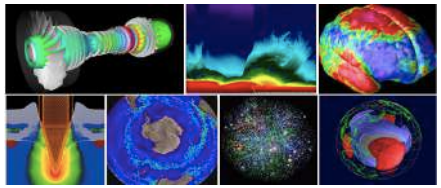
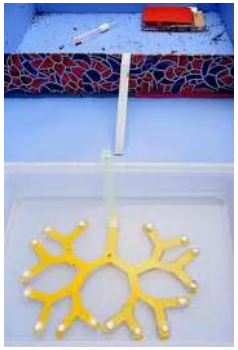
Tính toán song song và phân tán

PGS.TS. Trần Văn Lăng
langtv@vast.vn
lang@lhu.edu.vn



7. Mẫu phân chia (Paradigm)

1. Mô hình cây nhị phân (Binary Tree Paradigm)
2. Chia để trị (Devide and Conquer Paradigm)
3. Phân hoạch (Partitioning Paradigm)



7.1. Binary Tree Paradigm

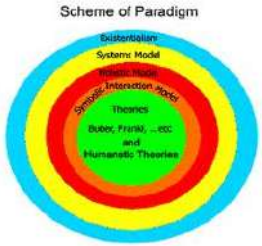
- Xét cây nhị phân đầy đủ với n lá có độ cao là $\log_2 n$ (hoặc ký hiệu $\log n$)
- Dữ liệu đặt ở n nút lá.
- Quá trình đi từ ngọn đến gốc mất $\log n$ thời gian.

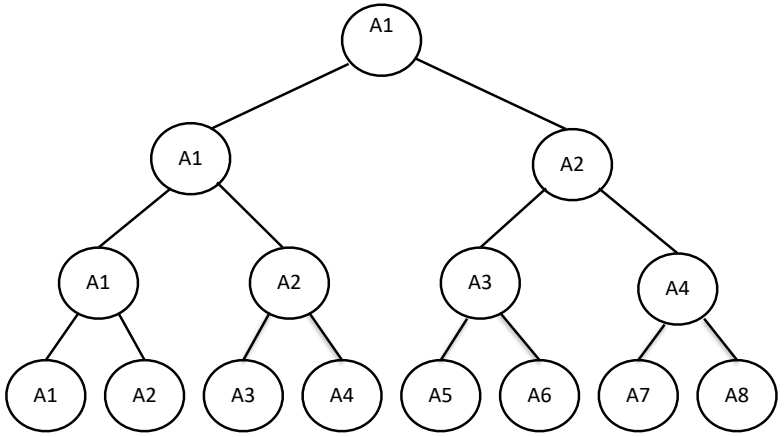
Khảo sát việc tính tổng

- Thuật giải tuần tự mất $O(n)$
- Xét với $n = 2^k$ để có được cây nhị phân đầy đủ
- Tứ đây chia dữ liệu thành 2 nhóm
- Số tiến trình cần thiết là $n/2$

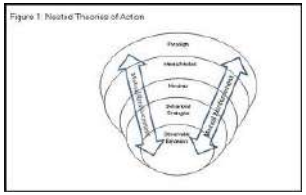
Minh họa

- Với $n = 8 = 2^3$, mỗi nhóm có 4 phần tử
  - Nhóm 1: A_1, A_2, A_3, A_4
  - Nhóm 2: A_5, A_6, A_7, A_8
- Cần 4 task
- Với dãy gồm 8 phần tử, mô hình như hình vẽ bên dưới

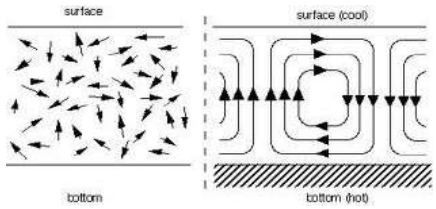




- Bốn tiến trình đồng thời tính các giá trị tổng của nó theo yêu cầu
- Rồi lưu vào các biến tương ứng
  - $A_1 \leftarrow A_1 + A_2$
  - $A_2 \leftarrow A_3 + A_4$
  - $A_3 \leftarrow A_5 + A_6$
  - $A_4 \leftarrow A_7 + A_8$



- Giai đoạn tiếp theo chỉ còn lại 4 phần tử. Các phần tử lưu trong 2 nhóm
  - Nhóm 1: A_1, A_2
  - Nhóm 2: A_3, A_4



- Khi đó 2 tiến trình đồng thời cộng
  - $A_1 \leftarrow A_1 + A_2$
  - $A_2 \leftarrow A_3 + A_4$
- Các kết quả được lưu vào A_1, A_2

Thuật giải

Algorithm: Tính tổng nhị phân
Input: Mảng A[1..n]
Output: A[1]
1. p = n/2
2. **While** p > 0 **do**
3. **For** i = 1 **to** p **doPar**
4. A(i) = A(2i-1) + A(2i)
5. **endFor**
6. p = p/2
7. **EndWhile**

Độ phức tạp

- Số process ban đầu là $p = n/2$
- Trong mỗi lần thực hiện số tiến trình chỉ còn $1/2$.
- Nên số tiến trình cần thiết là

$$P = n/2 = O(n)$$

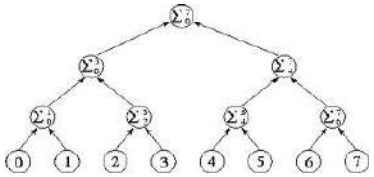
- Trong câu lệnh 4, chi phí thời gian là $O(1)$ cho 1 tiến trình, nên với $\log_2 n$ bước, chi phí thời gian $O(\log n)$.

Algorithm: Tính tổng nhị phân
Input: Mảng A[1..n]
Output: A[1]
1. p = n/2
2. **While** p > 0 **do**
3. **For** i = 1 **to** p **doPar**
4. A(i) = A(2i-1) + A(2i)
5. **endFor**
6. p = p/2
7. **EndWhile**

7.2 Chia để trị

- Chia bài toán thành các bài toán con để dễ tìm ra lời giải cho tứng bài toán con riêng lẽ.
- Hợp nhất các lời giải này lại để được lời giải của bài toán.

- Theo cách thực hiện của mô hình cây nhị phân
- Thuật giải song song chưa tối ưu, bởi:
  - Thuật giải tuần tự cần $O(n)$
  - Song song cần $O(\log n)$ thời gian với $O(n)$ task



- Nên hiệu suất $E_p(n)$(Efficiency) quá nhỏ khi n khá lớn: $\frac{O(n)}{O(n)O(\log n)} = \frac{1}{\log n} < 1$
- Chúng ta phải khảo sát bài toán sao cho $E_p(n)$ có giá trị là 1

- Với thuật giải tính tổng như trên, $1 = E_p(n) = \frac{O(n)}{p \times O(\log n)}$
- Suy ra $P = \frac{n}{\log n}$

Minh họa

- Chia mảng $A(1{:}n)$ thành $r = n/\log n$ nhóm, để huy động r tiến trình
- Mỗi nhóm có $\log n$ phần tử.
- Ở đây vẫn giả thiết $n = 2^k$ và $n/\log n$ là số nguyên ($k = \log n$)
- Các nhóm được phân bố như sau:

- $A_1, A_2, A_3, \ldots, A_k$
- $A_{k+1}, A_{k+2}, A_{k+3}, \ldots, , A_{2k}$

  .....
- $A_{(i-1)k+1}, A_{(i-1)k+2}, \ldots, A_{ik}$

  .....
- $A_{(r-1)k+1}, A_{(r-1)k+2}, \ldots, A_{rk}$

- Mỗi tiến trình cộng $\log n$ phần tử.
- Lưu kết quả lại trong mảng $B(1{:}r)$
- Sử dụng thuật giải song song (như phần 1) để tính tổng r phần tử của B.
- Thuật giải song song trên B này cần $O(\log n)$thời gian và $O(r)$ tiến trình.

b) Thuật giải

Algorithm: Tính tổng chia và chế ngự
Input: A(1..n)
Output: B(1)

```
1. For i=1 to n/logn doPar
2.       B(i) = 0
3.       For j=1 to logn do
4.                B(i) = B(i)+ A(ik+j-logn)
5.       EndFor
6. EndPar
```

Algorithm: Nhị phân tính tổng B(1..r)

Input: B(1..r)
Output: B(1)
1. p = r/2
2. While p > 0 do
3. **For** i=1 **to** p **doPar**
4. B(i) = B(2i-1) + B(2i)
5. **EndPar**
6. p = p/2
7. **EndWhile**

Độ phức tạp

- Các câu lệnh 2, 3, 4 cần $O(\log n)$ bước thời gian.
- Khi đó, các bước tứ 1 đến 5 dùng song song mất $O(\log n)$ thời gian $O(n/\log n)$ tiến trình.

- Các bước còn lại dùng thuật giải song song dạng cây nhị phân, cần thời gian với chi phí:
  $O\left(log(\frac{n}{\log n})\right) = O(\log n - \log\log n) = O(\log n)$
- Dùng $O\left(\frac{n}{\log n}\right)$ tiến trình.
- Như vậy, thuật giải cần $O(\log n)$ thời gian với $O\left(\frac{n}{\log n}\right)$ tiến trình.


```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <omp.h>

int main(){
    float *A, *B, *S;
    long int N, i, j;
    int R, K, p;

    printf("No. of elements:");
    scanf( "%ld", &N );
    K = log2(N);
    R = N/K;

    A = (float *)calloc( N,
sizeof(float) );
    for( i = 0; i < N; i++ )
        A[i] = i+1;
    B = (float *)calloc( R,
sizeof(float) );
    double wt = omp_get_wtime();
#pragma omp parallel for
    for( i = 0; i < R; i++ ){
        B[i] = 0.0;
        for(int j = 0; j < K;j++)
            B[i]+= A[(i+1)*K+j-K];
    }
    p = R/2;
    while( p > 0 ){
#pragma omp for
        for( i = 0; i < p; i++ )
            B[i]= B[2*i]+B[2*i+1];
        p /= 2;
    }
    printf( "Elapsed Parallel
Time: %lf sec\n",
            omp_get_wtime()-wt );
    printf( "Sum of Sequence:
%lf\n", B[0] );
    return 1;
}
```

6.3 Mô hình phân hoạch

- Theo cách tiếp cận chia để trị, thuật giải phải bao gồm 2 bước, trong đó có bước quan trọng là việc hợp nhất các bài toán con đã chia ra để được lời giải của bài toán xuất phát.
- Đôi khi việc thực hiện này làm ảnh hưởng đến kết quả của ban đầu.

- Trong mô hình phân hoạch, người ta không quan tâm đến quá trình hợp nhất.
- Tiến trình phân hoạch bảo đảm sao cho khi phân chia xong, việc giải các bài toán con cho kết quả là lời giải của bài toán đặt ra.

- Giả sử có 2 mảng A(1:n), B(1:n) đã được sắp xếp tăng, cần phải hợp nhất thành 1 mảng C(1:2n) cũng tăng.
- Ở đây giả thiết $n = 2^k$, và $r = n/\log n$ là số nguyên ($k = \log n$)

Thuật giải tuần tự

Algorithm: Merge //Thuật giải trộn tuần tự
Input: A(1..n), B(1..n)
Output: C(1..2n)

```
1. i = 1; j = 1; k = 1
2. While  k <= 2n do
3.       If A(i) < B(j) then
4.             C(k) = A(i)
5.             i = i + 1
6.       Else
7.             C(k) = B(j)
8.             j = j + 1
9.       EndIf
10.      k = k + 1
11. EndWhile
```

Minh họa thuật giải

- Phân hoạch A thành $r = n/\log n$ nhóm gồm k phần tử như sau:
  - $A_1, A_2, A_3, \ldots, A_k$
  - $A_{k+1}, A_{k+2}, A_{k+3}, \ldots, , A_{2k}$
  - .....
  - $A_{(i-1)k+1}, A_{(i-1)k+2}, \ldots, A_{ik}$
  - .....
  - $A_{(r-1)k+1}, A_{(r-1)k+2}, \ldots, A_{rk}$

- Tiếp theo cần tìm r số nguyên $j_1, j_2, \ldots, j_r$ sao cho
  - j_1 là chỉ số lớn nhất mà $A_k > B_{j_1}$
  - j_2 là chỉ số lớn nhất mà $A_{2k} > B_{j_2}$
  - .....
  - j_i là chỉ số lớn nhất mà $A_{ik} > B_{j_i}$
  - .....
  - j_r là chỉ số lớn nhất mà $A_{rk} > B_{j_r}$

- Từ đây, phân B thành r nhóm như sau:
  - $B_1, B_2, \ldots, B_{j_1}$
  - $B_{j_1+1}, B_{j_1+2}, \ldots, B_{j_2}$
  - .....
  - $B_{j_{i-1}+1}, B_{j_{i-1}+2}, \ldots, B_{j_i}$
  - .....
  - $B_{j_{r-1}+1}, B_{j_{r-1}+2}, \ldots, B_{j_r}$

- Các phần tử trong nhóm 1 của A đều nhỏ hơn hay bằng các phần tử trong nhóm 2, 3, ... của B.
- Có thể đồng thời hợp nhất các phần tử trong hai nhóm thứ i của A và B.

Algorithm: Thuật giải trộn

Input: A(1..n), B(1..n)
Output: C(1..2n)

1. **For** i = 1 **to** r **doPar**
2. j(i) = **max**{t/B(t)<A(ik)}
3. Merge(A((i-1)k+1..ik), B(j(i-1)+1..j(i))
4. **EndPar**

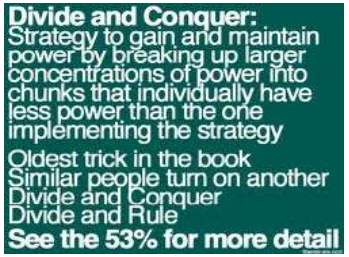
- Ở bước thứ 2, dùng thuật giải tìm kiếm nhị phân, chi phí $O(\log n)$.
- Bước 3, tùy theo kích thước của $B_{j_{i-1}+1:j_i}$
- Nếu lớn hơn k, chúng ta dùng thuật giải song song này để hợp nhất hai phần tương ứng của A và B.

- Còn khi kích thước này nhỏ hơn hay bằng k, thuật giải cần chi phí $O(\log n)$.
- Như vậy, thuật giản cần $O(\log n)$ thời gian, $O(n/\log n)$ tiến trình.

Minh họa

- A = {1,5,15,18,19,21,23,24,27,29,30, 31,32,37,42,49}
- B = {2,3,4,13,15,19,20,22,28,29,38, 41,42,43,48,49}
- n = 16 = 2^4, k =4, r = n/logn = 4

- A được phân thành 4 nhóm
  - 1,5,15,18
  - 19,21,23,24
  - 27,29,30,31
  - 32,37,42,49



- Khi đó, j = {5,8,10}
- B được phân thành 4 nhóm
  - 2,3,4,13,15
  - 19,20,22
  - 28,29
  - 38,41,42,43,48,49

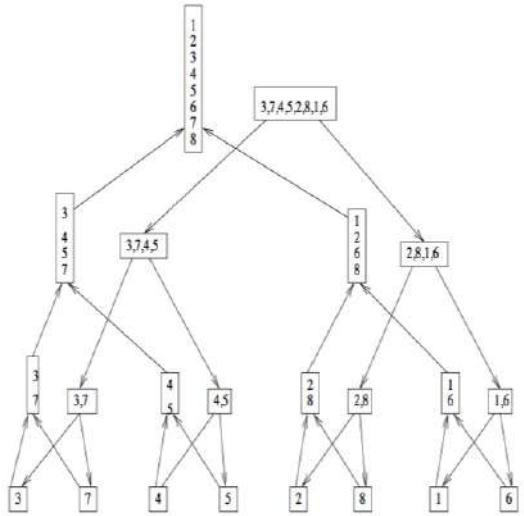
- Đồng thời hợp nhất các nhóm con của A và B.

Nhóm	1	2	3	4
A	1, 5, 15, 18	19, 21, 23, 24	27,29, 30,31	32,37, 42,49
B	2, 3, 4, 13, 15	19,20, 22	28,29	38,41,42, 43,48,49

- C(1:9) = {1,2,3,4,5,13,15,15,18}
- C(10:16) = {19,19,20,21,22,23,24}
- C(17:22) = {27,28,29,29,30,31}
- C(23:32) = {32,37,38,41,42,42,43,48,49,49}

Ứng dụng: thuật giải Merge-Sort

- Minh họa, với dãy 8 số nguyên, thuật giải trộn và sắp xếp như sau:



- Các tiến trình đồng thời sắp xếp tuần tự phần dữ liệu liên quan.
- Sau khi hoàn tất, trộn (merge) hai dãy đã sắp xếp thành một dãy sắp xếp
- Các tiến trình tiếp tục cho đến hết dữ liệu

Ví dụ

4. Phân tích thuật toán song song tính số π từ tích phân như sau: $\pi = \int_0^1 \frac{4}{1+x^2} dx$ bằng cách xấp xỉ $\pi \approx \sum_{i=0}^{N} \frac{4}{1+x_i^2} \Delta x,$

   Trong đó $\Delta x = \frac{1}{N}, x_i = i\Delta x$


```
#include <stdio.h>
#define N  10000000
int main(int argc, const char * argv[]){
   double delta, x, sum = 0.0, pi;
   delta = 1.0/N;
   for ( int i = 0; i < N; i++ ){
      x = i*delta;
      sum += 4.0/(1.0 + x*x);
   }
   pi = sum*delta;
   printf( "Pi = %20.18f\n", pi );
    return 0;
}
```

5. Phân tích thuật toán tính số π dùng Phương pháp Monte Carlo:
   - Đường tròn nội tiếp trong một hình vuông
   - Nếu đặt số chấm ngẫu nhiên lên hình vuông
   - Thì tỷ lệ giữa số chấm nằm trong hình tròn và số chấm nằm trong hình vuông là: $\frac{r^2\pi}{2r \times 2r} = \frac{\pi}{4}$


```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define npoints 10000000
int  main(int argc, const char * argv[]){
    int count = 0;
    double x, y, pi;
    srand( (unsigned int)time(NULL) );
    for ( int i = 1; i < npoints; i++ ){
        x = (double)rand()/RAND_MAX;
        y = (double)rand()/RAND_MAX;
        if ( x*x + y*y <= 1 )
            count += 1;
    }
    pi = 4.0*count/npoints;
    printf( "Pi = %20.18f\n", pi );
    return 0;
}
```