|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI  **TRƯỜNG ĐIỆN - ĐIỆN TỬ**  logo_128  **BÁO CÁO BÀI TẬP LỚN LẬP TRÌNH SONG SONG**  **Đề tài: Parallelization of the Sieve of**  **Eratosthenes algorithm**  **Nhóm sinh viên thực hiện:**   |  |  | | --- | --- | | **Hà Việt Anh** | ***20203893*** | | **Bùi Vũ Bình Giang** | ***20203897*** | | **Nguyễn Đoàn Quang Huy**  **Đào Thị Hồng Loan**  **Vũ Xuân Minh** | ***20200278***  ***20203903***  ***20203905*** |   **Giảng viên hướng dẫn: PGS.TS.Phạm Doãn Tĩnh**  **Hà Nội , 1/2024** |

MỤC LỤC

[Lời mở đầu 3](#_heading=h.dx8097blnssa)

[Chương 1 Tổng quan về tính toán song song 4](#_heading=h.lvdaffiym524)

[1.1 Tính toán song song 4](#_heading=h.vshb3i73sho6)

[1.1.1. Tính toán song song là gì 4](#_heading=h.shq6b6bar5vi)

[1.1.2 .Tại sao phải tính toán song song 4](#_heading=h.v8pkic71q24m)

[1.2 Phân loại máy tính song song 5](#_heading=h.v5bcyi88cggl)

[1.2.1 Phân loại dựa trên sự tương tác giữa các BXL 5](#_heading=h.u0l1t6ubgarq)

[1.2.2 Phân loại dựa trên cơ chế điều khiển chung 8](#_heading=h.y5iefkpgldp)

[1.3 Các mô hình lập trình song song 9](#_heading=h.ulfr78rjvzqy)

[1.3.1 Tổng quan về mô hình lập trình song song 9](#_heading=h.67qp9b7g7fj0)

[1.3.2 Mô hình chia sẻ bộ nhớ chung 10](#_heading=h.uu8ka6qevfw5)

[1.3.3. Mô hình luồng 10](#_heading=h.uvnb0570wrhi)

[1.3.4 Mô hình truyền thông điệp 11](#_heading=h.9ndoe8xluwh7)

[1.3.5. Mô hình song song dữ liệu 12](#_heading=h.szjlzi2ugh8x)

[1.3.6. Mô hình lai 12](#_heading=h.u9z4hjq1bl7u)

[1.4 Hiệu năng của tính toán song song 13](#_heading=h.l36wlsrx8csf)

[1.4.1 Định luật Amdahl’s 13](#_heading=h.3nyz36lyiwpp)

[1.4.2 Cân bằng tải 14](#_heading=h.g2ghwer5tkwh)

[Chương 2. Lập trình song song với OpenMP 17](#_heading=h.1t3h5sf)

[2.1. Giới thiệu về OpenMP 17](#_heading=h.8m4lnn8hvfjf)

[2.1.1. Khái niệm cơ bản về OpenMP 17](#_heading=h.mmx9e3pkt70i)

[2.1.2. Lịch sử của OpenMP 17](#_heading=h.rdt4fpe22twd)

[2.1.3. Mục đích và ứng dụng của OpenMP 17](#_heading=h.si6u5z49jrco)

[2.2. Mô hình lập trình song song OpenMP 17](#_heading=h.kwvdy5yzvt5)

[2.2.1. Song song hóa dựa trên cơ chế luồng (Thread based parallelism) 17](#_heading=h.v1dwmfydlpcv)

[2.2.2. Mô hình song song hiện (Explicit Parallelism) 17](#_heading=h.d0rdwehprrah)

[2.2.3 Mô hình Fork-Join 18](#_heading=h.y9vo2dxsfef1)

[2.3. Các chỉ thị trong OpenMP 18](#_heading=h.ujx24s25vm0b)

[2.3.1. Khuôn dạng chỉ thị trong OpenMP 18](#_heading=h.l6i2hstjnq5n)

[2.3.2. Cấu trúc vùng song song 18](#_heading=h.qsmbcdjmynka)

[2.4. Cấu trúc chia sẻ công việc 19](#_heading=h.id07tvq4jkl)

[2.4.1. Chỉ thị DO/for 19](#_heading=h.h1chw0r9tp4z)

[2.4.2. Chỉ thị SECTIONS 20](#_heading=h.7p6od31a80h3)

[2.4.3. Chỉ thị SINGLE 20](#_heading=h.u7yaocybox35)

[Chương 3: Song song hóa thuật toán sàng Eratosthenes 22](#_heading=h.3rdcrjn)

[3.1.Ý tưởng của thuật toán sàng nguyên tố Eratosthenes 22](#_heading=h.cmfdpwi1e6ro)

[3.2.Thuật toán sàng nguyên tố Eratosthenes 22](#_heading=h.6p7t9s4xjg3q)

[3.3.Các bước chính trong bài toán song song hóa thuật toán 23](#_heading=h.k0swbq2zms7p)

[3.3.1. Tìm các số nguyên tố theo tuần tự 23](#_heading=h.9aqwy3dkx70n)

[3.3.2.Tìm các số nguyên tố theo song song 23](#_heading=h.uyjf1qq1e6yf)

[3.3.3 Kiểm tra thuật toán tìm số nguyên tố song song và tuần tự 24](#_heading=h.fnvhtw58yx8o)

[3.3.4.Ghi các số nguyên tố vào một tệp văn bản 25](#_heading=h.k3psievjbk5o)

[3.4.Kết quả thực nghiệm 26](#_heading=h.2a7nay3jixzu)

[Kết luận 28](#_heading=h.sw3m9up4vwo2)

[Tài liệu tham khảo 29](#_heading=h.abn5i0r9bbgt)

# 

# 

# Lời mở đầu

Thuật toán sàng Eratosthenes là một thuật toán được sử dụng để tìm tất cả các số nguyên tố nhỏ hơn hoặc bằng một số nguyên dương cho trước. Thuật toán này hoạt động bằng cách bắt đầu với một mảng chứa tất cả các số nguyên từ 2 đến số nguyên cho trước. Sau đó, thuật toán loại bỏ tất cả các số nguyên tố từ mảng bằng cách đánh dấu chúng là không nguyên tố. Cuối cùng, thuật toán trả về tất cả các số nguyên trong mảng mà chưa được đánh dấu là không nguyên tố.

Thuật toán sàng Eratosthenes là một thuật toán đơn giản và hiệu quả, nhưng nó có thể mất nhiều thời gian để chạy cho các số nguyên lớn. Trong những trường hợp này, việc song song hóa thuật toán có thể giúp cải thiện hiệu năng.

Mục tiêu của nghiên cứu này là đề xuất một phương pháp song song hóa thuật toán sàng Eratosthenes. Phương pháp này được thiết kế để tận dụng sức mạnh tính toán của các hệ thống máy tính song song.

Phần mở đầu của bài nghiên cứu đã giới thiệu về thuật toán sàng Eratosthenes, nhu cầu song song hóa thuật toán này và mục tiêu của nghiên cứu. Phần tiếp theo sẽ trình bày chi tiết phương pháp song song hóa được đề xuất.

# 

# Chương 1 Tổng quan về tính toán song song

## 1.1 Tính toán song song

### 1.1.1. Tính toán song song

Như chúng ta đã thấy các phần mềm phổ biến ngày nay hầu hết đều được viết trên cơ sở của tính toán tuần tự. Các phần mềm này thường được thực hiện trên một máy tính đơn với duy nhất một bộ xử lý. Vấn đề ở đây được giải quyết thông qua một chuỗi các lệnh tuần tự được thực hiện bởi một bộ xử lý. Tại một thời điểm chỉ có một lệnh được thực hiện.

Tính toán song song ra đời là một sự cải tiến của tính toán tuần tự. Nó là sự giải quyết vấn đề dựa trên sự thực thi đồng thời của nhiều tài nguyên máy tính . Tài nguyên máy tính đây bao gồm:

§ Một máy tính đơn với nhiều bộ xử lý

§ Nhiều máy tính nối lại với nhau thành một mạng máy tính

§ Kết hợp cả hai loại trên

Tính toán song song thường được dùng để giải quyết các vấn đề hết sức phức tạp yêu cầu thời gian tính toán lớn hoặc làm việc với khối dữ liệu lớn như các bài toán dự báo thời tiết, mô phỏng tai nạn xe hơi, xây dựng các mô hình thương mại và các vấn đề khoa học như khai phá dữ liệu , trí tuệ nhân tạo, an toàn dữ liệu…

### 1.1.2 .Tại sao phải tính toán song song

Việc tính toán song song là rất cần thiết. Ngoài hai nguyên nhân chính là nó được dùng để tính toán các bài toán yêu cầu thời gian tính toán lớn và khối lượng dữ liệu lớn còn có các nguyên nhân khác như để sử dụng tài nguyên của các máy khác trong một mạng LAN hoặc thông qua mạng internet, có thể sử dụng nhiều tài nguyên tính toán nhỏ kết hợp lại tạo nên một siêu máy tính. Do giới hạn về không gian lưu trữ của bộ nhớ trên một máy đơn để giải quyết một vấn đề lớn việc sử dụng nhiều bộ nhớ trên nhiều máy tính là rất hữu hiệu trong trường hợp này.

Giới hạn của tính toán tuần tự bao gồm cả hai nguyên nhân thực tế và nguyên nhân vật lý. Để xây dựng nên một máy tính tuần tự tốc độ cao gặp rất nhiều hạn chế

§ Về tốc độ truyền dữ liệu: Tốc độ truyền của máy tính tuần tự phụ thuộc trực tiếp vào sự di chuyển dữ liệu trong phần cứng. Cho nên việc tăng tốc độ thực hiện phải chủ yếu căn cứ vào các yếu tố tính toán.

§ Về kích cỡ: Công nghệ chế tạo bộ xử lý cho phép gắn nhiều bóng bán dẫn trên một con chip. Tuy nhiên việc làm này sẽ làm tăng kích thước của bộ xử lý

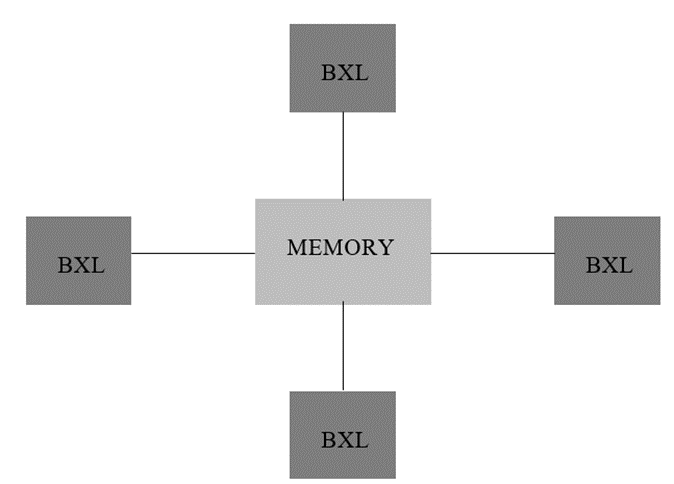
§ Về thương mại: Việc tạo ra một bộ xử lý tốc độ xử lý cao là rất tốn kém. Sử dụng nhiều bộ xử lý nhỏ đạt hiệu quả tương tự mà lại ít tốn kém hơn

## 1.2 Phân loại máy tính song song

### 1.2.1 Phân loại dựa trên sự tương tác giữa các BXL

Một trong những khía cạnh quan trọng của máy tinh song song là cơ chế trao đổi thông tin giũa các BXL.Có ba kiến trúc phổ biến nhất là kiến trúc chia sẻ bộ nhớ chung ( shared memory) kiến trúc bộ nhớ phân tán (distributed memory) và kiến trúc bộ nhớ lai(hybrit distributed-shared memory)

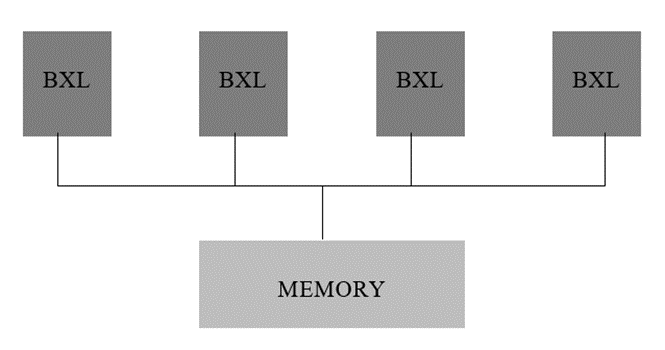
#### 1.2.1.1.Chia sẻ bộ nhớ chung

****

Hình 1.1: Máy tính song song chia sẻ bộ nhớ chung

Máy tính loại này sử dụng bộ nhớ chia sẻ toàn cục (global shared memory) mà tất cả các BXL đều có thể truy cập đến. Một BXL này có thể trao đổi thông tin với một BXL khác bằng cách ghi vào bộ nhớ toàn cục và BXL thứ hai sẽ đọc dữ liệu tại cùng vị trí đó trong bộ nhớ. Điều này cho phép trao đổi thông tin giữa các BXL.Tuy nhiên dẫn đến một vấn đề là đồng thời có nhiều BXL cùng truy cập tới cùng một vị trí trong bộ nhớ toàn cục. Máy tính loại này có hai loại chính dựa trên thời gian truy cập bộ nhớ

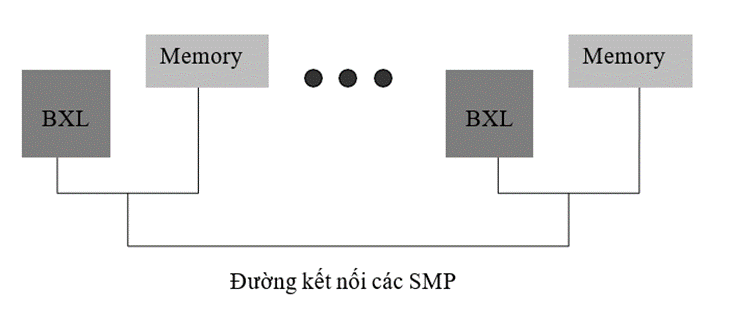
Thứ nhất là máy tính truy cập đồng bộ (UMA). Là loại máy tính với các BXL giống nhau. Tất cả các BXL đều có thể truy cập bộ nhớ đồng thời và thông qua một BUS dùng chung.



Hình 1.2: Máy tính Uniform Access Memory(UMA)

Máy tính loại này có loại gọi là Cache coheren-UMA (CC-UMA). Cache coheren ở đây có nghĩa là khi một BXL cập nhật một vị trí trong bộ nhớ thì tất cả các BXL khác đều nhận biết được sự cập nhật đấy

Thứ hai là máy tính truy cập không đồng bộ (NUMA) .Với máy tính loại này có một đường vật lý nối hai hay nhiều SMP lại với nhau.



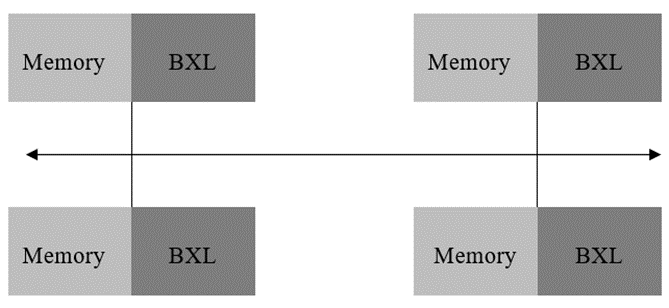
Hình 1.3: Máy tính Nun-Uniform Access Memory (NUMA)

Mỗi một SMP lại có thể truy cập tới bộ nhớ của SMP khác, tuy nhiên với kiến trúc kiểu này thì tất cả các BXL không thể truy cập cùng một lúc tới các bộ nhớ và với việc kêt nối các SMP bằng đường vật lý nên thời gian truy cập bộ nhớ chậm

Máy tính chia sẻ bộ nhớ chung có thuận lợi là giúp cho người lập trình thuận tiện khi viết các chương trình song song. Dữ liệu chia sẻ giữa các nhiệm vụ đảm bảo cả hai tiêu chuẩn nhanh và đồng thời. Tuy nhiên máy tính loại này có một số khó khăn là rất khó mở rộng số lượng các BXL vì việc thêm các BXL về phương diện hình học có thể làm tăng các đường kết nối giữa bộ nhớ toàn cục và các BXL. Đối với hệ thống Cache coheren thì làm tăng sự chuyền thông giữa cache và thiết bị quản lý bộ nhớ. Với máy tính loại này người lập trình phải chịu trách nhiệm đồng bộ chương trình để đảm bảo tính đúng đắn của dữ liệu dùng chung.

#### 1.2.1.2.Bộ nhớ phân tán

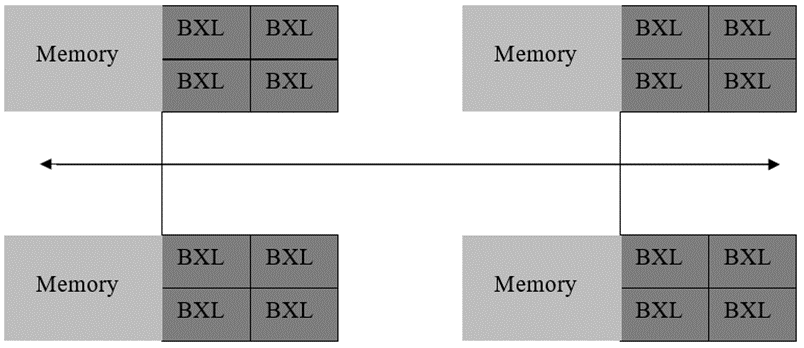
Ngược với máy tính chia sẻ bộ nhớ chung là máy tính với bộ nhớ phân tán trong đó không tồn tại bộ nhớ chia sẻ chung mà mỗi BXL có bộ nhớ cục bộ riêng của chúng. Trong máy tính song song có bộ nhớ phân tán các BXL liên lạc với nhau bằng các thông điệp (message) qua một mạng liên kết (interconnection network) gồm các liên kết truyền thông trực tiếp giữa một số cặp BXL.Một trong những lựa chọn quan trọng trong thiết kế lúc đó sẽ là các cặp BXL nào được nối với nhau.Tốc độ liên lạc là tối ưu khi các BXL được nối trực tiếp với nhau.Tuy nhiên điều này thường là không khả thi do số lượng các liên kết là quá lớn giẫn đến việc tăng giá thành của hệ thống. Cách thứ hai được sử dụng là các bộ sử lý liên lạc thông qua một BUS chia sẻ. Điều này dẫn đến việc độ trễ cao khi số lượng BXL lớn dẫn đến vấn đề tranh chấp BUS



Hình 1.4: Máy tính có bộ nhớ phân tán

#### 1.2.1.3.Máy tính với bộ nhớ lai

Hầu hết các máy tính nhanh và lớn ngày nay đều xây dựng dựa trên sự kết hợp giữa kiến trúc chia sẻ bộ nhớ chung và bộ nhớ phân tán. Sự kết hợp đó tạo nên một máy tính với tên gọi máy tính có bộ nhớ lai



Hình 1.5: Máy tính bộ nhớ lai

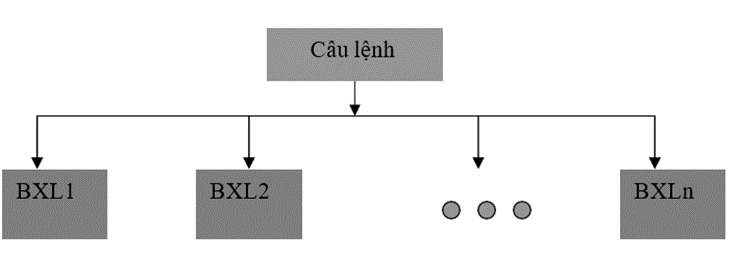
Các thành phần chia sẻ bộ nhớ chung trong máy tính bộ nhớ lai thường là các máy CC-SMP. Các BXL trong thành phần chia sẻ bộ nhớ chung có thể truy cập bộ nhớ toàn cục riêng của thành phần đó. Thành phần bộ nhớ phân tán được biết như là một mạng các SMP.Các SMP chỉ có thể truy cập đến bộ nhớ toàn cục trong thành phần chia sẻ bộ nhớ phân tán của chúng chứ không truy cập được bộ nhớ của các thành phần chia sẻ bộ nhớ chung khác. Cái mạng kết nối được xây dựng để chuyển dữ liệu từ SMP này đến SMP khác

### 1.2.2 Phân loại dựa trên cơ chế điều khiển chung

Phần lớn các máy tính song song thường có một cơ chế điều khiển chung song vấn đề đặt ra ở đây là các hoạt động của máy tính được điều khiển ở mức độ nào. Xem việc điều khiển theo hai khía cạnh khác nhau. Khía cạnh thứ nhất: Cơ chế điều khiển chung chỉ được sử dụng để nạp chương trình và dữ liệu vào các BXL còn sau đó các BXL hoạt động độc lập. Khía cạnh thứ hai: Cơ chế điều khiển được sử dụng để hướng dẫn các BXL các công việc phải làm tại mỗi bước. Giữa hai khía cạnh này là những cơ chế điều khiển trung gian. Hai loại cơ chế điều khiển phổ biến nhất là.

#### 1.2.2.1.Hệ thống đơn lệnh đa dữ liệu (SIMD)

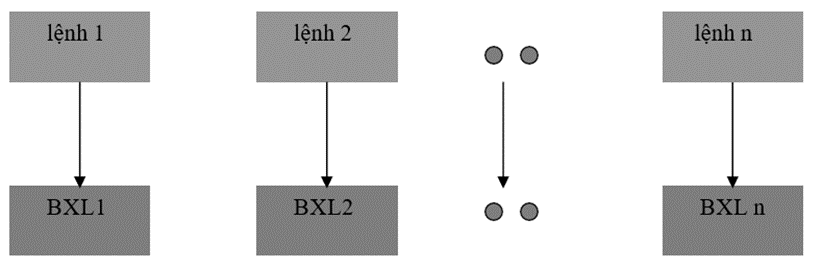
Các máy tính vector thuộc vào loại này. Mỗi máy tính vector có thể thực hiện một dòng lệnh. Tuy nhiên nó có nhiều BXL số học khác nhau mà mỗi BXL này có khả năng nạp và xử lý dữ liệu riêng của nó. Bởi vậy trong bất kỳ thời điểm nào một thao tác luôn ở cùng trạng thái thực thi trên nhiều đơn vị xử lý mà mỗi trong số chúng có thể xử lý dữ liệu riêng rẽ.



Hình 1.6: Hệ thống đơn lệnh đa dữ liệu(SIMD)

#### 1.2.2.2.Hệ thống đa lệnh đa dữ liệu (MIMD)

Phần lớn các máy tính đa xử lý hiện nay đều thuộc vào loại này. Trong các máy tính loại này nhiều dòng lệnh có thể thực hiện cùng một và mỗi dòng lệnh có thể xử lý dữ liệu riêng biệt. Các máy tính loại này ban đầu có rất ít tương tác giữa các BXL. Song hiện nay phần lớn các máy tính đều được thiết kế cho phép tương tác giữa các BXL được thực hiện một cách hiệu quả. Có thể liệt kê một số máy tính loại này như: Symmetry, TC2000, nCUBE2, Paragon XP/S và Connection Machine CM-5.



Hình 1.7: Hệ thống đa lệnh đa dữ liệu(MIMD)

## 1.3 Các mô hình lập trình song song

### 1.3.1 Tổng quan về mô hình lập trình song song

Việc đưa ra một mô hình máy tính chung cho việc lập trình giúp cho việc thiết kế giải thuật giải thuật trở nên đơn giản hơn. Lập trình song song đưa thêm những khó khăn mới vào mô hình lập trình tuần tự. Nếu chương trình được thực hiện ở mức thấp nhất thì không những số lệnh thực hiện là rất lớn mà nó còn phải quản lý trực tiếp quá trình thực hiện song song của hàng nghìn BXL và kết hợp hàng triệu tương tác liên BXL. Bởi vậy khả năng trừu tượng và tính toán module là các đặc tính rất quan trọng trong lập trình song song

Vậy mức độ trừu tượng nào sẽ phù hợp với lập trình song song. Các mô hình này cần cho phép đánh giá cụ thể về khả năng thực hiện đồng thời cũng như tính cục bộ để cho phép phát triển các chương trình có tính modul và có khả năng mở rộng.Và mô hình đó phải phù hợp với kiến trúc của máy tính song song.Các mô hình thông dụng bao gồm

§ Mô hình chia sẻ bộ nhớ chung

§ Mô hình luồng

§ Mô hình truyền thông điệp

§ Mô hình song song dữ liệu

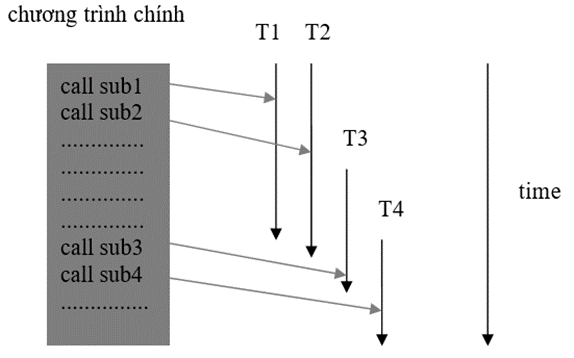
§ Mô hình lai

### 1.3.2 Mô hình chia sẻ bộ nhớ chung

Trong mô hình chia sẻ bộ nhớ chung các nhiệm vụ cùng chia sẻ một không gian địa chỉ chung có thể được truy cập đọc ghi theo phương thức không đồng bộ.Các cơ chế khác nhau như khóa (locks) và semaphore được điều khiển để truy cập đến bộ nhớ toàn cục.Xét theo quan điểm của lập trình viên thì ưu điểm của mô hình này là không có khái niệm sở hữu dữ liệu. Nghĩa là không phải chỉ định rõ ràng quá trình truyền dữ liệu giữa nhiệm vụ gửi và nhiệm vụ nhận dữ liệu. Tính chất này giúp cho phát triển các chương trình đơn giản hơn. Tuy nhiên khi đó việc hiểu và đảm bảo tính cục bộ trở nên khó khăn và cũng được chú ý nhiều nhất trong kiến trúc chia sẻ bộ nhớ chung. Việc viết các chương trình xác định cũng trở nên khó khăn

### 1.3.3. Mô hình luồng

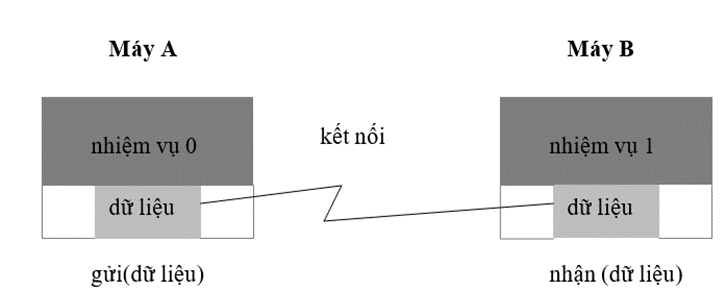
Trong mô hình luồng chương trình chính được chia thành các nhiệm vụ. Mỗi nhiệm vụ được thực hiện bởi các luồng một cách đồng thời. Mỗi luồng có dữ liệu riêng của nó và chia sẻ dữ liệu toàn cục của chương trình chính. Các nhiệm vụ đưa cho mỗi luồng là các thủ tục con của chương trình chính. Và bất kì luồng nào cũng có thể thực hiện bất kì thủ tục con nào tại cùng thời điểm với các luồng khác.Trong mô hình luồng các luồng kết nối với nhau thông qua bộ nhớ toàn cục với việc kết nối này thì chương trình phải được xây dựng một cách đồng bộ để tránh cùng một lúc có nhiều luồng cùng cập nhập một vị trí trong bộ nhớ toàn cục



Hình 1.8: Mô hình luồng

### 1.3.4 Mô hình truyền thông điệp

Trong mô hình truyền thông điệp chương trình song song được chia thành các nhiệm vụ. Mỗi nhiệm vụ sử dụng bộ nhớ cục bộ của nó. Các nhiệm vị này có thể được cư trú trên các máy vật lý giống nhau kết nối với nhau qua mạng với số lượng tùy ý

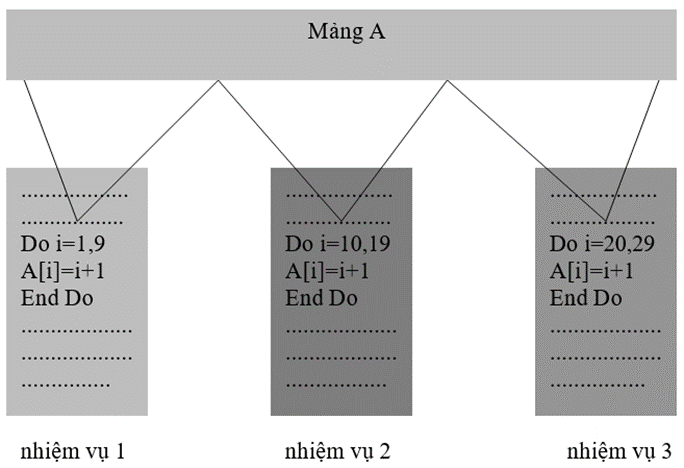


Hình 1.9: Mô hình truyền thông điệp

Các nhiệm vụ trao đổi dữ liệu với nhau qua hai phương thức gửi và nhận thông điệp. Xét trên khía cạnh lập trình thì các thông điệp chứa trong một thư viện thông điệp.Thư viện này phải được gắn vào mã nguồn của chương trình song song.MPI là một thư viện ngày nay được dùng rất phổ biến

### 1.3.5. Mô hình song song dữ liệu

Mô hình lập trình song song dữ liệu giúp lập trình các chương trình song song được thực hiện trên một tập dữ liệu lớn. Tập dữ liệu ở đây thường được xắp xếp theo một cấu trúc nhất định như là mảng hoặc theo khối



Hình 1.10: Mô hình lập trình song song dữ liệu

Với mô hình này thì các nhiệm vụ của chương trình làm việc với cùng một cấu trúc dữ liệu. Tuy nhiên mỗi nhiệm vụ sẽ làm việc trên từng phân vùng khác nhau của dữ liệu và các nhiệm vụ phải thưc hiện các thao tác giống nhau.

Trong kiến trúc chia sẻ bộ nhớ chung thì tất cả các nhiệm vụ truy cập vào cấu trục dữ liệu thông qua bộ nhớ toàn cục. Còn đối với kiến trúc bộ nhớ phân tán thì dữ liệu được chia ra và lưu trữ trên các bộ nhớ cục bộ của các BXL

### 1.3.6. Mô hình lai

Mô hình lai là sự kết hợp của hai hay nhiều mô hình lập trình song song kết hợp lại với nhau

Hiện nay thì mô hình lai phổ biến nhất là mô hình kết hợp giữa mô hình truyền thông điệp với mô hình luồng hoặc với mô hình chia sẻ bộ nhớ chung. Một mô hình lai khác nữa là sự kết hợp giữa mô hình song song dữ liệu với mô hình truyền thông điệp. Mô hình dạng này rất thuận tiện vì mô hình song song dữ liệu trên kiến trúc bộ nhớ phân tán sử dụng message passing để trao đổi dữ liệu giữa các nhiệm vụ một cách trong suốt đối với lập trình viên song song

### 

## 1.4 Hiệu năng của tính toán song song

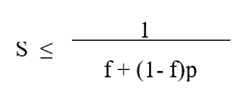
Trong phần này chúng ta sẽ trình bày một số vấn đề liên quan đến hiệu năng của tính toán song song bao gồm: khả năng tăng tốc độ tính toán,cân bằng tải (Load balancing) và sự bế tắc (Deadlock)

### 1.4.1 Định luật Amdahl’s

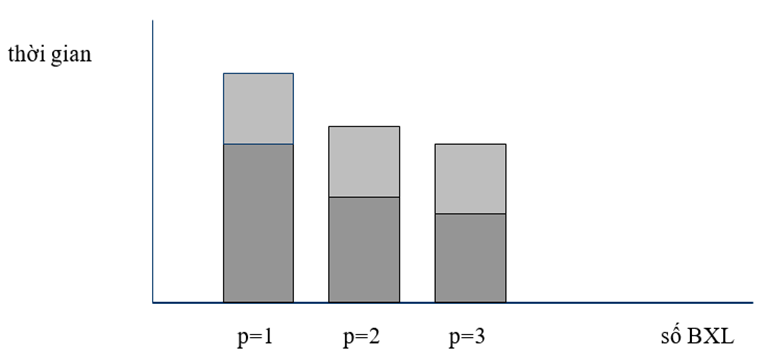
Trong nhiều ứng dụng thực tế đòi hỏi thời gian thực,vấn đề cần giải quyết có kính thước cố định,do đó khối lượng công việc phải làm cũng thường xác định được trước. Định luật do Amdahl phát biểu năm 1967 nhằm đánh giá hiệu năng của việc tính toán cho các bài toán thuộc loại này.

Khi tăng số lượng BXL trong máy tính song song, khối lượng công việc được được phân phối cho nhiều BXL thực hiện. Mục tiêu chính là tìm được kết quả của bài toán nhanh nhất có thể hay nói một cách khác là giảm đến mức tối đa thời gian tính toán.

Định luật Amdahl: Gọi f là phần nhỏ của thao tác tính toán trong quá trình tính toán phải thực hiện một cách tuần tự, 0 ≤ f ≤ 1. Tốc độ tối đa S có thể đạt được bằng cách sử dụng máy tính song song với p BXL được cho bởi công thức



Thời gian cho phần việc xử lý song song của ứng dụng sẽ dảm dần đến 0 khi ta tăng số lượng BXL. Thời gian cho việc xử lý tuần tự luôn là hằng số



Hình 1.11: Sự phụ thuộc thời gian vào số lượng BXL của đinh luật Amlahl

### 1.4.2 Cân bằng tải

Ta giả sử rằng nếu dữ liệu được phân tán trên các bộ nhớ địa phương của các BXL. Khi đó khối lượng công việc của các BXL cần phải được phân phối hợp lý trong suất quá trình tính toán. Trong nhiều trường hợp , giả sử này là đúng tuy nhiên trên thực tế điều này không phải lúc nào cũng thực hiện được. Giải pháp được đưa ra ở đây là cân bằng tải động nhằm mục đích làm thay đổi sự phân phối khối lượng công việc giữa các BXL trong quá trình thực hiện tính toán

Thông thường sau khi phân phối khối lượng công việc cho mỗi BXL, quá trình cân bằng tải động thực hiện theo bốn bước cơ bạn dưới đây Giám sát hiệu năng của mỗi BXL, trao đổi thông tin trạng thái giữa các BXL, tính toán và ra quết định phân phối lại khối lượng công việc và cuối cùng là thực hiện việc chuyển đổi dữ liệu thật sự

Để thực hiện điều này có rất nhiều thuật toán được thực hiện để cân bằng tải động được đề xuất. Theo kết quả Znstietal phân lớp các thuật toán này theo chiến lược tập trung, phân tán hòa toàn (Fully distributed) và phân tán một nửa(Semi – distributed)

#### 1.4.2.1.Các thuật toán cân bằng tải tập trung

Nhằm đưa ra quyết định có tính chất tổng thể trong việc phân phối lại khối lượng công việc cần thực hiện cho các BXL. Một vài thuật toán trong lớp này sử dụng thông tin hệ thống có tính chất toàn cục để lưu trạng thái của các máy riêng biệt trong hệ thống. Thông tin này sẽ cho phép thuật toán phân phối công việc cho các BXL một cách dễ dàng. Tuy nhiên khối lượng công việc tăng theo tỉ lệ thuận với số lượng các BXL, do đó đòi hỏi khối lượng lớn bộ nhớ trên một BXL để lưu trữ thông tin trạng thái. Vì vậy các thuật toán thuộc lớp này không được áp dụng một cách rông rãi .

#### 1.4.2.2.Các thuật toán cân bằng tải phân tán hoàn toàn

Trong chiến lược này, mỗi BXL có một bản sao về thông tin trạng thái của hệ thống . Các BXL trao đổi thông tin trạng thái với nhau và sử dụng các thông tin này để làm thay đổi một cách cục bộ việc phân chia công việc. Tuy nhiên các BXL chỉ có thông tin trạng thái cục bộ nên việc cân bằng tải không tốt bằng các thuật toán cân bằng tải tập trung

#### 1.4.2.3.Các thuật toán cân bằng tải phân tán một nửa

Các thuật toán thuộc lớp này chia các BXL thành từng miền. Trong mỗi miền sử dụng thuật toán cân bằng tải tập trung để phân phối công việc cho các BXL thuộc miền đó.

#### 1.4.2.4.Sự bế tắc(Deadlock)

Các tiến trình bị rơi vào tình trạng bế tắc nếu mỗi tiến trình đó nắm giữ tài nguyên mà một vài tiến trình khác yêu cầu sử dụng nó để xử lý.

Lý do tồn tại sự bế tắc là do nhiều tiến trình cùng sử dụng tài nguyên chung mà không có sự kiểm soát tốt. Sự bế tắc tồn tại trong các hệ điều hành đa nhiệm, cũng như các hệ thông đa BXL và đa máy tính

Đối với các hệ thống đa máy tính, một trong các sự bế tắc phổ biến là bế tắc vùng đệm (buffer deadlock) xẩy ra khi một tiến trình đợi một thông điệp mà thông điệp này có thể không bao giờ nhận được khi mà vùng đệm của hệ thống đã bị đầy

Xem xét hệ thống đa máy tính với các BXL xử lý không đồng bộ . BXL Pi gửi thông điệp cho BXL Pj không kết nối cho tới khi có thao tác thông điệp đó. Mặt khác BXL Pi gửi thông điệp cho BXL Pj nội dung của thông điệp được lưu trong vùng đệm của hệ thống cho đến khi BXL Pj nhận và đọc thông điệp. Giả sử rằng trong cùng một thời điểm có nhiều BXL cùng gửi thông điệp đến BXL Pj và điều này sẽ làm cho vùng đệm bị đầy. Việc gửi thông điệp tiếp theo chỉ được thực hiện khi BXL Pj đọc một hay nhiều thông điệp

Giả sử BXL Pk là một trong những BXL có khả năng gửi thông điệp đến BXL Pj. Nếu BXL Pj cố gắng đọc thông điệp do BXL Pk gửi đến nó sẽ bị kết khối cho đến khi nội dung thông điệp có trong vùng đệms. Rõ ràng BXL Pk bị kết khối cho tới khi BXL Pj loại bỏ một hay nhiều thông điệp từ vùng đệm như vậy BXL Pj và Pk rơi vào bế tắc.



Hình 1.12: Pk kết khối để gửi X cho Pj vì vùng đệm Pj bị đầy nên Pj không thể nhận được X . Pk và Pj rơi vào bế tắc

Bốn điều kiện gây nên bế tắc

**1.** Sự loại trừ lẫn nhau: Mỗi tiến trình có sự độc quyền khi sử dụng tài nguyên của nó

**2.** Không có sự ưu tiên: Mỗi tiến trình không bao giờ giải phóng tài nguyên mà tiến trình đó đang chiếm giữ cho đến tận khi không còn sử dụng chúng nữa

**3.** Sự chờ đợi tài nguyên: Mỗi tiến trình đang chiếm giữ tài nguyên trong khi lại chờ đợi các tiến trình khác giải phóng tài nguyên của chúng

**4.** Sự chờ đợi giữa các tiến trình: Tiến trình đợi tài nguyên mà tiến trình kế tiếp đang chiếm dữ mà tài nguyên đó không được giải phóng

Một số cách khắc phục sự bế tắc

Cách thứ nhất ta sử dụng là cố gắng dò tìm sự bế tắc khi chúng sẩy ra và khôi phục lại. Một cách khác để tránh sự bế tắc thông qua sử dụng các thông tin yêu cầu tài nguyên của các tiến trình để điều khiển sự phân phối để khi tiếp tục phân phối các tài nguyên không là nguyên nhân để các tiến trình rơi vào bế tắc. Cách thứ ba là ngăn cấm không để xảy ra đồng thời ba điều kiện cuối trong bốn điều kiện này sinh bế tắc.

# 

# Chương 2. Lập trình song song với OpenMP

## 2.1. Giới thiệu về OpenMP

### 2.1.1. Khái niệm cơ bản về OpenMP

OpenMP là một giao diện lập trình ứng dụng (API) được sử dụng để điều khiển các luồng trên cấu trúc chia sẻ bộ nhớ chung.

Thành phần của OpenMP bao gồm :

* Các chỉ thị biên dịch (Compiler Directives)
* Các thư viện runtime (Runtime Library Routines)
* Các biến môi trường (Emviroment Variables) .

### 2.1.2. Lịch sử của OpenMP

Năm 1994: chuẩn ANSI X3H5 ra đời.

Năm 1997: chuẩn OpenMP được đưa ra để thay thế chuẩn ANSI X3H5.

OpenMP được thiết kế có tính mở rộng, có các chỉ thị biên dịch nhằm hỗ trợ việc song song hóa.

### 2.1.3. Mục đích và ứng dụng của OpenMP

OpenMP ra đời với mục tiêu cung cấp một chuẩn chung cho rất nhiều kiến trúc và nền tảng phần cứng. Nó thiết lập một tập các chỉ thị biên dịch hỗ trợ việc lập trình song song trên máy tính chia sẻ bộ nhớ chung. Một mức song song chính thường được thực thi với ba đến bốn chỉ thị. OpenMP ra đời giúp cho việc lập trình song song một cách dễ dàng, nó cung cấp khả năng song song hóa chương trình tuần tự mà không dùng đến thư viện thông điệp v.v...

Ứng dụng của OpenMP:

* Giải quyết các vấn đề giới hạn về thời gian như: bài toán dự báo thời tiết,...
* Mô phỏng các vấn đề thực tế như: bài toán mô phỏng tai nạn xe hơi,...
* Giải quyết các bài toán khoa học yêu cầu khối lượng tính toán lớn như: bài toán mô phỏng N-Body, dự báo thời tiết …

## 2.2. Mô hình lập trình song song OpenMP

### 2.2.1. Song song hóa dựa trên cơ chế luồng (Thread based parallelism)

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.

### 2.2.2. Mô hình song song hiện (Explicit Parallelism)

Mô hình trên là một mô hình lập trình không tự động. Người lập trình có quyền điều khiển việc song song hóa một cách độc lập

### 2.2.3 Mô hình Fork-Join

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

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

## 2.3. Các chỉ thị trong OpenMP

### 2.3.1. Khuôn dạng chỉ thị trong OpenMP

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
* newlin : Yêu cầu bắt buộc với mỗi chỉ thị nó là tập mã lệnh nằm trong khối cấu trúc được bao bọc bởi chỉ thị a
* 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 thủ 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

### 2.3.2. Cấu trúc vùng song song

Một vùng song song là một khối mã nguồn được thực thi bởi nhiều luồng. Trong C/C++ một vùng song song có định dạng như sau:

#pragma omp parallel [clause...] newline

if (scalar\_expression)

private (list)

shared (list)

default (shared | none)

firstprivate (list)

reduction (operator : list)

copyin (list)

structured\_block

Khi mà một luồng gặp chỉ thị PARALLEL thì nó sẽ tạo ra một tập các luồng và luồng ban đầu sẽ là luồng chủ của tập các luồng đó. Luồng chủ ở đây cũng là một thành viên trong tập các luồng đó và là luồng số 0 .Để bắt đầu thực hiện một vùng song song thì đoạn mã nguồn trong vùng song song được sao ra những bản giống nhau đưa cho mỗi luồng thực hiện một cách song song. Đợi cho đến khi tất cả các luồng đều thực hiện song công việc của mình thì luồng chủ sẽ thực hiện công việc tuần tự còn lại ngoài vùng song song đó.

## 2.4. Cấu trúc chia sẻ công việc

Cấu trúc chia sẻ công việc dùng để chia việc thực hiện công việc trong vùng song song cho các 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:

* Cấu trúc DO/for
* Cấu trúc SECTIONS
* Cấu trúc SINGLE

### 2.4.1. 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

for\_loop Mệnh đề

### 2.4.2. 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

{

#pragma omp section newline

structured\_block

#pragma omp section newline

structured\_block

### 2.4.3. Chỉ thị SINGLE

Mệnh đề SINGLE chỉ ra rằng đoạn mã bao quanh chỉ thị chỉ được thực hiện bởi một luồng trong tập các luồng.

Trong C/C++ chỉ thị SINGLE được cho dưới dạng sau:

#pragma omp sections [clause...] newline

private(list)

firstprivate(list)

nowait

Structure\_block

Các luồng khác mà không thực thi đoạn mã trong chỉ thị SINGLE sẽ phải đợi đến khi luồng thực thi đoạn mã trong chỉ thị kết thúc mới được thực hiện các công việc ngoài chỉ thị SINGLE nếu không có mệnh đề NOWAIT được đưa ra.

# 

# Chương 3: Song song hóa thuật toán sàng Eratosthenes

## 3.1.Ý tưởng của thuật toán sàng nguyên tố Eratosthenes

Dựa theo lý thuyết về số nguyên tố: Một số nguyên tố là số chỉ có 2 ước là 1 và chính nó. Do vậy, nếu ta xác định được số x là số nguyên tố, ta có thể kết luận mọi số chia hết cho x đều không phải số nguyên tố. Do đó ta đã loại bỏ được rất nhiều số mà không cần kiểm tra.

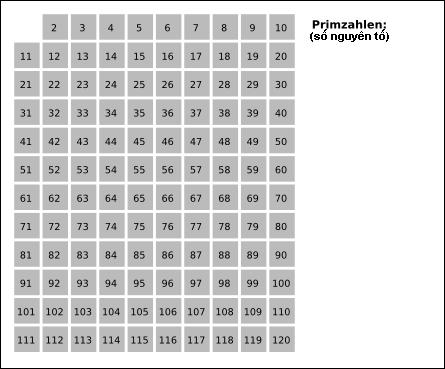
Ví dụ:

Số 2 là số nguyên tố => các số 4, 6, 8, 10, ... không phải số nguyên tố.

Số 3 là số nguyên tố => các số 9, 15, 21, ... không phải số nguyên tố. (Do 6, 12, 18 đã bị loại ở số 2)

## 3.2.Thuật toán sàng nguyên tố Eratosthenes

1. Tạo mảng đánh dấu cho tất cả các phần tử từ 2 đến N và mặc định tất cả đều là số nguyên tố
2. Xét số đầu tiên tìm được là số nguyên tố – giả sử x, đánh dấu tất cả các ước của x: 2x, 3x, 4x,… trong đoạn [x, N] không phải số nguyên tố.
3. Tìm số tiếp theo được đánh dấu là số nguyên tố trong [x, N]. Nếu không còn số nào, thoát chương trình. Nếu còn, gán nó bằng x và lặp lại bước 2.
4. Khi kết thúc giải thuật, các số không bị đánh dấu là các số nguyên tố



Mô tả thuật toán sàng nguyên tố Eratosthenes

## 

## 

## 3.3.Các bước chính trong bài toán song song hóa thuật toán

### 3.3.1. Tìm các số nguyên tố theo tuần tự

void SerialFindPrimes(long long int n, bool \*a){

double m;

// Khởi tạo tất cả các số [2...N] đều là số nguyên tố

for (long long int i = 2; i <= n; i++)

a[i] = true;

// Thuật toán sàng nguyên tố

// Nếu một số là số nguyên tố, thì tất cả các bội của nó không phải số nguyên tố

m = sqrt(n);

for (long long int i = 2; i <= m; i++)

if (a[i] == true)

for (int j = i \* i; j <= n; j = j + i)

a[j] = false;

}

### 3.3.2.Tìm các số nguyên tố theo song song

void ParallelFindPrimes(long long int n, bool \*a)

{

long long int m;

#pragma omp parallel for

for (long long int i = 2; i <= n; i++)

a[i] = true;

m = sqrt(n);

#pragma omp parallel for schedule(dynamic, 1) shared(a)

for (long long int i = 2; i <= m; i++)

{

if (a[i] == true)

for (long long int j = i \* i; j <= n; j = j + i)

a[j] = false;

}

}

Vòng lặp thứ nhất :

Cụ thể, mỗi thread sẽ được giao một phần công việc của phạm vi từ 2 đến n để thực hiện gán giá trị true cho tất cả các phần tử trong mảng a

Vòng lặp thứ 2 :

* Một giá trị m được tính bằng căn bậc hai của n. Đây là giá trị tối đa mà chúng ta cần kiểm tra để tìm các số nguyên tố trong khoảng từ 2 đến n.
* Sau đó, sử dụng #pragma omp parallel for schedule(dynamic, 1) shared(a) để tạo các luồng song song.
* Mỗi luồng sẽ kiểm tra từng số i từ 2 đến m. Nếu a[i] là true (tức là i là số nguyên tố), luồng này sẽ đánh dấu tất cả các bội số của i lớn hơn hoặc bằng i\*i và nhỏ hơn hoặc bằng n trong mảng a là false, chỉ ra rằng chúng không phải là số nguyên tố.

Mục tiêu của hàm này là sử dụng song song hóa để tối ưu hóa việc đánh dấu các số nguyên tố trong mảng a, nhưng vẫn đảm bảo tính chính xác bằng cách sử dụng kiểm tra tuần tự cho các số nguyên tố từ 2 đến m.

### 3.3.3 Kiểm tra thuật toán tìm số nguyên tố song song và tuần tự

void TestResult(long long int n, bool \*a)

{

bool \*b;

b = new bool[n + 1];

bool check = true;

SerialFindPrimes(n, b);

for (long long int i = 2; i <= n; i++)

{

if (a[i] != b[i])

{

check = false;

break;

}

}

if (check == false){

cout << "\nThe results of serial and parallel algorithms "

"are NOT identical. Check your code.";

}

else{

cout << "\nThe results of serial and parallel algorithms are "

"identical.";

}

delete[] b;

}

Trong đoạn code trên:

* Hàm tạo một mảng boolean b mới với cùng kích thước như mảng a.
* Sử dụng hàm SerialFindPrimes để tìm số nguyên tố từ 2 đến n và lưu kết quả vào mảng b.
* Tiến hành so sánh từng phần tử của hai mảng a và b. Nếu phát hiện bất kỳ sự khác biệt nào giữa hai mảng (tức là một số nguyên tố không được đánh dấu đúng), biến check sẽ được gán giá trị false và vòng lặp sẽ dừng lại ngay lập tức.
* Kết quả cuối cùng dựa trên giá trị của biến check. Nếu check vẫn là true sau khi so sánh, nghĩa là hai thuật toán đã tạo ra kết quả giống nhau, và thông báo sẽ được in ra màn hình. Ngược lại, nếu có sự khác biệt, thông báo sẽ báo cáo rằng hai thuật toán không tạo ra kết quả giống nhau.
* Cuối cùng, mảng b được giải phóng bộ nhớ sau khi kiểm tra xong. Điều này giúp tránh rò rỉ bộ nhớ và giữ cho việc quản lý bộ nhớ trở nên tốt hơn.

### 3.3.4.Ghi các số nguyên tố vào một tệp văn bản

void PrintPrimeInFile(long long int n, bool \*a, string FileName)

{

long long int count = 1, i;

fstream out;

out.open(FileName, ios::out);

out << "\nPrime numbers less than or equal " << n << ": \n";

#pragma omp parallel for ordered

for (i = 2; i <= n; i++)

{

#pragma omp ordered

if (a[i] == true)

{

if (count % 10 == 0)

{

out << "\n";

}

count++;

out << i << "\t";

}

}

out.close();

}

“#pragma omp parallel for ordered” là một chỉ thị trong OpenMP được sử dụng để song song hóa vòng lặp for và duy trì thứ tự của các lần lặp bên trong các block ordered.

* #pragma omp parallel for: Chỉ thị song song hóa vòng lặp for, cho phép các lần lặp của vòng for được thực hiện song song trên nhiều luồng.
* ordered: Chỉ thị này kết hợp với #pragma omp parallel for để đảm bảo rằng các lần lặp trong vòng for được thực hiện theo thứ tự. Nó đảm bảo rằng các kết quả xuất phát từ các lần lặp này sẽ được ghi ra theo thứ tự chính xác.

Khi sử dụng cả hai chỉ thị này cùng nhau, việc thực hiện song song vẫn đảm bảo rằng kết quả xuất phát từ các lần lặp được ghi ra theo thứ tự mà chúng được tạo ra, không làm thay đổi thứ tự này.

3.3.5.Tính toán thời gian thực hiện tuần tự , song song và đo hiệu suất

Start = GetTime();

SerialFindPrimes(n, Result\_Serial);

Finish = GetTime();

Duration\_Serial = Finish - Start;

Start = GetTime();

ParallelFindPrimes(n, Result\_Parallel);

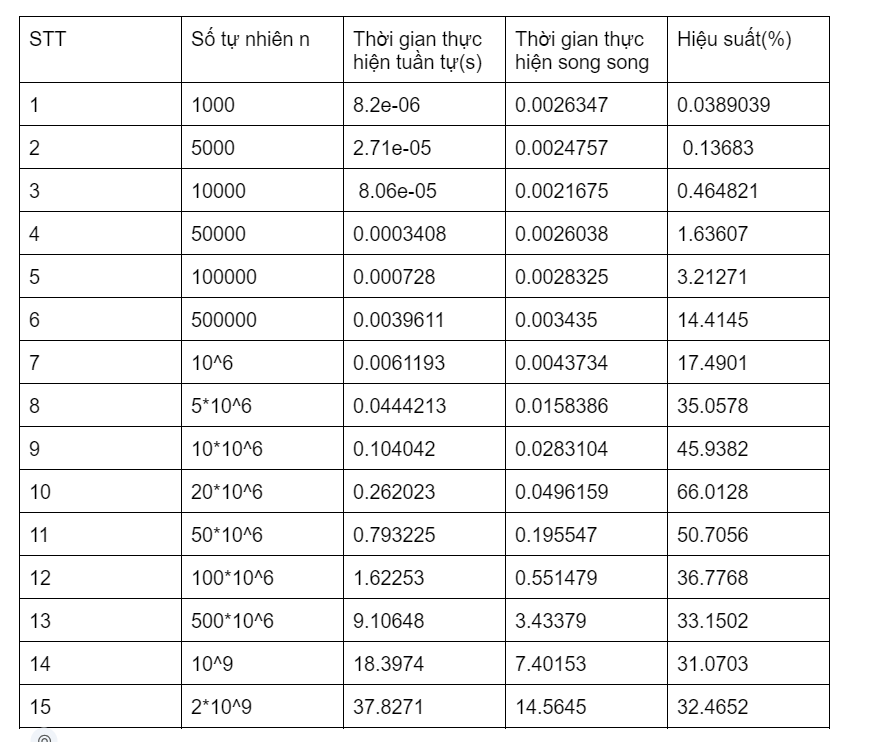
Finish = GetTime();

Duration\_Parallel = Finish - Start;

Efficiency = Duration\_Serial / (Duration\_Parallel \* p);

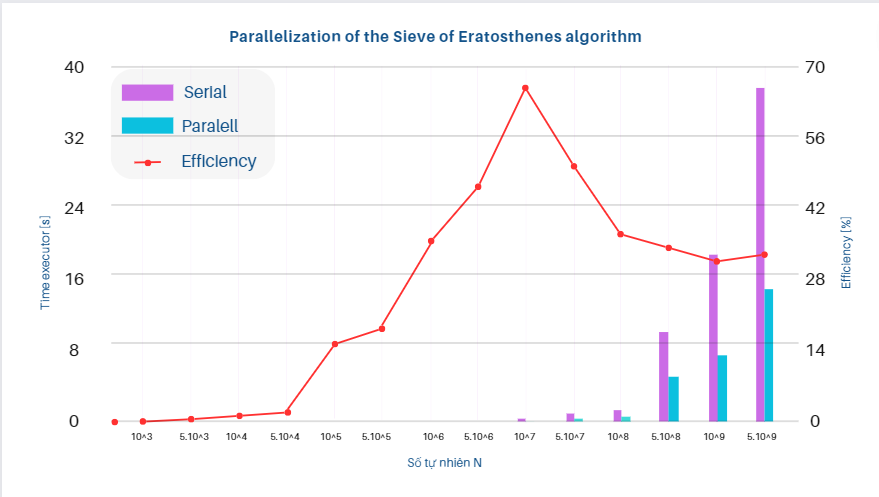
## 3.4.Kết quả thực nghiệm

#### 3.4.1.Bảng kết quả thực nghiệm



Bảng kết quả thực nghiệm trên cho thấy khi số lần lặp ít tính toán tuần tự có thời gian chạy ít hơn tính toán song song, đồng nghĩa với đó là hiệu suất thấp. Và ngược lại khi số vòng lặp lớn, thời tính toán song song chạy ít hơn rất nhiều so với tính toán tuần tự .

#### 3.4.2.Đồ thị biểu diễn thời gian và hiệu suất



Hiệu suất tính toán được tăng lên khi số vòng lặp tăng lên,hiệu suất đạt đỉnh điểm khi n bằng 2\*10^7 ,nhưng giảm dần khi số tự nhiên n tăng lên .

# 

# Kết luận

Trong bài nghiên cứu này, chúng em đã đề xuất một phương pháp song song hóa thuật toán sàng Eratosthenes. Phương pháp này được thiết kế để tận dụng sức mạnh tính toán của các hệ thống máy tính song song.

Phương pháp song song hóa được thực hiện dựa trên nguyên tắc chia để trị. Chúng em chia mảng chứa các số nguyên thành các phần nhỏ hơn và phân công cho các tiến trình song song xử lý từng phần. Các tiến trình sẽ phối hợp với nhau để loại bỏ các số nguyên tố khỏi mảng.

Chúng em đã thực hiện phân tích hiệu năng của phương pháp được đề xuất. Kết quả cho thấy phương pháp này có thể cải thiện hiệu năng của thuật toán sàng Eratosthenes đáng kể cho các số nguyên lớn.

Về phía trước, chúng em có thể tiếp tục nghiên cứu để cải thiện hiệu năng của phương pháp song song hóa được đề xuất. Một số hướng nghiên cứu có thể bao gồm:

* Sử dụng các kỹ thuật song song hóa hiệu quả hơn, chẳng hạn như kỹ thuật chia để trị đồng thời.
* Tối ưu hóa việc phân phối các phần mảng cho các tiến trình song song.
* Sử dụng các kỹ thuật giảm thiểu giao tiếp giữa các tiến trình.

Chúng em tin rằng các nghiên cứu tiếp theo sẽ giúp cải thiện hiệu năng của phương pháp song song hóa thuật toán sàng Eratosthenes hơn nữa.

Chúng em xin chân thành cảm ơn thầy Phạm Doãn Tĩnh vì đã cung cấp cho chúng em tài liệu hữu ích trong quá trình thực hiện bài tập lớn. Nhờ tài liệu mà thầy cung cấp cũng như tài liệu nhóm tìm trên Internet, chúng em đã có cơ hội được tiếp cận và nghiên cứu những nguồn thông tin chính xác và đáng tin cậy. Từ đó, chúng em đã tích lũy được nhiều kiến thức mới và đưa ra những ý kiến đóng góp xây dựng trong báo cáo của nhóm.

# Tài liệu tham khảo

[1] Ruud van der Pas , “Mastering OpenMP Performance”

[2]Tim Mattson, A Hands-on Introduction to OpenMP

[3] Introduction to OpenMP, <https://carleton.ca/rcs/rcdc/introduction-to-openmp/>

[4] Yiling, OpenMP - Scheduling(static, dynamic, guided, runtime, auto), <https://610yilingliu.github.io/2020/07/15/ScheduleinOpenMP/>

[5]Determining the Number of Threads for a parallel Region, <https://www.openmp.org/spec-html/5.0/openmpsu35.html>