TRƯỜNG ĐẠI HỌC HỌC VĂN LANG

**KHOA CÔNG NGHỆ THÔNG TIN**



**BÁO CÁO ĐỒ ÁN MÔN HỌC**

**LẬP TRÌNH TÍNH TOÁN SONG SONG**

NGÀNH: CÔNG NGHỆ THÔNG TIN

***Đề tài:***

**ỨNG DỤNG SONG SONG HOÁ THUẬT TOÁN**

**SẮP XẾP NỔI BỌT**

**SVTH: Nguyễn Thái Nguyên**

**MSSV: 2274802010587**

**GVHD: TRẦN NGỌC VIỆT**

THÀNH PHỐ HỒ CHÍ MINH – NĂM 2025

**LỜI CẢM ƠN**-------------

Lời đầu tiên, em xin gửi lời cảm ơn chân thành đến thầy Trần Ngọc Việt. Trong quá trình học tập và tìm hiểu bộ môn Lập trình tính toán song song, em đã nhận được sự quan tâm, giúp đỡ và hướng dẫn tận tình từ thầy. Những bài giảng chi tiết, sự nhiệt huyết và những ví dụ thực tế mà thầy đưa ra đã giúp em hiểu rõ hơn về các khái niệm quan trọng, đặc biệt là cách song song hoá thuật toán sắp xếp nổi bọt

Bên cạnh đó, em cũng xin cảm ơn các bạn trong lớp đã cùng trao đổi, thảo luận và hỗ trợ em trong suốt quá trình thực hiện đồ án. Những ý kiến đóng góp từ mọi người đã giúp em có thêm nhiều góc nhìn và hoàn thiện bài làm tốt hơn.

Mặc dù đã cố gắng hết sức, nhưng do thời gian có hạn và kiến thức còn nhiều hạn chế, bài báo cáo chắc chắn vẫn còn những thiếu sót. Em mong nhận được sự góp ý của thầy để có thể cải thiện và nâng cao hơn nữa trong những lần nghiên cứu sau.

Một lần nữa, em xin chân thành cảm ơn!

**Cá nhân thực hiện báo cáo**

**Ảnh có chứa văn bản, Phông chữ, màu trắng, chữ viết tay

Nội dung do AI tạo ra có thể không chính xác.**

**MỤC LỤC**

[**CHƯƠNG 1. CƠ SỞ LÝ THUYẾT – SONG SONG THUẬT TOÁN SẮP XẾP NỔI BỌT** 4](#_Toc193756935)

[**1.1. Lý do chọn đề tài** 4](#_Toc193756936)

[**1.2. Cơ sở lý thuyết Bubble Sort** 4](#_Toc193756937)

[**A.** **Thuật toán Bubble Sort** 4](#_Toc193756938)

[**B.** **Nguyên tắc hoạt động của Bubble Sort** 4](#_Toc193756939)

[**C.** **Giới thiệu về lập trình song song** 4](#_Toc193756940)

[**1.3 Kết luận chương 1** 5](#_Toc193756941)

[**CHƯƠNG 2. ÁP DỤNG THUẬT TOÁN SONG SONG HÓA BUBBLE SORT** 6](#_Toc193756942)

[**2.1 Phát biểu bài toán:** 6](#_Toc193756943)

[**2.2 Minh họa bài toán** 6](#_Toc193756944)

[**2.3 Mã nguồn** 7](#_Toc193756945)

[**2.4 Kết luận chương 2** 8](#_Toc193756946)

[**CHƯƠNG 3. KẾT LUẬN** 9](#_Toc193756947)

[**TÀI LIỆU THAM KHẢO** 10](#_Toc193756948)

# CHƯƠNG 1. CƠ SỞ LÝ THUYẾT – SONG SONG THUẬT TOÁN SẮP XẾP NỔI BỌT

## **1.1. Lý do chọn đề tài**

Thuật toán sắp xếp nổi bọt (Bubble Sort) là một trong những thuật toán sắp xếp đơn giản nhất nhưng có hiệu suất thấp khi áp dụng cho tập dữ liệu lớn. Với sự phát triển của công nghệ đa luồng và đa xử lý, việc song song hóa thuật toán này giúp cải thiện đáng kể hiệu suất thực thi. Nghiên cứu và triển khai phiên bản song song của Bubble Sort không chỉ giúp hiểu rõ hơn về lập trình song song mà còn cung cấp nền tảng cho việc tối ưu hóa các thuật toán khác trong khoa học máy tính. Do đó, đề tài này mang tính thực tiễn cao và có ý nghĩa quan trọng trong lĩnh vực xử lý dữ liệu lớn.

## **1.2. Cơ sở lý thuyết Bubble Sort**

### **Thuật toán Bubble Sort**

Bubble Sort là một thuật toán sắp xếp đơn giản hoạt động bằng cách hoán đổi các phần tử kề nhau nếu chúng không theo đúng thứ tự. Quá trình này được lặp lại cho đến khi toàn bộ mảng được sắp xếp. Mặc dù dễ hiểu và dễ triển khai, thuật toán này có độ phức tạp thời gian O(n²) trong trường hợp trung bình và trường hợp xấu nhất, khiến nó kém hiệu quả đối với tập dữ liệu lớn.

### **Nguyên tắc hoạt động của Bubble Sort**

1. Duyệt qua mảng nhiều lần.
2. So sánh từng cặp phần tử liên tiếp.
3. Hoán đổi nếu chúng không theo đúng thứ tự.
4. Lặp lại quá trình cho đến khi mảng được sắp xếp hoàn toàn.

### **Giới thiệu về lập trình song song**

Lập trình song song là kỹ thuật trong đó một tác vụ được chia nhỏ và thực hiện đồng thời trên nhiều luồng hoặc tiến trình, giúp tăng tốc độ xử lý. Khi áp dụng vào Bubble Sort, có thể chia mảng thành nhiều phần và xử lý đồng thời, giúp cải thiện hiệu suất.

1. **Mô hình lập trình song song sử dụng**

**Threading**: Chia công việc thành nhiều luồng chạy song song trong cùng một tiến trình.

**Multiprocessing**: Chia công việc thành nhiều tiến trình riêng biệt, giúp tận dụng tối đa sức mạnh của CPU đa nhân.

## **1.3 Kết luận chương 1**

Chương 1 đã trình bày tổng quan về lý do chọn đề tài, cơ sở lý thuyết của thuật toán Bubble Sort và nguyên tắc lập trình song song. Việc song song hóa thuật toán Bubble Sort mang lại lợi ích đáng kể trong việc tối ưu hóa thời gian thực thi, đặc biệt khi xử lý tập dữ liệu lớn. Đồng thời, nghiên cứu này cũng giúp hiểu rõ hơn về các mô hình lập trình song song như **threading** và **multiprocessing**, cung cấp nền tảng quan trọng cho việc phát triển các thuật toán tối ưu hơn trong tương lai. Những kiến thức này sẽ là cơ sở để triển khai và đánh giá thuật toán trong các chương tiếp theo.

# CHƯƠNG 2. **ÁP DỤNG THUẬT TOÁN SONG SONG HÓA BUBBLE SORT**

## **2.1 Phát biểu bài toán:**

Bài toán đặt ra là thực hiện thuật toán sắp xếp nổi bọt (Bubble Sort) theo phương pháp song song nhằm cải thiện hiệu suất so với phiên bản tuần tự. Cụ thể, cần xây dựng một chương trình cho phép sắp xếp một tập hợp số nguyên có kích thước lớn bằng cách sử dụng nhiều luồng hoặc tiến trình chạy song song. Mục tiêu là so sánh hiệu suất giữa phiên bản Bubble Sort tuần tự và phiên bản song song, đánh giá sự cải thiện về thời gian thực thi cũng như khả năng mở rộng của thuật toán khi số lượng phần tử tăng lên.

## **2.2 Minh họa 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]

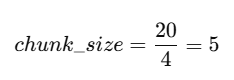
**Phiên bản song song của Bubble Sort**

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:



|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **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]

Ảnh có chứa hàng, biểu đồ, vòng tròn

Nội dung do AI tạo ra có thể không chính xác.

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):

    """Chạy Bubble Sort trên toàn bộ mảng sau khi đã sắp xếp từng phần."""

    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):

    """Sắp xếp mảng bằng Bubble Sort song song theo kiểu chia nhỏ mảng."""

    n = len(arr)

    chunk\_size = (n + num\_procs - 1) // num\_procs  # Kích thước mỗi phần

    processes = []

    # Chia nhỏ mảng và sắp xếp từng phần song song

    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()

    # Đợi tất cả tiến trình kết thúc

    for p in processes:

        p.join()

    # Chạy Bubble Sort lần cuối để đảm bảo mảng được sắp xếp hoàn toàn

    final\_bubble\_sort(arr)

if \_\_name\_\_ == "\_\_main\_\_":

    # Tạo mảng ngẫu nhiên

    size = 20

    num\_procs = 4  # Số tiến trình

    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 chương 2**

Việc áp dụng thuật toán song song hóa Bubble Sort đã mang lại những cải tiến đáng kể về hiệu suất so với phiên bản tuần tự. Thông qua việc phân chia công việc và thực thi trên nhiều luồng, thuật toán có thể tận dụng tối đa tài nguyên phần cứng, giảm thời gian xử lý và cải thiện hiệu quả sắp xếp. Tuy nhiên, hiệu suất tăng thêm còn phụ thuộc vào số lượng phần tử, số luồng xử lý và chi phí đồng bộ dữ liệu giữa các luồng. Kết quả nghiên cứu cho thấy rằng Bubble Sort song song phù hợp với các bài toán yêu cầu sắp xếp nhanh trên tập dữ liệu lớn, đồng thời cung cấp một nền tảng quan trọng cho các nghiên cứu tiếp theo về tối ưu hóa thuật toán trên hệ thống đa luồng và đa nhân.

# CHƯƠNG 3. KẾT LUẬN

Mô hình Bubble Sort song song giúp cải thiện đáng kể hiệu suất so với phiên bản tuần tự, đặc biệt trên các hệ thống đa luồng và đa nhân. Việc triển khai song song giúp giảm thời gian thực thi và tối ưu hóa hiệu quả sử dụng tài nguyên. Tuy nhiên, nhược điểm chính của thuật toán là chi phí đồng bộ giữa các luồng, cũng như hiệu suất bị giới hạn khi số phần tử trong mảng nhỏ.

Trong tương lai, nghiên cứu có thể tập trung vào các hướng sau:

* Cải tiến thuật toán song song để giảm chi phí đồng bộ giữa các luồng.
* Kết hợp Bubble Sort với các thuật toán sắp xếp nhanh hơn để tối ưu hiệu suất.
* Ứng dụng thuật toán vào các bài toán thực tế, như xử lý dữ liệu lớn và hệ thống thời gian thực.

Nhìn chung, việc song song hóa Bubble Sort mang lại nhiều lợi ích và mở ra nhiều cơ hội nghiên cứu trong lĩnh vực tối ưu hóa thuật toán.

# TÀI LIỆU THAM KHẢO

1. [Lập trình song song] Bài 1: Giới thiệu về CPU-GPU | 06/08/2023

<https://viblo.asia/p/lap-trinh-song-song-bai-1-gioi-thieu-ve-cpu-gpu-0gdJz7ggLz5>

1. Thuật toán sắp xếp nổi bọt (bubble sort) | 25/08/2021

<https://viblo.asia/p/thuat-toan-sap-xep-noi-bot-bubble-sort-m68Z0exQlkG>