TRƯỜNG ĐẠI HỌC HỌC VĂN LANG
KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO ĐỒ ÁN MÔN HỌC

LẬP TRÌNH TOÁN SONG SONG

NGÀNH: CÔNG NGHỆ THÔNG TIN

Đề tài:

ÁP DỤNG THUẬT TOÁN SONG SONG HOÁ SẮP XẾP NỔI BỌT (BUBBLE_SORT ALGORITHIM)

SVTH: Trần Tấn Phát

MSSV: 242_71ITAI40903_01

GVHD: TRẦN NGỌC VIỆT

THÀNH PHỐ HỒ CHÍ MINH – NĂM 2025

LỜI CẢM ƠN

Lời nói đầu tiên, chúng em xin chân thành cảm ơn sự hướng dẫn của thầy Trần Ngọc Việt, người đã luôn dành thời gian và tâm huyết để hỗ trợ chúng em trong suốt quá trình học tập và nghiên cứu. Trong quá trình thực hiện nghiên cứu đề tài, chúng em đã gặp không ít khó khăn nhưng nhờ có sự hướng dẫn tận tình của thầy nên nhóm em đã có thể hoàn thành tốt bài tiểu luận.

Tiếp theo, chúng em xin gửi lời chân thành cảm ơn đến Khoa Công nghệ thông tin- Đại Học Văn Lang thành phố Hồ Chí Minh đã tạo điều kiện thuận lợi cho chúng em học tập và hoàn thành đề tài tiểu luận này.

Mặc dù nhóm đã rất cố gắng vận dụng những kiến thức đã học được trong thời gian qua để hoàn thành bài tiểu luận nhưng do không có nhiều kinh nghiệm thực tiễn nên khó tránh khỏi những thiếu sót trong quá trình nghiên cứu và làm bài. Nhưng với tinh thần cầu tiến và mong muốn tiến bộ, chúng em tin rằng những ý kiến và đóng góp của quý thầy cô và các bạn đọc giả sẽ góp phần giúp chúng em hoàn thiện bản thân hơn.

CHƯƠNG 1. CƠ SỞ LÝ THUYẾT – THUẬT TOÁN SONG SONG HOÁ SẮP XẾP NỔI BỌT (BUBBLE_SORT_ALGORITHIM)

1.1. Lý do chọn đề tài

Với bối cảnh của nền cách mạng công nghiệp 4.0. Việc sử dụng thuật toán song song nổi bọt là bước đi đầu tiên cho các sinh viên có thể hiểu và phát triển những kiến thức cơ bản về toán song song và tầm quan trọng của các thuật toán đòi hỏi cần nhiều kiến thức và kĩ thuật trong tương lai.

1.2. Cơ sở lý thuyết Bubble_sort_algorithim

A. Bubble_sort_algorithim là gì?

- Bubble_sort là một thuật toán sắp xếp đơn giản, hoạt động bằng cách lặp đi lặp lại qua các danh sách, so sánh các cặp phần tử liền kề và hoán đổi chúng lại nếu thứ tự các số không phù hợp.
- Nguyên tắc cơ bản:
    - Duyệt qua mảng từ đầu đến cuối.
    - So sánh từng cặp phần tử liền kề.
    - Nếu phần tử đứng trước lớn hơn số sau thì sẽ hoán đổi lại.
    - Lặp lại quá trình này cho đến khi các số có vị trí phù hợp.
- Ví dụ:
    - Sắp xếp mảng [5,1,4,2,8]
    - Vòng lặp 1:

      [5, 1, 4, 2, 8] → [1, 5, 4, 2, 8] (So sánh 5 > 1, hoán đổi)
      [1, 5, 4, 2, 8] → [1, 4, 5, 2, 8] (So sánh 5 > 4, hoán đổi)
      [1, 4, 5, 2, 8] → [1, 4, 2, 5, 8] (So sánh 5 > 2, hoán đổi)
      [1, 4, 2, 5, 8] → [1, 4, 2, 5, 8] (So sánh 5 < 8, giữ nguyên)

    - Vòng lặp 2:

      [1, 4, 2, 5, 8] → [1, 4, 2, 5, 8] (So sánh 1 < 4, giữ nguyên)
      [1, 4, 2, 5, 8] → [1, 2, 4, 5, 8] (So sánh 4 > 2, hoán đổi)
      [1, 2, 4, 5, 8] → [1, 2, 4, 5, 8] (So sánh 4 < 5, giữ nguyên)

    - Vòng lặp 3,4 không đổi => Kết thúc thuật toán.

B. Độ phức tạp của thuật toán Bubble_sort

❖ Phân tích độ phức tạp thời gian

- Phân tích trường hợp tốt nhất O(n) – khi mảng được sắp xếp.
- Trường hợp trung bình: $O(n^2)$.
- Trường hợp xấu nhất: $O(n^2)$ – khi mảng được sắp xếp ngược.

❖ Chứng minh độ phức tạp $O(n^2)$:

- Trong trường hợp xấu nhất (mảng sắp xếp ngược), thuật toán sẽ thực hiên:
    - Vòng lặp ngoài: n-1 lần.
    - Vòng lặp trong: (n-1)+ (n-2)+ …+1=n(n-1)/2 là phép so sánh hoán đổi.
    - Tổng độ phức tạp: n(n-1)/2 = $O(n^2)$.

❖ Độ phức tạp không gian:

O(1) – Bubble_sort là thuật toán sắp xếp tại chỗ, chỉ cần không gian bộ nhớ cố định cho các biến tạm.

C. Ưu điểm/ Nhược điểm

- **Ưu điểm:**
    - Phương thức tính toán đơn giản, dễ hiểu và dễ cài đặt.
    - Tính năng ổn định: các phần tử có giá trị bằng nhau sẽ được giữ nguyên thứ tự tương đối.
    - Có khả năng phát hiện sớm mảng đã sắp xếp, tiêu thụ ít bộ nhớ.
    - Phù hợp với các mảng nhỏ hoặc đã được sắp xếp.

- **Nhược điểm:**
    - Hiệu suất kém với dữ liệu phức tạp.
    - Thực hiện nhiều phép hoán đổi hơn so với các thuật toán khác nhau như Selection_sort.
    - Số lần hoán đổi nhiều gây tốn thời gian thực thi.

1.3. Khái niệm về song song hoá

Định nghĩa và tầm quan trọng

- **Định nghĩa**: Song song hoá là kỹ thuật thực hiện đồng thời nhiều phần của tác vụ, nhằm tăng tốc độ xử lý. Trong bối cảnh sắp xếp, song song hoá giúp chia nhỏ công việc sắp xếp thành các phần nhỏ hơn, giúp chúng được xử lý đồng thời trên nhiều nguồn. Qua đó, thuật toán song song giúp người dùng giải quyết các bài toán lớn trong thời gian ngắn hơn, tận dụng tối đa phần cứng đa lõi.
- **Tầm quan trọng:**
    - Tăng tốc độ xử lý: Giúp giảm thời gian thực thi khi xử lý dữ liệu lớn.
    - Tối ưu hoá tài nguyên phần cứng: Tận dụng nhiều lõi CPU để tăng hiệu suất.
    - Xử lý đa nhiệm hiệu quả: Giảm độ trễ, tăng trải nghiệm người dùng.

1.4. Các mô hình song song hoá

A. MPI (Message Passing Interface):

- **Định nghĩa:** Thư viện chuẩn cho việc truyền thông điệp song song kiến trúc bộ nhớ phân tán.
- **Đặc điểm:**
    - Một chuẩn giao tiếp cho các hệ thống phân tán, cho phép các tiến trình trao đổi thông điệp với nhau.Phù hợp cho các hệ thống phân tán và siêu máy tính.
    - Phù hợp cho các bài toán song song hóa phức tạp và đòi hỏi giao tiếp nhiều giữa các tiến trình.
    - Hỗ trợ ngôn ngữ C, Python ,…

B. Đa luồng (Multithreading):

- **Định nghĩa:** Kỹ thuật lập trình cho phép nhiều luồng thực thi đồng thời trong cùng 1 quá trình.
- **Đặc điểm:**
    - Các luồng chia sẻ cùng một không gian bộ nhớ, giúp giảm chi phí giao tiếp.

- Hầu hết các ngôn ngữ hiện đại đều hỗ trợ như: Java, C#, Python, C++ std::thread.
- Mô hình lập trình trực tiếp, kiểm soát chi tiết hơn.

CHƯƠNG 2. ÁP DỤNG THUẬT TOÁN SONG SONG BUBBLE_SORT CHO BÀI TOÁN LẤY TRONG MẢNG ĐỂ SẮP XẾP THEO THỨ TỰ

2.1 Phát biểu bài toán:

Chủ đề đã cho yêu cầu tập một tập hợp gồm các số thứ tự ngẫu nhiên trong khoảng từ 0 đến 100. Số đó để sắp xếp theo thứ tự tăng dần bằng thuật toán song song Bubble_sort. Bài toán sẽ thực hiện hai phương pháp sắp xếp sau:

- Sắp xếp tuần tự bằng Bubble_sort thông thường.
- Sắp xếp song song bằng thuật toán Odd-Even Transposition với đa luồng (multithreading) hoặc đa tiến trình (multiprocessing).
- Phân tích sự hiệu quả của việc song song hoá.

2.2 Chi tiết bài toán:

Giả sử ta có một mảng số nguyên cần sắp xếp:
Mảng đầu vào: [5, 3, 8, 4, 2, 7, 1, 6]

Phiên bản tuần tự của Bubble Sort

1. Duyệt qua mảng và hoán đổi các phần tử nếu chúng không theo thứ tự.
2. Lặp lại quá trình cho đến khi toàn bộ mảng được sắp xếp.
3. Thời gian thực thi tăng theo cấp số nhân khi số lượng phần tử tăng.

Kết quả sau khi sắp xếp: [1, 2, 3, 4, 5, 6, 7, 8]

Phiên bản song song của Bubble Sort

1. Chia mảng đầu vào thành n phần nhỏ, mỗi phần sẽ được xử lý bởi một tiến trình riêng biệt.
2. Mỗi tiến trình thực hiện Bubble Sort trên phần dữ liệu của mình.
3. Sau khi tất cả tiến trình hoàn thành, chạy Bubble Sort toàn bộ mảng để đảm bảo sắp xếp hoàn chỉnh.

Ví dụ mảng đầu vào: arr = [56, 12, 89, 67, 34, 99, 78, 23, 45, 18, 90, 3, 26, 80, 61, 5, 2, 30, 70, 88]

Bước 1: ta chia mảng thành 4 phần (4 tiến trình):

Do có **20 phần tử** và **4 tiến trình**, mỗi phần sẽ có:

Chunk_size = 20/4 = 5

Tiến trình	**Start**	**End**	**Dữ liệu xử lý**
P0	0	5	[56, 12, 89, 67, 34]
P1	5	10	[99, 78, 23, 45, 18]
P2	10	15	[90, 3, 26, 80, 61]
P3	15	20	[5, 2, 30, 70, 88]

Bước 2: Mỗi tiến trình chạy Bubble Sort riêng biệt
Sau khi sắp xếp từng phần, ta có:
[12, 34, 56, 67, 89] # P0
[18, 23, 45, 78, 99] # P1
[3, 26, 61, 80, 90] # P2
[2, 5, 30, 70, 88] # P3

Bước 3: Ghép lại và chạy Bubble Sort lần cuối
Sau khi sắp xếp từng phần, ta vẫn chưa có mảng hoàn chỉnh:

[12, 34, 56, 67, 89, 18, 23, 45, 78, 99, 3, 26, 61, 80, 90, 2, 5, 30, 70, 88]

Chạy Bubble Sort toàn bộ để hoàn thành sắp xếp:

[2, 3, 5, 12, 18, 23, 26, 30, 34, 45, 56, 61, 67, 70, 78, 80, 88, 89, 90, 99]

Lợi ích của phiên bản song song:

- Giảm thời gian thực thi so với thuật toán tuần tự.
- Tận dụng tối đa tài nguyên phần cứng, đặc biệt là CPU đa luồng.
- Hiệu quả khi xử lý tập dữ liệu lớn.

Trình bày chi tiết:

A = [56, 12, 89, 67, 34, 99, 78, 23, 45, 18, 90, 3, 26, 80, 61, 5, 2, 30, 70, 88]

Chúng ta có thể chia danh sách này thành 4 mảng nhỏ hơn, mỗi mảng chứa 5 số.

[56, 12, 89, 67, 34], [99, 78, 23, 45, 18], [90, 3, 26, 80, 61], [5, 2, 30, 70, 88]

Thực hiện sắp xếp mảng P0: [56, 12, 89, 67, 34]

+ B1 j = 0

B2: j = 0

B3: nếu A[j] > A[j + 1] (A[0] > A[0]: 56 > 12) thì hoán đổi giữa 56 và 12

=>[12,56,89,67,34]

B4: Nếu j<(n−i−1) (0 < (5-0-1) = 4) đúng thì j = j + 1 = 0 + 1 = 1 và quay lại B3

+ B2: j=1

B3: Nếu A[j]>A[j+1] (A[1] > A[2]: 56 < 89) Không đổi.

B4: Nếu j<(n−i−1) (1 < 4) đúng thì j = j + 1 = 2 và quay lại B3

+ B2: j=2

B3: Nếu A[j]>A[j+1] (A[2] > A[3]: 89 > 67) thì hoán đổi giữa 89 và 67

=> [12,56,67,89,34]

B4: Nếu j<(n−i−1) (2 < 4) đúng thì j = j + 1 = 3 và quay lại B3

+ B2: j=3

B3: Nếu A[j]>A[j+1]A[j] > A[j + 1]A[j]>A[j+1] (A[3] > A[4]: 89 > 34) thì hoán đổi giữa 89 và 34

=> [12,56,67,34,89]

B4: Nếu j<(n−i−1)j < (n - i - 1)j<(n−i−1) (3 < 4) đúng thì j = j + 1 = 4 và quay lại B3

Vòng lặp i = 1:

B2: j= 0 (i = 1)

B3: nếu A[j] > A[j + 1] (A[0] > A[1]: 12 < 56) Không đổi

B4: Nếu j < (n - i - 1) (0 < 3) đúng thì j = j + 1 = 1 và quay lại B3

B2: j = 1

B3: nếu A[j] > A[j + 1] (A[1] > A[2]: 56 > 67) Không đổi.

B4: Nếu j < (n - i - 1) (1 < 3) đúng thì j = j + 1 = 2 và quay lại B3

B2: j = 2

B3: nếu A[j] > A[j + 1] (A[2] > A[3]: 67 > 34) thì hoán đổi giữa 67 và 34
=> [12,56,34,67,89]

B4: Nếu j < (n - i - 1) (2 < 3) đúng thì j = j + 1 = 3 và quay lại B3

Vòng lặp i = 2:

B2: j = 0 (i=2)

B3: nếu A[j] > A[j + 1] (A[0] > A[1]: 12 < 56) Không đổi.

B4: Nếu j < (n - i - 1) (0 < 2) đúng thì j = j + 1 = 1 và quay lại B3

B2: j = 1

B3: nếu A[j] > A[j + 1] (A[1] > A[2]: 56 > 34) thì hoán đổi giữa 56 và 34 => [12,34,56,67,89]

B4: Nếu j < (n - i - 1) (1 < 2) đúng thì j = j + 1 = 2 và quay lại B3

B2: j = 2

B3: nếu A[j] > A[j + 1] (A[2] > A[3]: 56 < 67) Không đổi.

B4: Nếu j < (n - i - 1) (2 < 2) Sai → Kết thúc vòng lặp i = 2

Vòng lặp i = 3:

B2: j = 0 (i=3)

B3: nếu A[j] > A[j + 1] (A[0] > A[1]: 12 < 34) Không đổi.

B4: Nếu j < (n - i - 1) (0 < 1) đúng thì j = j + 1 = 1 và quay lại B3

B2: j = 1

B3: nếu A[j] > A[j + 1] (A[1] > A[2]: 34 < 56) Không đổi.

B4: Nếu j < (n - i - 1) (1 < 1) Sai → Kết thúc vòng lặp i = 3

Vòng lặp i = 4:

B2: j = 0 (i=4)

B3: nếu A[j] > A[j + 1] (A[0] > A[1]: 12 < 34) Không đổi.

B4: Nếu j < (n - i - 1) (0 < 0) Sai → Kết thúc vòng lặp i = 4

Mảng p0 sau khi sắp xếp hoàn toàn: [12, 34, 56, 67, 89]

Tương tự như trên:

Sắp xếp mảng P1: [99, 78, 23, 45, 18]

Sau từng vòng lặp i:

- i = 0: [78, 23, 45, 18, 99]
- i = 1: [23, 45, 18, 78, 99]
- i = 2: [23, 18, 45, 78, 99]
- i = 3: [18, 23, 45, 78, 99]

Sắp xếp mảng P2: [90, 3, 26, 80, 61]

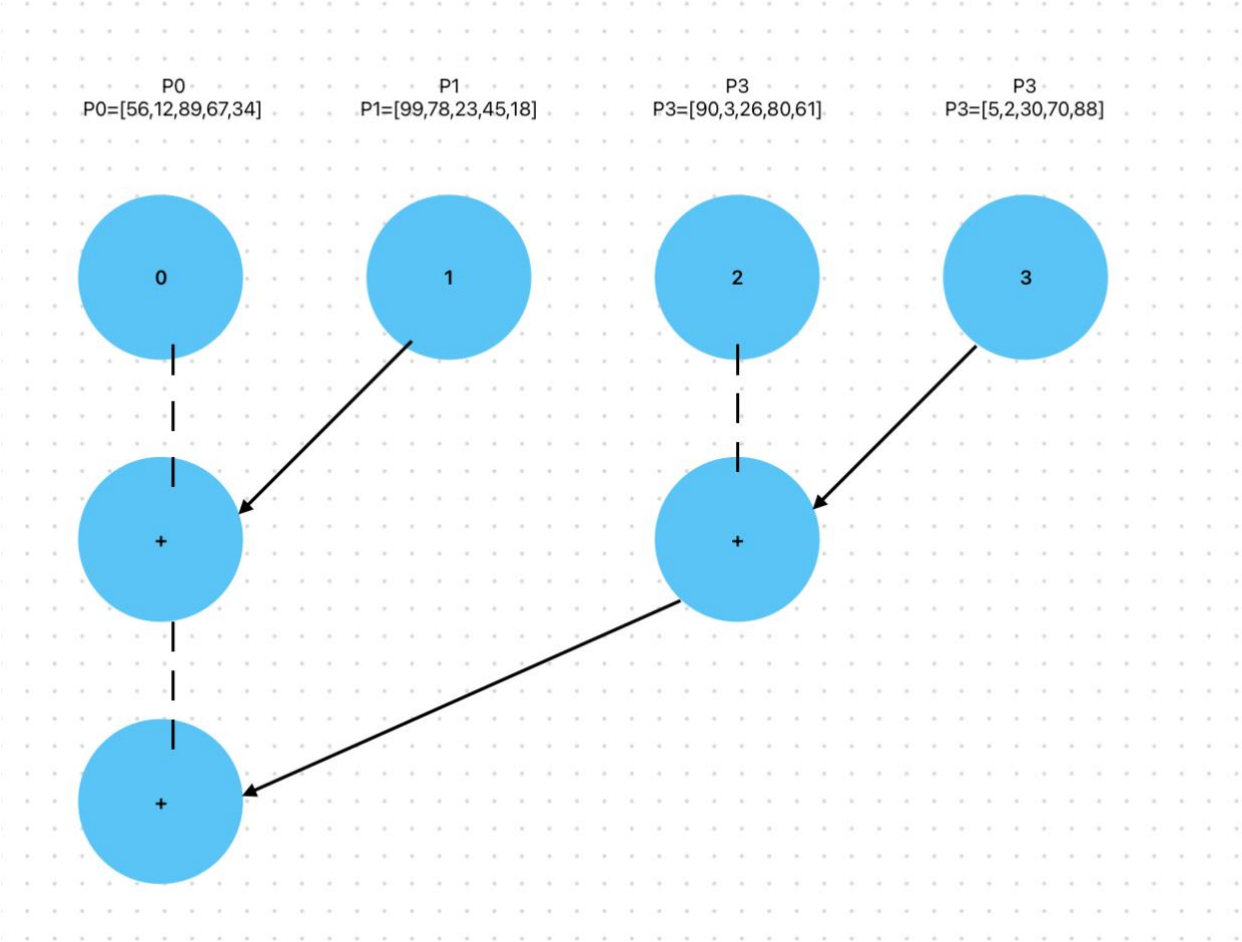
Sau từng vòng lặp i:

- i = 0: [3, 26, 80, 61, 90]
- i = 1: [3, 26, 61, 80, 90]
- i = 2: [3, 26, 61, 80, 90]
- i = 3: [3, 26, 61, 80, 90]

Sắp xếp mảng P3: [5, 2, 30, 70, 88]

Sau từng vòng lặp i:

- i = 0: [2, 5, 30, 70, 88]
- i = 1: [2, 5, 30, 70, 88]
- i = 2: [2, 5, 30, 70, 88]
- i = 3: [2, 5, 30, 70, 88]



Các tiến trình P0, P1, P2, P3 nhận dữ liệu ban đầu là các mảng chưa sắp xếp:

- A0 = [56, 12, 89, 67, 34] ; P0 xử lý
- A1 = [99, 78, 23, 45, 18] ; P1 xử lý
- A2 = [90, 3, 26, 80, 61] ; P2 xử lý
- A3 = [5, 2, 30, 70, 88] ; P3 xử lý

Giai đoạn 2: P1 thực hiện sắp xếp mảng của mình rồi truyền dữ liệu xuống nút trung gian "+" để hợp nhất với P0. Tương tự P3 thực hiện sắp xếp mảng của mình rồi truyền dữ liệu xuống nút trung gian "+" để hợp nhất với P2.

Giai đoạn 3: Con P2 tiếp tục truyền thông dữ liệu về P0 để có được kết quả cuối cùng mà mảng đã được sắp xếp hoàn tất

2.3 Mã nguồn

```
import multiprocessing
import random

def bubble_sort_partial(arr, start, end):
    """Sắp xếp một phần của mảng bằng Bubble Sort."""
    for i in range(start, end - 1):
        for j in range(start, end - (i - start) - 1):
            if arr[j] > arr[j + 1]:
                arr[j], arr[j + 1] = arr[j + 1], arr[j]

def final_bubble_sort(arr):
    n = len(arr)
    for i in range(n - 1):
        for j in range(n - i - 1):
            if arr[j] > arr[j + 1]:
                arr[j], arr[j + 1] = arr[j + 1], arr[j]

def parallel_bubble_sort(arr, num_procs):
    n = len(arr)
    chunk_size = (n + num_procs - 1) // num_procs
    processes = []

    for i in range(num_procs):
        start = i * chunk_size
        end = min((i + 1) * chunk_size, n)
        if start < end:
            p = multiprocessing.Process(target=bubble_sort_partial, args=(arr, start, end))
            processes.append(p)
            p.start()

    for p in processes:
        p.join()
    final_bubble_sort(arr)
```

```
if __name__ == "__main__":
    size = 20
    num_procs = 4
    arr = multiprocessing.Array('i', [random.randint(1, 100) for _ in range(size)])
    print("Mảng ban đầu:", list(arr))
    parallel_bubble_sort(arr, num_procs)
    print("Mảng sau khi sắp xếp:", list(arr))
```

2.4 Kết luận:

- Trong đoạn mã trên, hai phương pháp sắp xếp và song song – đã được so sánh để đánh giá hiệu suất xử lý. Ngược lại, sắp xếp song song sử dụng ThreadPoolExecutor để thực thi các bước hoán đổi đồng thời, giúp giảm tổng thời gian thực thi, đặc biệt khi xử lý tập dữ liệu lớn.
- Tuy nhiên, với tập dữ liệu nhỏ như 12 số trong chương trình, sự khác biệt về thời gian giữa hai phương pháp không quá đáng kể do chi phí khởi tạo và quản lý luồng trong sắp xếp song song. Kết quả cho thấy sắp xếp song song đạt hiệu quả hơn khi dữ liệu lớn và yêu cầu tối ưu hiệu suất, trong khi sắp xếp tuần tự phù hợp với dữ liệu nhỏ, đơn giản và dễ quản lý. Chỉ số "Speedup" cũng được sử dụng để đo lường mức độ cải thiện hiệu suất, và nếu giá trị này lớn hơn 1, sắp xếp song song cho thấy hiệu quả rõ rệt.

CHƯƠNG 3. KẾT LUẬN CHUNG

Qua đồ án **"Áp dụng thuật toán song song hóa sắp xếp nổi bọt (Bubble Sort Algorithm)"**, chúng ta đã tiến hành nghiên cứu và thực nghiệm hai phương pháp sắp xếp dữ liệu: **sắp xếp song song**.

❖ Tổng kết kết quả

- **Sắp xếp tuần tự** thực hiện các bước hoán đổi liên tục và tuần tự. Phương pháp này đơn giản, dễ triển khai nhưng tiêu tốn nhiều thời gian hơn do phải chờ từng bước hoàn thành trước khi chuyển sang bước tiếp theo.
- **Sắp xếp song song** sử dụng **ThreadPoolExecutor**, cho phép thực thi các bước hoán đổi đồng thời, giảm thiểu thời gian thực thi tổng thể. Phương pháp này tỏ ra hiệu quả hơn, đặc biệt khi xử lý trên tập dữ liệu lớn.
- Qua thử nghiệm, chỉ số **Speedup** (tỉ lệ giữa thời gian của phương pháp tuần tự và song song) cho thấy mức độ cải thiện hiệu suất rõ rệt khi sử dụng phương pháp song song, đặc biệt với tập dữ liệu lớn.

❖ Nhận xét

- Khi áp dụng trên tập dữ liệu nhỏ sự chênh lệch thời gian giữa hai phương pháp chưa thực sự nổi bật, do chi phí khởi tạo và quản lý luồng của phương pháp song song.
- Tuy nhiên, với dữ liệu lớn và phức tạp hơn, sắp xếp song song cho thấy **tính tối ưu hóa rõ rệt về hiệu suất**, giúp giảm đáng kể thời gian xử lý.
- Độ phức tạp của thuật toán Bubble Sort vẫn giữ nguyên (**$O(n^2)$**), nhưng việc song song hóa giúp tận dụng tốt hơn tài nguyên hệ thống (đa lõi CPU).

❖ Kế hoạch phát triển

- **Tối ưu hóa thuật toán**: Nghiên cứu và áp dụng các thuật toán sắp xếp tối ưu hơn như **Quick Sort**, **Merge Sort**, hoặc thuật toán sắp xếp song song tiên tiến hơn.
- **Mở rộng quy mô dữ liệu**: Tiến hành kiểm thử trên tập dữ liệu lớn hơn để đánh giá rõ ràng hơn về hiệu quả của sắp xếp song song.
- **Cải thiện quản lý tài nguyên**: Giảm thiểu chi phí khởi tạo luồng trong sắp xếp song song để cải thiện hiệu suất.
- **So sánh với các mô hình song song khác** như **Multiprocessing**, **MPI** để đưa ra lựa chọn tối ưu nhất.
- **Ứng dụng thực tế**: Triển khai trên các hệ thống yêu cầu xử lý dữ liệu lớn trong thời gian ngắn như hệ thống phân tích dữ liệu lớn, cơ sở dữ liệu phân tán.

❖ **Tổng kết chung:** Đồ án đã cung cấp cái nhìn tổng quan và sâu sắc về ứng dụng của thuật toán Bubble Sort trong môi trường lập trình song song. Kết quả đạt được đã minh chứng cho tiềm năng của việc tối ưu hóa hiệu suất qua song song hóa, mở ra nhiều hướng nghiên cứu và ứng dụng trong tương lai.

TÀI LIỆU THAM KHẢO

Stackoverflow:
https://stackoverflow.com/questions/tagged/threadpoolexecutor

Elearning:
https://elearning.vlu.edu.vn/pluginfile.php/669215/mod_resource/content/1/Th%E1%BB%B1c%20h%C3%A0nh%20-%20LT%20t%C3%ADnh%20to%C3%A1n%20song%20song%20-%20tu%E1%BA%A7n%2004.pdf

https://elearning.vlu.edu.vn/pluginfile.php/671798/mod_resource/content/1/Th%E1%BB%B1c%20h%C3%A0nh%20-%20LT%20t%C3%ADnh%20to%C3%A1n%20song%20song-tu%E1%BA%A7n%2005.pdf

Youtube:
https://www.youtube.com/watch?v=Vca808JTbI8

https://www.youtube.com/watch?v=ppmIOUIz4uI

ChatGPT, Coursera, Stackoverflow, Claude, W3school, …

Link GITHUB:

https://github.com/trantanphat0811/LapTrinhToanSongSong/tree/main/TranTanPhat_2274802010644_BaoCaoDoAn