LẬP TRÌNH SONG SONG VỚI GPU

1. Giới thiệu về lập trình song song và CUDA

1.1 Lập trình/Tính toán song song là gì? What is Parallel Programming/Computing?

1.1.1 What is Parallel Programming/Computing?

1.1.1.1 Serial Programming/Computing

Trước khi đến với khái niệm về Parallel Programming/Computing, chúng ta hãy xem lại về Serial Programming/Computing (Lập trình/Tính toán tuần tự)

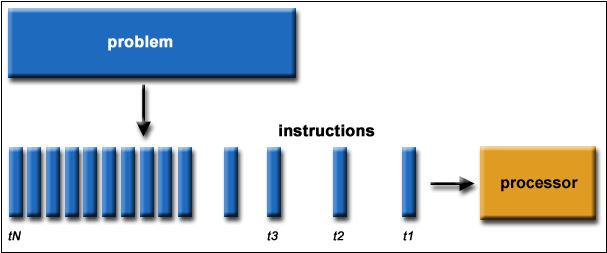
Traditionally, software has been written for **serial computation**:

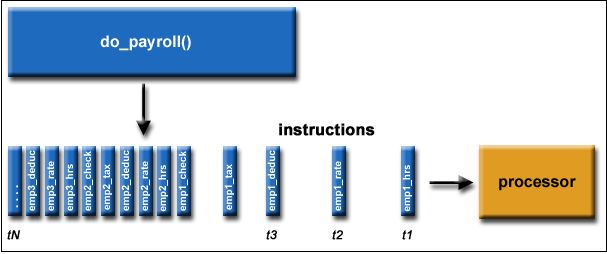
+ A problem is broken into a discrete series of instructions

+ Instructions are executed sequentially one after another

+ Executed on a single processor

+ Only one instruction may execute at any moment in time





+ 1 vấn đề lớn sẽ được chia làm chuỗi những câu lệnh thực thi.

+ Những câu lệnh thực thi được thực hiện tuần từ, lệnh này đến lệnh khác.

+ Quá trình thực thi trên 1 bộ xử lý đơn.

+ Tại mỗi thời điểm chỉ có 1 instruction được thực hiện.

1.1.1.2 Parallel Programming/Computing

In the simplest sense, parallel computing is the simultaneous use of multiple compute resources to solve a computational problem.

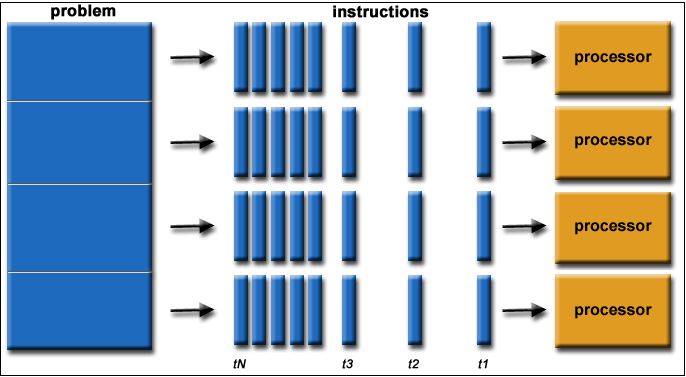
Hiểu đơn giản, tính toán song song là việc sử dụng đồng thời nhiều tài nguyên tính toán để giải quyết một vấn đề.

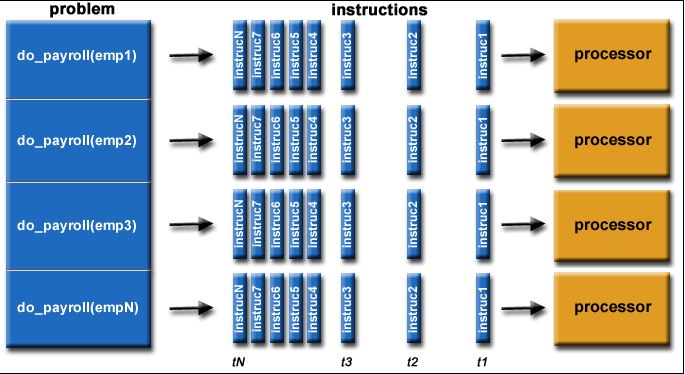
+ A problem is broken into discrete parts that can be solved concurrently

+ Each part is further broken down to a series of instructions

+ Instructions from each part execute simultaneously on different processors

+ An overall control/coordination mechanism is employed





+ Một vấn đề được chia thành các phần rời rạc có thể được giải quyết đồng thời  
+ Mỗi phần lại được chia nhỏ thành một loạt các hướng dẫn  
+ Instructions(các câu lệnh) từ mỗi phần thực hiện đồng thời trên các bộ xử lý khác nhau  
+ Có một cơ chế để kiểm soát / điều phối tổng thể.

1.1.2. Parallel Programming Models

There are several parallel programming models in common use:

Shared Memory (without threads) Bộ nhớ dùng chung

**Threads**

Distributed Memory / Message Passing

Data Parallel

Hybrid

Single Program Multiple Data (SPMD)

Multiple Program Multiple Data (MPMD)

**Threads Model**

This programming model is a type of shared memory programming. Nó là 1 dạng chia sẻ bộ nhớ.

In the threads model of parallel programming, a single "heavy weight" (chính/quan trọng) process can have multiple "light weight"(nhỏ, phụ), concurrent (đồng thời) execution paths.

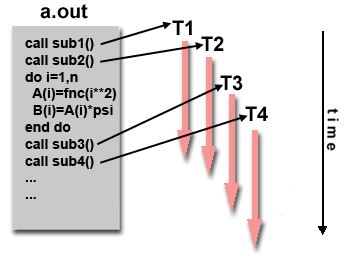
Explain:

- The main program a.out is scheduled to run by the native operating system. a.out loads and acquires all of the necessary system and user resources to run. This is the "heavy weight" process.

Chương trình chính a.out được lên lịch để chạy bởi hệ điều hành gốc. a.out load và giành lại tất cả các tài nguyên hệ thống và tài nguyên người dùng cần thiết để chạy. Đây là quá trình "chính/quan trọng".

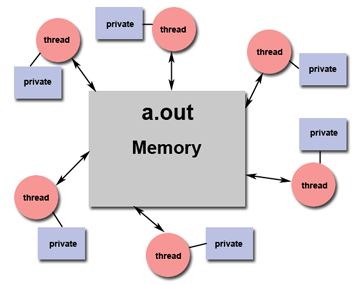
- a.out performs some serial work, and then creates a number of tasks (threads) that can be scheduled and run by the operating system concurrently.

a.out thực hiện một số công việc tuần tự, và sau đó tạo ra một số nhiệm vụ (luồng) có thể được lập lịch và chạy bởi hệ điều hành 1 cách đồng thời.



- Each thread has local data, but also, shares the entire resources of a.out. This saves the overhead associated with replicating a program's resources for each thread ("light weight"). Each thread also benefits from a global memory view because it shares the memory space of a.out.

Mỗi luồng có dữ liệu local, nhưng cũng chia sẻ toàn bộ tài nguyên của a.out. Điều này tiết kiệm chi phí liên quan đến việc sao chép tài nguyên của một chương trình cho mỗi luồng ("trọng lượng nhẹ"). Mỗi luồng cũng được hưởng lợi từ chế độ xem bộ nhớ chung vì nó chia sẻ không gian bộ nhớ của a.out.



A thread's work may best be described as a subroutine within the main program. Any thread can execute any subroutine at the same time as other threads.

Một công việc của thread có thể được mô tả như là một chương trình con trong chương trình chính. Bất kỳ luồng nào cũng có thể thực thi bất kỳ chương trình con nào cùng lúc với các luồng khác.

Threads communicate with each other through global memory (updating address locations). This requires synchronization constructs to ensure that more than one thread is not updating the same global address at any time.

Các luồng giao tiếp với nhau thông qua bộ nhớ toàn cục (cập nhật vị trí địa chỉ). Điều này đòi hỏi các cấu trúc đồng bộ hóa để đảm bảo rằng không có nhiều hơn một luồng cập nhật cùng một địa chỉ toàn cục bất cứ lúc nào.

Threads can come and go, but a.out remains present to provide the necessary shared resources until the application has completed.

Các threads có thể được gọi hoặc kết thúc, nhưng a.out vẫn tồn tại để cung cấp các tài nguyên được chia sẻ cần thiết cho đến khi ứng dụng hoàn tất.

1.2 GPU là gì?

1.2.1 CPU? Central Processing Unit

Trước khi tìm hiểu GPU chúng ta hãy xem lại về CPU.

The central processing unit (CPU) is the unit which performs most of the processing inside a computer. To control instructions and data flow to and from other parts of the computer, the CPU relies heavily on a chipset, which is a group of microchips located on the motherboard.

Bộ xử lý trung tâm (CPU) là đơn vị thực hiện hầu hết quá trình xử lý bên trong máy tính. Để kiểm soát instruction và data flow đến và đi từ các phần khác của máy tính, CPU dựa chủ yếu vào chipset, là một nhóm vi mạch nằm trên bo mạch chủ.

The CPU has two components:

Control Unit: extracts instructions from memory and decodes and executes them

Trích xuất các instructions từ bộ nhớ, giải mã và thực thi.

Arithmetic Logic Unit (ALU): handles arithmetic and logical operations

For 30 years, one of the important methods for the improving the performance of consumer computing devices has been to increase the speed at which the processor’s clock operated.

Trong vòng 30 năm từ 1980s đến khoảng 2004, thì phương thức tối ưu để tăng tốc độ máy tính là việc tăng tần số. Vì như ta biết “thời gian chạy 1 chương trình bằng tổng số câu lệnh nhân với thời giant rung bình của 1 câu lệnh”. => nguyên nhân chính là ở thời điểm này hầu hết PC vẫn là single processor. Điểm yếu của việc tăng tần số là sẽ làm tăng năng lượng.

In 2005, CPU manufacturers began offering processors with two computing cores instead of one.

Over the following years, they followed this development with the release of three-, four-, six-, and eight-core central processor units. Sometimes referred to as the multicore revolution, this trend has marked a huge shift in the evolution of the consumer computing market.

Năm 2005, máy tính 2 lõi ra đời, và nhiều năm sau đó lần lượt 3,4,6,8 lõi được sản xuất. Đánh dấu 1 bước đột phá lớn của máy tính.

Leading CPU manufacturers have already announced plans for 12- and 16-core CPUs, further confiming that parallel computing has arrived for good.

1.2.2 GPU? Graphics processing unit.

A graphics processing unit (GPU) is a specialized electronic circuit designed to rapidly manipulate and alter memory to accelerate the creation of images in a frame buffer intended for output to a display device. GPUs are used in embedded systems, mobile phones, personal computers, workstations, and game consoles. Modern GPUs are very efficient at manipulating computer graphics and image processing. Their highly parallel structure makes them more efficient than general-purpose CPUs for algorithms that process large blocks of data in parallel. In a personal computer, a GPU can be present on a video card or embedded on the motherboard. In certain CPUs, they are embedded on the CPU die.

Một đơn vị xử lý đồ họa (GPU) là một mạch điện tử chuyên dụng được thiết kế để thao tác nhanh chóng và thay đổi bộ nhớ để tăng tốc việc tạo ra các hình ảnh trong một khung bộ đệm để xuất ra các thiết bị display. GPU được sử dụng trong các hệ thống nhúng, điện thoại di động, máy tính cá nhân, máy trạm và bảng điều khiển trò chơi. GPU hiện đại rất hiệu quả trong việc thao tác đồ họa máy tính và xử lý hình ảnh. Cấu trúc song song cao của chúng giúp chúng hoạt động hiệu quả hơn các CPU có mục đích chung cho các thuật toán xử lý các khối dữ liệu lớn song song. Trong một máy tính cá nhân, một GPU có thể có mặt trên một card video hoặc được nhúng trên bo mạch chủ. Đôi khi nó được nhúng trong chính CPU.

In short, a GPU is a processor that is specially-designed to handle intensive graphics rendering tasks. Such as: 3D

Tóm lại, GPU là một bộ xử lý được thiết kế đặc biệt để xử lý các tác vụ dựng hình đồ họa chuyên sâu.

In today, the GPU can now take on many multimedia tasks, such as accelerating Adobe Flash video, transcoding (translating) video between different formats, image recognition, virus pattern matching and others. More and more, the really hard problems to solve are those that have an inherent parallel nature – video processing, image analysis, signal processing.

Ngày nay, GPU hiện có thể thực hiện nhiều tác vụ đa phương tiện, chẳng hạn như tăng tốc video Adobe Flash, chuyển mã (dịch) video giữa các định dạng khác nhau, nhận dạng hình ảnh, kết hợp mẫu vi rút và các định dạng khác. Ngày càng có nhiều vấn đề khó giải quyết là những vấn đề có bản chất song song vốn có - xử lý video, phân tích hình ảnh, xử lý tín hiệu.

1.2.3 CPU vs GPU? Hay GPU hơn gì CPU?

Architecturally, the CPU is composed of just few cores with lots of cache memory that can handle a few software threads at a time. In contrast, a GPU is composed of hundreds of cores that can handle thousands of threads simultaneously. The ability of a GPU with 100+ cores to process thousands of threads can accelerate some software by 100x over a CPU alone. What’s more, the GPU achieves this acceleration while being more power- and cost-efficient than a CPU.

Về mặt kiến trúc, CPU chỉ gồm một hoặc một vài lõi với nhiều bộ nhớ đệm có thể xử lý một vài luồng (phần mềm) cùng một lúc. Ngược lại, một GPU bao gồm hàng trăm lõi có thể xử lý hàng nghìn luồng cùng một lúc. Điều đó cho phép GPU xử lý some software nhanh gấp 100x so với CPU. Hơn nữa, GPU đạt được khả năng tăng tốc này trong khi tiết kiệm năng lượng và chi phí hơn so với CPU.

GPUs are often favoured over CPUs for use in machine learning too, as they can process more functions in a given period of time than CPUs. This makes them better-suited to creating neural networks, due to the volume of data they need to deal with.

GPU thường được ưu tiên hơn CPU để sử dụng trong machine learning, vì chúng có thể xử lý nhiều chức năng hơn trong một khoảng thời gian nhất định so với CPU. Điều này làm cho chúng phù hợp hơn với việc tạo các mạng nơron, do khối lượng dữ liệu mà chúng cần phải xử lý.

The combination of a CPU with a GPU can deliver the best value of system performance, price, and power.

Sự kết hợp giữa CPU và GPU đem lại giá trị tốt nhất về hiệu năng và sức mạnh(khả năng).

1.3 Lập trình song song với GPU là như thế nào?/Why GPU computing?

GPU faster and power than CPU. Mặc dù không hoàn toàn, nhưng ta có thể xem như GPU nhanh hơn và mạnh hơn CPU.

GPU computing accelerate Science Applications

GPU Programming Models:

+ **CUDA**

+ OpenCL

More information: https://www.khronos.org/opencl/

+ OpenMP

More information: http://openmp.org

+ OpenACC

More information at: <http://www.openacc-standard.org/>

1.4 CUDA là gì?

CUDA (Compute Unified Device Architecture) is a parallel computing platform and programming model developed by Nvidia for general computing on its own GPUs (graphics processing units). CUDA enables developers to speed up compute-intensive applications by harnessing the power of GPUs for the parallelizable part of the computation:

+ Developed by NVIDIA. Specific to NVIDIA hardware.

+ Based on C/C++ with some extensions (Adds new syntactic elements to language)

+ NVIDIA and 3-rdparty support for numerical libraries, infrastructure (cuBLAS, cuFFT,...)

+ Lots of example code and good documentation– fairly short learning curve for those with experience of OpenMP and MPI programming

+ Large user community on NVIDIA forums

1.5 CUDA Components

Installing CUDA on a system, there are 3 components:

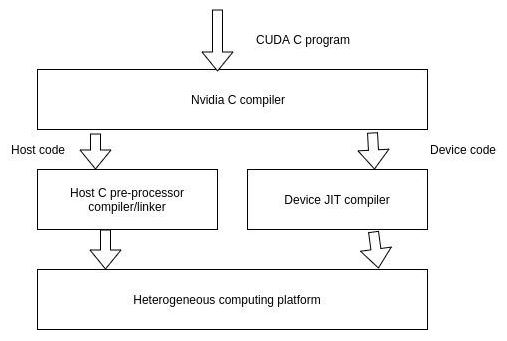
+ driver:

+ low-level software that controls the graphics card

+ toolkits:

+ nvcc CUDA compiler (nvcc means: nvidia c compiler)

Trong khi mã máy có thể được biên dịch bởi trình biên dịch C truyền thống như GCC, mã cuda cần một trình biên dịch đặc biệt để hiểu các hàm api được sử dụng. Đối với GPU Nvidia, trình biên dịch được gọi là NVCC (Nvidia C Compiler).



+ Nsight IDE plugin for Eclipse or Visual Studio

+ profiling and debugging tools

+ several libraries

Ta có thể tạm gọi mã device chạy trên GPU và mã host chạy trên CPU. NVCC xử lý chương trình CUDA và tách mã máy chủ khỏi mã thiết bị. Để thực hiện điều này, các keyword CUDA đặc biệt được tìm kiếm. Mã được thiết kế để chạy GPU (mã thiết bị) được đánh dấu bằng các từ khóa CUDA đặc biệt để ghi nhãn các hàm dữ liệu song song, được gọi là ‘Kernels’. Mã thiết bị được biên dịch bởi NVCC và được thực thi trên GPU.

+ SDK:

+ lots of demonstration examples

+ some error-checking utilities

+ not officially supported by NVIDIA

+ almost no documentation

2. Tổ chức thread trong CUDA

2.1. Execution of a CUDA C program

While writing a CUDA program, the programmer has explicit control on the number of threads that he wants to launch (this is a carefully decided-upon number). These threads collectively form a three-dimensional grid (threads are packed into blocks, and blocks are packed into grids). Each thread is given a unique identifier, which can be used to identify what data it is to be acted upon.

Trong khi viết một chương trình CUDA, lập trình viên đã kiểm soát rõ ràng về số lượng chủ đề mà anh ta muốn khởi chạy (đây là một số được quyết định cẩn thận). Các luồng này tạo thành một lưới ba chiều (các luồng được đóng gói thành các khối và các khối được đóng gói thành các lưới). Mỗi luồng được cung cấp một mã định danh duy nhất, có thể được sử dụng để xác định dữ liệu nào cần được thực hiện.

To execute a kernel on the GPU, the programmer needs to allocate separate memory on the GPU by writing code. The CUDA API provides specific functions for accomplishing this. Here is the flow sequence:

Để thực thi hạt nhân trên GPU, lập trình viên cần phân bổ bộ nhớ riêng trên GPU bằng cách viết mã. API CUDA cung cấp các chức năng cụ thể để hoàn thành việc này.

After allocating memory on the device, data has to be transferred from the host memory to the device memory.

Sau khi cấp phát bộ nhớ trên device, dữ liệu phải được chuyển từ bộ nhớ host sang bộ nhớ device.

After the kernel is executed on the device, the result has to be transferred back from the device memory to the host memory.

Sau khi kernel được thực hiện trên device, kết quả phải được chuyển trở lại từ bộ nhớ device đến bộ nhớ host.

The allocated memory on the device has to be freed-up. The host can access the device memory and transfer data to and from it, but not the other way round.  
     Bộ nhớ được cấp phát trên device phải được giải phóng. Host có thể truy cập vào bộ nhớ device và truyền dữ liệu đến và từ đó.

2.2 Resource Assignment to Blocks

Execution resources are assigned to threads per block.

Các tài nguyên thực thi được gán cho các threads cho mỗi block.

Resources are organized into Streaming Multiprocessors (SM).

Tài nguyên được tổ chức thành Bộ xử lý đa luồng (SM).

Multiple blocks of threads can be assigned to a single SM.

Nhiều blocks of threads có thể được gán cho một SM duy nhất.

In recent CUDA devices, a SM can accommodate up to 1536 threads. The configuration depends upon the programmer. This can be in the form of 3 blocks of 512 threads each, 6 blocks of 256 threads each or 12 blocks of 128 threads each. The upper limit is on the number of threads, and not on the number of blocks.

Trong các thiết bị CUDA gần đây, Streaming Multiprocessors (SM) có thể chứa tới 1536 threads. Cấu hình phụ thuộc vào lập trình viên. Điều này có thể ở dạng 3 blocks 512 threads, 6 khối 256 threads hoặc 12 khối của 128 threads. Giới hạn trên là số lượng chủ đề chứ không phải số lượng khối.

Thus, the number of threads that can run parallel on a CUDA device is simply the number of SM multiplied by the maximum number of threads each SM can support. In this case, the value comes out to be SM x 1536.

Do đó, số lượng các threads có thể chạy song song trên một thiết bị CUDA đơn giản là số SM nhân với số lượng threads tối đa mà mỗi SM có thể hỗ trợ. Trong trường hợp này, giá trị xuất hiện là SM x 1536.

Refer:

<https://techblog.vn/xu-li-song-song>

<https://computing.llnl.gov/tutorials/parallel_comp/>

<http://www.thegpu.com/>