Đại học quốc gia TP.HCM
Trường đại học công nghệ thông tin
Khoa khoa học máy tính
Lớp CS112.N11.KHTN

BÁO CÁO BÀI TẬP VỀ NHÀ

THUẬT TOÁN MERGESORT SONG SONG

Nhóm 4
Võ Trần Thu Ngân (21520069)
Tô Anh Phát (21520085)

Mục lục

1 Phát biểu bài toán **2**

2 Thuật toán Merge Sort tuần tự **2**

2.1 Thuật toán 2

2.1.1 Ý tưởng 2

2.1.2 Độ phức tạp 3

2.2 Cài đặt 3

3 Thuật toán Merge Sort song song **4**

3.1 Thuật toán 4

3.2 Cài đặt 6

3.2.1 Cấu trúc dữ liệu 6

3.2.2 Code tham khảo 6

3.2.3 Độ phức tạp 8

4 Kiểm thử và so sánh **9**

4.1 Môi trường thực hiện 9

4.2 Kiểm thử 9

4.3 So sánh 9

4.3.1 Đối tượng so sánh 9

4.3.2 Kết quả so sánh 9

4.3.3 Nhận xét 11

5 Kết luận **11**

6 Tài liệu tham khảo **12**

1 Phát biểu bài toán

Cho một dãy số nguyên gồm N phần tử. Yêu cầu: Cần thiết kế thuật toán song song thực hiện Merge Sort để sắp xếp dãy số theo thứ tự tăng dần.

Input: Số đầu tiên là số nguyên N - số phần tử của dãy. Tiếp theo là N số nguyên có trong dãy

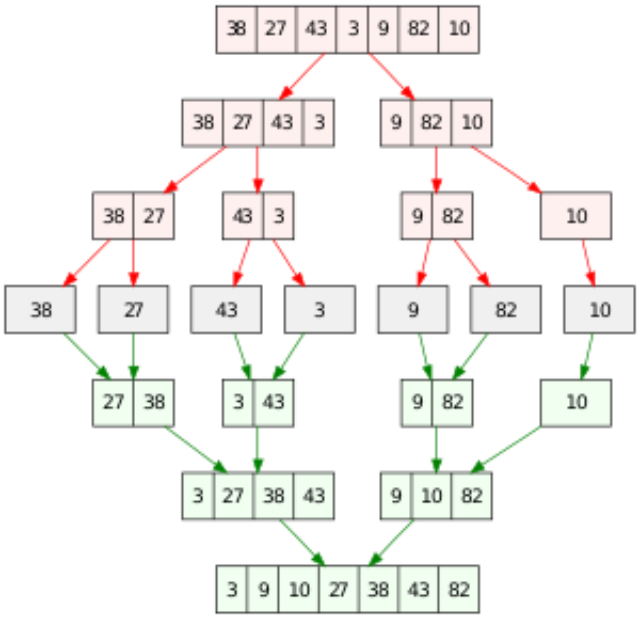
Output: Dãy số đã được sắp xếp theo thứ tự tăng dần

2 Thuật toán Merge Sort tuần tự

2.1 Thuật toán

2.1.1 Ý tưởng

Merge Sort tuần tự sử dụng cách tiếp cận chia để trị. Ý tưởng của thuật toán là chia dãy chưa sắp xếp thành các đoạn nhỏ hơn, sắp xếp và gộp chúng lại thành một dãy được sắp xếp. Hình sau mô tả cách thực hiện của thuật toán:



Hình 1: Thuật toán Merge Sort tuần tự

Các bước của thuật toán, sử dụng kĩ thuật đệ quy:

1. Nếu dãy đầu vào có ít hơn 2 phần tử, return.
2. Chia dãy thành hai phần bằng nhau (hoặc chênh lệch nhau 1 phần tử).
3. Gọi hàm đệ quy để thực hiện thuật toán Merge Sort cho hai dãy trên.
4. Thực hiện gộp hai dãy đã sắp xếp thành một dãy theo thứ tự tăng dần. Đây là dãy output.

2.1.2 Độ phức tạp

- Độ phức tạp thời gian: $O(NlogN)$ (cả trong worst case, average case và best case) vì thuật toán thực hiện chia dãy thành hai nửa tại mỗi bước và mất thời gian tuyến tính để gộp hai dãy lại.
- Độ phức tạp không gian: $O(N)$

2.2 Cài đặt

Sau đây là cách cài đặt tham khảo cho thuật toán Merge Sort tuần tự, sử dụng Python:

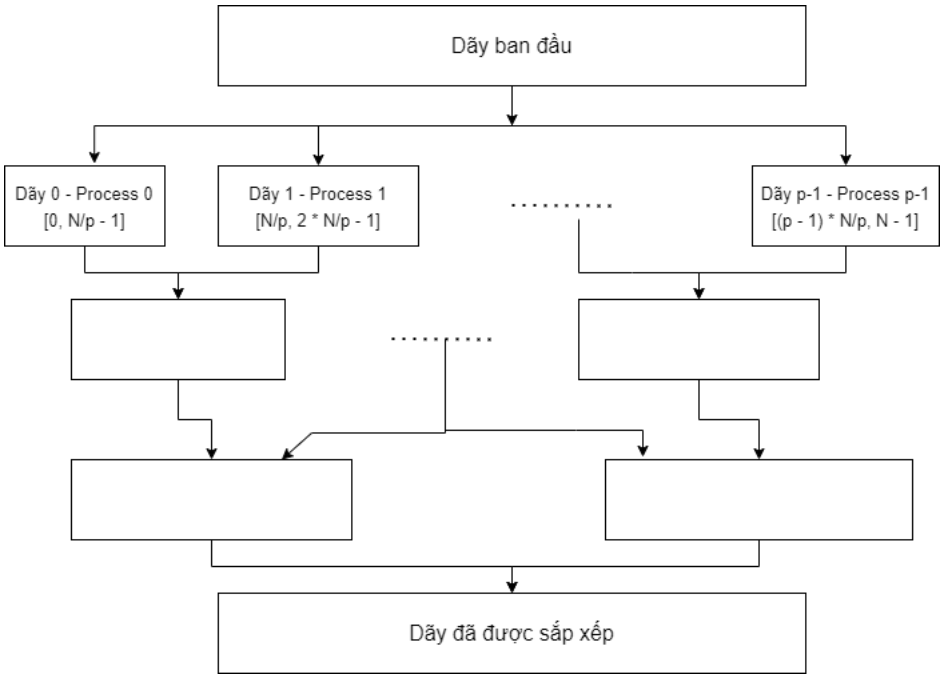
```
# Merge 2 phần sau khi sort
def merge(left, right):
        res = []

        n_left = len(left)
        n_right = len(right)
        i = j = 0

        # Chọn phần tử nhỏ nhất ở cả hai mảng để đưa vào vị trí tiếp theo
        while i < n_left and j < n_right:
                if left[i] <= right[j]:
                        res.append(left[i])
                        i += 1
                else:
                        res.append(right[j])
                        j += 1
```

```
        # Xếp lần lượt các phần tử còn lại vào phía sau các phần tử đã sắp xếp
        if i < n_left:
                res += left[i:]

        if j < n_right:
                res += right[j:]

        return res

# Thực hiện thuật toán Merge Sort
def merge_sort(a):
        n = len(a)

        if n == 1:
                return a

        # Chia thành 2 phần
        mid = n // 2

        # Dùng merge sort cho phần bên trái
        left = merge_sort(a[:mid])
        # Dùng merge sort cho phần bên phải
        right = merge_sort(a[mid:])

        # Merge 2 phần đã sắp xếp
        res = merge(left, right)

        return res
```

3 Thuật toán Merge Sort song song

3.1 Thuật toán

Ý tưởng của cách cài đặt song song là sử dụng nhiều processes, mỗi processes sẽ đảm nhiệm việc thực hiện thuật toán Merge Sort tuần tự trên một đoạn của dãy số. Sau đó sẽ tiếp tục sử dụng nhiều processes để gộp các đoạn đã được sắp xếp thành dãy output. Hình sau mô tả ý tưởng cài đặt song song với p chỉ số processes sử dụng:



Hình 2: Thuật toán Merge Sort song song

Các bước thực hiện thuật toán song song như sau:

1. Chia dãy ban đầu thành p (số processes) dãy con, dãy con thứ i chứa đoạn $[i.N/p, (i+1)N/p - 1]$.
2. Process thứ i thực hiện thuật toán Merge Sort tuần tự trên dãy thứ i.
3. Lưu p dãy con đã được sắp xếp vào data
4. Nếu data chỉ chứa một dãy, đến bước 8.
5. Process thứ i thực hiện merge dãy data$[2 * i]$ và data$[2 * i + 1]$.
6. Cập nhật lại data lưu các dãy kết quả mới
7. Quay lại bước 4
8. Trả về data[0] là dãy đã được sắp xếp.

3.2 Cài đặt

3.2.1 Cấu trúc dữ liệu

- Để hiện thực giải thuật trên, ta có thể dùng lớp `Pool` thư viện `multiprocessing` của Python.
- Một đối tượng `Pool` có thể quản lý một tập các processes và phân các tasks cho chúng. `Pool` cung cấp cơ chế song song hóa việc thực thi một hàm với nhiều input khác nhau thông qua việc tự động gán các input cho các processes và thực thi chúng song song. Ví dụ:

```
from multiprocessing import Pool

def f(x):
    return x*x

if __name__ == '__main__':
     with Pool(5) as p:
        print(p.map(f, [1, 2, 3]))
```

Kết quả in ra là $[1, 4, 9]$

3.2.2 Code tham khảo

Sau đây là cách cài đặt tham khảo cho thuật toán Merge Sort song song, sử dụng thư viện multiprocessing của Python:

```
from multiprocessing import Pool
import time
import random

# Merge 2 phần sau khi sort
def merge(data):
        left, right = data

        res = []

        n_left = len(left)
        n_right = len(right)
```

```
        i = j = 0

        # Chọn phần tử nhỏ nhất ở cả hai mảng để đưa vào vị trí tiếp theo
        while i < n_left and j < n_right:
                if left[i] <= right[j]:
                        res.append(left[i])
                        i += 1
                else:
                        res.append(right[j])
                        j += 1

        # Xếp lần lượt các phần tử còn lại vào phía sau các phần tử đã sắp xếp
        if i < n_left:
                res += left[i:]
        if j < n_right:
                res += right[j:]

        return res

# Hàm merge sort
def merge_sort(a):
        n = len(a)
        if n == 1:
                return a
        # Chia thành 2 phần
        mid = n // 2
        # Dùng merge sort cho phần bên trái
        left = merge_sort(a[:mid])
        # Dùng merge sort cho phần bên phải
        right = merge_sort(a[mid:])

        # Merge 2 phần đã sắp xếp
        res = merge((left, right))

        return res

# Chia đều n phần tử cho các bộ xử lý
def merge_sort_parallel(a, NUM_PROCESS):
        p = Pool(NUM_PROCESS)
```

```
    items = []

    for i in range(NUM_PROCESS):
            # processor thứ i sẽ xử lý trong khoảng (i * n // NUM_PROCESS,
            ↪  (i+1) * n // NUM_PROCESS - 1)
            low = i * n // NUM_PROCESS
            high = (i+1) * n // NUM_PROCESS - 1
            items.append(a[low:high+1])

    # Các phần sau khi xử lý sẽ đưa vào data
    data = p.map(merge_sort, items)

    # Phân processes để merge các phần lại

    while len(data) > 1:
            extra = data.pop() if len(data) % 2 == 1 else None
            data = [(data[i], data[i + 1]) for i in range(0, len(data), 2)]
            data = p.map(merge, data) + ([extra] if extra else [])

    return data[0]
```

Chi tiết code có thể xem tại repository GitHub của nhóm:

3.2.3 Độ phức tạp

1. Độ phức tạp không gian: $O(N)$
2. Độ phức tạp thời gian
    - Xét bước thực hiện Merge Sort: vì các processes cùng thực hiện thuật toán Merge Sort tuần tự nên số phép toán xấp xỉ $\frac{N}{p}.log(\frac{N}{p})$, với p là số processes sử dụng. Vậy độ phức tạp tương ứng là $O(NlogN)$.
    - Xét bước thực hiện merge các dãy con đã sắp xếp:
        - Số bước thực hiện merge hai dãy có độ dài s là $2s - 1$
        - Sau một lần lặp `while`, số dãy con cần merge giảm đi một nửa, vì vậy số lần lặp của vòng lặp `while` tối đa là $logN$. Vậy số bước tối

đa cần thực hiện để có được dãy kết quả là:

$$\Sigma_{i=1}^{logN}(2^i - 1)$$

tương ứng với độ phức tạp là $O(N)$.

- Vậy tổng độ phức tạp thời gian của thuật toán là $O(NlogN)$.

4 Kiểm thử và so sánh

4.1 Môi trường thực hiện

1. Phần cứng
   - CPU: AMD Ryzen 5 4500U (6 cores, 6 threads, 2.5GHz - 4.0GHz)
   - RAM: 12GB DDR4 3200MHz
2. Phần mềm: Chương trình được chạy trên Visual Studio Code, Python 3.1

4.2 Kiểm thử

Thực thi chương trình cài đặt thuật toán Merge Sort song song trên các test-cases được sinh ngẫu nhiên với các kích thước khác nhau trong đoạn $[50, 10^7]$.

Kết quả kiểm thử: Thuật toán cho ra kết quả đúng trên tất cả testcases.

4.3 So sánh

4.3.1 Đối tượng so sánh

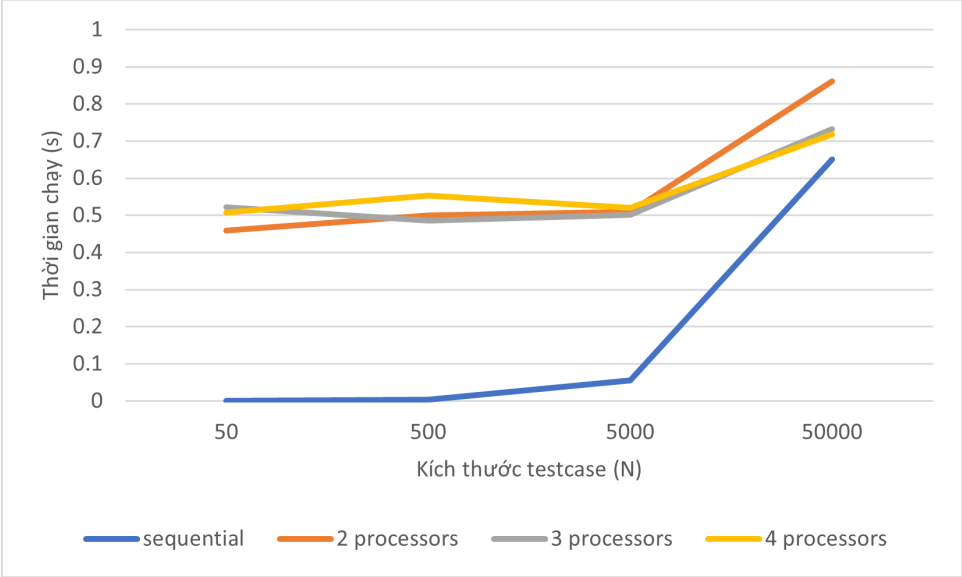
Chạy thử và so sánh thuật toán Merge Sort tuần tự và song song với số lượng processes là 2, 3, 4 trên các testcases được sinh ngẫu nhiên với số phần tử trong đoạn $[50, 10^7]$

4.3.2 Kết quả so sánh

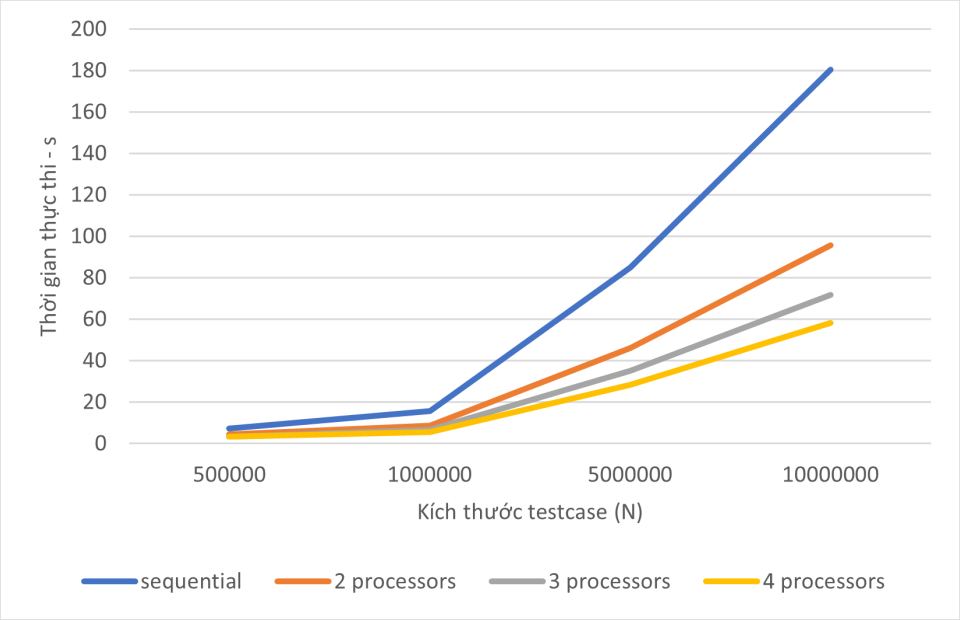
Kết quả được thể hiện trong bảng sau:

Thuật toán \ N	50	500	5000	50000	500000	1000000	5000000	10000000
Sequential Merge Sort	0.00033	0.00417	0.05458	0.64977	7.33055	15.590708	85.05088	180.56426
parallel with 2 processes	0.45943	0.4994	0.50918	0.8605	4.40522	8.81671	46.02554	95.64511
parallel with 3 processes	0.52131	0.48556	0.50193	0.73249	3.4253	6.804577	35.08721	71.72688
parallel with 4 processes	0.50804	0.55243	0.52052	0.71772	3.24518	5.705487	28.40463	58.13970

Hình 3: Thời gian thực thi của các thuật toán trên các testcases (đơn vị: giây)



Hình 4: Đồ thị so sánh thời gian thực thi trên các testcases có kích thước nhỏ ($N \leq 50000$)



Hình 5: Đồ thị so sánh thời gian thực thi trên các testcases có kích thước lớn ($5.10^5 \leq N \leq 10^7$)

4.3.3 Nhận xét

Qua thực nghiệm, có thể thấy thuật toán Merge Sort song song chỉ nên sử dụng trong trường hợp dãy số có kích thước lớn (khoảng $\geq 10^5$), vì với những testcases có kích thước nhỏ hơn, thì thời gian khởi tạo các processes sẽ lớn hơn so với thời gian thực thi thuật toán Merge Sort tuần tự.

Ngoài ra, cũng có thể thấy được, với những testcases có kích thước lớn, việc tăng thêm processes sẽ giúp giảm thời gian thực thi đi đáng kể (giảm khoảng 20% thời gian cho mỗi process tăng lên).

5 Kết luận

Vậy bài báo cáo này đã đưa ra một giải thuật và cách cài đặt song song theo thuật toán Merge Sort để sắp xếp dãy số theo thứ tự tăng dần. Đồng thời, bài báo cáo đã đưa ra so sánh về thời gian thực thi giữa thuật toán tuần tự với thuật toán song song, giữa các cách cài đặt thuật toán song song với số lượng processes khác nhau.

6 Tài liệu tham khảo

1. *Alexandra Yang* (26/11/2022). Approaches to the Parallelization of Merge Sort in Python

2. *Ricardo Rocha and Fernando Silva, University of Porto*. Slides of Parallel Computing 2015/2016

3. *Python Documentation*. multiprocessing — Process-based parallelism