bộ, quá trình gửi dữ liệu chỉ kết thúc khi quá trình nhận dữ liệu được bắt đầu.
* Buffer: một vùng nhờ đệm sẽ được tạo ra để chứa dữ liệu trước khi được gửi đi, người dùng sẽ ghi đè lên vùng bộ nhớ chứa dữ liệu mà không sợ làm mất dữ liệu chuẩn bị gửi.
* Ready: quá trình gửi dữ liệu chỉ có thể được bắt đầu khi quá trình nhận dữ liệu đã sẵn sàng.

## 2.3.2 Collective communication (giao tiếp tập thể).

Liên quan đến tất cả các tiến trình nằm trong phạm vi communicator (tập hợp các tiến trình), các kiểu giao tiếp trong cơ chế này gồm có:

* Broadcast: dữ liệu giống nhau được gửi từ tiến trình gốc đến tất cả các tiến trình khác trong communicator. Broadcasting có hai loại:
* All-to-all: tất cả các bộ xử lý gửi thông điệp tới tất cả các bộ xử lý khác.
* One-to-all: một bộ xử lý gửi cùng một thông điệp tới tất cả các bộ xử lý khác.

Hình 2‑ Cơ chế Broadcast

* Scatter: các dữ liệu khác nhau được gửi từ tiến trình gốc đến tất cả các tiến trình khác trong communicator.

Hình 2‑2 Cơ chế Scatter

* Gather: các dữ liệu khác nhau được gửi đến tiến trình gốc từ tất cả các tiến trình khác trong communicator.

Hình 2‑3 Cơ chế Gather

* Reduce: phương thức này cho phép ta có thể lấy dữ liệu từ mỗi tiến trình, rút gọn dữ liệu, lưu trữ dữ liệu vào trong tiến trình gốc hoặc trong tất cả các tiến trình.

Hình 2‑4 Cơ chế Reduce

**2.4 Một số lệnh trong MPI.**

Trong thư viện MPI có khoảng 125 hàm. Tuy nhiên, trong khóa luận này chúng ta chỉ sử dụng 6 hàm chủ yếu sau:

**MPI\_Init**: khởi tạo môi trường MPI để thực hiện tiếm trình song song, lệnh này trả về một giá trị nguyên trong quá trình khởi tạo môi trường.

MPI\_Init(&argc, &argv);

Trong đó, tham số argc và argv của hàm MPI\_Init được nhận thông qua các câu lệnh của chương trình C.

**MPI\_Comm\_size**: trả về tổng số tiến trình song song đang thực hiện trong communicator.

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

Với MPI\_COMM\_WORLD là một communicators mặc định do MPI định nghĩa ra, và nstask là tổng số tác vụ của chương trình

**MPI\_Comm\_rank**: trả về chỉ số (rank) của tác vụ, chỉ số này có giá trị từ 0 đến nstask -1 (với nstask là tổng số tác vụ), và được sử dụng đề nhận biết tác vụ khi điều khiển gửi/nhận thông tin.

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

Tương tự với hàm MPI\_Comm\_size thì tham số MPI\_COMM\_WORLD ở trong hàm này cũng là một communicators mặc định do MPI định nghĩa ra, và tham số mystask là chỉ số của tác vụ.

**MPI\_Finalize**: kết thúc môi trường MPI, tuy nhiên các tác vụ chạy song song đang được thực thi vẫn tiếp tục chạy, và các lệnh MPI được gọi sau MPI\_Finalize đều không thực hiện được và báo lỗi.

MPI\_Finalize();

Nếu MPI\_Finalize thực hiện thành công thì nó sẽ trả vể giá trị là MPI\_SUCCESS.

**MPI\_Send** (&buffer, count, datatype, destination, tag, communicator ): để gửi thông điệp giữa các tác vụ.

**MPI\_Recv** (&buffer, count, datatype, source, tag, communicator, &status): để nhận thông điệp giữa các tác vụ.

Trong đó

buffer: mảng dữ liệu cần chuyển/nhận

count: số phần tử trong mảng

datatype: kiểu dữ liệu

destination: chỉ số của tác vụ đích

source: chỉ số của tác vụ nguồn

tag: nhãn thông điệp

communicator: tập hợp các tác vụ

status: trạng thái của thông điệp

ierror: mã số lỗi

# CHƯƠNG 3.

# Đánh giá speed up, efficiency của việc tính toán song song. Định luật Amdahl’s law.

# 3.1. Speed up (tốc độ) và Efficiency (hiệu quả).

Khi đánh giá một hệ thống song song, chúng ta thường quan tâm đến hiệu quả xử lý của chúng so với việc thực hiện tuần tự, bằng cách cho chạy cùng một chương trình trên cả hai hệ thống song song và tuần tự để so sánh kết quả. Ngoài đánh giá về mặt hiệu quả thì tốc độ cũng là một đại lượng tương đối thể hiện khả năng giải quyết vấn đề song song. Nó được định nghĩa là tỷ số thời gian để giải quyết vấn đề trên một máy tuần tự (một phần tử xử lý duy nhất) chia cho thời gian cần thiết để giải quyết cùng một vấn đề trên một máy tính song song (các phần tử xử lý giống hệt nhau).

Công thức tính như sau:

**Speed up** =

**Efficiency** =

Và việc để gia tăng tốc độ của chương trình còn được thể hiện rõ nét hơn trong định luật Amdahl’s law.

# 3.2. Định luật Amdahl’s law.

Định luật Amdahl’s law được sử dụng trong [tính toán song song](https://vi.wikipedia.org/wiki/T%C3%ADnh_to%C3%A1n_song_song) để dự đoán sự [tăng tốc](https://vi.wikipedia.org/w/index.php?title=T%C4%83ng_t%E1%BB%91c&action=edit&redlink=1) trên lý thuyết khi chúng ta sử dụng nhiều bộ xử lý. Tuy nhiên, sự tăng tốc của một chương trình sử dụng nhiều bộ xử lý trong tính toán song song vẫn bị hạn chế bởi một phần tính toán tuần tự của chương trình, nên đến một lúc nào đó số bộ xử lý dùng trong tính toán song song lớn bao nhiêu chăng nữa, thì thời gian tối thiểu để thực hiện chương trình cũng sẽ không còn cải thiện thêm được nữa.

Ví dụ: một bộ xử lý cần 20 giờ để thực hiện chương trình. Trong đó, có một phần của chương trình thực hiện tuần tự (tương ứng với 1 giờ thực hiện). Từ đó, ta có 95% của chương trình (tương ứng với 19 giờ) có thể xử lý song song. Như vậy số bộ xử lý dùng trong tính toán song song lớn bao nhiêu chăng nữa, thì thời gian tối thiểu để thực hiện chương trình không thể nhỏ hơn 1 giờ (do có một phần của chương trình thực hiện tuần tự trong 1 giờ).

|  |  |
| --- | --- |
| Tuần tự | Song song |

Tuần tự

**T(1)**

Trong đó: : khoảng thời gian thực hiện bằng giải thuật tuần tự.

: khoảng thời gian thực hiện bằng giải thuật song song.

khoảng thời gian thực hiện toàn bộ chương trình.

Song song

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Tuần tự |  |  |  |  |  |  |  |  |  |  |

**T(N)**

với : khoảng thời gian thực hiện bằng giải thuật tuần tự và trên 1 máy song song.

N: số bộ vi xử lý.

= T(1) =

T(N) = T(1) +

**Speed up** =

**Speed up** = = (N )

# CHƯƠNG 4.

# Giới thiệu một số thuật toán song song và tuần tự.

# 4.1 Tính số Pi.

## 4.1.1 Phương pháp toán học.

Số Pi là tỉ số giữa chu vi hình tròn và đường kính của nó. Tỉ số này không đổi với tất cả các hình tròn, dù hình tròn có đường kính rất nhỏ hay hình tròn có đường kính rất lớn, thì giá trị của số Pi vẫn luôn luôn là 3,14161265318…

Cho đến nay, có rất nhiều công thức để tính số Pi. Và trong giải thuật này ta áp dụng công thức:

= , với =

## 4.1.2 Giải thuật tính số Pi.

### 4.1.2.1 Giải thuật tuần tự.

**-** Gán tổng = 0.

**-** Cho lặp từ i = 1 đến n tính = và tính tổng = tổng + (4/(\* )) .

**-** Số Pi = \* tổng.

### 4.1.2.2 Mô tả giải thuật song song.

**-** Theo công thức tính số pi như trên phương pháp toán học ta có thể viết lại dưới dạng:

Pi = f(1) + f(2) + f(3) + f(4) +….+ f(n) (với f(n) = \* )

Khi đó, các tiến trình sẽ được phân chia tính toán theo từng f(n) riêng lẽ dựa vào chỉ số của tiến trình và bước nhảy của nó, ví dụ như: tiến trình 1 sẽ tính f(1) và f(3), tiến trình 2 sẽ tính f(2) và f(4)… cho đến khi phân chia hít cho các tiến trình. Sau khi được phân chia xong thì các tiến trình con sẽ thực hiện công việc tính toán lần lượt từng f(n) và gửi về cho tiến trình chủ. Cuối cùng, tiến trình chủ sẽ tập hợp kết quả các f(n) từ các tiến trình con, để tính tổng rồi xuất ra kết quả số pi cuối cùng.

### 4.1.2.3 Giải thuật song song.

**-** Tác vụ chủ sẽ truyền tham số n cho tất cà các tác vụ con

**-** Nếu n = 0 tác vụ con sẽ thoát.

**-** Ngược lại, sẽ gán h = 1.0/(double)n;

**-** Gán sum =0.0.

**-** Cho vòng lặp i chạy từ chỉ số tác vụ + 1 đến n, để tính x, sum.

**-** Tính số Pi bằng h\*sum.

**-** Các tác vụ con sẽ lưu kết quả và gửi về cho tác vụ chính.

**-** Tác vụ chính sẽ tính tổng từ các kết quả tác vụ con đã gửi về, và xuất kết quả.

Bảng kết quả chạy 30 lần thuật toán tính số Pi với n = 1.000.000.000 giữa giải thuật tuần tự và song song với đơn vị tính bằng giây.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Lần chạy thứ** | **Tuần tự** | **Song song** | | |
| **2 máy** | **3 máy** | **4 máy** |
| **1** | 19,00 | 10,07 | 6,80 | 5,19 |
| **2** | 19,20 | 10,11 | 6,93 | 5,47 |
| **3** | 19,11 | 10,10 | 6,96 | 5,44 |
| **4** | 19,29 | 10,23 | 6,88 | 5,45 |
| **5** | 19,22 | 10,10 | 6,92 | 5,45 |
| **6** | 19,12 | 10,26 | 6,90 | 5,58 |
| **7** | 19,19 | 10,18 | 6,96 | 5.53 |
| **8** | 19,16 | 10,17 | 6,86 | 5,48 |
| **9** | 19,24 | 10,11 | 6,92 | 5,60 |
| **10** | 19,51 | 10,15 | 6,92 | 5,88 |
| **11** | 19,39 | 10,25 | 6,89 | 5,99 |
| **12** | 19,64 | 10,07 | 6,90 | 5,63 |
| **13** | 19,75 | 10,09 | 6,91 | 5,64 |
| **14** | 19,42 | 10,13 | 6,95 | 5,94 |
| **15** | 19,45 | 10,15 | 6,92 | 5,52 |
| **16** | 19,44 | 10,08 | 6,95 | 5,98 |
| **17** | 19,67 | 10,03 | 6,96 | 5,49 |
| **18** | 19,42 | 10,15 | 6,86 | 5,80 |
| **19** | 19,49 | 10,18 | 6,84 | 5,72 |
| **20** | 19,31 | 10,18 | 6,98 | 5,20 |
| **21** | 19,51 | 10,21 | 6,87 | 5,71 |
| **22** | 19,33 | 10,16 | 6,90 | 5,88 |
| **23** | 19,52 | 10,21 | 6,94 | 5,76 |
| **24** | 19,43 | 10,16 | 6,89 | 5,47 |
| **25** | 19,42 | 10,09 | 6,89 | 5,17 |
| **26** | 19,32 | 10,12 | 6,89 | 5,48 |
| **27** | 19,53 | 10,11 | 6,94 | 5,37 |
| **28** | 19,34 | 10,09 | 6,89 | 5.86 |
| **29** | 19,35 | 10,11 | 6,96 | 5,52 |
| **30** | 19,05 | 10,10 | 6,96 | 5,41 |
| **Trung bình thời gian** | 19,36 | 10,14 | 6,90 | 5,59 |

Bảng 4‑1 So sánh kết quả chạy 30 lần thuật toán tính số Pi giữa giải thuật tuần tự và song song với đơn vị tính bằng giây.

* Tính Speed up và Efficiency của giải thuật:
* Giữa thực hiện tuần tự và song song trên 2 máy

**Speed up** = = = 1,91(lần)

**Efficiency** = = = 95(%)

* Giữa thực hiện tuần tự và song song trên 3 máy

**Speed up** = = = 2,80(lần)

**Efficiency** = = = 93,33(%)

* Giữa thực hiện tuần tự và song song trên 4 máy

**Speed up** = = = 3,47(lần)

**Efficiency** = = = 86,57(%)

* Nhận xét: Khi tăng từ một máy chạy tuần tự lên hai máy chạy song song thì speed up = 1,91 lần nghĩa là tốc độ về mặt thời gian giữa chạy song song trên hai máy tăng gấp 1,91 lần so với chạy tuần tự. Và efficiency = 95% tức là khi thêm một máy vào chạy song song thì máy đó đóng góp 95% sức mạnh của nó. Tương tự, khi chạy song song trên ba máy thì speed up = 2,8 lần và efficiency = 93,33%. Chạy song song trên bốn máy thì speed up = 3,47 lần và efficiency = 86,57%. Qua đó, ta thấy không phải khi số máy thực hiện tăng thêm thì tốc độ và hiệu quả cũng tăng lên tuyến tính (tăng thêm một máy thì tăng gấp đôi, tăng thêm hai máy thì tăng gấp ba), mà nó sẽ chỉ tăng đến một lúc nào đó rồi sẽ không thay đổi, đúng như định luật Amdahl’s law.

# 4.2 Nhân hai ma trận.

## 4.2.1 Phương pháp toán học.

Cho hai ma trận A = , B = (số cột của A bằng số dòng của B). Tích của hai ma trận AxB là ma trận C:

. = =

Trong đó, = + + +….+

Tức bằng tổng các tích phần tử tương ứng nằm ở dòng thứ i của A với các phần tử nằm ở cột thứ j của B.

## 4.2.2 Giải thuật nhân ma trận với ma trận.

### 4.2.2.1 Giải thuật tuần tự.

**-** Cho biến i chạy từ 0 đến số dòng của ma trận A

**-** Cho biến j chạy từ 0 đến số cột của ma trận A

**-** Phần tử của ma trận A là tổng của i+j

**-** Cho biến i chạy từ 0 đến số cột của ma trận A

**-** Cho biến j chạy từ 0 đến số cột của ma trận B

**-** Phần tử của ma trận B là tích của i\*j

**-** Cho biến i chạy từ 0 đến số dòng của ma trận A

**-** Cho biến j chạy từ 0 đến số cột của ma trận B

**-** Gán matrantich[i][j] = 0

**-** Cho biến k chạy từ 0 đến số cột của ma trận B

**-** matrantich[i][j] = matrantich[i][j] + a[i][k] \* b[k][j]

**-** Cho biến i chạy từ 0 đến số cột của ma trận A

**-** Cho biến j chạy từ 0 đến số cột của ma trận B

- In kết quả matrantich[i][j]

### 4.2.2.2 Mô tả giải thuật song song.

**-** Bước 1: khởi tạo 2 ma trận A, B.

-Bước 2: tiến trình chủ sẽ gửi một số dòng của ma trận A và toàn bộ ma trận B đến tiến trình con. Trong đó, một số dòng của ma trận A được xác định phụ thuộc vào số tiến trình con là chẵn hay lẻ. Ví dụ: ma trận A có 9 dòng, và có 3 tiến trình con thì mỗi tiến trình sẽ giữ 3 dòng. Nhưng trường hợp ma trận A có 9 dòng, mà có đến 4 tiến trình con thì mỗi tiến trình con sẽ giữ 2 dòng và còn dư 1 dòng sẽ gửi cho tiến trình đầu tiên.

- Bước 3: tiến trình con nhận một số dòng của ma trận A và toàn bộ ma trận B từ tiến trình chính đã gửi và tính toán nhân hai ma trận, trong đó ma trận thứ nhất là một số dòng ma trận A và ma trận thứ hai là toàn bộ ma trận B. Sau đó, trả kết quả về cho tiến trình chính.

- Bước 4: tiến trình chính nhận kết quả từ các tiến trình con, và dựa vào chỉ số của các tiến trình để sắp xếp lại ma trận, rồi xuất kết quả.

### 4.2.2.3 Giải thuật song song.

- Khởi tạo 2 ma trận A, B tương tự code tuần tự bằng cách:

+ Cho biến i chạy từ 0 đến số dòng của ma trận A

+ Cho biến j chạy từ 0 đến số cột của ma trận A

+ Phần tử của ma trận A là tổng của i+j

+ Cho biến i chạy từ 0 đến số cột của ma trận A

+ Cho biến j chạy từ 0 đến số cột của ma trận B

+ Phần tử của ma trận B là tích của i\*j

- Gọi numworker là số tác vụ con.

- Tiến trình chủ sẽ gửi một số dòng của ma trận A và toàn bộ ma trận B đến tiến trình con. Trong đó, một số dòng của ma trận A được xác định phụ thuộc vào số chỉ số tác vụ và số dòng của ma trận A chia lấy dư cho numworker:

+ Nếu chỉ số tác vụ nhỏ hơn hoặc bằng số dòng của ma trận A chia lấy dư cho numworker thì số dòng của mà ma trận A gửi đến các tiến trình con là bằng số dòng của ma trận A chia lấy nguyên cho numworker rồi cộng thêm 1.

+ Ngược lại, thì số dòng của mà ma trận A gửi đến các tiến trình con là bằng số dòng của ma trận A chia lấy nguyên cho numworker.

- Tiến trình con nhận một số dòng của ma trận A và toàn bộ ma trận B từ tiến trình chính. Sau đó, tiến trình con sẽ thực hiện các công việc:

+ Tính toán nhân ma trận như cách nhân ma trận với ma trận theo thuật toán tuần tự.

+ Trả kết quả đã tính toán được về cho tiến trình chủ.

- Tiến trình chủ nhận kết quả nhân 2 ma trận mà tiến trình con đã gửi trả về. Và sắp xếp chúng dựa vào chỉ số của các tiến trình con.

- Cuối cùng, tiến trình chủ sẽ in kết quả.

Bảng kết quả chạy 30 lần thuật toán nhân 2 ma trận ( với ma trận cấp 600) giữa giải thuật tuần tự và song song với đơn vị tính bằng giây.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Lần chạy thứ** | **Tuần tự** | **Song song** | | |
| **3 máy** | **4 máy** |
| **1** | 5,30 | 4,45 | 3,07 |
| **2** | 5,33 | 4,32 | 3,88 |
| **3** | 5,43 | 4,00 | 3,62 |
| **4** | 5,93 | 4,16 | 3,72 |
| **5** | 5,56 | 4,20 | 3,76 |
| **6** | 5,69 | 4,24 | 3,88 |
| **7** | 5,55 | 4,50 | 3,96 |
| **8** | 5,52 | 4,25 | 3,46 |
| **9** | 5,53 | 4.37 | 3,65 |
| **10** | 5,28 | 4,62 | 3,99 |
| **11** | 5,92 | 4,24 | 3,48 |
| **12** | 5,92 | 4,58 | 3,40 |
| **13** | 5,46 | 4,18 | 3,53 |
| **14** | 5,92 | 4,00 | 3,89 |
| **15** | 5,57 | 4,20 | 3,57 |
| **16** | 5,61 | 4,34 | 3,80 |
| **17** | 5,51 | 4,02 | 3,55 |
| **18** | 5,70 | 4,24 | 3,78 |
| **19** | 5,71 | 4,58 | 3,75 |
| **20** | 5,38 | 4,57 | 3,88 |
| **21** | 5,72 | 4,20 | 3,97 |
| **22** | 5,78 | 4,17 | 3,30 |
| **23** | 5,47 | 4,34 | 3,51 |
| **24** | 5,99 | 4,61 | 3,95 |
| **25** | 5,45 | 4,22 | 3,56 |
| **26** | 5,54 | 4,20 | 3,56 |
| **27** | 5,53 | 4,24 | 3,95 |
| **28** | 5,41 | 4,12 | 3,02 |
| **29** | 5,94 | 4,35 | 3,41 |
| **30** | 5,87 | 4,47 | 3,37 |
| **Trung bình thời gian** | 5,62 | 4,30 | 3,63 |

Bảng 4‑2 So sánh kết quả chạy 30 lần thuật toán nhân ma trận với ma trận giữa giải thuật tuần tự và song song với đơn vị tính bằng giây.

* Tính Speed up và Efficiency của giải thuật:
* Giữa thực hiện tuần tự và song song trên 3 máy

**Speed up** = = = 1,31(lần)

**Efficiency** = = = 43,55(%)

* Giữa thực hiện tuần tự và song song trên 4 máy

**Speed up** = = = 1,55(lần)

**Efficiency** = = = 38,7(%)

* Nhận xét: Khi tăng từ một máy chạy tuần tự lên ba máy chạy song song thì speed up = 1,31 lần nghĩa là tốc độ về mặt thời gian giữa chạy song song trên ba máy tăng gấp 1,31 lần so với chạy tuần tự. Và efficiency = 43,55% tức là khi thêm hai máy vào chạy song song thì mỗi máy đóng góp 43,55% sức mạnh của nó. Tương tự, khi chạy song song trên bốn máy thì speed up = 1,55 lần và efficiency = 38,7%. Qua đó, ta thấy không phải khi số máy thực hiện tăng thêm thì tốc độ và hiệu quả cũng tăng lên tuyến tính (tăng thêm một máy thì tăng gấp đôi, tăng thêm hai máy thì tăng gấp ba), mà nó sẽ chỉ tăng đến một lúc nào đó rồi sẽ không thay đổi, đúng như định luật Amdahl’s law.

# 4.3 Nhân ma trận với véc-tơ.

Tương tự nhân ma trận với ma trận, nhân ma trận với véc-tơ là dạng đặc biệt của nó, ứng với ma trận cấp n x m nhân và véc-tơ là ma trận cấp m x 1, kết quả là ma trận cấp n x 1

Bảng kết quả chạy 30 lần thuật toán nhân ma trận với véc tơ (với ma trận cấp 600 x 600 và véc-tơ là ma trận cấp 600 x 1) giữa giải thuật tuần tự và song song với đơn vị tính bằng giây.

|  |  |  |  |
| --- | --- | --- | --- |
| **Lần chạy thứ** | **Tuần tự** | **Song song** | |
| **3 máy** | **4 máy** |
| **1** | 0,221 | 0,143 | 0,110 |
| **2** | 0,232 | 0,128 | 0,107 |
| **3** | 0,256 | 0,140 | 0,111 |
| **4** | 0,289 | 0,135 | 0,113 |
| **5** | 0,250 | 0,146 | 0,116 |
| **6** | 0,231 | 0,127 | 0,122 |
| **7** | 0,212 | 0,158 | 0,110 |
| **8** | 0,237 | 0,132 | 0,108 |
| **9** | 0.295 | 0,124 | 0,113 |
| **10** | 0,287 | 0,148 | 0,110 |
| **11** | 0,254 | 0,125 | 0,119 |
| **12** | 0,263 | 0,187 | 0,111 |
| **13** | 0,267 | 0,198 | 0,112 |
| **14** | 0,232 | 0,142 | 0,106 |
| **15** | 0,244 | 0,158 | 0,103 |
| **16** | 0,265 | 0,164 | 0,111 |
| **17** | 0,276 | 0,122 | 0,111 |
| **18** | 0,246 | 0,173 | 0,123 |
| **19** | 0,281 | 0,120 | 0,109 |
| **20** | 0,225 | 0,158 | 0,118 |
| **21** | 0,236 | 0,121 | 0,120 |
| **22** | 0,218 | 0,138 | 0,110 |
| **23** | 0,220 | 0,144 | 0,115 |
| **24** | 0,243 | 0,153 | 0,116 |
| **25** | 0,265 | 0,164 | 0,112 |
| **26** | 0,237 | 0,192 | 0,105 |
| **27** | 0,248 | 0.168 | 0,112 |
| **28** | 0,221 | 0,137 | 0,110 |
| **29** | 0,238 | 0,145 | 0,116 |
| **30** | 0,210 | 0,129 | 0,123 |
| **Trung bình thời gian** | 0,254 | 0,147 | 0,074 |

Bảng 4‑3 So sánh kết quả chạy 30 lần thuật toán nhân ma trận với véc-tơ giữa giải thuật tuần tự và song song với đơn vị tính bằng giây.

* Tính Speed up và Efficiency của giải thuật:
* Giữa thực hiện tuần tự và song song trên 3 máy

**Speed up** = = = 1,73(lần)

**Efficiency** = = = 57,56(%)

* Giữa thực hiện tuần tự và song song trên 4 máy

**Speed up** = = = 2,23(lần)

**Efficiency** = = = 55,57(%)

* Nhận xét: Khi tăng từ một máy chạy tuần tự lên ba máy chạy song song thì speed up = 1,73 lần nghĩa là tốc độ về mặt thời gian giữa chạy song song trên ba máy tăng gấp 1,73 lần so với chạy tuần tự. Và efficiency = 57,56% tức là khi thêm hai máy vào chạy song song thì mỗi máy đóng góp 57,56% sức mạnh của nó. Tương tự, khi chạy song song trên bốn máy thì speed up = 2,23 lần và efficiency = 55,57%. Qua đó, ta thấy không phải khi số máy thực hiện tăng thêm thì tốc độ và hiệu quả cũng tăng lên tuyến tính (tăng thêm một máy thì tăng gấp đôi, tăng thêm hai máy thì tăng gấp ba), mà nó sẽ chỉ tăng đến một lúc nào đó rồi sẽ không thay đổi, đúng như định luật Amdahl’s law.

# KẾT LUẬN

Như vậy, sau khoảng thời gian nghiên cứu và thực hiện luận văn, em đã có được một số kết quả nhỏ sau:

+ Biết về lập trình song song là gì, ứng dụng của nó trong đời sống.

+ Đồng thời, biết thêm về các cơ cấu và các mô hình của lập trình song song.

+ Tìm hiểu cách thiết kế thuật toán song song, và các thư viện được sử dụng trong lập trình song song. Cụ thể, trong khóa luận này là thư viện MPI.

+ Cài đặt các giải thuật sử dụng lập trình song song như: giải thuật tính số pi, na trận nhân ma trận, ma trận nhân véc-tơ.

Hướng phát triển của luận văn:

- Nghiên cứu về chuẩn MPI 2, tìm hiểu OpenMP để kết hợp với MPI để làm tăng hiệu năng của xử lý song song.

- Nghiên cứu các bài toán lớn hơn và có ý nghĩa trong thực tế.

**PHỤ LỤC**

**1. Tính số Pi.**

**1.1 Code tuần tự.**

#include<stdio.h>

#include<conio.h>

#include<math.h>

double pi(double n)

{

double tong = 0;

double x;

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

{

x = (i - 1/2) / n;

tong = tong + (4/(1+(x\*x));

}

return 1/n \* tong;

}

void main()

{

double a;

printf("Nhap so chu so can sau dau phay thap phan:");

scanf\_s("%d",a);

printf("%.16lf\n", pi(a));

}

**1.2 Code song song.**

#include <stdio.h>

#include <math.h>

#include <mpi.h>

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

{

int n, myid, numprocs, i;

double PI25DT = 3.141592653589793238462643;

double mypi, pi, h, sum, x;

double startwtime = 0.0, endwtime;

MPI\_Init(&argc, &argv);

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

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

while (1)

{

if (myid == 0)

{

printf("Nhap so chu so can sau dau phay thap phan: (0 la thoat)");

scanf("%d",&n);

}

startwtime = MPI\_Wtime();/\*bắt đầu thời gian tính song song\*/

MPI\_Bcast(&n, 1, MPI\_INT, 0, MPI\_COMM\_WORLD);

if (n == 0)

break;

else

{

h = 1.0 / (double)n;

sum = 0.0;

for (i = myid + 1; i <= n; i += numprocs)

{

x = h \* ((double)i - 0.5);

sum += (4.0 / (1.0 + x\*x));

}

mypi = h \* sum;

MPI\_Reduce(&mypi, &pi, 1, MPI\_DOUBLE, MPI\_SUM, 0, MPI\_COMM\_WORLD);

if (myid == 0)

{

endwtime = MPI\_Wtime();/\*kết thúc thời gian tính toán song song\*/

printf("So pi xap xi %.16f, sai so la %.16f\n", pi, fabs(pi - PI25DT));

printf("Thoi gian thuc hien la = %f\n", endwtime-startwtime);

printf ("Ket thuc.\n");

}

}

}

MPI\_Finalize();

return 0;

}

**2. Ma trận nhân ma trận.**

**2.1 Code tuần tự.**

#include<stdio.h>

#include<conio.h>

#define NRA 600

#define NCA 600

#define NCB 600

void main()

{

int a[100][100], b[100][100], matrantich[100][100];

int i;

int j;

int k;

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

{

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

{

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

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

{

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

{

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

}

}

}

}

printf("Ket qua Ma tran:\n");

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

{

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

{

matrantich[i][j] = 0;

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

{

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

}

}

}

// hien thi ket qua

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

{

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

{

printf("%d\t", matrantich[i][j]);

}

printf("\n");

}

}

**2.2 Code song song.**

#include "mpi.h"

#include <stdio.h>

#include <stdlib.h>

#define NRA 600 /\* số dòng của ma trận A \*/

#define NCA 600 /\* số cột của ma trận A \*/

#define NCB 600 /\* số cột của ma trận B \*/

#define MASTER 0 /\* chỉ số tác vụ của task 1 \*/

#define FROM\_MASTER 1 /\* thể loại thông điệp \*/

#define FROM\_WORKER 2 /\* thể loại thông điệp \*/

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

{

int numtasks, /\* số tác vụ \*/

taskid, /\* chỉ số của tác vụ \*/

numworkers, /\* số công việc tương ứng với số tác vụ con \*/

source, /\* chỉ số tác vụ nguồn \*/

dest, /\* chỉ số tác vụ đích \*/

mtype, /\* thông điệp \*/

rows, /\* số dòng của ma trận A gửi tới mỗi tác vụ \*/

averow, extra, offset, /\* dùng để xác định số dòng gửi cho mỗi tác vụ \*/

i, j, k, rc; /\* biến để chạy vòng lặp \*/

int len;

char hostname[MPI\_MAX\_PROCESSOR\_NAME];

double a[NRA][NCA], /\* ma trận A được đem nhân \*/

b[NCA][NCB], /\* ma trận B được đem nhân \*/

c[NRA][NCB]; /\* kết quả là ma trận C \*/

double startwtime = 0.0, endwtime;

MPI\_Status status;

MPI\_Init(&argc,&argv);

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

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

MPI\_Get\_processor\_name(hostname, &len);

if (numtasks < 2 ) {

printf("Can it nhat 2 tien trinh MPI. Thoat...\n");

MPI\_Abort(MPI\_COMM\_WORLD, rc);

exit(1);

}

numworkers = numtasks-1;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Công việc của tác vụ chủ\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

if (taskid == MASTER)

{

printf("mpi\_mt bat dau voi %d tien trinh.\n",numtasks);

printf("A[%d][%d]\*B[%d][%d]=C[%d][%d]\n",NRA, NCA, NCA, NCB, NRA, NCB);

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

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

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

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

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

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

startwtime = MPI\_Wtime();/\*bắt đầu tính thời gian chạy song song\*/

/\* Gửi dữ liệu ma trận đến các tác vụ con \*/

averow = NRA/numworkers;

extra = NRA%numworkers;

offset = 0;

mtype = FROM\_MASTER;

for (dest=1; dest<=numworkers; dest++)

{

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

printf("Gui %d cot tu tien trinh %d offset=%d\n",rows,dest,offset);

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

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

MPI\_Send(&a[offset][0], rows\*NCA, MPI\_DOUBLE, dest, mtype,

MPI\_COMM\_WORLD);

MPI\_Send(&b, NCA\*NCB, MPI\_DOUBLE, dest, mtype, MPI\_COMM\_WORLD);

offset = offset + rows;

}

/\* Nhận kết quả từ các tác vụ con \*/

mtype = FROM\_WORKER;

for (i=1; i<=numworkers; i++)

{

source = i;

MPI\_Recv(&offset, 1, MPI\_INT, source, mtype, MPI\_COMM\_WORLD, &status);

MPI\_Recv(&rows, 1, MPI\_INT, source, mtype, MPI\_COMM\_WORLD, &status);

MPI\_Recv(&c[offset][0], rows\*NCB, MPI\_DOUBLE, source, mtype,

MPI\_COMM\_WORLD, &status);

printf("Nhan ket qua tu tien trinh %d\n",source);

}

endwtime = MPI\_Wtime();/\*dừng tính thời gian chạy song song\*/

/\* In kết quả \*/

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("Ket qua Ma tran:\n");

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

{

printf("\n");

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

printf("%6.2f ", c[i][j]);

}

printf("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("Thoi gian thuc hien la = %f\n", endwtime-startwtime);

printf ("Ket thuc.\n");

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Công việc của tác vụ con\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

if (taskid > MASTER)

{

mtype = FROM\_MASTER;

MPI\_Recv(&offset, 1, MPI\_INT, MASTER, mtype, MPI\_COMM\_WORLD, &status);

MPI\_Recv(&rows, 1, MPI\_INT, MASTER, mtype, MPI\_COMM\_WORLD, &status);

MPI\_Recv(&a, rows\*NCA, MPI\_DOUBLE, MASTER, mtype, MPI\_COMM\_WORLD, &status);

MPI\_Recv(&b, NCA\*NCB, MPI\_DOUBLE, MASTER, mtype, MPI\_COMM\_WORLD, &status);

printf("Tien trinh %d tren %s dang lam viec tren offset %d, rows %d:\n",taskid, hostname, offset,rows);

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

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

{

c[i][k] = 0.0;

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

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

}

mtype = FROM\_WORKER;

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

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

MPI\_Send(&c, rows\*NCB, MPI\_DOUBLE, MASTER, mtype, MPI\_COMM\_WORLD);

}

MPI\_Finalize();

}

**TÀI LIỆU THAM KHẢO**

**Tài liệu tiếng Việt:**

1. Nguyễn Hữu Hoàng Thọ (2006), *Kỹ thuật đường ống và cân bằng tải trong lập trình song song*, khóa luận tốt nghiệp.

**Tài liệu tiếng Anh:**

1. Ananth Grama, Anshul Gupta, George Karypis and Vipin Kumar (2003), Addison Wesley, *Introduction to Parallel Computing, Second Edition,* 856.
2. Michael J.Quinn, *Parallel Programming in C with MPI and OpenMP* (2003), Oregon State University, 516.

**Tài liệu website:**

1. <http://www.mpich.org/downloads/>
2. <https://www.vmware.com/go/downloadworkstation>
3. <http://vnreview.vn/tin-tuc-khoa-hoc-cong-nghe/-/view_content/content/1693055/ibm-nghien-cuu-mau-dien-tu-5d-de-tang-toc-chip-xu-ly>