CS112 - Phân tích và thiết kế thuật toán

Parralel Algorithm
Group 6 Report

Sinh viên : *Trần Nhật Khoa - 22520591, Lê Trần Quốc Khánh - 22520638*

9 - 2023

Mục lục

1 Lời nói đầu

Đây là source code về report của nhóm 6 về **Thuật toán song song** cho thuật toán **Merge Sort**, ngoài ra còn có một Notebook dự đoán chiều cao dựa trên cân nặng:

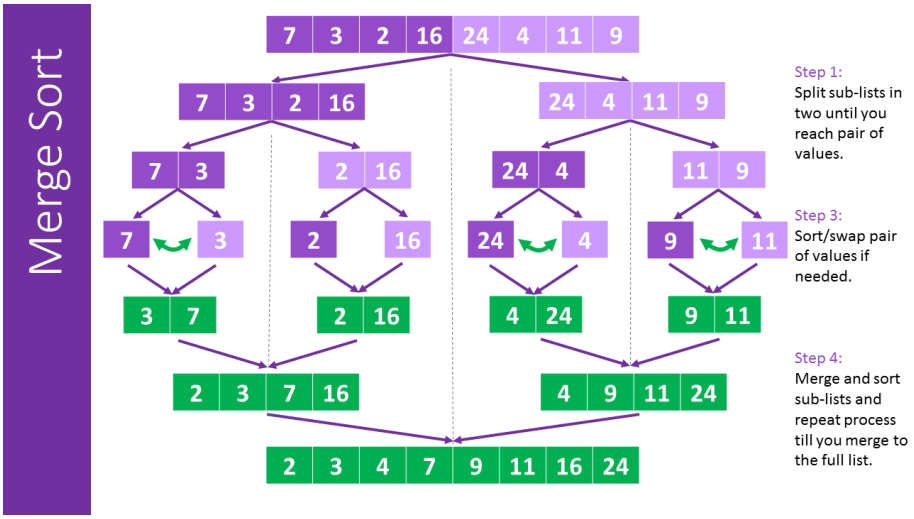
Link collab:
`https://colab.research.google.com/drive/1I8GIOtiLTByuJ-ZsNb2jKOTh5H5NfJTT?hl=vi#scrollTo=5MCs75ObgRHj`

Link soure code:

`https://github.com/LeeKhanhs/CS112/tree/main/Multy_process`

2 Merge Sort truyền thống

Thuật toán sắp xếp trộn (**Merge Sort** truyền thống lấy ý tưởng chia để trị để giảm thời gian. Đây là ảnh minh họa cho thuật toán:



Link code:

`https://github.com/LeeKhanhs/CS112/blob/main/Multy_process/merge_sort.py`

Tuy nhiên, hướng thuật toán xử dụng thông thường sẽ là gọi đệ quy, và các process được đưa vào **hàng đợi** để chờ được xử lý. Điều này khá hạn chế khi các process này có thể hoạt động **cùng lúc** mà **không ảnh hưởng tới nhau**.

3 Merge Sort với Parallel algorithm

Vì lý do trên mà **thuật toán song song** ra đời. Trong trường hợp Merge sort, ta có thể nghĩ đơn giản là chia process chính của chúng ta thành các child process, mỗi child process sẽ là một mảng con sẽ được sắp xếp, sau đó ta sẽ hợp nhất các kết quả của các child process, từ đó giảm được thời gian cho process chính.

Theo kiến thức của em về Hiệu điều hành, thì việc chia thành các child process nó còn giảm về tài nguyên vì các process sẽ chia sẻ một số thành phần memory như memory ở vùng **heap**, **stack**, **data** cho child process, và mỗi child process sẽ được **phân vùng cpu** phân vùng cpu từ process chính, sau đó xử lý trên các cpu này một cách song song.

Link code:

`https://github.com/LeeKhanhs/CS112/blob/main/Multy_process/merge_sort.py`

4 Related work

Đầu tiên chúng em tiến hành gen array data có kích thước khác nhau, sau đó tiến hành code và thử nghiệm đo thời gian trên các mảng có độ dài khác nhau. Cụ thể chúng em tiến hành code theo kiến thức của hệ điều hành sử dụng thư viện **multyprocess** (tương tự một số chức năng như **fork** ở các ngôn ngữ khác), và có tham khảo thuật toán từ wikipeida:

```
// Sort elements lo through hi (exclusive) of array A.
algorithm mergesort(A, lo, hi) is
    if lo+1 < hi then  // Two or more elements.
        mid := ⌊(lo + hi) / 2⌋
        fork mergesort(A, lo, mid)
        mergesort(A, mid, hi)
        join
        merge(A, lo, mid, hi)
```

Link code:

`https://github.com/LeeKhanhs/CS112/blob/main/Multy_process/improve.py`

`https://github.com/LeeKhanhs/CS112/blob/main/Multy_process/main.py`

5 Kết luận

1. Với những mảng có độ dài nhỏ thì việc xử lý đa luồng có vẻ chậm hơn với cách xử lý truyền thống. Có vẻ là do các tiến trình song song phải đợi cùng nhau để có thể merge lại về sau (vì không đảm bảo tiến trình nào sẽ xong trước nên em sẽ code theo format dùng **start** và dùng **join**). Ngược lại với mảng có độ dài tầm 10^6 phần tử, thì xử lý đa luồng trên máy của em nhanh hơn gần gấp 4 lần, hơn 1,5 lần trên Google collab so với cách xử lý thông thường

2. Việc tăng **worker**(lõi cpu) có thể giúp ta cải tiến về mặt thời gian, nhưng có thể bị xung đột về memory do các process có thể không đủ tài nguyên để xử lý.

– Hết –