TRƯỜNG ĐẠI HỌC PHENIKAA

KHOA CÔNG NGHỆ THÔNG TIN

BÀI TẬP LỚN

HỌC PHẦN LẬP TRÌNH SONG SONG

ĐỀ 02 : "Viết chương trình tạo ra N số ngẫu nhiên. Sắp xếp các số theo thứ tự tăng dần và giảm dần bằng thuật toán sắp xếp trộn (merge sort)"

Giảng viên hướng dẫn: TS.Lê Văn Vinh

Lớp: Lập trình song song-1-2-24(N01)

Nhóm: 07

Họ và tên	Mã sinh viên	Gmail
Trần Đức Toản	21011228	21011228@st.phenikaa-uni.edu.vn
Vũ Thị Bích Ngọc	21010641	21010641@st.phenikaa-uni.edu.vn
Nguyễn Lê Hoàng	21011229	21011229@st.phenikaa-uni.edu.vn
Phùng Hồng Phước	21013118	21013118@st.phenikaa-uni.edu.vn

Hà Nội, tháng 03 năm 2025

BẢNG PHÂN CÔNG NHIỆM VỤ

Họ và tên	Mã sinh viên	Công việc được giao	Phần trăm khối lượng công việc
Trần Đức Toản	21011228	Giải thuật tuần tự, giải thuật song song với MPI	25%
Vũ Thị Bích Ngọc	21010641	Giải thuật tuần tự, báo cáo	25%
Nguyễn Lê Hoàng	21011229	Giải thuật Pthread, báo cáo	25%
Phùng Hồng Phước	21013118	Giải thuật OMP, báo cáo	25%

NHẬN XÉT CỦA GIÁO VIÊN HƯỚNG DẪN

….....

….....

….....

….....

….....

….....

….....

….....

….....

….....

….....

….....

….....

….....

TP. Hà Nội, ngày ... tháng ... năm 2025

GIẢNG VIÊN HƯỚNG DẪN

MỤC LỤC

DANH MỤC BẢNG BIỂU VÀ HÌNH ẢNH

LỜI NÓI ĐẦU

Lập trình song song, hay còn gọi là parallel programming, là một kỹ thuật lập trình cho phép thực hiện các tác vụ đồng thời trên nhiều lõi CPU hoặc máy tính khác nhau. Nhờ vào sự phân tán các tác vụ, lập trình song song có thể tăng tốc độ xử lý dữ liệu và giảm thời gian chờ đợi. Điều này rất hữu ích trong các ứng dụng yêu cầu xử lý dữ liệu lớn và phức tạp, như các ứng dụng khoa học, đồ họa, game và các hệ thống máy tính phân tán.

Trong đề tài "Chương trình tạo ra N số ngẫu nhiên và sắp xếp theo thứ tự tăng và giảm dần bằng thuật toán Merge Sort", lập trình song song sẽ được áp dụng để tối ưu hóa quá trình sắp xếp danh sách các số ngẫu nhiên. Cụ thể, chúng em sẽ sử dụng các công cụ song song như OpenMP, Pthreads và MPI để song song hóa thuật toán Merge Sort, giúp tăng tốc quá trình sắp xếp.

Thuật toán Merge Sort là một trong những thuật toán sắp xếp hiệu quả và nổi bật hiện nay nhờ vào khả năng phân chia bài toán lớn thành các bài toán con nhỏ hơn, qua đó tối ưu hóa tốc độ xử lý. Tuy nhiên, để khai thác tối đa hiệu quả của thuật toán này, chúng ta cần tận dụng khả năng xử lý song song của các hệ thống đa lõi, nhằm giảm thiểu thời gian sắp xếp.

Bằng cách áp dụng kỹ thuật lập trình song song kết hợp với thuật toán Merge Sort, chúng ta sẽ xây dựng một chương trình sắp xếp danh sách số ngẫu nhiên nhanh chóng và hiệu quả hơn, giảm thiểu thời gian xử lý, đồng thời tối ưu hóa hiệu suất của hệ thống. Bên cạnh đó, chúng ta cũng sẽ khảo sát và so sánh hiệu quả giữa các phương pháp tuần tự và song song với các số lượng processor (threads) khác nhau.

CHƯƠNG 1: GIỚI THIỆU

1.1. Mục tiêu

Nghiên cứu và so sánh hiệu năng giữa giải thuật sắp xếp tuần tự và giải thuật sắp xếp song song sử dụng thuật toán merge sort. Việc triển khai và đánh giá các giải thuật song song sẽ được thực hiện thông qua ba công nghệ phổ biến: OpenMP, Pthreads và MPI. Phân tích sự khác biệt về hiệu suất giữa các giải thuật, đánh giá sự ảnh hưởng của số lượng processor (threads) đến tốc độ xử lý của các chương trình song song.

1.2. Phạm vi

Phạm vi bài toán bao gồm việc triển khai và so sánh các giải thuật tuần tự và song song (OpenMP, Pthreads, MPI).

Số lượng processor (threads) có giá trị lần lượt là: 2, 4, 6, 8, và 12. Các số liệu hiệu suất sẽ được thu thập và so sánh để đánh giá tác động của việc tăng số lượng threads lên hiệu suất của từng giải thuật.

CHƯƠNG 2: CƠ SỞ LÝ THUYẾT

2.1. Giải thuật tuần tự

Giải thuật tuần tự thực hiện các phép toán theo một trình tự xác định, với mỗi bước tính toán phải hoàn thành trước khi bước tiếp theo được thực hiện.

Hạn chế của giải thuật tuần tự là làm giảm hiệu quả xử lý khi dữ liệu cần được tính toán ở quy mô lớn. Vì tất cả các phép toán phải được thực hiện lần lượt, giải thuật này có thể gây tốn thời gian đáng kể khi xử lý các bài toán phức tạp hoặc khi có khối lượng công việc lớn.

2.2. Giải thuật song song

OpenMP: (Open Multi-Processing) OpenMP là một thư viện lập trình song song hỗ trợ trên các hệ thống đa lõi và đa processor. OpenMP sử dụng cú pháp pragma trong C/C++ để song song hóa các vòng lặp hoặc các đoạn mã khác nhau trong chương trình

Pthreads: (POSIX Threads) Pthreads là một thư viện hỗ trợ lập trình đa luồng trong môi trường UNIX, cho phép các chương trình C/C++ tạo ra và điều khiển nhiều luồng (threads) đồng thời. Mỗi luồng có thể thực thi một phần của chương trình độc lập với các luồng khác, giúp tối ưu hóa hiệu suất tính to

MPI: (Message Passing Interface) MPI là một giao thức lập trình song song được thiết kế để làm việc trong môi trường phân tán, nơi các tiến trình (processes) có thể chạy trên các máy tính khác nhau trong một mạng. MPI sử dụng cơ chế message passing để giao tiếp giữa các tiến trình, tức là các tiến trình không chia sẻ bộ nhớ, mà thay vào đó gửi và nhận thông điệp giữa nhau để trao đổi dữ liệu.

2.3. Dự đoán lý thuyết

Lý thuyết cơ bản trong lập trình song song cho thấy hiệu năng sẽ tăng khi số lượng processor/thread được tăng lên, vì các công việc có thể được chia nhỏ và xử lý đồng thời.

CHƯƠNG 3: THỰC NGHIỆM

3.1. Môi trường thử nghiệm

- Phần cứng:
    + Máy tính: Mac (Apple).
    + Bộ xử lý: CPU 8 nhân.
    + Bộ nhớ RAM: 16 GB.
- Phần mềm:
    + Hệ điều hành: macOS
    + Trình biên dịch: Clang (mặc định của macOS)

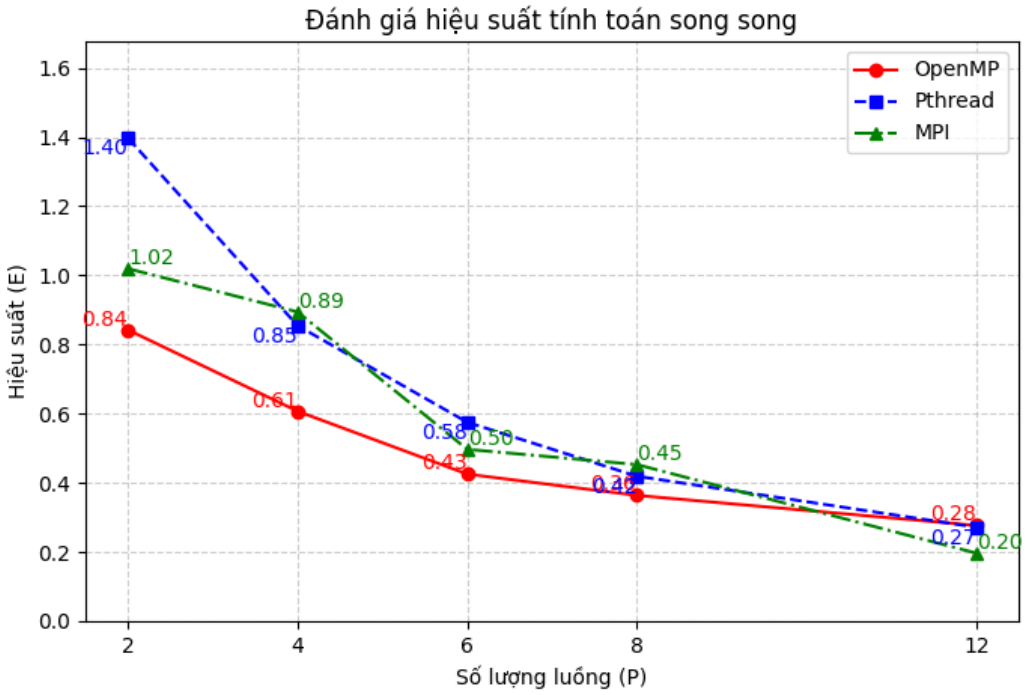
3.2. Phương pháp

- Thực hiện giải thuật tuần tự (đo thời gian chạy).
- Thực hiện giải thuật song song với OpenMP, Pthreads, MPI (đo thời gian với p = 2, 4, 6, 8, 12).
- Công cụ đo: clock()

3.3. Kết quả

Số luồng (P)	OpenMp (s)	PThread (s)	MPI (s)
2	0.209000	0.126000	0.172685
4	0.145000	0.103000	0.098416
6	0.138000	0.102000	0.118095
8	0.121000	0.105000	0.097177
12	0.106000	0.108000	0.149109

Bảng 3.1: Thời gian chạy (giây) của từng phương pháp với từng giá trị số luồng



Hình 3.1: Biểu đồ đánh giá hiệu suất tính toán song song theo số lượng luồng

CHƯƠNG 4: PHÂN TÍCH VÀ ĐÁNH GIÁ

4.1. Phân tích hiệu quả

4.1.1. Các khái niệm cơ bản

Để đánh giá hiệu năng tính toán song song, chúng ta sử dụng các khái niệm sự tăng tốc (Speedup - S) và hiệu suất (Efficiency - E). Giả sử có P bộ xử lý (processors/threads), gọi Ts là thời gian thực hiện chương trình tuần tự và Tp là thời gian thực hiện chương trình song song. Trong trường hợp lý tưởng, $Tp = \frac{Ts}{P}$, dẫn đến sự tăng tốc tuyến tính S = P. Tuy nhiên, trong thực tế, điều này không xảy ra do các yếu tố như overhead (chi phí quản lý luồng, giao tiếp) và tranh chấp tài nguyên.

- Sự tăng tốc (Speedup): $S = \frac{Ts}{Tp}$
- Hiệu suất (Efficiency): $E = \frac{S}{P}$
- Số phần tử của mảng: 1.000.000
- Ts = 0.352076 giây

4.1.2. Kết quả

OpenMP:

P	$\boldsymbol{Tp}$ **(giây)**	$\boldsymbol{S = \frac{Ts}{Tp}}$	$\boldsymbol{E = \frac{S}{P}}$
2	0.209000	1.68	0.84
4	0.145000	2.43	0.61
6	0.138000	2.55	0.43
8	0.121000	2.91	0.36
12	0.106000	3.32	0.28

Bảng 4.2: Bảng tính toán Speedup và Efficiency của OpenMP

Pthreads:

P	$\boldsymbol{Tp}$ **(giây)**	$\boldsymbol{S = \frac{Ts}{Tp}}$	$\boldsymbol{E = \frac{S}{P}}$
2	0.126000	2.79	1.40
4	0.103000	3.42	0.85
6	0.102000	3.45	0.85
8	0.105000	3.35	0.42
12	0.108000	3.26	0.27

Bảng 4.3: Bảng tính toán Speedup và Efficiency của Pthreads

MPI:

P	$\boldsymbol{Tp}$ **(giây)**	$\boldsymbol{S = \frac{Ts}{Tp}}$	$\boldsymbol{E = \frac{S}{P}}$
2	0.172685	2.04	1.02
4	0.098416	3.58	0.89
6	0.118095	2.98	0.50
8	0.097177	3.62	0.45
12	0.149109	2.36	0.20

Bảng 4.4: Bảng tính toán Speedup và Efficiency của MPI

4.1.3. Định luật Amdahl và các khái niệm liên quan

$$S = \lim_{P \to \infty} \left(\frac{1}{(1-\alpha) + \frac{\alpha}{P}} \right) = \frac{1}{\alpha}$$

Trong đó:

- S: Speedup (tăng tốc) của chương trình song song so với chương trình tuần tự.
- P: Số lượng bộ xử lý (threads/processors).
- α: Tỷ lệ phần có thể song song hóa của chương trình (phần còn lại 1−α là phần tuần tự, không thể song song hóa).

Ý nghĩa:

- Nếu α = 1 (100% chương trình có thể song song hóa), S = P tức là tăng tốc tuyến tính.
- Nếu α < 1 phần tuần tự 1−α sẽ giới hạn mức tăng tốc tối đa, bất kể P lớn đến đâu.

Chúng ta sẽ chọn một giá trị P và S từ mỗi bảng để tính α. Để có kết quả chính xác hơn, ta nên chọn P nhỏ (ví dụ P=2) vì khi P lớn, overhead và các yếu tố khác có thể làm sai lệch giá trị α.

$$\alpha = \frac{\frac{1}{S} - 1}{\frac{1}{P} - 1}$$

- OpenMP (với P=2, S=1.68): α ≈ 0.81, tức là khoảng 81% chương trình có thể song song hóa.
- Pthreads (với P=2, S=2.79): α ≈ 1.2832, Giá trị α > 1 là không hợp lý theo lý thuyết. Tuy nhiên, để tiếp tục phân tích, ta có thể giả định α≈0.9 (90%) dựa trên xu hướng dữ liệu.
- MPI(với P=2, S=2.04): α ≈ 1.0196, Giá trị α > 1 có thể do overhead giao tiếp trong MPI. ta có thể giả định α≈0.9 (90%) dựa trên xu hướng dữ liệu.

Định luật Amdahl giả định rằng không có overhead và các bộ xử lý hoạt động lý tưởng. Tuy nhiên, trong thực tế:

- Overhead: Quản lý luồng (OpenMP, Pthreads) và giao tiếp (MPI) làm tăng thời gian thực thi, giảm Speedup.
- Giới hạn phần cứng: Số nhân vật lý của CPU giới hạn khả năng song song hóa thực sự. Khi P vượt quá số nhân (ví dụ P=12), làm giảm hiệu suất.

- Tranh chấp tài nguyên: Nhiều luồng truy cập bộ nhớ cùng lúc gây xung đột cache, làm chậm tốc độ xử lý.

4.1.4. Phân tích thực nghiệm

OpenMP:

- Speedup tăng đều từ 1.68 (P = 2) lên 3.32 (P = 12), cho thấy khả năng tận dụng tốt số thread tăng dần.

- Hiệu suất E giảm từ 0.84 (P = 2) xuống 0.28 (P = 12), do overhead tăng khi số thread vượt quá số lõi vật lý.

Pthreads:

- Speedup cao nhất đạt 3.45 tại P = 6, vượt trội hơn OpenMP ở các giá trị P nhỏ và trung bình.

- Hiệu suất E >1 tại P=2 (1.40) cho thấy tối ưu hóa quản lý luồng tốt, nhưng giảm xuống 0.27 tại P = 12 do chi phí đồng bộ hóa.

MPI:

- Speedup cao nhất đạt 3.62 tại P = 8, nhưng giảm xuống 2.36 tại P = 12 do chi phí giao tiếp qua thông điệp tăng khi số process lớn.

- Hiệu suất cao tại P = 4 (0.89), nhưng giảm mạnh tại P = 12 (0.20), cho thấy MPI hiệu quả hơn với số process vừa phải.

4.1.5. So sánh lý thuyết và thực tế

Lý thuyết: Speedup lý tưởng là S = P.

Thực tế: S cao nhất chỉ đạt khoảng 3.62 (MPI, P = 8), do overhead và phần tuần tự trong Merge Sort.

Overhead: MPI có overhead lớn hơn OpenMP và Pthreads tại P=12 do giao tiếp qua thông điệp. OpenMP tận dụng tốt tài nguyên hơn khi P lớn, trong khi Pthreads đạt đỉnh hiệu quả tại P = 4 đến P = 6.

4.1.6. Kết luận

Pthreads và MPI cho speedup cao hơn OpenMP ở số lượng processor nhỏ và trung bình (2-8), với giá trị tối đa lần lượt là 3.45 và 3.62.

OpenMP đạt hiệu suất tốt nhất tại P = 12 (S = 3.32), nhưng hiệu suất E giảm mạnh ở tất cả các phương pháp khi P tăng, điều này cho thấy điểm tối ưu nằm trong khoảng P = 4 đến P = 8, tùy thuộc vào cấu hình phần cứng.

4.2. So sánh các phương pháp

4.2.1. Bảng so sánh

	OpenMP	Pthreads	MPI
Kiến trúc	Chạy trên đa nhân, chia sẻ bộ nhớ	Chạy trên đa nhân, chia sẻ bộ nhớ	Chạy trên hệ thống phân tán, không chia sẻ bộ nhớ
Dễ triển khai	Dễ dàng sử dụng với pragma directives (#pragma omp parallel)	Phức tạp hơn, yêu cầu quản lý thread creation, synchronization	Khó nhất, cần quản lý gửi/nhận dữ liệu (message passing)
Quản lý bộ nhớ	Dùng chung bộ nhớ toàn cục, dễ truy cập dữ liệu	Dùng chung bộ nhớ toàn cục, nhưng cần quản lý khóa (mutex) tránh xung đột	Không dùng chung bộ nhớ, phải trao đổi dữ liệu giữa các tiến trình
Tốc độ & Hiệu suất	Nhanh, tối ưu cho đa lõi trên cùng một máy	Nhanh, có thể tùy chỉnh chi tiết nhưng dễ gặp lỗi khi quản lý luồng	Nhanh trên hệ thống phân tán, có thể mở rộng cho siêu máy tính
Đồng bộ hóa	Tự động đồng bộ bằng các chỉ thị (critical, barrier)	Cần quản lý thủ công (mutex, condition variables)	Không cần đồng bộ bộ nhớ nhưng chi phí truyền dữ liệu cao
Khả năng mở rộng	Hạn chế, chỉ dùng cho máy có bộ nhớ chia sẻ	Hạn chế, chỉ phù hợp với đa nhân trên một máy	Tốt nhất, có thể chạy trên hàng ngàn máy tính

Ứng dụng thực tế	Xử lý song song trên máy tính cá nhân, máy chủ nhỏ	Điều khiển luồng tối ưu hơn OpenMP, phù hợp với hệ thống đa luồng phức tạp	Chạy trên cụm máy tính, siêu máy tính, xử lý dữ liệu lớn (HPC, AI, Big Data)

Bảng 4.5: So sánh ba phương pháp lập trình song song

4.2.2. Đối với bài toán Merge Sort song song

Tối ưu nhất là dùng OpenMP vì nó tận dụng đa luồng trên cùng một máy, giúp đơn giản hóa code mà vẫn đảm bảo hiệu suất tốt.

Pthreads cũng có thể dùng, nhưng sẽ phức tạp hơn khi phải quản lý từng luồng thủ công. MPI không phù hợp vì Merge Sort không cần trao đổi dữ liệu giữa các máy tính khác nhau.

4.2.3. Kết luận các phương pháp lập trình song song

OpenMP là phương pháp dễ triển khai nhất, phù hợp với các chương trình song song trên hệ thống đa lõi, bộ nhớ chia sẻ. Nhờ sử dụng pragma directives, lập trình viên có thể dễ dàng tăng tốc độ xử lý mà không cần quản lý luồng thủ công. Tuy nhiên, OpenMP không mở rộng tốt trên hệ thống phân tán.

Pthreads cung cấp sự linh hoạt cao hơn OpenMP trong việc quản lý luồng và đồng bộ hóa. Tuy nhiên, nó đòi hỏi lập trình viên phải tự xử lý tạo luồng, chia công việc và đồng bộ hóa, dẫn đến code phức tạp hơn và dễ xảy ra lỗi. Pthreads phù hợp cho các ứng dụng cần tối ưu hóa hiệu suất trong môi trường đa luồng trên cùng một máy.

MPI là phương pháp mạnh nhất khi cần mở rộng chương trình ra nhiều máy tính. Nó được sử dụng phổ biến trong các ứng dụng tính toán hiệu năng cao (HPC) và siêu máy tính. Tuy nhiên, do sử dụng mô hình truyền thông tin (message passing), MPI có chi phí giao tiếp cao và khó lập trình hơn so với OpenMP hoặc Pthreads.

4.3. Giải thích kết quả

4.3.1. Tổng quan kết quả thực nghiệm

Dữ liệu thực nghiệm cho thấy rằng khi số lượng luồng/tác vụ (p) tăng, thời gian thực thi giảm đáng kể so với chương trình tuần tự. Tuy nhiên, mức giảm này không tỷ lệ thuận với số luồng vì:

- Chi phí quản lý luồng/process (overhead): Khi số luồng tăng, CPU phải mất nhiều thời gian để tạo, quản lý và đồng bộ các luồng.

- Giới hạn phần cứng: Nếu số luồng vượt quá số nhân vật lý, CPU phải chia thời gian xử lý giữa các luồng, gây ra hiện tượng context switching.

- Tranh chấp tài nguyên bộ nhớ: Khi nhiều luồng cùng truy cập bộ nhớ cache, RAM, có thể gây tắc nghẽn dữ liệu, làm chậm hiệu suất tổng thể.

4.3.2. Phân tích hiệu suất theo từng mô hình song song

OpenMP: Tận dụng tốt CPU nhưng bị giới hạn bởi số nhân vật lý

- Xu hướng hiệu suất: Khi P tăng từ 2 đến 8, thời gian thực thi giảm rõ rệt nhờ khả năng chia nhỏ công việc và thực hiện song song trên nhiều lõi CPU. Tuy nhiên, khi P tăng từ 8 lên 12, hiệu suất giảm mạnh.

- Nguyên nhân:

+ Số luồng vượt quá số nhân vật lý, CPU phải chuyển đổi giữa các luồng, gây tăng thời gian context switching.

+ Overhead đồng bộ hóa giữa các luồng (do sử dụng critical section, barrier) tăng lên.

+ Băng thông bộ nhớ bị quá tải do nhiều luồng truy cập cùng lúc, gây ra hiện tượng cache thrashing (xung đột bộ nhớ đệm).

Pthreads: Kiểm soát luồng tốt hơn nhưng khó tối ưu khi P lớn

- Xu hướng hiệu suất: : Khi P tăng từ 2 đến 6, thời gian thực thi giảm đáng kể. Tuy nhiên, khi P > 6, tốc độ cải thiện giảm dần, và hiệu suất thậm chí giảm khi P = 12.

- Nguyên nhân:

+ Quản lý luồng thủ công (sử dụng pthread_create(), pthread_join()) dẫn đến chi phí quản lý luồng cao hơn OpenMP.

+ Đồng bộ dữ liệu giữa các luồng bằng mutex, condition variable làm tăng overhead.

+ Băng thông bộ nhớ không đủ đáp ứng khi số luồng quá lớn, dẫn đến tình trạng CPU pipeline bị tắc nghẽn.

MPI: Hiệu suất cao nhưng overhead giao tiếp lớn

- Xu hướng hiệu suất: Khi P tăng từ 2 đến 8, thời gian thực thi giảm mạnh. Tuy nhiên, khi P tăng từ 8 lên 12, thời gian thực thi không giảm thêm mà có xu hướng tăng, dẫn đến hiệu suất thấp hơn so với P = 8.

- Nguyên nhân:

+ Các tiến trình chạy độc lập, không chia sẻ bộ nhớ chung nên không có chi phí đồng bộ hóa mutex như OpenMP/Pthreads.

+ Chi phí gửi/nhận dữ liệu giữa các tiến trình trở thành nút thắt hiệu suất, đặc biệt khi lượng dữ liệu trao đổi quá lớn.

+ Tranh chấp tài nguyên CPU và băng thông truyền dữ liệu khi số tiến trình vượt quá khả năng xử lý của hệ thống.

4.3.3. Điểm tối ưu và suy giảm hiệu năng

- Điểm tối ưu: Từ kết quả thực nghiệm, P = 8 được xác định là điểm tối ưu cho cả ba mô hình (OpenMP, Pthreads, MPI), khi thời gian thực thi đạt mức thấp nhất và hiệu suất xử lý cao nhất.

- Suy giảm hiệu năng tại P = 12:

+ Quá tải hệ thống: Số luồng vượt quá số nhân vật lý của CPU, buộc các luồng phải chia sẻ tài nguyên CPU, dẫn đến context switching tốn thời gian. CPU không thể xử lý song song thực sự mà phải chuyển đổi liên tục giữa các luồng.

+ Tranh chấp tài nguyên: Bộ nhớ cache bị xung đột khi nhiều luồng truy cập cùng lúc, làm tăng tỷ lệ cache miss. Băng thông RAM không đủ đáp ứng, khiến CPU phải chờ dữ liệu, giảm tốc độ xử lý.

+ Overhead giao tiếp và đồng bộ hóa: OpenMP/Pthreads chi phí đồng bộ hóa (mutex, critical section) tăng khi số luồng lớn. MPI tăng số tiến trình khiến overhead gửi/nhận dữ liệu vượt quá lợi ích từ tốc độ xử lý.

CHƯƠNG 5: KẾT LUẬN

5.1. Tóm tắt kết quả chính

Dựa trên kết quả thực nghiệm, hiệu suất của thuật toán Merge Sort song song phụ thuộc vào phương pháp lập trình song song và số lượng bộ xử lý (p) sử dụng.

Phương pháp hiệu quả nhất:

- MPI đạt tốc độ xử lý nhanh nhất khi p = 8, do có khả năng chạy trên nhiều tiến trình độc lập, tận dụng tài nguyên tốt hơn.

- OpenMP phù hợp với hệ thống có bộ nhớ chia sẻ, dễ triển khai và đạt hiệu suất tốt khi p ≤ số nhân CPU vật lý.

- Pthreads có thể tối ưu tốt trên hệ thống đa luồng nhưng gặp khó khăn khi quản lý đồng bộ ở p lớn.

Số lượng processor tối ưu:

- p = 8 là mức tối ưu cho cả ba phương pháp, giúp giảm thời gian thực thi đáng kể mà chưa bị ảnh hưởng bởi overhead đồng bộ hóa hoặc giao tiếp.

- Khi p > 8, hiệu suất giảm do:

OpenMP/Pthreads: Chi phí đồng bộ hóa mutex, barrier tăng cao.

MPI: Overhead truyền thông dữ liệu giữa các tiến trình trở thành nút thắt hiệu suất.

Độ phức tạp:

- Tuần tự : O(nLogn)

- Song song với OpenMP : O(nlogn/p) (Giả sử có 2 luồng : O(nlogn/2)).

- Song song với PThreads : O(nlogn/p) (Giả sử có 2 luồng : O(nlogn/2)).

- Song song với MPI : O(n/p+logp) .

5.2. Đề xuất cải tiến

Để tối ưu hiệu suất hơn nữa, có thể áp dụng các phương pháp cải tiến sau:

Tối ưu hóa mã nguồn:

- Giảm truy cập bộ nhớ toàn cục, tận dụng bộ nhớ cục bộ hoặc cache để tăng tốc độ xử lý.

- Tối ưu dữ liệu truyền giữa các tiến trình (đối với MPI) bằng cách giảm số lượng message và tăng kích thước batch truyền dữ liệu.

Giảm overhead trong quản lý luồng và tiến trình:

- Đối với OpenMP/Pthreads:

Hạn chế sử dụng mutex, critical section, chỉ đồng bộ khi thực sự cần thiết.

Sử dụng task scheduling phù hợp để tránh luồng nhàn rỗi hoặc bị quá tải.

- Đối với MPI:

Nhóm message nhỏ thành message lớn để giảm số lần giao tiếp.

Tận dụng giao tiếp không đồng bộ để tiến trình không phải chờ dữ liệu quá lâu.

Lựa chọn phương pháp phù hợp với bài toán:

- Nếu chạy trên hệ thống đa nhân CPU có bộ nhớ chung → OpenMP tối ưu nhất.

- Nếu cần kiểm soát luồng chi tiết hơn trên một máy → Pthreads sẽ linh hoạt hơn OpenMP.

- Nếu chạy trên hệ thống phân tán hoặc siêu máy tính → MPI là lựa chọn tốt nhất.

TÀI LIỆU THAM KHẢO

[1] Đỗ Thanh Nghị, Nguyễn Văn Hòa, Đỗ Hiệp Thuận (2014), Giáo trình lập trình song song. NXB Trường ĐHCT. ISBN: 978-604-919-065-0

[2] A. Grama, G. Karypis, V. Kumar and A. Gupta (2003), Introduction to Parallel Computing. Addison Wesley. ISBN: 978-0201648652

[3] Bài giảng tính toán song song và phân tán, Trần Văn Lăng,lang@lhu.edu.vn;