TÍNH TOÁN SONG SONG

PARALLEL COMPUTING

Phan Trọng Tiến
Bộ môn CNPM – Khoa CNTT
Học Viện Nông nghiệp Việt Nam
Email: phantien84@gmail.com
Website: http://timoday.edu.vn

Nội dung

- Xử lý song song
- Xu hướng phát triển của CPU
- Các mô hình lập trình song song
  - Truyền thống
  - Dựa trên dữ liệu

XỬ LÝ SONG SONG

Vai trò của xử lý song song trong cuộc sống

- Xử lý song song hoàn toàn không xa lạ trong cuộc sống
  - Quầy tính tiền ở siêu thị
  - Mua vé vào công viên
  - Đường cao tốc nhiều làn xe
- Nhiều sự việc phức tạp trong cuộc sống đều xảy ra đồng thời

Quầy tính tiền ở siêu thị

Source: http://www.baobinhthuan.com.vn/web/data/news/2009/1/21476/thitruong.jpg

Source: **Checkouts**
http://www.sturm.si/en/images/iman/gal_img.0077.jpg

Mua vé vào công viên

*Source:*http://www.vtc.vn/newsimage/original/vtc_216925_suoitien.jpg

Source: **The Old Entrance**
http://www.matterhorn1959.com/blog1/4.TicketBooths.jpg

Đường cao tốc nhiều làn xe

Source: **Stock Photo: Cars In Traffic On Multi-lane Highway**
http://www.worldofstock.com/closeups/TRC4948.php

Vai trò của xử lý song song trong cuộc sống

- Tại sao lại phải xử lý song song?
  - Tiết kiệm thời gian + tiền bạc

  - Chia nhỏ ra để xử lý nhanh hơn

➔xử lý song song giúp nâng cao năng suất

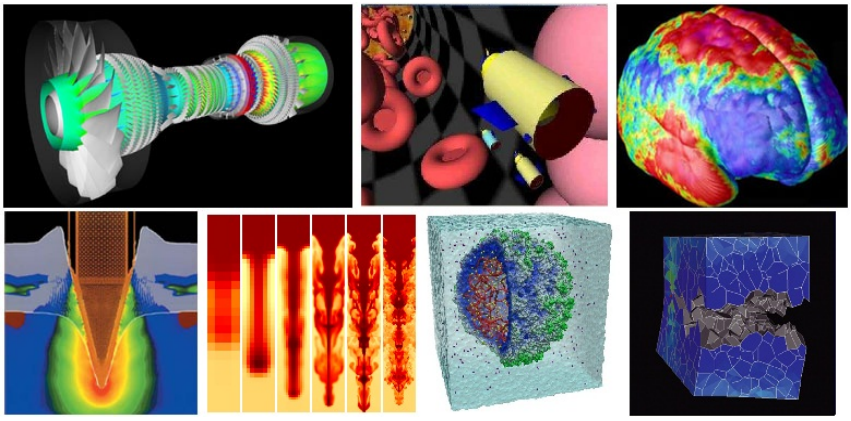
Ứng dụng xử lý song song

- Những bài toán phức tạp ở nhiều lĩnh vực thực tế đòi hỏi cao về tốc độ
- Đưa ra quyết định nhanh dựa trên lượng lớn dữ liệu như:
  - Dự báo thời tiết (dự báo bão, lũ, …)
  - Chuẩn đoán y khoa
  - Kinh tế - tài chính (mua bán chứng khoán)
  - Quân sự
  - …
- Xây dựng mô hình để tính toán và phân tích trên máy tính

Ứng dụng xử lý song song

- Mô phỏng thực tế xem xét đến nhiều yếu tố (tham số) khác nhau ➔ nhiều khả năng/thể hiện của một bài toán

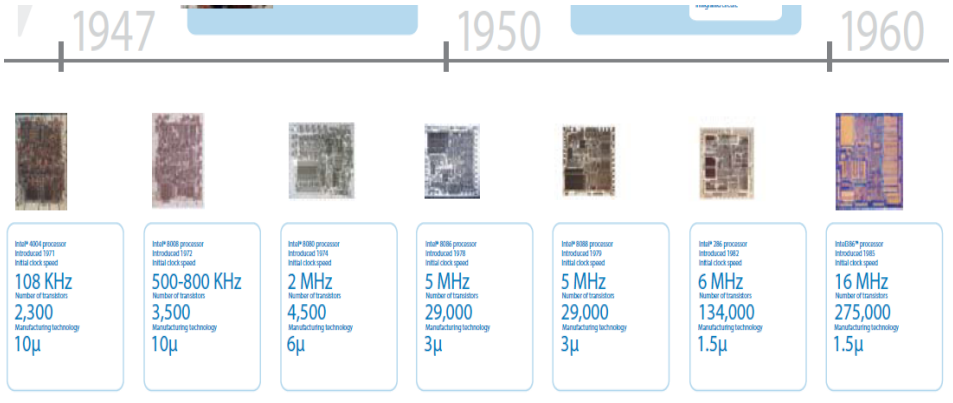
➔ có thể được xử lý song song



XU HƯỚNG PHÁT TRIỂN CỦA CPU

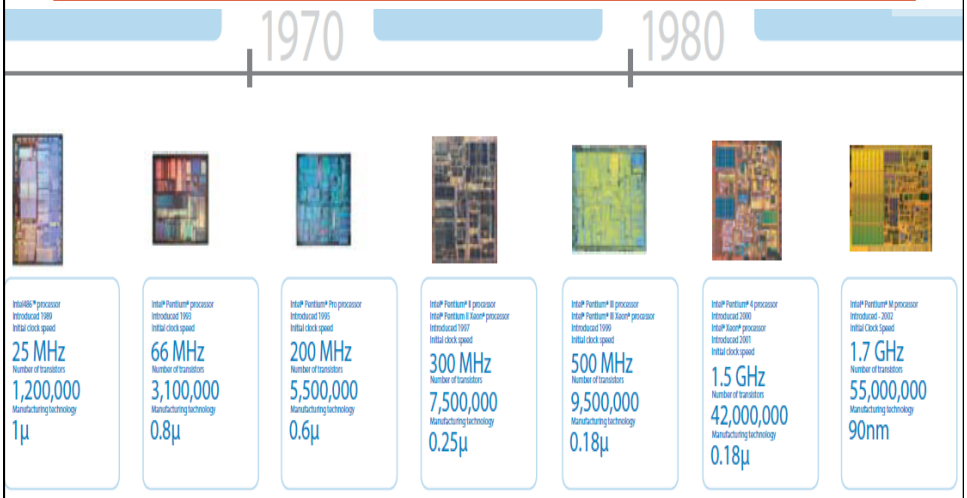
Sự phát triển của CPU

- Central Processing Unit (CPU)
- 60 năm phát triển của CPU Intel



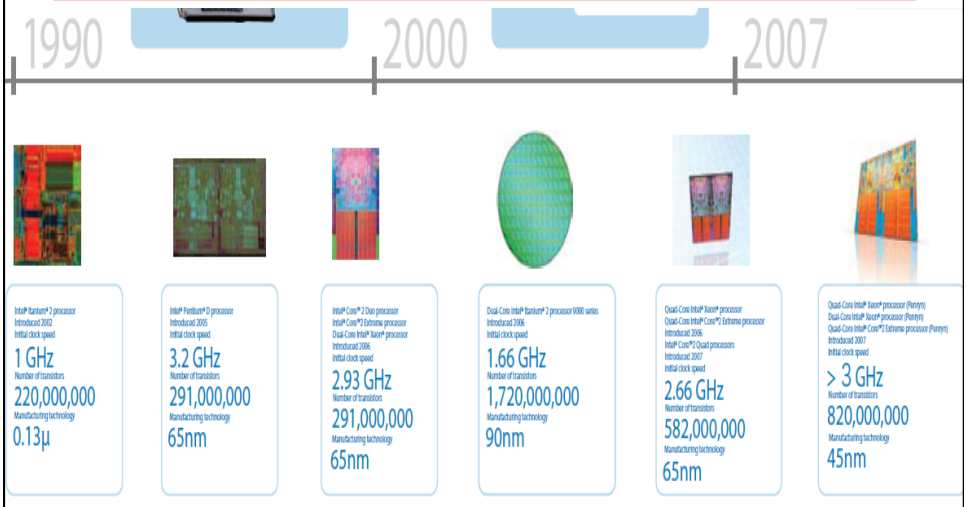
60 YEARS OF THE TRANSISTOR: 1947 – 2007
http://www.intel.com/technology/timeline.pdf

Sự phát triển của CPU



60 YEARS OF THE TRANSISTOR: 1947 – 2007
http://www.intel.com/technology/timeline.pdf

Sự phát triển của CPU



60 YEARS OF THE TRANSISTOR: 1947 – 2007
http://www.intel.com/technology/timeline.pdf

Sự phát triển của CPU

- CPU nhiều lõi ngày càng phổ dụng
- Tại sao lại phải chuyển từ đơn lõi sang nhiều lõi?
- Từ 1975 hiệu năng CPU vẫn tăng liên tục (100x/10 năm)
- Những rào cản khi tăng tốc CPU đơn lõi
  - **Power Wall**
  - **Memory Wall**
  - **Complexity Wall**

Power Wall

Power Wall

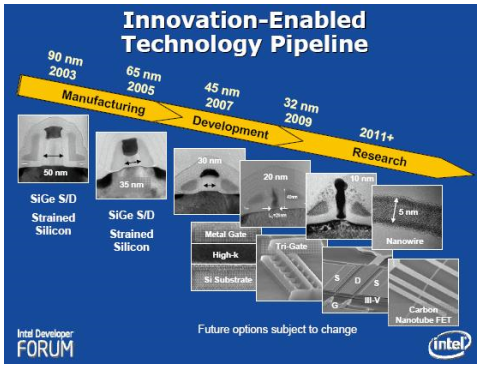
- Công suất (*W*) của CPU tỉ lệ với NCV^2f
  - *N*: số lượng transistor
  - *C*: điện dung
  - *V*: số vol
  - *f*: tần số
- Xu hướng ↑*N* ↓*C* ↓*V*

(Công nghệ transistor mới)

➔ Sẽ như thế nào nếu ↑f

Power Wall

- Mỗi thế hệ mạch in mới (90, 60, 45, 32, 22, 16, 11 nm)
  - Số lượng transistor/die tăng gấp đôi (↑N)
  - Kích thước transistors thu nhỏ hơn (↓C)
  - Sử dụng số vol thấp hơn (↓V).



http://www.digital-daily.com/cpu/intel_penryn/

Power Wall

- Điện năng cung cấp đã giảm từ 15V xuống còn 1V trong vòng gần 30 năm
- Ngưỡng tối thiểu là 0.7V

➔ còn giảm thêm được $(1.0/0.7)^2=2X$

- Nhưng khi tăng mật độ (↑N) và xung nhịp (↑f) của CPU lên thì mức tiêu hao năng lượng tăng từ 1 W lên 100 W chỉ trên 1 cm^2

➔ khó tản nhiệt

- Đã đạt tới giới hạn ➔ xung nhịp CPU không giúp tăng tốc hệ thống như trước nữa (kể từ P4)

Memory Wall

Memory Wall

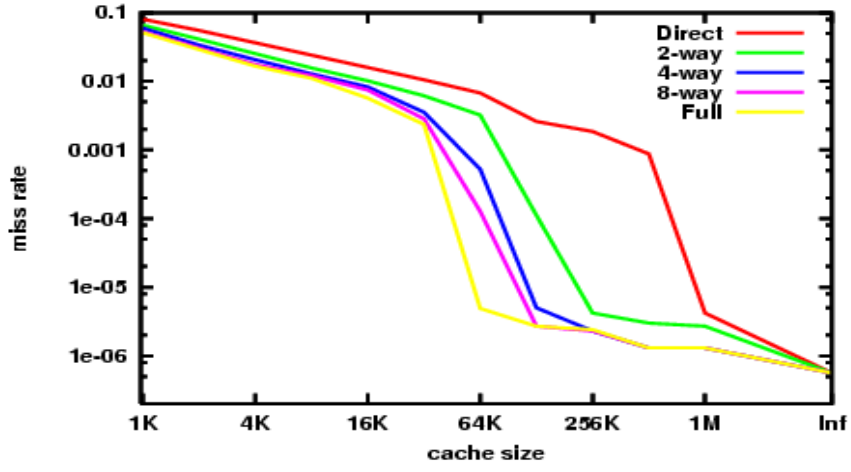
- Độ trễ của DRAM cải thiện không đáng kể ➔ dùng cache của CPU
- CPU cache tốn kém do miss (có thể mất 300 xung đồng hồ)
- Để giảm miss ½ ➔ tăng gấp 4 lần dung lượng cache (kích thước thực sự tăng!)

  ➔Nhiều transitor trong CPU được dùng cho việc xử lý truy xuất bộ nhớ này
- Cách dễ dàng hơn để tăng băng thông bộ nhớ ➔ Truy xuất bộ nhớ song song

➔ Nhìn chung hiệu năng được nâng cao

Memory Wall

- Tỉ lệ miss khi tăng dung lượng cache



http://en.wikipedia.org/wiki/File:Cache,missrate.svg

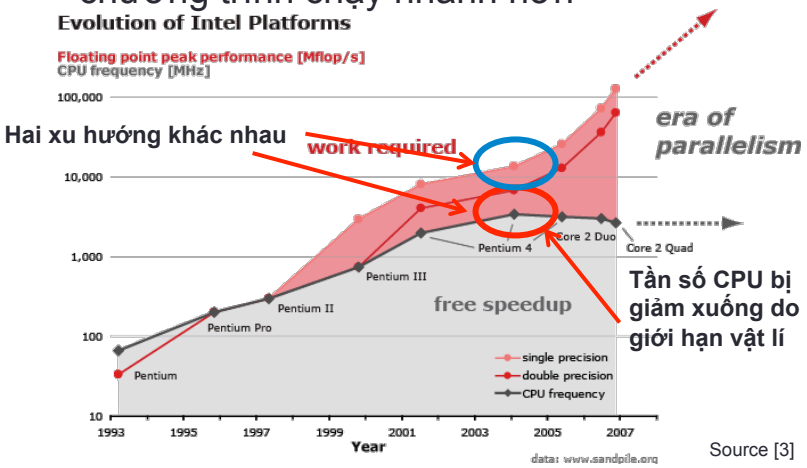
Xu hướng phát triển CPU

- Do những rào cản trên nên CPU đơn lõi
  - Hiệu năng sẽ tăng rất chậm (5 – 10%)
  - Tốt cho phần mềm truyền thống (chạy tuần tự
- Giải pháp của nhà sản xuất phần cứng để tăng tốc CPU 100x là tăng số lõi/nhân thay vì tăng f.

➔ mở ra một kỉ nguyên song song mới

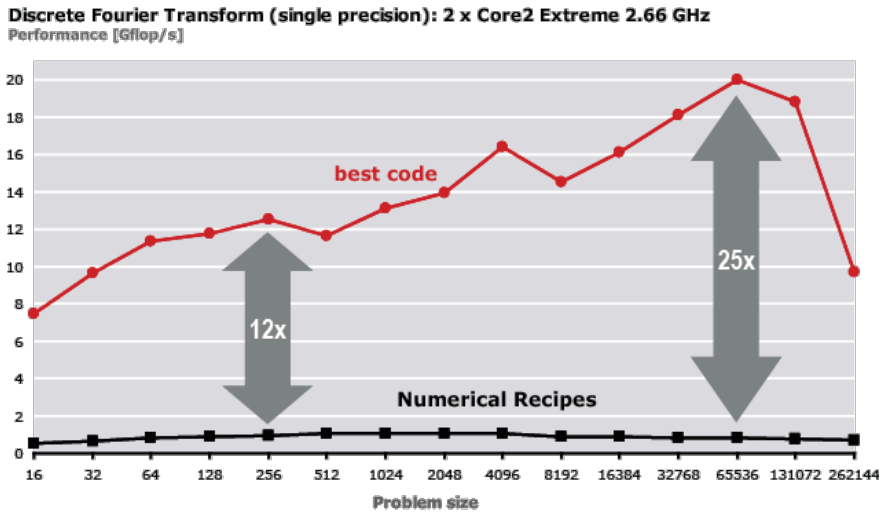
Máy tính và hiệu năng phần mềm

- Trước đây chỉ cần nâng cấp phần cứng ➔ chương trình chạy nhanh hơn



Source [3]

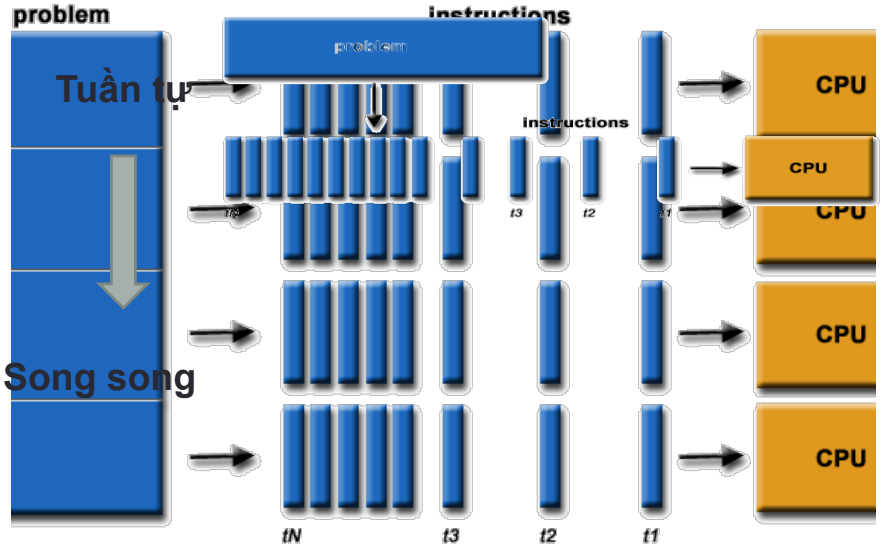
Máy tính và hiệu năng phần mềm



Source [3]

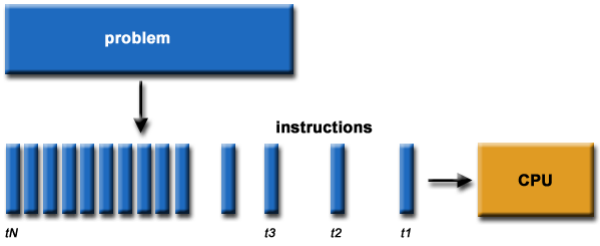
LẬP TRÌNH SONG SONG

Lập trình song song truyền thống



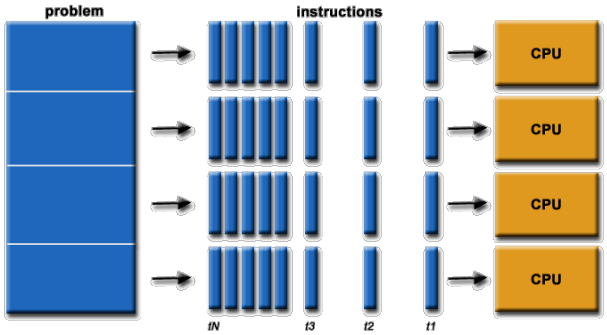
Tính toán song song là gì? (1)

- Thông thường, phần mềm được viết cho tính toán **tuần tự** *(serial computation)*:
  - Được chạy trên máy tính đơn với một bộ xử lý trung tâm (CPU).
  - Mộ bài toán (problem) sẽ được chia thành một chuỗi cá câu lệnh rời rạc.
  - Các câu lệnh được thực hiện một cách tuần tự.
  - Tại mỗi thời điểm chỉ thực hiện được một câu lệnh.



Tính toán song song là gì? (2)

- Ý nghĩa đơn giản nhất, ***tính toán song song*** là việc sử dụng đồng thời nhiều tài nguyên máy tính để giải quyết bài toán về tính toán.
  - Để chạy trên nhiều CPU
  - Một bài toán được chia thành các phần riêng biệt mà có thể được giải quyết đồng thời.
  - Mỗi phần được chia nhỏ hơn dưới một dãy các câu lệnh
- Các câu lệnh của mỗi phần thực thi đồng thời trên các CPU khác nhau



Tính toán song song: Các tài nguyên

- Các nguồn tài nguyên tính toán có thể bao gồm:
  - Một máy tính đơn với nhiều bộ vi xử lý (CPU);
  - Một máy tính đơn với một hoặc nhiều CPUvà một số tài nguyên chuyên dụng như GPU, FPGA …;
  - Một số lượng tuỳ ý các máy tính được kết nối bởi một mạng máy tính;
  - Hoặc kết hợp của cả hai loại trên.

Tính toán song song: Vấn đề tính toán

- Vấn đề tính toán thường được thể hiện qua các đặc điểm như khả năng:
  - Chia thành các phần riêng biệt các công việc để có thể giải quyết cùng một lúc;
  - Thực thi nhiều câu lệnh chương trình tại nhiều thời điểm;
  - Giải quyết bài toán trong thời gian ít hơn với nhiều nguyền tài nguyên tính toán hơn là thực thi chỉ trên một tài nguyên tính toán duy nhất.

Tính toán song song: để làm gì? (1)

- Tính toán song song là sự tiến hoá của tính toán tuần tự để cố gắng mô phỏng các trạng thái diễn ra trong thế giới tự nhiên: nhiều phức tạp, các sự kiện liên quan xảy ra cùng một thời điểm, nhưng trong cùng một chuỗi.
- Ví dụ:
  - Quỹ đạo hành tinh và thiên hà
  - Các mô hình thời tiết và đại dương
  - Kiến tạo địa chất
  - Giờ cao điểm ở Hà Nội
  - Dây truyền lắp ghép ô tô
  - Các hoạt động hàng ngày trong một doanh nghiệp
  - Xây dựng một trung tâm mua sắm
  - …

Tính toán song song: để làm gì? (2)

- Tính toán song song có thể được coi là "tính toán hiệu năng cao" và là động lực để mô phỏng cho các hệ thống phức tạp và giải quyết "các bài lớn" như:
  - Dự báo thời tiết và khí hậu
  - Các phản ứng hoá học và hạt nhân
  - Các bài toán sinh học và gen người
  - Các hoạt động địa chất
  - Các thiết bị cơ khí – như chân tay giả cho tàu vũ trụ
  - Các mạch điện tử
  - Các quy trình sản xuất

Tính toán song song: để làm gì? (3)

- Ngày nay các ứng dụng thương mại đang là động lực thúc đẩy các nhà phát triển máy tính và phần mềm tạo ra các máy tính có tốc độ nhanh hơn. Vì các ứng dụng này yêu cầu xử lý một số lượng lớn dữ liệu và tinh vi phức tạp. Ví dụ như các ứng dụng:
  - Các cơ sở dữ liệu song song, data mining
  - Thăm dò dầu khí
  - Các máy chủ tìm kiếm, các dịch vụ thương mại
  - Máy tính trợ giúp chuẩn đoán trong y học
  - Quản lý các tập đoàn quốc gia và đa quốc gia
  - Đồ hoạ cải tiến và ảo hoá
  - Video mạng và các công nghệ đa phương tiện
  - Môi trường làm việc cộng tác
- Cuối cùng, tính toán song song là một cố gắng để tối đa hoá những yêuvô hạn nhưng dường như chúng ta vẫn cần thêm thời gian..

Tại sao phải tính toán song song? (1)

- Đây là một câu hỏi nhiều người đặt ra! Tính toán song song là phức tạp trên nhiều khía cạnh!
- Các lý do chính sử dụng tính toán song song:
  - Tiết kiệm thời gian
  - Giải quyết các bài toán lớn
  - Xử lý đồng thời tại cùng một thời điểm

Lập trình song song truyền thống

Tuần tự

```
void quicksort(int * a, int n)
{
  if (n <= 1) return;
  int s = partition(a,n);

  quicksort(a,s);
  quicksort(a+s,n-s);
}
```

Song song

```
void quicksort(int * a, int n)
{
  if (n <= 1) return;
  int s = partition(a,n);
  parallel_invoke(
     [&]{quicksort(a,s);},
     [&]{quicksort(a+s,n-s);});
}
```

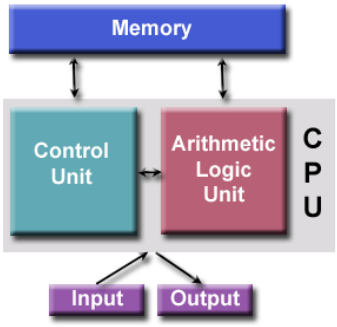
- Ý tưởng đơn giản nhưng mang lại hiệu quả cao

Lập trình song song truyền thống

- Lập trình song song không đơn giản
- 3 khó khăn chủ đạo
  - Cách suy nghĩ tuần tự
  - Chuyển đổi từ tuần tự sang song song
  - Khả năng mở rộng theo phần cứng
- Còn nhiều vấn đề khác như debug, kiểm thử, hiệu năng …

Cách suy nghĩ tuần tự

- Kiến trúc Von Neumann



Source [2]

➔ kết quả có tính tất định (chắc chắn)

- Không còn phù hợp khi chuyển sang song song ➔ trạng thái không xác định

Cách suy nghĩ tuần tự

Tuần tự

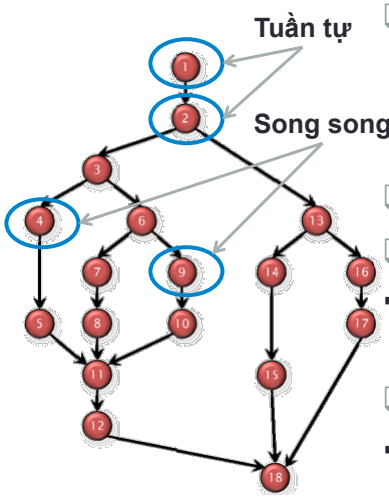
```
void quicksort(int * a, int n)
{
  if (n <= 1) return;
  int s = partition(a,n);

1 quicksort(a,s);
2 quicksort(a+s,n-s);
}
```

Song song

```
void quicksort(int * a, int n)
{
  if (n <= 1) return;
  int s = partition(a,n);
3 parallel_invoke(
    [&]{quicksort(a,s);},
    [&]{quicksort(a+s,n-s);});
}
```

- Thời điểm 1 & 2 ➔ biết **chắc** tình trạng của mảng a
- Thời điểm 3 ➔ không biết **chắc** tình trạng của mảng a

Chuyển đổi tuần tự sang song song



Source [4]

- Mô hình mối quan hệ giữa **tuần tự** và **song song** bằng một đồ thị có hướng
- Đỉnh – lệnh
- Cạnh nối x → y

➔ lệnh x thực thi trước lệnh y (tuần tự)

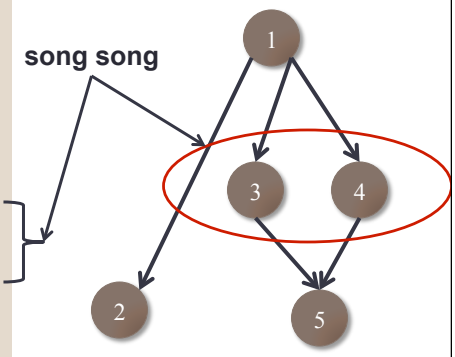
- Không có cạnh nối x và y

➔ x || y (song song)

Chuyển đổi tuần tự sang song song

- Hàm tính Fibonacci đệ qui

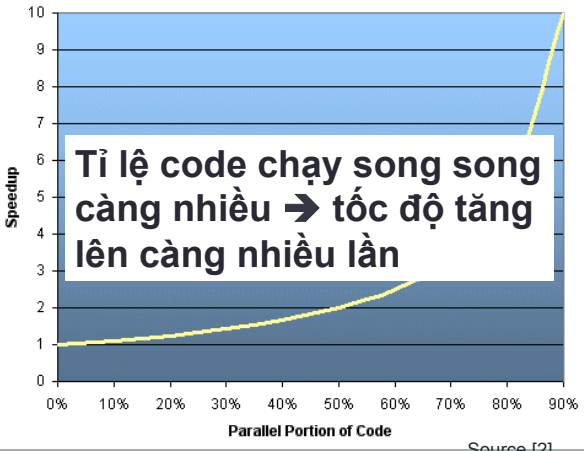
```
   int fib(int n)
   {
1      if (n < 2)
2            return n;
       else
       {
3            int x = fib(n-1);
4            int y = fib(n-2);
5            return x + y;
       }
   }
```



Chuyển đổi tuần tự sang song song

- Định luật Amdahl

$$speedup = \frac{1}{1-p}$$

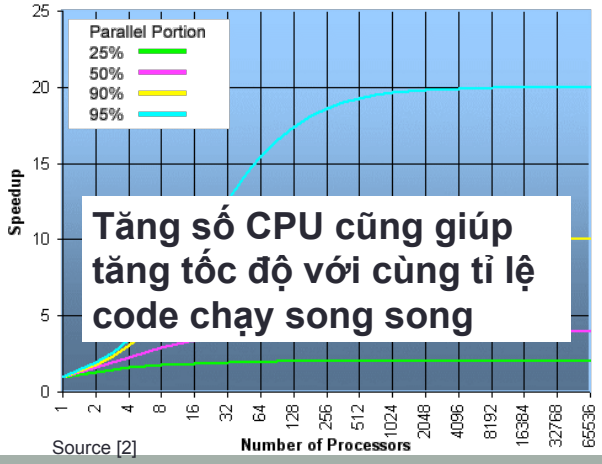


Source [2]

Chuyển đổi tuần tự sang song song

- Khi tăng số CPU lên

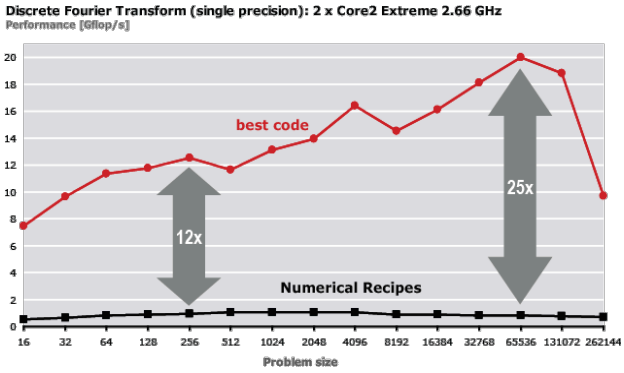
$$speedup = \frac{1}{\frac{p}{N} + s}$$



Source [2]

Khả năng mở rộng theo phần cứng

- Liệu khi phần cứng thay đổi (số CPU thay đổi) thì code có phải thay đổi không?



➔phải thay đổi code ➔ vấn đề lớn ???

```
int main(int argc, char *argv[])
{
  pthread_t thread;
  thread_args args;
  int status;
  int result;
  int thread_result;
  if (argc < 2) return 1;
  int n = atoi(argv[1]);
  if (n < 30) result = fib(n);
  else {
    args.input = n-1;
    status = pthread_create(&thread, NULL, thread_func,
(void*) &args );
    // main can continue executing
executes.
    result = fib(n-2);
    // Wait for the thread to term
    pthread_join(thread, NULL);
    result += args.output;
  }
  printf("Fibonacci of %d is %d.\n", n, result);
  return 0;
}
```

```
typedef struct
{
  int input;
  int output;
} thread_args;

void *thread_func ( void *ptr )
{
  int i = ((thread_args *) ptr)->input;
  ((thread_args *) ptr)->output =
fib(i);
  return NULL;
}
```

- Tăng tốc 1.5 lần
- Code cho CPU 2 nhân
- CPU 4 nhân

➔ Sửa code

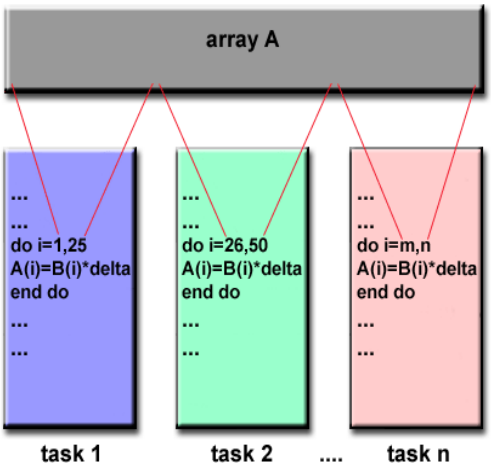
45

Lập trình song song kiểu truyền thống

- Đặc điểm
  - Theo mô hình song song về tác vụ
  - Không phù hợp với các kiến trúc máy tính đa nhân/đa lõi mới.
  - Có nhiều vấn đề về lý thuyết chưa khắc phục được trong mô hình lập trình song song theo tác vụ.

Lập trình song song dựa trên dữ liệu

- Mỗi phần của dữ liệu được chia cho một bộ xử lý (tác vụ) thực hiện



Source [2]

Lập trình song song dựa trên dữ liệu

- Đặc điểm của mô hình
  - Song song các thao tác trên một tập dữ liệu (VD: mảng hoặc ma trận)
  - Mỗi tác vụ xử lý một phần dữ liệu của cùng một CTDL
  - Các tác vụ thực hiện cùng một thao tác trên dữ liệu
  - Phù hợp với kiến trúc đa nhân/đa lõi mới
  - Khắc phục nhiều vấn đề của lập trình song song theo tác vụ.

Lập trình song song dựa trên dữ liệu

- Môi trường lập trình:
  - Ngôn ngữ truyền thống (Fortran)
  - Thư viện đồ họa (OpenGL, Direct3D)
  - Ngôn ngữ mở rộng (CUDA)
  - Ngôn ngữ xử lý theo kiểu mảng

Ngôn ngữ truyền thống

- Bắt nguồn từ lĩnh vực tính toán hiệu năng cao (High Performance Computing)
- Sử dụng rộng rãi trong các siêu máy tính
- Ngôn ngữ Fortran (Fortran 90, HPF)
  - Ví dụ: Cộng hai ma trận A và B

Fortran 90	Fortran 77
C = A + B	DO I = 1, N DO J = 1, N C(I, J) = A(I, J) + B(I, J) END DO END DO

Ngôn ngữ truyền thống

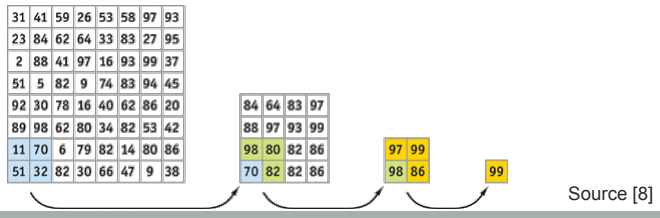
- Ưu điểm
  - Dựa trên những ngôn ngữ phổ biến (Fortran)
- Khuyết điểm
  - Không hỗ trợ nền tảng desktop

Thư viện đồ họa

- Shading Language
  - OpenGL's shading Language (GLSL)
  - DirectX High Level Shader Language (HLSL)
- Ưu điểm
  - Dựa trên đặc điểm chung của phần cứng GPU ➔ làm được trên nhiều GPU khác nhau
- Khuyết điểm
  - Không thể hiện được đặc điểm riêng của mỗi card đồ họa
  - Khó sử dụng

Thư viện đồ họa – ví dụ

```
float main(float2 texcoord  : TEXCOORD0,
uniform samplerRECT img) : COLOR
{
  float a, b, c, d;
  a = f1texRECT(img, texcoord);
  b = f1texRECT(img, texcoord + float2(0, 1));
  c = f1texRECT(img, texcoord + float2(1, 0));
  d = f1texRECT(img, texcoord + float2(1, 1));
  return max(max(a, b), max(c, d));
}
```



Source [8]

Ngôn ngữ mở rộng

- Được phát triển và hỗ trợ bởi nhà sản xuất phần cứng
- Mở rộng dựa trên ngôn ngữ quen thuộc
- Gồm
  - XMT-C (PRAM trên Chip)
  - CUDA - NVIDIA năm 2007
  - CAL(Compute Abstraction Layer) – AMD - Radeon

Ngôn ngữ mở rộng

- Ưu điểm
  - Gần với ngôn ngữ quen thuộc (chủ yếu C)
  - Đơn giản hóa (che đi phần song song)
- Khuyết điểm
  - Khó tối ưu

C	CUDA

void incrementArray (float *a, int N)
{
int i;
for (i=0; i < N; i++)
a[i] = a[i] +1.0f;
}
	__global__ void incrementArray (float *a, int N)
	{
	int idx = blockIdx.x*blockDim.x + threadIdx.x;
	if (idx < N)
	a[idx] = a[idx] + 1.0f;
	}

Ngôn ngữ xử lý theo kiểu mảng

- Tận dụng sức mạnh của các CPU/GPU nhiều nhân
- Ưu điểm
  - Code ngắn gọn và rõ ràng
- Khuyết điểm
  - Ý tưởng thiết kế dựa trên mảng
- Gồm
  - RapidMind – CPU/GPU
  - Acceleware – CPU/GPU

Ngôn ngữ xử lý theo kiểu mảng

C++	RapidMind
Thực hiện tính toán	

`float results[10000];`
 	
`for(int i = 0; i < 10000; ++i)`
`{`
`  result[i] = input1[i] + input2[i];`
`}`
	`Array< 1 , Value1f > output;`
	
	`//Stream Program chạy trên dữ liệu`
	`Program prg = RM_BEGIN {`
	`In<Value1f> a;  // input 1`
	`In<Value1f> b;  // input 2`
	`Out<Value1f> c; // output`
	
	`c = a + b; //thao tác`
	`} RM_END;`
	
	`output = prg(input1, input2);`

Ngôn ngữ xử lý theo kiểu mảng

C++	RapidMind
Xuất kết quả	

`for (int i = 0; i < 10000; ++i)`
`{`
`  std::cout << "output["`
`  << i << "] = ("`
`  << results[i] << ")"`
`  << std::endl;`
`}`
	`const float* results = output.read_data();`
	
	`for (int i = 0; i < 10000; ++i)`
	`{`
	`  std::cout << "output["`
	`  << i << "] = ("`
	`  << results[i] << ")"`
	`  << std::endl;`
	`}`

Tài liệu tham khảo

[1] Chas Boyd, "*Data-parallel Computing*", ACM Queue vol. 6, no. 2, 2008

[2] Blaise Barney, "*Introduction to Parallel Computing*", High Performance Computing Training Workshop, Lawrence Livermore National Laboratory, 2009

[3] Bài viết "*The Problem: Moore's Law and Fast Numerical Software*"

[4] Charles E. Leiserson và Ilya B. Mirman "*How to Survive the Multicore Software Revolution*", Cilk Arts

[5] "*Taking Parallelism Mainstream*", Parallel Computing Developer Center, Microsoft, 2009

[6] Stuart Oberman, "*GPUs: High Performance Arithmetic for Graphics and General Purpose Computation*" ARITH 19, 2009

[7] Ejaz Anwer, "*Handling Multiple Processors in Your Code Using RapidMind*", Codeguru, 2007

[8] Ian Buck and Tim Purcell, "*GPU Gems: Programming Techniques, Tips and Tricks for Real-Time Graphics*", ch. 37, Addison-Wesley, 2004