Báo cáo bài tập Thiết kế thuật toán song song - Nhóm 11

1) Mục tiêu:

Chọn CTDL và thiết kế thuật toán song song để thực hiện sắp xếp dãy tăng dần theo thuật toán merge sort

2) Quy trình thực hiện:

- Chọn CTDL.
- Thiết kế thuật toán.
- Tạo dataset
- Viết chương trình.
- Thực hiện sắp xếp các tập dữ liệu bằng thuật toán Merge Sort truyền thống.
- Thực hiện sắp xếp các tập dữ liệu bằng thuật toán Parallel Merge Sort tự thiết kế.
- So sánh thời gian chạy và rút ra nhận xét.

3) Tiến hành:

a) Chọn CTDL:

Cấu trúc dữ liệu của dataset là list.

<u>Lý do:</u>

- Thao tác đơn giản: Có các thao tác cơ bản vòng lặp, điều kiện,...
- Cung cấp hàm sort() dựa trên thuật toán TimSort.
- Dễ thực hiện thuật toán chạy song song: Python có hỗ trợ các thư viện hỗ trợ xử lí đa luồng hoặc đa tiến trình như multiprocessing và threading.
- Hỗ trợ trong việc vẽ biểu đồ, in kết quả để visualize kết quả chương trình.

Các biến sử dụng :

arr: Đây là mảng (list) chứa dữ liệu cần sắp xếp.

chunk_size: Kích thước của mỗi "chunk" (mảng con) được tạo từ mảng gốc để chia công việc sắp xếp thành các phần nhỏ hơn để thực hiện song song.

chunks: Một danh sách (list) chứa các "chunk" được tạo ra từ mảng gốc.

sorted_chunks: Một danh sách chứa các "chunk" đã được sắp xếp bằng cách sử dụng nhiều tiến trình song song.

sorted_arr: Mảng cuối cùng sau khi hợp nhất (merge) các "chunk" đã sắp xếp.

time_list: Một từ điển (dictionary) được sử dụng để lưu trữ thời gian thực hiện của thuật toán sắp xếp truyền thống và thuật toán sắp xếp song song.

b) Thiết kế thuật toán:

1. Chia mảng thành các mảng con (chunks):

   Trước khi thực hiện sắp xếp, mảng đầu vào (arr) được chia thành các mảng con bằng cách cắt mảng gốc thành các phần nhỏ hơn.

   Số lượng phần con (chunks) được tạo ra phụ thuộc vào biến process_count (số lượng tiến trình) và kích thước của mảng (chunk_size).

2. Sắp xếp các mảng con đồng thời:

   Mỗi mảng con trong danh sách chunks được sắp xếp độc lập.

   Sử dụng một pool của nhiều tiến trình (thông qua multiprocessing.Pool), các mảng con này được sắp xếp song song bằng cách gọi hàm sorted trên từng mảng con riêng lẻ.

   Mỗi tiến trình trong pool sẽ xử lý một mảng con, cho phép việc sắp xếp các phần tử của mảng con này diễn ra đồng thời và hiệu quả hơn.

3. Hợp nhất (Merge) các mảng con đã sắp xếp:

   Sau khi tất cả các mảng con đã được sắp xếp, kết quả từ các mảng con này được kết hợp lại để tạo ra một mảng đã sắp xếp hoàn chỉnh. Quá trình hợp nhất sử dụng hàm merge, một phần của mã mà bạn không đã đưa ra.

4. Đo thời gian thực hiện:

   Thời gian thực hiện của thuật toán song song được đo từ thời điểm bắt đầu chia mảng con cho đến khi kết thúc quá trình hợp nhất để tạo mảng đã sắp xếp.

5. Trả kết quả và thời gian thực hiện:

Hàm parallel_merge_sort trả về mảng đã sắp xếp và thời gian thực hiện của thuật toán song song.

4) Tạo dataset:

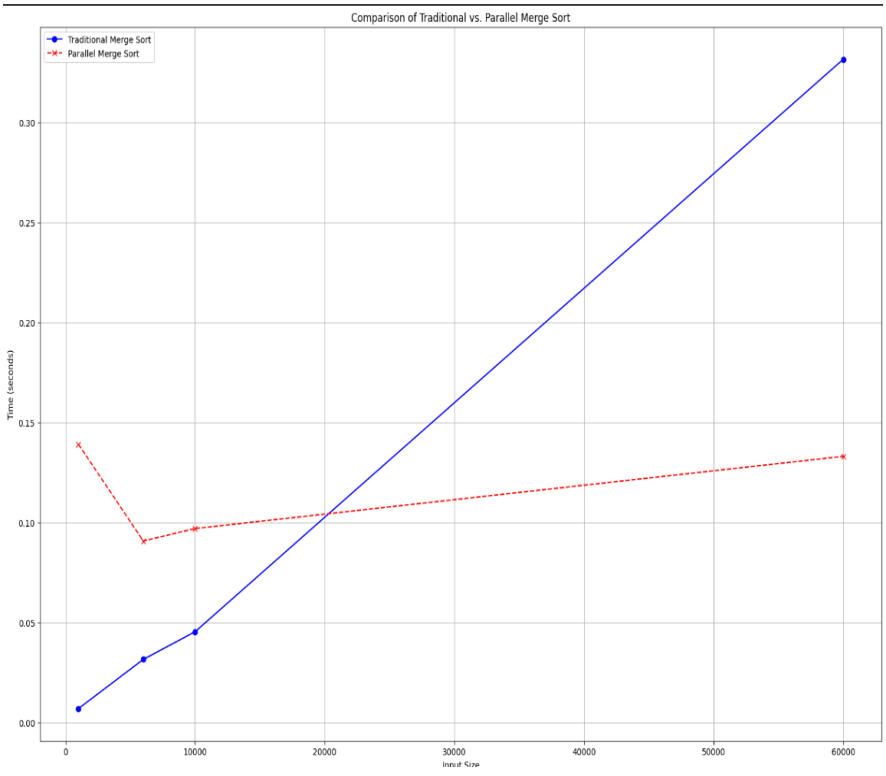
Sinh các bộ dữ liệu lần lượt gồm 10^3, $5*10^3$, 10^4, $5*10^4$, 10^5, $5*10^5$, 10^6, $5*10^6$, 10^7 số tự nhiên được chọn ngẫu nhiên trong [1,1,000,000,000).

5) Viết chương trình:

Đọc source code tại Github hoặc Colab.

6) Kết quả chạy thực nghiệm:

```
Size = 1000
Traditional Merge Sort Time: 0.007036 seconds
Parallel Merge Sort Times: 0.139084 seconds

Size = 6000
Traditional Merge Sort Time: 0.031610 seconds
Parallel Merge Sort Times: 0.090787 seconds

Size = 10000
Traditional Merge Sort Time: 0.045451 seconds
Parallel Merge Sort Times: 0.097006 seconds

Size = 60000
Traditional Merge Sort Time: 0.331449 seconds
Parallel Merge Sort Times: 0.133076 seconds

Size = 100000
Traditional Merge Sort Time: 0.596950 seconds
Parallel Merge Sort Times: 0.155604 seconds

Size = 600000
Traditional Merge Sort Time: 5.605855 seconds
Parallel Merge Sort Times: 0.596374 seconds

Size = 1000000
Traditional Merge Sort Time: 7.305054 seconds
Parallel Merge Sort Times: 1.626063 seconds

Size = 6000000
Traditional Merge Sort Time: 58.060172 seconds
Parallel Merge Sort Times: 6.144167 seconds

Size = 10000000
Traditional Merge Sort Time: 103.568086 seconds
Parallel Merge Sort Times: 11.216323 seconds
```



7) So sánh kết quả - Nhận xét:

Ưu điểm của Parallel Merge Sort:

Tận dụng nhiều lõi CPU: Thuật toán này sử dụng song song hóa để tận dụng nhiều lõi CPU hoặc tiến trình để sắp xếp mảng con đồng thời, giúp cải thiện tốc độ sắp xếp trên các hệ thống có nhiều CPU hoặc lõi CPU mạnh.

Phân chia công việc: Việc chia mảng thành các mảng con và sắp xếp độc lập trên từng mảng con giúp giảm thời gian thực hiện tổng cộng so với việc sắp xếp toàn bộ mảng một lúc.

Nhược điểm của Parallel Merge Sort:

Tốn tài nguyên: Song song hóa có thể tạo ra nhiều tiến trình hoặc luồng, dẫn đến tốn tài nguyên hệ thống. Điều này có thể không hiệu quả trên các hệ thống có tài nguyên hạn chế.

Phức tạp hơn: Implementing một thuật toán song song thường phức tạp hơn so với phiên bản đơn giản hơn của nó, và có thể gây khó khăn trong việc xử lý lỗi và điều chỉnh hiệu suất.

Kết luận :

Việc sử dụng thuật toán song song có thể mang lại hiệu suất cao hơn cho một loạt các ứng dụng và tác vụ, nhưng nó cũng đòi hỏi sự cân nhắc kỹ lưỡng và phải xem xét cụ thể từng trường hợp để đảm bảo rằng lợi ích của việc song song vượt qua được các chi phí và khó khăn của nó(overhead do cần phải quản lý các tiến trình, tài nguyên hệ thống, v.v..). Thuật toán Parallel Merge Sort tốt hơn so với thuật toán truyền thống với bộ dữ liệu kích thước lớn, với bộ dữ liệu kích thước nhỏ do việc chia và quản lý dữ liệu dẫn đến sự trì hoãn(overhead) đáng kể nên không đạt được hiệu quả mong muốn.

Nguồn tham khảo:

Do you know the time complexity of Python's Sorted() function? - Charan - Medium
Approaches to the Parallelization of Merge Sort in Python by Alexandra Yang
Parallel Merge Sort - OpenGenus IQ