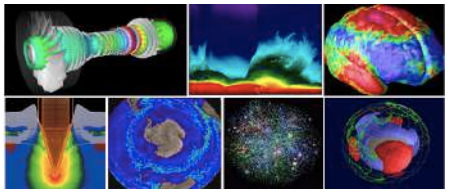
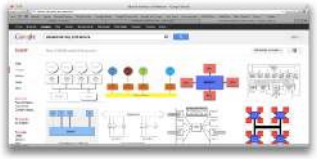
Tính toán song song và phân tán

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



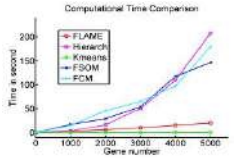
6. Đánh giá thuật giải song song

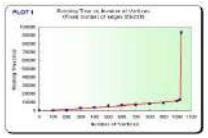
1. Độ phức tạp thời gian
2. Speedup
3. Efficiency
4. Định luật Amdahl
5. Chi phí thời gian



6.1 Độ phức tạp thời gian

- Để đánh gía thuật giải song song thường căn cứ vào:
  - Thời gian tính toán (hay Time Complexity)
  - Yêu cầu số bộ xử lý (hay Processor Complexity)





- $T(n) = O(f(n)) \iff \forall C > 0, \exists n_0 / T(n) < Cf(n), \forall n > n_0$
- $T(n) = \Omega(f(n)) \iff \forall C > 0, \exists n_0 / T(n) > Cf(n), \forall n > n_0$
- $T(n) = \theta(f(n)) \iff \forall C_1, C_2 > 0, \exists n_1, n_2 /$
  $T(n) < C_1 f(n), \forall n > n_1$ và $T(n) > C_2 f(n), \forall n > n_2$

- Ta cũng có
  $T(n) = \theta(f(n)) \iff T(n) = O(f(n))\ \&\ T(n) = \Omega(f(n))$
- Và $T(n) = O(f(n)) \iff f(n) = \Omega(T(n))$
- $T(n) = o(f(n))$ nếu $\forall C > 0, \exists n_0 \Big/ T(n) < Cf(n), \forall n > n_0$
- $T(n) = \omega(f(n))$ nếu $\forall C > 0, \exists n_0 \Big/ T(n) > Cf(n), \forall n > n_0$

Khi đó,

- $T(n) = o(f(n)) \iff f(n) = \omega(T(n))$
- $T(n) = o(f(n)) \Longrightarrow T(n) = O(f(n))$
- $T(n) = \omega(f(n)) \Longrightarrow T(n) = \Omega(f(n))$

- Một thuật giải bao gồm 2 phần, phần thứ nhất có độ phức tạp là $O(f_1(n))$, phần thứ hai là $O(f_2(n))$.
- Khi đó, thuật giải sẽ có độ phức tạp là $O\big(\max(f_1(n), f_2(n))\big)$

- Một thuật giải là sự lồng nhau của 2 phần, phần thứ nhất có độ phức tạp là $O(f_1(n))$, phần thứ hai là $O(f_2(n))$.
- Khi đó, thuật giải sẽ có độ phức tạp là $O\big(f_1(n)f_2(n)\big)$

- Xét $L = \lim_{n\to\infty} \frac{T(n)}{f(n)}$
- Nếu L = 0, điều đó có nghĩa $f(n)$tăng nhanh hơn $T(n)$, nên $T(n) = o(g(n))$
- Nếu $L = \infty$, thì $T(n) = \omega(f(n))$
- Lấu L khác không và hữu hạn, i.e. $T(n)$ và $f(n)$ tăng cùng cấp độ, nên $T(n) = \theta(f(n))$.

- Xét $\lim_{n\to\infty} \frac{T(n)}{f(n)} = \frac{n^2+n}{2n^2} = \frac{1}{2}$
- Khi đó $T(n) = \frac{n(n+1)}{2}, f(n) = n^2$
- Vậy $T(n) = \theta(n^2)$
- Tương tự, $P(n)$ có cấp m, thì $P(n) = \theta(n^m)$

- Hoặc $T(n) = n^{100}, f(n) = 2^n$, thì $T(n) = o(2^n)$ và $f(n) = \omega(n^{100})$.
- Ta có thể chứng minh điều này bằng lấy giới hạn của tỷ số $T(n)$ và $f(n)$, sau đó dùng khai triển L'Hospital đến số hạng thứ 100, với

$$f(x) = f(0) + \frac{x}{1!}f'(0) + \frac{x^2}{2!}f''(0) + \ldots + \frac{x^{100}}{100!}f^{(100)}(0)$$

$$\frac{d}{dx}e^{f(x)} = e^{f(x)}f'(x)$$

6.2 Speedup

- Thuật giải tuần tự với thời gian là $T_s(n)$.
- Thuật giải song song trên P bộ xử lý có thời gian là $T_p(n)$
- Khi đó, speedup $S_p(n)$ được định nghĩa

$$S_p(n) = \frac{T_s(n)}{T_p(n)}$$

- Speedup của hệ thống song song không phải là một số cố định. Mà phụ thuộc vào kích thước bài toán và số bộ xử lý. Thực chất là hàm của (n, P).
- Người ta thường phân tích để đánh giá định tính của thuật giải song song bằng cách cố định n hoặc cố định P để vẽ các đường cong tương ứng là giá trị của speedup.

- Cố định n, vẽ đồ thị đường cong speedup theo P - một trục là P, trục còn lại là speedup. Qua đó có thể phân tích hiệu năng của thuật giải khi số bộ xử lý tăng lên.
- Cố định P, vẽ đồ thị đường cong speedup theo n. Qua đó có thể nghiên cứu dáng điệu của thuật giải khi kích thước bài toán tăng lên.

6.3 Efficiency

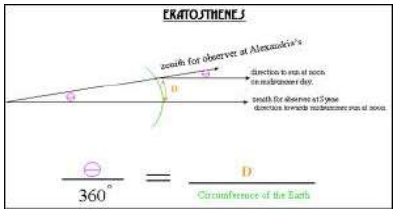
- Efficiency (hiệu suất) có giá trị tối đa là 1.
- Efficiency cung cấp dấu hiệu về sự tận dụng có hiệu quả của các bộ xử lý.

  Chính vì vậy, efficiency $E_p(n)$:

$$E_p(n) = \frac{T_s(n)}{pT_p(n)} = \frac{S_p(n)}{p}$$

Ví dụ: Thuật giải sàn Eratosthenses

- Mục tiêu tìm các số nguyên tố nhỏ hơn hay bằng số nguyên dương N cho trước.
  - Bắt đầu từ số nguyên tố thứ nhất t (số 2).
  - Sàn bỏ các bội của số t này kể từ t^2 cho đến N.



- Lấy số tiếp theo còn trên sàn (là số 3).
- Quay trở lại bước thứ hai cho đến khi $t^2 > N$.
- Các số còn lại trên sàn là số nguyên tố.

- Minh họa, với $N = 1000$, các số nguyên tố nhỏ hơn $\sqrt{N}$ là 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31.
- Thuật giải để sàn một số nguyên tố t nào đó được minh họa như sau:

Thuật giải tuần tự

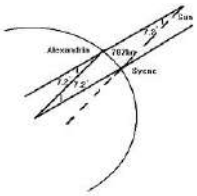
```
i = t²
While i <= N {
   Sàn số thứ i trong dãy;
   i = i + t;
}
```

- Chi phí thời gian thực hiện của thuật giải được tính theo công thức:
- $\left\lceil \frac{N + 1 - t^2}{t} \right\rceil$

Chi phí để sàn từng số nguyên tố

Số nguyên tố	Thời gian
2	499
3	331
5	196
7	137
11	80
13	64
17	42
19	34
23	21
29	6
31	1

- Chi phí tuần tự là: 1411
- Chi phí song song được tính theo số task tham gia.



Trường hợp thứ I

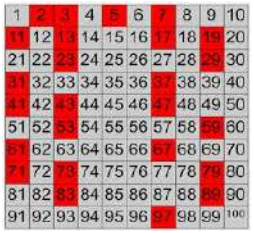
- Trường hợp có 2 task tham gia giải quyết.
  - Task 1: Sàn các bội của 2, 7, 17, 23, 29, 31.
  - Task 2: Sàn các bội của 3, 5, 11, 13, 19.
- Khi đó chi phí thời gian là 706
- $Speedup = \frac{1411}{706} = 1{,}9985$

Trường hợp thứ II

- Trường hợp có 3 task tham gia giải quyết.
  - Task 1: Sàn các bội của 2.
  - Task 2: Sàn các bội của 3, 11, 19, 29, 31.
  - Task 3: Sàn các bội của 5, 7, 13, 17, 23.
- Khi đó chi phí thời gian là 499.
- $Speedup = \frac{1411}{499} = 2{,}8276$

Trường hợp thứ III

- Trường hợp có nhiều hơn 3 task.
  - Task 1: Sàn các bội của 2.
  - Task 2: Sàn các bội của 3, ...
  - Task 3: Sàn các bội của 5, ...
  - Task 4: Sàn các bội của 7, ...
  - ...



- Khi đó chi phí thời gian vẫn là 499.
- $Speedup = \frac{1411}{499} = 2{,}8276$

Kết luận

Case	N. of tasks	Speedup	Efficiency
1	2	1,9985	**0,9993**
2	3	2,8276	0,9425
3	4	2,8276	0,7069
4	5	2,8276	0,5655

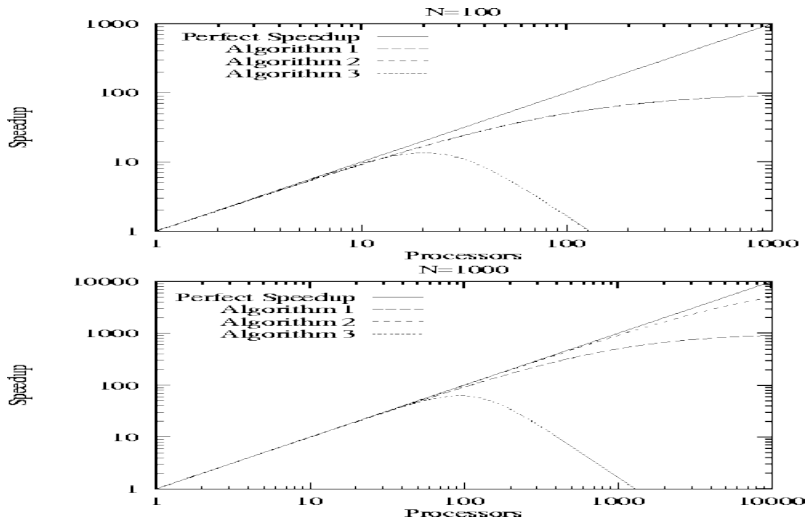
Ví dụ: Kết quả theo biểu thức

- Xét bài toán có kích thước N trên máy P bộ xử lý. Thuật giải tuần tự có thời gian thực hiện là $N + N^2$.
- Giả sử có 3 thuật giải song song:
  - $T_{1p}(N) = \frac{N^2}{P} + N$
  - $T_{2p}(N) = \frac{N + N^2}{P} + 100$
  - $T_{3p}(N) = \frac{N + N^2}{P} + 0.6P^2$

- Xét khi N=100, P=12, $Speedup$ = 10,8
- Khi P tăng lên, thuật giải T_{3p} rất xấu do
- $S_P^{(3)}(n) = \frac{N + N^2}{\frac{N + N^2}{P} + 0.6P^2} \to 0$, khi $P \to \infty$

- Khi P khá lớn, thuật giải T_{1p}, T_{2p} có dạng như sau:
- $S_P^{(1)}(n) = \dfrac{N+N^2}{N+\frac{N^2}{P}} \to \dfrac{N+N^2}{N}$, khi $P \to \infty$
- $S_P^{2}(n) = \dfrac{N+N^2}{\frac{N+N^2}{P}+100} \to \dfrac{N+N^2}{100}$, khi $P \to \infty$

- Tứ đây, với N=100 hai thuật giải T_{1p}, T_{2p} có Speedup như nhau.
- Với $P > 1$ và khi đó nếu $N > \dfrac{100P}{P-1} \Leftrightarrow N > 100 + \dfrac{N}{P}$
- Suy ra $\dfrac{N^2}{P} + N > \dfrac{N^2}{P} + 100 + \dfrac{N}{P} \Leftrightarrow N + \dfrac{N^2}{P} > \dfrac{N+N^2}{P} + 100$
- Thì thuật giải T_{2p} tốt hơn T_{1p}



6.4 Định luật Amdahl

- Trong một vài chương trình có nhiều đoạn khác nhau, có thể có những đoạn dễ dàng song song hóa, nhưng có những đoạn chỉ thực hiện tuần tự.
- Khi đó, đoạn tuần tự không thể song song được, gọi là tình trạng bottle-neck (thắt cổ chai).

- **Định luật:** Một chương trình bao gồm hai đoạn, một đoạn chỉ có thể thực hiện tuần tự và một đoạn hoàn toàn có thể song song hóa. Nếu đoạn tuần tự tiêu phí phần *f* của tổng thời gian thi hành chương trình, thì speedup của thuật giải song song không vượt quá $\frac{1}{f}$.

- Để hình dung, chia thời gian thi hành $T_s(n)$ của thuật giải tuần tự (ban đầu) thành *f* phần.
- Khi đó có thể viết $T_s(n) = fT_s(n) + (1-f)T_s(n)$

- Nếu thuật giải luôn luôn cần $\frac{1}{f}$ tuần tự, phần còn lại có thể song song, thì thời gian thi hành thuật giải song song $T_p(n)$ của cùng bài toán trên *p* bộ xử lý là: $T_p(n) = fT_s(n) + \frac{(1-f)T_s(n)}{P}$

- Từ đây, độ tăng tốc speedup là

$$S_p(n) = \frac{T_s(n)}{T_p(n)} = \frac{T_s(n)}{fT_s(n) + \frac{(1-f)T_s(n)}{P}} = \frac{1}{f + \frac{1-f}{P}}$$

- Khi p khá lớn, Speedup bị chặn trên bởi:

$$S_p(n) = \frac{1}{f + \frac{1-f}{p}} \leq \frac{1}{f}$$

- Chẳng hạn, phần tuần tự là 10% thì speedup tối đa là 10.

6.5 Chi phí thời gian

- Chi phí thời gian của thuật giải song song thông thường có các loại thời gian như:
  - Thời gian thi hành (*execution time*)
  - Thời gian tính toán (*computation time*)
  - Thời gian giao tiếp (*communication time*)
  - Thời gian chờ/nhàn rỗi (*idle time*)

Thời gian thi hành

- Thời gian thi hành T của thuật giải song song bao gồm:
  - Thời gian tính toán
  - Thời gian giao tiếp
  - Thời gian chờ
- Thời gian của tiến trình thực hiện chậm nhất

$$T = \max_i (T^i_{comp} + T^i_{comm} + T^i_{idle})$$

Tuy nhiên, đôi khi

Còn lấy tổng thời gian tính toán, thời gian giao tiếp và thời gian chờ đợi của một tiến trình thứ i bất kỳ nào đó:

$$T = T^i_{comp} + T^i_{comm} + T^i_{idle}$$

- Hoặc trung bình của tổng thời gian tính toán, giao tiếp và chờ trên tất cả các process (task).

$$T = \frac{1}{P}(T_{comp} + T_{comm} + T_{idle})$$

$$= \frac{1}{P}\left(\sum_{i=0}^{P-1} T_{comp}^{i} + \sum_{i=0}^{P-1} T_{comm}^{i} + \sum_{i=0}^{P-1} T_{idle}^{i}\right)$$

Thời gian tính toán

- Thời gian tính toán của thuật giải T_{comp} là thời gian sử dụng cho những thao tác tính toán.
- Thời gian tính toán thường phụ thuộc vào kích thước bài toán.

- Kích thước được biểu diễn bằng một tham số N, hoặc được biểu diễn bằng một tập hợp các tham số $N_1, N_2, ..., N_m$.
- Nếu thuật giải song song lặp lại việc tính toán, thì thời gian tính toán cũng sẽ phụ thuộc vào số task hay số process.

Thời gian giao tiếp

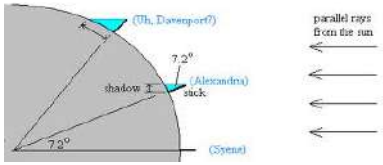
- Thời gian giao tiếp T_{comm} là thời gian mà những task sử dụng cho việc truyền các thông điệp giữa các task với nhau.
- Chi phí cuả việc gởi thông điệp giữa hai task được định vị trên những processor khác nhau có thể biểu diễn bằng hai tham số:

- Thời gian khởi động việc truyền thông điệp t_s,
- Và thời gian truyền L dữ liệu t_wL (trong đó t_w là thời gian truyền 1 dữ liệu

- Vậy $T_{comm} = t_s + t_wL$

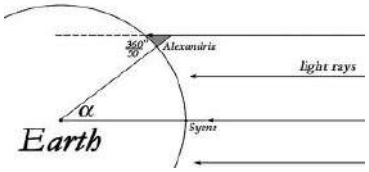
Thời gian chờ

- Một processor thường rãnh rỗi do việc thiếu công việc tính toán hoặc thiếu dữ liệu.
- Trong trường hợp thứ nhất, thời gian rãnh rỗi này có thể ngăn chặn bằng cách sử dụng kỹ thuật load balancing.



- Trường hợp thứ hai, processor ở tình trạng nhàn rỗi do việc tính toán và truyền thông điệp đòi hỏi những dữ liệu từ xa.
- Thời gian này có thể tránh được bằng cách cho processor thực hiện những tính toán và giao tiếp khác.

- Kỹ thuật này được biết như là *overlapping computation and communication,*
- Khi đó một tính toán địa phương được thực hiện đồng thời với những tính toán và giao tiếp tứ xa.



- Hoặc tạo ra nhiều task trên một processor. Khi một task làm trở ngại việc đợi dữ liệu tứ xa, nó có thể chuyển sang một task khác mà dữ liệu đã sẵn sàng thực hiện.
- Cách này tiện lợi vì đơn giản, nhưng chỉ có hiệu qủa nếu như chi phí thực hiện một task mới ít hơn chi phí thời gian nhàn rỗi

Ví dụ: Minh họa thời gian

- Chúng ta xem xét một miền gồm N x N x Z điểm lưới, trong đó Z là số điểm theo phương thẳng đứng, N là số điểm theo hai phương ngang.
- Một task tương ứng một miền con gồm N x (N/P) x Z điểm lưới, với P là số bộ xử lý (*chia miền theo một phương ngang*).

- Giả sử ở mỗi bước thời gian, mỗi task thực hiện cùng một tính toán tại mỗi điểm lưới.
- Khi đó thời gian tính toán là

$$T_{comp} = t_c N^2 Z$$

Trong đó t_c là thời gian tính toán trung bình cho một điểm lưới.

- Đối với thời gian truyền dữ liệu, mỗi task sẽ giao tiếp với hai task lân cận (hai mặt), trong đó mỗi mặt gồm NZ phần tử, nên mỗi task trao đổi $2NZ$ dữ liệu.
- Khi đó tổng chi phí truyền tin (gửi và nhận) trên P bộ xử lý sẽ là

$$T_{comm} = 2P(t_s + t_w 2NZ)$$

- Giả sử P chia hết N,số lượng tính toán trên mỗi điểm lưới là hằng, thời gian chờ không đáng kể.
- Chi phí thời gian:

$$T = \frac{T_{comp} + T_{comm}}{P} = \frac{N^2 Z}{P} t_c + 2t_s + 4NZT_w$$

- Efficiency của thuật giải như sau:

$$E = \frac{t_c N^2 Z}{PT} = \frac{t_c N^2 Z}{N^2 Z t_c + P2t_s + P4NZt_w}$$

- Với thuật giải có chi phí thời gian và hiệu năng như trên, nhận xét rằng:
  - Hiệu suất sẽ giảm đi khi tăng P, t_s và t_w.
  - Hiệu suất sẽ tăng lên khi tăng N, Z và t_c.
  - Thời gian thi hành giảm khi tăng P nhưng bị chặn bởi chi phí chuyển đổi giữa hai miếng dữ liệu.
  - Thời gian thi hành tăng khi tăng N, Z, t_c, t_s và t_w.

Bài tập

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 $\Delta x = \frac{1}{N}$, $x_i = i\Delta x, \forall i = 0,1,\ldots,N$

Thuật toán tuần tự

```
#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;
}
```


Hiện thực song song với OpenMP

```
#include <stdio.h>
#define N 10000000
#include <omp.h>
int main() {
   double delta, x, sum =0.0, s = 0.0, pi;
   delta = 1.0/N;
#pragma omp parallel private(i,sum)
   {
#pragma omp for
      for ( int i = 0; i < N; i++ ){
         x = i*delta;
         sum += 4.0/(1.0 + x*x);
      }
#pragma omp critical
      s += sum;
   }
   pi = s*delta;
   printf( "Pi = %20.18f\n", pi );
   return 1;
}
```