ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
ĐẠI HỌC CÔNG NGHỆ THÔNG TIN
KHOA KHOA HỌC MÁY TÍNH

CS112.N21.KHTN - Phân tích và thiết kế thuật toán

Báo cáo bài tập tuần 2

Thuật toán song song

Giảng viên: ThS. Nguyễn Thanh Sơn

THÀNH PHỐ HỒ CHÍ MINH, THÁNG 3 2023

Thành viên

No.	Họ tên	MSSV
1	Lê Hoài Thương	21520474
2	Nguyễn Hoàng Hải	21522034

Contents

1 Phát biểu bài toán

Sử dụng 1 thuật toán song song để sắp xếp lại 1 dãy số theo thứ tự tăng dần theo thuật toán merge sort.
Input: 1 dãy số gồm n phần tử.
Output: Dãy số n phần tử đã được sắp xếp theo thứ tự tăng dần

2 Thuật toán Merge sort

Thuật toán Merge Sort là một thuật toán sắp xếp đệ quy dựa trên việc chia mảng thành các mảng con, sắp xếp các mảng con và sau đó kết hợp các mảng con đã sắp xếp để tạo ra một mảng hoàn chỉnh đã sắp xếp.
Các bước thực hiện thuật toán Merge Sort:

1. Chia mảng ban đầu thành hai mảng con bằng cách chọn một chỉ số trung tâm.
2. Đệ quy sắp xếp hai mảng con.
3. Kết hợp hai mảng con đã sắp xếp để tạo ra một mảng hoàn chỉnh đã sắp xếp.

Mã giả:

```
1 Function MergeSort(Array,l,r):
2     if l > r then
3         return
4     end
5     mid = (l + r)/2
6     MergeSort(Array, l, mid)
7     MergeSort(Array, mid + 1, r)
8     sortedArray =Merge(array, l, mid, right)
9     return sortedArray
```

Algorithm 1: Sequential Merge sort

3 Thiết kế giải thuật

3.1 Ý tưởng thuật toán

Giả sử có k processors. Ta chia đều mảng n phần tử đều cho k process. Khi đó, mỗi process sẽ được chia một đoạn gồm không quá n/k phần tử. Các process song song tiến hành sắp xếp lại các đoạn đó theo thuật toán Merge sort tuần tự. Sau khi sắp xếp xong k đoạn này, ta tiếp tục phân chia công việc gộp các đoạn này lại cho các process cho đến khi tất cả các đoạn được gộp lại thành một.
Mô hình thuật toán được mô tả như hình sau:

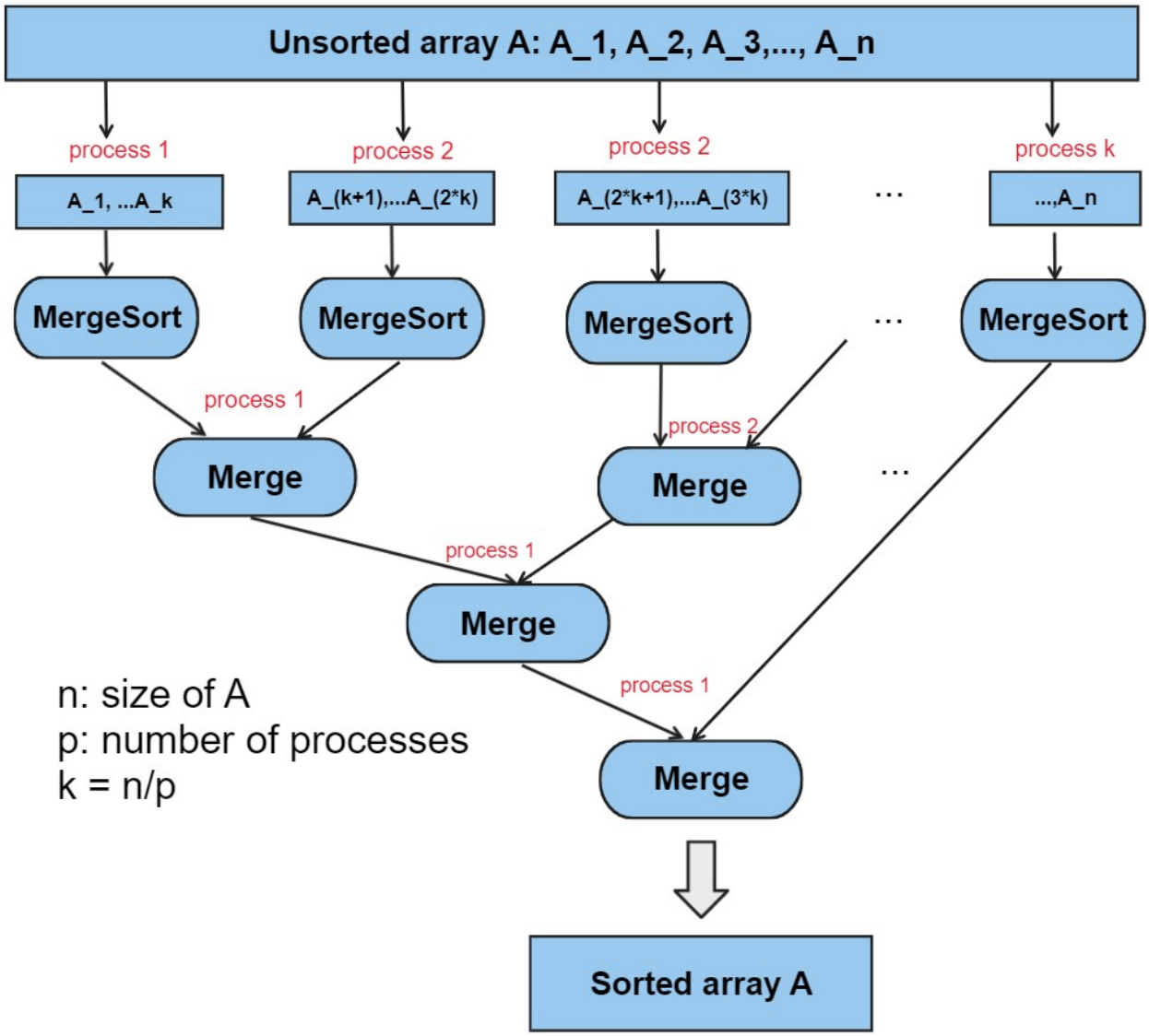

Figure 1: Sơ đồ thuật toán Parallel merge sort

3.2 Mã giả

```
Function MergeSortParallel(Array):
    Divide the array into k parts data[1], data[2],...data[k]
    for i in k parts do
        Assign i − th part to i − th process
        sortedData[i] = MergeSort(process[i],data[i])
        while len(sortedData)>1 do
            if len(ans) odd then
                Odd = sortedData.last
            else
                Odd = None
            end
            i = 1
            for i < len(sortedData) do
                Merge(sortedData[i], sortedData[i+1], process[i])
                i = i + 2
            end
        end
        sortedData.append(Odd)
    end
    return sortedData
```

Algorithm 2: Parallel Merge Sort

4 Code

Sau đây là phần code Python thực nghiệm thuật toán parallel merge sort (nguồn: https://gist.github.com/stephenmcd/39ded69946155930c347) đã được nhóm em điều chỉnh và chạy được code ở cả hai nền tảng Google Colaboratory và Kaggle. Cấu trúc dữ liệu mà chúng em sử dụng để chứa dãy số là **list**, kiểu dữ liệu của các số trong dãy là **interger**.

- Khai báo các thư viện cần thiết

```
import math
import multiprocessing
import random
import sys
import time
```

- Thuật toán gộp hai mảng đã được sắp xếp tăng dần thành một mảng tăng dần (bước 3 mục 2)

```
def merge(*args):
    # Support explicit left/right args, as well as a two-item
    # tuple which works more cleanly with multiprocessing.
    left, right = args[0] if len(args) == 1 else args
    left_length, right_length = len(left), len(right)
    left_index, right_index = 0, 0
    merged = []
    while left_index < left_length and right_index < right_length:
        if left[left_index] <= right[right_index]:
            merged.append(left[left_index])
            left_index += 1
        else:
            merged.append(right[right_index])
            right_index += 1
    if left_index == left_length:
        merged.extend(right[right_index:])
    else:
        merged.extend(left[left_index:])
    return merged
```

- Thuật toán merge sort (bước 1 và 2 mục 2)

```python
def merge_sort(data):
    length = len(data)
    if length <= 1:
        return data
    middle = int(length / 2)
    left = merge_sort(data[:middle])
    right = merge_sort(data[middle:])
    return merge(left, right)
```

- Thuật toán parallel merge sort.

```python
def merge_sort_parallel(data):
    # Creates a pool of worker processes, one per CPU core.
    # We then split the initial data into partitions, sized
    # equally per worker, and perform a regular merge sort
    # across each partition.
    processes = multiprocessing.cpu_count()
    print(processes)
    pool = multiprocessing.Pool(processes=processes)
    size = int(math.ceil(float(len(data)) / processes))
    data = [data[i * size:(i + 1) * size] for i in range(processes)]
    data = pool.map(merge_sort, data)
    # Each partition is now sorted - we now just merge pairs of these
    # together using the worker pool, until the partitions are reduced
    # down to a single sorted result.
    while len(data) > 1:
        # If the number of partitions remaining is odd, we pop off the
        # last one and append it back after one iteration of this loop,
        # since we're only interested in pairs of partitions to merge.
        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 = pool.map(merge, data) + ([extra] if extra else [])
    return data[0]
```

Ở thuật toán parallel merge sort, trọng tâm chính là việc sử dụng hàm pool.map(), một phương thức trong module multiprocessing của Python, được dùng để thực hiện các tác vụ song song trên một tập hợp các đối tượng.

Cụ thể, pool.map() nhận vào một hàm và một tập hợp các đối tượng, và áp dụng hàm đó vào mỗi đối tượng trong tập hợp đó. Kết quả được trả về là một danh sách các kết quả tương ứng với mỗi đối tượng

trong tập hợp.
Phương thức này có thể được sử dụng để tăng tốc độ xử lý cho các tác vụ mà có thể được chia nhỏ thành nhiều phần, và các phần này có thể được xử lý độc lập và song song trên nhiều luồng. Các tác vụ đó có thể được thực hiện trên một hoặc nhiều CPU, tùy thuộc vào cấu hình hệ thống.
Các bước tiếp theo của thuật toán parallel merge sort sau khi có các dãy số được sắp xếp theo thứ tự tăng dần chính là bước cuối cùng trong thuật toán merge sort (mục 2), kết hợp các dãy con để có được dãy số ban đầu theo thứ tự tăng dần.

- Đối sánh parallel merge sort và merge sort trong trường hợp mảng chứa 1 triệu phần tử ngẫu nhiên

```python
if __name__ == "__main__":
    size = 1000000
    data_unsorted = [random.randint(0, size) for _ in range(size)]
    for sort in merge_sort, merge_sort_parallel:
        start = time.time()
        data_sorted = sort(data_unsorted)
        end = time.time() - start
        print(sort.__name__, end, sorted(data_unsorted) == data_sorted)
```

Source code tại github

5 Tính đúng đắn và hiệu quả của thuật toán

5.1 Tính đúng đắn

Về tổng quan, thuật toán parallel merge sort có 3 bước giống với thuật toán merge sort (mục 2), chỉ khác ở điểm nó có bước chia dãy số ban đầu thành k đoạn và phân phối các tác vụ sắp xếp k đoạn đó cho GPU/CPU thực thi các process (ở đây, thuật toán merge sort được dùng để sắp xếp các đoạn này), các tác vụ sẽ được xử lý độc lập và song song trên nhiều luồng, điều này giúp tăng tốc độ thực thi thuật toán. Việc phân phối các tác vụ mà vẫn đảm bảo được tính toàn vẹn và tính đúng trong việc thực thi thuật toán merge sort chính là điều đặc biệt nhất của thuật toán parallel merge sort, bởi lẽ, điều này có thể xảy ra vì thuật toán merge sort mang tư tưởng điển hình của một thuật toán chia để trị. Vì vậy, thuật toán parallel merge sort, cũng tương tự như thuật toán merge sort, có tính chính xác tuyệt đối trong việc giải quyết bài toán đặt ra ban đầu.
Bên cạnh đó, chúng em đã thực nghiệm tính đúng đắn cũng như tính hiệu quả về mặt thời gian của thuật toán parallel merge sort trong 4 trường hợp có số process được sử dụng khác nhau, lần lượt là: 2, 4, 8, 16. Với mỗi trường hợp, chúng em thử nghiệm với 100 dãy số được sinh ngẫu nhiên có số phần tử lần lượt là 2, 10002, 20002, 30002,... Việc thực nghiệm hoàn toàn được làm trên nền tàng Kaggle, sử dụng TPU VM v3-8 cho việc chạy code. Kết quả thực nghiệm cho thấy việc dùng thuật toán parallel merge sort để giải quyết bài toán luôn cho ra kết quả chính xác. Hơn thế nữa, khi vẽ biểu đồ tỉ lệ giữa thời gian thực thi thuật toán merge sort và thời gian thực thi thuật toán parallel merge sort trên cùng một dãy số, chúng em đã có những nhận xét đáng kinh ngạc về tính hiệu quả của thuật toán parallel merge sort.

5.2 Hiệu quả

Dù có cùng độ phức tạp tính toán, thuật toán parallel hoàn thành trong thời gian ngắn hơn đáng kể so với thuật toán merge sort truyền thống vì các processors thực hiện các phép toán song song. Dưới đây là biểu đồ thể hiện tỉ lệ giữa thời gian chạy merge sort tuần tự và parallel merge sort:

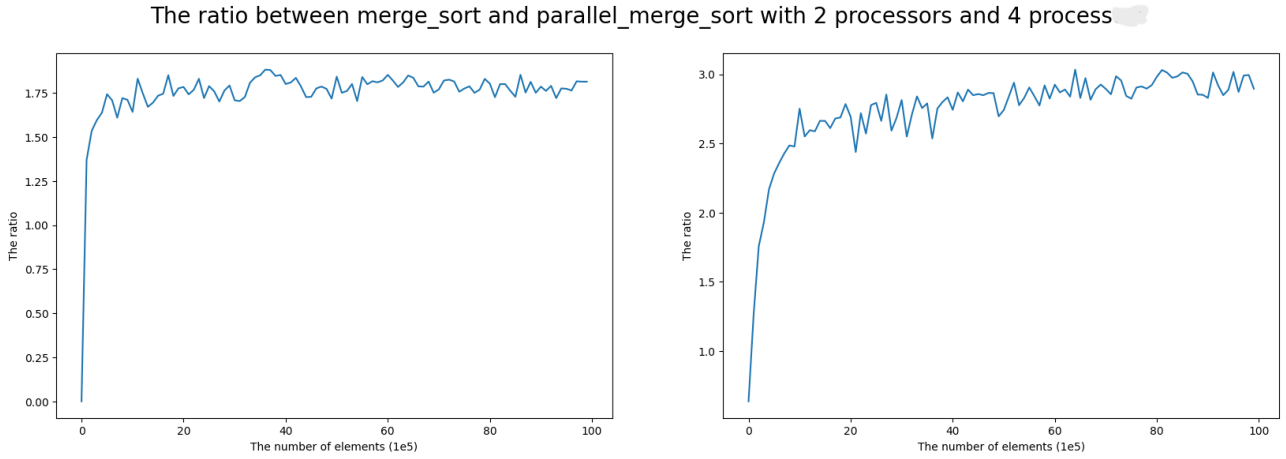

Figure 2: So sánh thuật toán merge sort và parallel merge sort khi dùng 2 và 4 processes song song

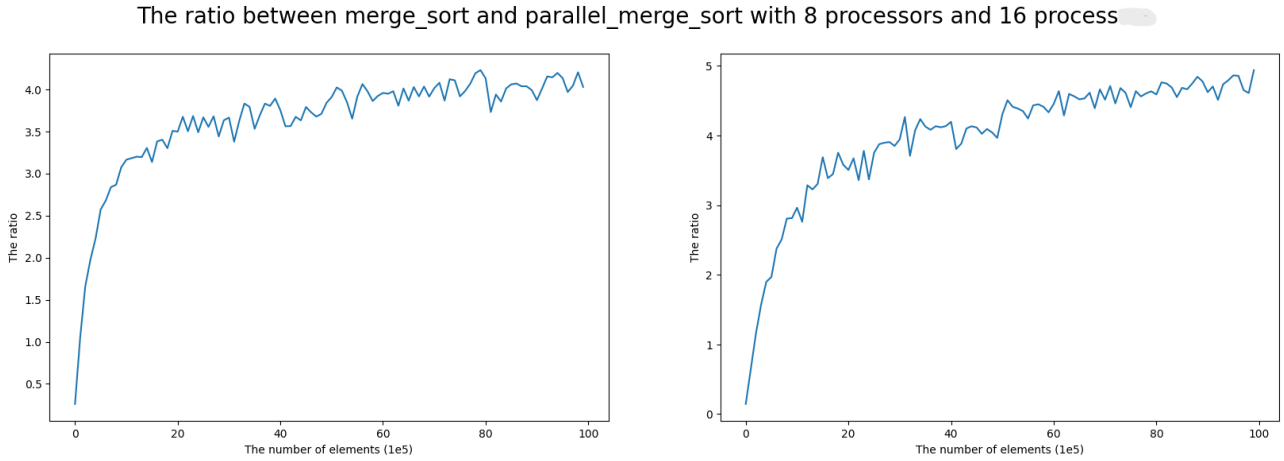

Figure 3: So sánh thuật toán merge sort và parallel merge sort khi dùng 8 và 16 processes

Nhận xét:

- Ta nhận thấy với số processor càng lớn thì chênh lệnh trong thời gian xử lí giữa 2 thuật toán càng tăng(tỉ số giữa thời gian chạy merge_sort/ thời gian chạy parallel tăng).

Ngoài ra, chúng em tiếp tục thử nghiệm thuật toán parallel merge sort với 2 và 4 process với 100 dãy số được sinh ngẫu nhiên, có số phần tử nhỏ hơn nhiều so với các thử nghiệm trên, lần lượt là 1, 101, 201,...

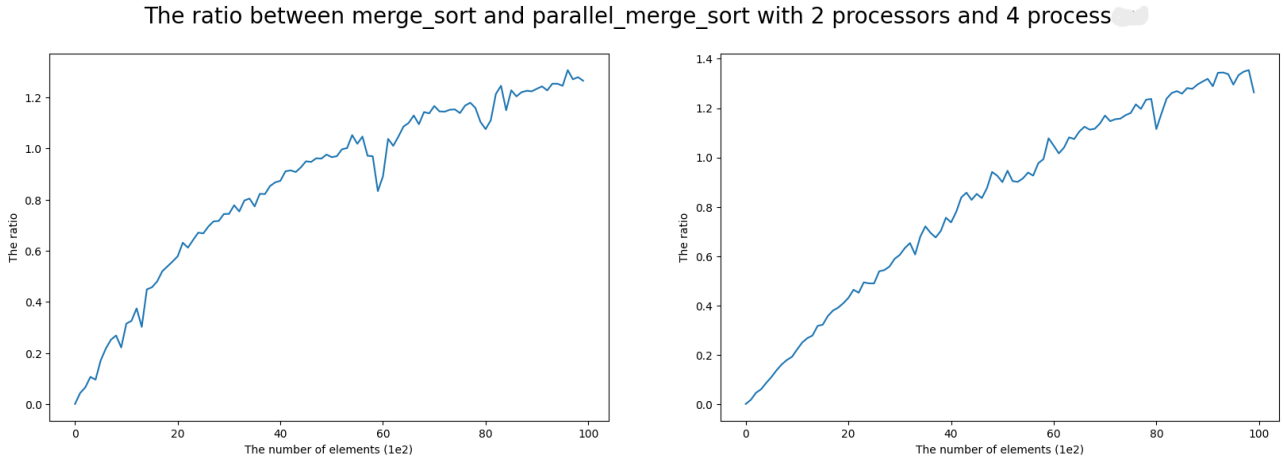

Figure 4: So sánh thuật toán merge sort và parallel merge sort khi dùng 2 và 4 processes trên cách test nhỏ

Nhận xét:

- Khi số phần tử trong mảng cần sắp xếp nhỏ thì merge_sort hiệu quả hơn parallel_merge_sort vì hàm pool.map tốn thời gian đáng kể để chia việc cho các process.

6 Phân tích độ phức tạp

6.1 Độ phức tạp thuật toán merge sort nguyên bản

- Độ phức tạp không gian lưu trữ: $O(n)$
- Độ phức tạp tính toán của thuật toán merge hai mảng đã được sắp xếp thành một mảng n phần tử: $O(n)$
- Độ phức tạp tính toán thuật toán merge sort: $T(n) = 2 * T(n/2) + \Theta(n) \Rightarrow$ Áp dụng master theorem: $O(nlog_2n)$:
- Độ phức tạp tính toán của merge sort ở các trường hợp khác nhau:
  - Độ phức tạp tính toán trường hợp tốt nhất: $O(nlog_2n)$
  - Độ phức tạp tính toán trung bình: $O(nlog_2n)$
  - Độ phức tạp tính toán trường hợp xấu nhất: $O(nlog_2n)$

Vậy thuật toán merge sort có độ phức tạp bộ nhớ là $O(n)$ và độ phức tạp tính toán là $O(nlog_2n)$

6.2 Độ phức tạp thuật toán parallel merge sort

Với k là số processor, thuật toán parallel merge sort chúng em trình bày sẽ chia dãy số ban đầu thành k đoạn, mỗi đoạn có $\lceil \frac{n}{k} \rceil$ (đoạn cuối cùng có $n - \lceil \frac{n}{k} \rceil * (k-1)$ phần tử). Các processor sẽ được yêu cầu thực hiện thuật toán merge sort trên mỗi đoạn, vậy nên độ phức tạp tính toán ở bước này, để có k đoạn được sắp xếp, là $O(k * \frac{n}{k} log_2 \frac{n}{k}) = O(nlog_2 \frac{n}{k})$.

Độ phức tạp tính toán của bước merge k mảng thành một mảng có công thức là: $T(n) = 2T(\frac{n}{2}) + \Theta(n)$ với $(n > \frac{\text{The number of elements of an original array}}{k})$, giả sử không mất tính tổng quát: $\frac{n}{k} = 2^d$.

Áp dụng master theorem, ta có: $T(n) = T_1(n_1) - T_2(n_2)$ với $T_1(n_1) = 2T_1(\frac{n_1}{2}) + \Theta(n_1)$ và $T_2(n_2) = 2T_2(\frac{n_2}{2}) + \Theta(n_2)$ và n_1 là n (số phần tử của bản ban đầu), $n_2 = \frac{n_1}{k} \Rightarrow T(n) = O(nlog_2n - \frac{n}{k} log_2 \frac{n}{k})$.

Vậy độ phức tạp tính toán của thuật toán parallel merge sort là $O(nlog_2n + (n - \frac{n}{k}) log_2 \frac{n}{k}) = O(nlog_2n)$ và độ phức tạp bộ nhớ của thuật toán này cũng tương tự với thuật toán merge sort, $O(n)$.

7 Chương trình minh họa

- Chương trình Parallel merge sort tại mục 4 (Code): github
- Chương trình thực nghiệm Parallel merge sort tại mục 5: github