TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN TOÁN ỨNG DỤNG & TIN HỌC

BÀI GIẢNG

TÍNH TOÁN SONG SONG

Giảng viên: Đoàn Duy Trung
Bộ môn : Toán Tin

BÀI 4. THIẾT KẾ CÁC CHƯƠNG TRÌNH SONG SONG

Nội dung bài học

- Song song tự động và song song thủ công
- Nhận thức về vấn đề và chương trình có thể song song được
- Phân hoạch
- Truyền thống
- Tích tụ
- Ánh xạ
- Phụ thuộc dữ liệu
- Cân bằng tải

5.1.Việc song song hóa

- Bao gồm 2 việc:
  - Nhận dạng các phần công việc có thể thực thi song song
  - Thực hiện việc song song
- Thực hiện phức tạp, tốn thời gian, hay bị lỗi, lặp..
- Công cụ hỗ trợ song song:
  - Trình biên dịch song song hóa
  - Bộ tiền vi xử lý

5.1.Việc song song hóa

- Bao gồm 2 dạng:
  - Thủ công
  - Tự động
  - Bán tự động
- Thủ công:
  - Việc song song hoàn toàn do lập trình viên thực hiện

5.1.Việc song song hóa

- Tự động hoàn toàn:
  - Trình biên dịch phân tích mã nguồn và xác định cơ hội song song
  - Phân tích này bao gồm việc tìm ra các phần khó có cơ hội song song, các trọng số có thể để cải thiện hiệu năng
  - Các vòng lặp (do, for) là đối tượng xem xét khi cần song song tự động

5.1.Việc song song hóa

- Bán tự động:
  - Có sự can thiệp của lập trình viên:
    - Chỉ thị của lập trình viên, cần chèn vào đoạn nào?
    - Sử dụng cờ biên dịch phải chỉ rõ khi nào cần sử dụng song song.
- Lưu ý khi sử dụng song song hóa tự động:
  - Đưa ra kết quả sai
  - Hiệu năng giảm
  - Hạn chế một số đoạn lệnh
  - Đoạn mã có thể không cần song song

5.2.Bài toán và chương trình song song

- Hiểu được vấn đề , công việc cần giải quyết song song.
- Nếu có một chương trình tuần tự cần phải hiểu rõ mã nguồn, thuật toán của tuần tự.
- Bài toán có song song được không?

5.2.Bài toán và chương trình song song

- Ví dụ về bài toán có thể song song hóa được:
  - Tính toán năng lượng trong vài miền phân tử độc lập, tìm năng lượng tối thiểu
  - Tính tổng hai véc tơ A(n) và B(n) có cùng số phần tử
- Ví dụ về bài toán không song song được:
  - Tính chuỗi Fibonaci bằng cách sử dụng công thức F(k+2)=F(k+1)+F(k)
  - Tính n! sử dụng chu trình For hoặc dùng hàm đệ quy.

5.2.Bài toán và chương trình song song

- Nếu chương trình song song được:
  - Xác định các điểm nóng trong chương trình có thể / không tính toán song song được.
    - Xác định điểm sử dụng nhiều tài nguyên CPU, bỏ qua những điểm ít sử dụng CPU
  - Xác định điểm tắc nghẽn trong chương trình
    - Phần thực hiện với thời gian không cân đối với các phần tử khác => phá hủy cơ hội song song
    - Cơ cấu lại chương trình hoặc sử dụng thuật toán khác

5.2.Bài toán và chương trình song song

- Nếu chương trình song song được:
  - Nhận diện các hạn chế xử lý song song:
    - Lớp hạn chế phổ biến nhất là sự phụ thuộc dữ liệu (data dependence). VD dãy Fibonaci
    - Nghiên cứu thuật toán khác để thay thế
    - Tận dụng lợi thế của các phần mềm song song cũng như những thư viện toán học hiệu quả của các nhà cung cấp.

5.2.Bài toán và chương trình song song

- Đặc điểm của chương trình song song:
  - Đồng thời (Concurrency)
  - Tỷ lệ (Scalability)
  - Địa phương (locality)
  - Mô đun (modularity)

5.2.Bài toán và chương trình song song

- Đặc điểm của chương trình song song
  - Đồng thời (Concurrency): Nhằm cho phép thực hiện trên nhiều bộ xử lý
  - Tính tỷ lệ (Scalability) (co giãn được, hoặc mở rộng):
    - Thuật giải cài đặt mà không quan tâm đến số lượng bộ vi xử lý
    - P tăng => task giảm

5.2.Bài toán và chương trình song song

- Đặc điểm của chương trình song song:
  - Tính địa phương (locality): giảm chi phí thời gian của thuật giải:
    - Do việc truy cập đến local data thường xuyên hơn việc truy cập đến remote data
  - Tính mô đun hóa (modularity): Phân hoạch những thực thể phức tạp thành các thành phân đơn giản

5.3.Phân chia (partitioning)

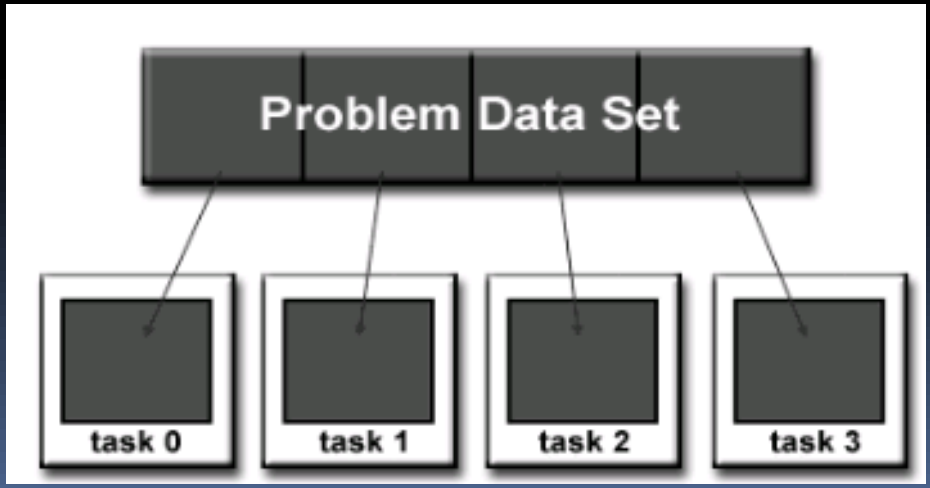
- 4 giai đoạn cần thiết trong việc thiết kế một chương trình song song:
  - Phân chia (Partitioning)
  - Giao tiếp (Communication)
  - Tích tụ (Agglomeration)
  - Ánh xạ (Mapping)

5.3.Phân chia (Partitioning)

- Phân chia bài toán thành các phần để phân phối đến nhiều task
- Đó là sự phân rã (decomposition) hoặc phân chia (partitioning)
- Khi phân rã việc tính toán được thực hiện trên những phần nhỏ hơn, và các hành động cũng được đưa về những tiến trình nhỏ hơn
- 2 cách phân chia:
  - Phân rã theo miền dữ liệu(Domain decomposition)
  - Phân rã theo chức năng (Functional decomposition)

5.3.Phân chia (Partitioning)

- Phân rã theo miền dữ liệu:
  - Dữ liệu liên kết với bài toán được chia thành những phần độc lập
  - Mỗi tác vụ song song hoạt động trên một phần của dữ liệu được phân hoạch

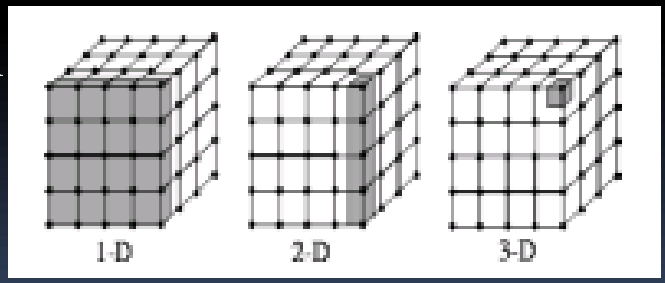


5.3.Phân chia (Partitioning)

- Phân rã theo miền dữ liệu:
  - Việc phân rã theo miền được căn cứ vào:
    - Dữ liệu đầu vào của chương trình
    - Kết quả đầu ra được tính toán
    - Những dữ liệu lưu giữ giá trị trung gian quá trình thực hiện
  - Bao gồm 2 bước:
    - Dữ liệu trong bước tính toán phân ra thành từng phần
    - Phần dữ liệu này được chuyển cho các task để thực hiện

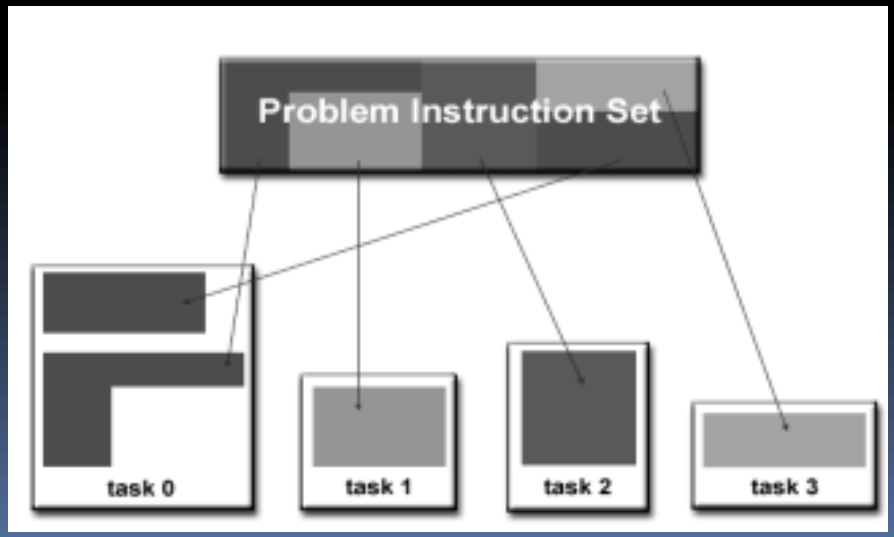
5.3.Phân chia (Partitioning)

- Phân rã theo miền dữ liệu:
  - Ví dụ:
    - Tính tổng hai véc tơ
    - Chia miền trong lưới tính toán
      - Phân rã thành lưới 3 chiều
      - Miền được phân chia:
        - 1-D: mỗi task bao gồm 5x5 điểm
        - 2-D: mỗi task bao gồm 5 điểm
        - 3-D: mỗi task chỉ có 1 nút



5.3.Phân chia (Partitioning)

- Phân rã theo chức năng:
  - Tập trung vào việc tính toán có thể thực hiện được hay không
  - Phân chia theo công việc cần thực hiện, mỗi tác vụ thực hiện công việc riêng.



5.3.Phân chia (Partitioning)

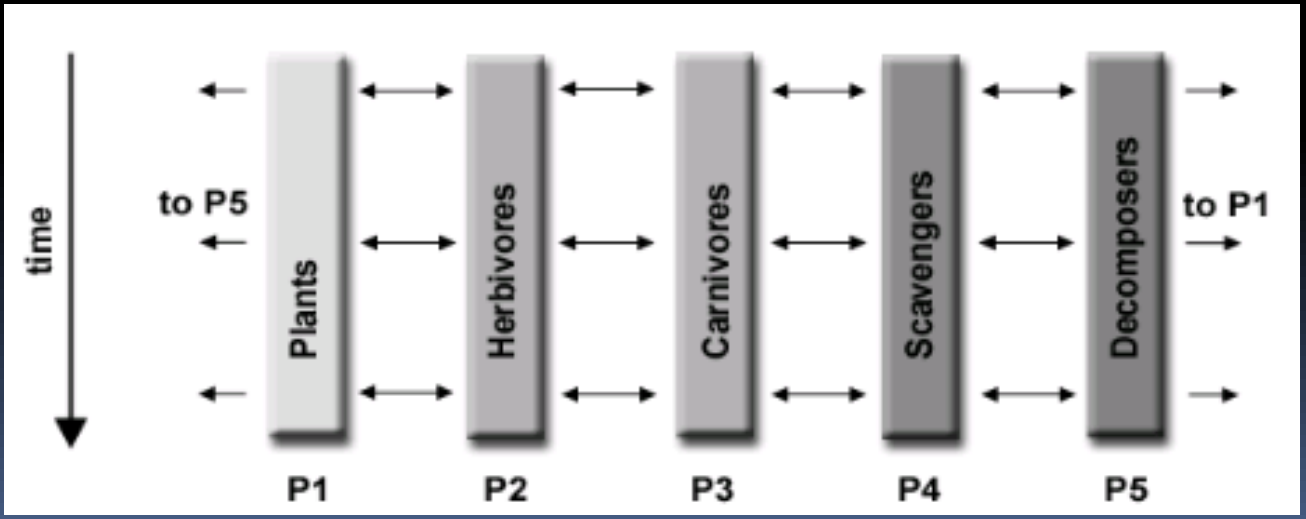
- Phân chia theo chức năng :
  - Ví dụ:
    - Dây chuyền sản xuất áo sơ mi: may thân áo, cổ áo, thừa khuy… phân chia dây chuyền này theo công đoạn.
    - S = ½ *a*b*sin(a,b) chia làm các chức năng nhập dữ liệu a.b>0; nhập góc <a.b>….

5.3.Phân chia (Partitioning)

- Phân chia theo chức năng:
  - Xét mô hình hệ sinh thái
    - Mỗi chương trình tính toán quần thể của một nhóm nào đó, mà ở đó sự tăng trưởng của mỗi nhóm phụ thuộc nhóm lân cận.
    - Mỗi quá trình sẽ tính toán trạng thái hiện thời của nó và trao đổi thông tin với nhóm lân cận
    - Tất cả các task tiến triển tính toán trạng thái bước tiếp theo

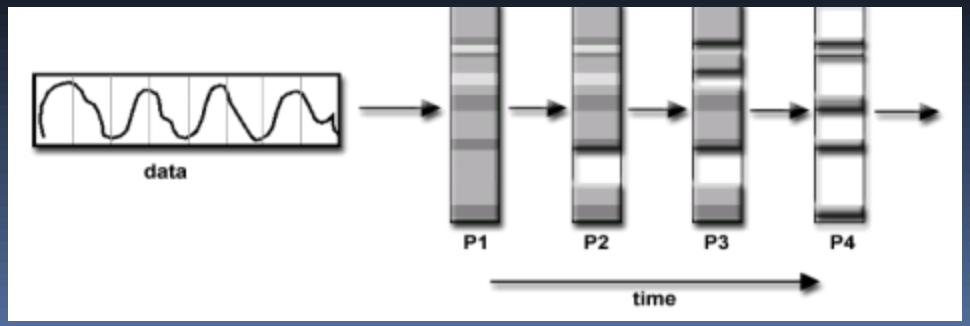
5.3.Phân chia (Partitioning)

- Phân chia theo chức năng:
  - Xét mô hình hệ sinh thái:
    - Xét các nhóm: Thực vật, Động vật ăn cỏ, động vật ăn thịt, động vật ăn xác chết, sinh vật phân hủy



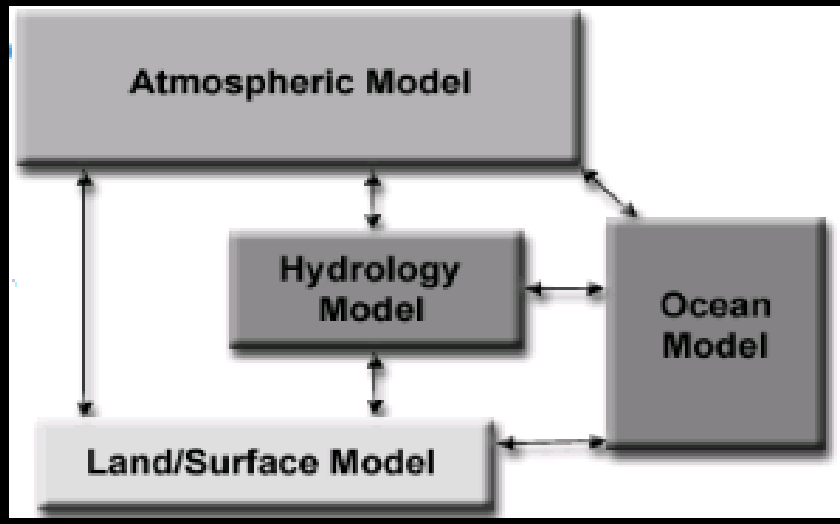
5.3.Phân chia (Partitioning)

- Phân chia theo chức năng:
  - Xét mô hình xử lý tín hiệu:
    - Tập hợp dữ liệu âm thanh truyền qua bốn bộ lọc riêng biệt có thể tính toán được.
    - Mỗi bộ lọc là 1 tiến trình riêng biệt.
    - Phân đoạn dữ liệu chuyển qua bộ lọc đầu tiên trước khi đến bộ lọc thứ 2… Theo thời gian 4 bộ lọc đều ở trạng thái busy



5.3.Phân chia (Partitioning)

- Phân chia theo chức năng:
  - Xét mô hình khí hậu (Climate Modeling)
    - Sự liên quan giữa các mô hình con: Mô hình khí quyển; mô hình thủy học; mô hình biển, đại dượng; mô hình đất liền/ bề mặt trái đất.
    - Mỗi thành phần mô hình dùng 1 task riêng biệt, mũi tên đại diện trao đổi dữ liệu giữa các thành phần trong quá trình tính toán.



5.3.Phân chia (Partitioning)

- Lưu ý:
  - Trong quá trình phân chia còn sử dụng kết hợp cả 2 phương pháp:
    - Sau khi chia tính toán (phân rã theo chức năng) thành những tiến trình rời, có thể tiếp tục tìm ra dữ liệu cần cho những tiến trình này
    - Những dữ liệu này có thể được tách rời ra theo dữ liệu theo miền

5.4.Truyền thông

- Là sự liên lạc qua lại (sự phối hợp giữa các task để có được sự hoạt động phù hợp)
- Những task sinh ra bởi sự phân rã được dùng để thi hành đồng thời.
- Tuy nhiên trong trường hợp tổng quát những tiến trình này không thể thi hành đồng thời: đòi hỏi dữ liệu liên kết với tiến trình khác -> dữ liệu được truyền giữa tiến trình các tính toán mới tiếp tục -> cần thiết về truyền thông (giao tiếp)

5.4. Truyền thông

- Trong việc phân rã theo miền, những đòi hỏi về sự giao tiếp khó có thể xác định.
- Khi đó việc truyền dữ liệu cần phải quản lý chặt chẽ để đảm bảo sự hoạt động của tiến trình
- Thường tương thích với dòng dữ liệu truyền giữa các tiến trình.

5.4.Truyền thông

- Yếu tố cần quan tâm về truyền thông
  - Chi phí truyền thông:
    - Truyền thông thường đòi hỏi sự đồng bộ giữa các task -> tốn chi phí về thời gian chờ.
    - Cạnh tranh giữa các thiết ảnh hưởng đến thông lượng của băng thông mạng, hiệu năng tính toán.
  - Trễ so với băng thông:
    - Độ trễ là thời gian tối thiểu cần để gửi một thông báo từ điểm A đến điểm B
    - Băng thông là lượng dữ liệu có thể truyền trên 1 đơn vị thời gian
    - Gửi nhiều thông báo -> gây ra độ trễ làm tăng chi phí truyền thông

5.4.Truyền thông

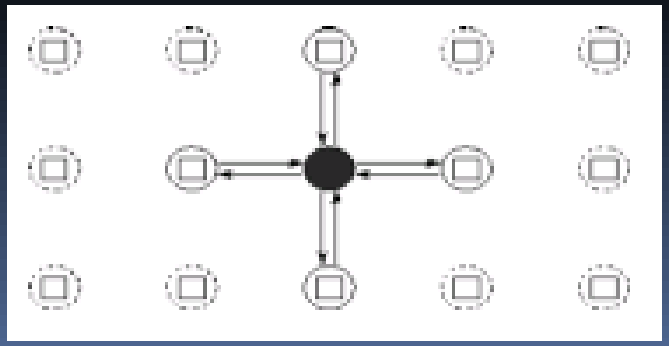
- Yếu tố cần quan tâm về truyền thông:
  - Truyền thông đồng bộ & không đồng bộ:
    - Yêu cầu thiết lập quan hệ “handshaking” giữa các task chia sẻ dữ liệu
    - Truyền thông đồng bộ bao gồm truyền và nhận trọn vẹn cả khối thông tin -> các task khác phải đợi đến khi truyền thông hoàn thành.
    - Truyền thông không đồng bộ cho phép các task chuyển dữ liệu một cách độc lập.
    - Việc xen kẽ tính toán tốt nhất với truyền thông không đồng bộ.

5.4.Truyền thông

- Yếu tố cần quan tâm về truyền thông:
  - Phạm vi truyền thông:
    - Point-to-point: liên quan đến 2 task trong đó 1 task gửi và 1 task nhận.
    - Truyền thông tập thể: chia sẻ giữa nhiều hơn 2 task

5.4.Truyền thông

- Ví dụ về truyền thông:
  - Bài toán lan truyền nhiệt:
    - Nhiệt độ của mỗi điểm được tính toán dựa trên nhiệt độ của điểm bên canh.
    - Nếu bên cạnh do một task khác tính toán thì phải truyền dữ liệu nhiệt độ này cho task đó.
    - Rời rạc hóa tính toán trên máy đưa vào lưới sai phân



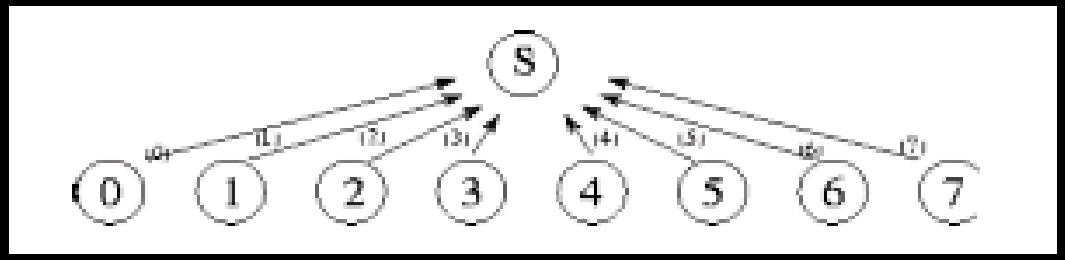
5.4.Truyền thông

- Ví dụ về truyền thông:
  - Bài toán truyền nhiệt:
    - Thuật giải có thể viết như sau:

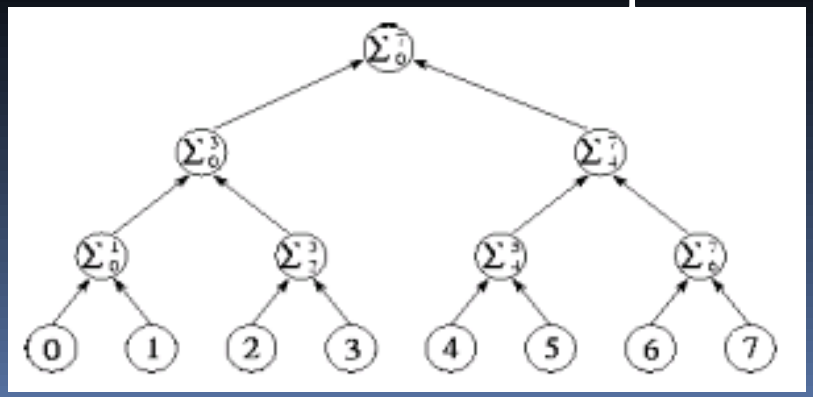
```
for t = 0 to T-1 do
    send Xij to each neighbor
    receive Xi-1j, Xi+1j, Xij-1, Xij+1 from neighbors
    compute Xij
endFor
```

5.4.Truyền thông

- Ví dụ về truyền thông:
  - Bài toán tính tổng, giả sử Task S phân chia như sau



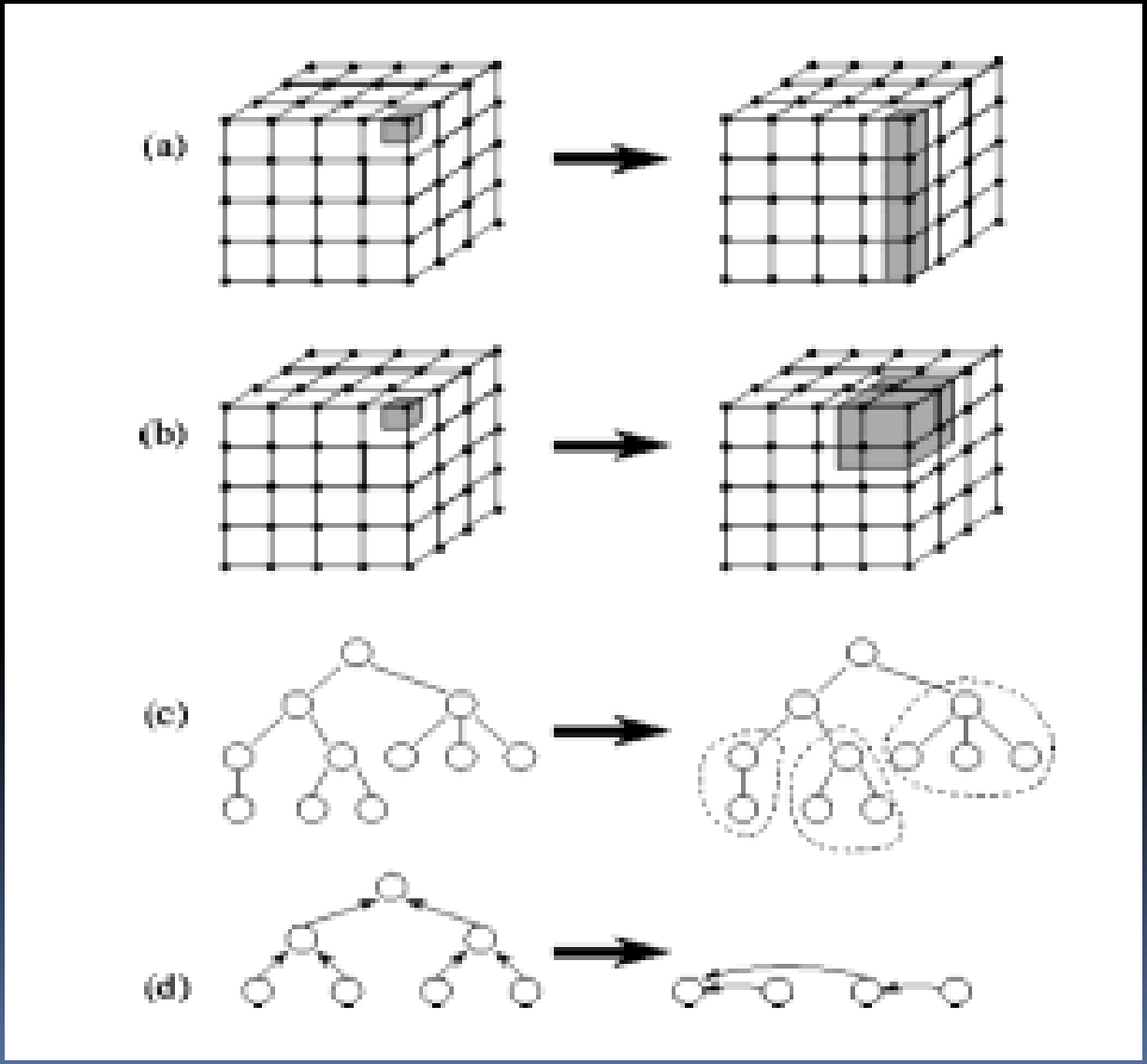
  - Như vậy Task S phải giao tiếp với các task khác từ 0-7 còn các task khác thì không
  - Phân chia thành các bài toán xấp xỉ



5.5.Agglomeration (Tích tụ)

- Giai đoạn gom các task lại để tăng tính địa phương và giảm chi phí giao tiếp.
- Ví dụ thay vì tách thành các miền con 3-D; chỉ tách thành các miền con 2-D, hoặc 1-D:
  - Do đó mỗi tiến trình có thể bao gồm nhiều nút hơn
- Hoặc nhóm các cây con lại với nhau

5.5.Agglomeration

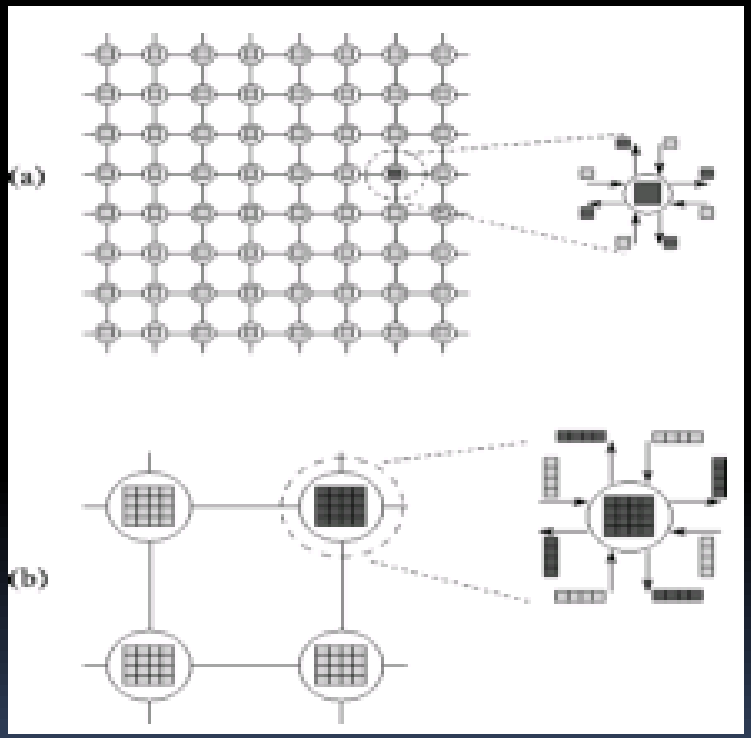


5.5.Agglomeration

- Ví dụ bài toán lan truyền nhiệt
  - Một task bao gồm 1 điểm sai phân, có thể tích tụ lại để có một task gồm 16 điểm sai phân
  - Khi đó dữ liệu được truyền giảm từ 256 xuống còn 64
  - Như vậy số lượng giao tiếp thực hiện trên một đơn vị tính toán sẽ giảm đi

5.5.Agglomeration

- Hình a: Sự tính toán trên lưới 8*8 được phân thành 8*8 = 64 task,mỗi task chịu trách nhiệm 1 điểm.
- Hình b: Với cùng tính toán được phân thành 2*2=4 task mỗi task chịu trách nhiệm 16 điểm
- Mỗi task giao tiếp với 4 tasks xung quanh.
- Số dữ liệu được truyền:
  - 64 * 4 * 1 = 256 dữ liệu
  - 4 * 4 * 4 = 64 dữ liệu



5.6.Mapping

- Tiến trình được gọi là task hay tác vụ
- Số lượng tiến trình của các bài toán khác nhau hoàn toàn không giống nhau
- Mục đích mapping (ánh xạ) làm cho tổng thời gian thi hành đạt cực tiểu

5.6.Mapping

- Mục tiêu:
  - Các tiến trình có thể thực thi đồng thời đặt trên những bộ xử lý khác nhau để tăng khả năng song song
  - Các tiến trình thường ttao đổi với nhau đặt trên cùng một bộ xử lý để tăng tính địa phương, giảm chi phí
- Có 2 kiểu mapping:
  - Static mapping
  - Dynamic mapping

5.6.Mapping

- Static Mapping:
  - Phân phối các tiến trình đến các bộ xử lý trước khi thực thi thuật toán
  - Đối với các tiến trình được tạo ra tĩnh thì ánh xạ động hay tĩnh đều có thể sử dụng được

5.6.Mapping

- Dynamic Mapping:
  - Khi static mapping có thể gây ra việc phân phối công việc không cân bằng giữa các bộ xử lý
  - Kỹ thuật ánh xạ động phân phối các công việc cho các bộ xử lý trong suốt quá trình thuật toán thực thi
  - Tiến trình động -> ánh xạ động
  - Kích cỡ tiến trình không biết trước, sử dụng static mapping gây ra việc không cân bằng tải
  - Nếu dữ liệu trong tiến trình lớn, khi đó ánh xạ động sẽ làm tăng sự di chuyển của dữ liệu giữa các tiến trình -> sử dụng static mapping trong trường hợp này hiệu quả hơn

5.7.Sự phụ thuộc dữ liệu

- Sự phụ thuộc dữ liệu xảy ra khi:
  - Thứ tự thực hiện câu lệnh ảnh hưởng đến kết quả của chương trình
  - Một vùng chứa dữ liệu trong bộ nhớ được dùng nhiều lần bởi nhiều task khác nhau.
- Xét đến sự phụ thuộc dữ liệu là cần thiết bởi vì nó là yếu tố ngăn cản khả năng song song của thuật toán

5.7.Sự phụ thuộc dữ liệu

- Ví dụ:
  - Xét một vòng lặp về sự phụ thuộc dữ liệu

```
for (j = mystart; j < myend; j++)
    A[j] = A[j-1] * 2.0;
```

  - A[j] thể hiện sự phụ thuộc dữ liệu vào A[j-1] -> không có khả năng song song

5.7.Sự phụ thuộc dữ liệu

- Nếu task 2 có A[j] và task 1 có A[j-1], việc tính toán giá trị đúng của A[j] cần:
  - Kiến trúc bộ nhớ phân tán: task 2 phải nhận giá trị A[j-1] từ task 1 sau khi task 1 hoàn thành công việc tính toán của nó
  - Kiến trúc bộ nhớ chia sẻ: task 2 phải đọc lại A[j-1] sau khi task 1 cập nhật giá trị của A[j-1]

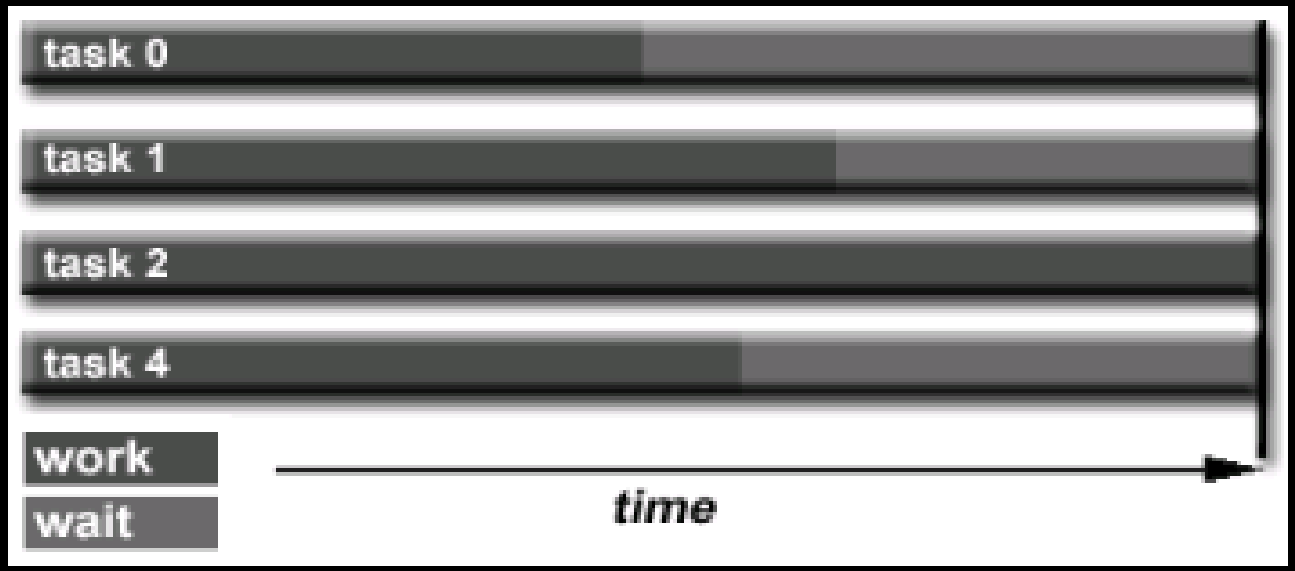
5.7. Sự phụ thuộc dữ liệu

- Khắc phục sự phụ thuộc dữ liệu:
  - Distributed Memory: Giao tiếp dữ liệu cần thiết tại các điểm đồng bộ hóa
  - Shared Memory: Đồng bộ hóa thao tác read / write giữa các tiến trình

5.8.Load Balancing

- Phân bố đều công việc cho các task sao cho các task busy giống nhau và finish giống nhau.
- Giảm thiểu thời gian nhàn rỗi
- Là 1 yếu tố cực kỳ quan trọng cho các chương trình song song
- Ví dụ:
  - Nếu tất cả công việc đồng bộ hóa tại một ngưỡng thì công việc chậm nhất sẽ ảnh hưởng đến tổng thể

5.8.Load Balacing



- 2 phương pháp:
  - Phương pháp chia đều công việc cho các tác vụ
  - Phương pháp gán động công việc

5.8.Load balancing

- Phương pháp chia đều công việc cho các tác vụ:
  - Phân chia đều công việc tuy nhiên ngưỡng của mỗi tác vụ thường không giống nhau.
  - Giảm tải thời gian chờ:
    - Đối với thao tác mảng: mỗi tác vụ thực hiện công việc đều nhau, phân tán đều dữ liệu cho các tác vụ
    - Đối với vòng lặp: công việc cho mỗi vòng lặp như nhau, phân tán đều các lần lặp cho các tác vụ
    - Nếu có sự tham gia các máy không đều nhau thì phải sử dụng công cụ cân bằng tải để phát hiện điều chỉnh công việc

5.8.Load balancing

- Phương pháp gán động công việc:
  - Một vài bài toán mất cân bằng tải ngay cả khi có sự phân bố đều dữ liệu giữa các task:
    - Ma trận thưa: Một vài task có dữ liệu thực sự để làm việc trong khi một số task khác chủ yếu là số 0
    - Khi số lượng công việc của mỗi task là một đại lượng biến thiên, nên không thể dự đoán được -> phải tiếp cận theo hướng lập lịch.-> mỗi task sau khi hoàn thành công việc sẽ đợi để nhận công việc mới.

5.8.Vai trò của I/O

- Những hạn chế của I/O:
  - Vận hành của I/O làm hạn chế việc xử lý song song.
  - Các hệ thống I/O song song chưa đủ tốt hoặc không sẵn sàng cho tất cả nền tảng song song cơ bản.
  - Task dùng chung không gian tập tin -> thông tin bị ghi đè.
  - Máy chủ xử lý yêu cầu đọc nhiều lần
  - I/O tiến hành qua mạng -> gây tắc nghẽn mạng, máy chủ file bị hỏng

5.8.Vai trò của I/O

- Thuận lợi của I/O:
  - GPFS: General Parallel File System for AIX (IBM)
  - Lustre: for Linux clusters (SUN Microsystems)
  - PVFS/PVFS2: Parallel Virtual File System for Linux clusters
  - PanFS: Panasas Active Scale File System for Linux clusters
  - Kỹ thuật lập trình giao diện song song I/O cho máy MPI đã có từ năm 1996

HẾT BÀI