TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

UIT

Môn: Phân tích và thiết kế thuật toán

Đề tài: Thuật toán song song (Merge Sort)

Họ và tên: Nguyễn Thanh Hỷ

Lớp: KHTN2022

MSSV: 22520593

Giảng viên hướng dẫn: Nguyễn Thanh Sơn

TP.HCM, ngày 28, tháng 9, năm 2023

THUẬT TOÁN SONG SONG MERGE SORT

Như đã học ở môn Cấu trúc dữ liệu và giải thuật, thuật toán Merge Sort dựa trên ý tưởng chia để trị (Divide and Conquer)

Merge Sort có độ phức tạp về thời gian là O(nlogn) và độ phức tạp về thời gian là O(n)

- Ý tưởng của thuật toán:
- Liên tục chia mảng thành nhiều phần nhỏ cho đến khi chỉ mỗi phần nhỏ chỉ còn một phần tử.
- Sau đó ta thực hiện quá trình "trộn" hai mảng nhỏ này lại, ta sẽ có được một mảng gồm hai phần tử đã được sắp xếp.
- Và những mảng đã được sắp xếp này lại tiếp tục được "trộn" lại với nhau cho đến khi chỉ còn một mảng cuối cùng.

➔ Và mảng cuối cùng này chính là mảng đích.

Đây là source code của thuật toán Merge Sort, gồm hai hàm:

- Hàm "merge": Dùng để "trộn" hai mảng.

```python
def merge(*args):
    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
```

- Hàm “merge_sort”: Là hàm chính dùng để sắp xếp.

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

Từ đầu đến giờ chính là phần Merge Sort mà ta đã học trước đó.

PHẦN CHÍNH

- Phần thuật toán song song Merge Sort:
- Thuật toán này dựa trên ý tưởng dùng độc lập từng cpu khác nhau để sắp xếp trên những mảng con nhỏ và sau đó ghép chúng lại thành một mảng hoàn chỉnh.

- Muốn chia mảng và dung riêng biệt từng cpu ta sẽ sử dụng thư viên “multiprocessing” của Python và phương thức chính ta dùng trong phần này là “Pool” và “map”.

```python
def merge_sort_parallel(arr, num_processes):
    if len(arr) < num_processes:
        num_processes = len(arr)  # Chỉ sử dụng số tiến trình bằng số phần tử nếu dãy quá nhỏ
    chunk_size = len(arr) // num_processes

    # Chia dãy thành các phần nhỏ
    chunks = [arr[i:i + chunk_size] for i in range(0, len(arr), chunk_size)]

    # Sắp xếp các phần nhỏ bằng các tiến trình riêng biệt
    with multiprocessing.Pool(num_processes) as pool:
        sorted_chunks = pool.map(merge_sort, chunks)

    # Hợp nhất các phần nhỏ đã sắp xếp để có mảng đã sắp xếp hoàn chỉnh
    sorted_arr = sorted_chunks[0]
    for i in range(1, len(sorted_chunks)):
        sorted_arr = merge(sorted_arr, sorted_chunks[i])

    return sorted_arr
```

Giải thích code:

- Đầu tiên ta sẽ cần phải biết được số process của thiết bị, đó là tham số “num_processes”, tham số “arr” chính là mảng ta cần sắp xếp.
- Ta nhân thấy: Nếu mảng có số phần tử ít hơn số process thì ta sẽ không sử dụng tất cả process của thiết bị mà chỉ sử dụng một phần bằng độ dài của mảng.
- Sau đó ta thực hiện việc chia mảng thành nhiều mảng con sao cho số lượng mảng con bằng với biến num_processes.
- Ta sẽ dung cả phương thức “Pool” và phương thức “map” để thực hiện quá trình merge sort song song trên các mảng con này.
- Sau khi đã có được danh sách các mảng con đã được sắp xếp, cuối cùng ta thực hiện merge chúng lại.

➔ Kết quả chính là mảng sau khi đã được sắp xếp bằng thuật toán song song.

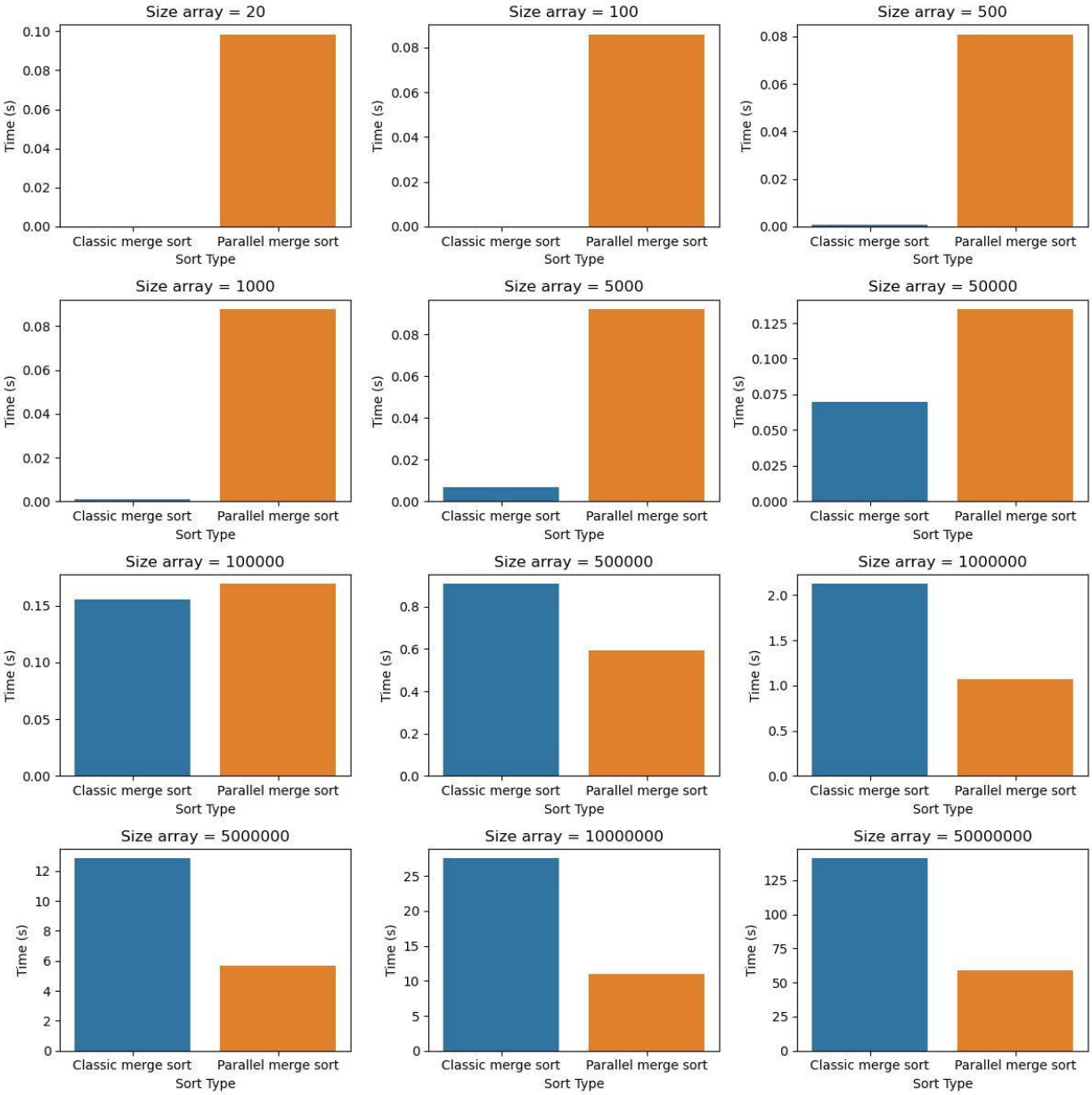
Kiểm tra và đánh giá:

```python
arr = np.arange(0, 21, 1)
merge_sort_parallel(arr, 20)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
```

Ở đây em đã thực hiện việc kiểm tra trên một mảng nhỏ trước và nhận thấy thuật toán hoạt động chính xác khi đã sắp xếp tuần tự các giá trị trong mảng.

So sánh thuật toán merge sort song song và merge sort cổ điển.



Ở đây em đã thử nghiệm trên 12 mảng có độ lớn và giá trị khác nhau, độ lớn các mảng lần lượt là “Size array” trong ảnh.

Ta có thể thấy đối với những mảng có kích thước nhỏ (dưới 100 000 phần tử) thì thuật toán merge sort song song sẽ phải mất nhiều thời gian hơn để sắp xếp xong.

Nhưng thuật toán merge sort song song sẽ phát huy được sức mạnh của mình khi độ lớn của mảng lớn hơn 100 000 phần tử, ta có thể thấy được trong ảnh: nếu độ lớn của mảng hơn 1

000 000 thì thuật toán merge sort song song chỉ mất nửa thời gian để thực hiện việc sắp xếp, thử nghĩ nếu ta có 1 mảng với kích thước 1 000 000 000 hay thậm chí là nhiều phần tử thì sao, cách biệt thời gian đó là rất lớn.

Tổng kết:

Thuật toán merge sort cổ điển hiệu quả hơn thuật toán merge sort song song trong trường hợp số lượng phần tử còn sắp xếp nhỏ, thuật toán merge sort song song sẽ hiệu quả hơn trong trường hợp dữ liệu là lớn và rất lớn.

Tóm lại:

- Mọi thuật toán đều có điểm mạnh và điểm yếu riêng, ta cần phải sử dụng chúng cho hợp lí, cho phù hợp với bài toán mà ta đang phải đối mặt.
- Thuật toán song song là một thuật toán hiệu quả khi có thể khai phá hết khả năng của cpu, dung trong xử lí dữ liệu lớn thì thuật toán này đạt một sức mạnh không tưởng !

Link Github: [nguyenthanhhy0108/CS112-MergeSortParallel: MergeSortParallel (github.com)]

Colab: https://colab.research.google.com/drive/1EpSTgjynfj95XOkOTomtA9kWjlaqFSya#scrollTo=HKdCh7i5s4FK

THANK YOU!