BIGR LAB

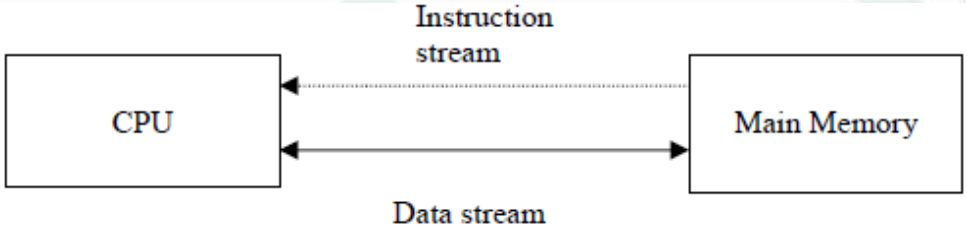
big data research laboratory

History of GPU Computing

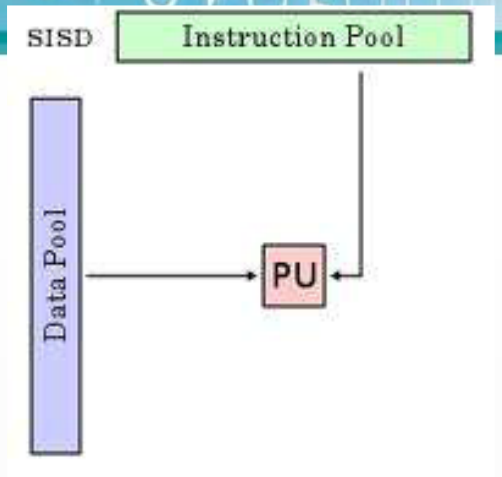
蘇育生

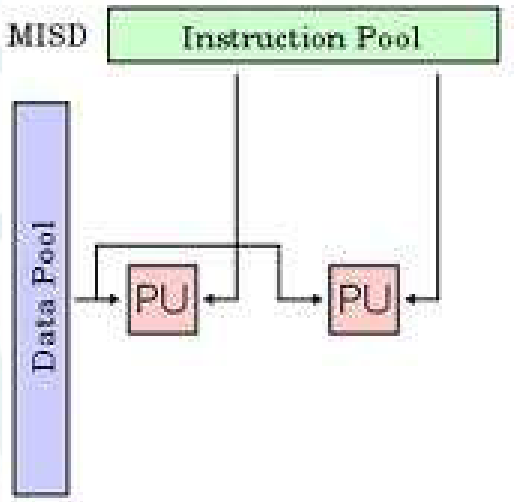
Flynn Classification

- A taxonomy of computer architecture proposed by Flynn in 1966
- It is based two things:
  - Instruction stream
  - Data stream

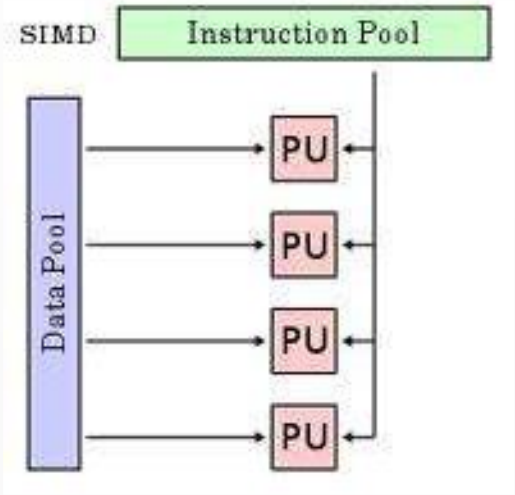


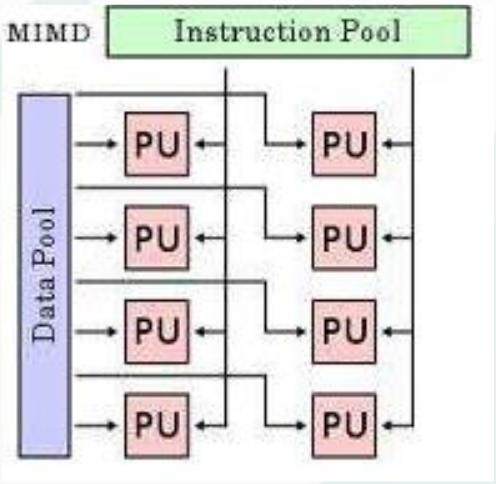
	Single instruction	Multiple instruction
Single data	SISD (Single Instruction stream over a Single Data stream)	MISD (Multiple Instruction streams over a Single Data stream)
Multiple data	SIMD (Single Instruction stream over Multiple Data streams)	MIMD (Multiple Instruction streams over Multiple Data stream)





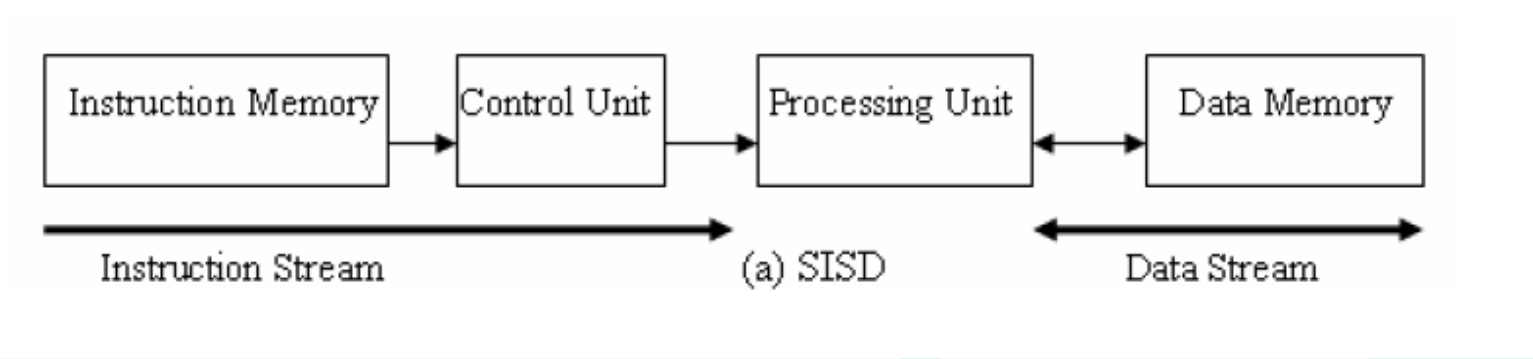
Which one is closest to GPU?





SISD

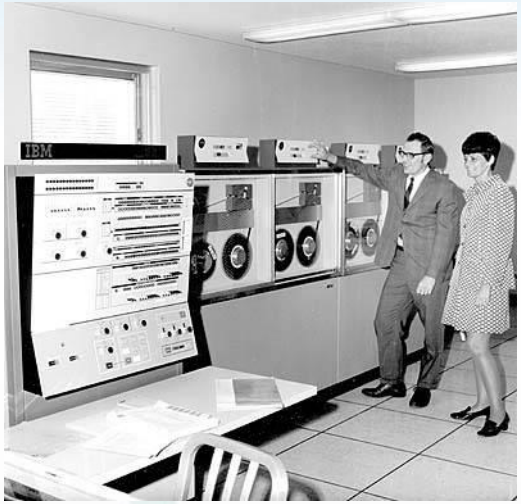
- At one time, one instruction operates on one data
- Traditional sequential architecture



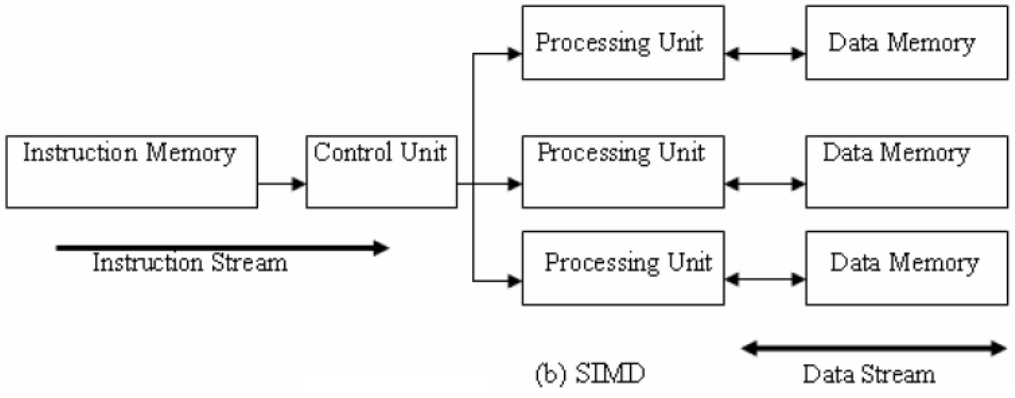
  - PC (currently manufactured PCs have multiple processors) or old mainframes

SIMD

- At one time, one instruction operates on many data
  - Data parallel architecture
  - Vector architecture has similar characteristics, but achieve the parallelism with pipelining.
- Array processors
  - Vector supercomputers of the early 1970s
    - 美國利沃(握)莫國家實驗室的CDC Star-100 (100 MFLOPS)
- Batch-pipeline systems
  - GPU (NVidia, AMD, Intel..)

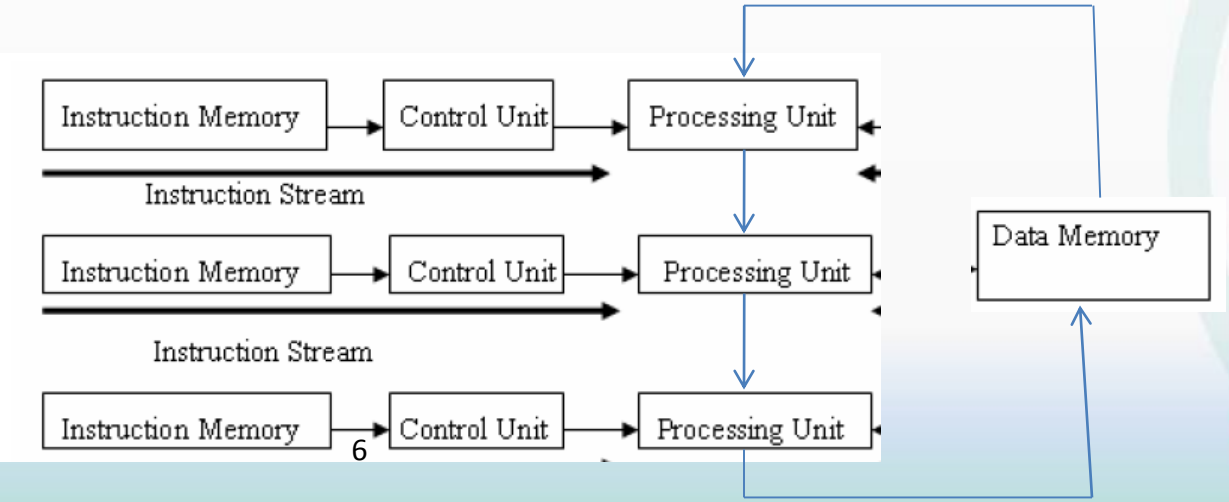


1974: CDC STAR-100 A Cray-less CDC delivered the STAR-100 to its first commercial user, General Motors



MISD

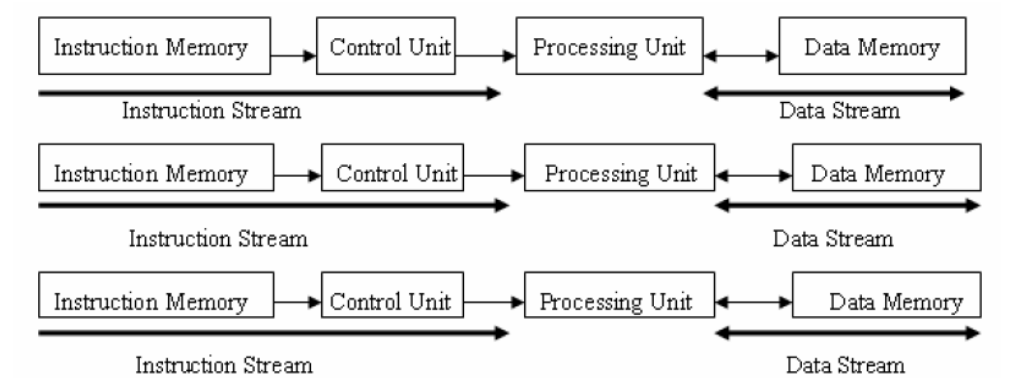
- At one time, many instruction operates on one data
  - Not commonly seen.
- Systolic array (**心藏壓縮式陣列**) is one example of an MISD architecture.
  - Systolic array是一个具有同步性和高效并行性的计算网络，使用在VLSI上。
    - Contains 128 processors shared into 32 full custom VLSI chips. One chip houses 4 processors, and one processor performs進行每秒10百萬矩陣細胞
    - Samba 收縮加速裝置應用於生物學上



Samba: Systolic Accelerator for Molecular Biological Applications

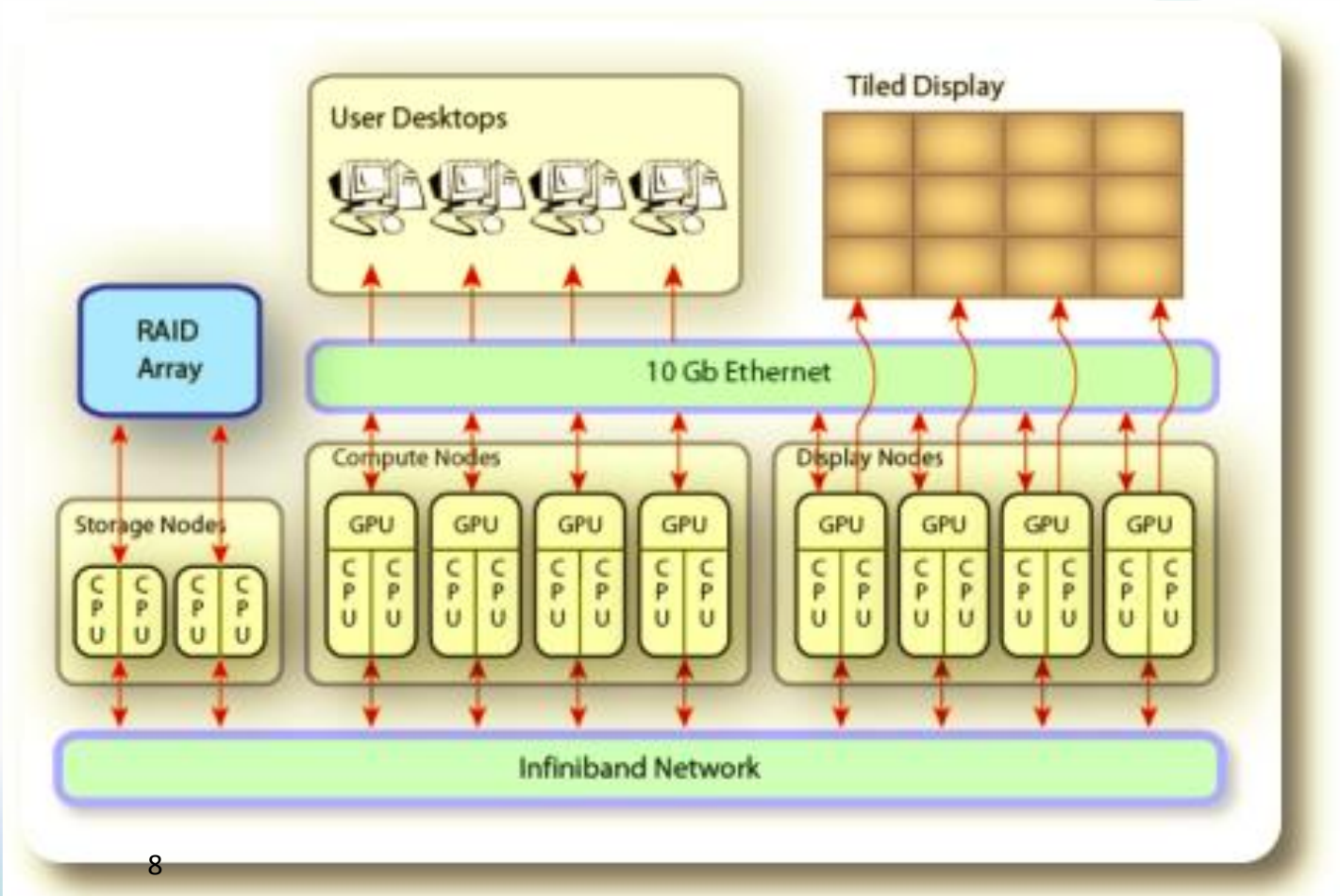
MIMD

- Multiple instruction streams operating on multiple data streams
- Machines using MIMD have a number of processors that function asynchronously and independently
- At any time, different processors may be executing different instructions on different pieces of data
  - Shared-memory: Symmetric Multiprocessors (SMP)
  - Distributed-memory: MPP (massively parallel processors) and CoC (Cluster of Computers)



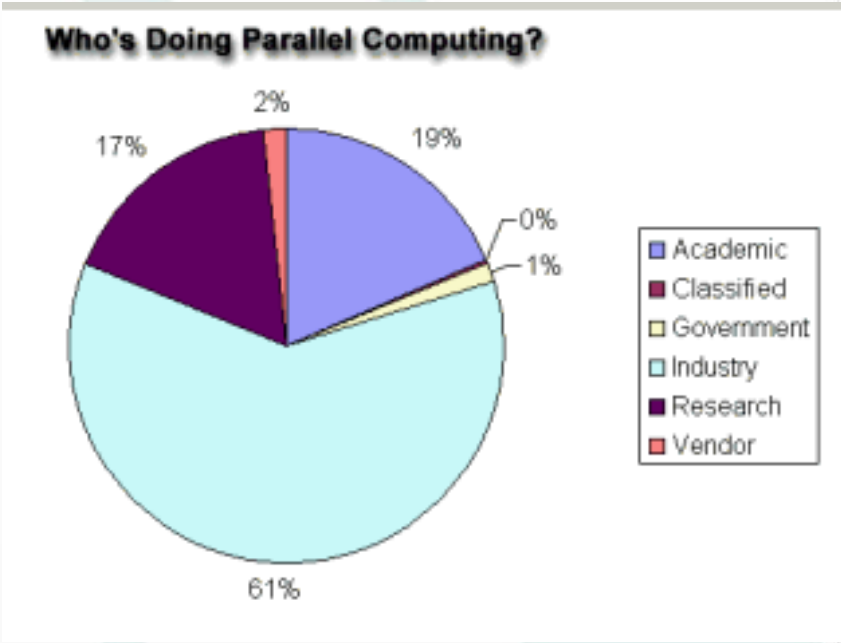
Maryland（馬里蘭）CPU/GPU Cluster Infrastructure

馬里蘭 CPU-GPU集群是一種獨特的計算基礎設施， 利用協同連接叢集 CPU，圖形處理器，顯示器和存儲器。



MIMD

- Top500 supercomputers provides statistics on parallel computing users.

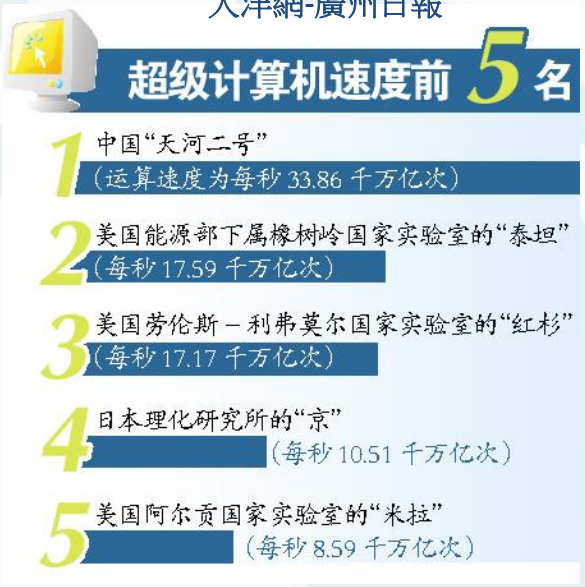


- The industry trends
  - Faster networks
    - GigE, 10GigE, InfiniBand, etc.
  - Distributed systems
    - Grid Engine
  - Multi-processor computer architectures
    - Dual Core, 4-, 6-, 8-, and 12-core, GPGPU & CUDA/OpenCL, etc.
  - Clearly show that parallelism is the future of computing.

Top500 Supercomputers

- Tianhe-2: #1 in latest Top500
  - June 2013
  - 16,000 computer nodes
    - Intel Xeon E5-2692v2 (12-core CPU)
    - Intel Xeon Phi 31S1P (GPUs)
  - A total of 3,120,000 cores
  - Speed: 33.86 peta FLOPS
    - Beat out Titan (17.59 peta FLOPS)
      - AMD opteron 6274 16-core CPU
      - Nvidia Tesla K20X GPUs
  - Cost: 390 million USD
  - Kylin OS
    - SLURM (Simple Linux Utility for Resource Management)

大洋網-廣州日報



From the June 2013 Top500 Supercomputers

- From the top 5: 3 are using GPUs
- A total of 54 systems on the list are using GPU technology
- 39 of these use NVIDIA chips, 12 use Intel Xeon Phi processors, and 3 use ATI (AMD) Radeon

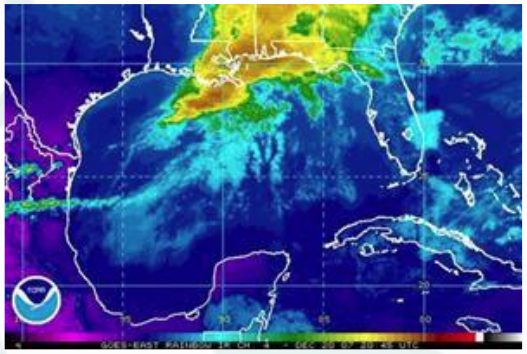
When is GPUs appropriate SIMD ?

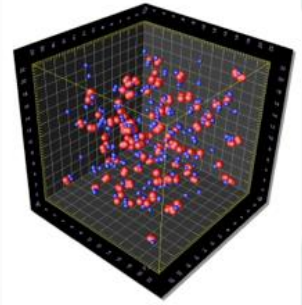
- Applications
  - Traditional GPU Applications: Gaming, image processing
    - i.e., manipulating image pixels, oftentimes the same operation on each pixel
  - Scientific and Engineering Problems: physical modeling, matrix algebra, sorting, etc.
- Data parallel algorithms:
  - Large data arrays
  - Single Instruction, Multiple Data (SIMD) parallelism
  - Floating point computations

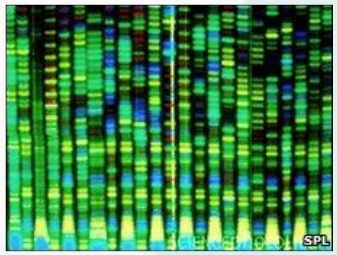
	Single instruction	Multiple instruction
Single data	SISD (Intel Pentium 4)	MISD (Non-practical model)
Multiple data	SIMD (GPU)	MIMD (Cluster of computers)

Scientific and Engineering Problems

- Simulations of physical phenomena
  - Weather forecasting
  - Earthquake forecasting
  - Galaxy formation
  - Oil reservoir management
  - Molecular dynamics
- Data Mining: Finding needles of critical information in a haystack of data
  - Bioinformatics
  - Signal processing
  - Detecting storms that might turn into hurricanes
- Visualization: turning a vast sea of data into pictures that scientists can understand
- At its most basic level, all of these problems involve many, many floating point operations.







Hardware Accelerators

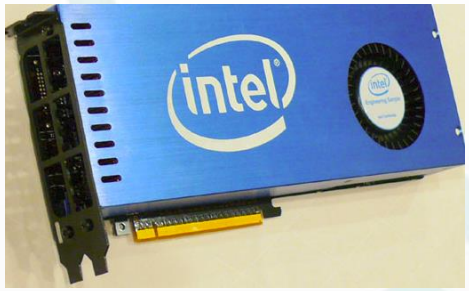
- In HPC, an accelerator is hardware component whose role is to speed up some aspect of the computing workload.
- In the olden days (1980s), supercomputers sometimes had array processors, which did vector operations on arrays.
- PCs sometimes had floating point accelerators: little chips that did the floating point calculations in hardware rather than software.

To Accelerate Or Not To Accelerate

- Pro
  - They make your code run faster
- Cons
  - They’re expensive
  - They’re hard to program
  - Your code may not be cross-platform

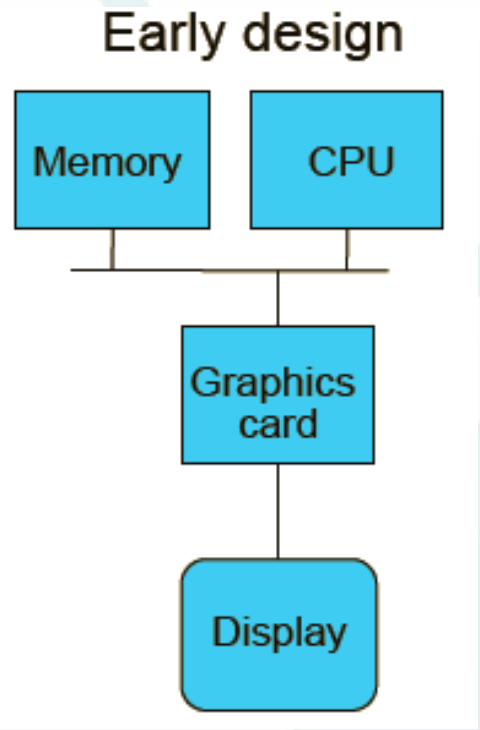
Why GPU for HPC?

- Graphics Processing Units (GPUs) were originally designed to **accelerate graphics tasks like image rendering**.
- They became very popular with **video gamers**, because they've produced better and better images, and lightning fast.
- And, prices have been extremely good, ranging from three figures at the low end to four figures at the high end.
- Chips are expensive to design (hundreds of millions of $$$), expensive to build the factory for (billions of $$$), but cheap to produce.
- For example, in 2006 – 2007, GPUs sold at a rate of about 80 million cards per year, generating about $20 billion per year in revenue.
- This means that the GPU companies have been able to recoup the huge fixed costs
- Remember: GPUs mostly do stuff like rendering images. This is done through mostly floating point arithmetic – the same stuff people use supercomputing for!



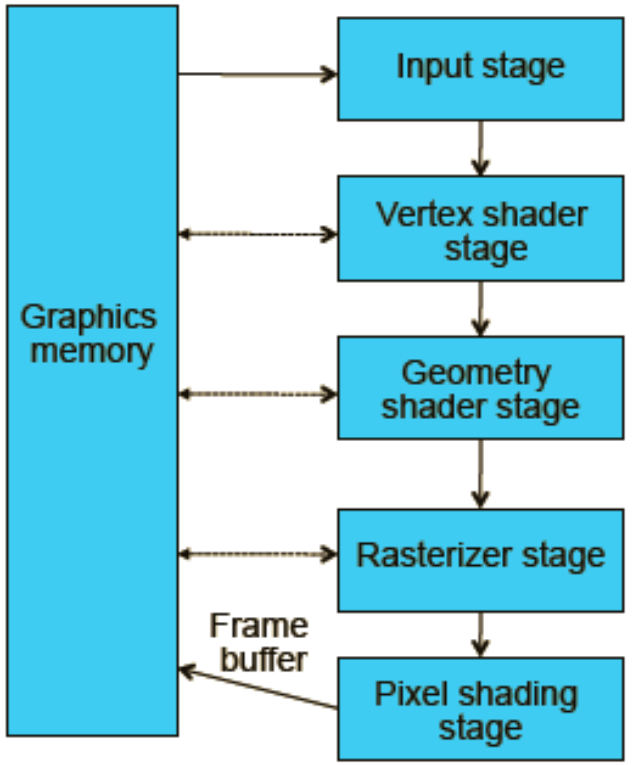
What are GPUs?

- GPUs have developed from graphics cards into a platform for high performance computing (HPC)
  - perhaps the most important development in HPC for many years.
- Co-processors
  - very old idea that appeared in 1970s and 1980s with floating point co-processors attached to microprocessors that did not then have floating point capability.
- These co-processors simply executed floating point instructions that were fetched from memory.
- Around same time, interest to provide hardwar support for displays, especially with increasing use of graphics and PC games.
- Led to graphics processing units (GPUs) attached to CPU to create video display



Modern GPU Design

- By late 1990's, graphics chips needed to support 3-D graphics, especially for games and graphics APIs such as DirectX and OpenGL.
- Graphics chips generally had a pipeline structure with individual stages performing specialized operations, finally leading to loading frame buffer for display.
- Individual stages may have access to graphics memory for storing intermediate computed data.

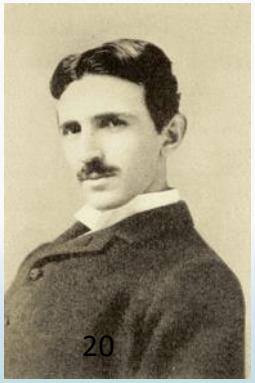


General Purpose GPU (GPGPU) Designs

- High performance pipelines call for high-speed (IEEE) floating point operations
- Known as GPGPU (General-purpose computing on graphics processing units) - Difficult to do with specialized graphics pipelines, but possible.)
- By mid 2000's, recognized that individual stages of graphics pipeline could be implemented by a more general purpose processor core (although with a data-parallel paradigm)
- 2006 -- First GPU for general high performance computing as well as graphics processing, NVIDIA GT 80 chip/GeForce 8800 card.
- Unified processors that could perform vertex, geometry, pixel, and **general computing operations**
- Could now write programs in C rather than graphics APIs.
- Single-instruction multiple thread (SIMT) programming model

NVIDIA Tesla Platform

- NVIDIA Tesla series was their first platform for the high performance computing market
- Named for Nikola Tesla, a pioneering mechanical and electrical engineer and inventor.



GTX 480

C2070

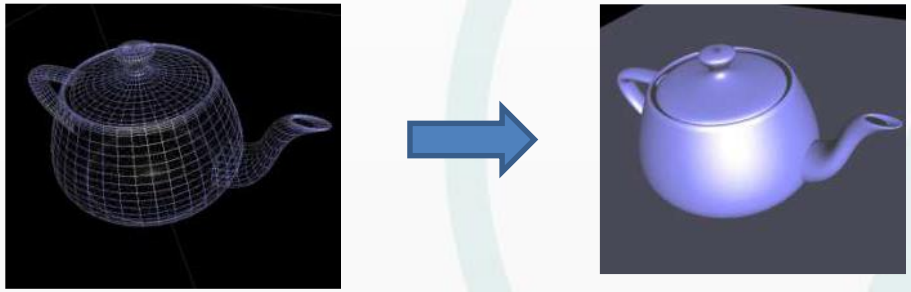
NVIDIA developed a more general purpose GPU

- Can programming it like a regular processor
- Must **explicitly** declare the data parallel parts of the workload
  - Shader processors → fully programming processors with instruction memory, cache, sequencing logic
  - Memory load/store instructions with random byte addressing capability
  - Parallel programming model primitives; threads, barrier synchronization, atomic operations

NVIDIA創始人兼執行長黃仁勳

A Little Bit of Vocabulary

- Rendering: the process of generating an image from a model
- Vertex: the corner of a polygon (usually that polygon is a triangle)
- Pixel: smallest addressable screen element



Graphics Pipeline Elements

1. A scene description: vertices, triangles, colors, lighting
2. Transformations that map the scene to a camera viewpoint
3. “Effects”: texturing, shadow mapping, lighting calculations
4. Rasterizing: converting geometry into pixels
5. Pixel processing: depth tests, stencil tests, and other per-pixel operations.

From Numbers to Screen

```
0.748952 -0.764952 -0.210132,
0.872246 -0.600862 -0.210132,
1.00016 -0.369696 -0.210132,
1.00939 -0.084084 -0.210132,
1.14496 0.324436 -0.210132,
1.15747 0.601712 -0.210132,
1.08016 0.793529 -0.210132,
0.98164 0.972032 -0.210132,
0.808263 0.929016 -0.210132,
0.442563 0.985585 -0.210132,
0.221794 1.00159 -0.210132,
0 1.0053 -0.210132,
-0.221794 1.00159 -0.210132,
-0.442563 0.985585 -0.210132,
-0.808263 0.929016 -0.210132,
```

URL

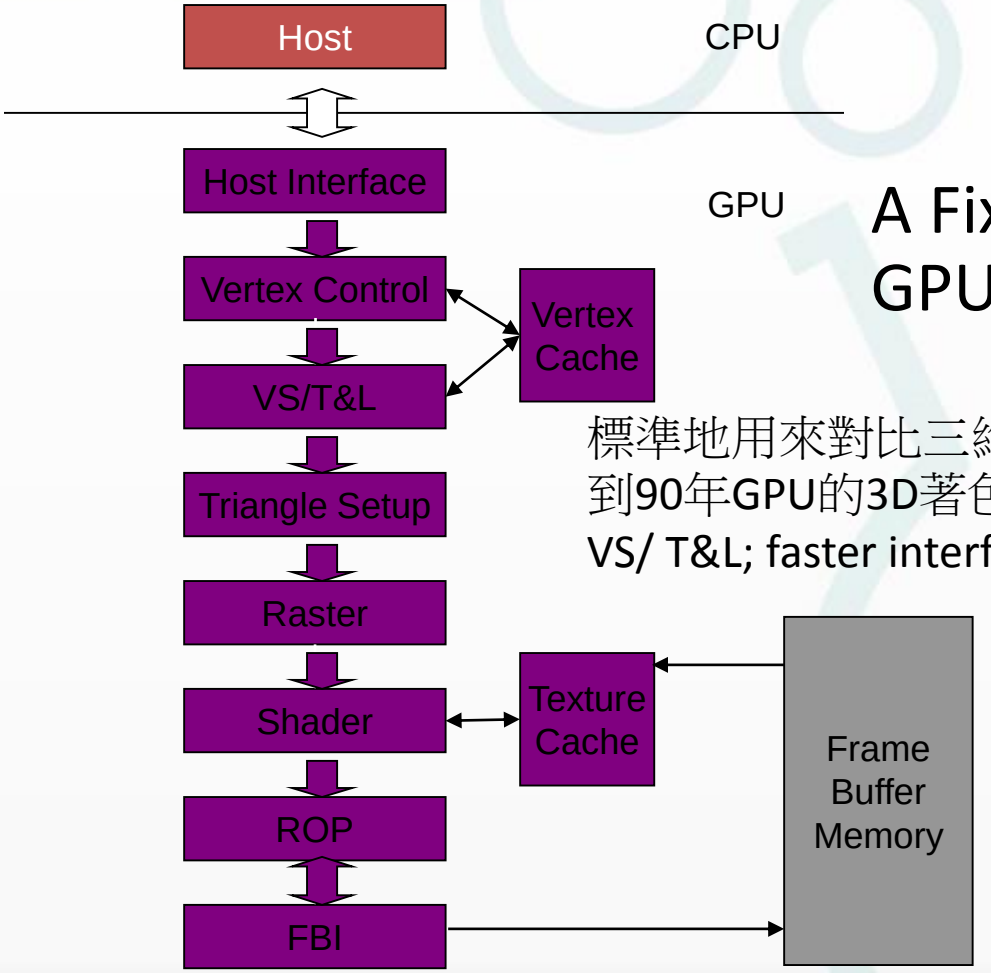
Before GPUs

- Vertices to pixels:
  - Transformations done on CPU
  - Compute each pixel "by hand", in series…**slow!**

- Example: 1 million triangles * 100 pixels per triangle * 10 lights * 4 cycles per light computation = **4 billion cycles**

早期只能跑cpu，需要花費數十倍或數百倍
時間才能跑完

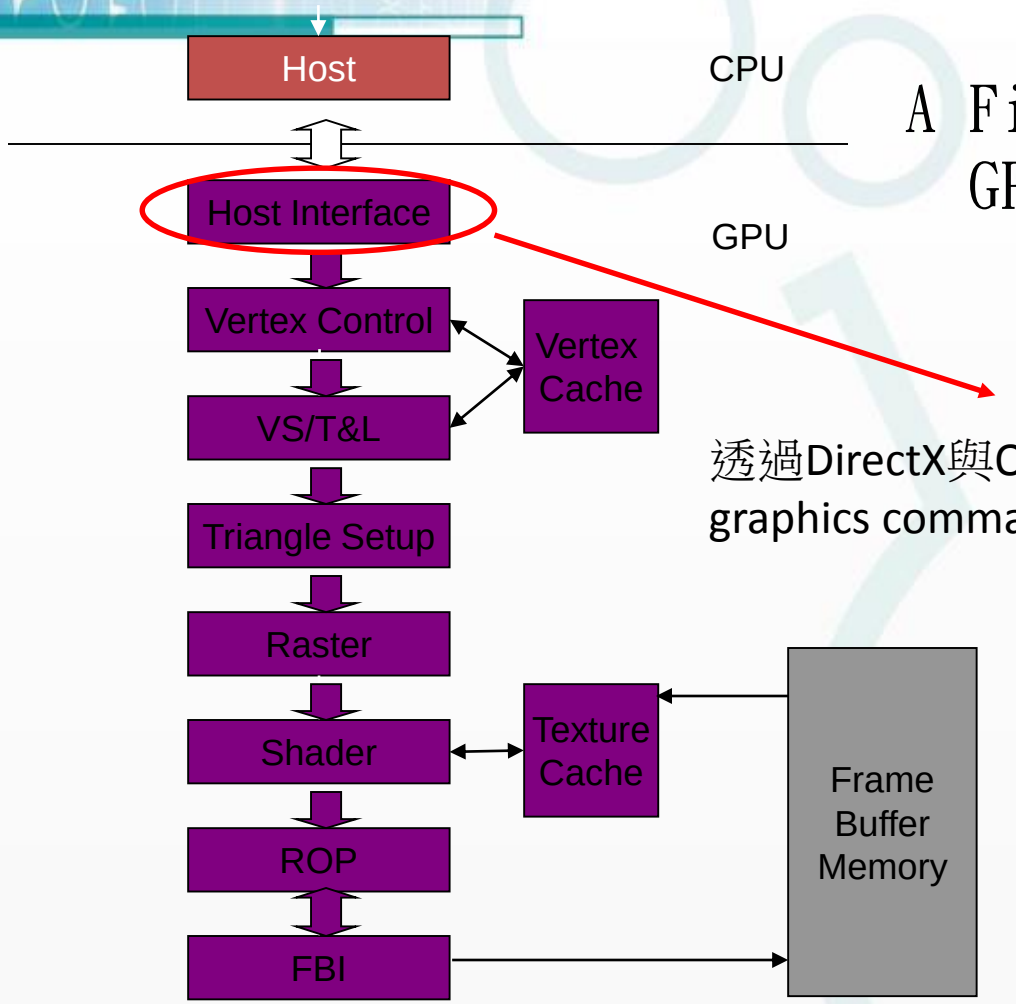
Early GPUs: Early 80s to Late 90s



A Fixed Function GPU Pipeline

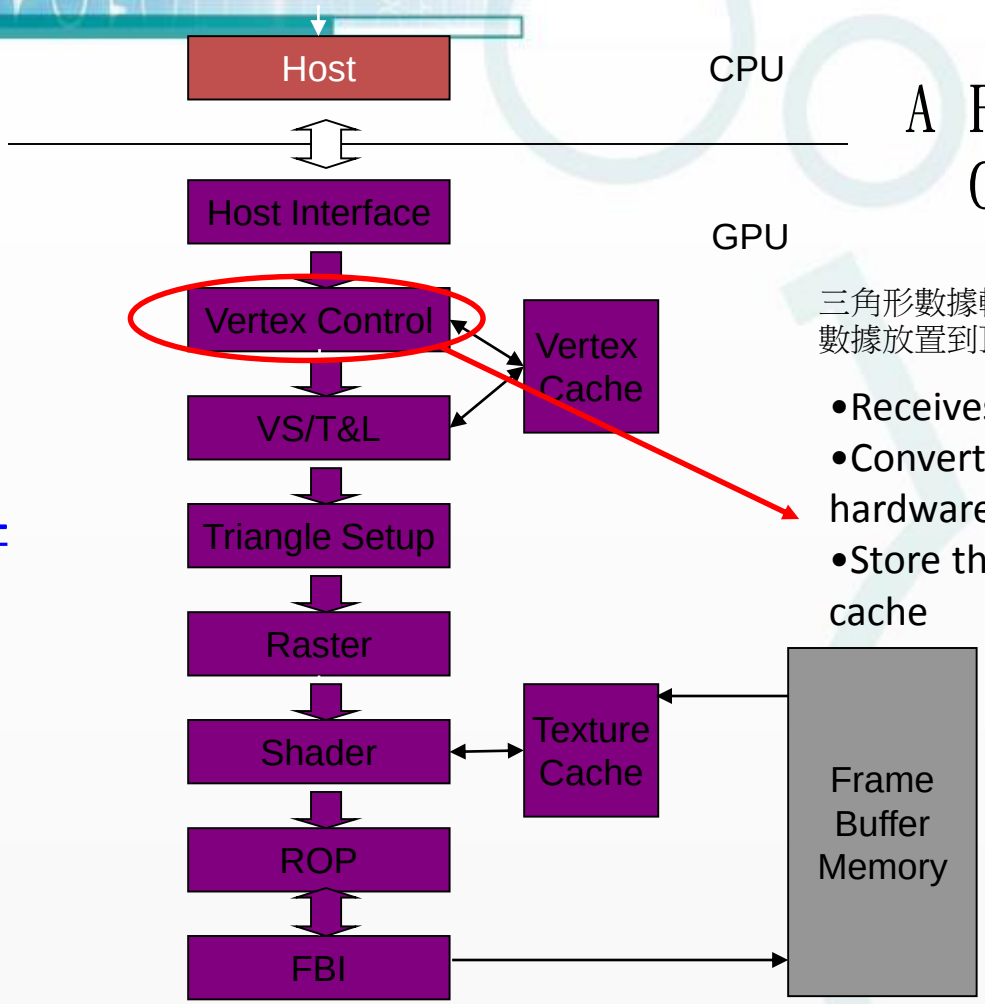
標準地用來對比三維圖形API，也是更早80年到90年GPU的3D著色器圖形架構，加入raster, VS/ T&L; faster interface來加速圖形計算。

A Fixed Function GPU Pipeline



透過DirectX與OpenGL API Receives graphics commands and data from CPU

A Fixed Function GPU Pipeline

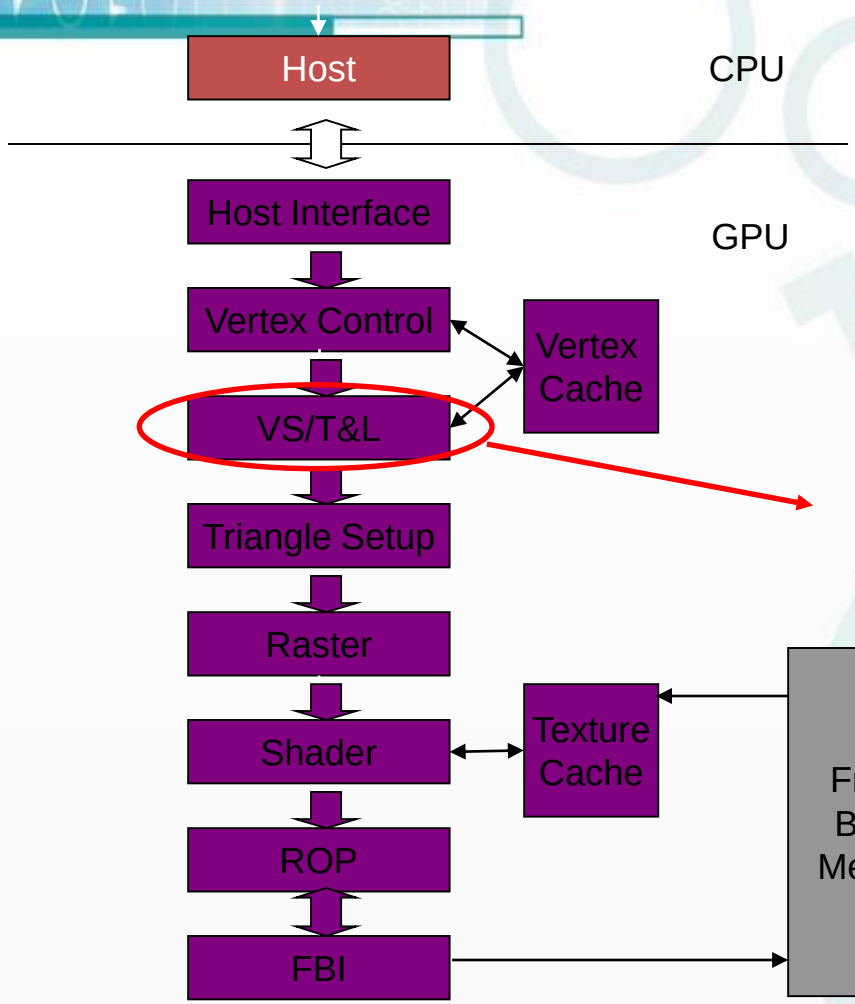


URL

三角形數據轉換成形成硬體了解和準備的數據放置到頂點緩存。

- Receives triangle data
- Converts them into a form that hardware understands
- Store the prepared data in vertex cache

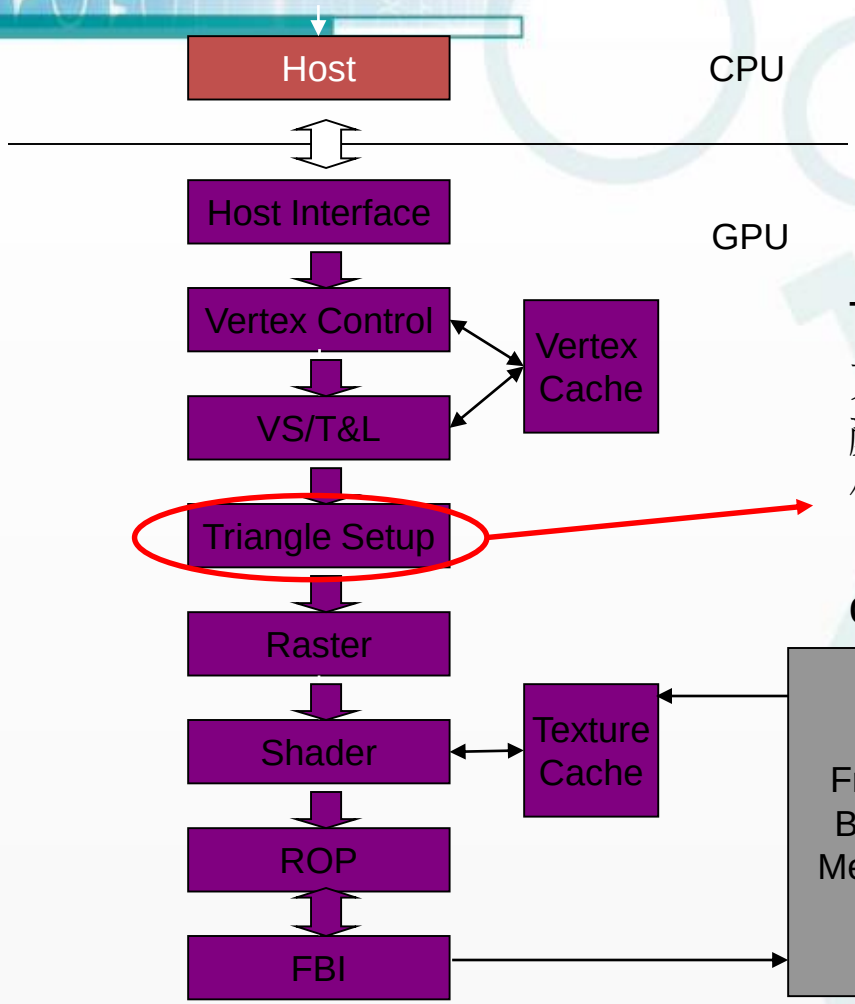
A Fixed Function GPU Pipeline



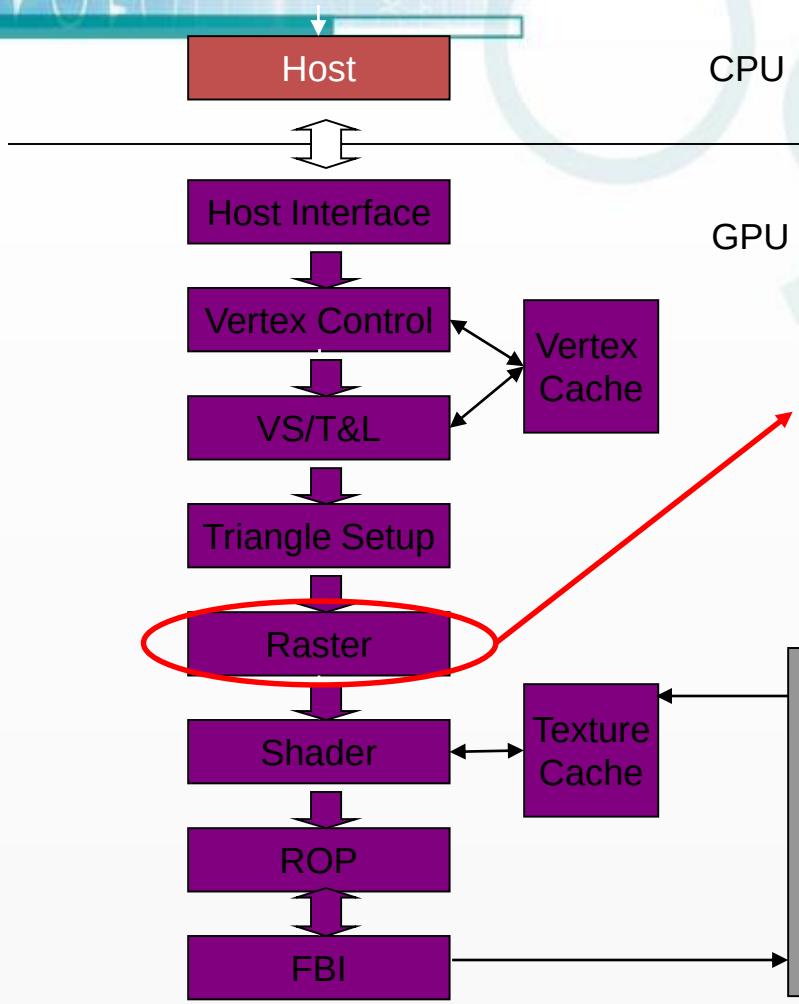
- Vertex Shading Transform and Lighting (VS/T&L)
- Assigns per-vertex value

轉換頂點，並指定每個頂點的值（例如，顏色，法線，紋理坐標，切線 colors, normals, texture coordinates, tangents）。

A Fixed Function GPU Pipeline



Triangle Setup 進一步建立邊緣方程式由三角形觸動像素內插顏色interpolate colors和其他每個頂點的數據per-vertex data（如材質貼圖坐標 texture coordinates）。

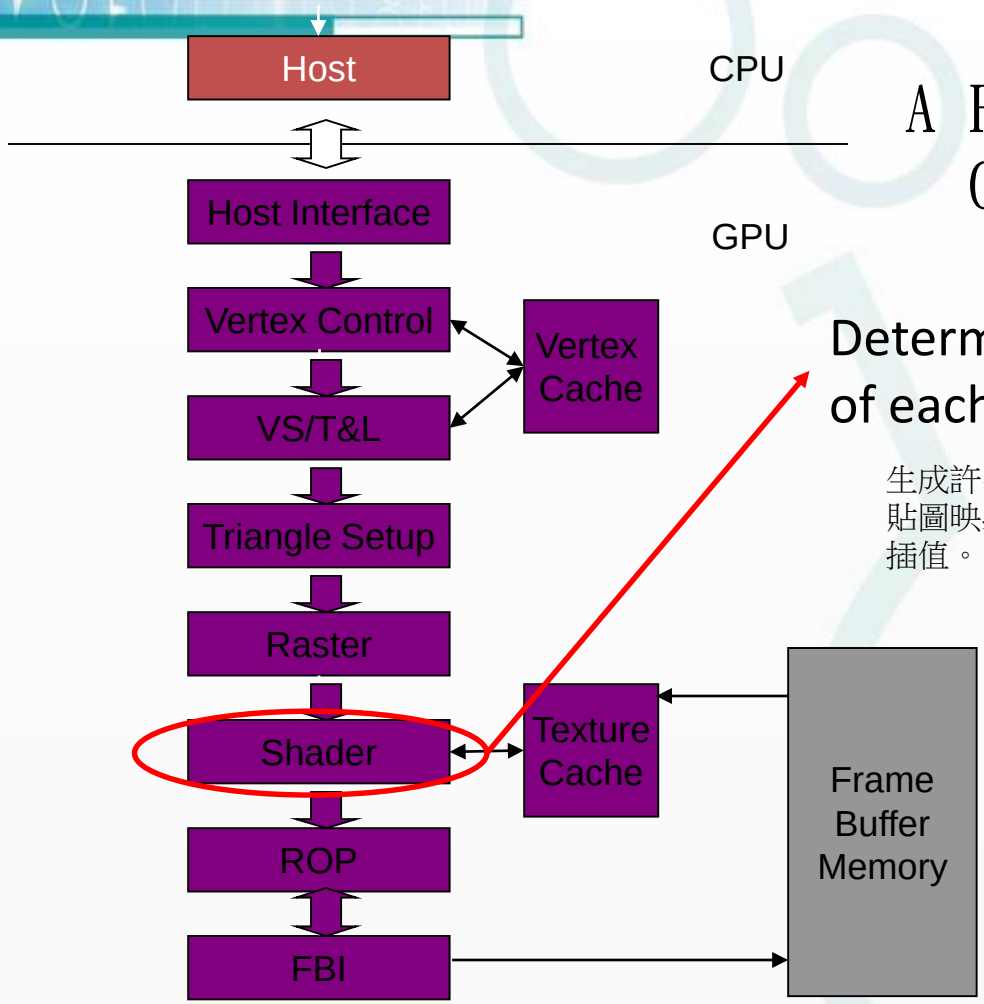


A Fixed Function GPU Pipeline

- Determines which pixel falls in which triangle
- For each pixel, interpolate per-pixel values (插值每個頂點)

包括顏色，位置，質地位置(texture position)將被陰影（塗）上的像素值

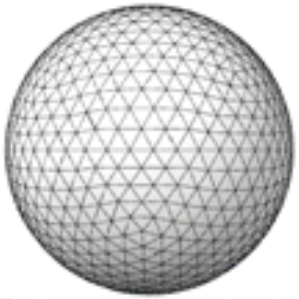
A Fixed Function GPU Pipeline



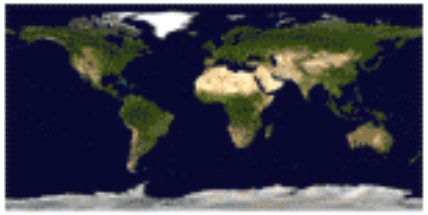
Determines the final color of each pixel

生成許多技術的綜合效應：頂點顏色，貼圖映射，逐像素光照數學，反射和更插值。

Texture Mapping Example

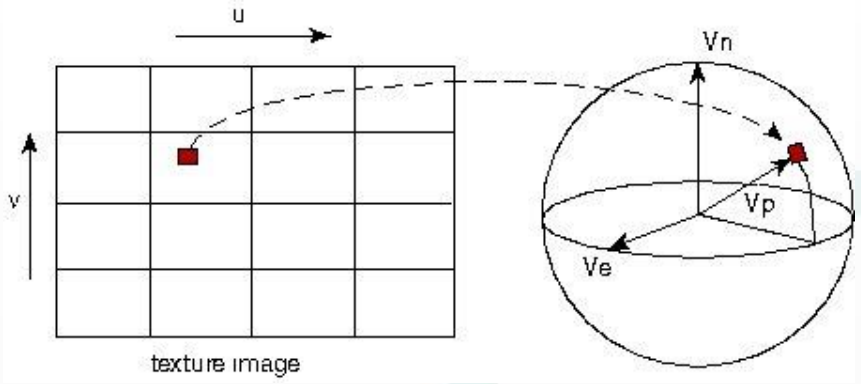


Sphere with no texture



Texture image

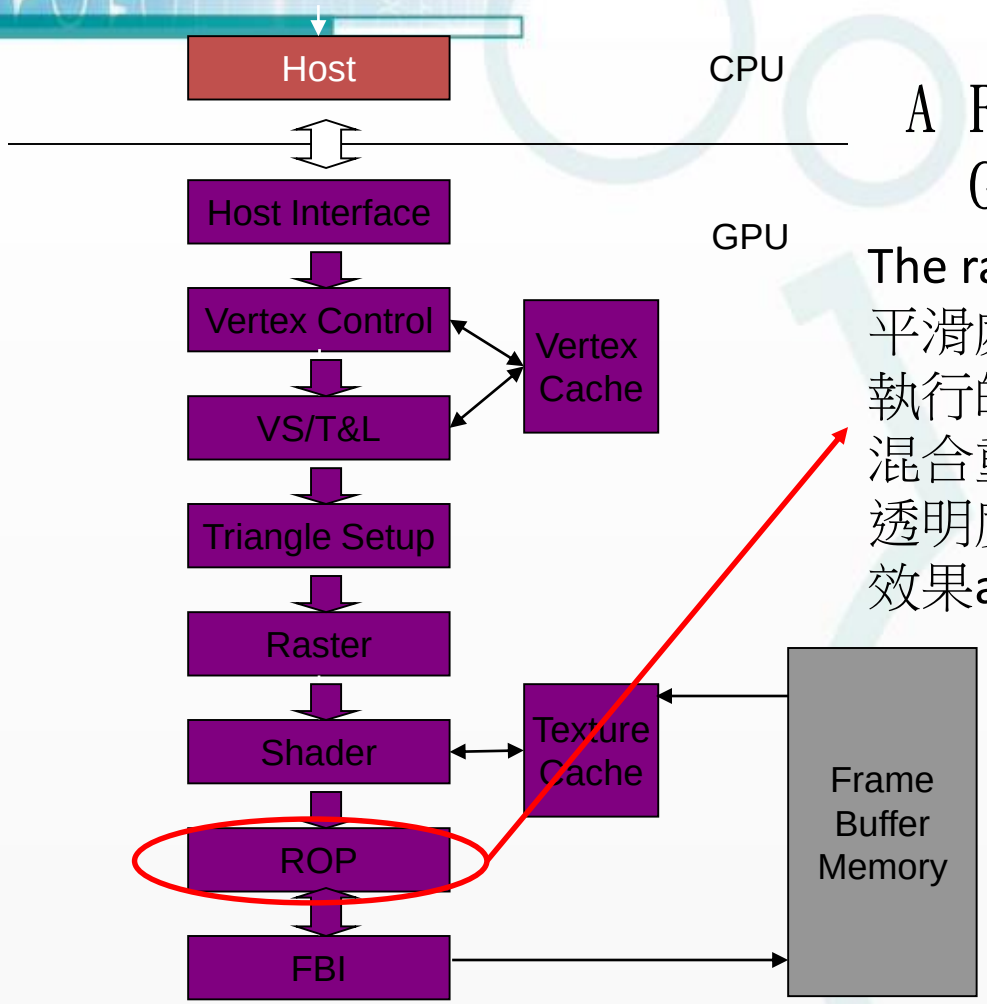
Sphere with texture



Texture Mapping 材質貼圖，著色器階段的功能之一。它示出了其中的世界地圖的紋理被映射到一個球體對象的示例。

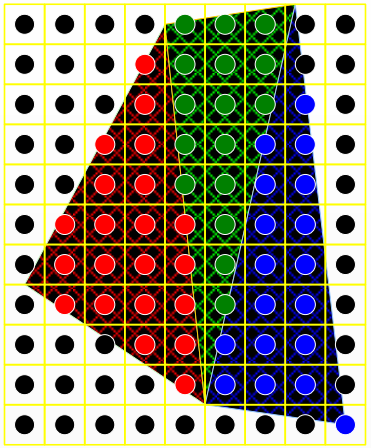
儘管著色器階段必須執行只有少數的坐標變換的計算，以確定貼圖點，這將在三角形中的一個，它描述球體對象被畫上一個點的準確坐標，包括在圖像中的像素的絕對數量需要著色器階段來執行非常大量的坐標變換對每個幀。

A Fixed Function GPU Pipeline

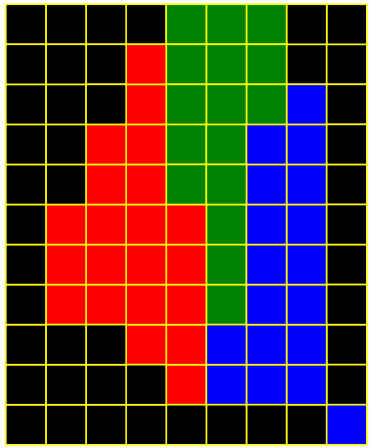


The raster operation (ROP 平滑處理運算):
執行的彩色平滑處理運算該混合重疊/相鄰對象的顏色，透明度transparency和抗鋸齒效果antialiasing effects

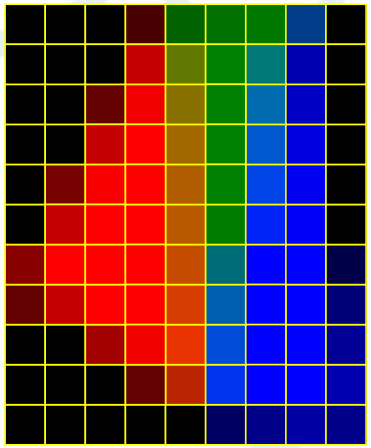
ROP(平滑處理):Anti-Aliasing（抗鋸齒）Example



Triangle Geometry



Aliased



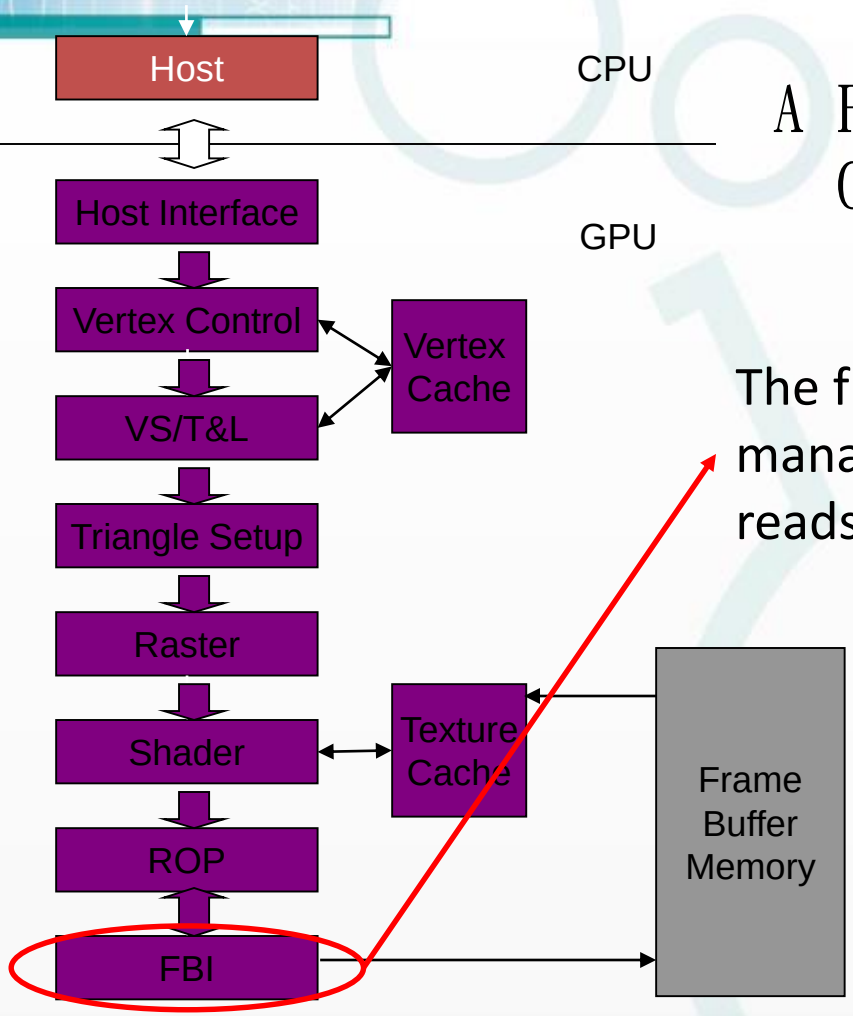
Anti-Aliased

請注意下列三個相鄰三角形與黑色背景。

- 在**鋸齒的輸出，每個像素假定的對象或背景中一個的顏色**。
- 在**有限的分辨率，使邊緣看起來歪和物體的形狀扭曲**。
- 問題是，許多像素部分中的一個對象，並部分地在另一個對象或背景。**迫使這些像素的假設中的一個對象的顏色失真引入到對象的邊緣**。
- 平滑處理運算(ROP)給出每個像素被混合，或線性組合，從所有的對象和背景部分重疊的像素的顏色。

1. No anti-aliasing
2. Anti-aliasing

A Fixed Function GPU Pipeline



The frame buffer interface manages memory reads/writes

frame buffer非常高的帶寬要求

其一是，繪圖管線通常使用提供帶寬比系統內存更高的特殊存儲器設計。
其次，frame buffer interface同時manages multiple memory channels that connect to multiple memory banks。
The combined bandwidth improvement of multiple channels and special memory structures gives the frame buffers much higher bandwidth than their contemporaneous system memories.
如此高的內存帶寬已經延續到今天，已成為現代GPU的設計特色。

Next Steps

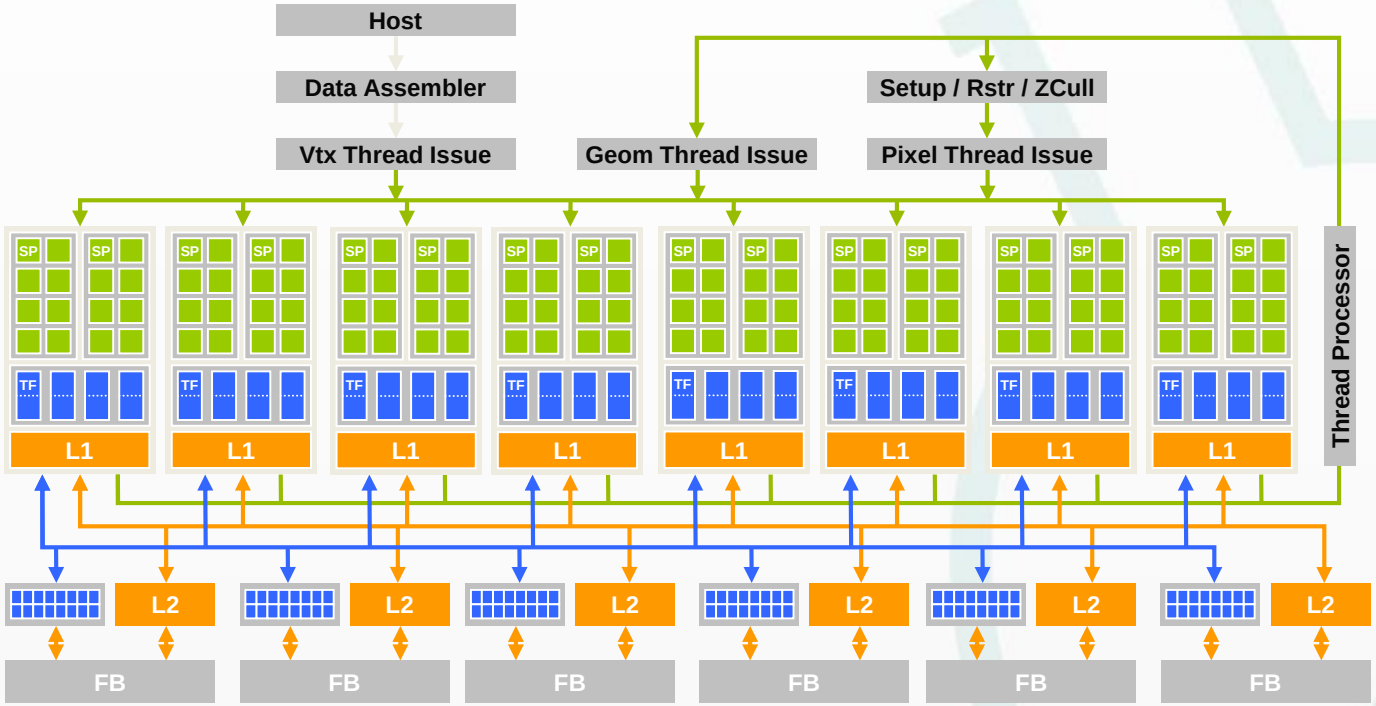
- In 2001:
  - the NVIDIA GeForce 3邁出了實現真正的通用可編程著色器的第一步。
  - 開發者使用Microsoft’s DirectX and OpenGL公開的應用程序使用VS/T&L(Vertex Shading Transform and Lighting)的內部專用指令集.
    - 例如，顏色，法線，紋理坐標，切線colors, normals, texture coordinates, tangents.
- Later:
  - General programmability extended and floating-point capability to shader stage
  - Data independence是GPU和CPU的設計假定之間的主要區別使用硬件並行性來利用此數據獨立性的機會是巨大的。

In 2006

- NVIDIA GeForce 8800 mapped separate graphics stage to a unified array of processors
  - Logical graphics pipeline visits processors three times with fixed-function graphics logic between visits
  - 負載均衡成為可能;可編程階段在不同的渲染算法存在不同的負載
    - Dynamically allocated (動態分配) from unified processors
    - 傳統顯示核心的架構分為頂點和像素著色引擎。
      - 當頂點著色引擎負荷很重時，像素著色引擎可能閒置著，反之亦然。這就造成顯示核心運算能力不被充分發揮，浪費資源。
    - DirectX 10將頂點著色、幾何著色和像素著色合併成一個渲染流程。每一個統一流處理器都能處理頂點、幾何和像素資料，不會有閒置問題，效率顯著提升。

Unified Graphics Pipeline
GeForce 8800

使用GPU來解決計算密集型問題 GeForce 8800顯卡
大問題拆成小問題同時發送數千threads解決



Exploring the use of GPUs to solve compute intensive problems
The birth of GPGPU but there are many constraints
GPUs and associated APIs were designed to process graphics data

預早偵測出不會在畫面顯示的物件，能減少不必要的運算，增加資源。

What is (Historical) GPGPU ?

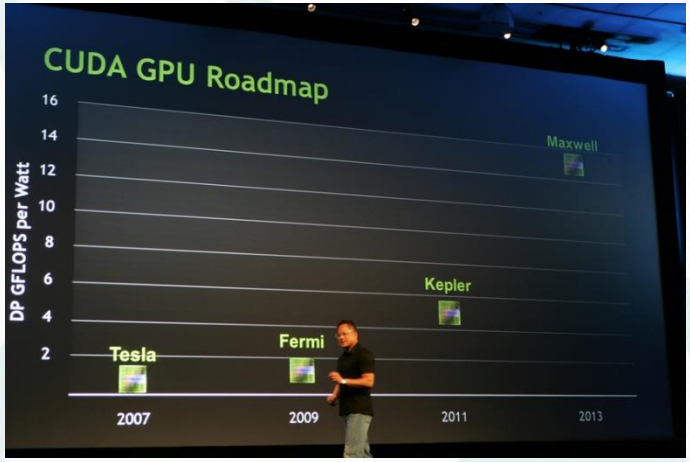
- General Purpose computation using GPU and graphics API in applications other than 3D graphics
  - GPU accelerates critical path of application
- Data parallel algorithms leverage GPU attributes
  - Large data arrays, streaming throughput
  - Fine-grain SIMD parallelism
  - Low-latency floating point (FP) computation
- Applications – see http://gpgpu.org
  - Game effects (FX) physics, image processing
  - Physical modeling, computational engineering, matrix algebra, convolution, correlation, sorting

The Birth of GPU Computing

- **Step 1**: Designing high-efficiency floating-point and integer processors.
- **Step 2**: Exploiting data parallelism by having large number of processors
- **Step 3**: Shader processors fully programmable with large instruction cache, instruction memory, and instruction control logic.
- **Step 4**: Reducing the cost of hardware by having multiple shader processors to share their cache and control logic.
- **Step 5**: Adding memory load/store instructions with random byte addressing capability
- **Step 6**: Developing CUDA C/C++ compiler, libraries, and runtime software models.

Coming: Kepler and Maxwell

- NVIDIA's 20-series is also known by the codename "Fermi費米." It runs at about 0.5 TFLOPs per GPU card (peak).
- The next generation, to be released in 2011, is codenamed "Kepler" and will be capable of something like 1.4 TFLOPs double precision per GPU card.
- After "Kepler" will come "Maxwell" in 2013, capable of something like 4 TFLOPs double precision per GPU card
- So, the increase in performance is likely to be roughly 2.5x - 3x per generation, roughly every two years



Parallel Computing on a GPU

- 8-series GPUs deliver 25 to 200+ GFLOPS on compiled parallel C applications
  - Available in laptops, desktops, and clusters
- GPU parallelism is doubling every year
- Programming model scales transparently
- Programmable in C with CUDA tools
- Multithreaded SPMD model uses application data parallelism and thread parallelism

GeForce 8800

Tesla D870

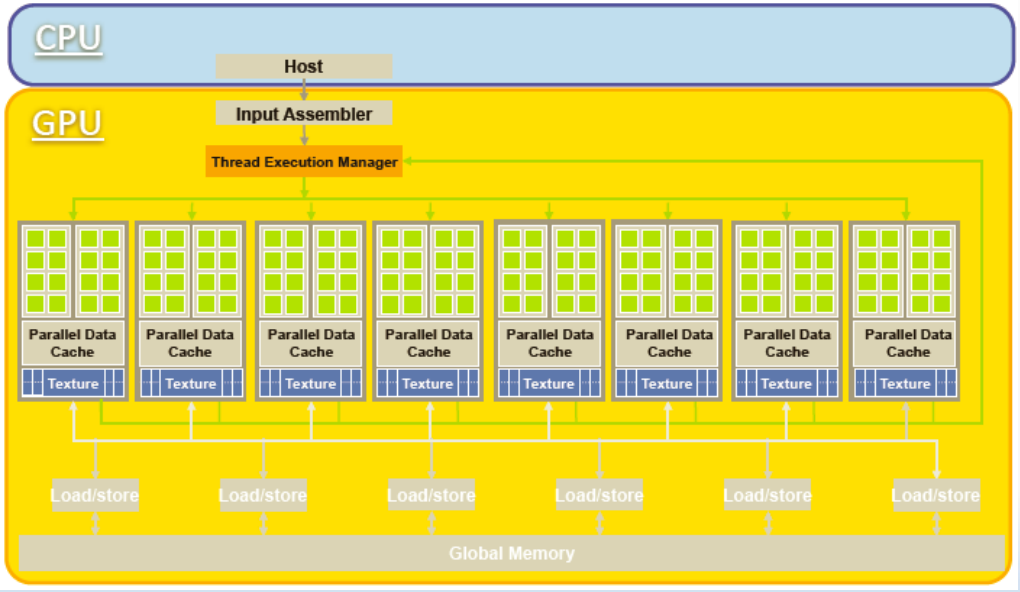
Tesla S870

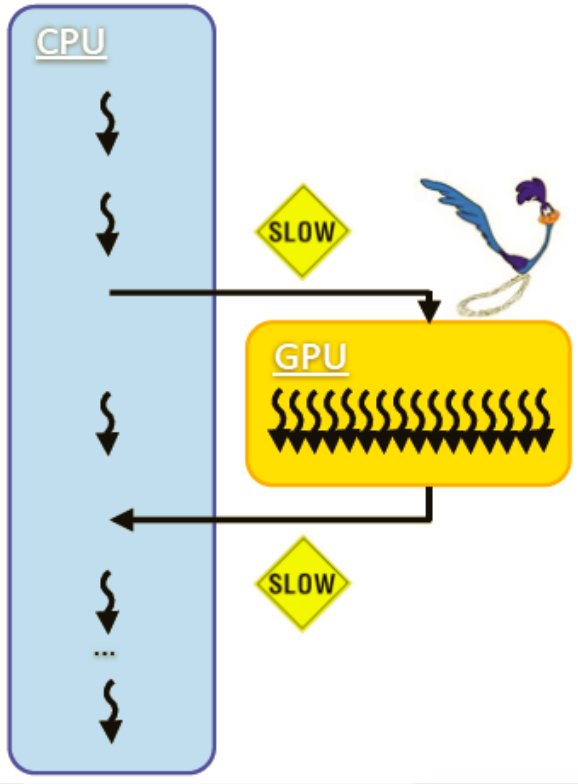
To Accelerate Or Not To Accelerate

- Pro (Advantages)
  - They make your code run faster
- Cons (Disadvantages)
  - ~~They're expensive~~
  - They're hard to program
  - ~~Your code may not be cross-platform~~

CUDA-capable GPU Hardware Architecture

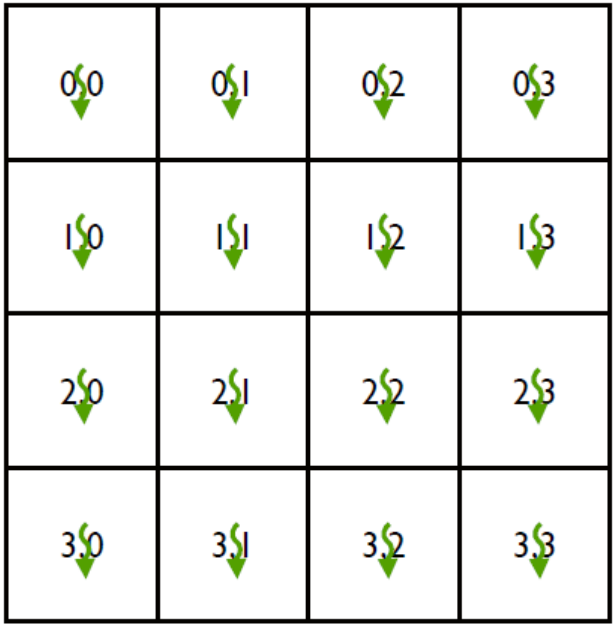
- Processors execute computing threads
- Thread execution managers issues threads
- 128 thread processors grouped into 16 streaming multiprocessors (SMs)
- Parallel Data Cache enables thread cooperation





Single Instruction, Multiple Threads (SIMT)

- A version of SIMD used in GPUs.
- GPUs use a thread model to achieve a high parallel performance and hide memory latency.
- On a GPU, 10,000s of threads are mapped on to available processors that all execute the same set of instructions (on different data addresses).



Is it hard to program on a GPU?

- In the olden days (pre-2006) - programming GPUs meant either:
  - using a graphics standard like OpenGL (which is mostly meant for rendering), or
  - getting fairly deep into the graphics rendering pipeline.
- To use a GPU to do general purpose number crunching, you had to make your number crunching pretend to be graphics.
- This is hard. Why bother?

How to Program on a GPU Today

- Proprietary programming language or extensions
  - NVIDIA: CUDA (Compute Unified Device Architecture) (C/C++)
  - AMD/ATI: StreamSDK/Brook+ (C/C++)

- OpenCL (Open Computing Language): an industry standard for doing number crunching on GPUs.

- OpenMP version 4.0 and OpenACC 1.0 include directives for accelerators.
  - Intel R Xeon PhiTM Coprocessors

We help you to understand new technologies and trends!