ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN
KHOA CÔNG NGHỆ PHẦN MỀM

CS112.N21.KHTN

Thiết kế thuật toán song song để thực hiện sắp xếp dãy tăng dần theo thuật toán Merge Sort

GV hướng dẫn : Nguyễn Thanh Sơn
Nhóm thực hiện: 14

Nguyễn Hoàng Thuận	:	21521501
Lê Thị Liên	:	21522282

Thành phố Hồ Chí Minh, 13 tháng 3 năm 2023

Contents

I. Understanding the Problem

Bài toán đặt ra là sắp xếp một mảng các số nguyên theo thứ tự tăng dần bằng cách sử dụng thuật toán Merge Sort. Mục tiêu là thiết kế một thuật toán song song có thể tăng tốc quá trình sắp xếp.

- Input của bài toán là một dãy số nguyên không có thứ tự, cần được sắp xếp tăng dần.
- Output của bài toán là một dãy số nguyên đã được sắp xếp tăng dần theo thuật toán merge sort.

II. Ascertaining the Capabilities of the Computational Device

Sử dụng thư viện multiprocessing trên Python, cho phép xử lý song song trên CPU đa lõi hoặc một cụm máy tính. Số lượng core khả dụng xác định số lượng process tối đa có thể chạy song song.

III. Choosing between Exact and Approximate Problem Solving

Sắp xếp một mảng các số nguyên là một vấn đề chính xác có thể được giải quyết bằng các thuật toán xác định.

IV. Algorithm Design Techniques

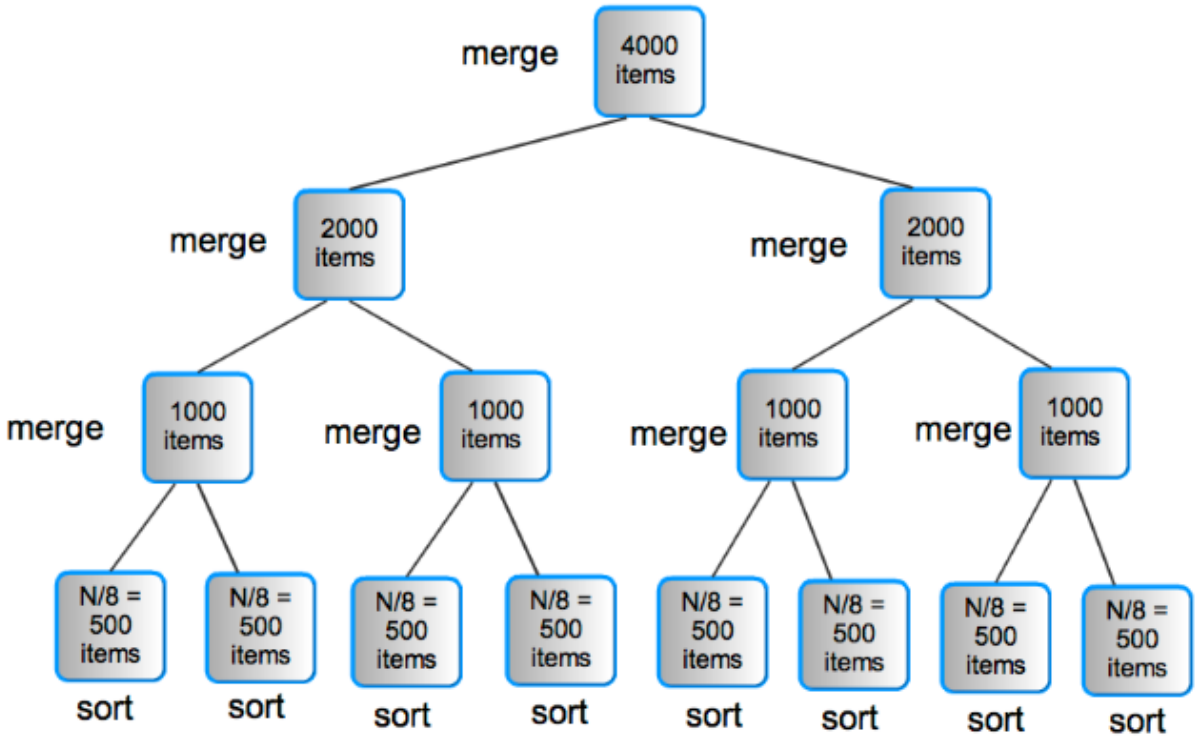
Thuật toán Merge Sort song song có thể được thiết kế bằng cách sử dụng kỹ thuật Chia để Trị, trong đó mảng được phân tách đệ quy thành các mảng con cho đến khi mỗi mảng con chỉ chứa một phần tử. Sau đó, các mảng con được merge song song từng cặp cho đến khi mảng ban đầu được sắp xếp.

V. Designing an Algorithm and Data Structures

Thuật toán có thể được thiết kế như sau:

- Chia mảng đầu vào thành numproc mảng con bằng nhau, trong đó numproc là số lượng process sẽ được sử dụng.
- Chỉ định mỗi mảng con cho một process riêng biệt..
- Mỗi process thực hiện merge sort tuần tự trên mảng con được chỉ định của nó.
- Merge theo cặp song song các mảng con đã sắp xếp cho đến khi mảng ban đầu được sắp xếp.

Ví dụ :



Scaling parallel merge sort: an example where the number of data items, N is 4000 and the number of processors, n is 8.

VI. Methods of Specifying an Algorithm

Mã giả cho thuật toán :

```
function parallel_merge_sort(arr, nproc):
begin
    if len(arr) > 1:
    begin

        // Divide the array into subarrays
        args = divide_array(arr, nproc)

        // Create a thread pool
        pool = create_pool(nproc)

        // Sort each subarray in parallel
        for subarray in args:
        begin
            task = create_sort_task(subarray)
            pool.add_task(task)
        end

        // Wait for all sorting tasks to complete
```

```
        pool.wait_for_completion()

        // Merge all subarrays in parallel
        while len(args) > 1:
        begin
            merged_subarrays = []
            for i in range(0, len(args), 2):
                if i+1 < len(args):
                    task = create_merge_task(args[i], args[i+1])
                    pool.add_task(task)
                else:
                    merged_subarrays.append(args[i])
            pool.wait_for_completion()
            args = merged_subarrays
        end

        return args[0]
    end

    else:
        // The array has only one element
        return arr
end

function create_merge_task(arr1, arr2):
begin
    i = 0
    j = 0
    res = []

    while i<len(left) and j<len(right):
        if left[i]<=right[j]:
            res.append(left[i])
            i+=1
        else:
            res.append(right[j])
            j+=1

    while(i<len(left)):
        res.append(left[i])
        i+=1

    while(j<len(right)):
        res.append(right[j])
        j+=1

    return res
end

function divide_array(arr, nproc):
begin
    res = []
    for (i = 0, k = len(arr) / nproc; i < nproc; i++)
    begin
        cur = []
        for (j = i * k, t = 0; t < k; t++, j++)
        begin
            cur.append[arr[j]]
        end
        res.append(cur)
    end
```

```
    return res
end

function main()
begin
    arr, nproc
    read(arr, nproc)
    res = parallel_merge_sort(arr, nproc)
    print(res)
end
```

VII. Analyzing an Algorithm

- Độ phức tạp về thời gian của thuật toán merge sort song song là O(n log n), trong đó n là kích thước của mảng đầu vào.
    - Mỗi lần chia nhỏ dãy đều đôi, do đó ta có thể đạt được log2 n cấp chia nhỏ dãy.
    - Sau đó, trộn các dãy con lại với nhau, trong đó, ta phải xem xét tất cả các phần tử của mỗi dãy con ít nhất một lần để trộn chúng. Do đó, số lần trộn các dãy con lại với nhau cũng là O(n).
    - Vì vậy, tổng thời gian của thuật toán merge sort là O(n log n).

Worst Case	Best Case	Averge Case
O(n log n)	O(n log n)	O(n log n)

- Độ phức tạp không gian là O(n) do sử dụng các mảng tạm thời trong quá trình hợp nhất.

VIII. Coding an Algorithm

- **Github** : *https://github.com/hoho303/CS112.N21.KHTN/tree/Parallel-Algorithm*

IX. Run the code

Kích thước mảng	Thời gian chạy	
	Merge Sort	Parallel Merge Sort
1000	0.003	0.18
100000	0.4	0.26
500000	2.35	0.785
1000000	4.788	1.38
10000000	~ 1 phút	15.79

- Nếu kích thước mảng quá nhỏ, chi phí phân chia và ghép các mảng con lại tốn nhiều khá thời gian. Vì vậy, đối với các kích thước mảng nhỏ, việc sử dụng thuật toán tuần tự có thể hiệu quả hơn thuật toán song song.
- Khi kích thước mảng lớn hơn thì thuật toán song song thường có khả năng chạy nhanh hơn. Điều này xảy ra vì khi kích thước mảng lớn hơn, việc sử dụng song song giúp tận dụng tốt các tài nguyên có sẵn trên các thiết bị tính toán hiện đại, đồng thời giảm thiểu thời gian truyền dữ liệu và tăng tốc độ tính toán.
- Tốc độ tăng tốc đạt được bằng thuật toán song song phụ thuộc vào số lượng lõi có sẵn và kích thước của mảng đầu vào.