Lecture 11: Multi-Core and GPU

- Multi-core computers
- Multithreading
- GPUs
- General Purpose GPUs

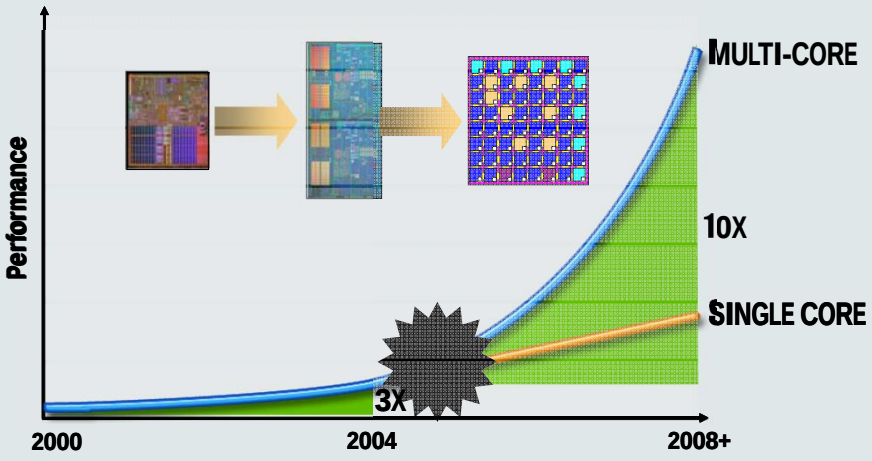
Introduction to Multi-Core System

- Integration of multiple processor cores on a single chip.
  - To provide a cheap parallel computer solution.
  - To increase the computation power to PC platform.
- Multi-core processor is a special kind of multiprocessors:
  - All processors are on the same chip — also called Chip Multiprocessor.
  - MIMD: Different cores execute different codes (threads) operating on different data.
  - A shared memory multiprocessor: all cores share the same memory via some cache organization.

Why Multi-Core?

- Limitations of single core architectures:
  - High power consumption due to high clock rates (2-3% power increase per 1% performance increase).
  - Heat generation (cooling is expensive).
  - Limited parallelism (Instruction Level Parallelism only).
  - Design time and complexity increased due to complex methods to increase ILP.
- Many new applications are multithreaded, suitable for multi-core.
  - Ex. Multimedia applications.
- General trend in computer architecture (shift towards more parallelism).
- Much faster cache coherency circuits, in a single chip.
- Smaller in physical size than SMP.

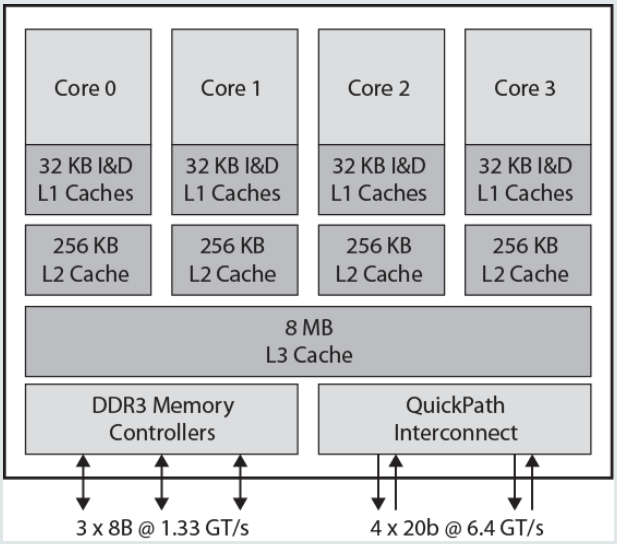
Single Core vs. Multi Core



Normalized Performance vs. Initial Intel® Pentium® 4 Processor

Source: Intel

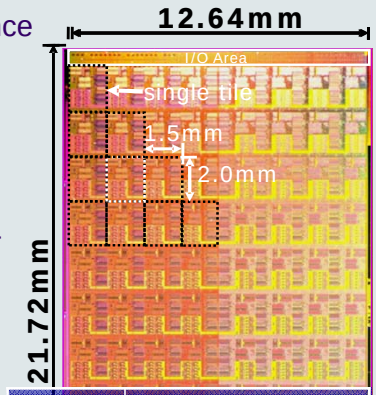
Intel Core i7



- Four identical x86 processors.
- November 2008.
- Dedicated L2, shared L3 cache.
- Cache coherent point-to-point link.
- High speed communications between processor chips.

Intel Polaris

- 80 cores with Teraflop (10^{12}) performance on a single chip (1st chip to do so).
- Mesh network-on-a-chip.
- Frequency target at 5GHz.
- Workload-aware power management:
  - Instructions to make any core sleep or wake as apps demand.
  - Chip voltage & frequency control.
- Peak of 1.01 Teraflops at 62 watts.
  - Peak power efficiency of 19.4 Gflops/Watt (almost 10X better than supercomputers).
- Short design time.



Technology	65nm, 1 poly, 8 metal (Cu)
Transistors	100 Million (full-chip) 1.2 Million (tile)
Die Area	275mm² (full-chip) 3mm² (tile)
C4 bumps #	8390

Innovations on Intel's Polaris

- Rapid design – The tiled-design approach allows designers to use smaller cores that can easily be repeated across the chip.
- Network-on-a-chip – The cores are connected in a 2D mesh network that implement message-passing. This scheme is much more scalable.
- Fine-grain power management - The individual compute engines and data routers in each core can be activated or put to sleep based on the performance required by the applications.

Lecture 11: Multi-Core and GPU

- **Multi-core computers**
- **Multithreading**
- **GPUs**
- **General Purpose GPUs**

Thread-Level Parallelism (TLP)

- This is parallelism on a more coarser scale than instruction-level parallelism (ILP).
- Instruction stream divided into smaller streams (threads) to be executed in parallel.
- Thread has its own instructions and data.
  - May be part of a parallel program or independent programs.
  - Each thread has all state (instructions, data, PC, etc.) needed to execute.
- Single-core superscalar processors cannot fully exploit TLP.
- Multi-core architectures can exploit TLP efficiently.
  - Use multiple instruction streams to improve the throughput of computers that run several programs .
- TLP are more cost-effective to exploit than ILP.

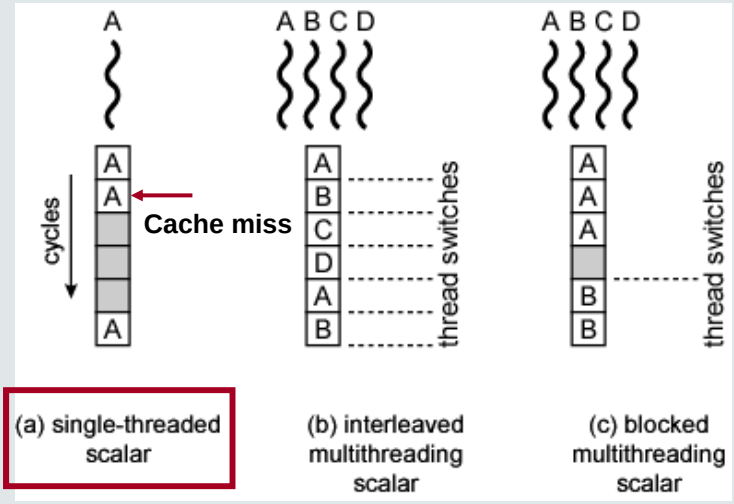
Threads vs. Processes

- Process: an instance of a program running on a computer.
  - Resource ownership.
    - Virtual address space to hold process image including program, data, stack, and attributes.
  - Execution of a process follows a path though the program.
  - Process switch — an expensive operation due to the need to save the control data and register contents.
- Thread: a dispatchable unit of work within a process.
  - Interruptible: processor can turn to another thread.
  - All threads within a process share code and data segments.
  - Thread switch is usually much less costly than process switch.

Multithreading Approaches

- Interleaved (fine-grained)
  - Processor deals with several thread contexts at a time.
  - Switching thread at each clock cycle (hardware need).
  - If a thread is blocked, it is skipped.
  - Hide latency of both short and long pipeline stalls.
- Blocked (coarse-grained)
  - Thread executed until an event causes delay (e.g., cache miss).
  - Relieves the need to have very fast thread switching.
  - No slow down for ready-to-go threads.
- Simultaneous (SMT)
  - Instructions simultaneously issued from multiple threads to execution units of superscalar processor.
- Chip multiprocessing
  - Each processor handles separate threads.

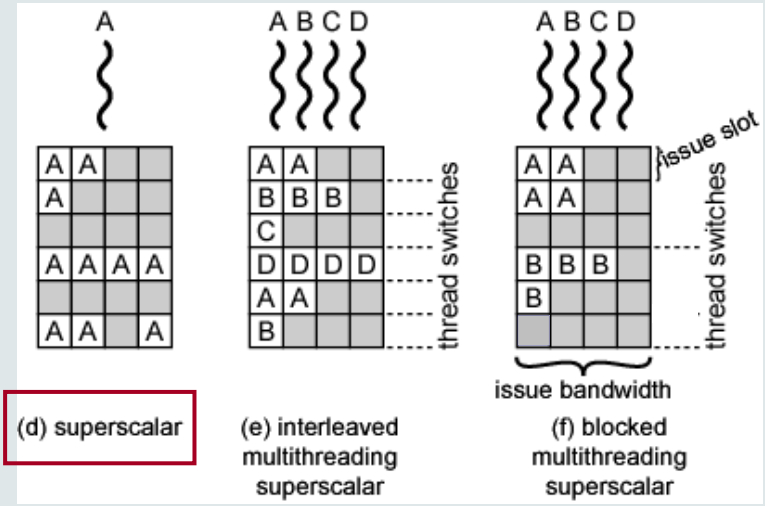
Scalar Processor Case



Scalar Processor Approaches

- Single-threaded scalar
  - Simple pipeline and no multithreading.
- Interleaved multithreaded scalar
  - Easiest multithreading to implement.
  - Switch threads at each clock cycle.
  - Pipeline stages kept close to fully occupied.
  - Hardware needs to switch thread context between cycles.
- Blocked multithreaded scalar
  - Thread executed until latency event occurs to stop the pipeline.
  - Processor switches to another thread.

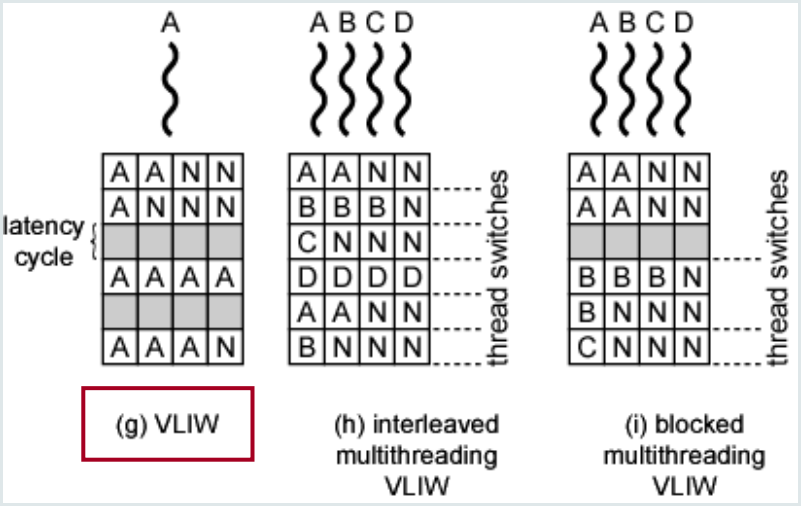
Multi-Instruction Issue



Multi-Instruction Processors

- Superscalar
  - No multithreading.
- Interleaved multithreading superscalar:
  - Each cycle, as many instructions as possible issued from single thread.
  - Delays due to thread switches eliminated.
  - Number of instructions issued in a cycle is limited by dependencies.
- Blocked multithreaded superscalar
  - Instructions from one thread are executed at a time.
  - Blocked multithreading used.

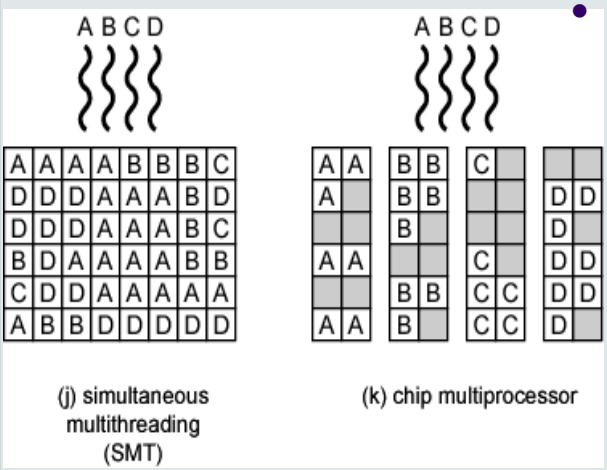
Multi-Instruction Issue (2)



Multi-Instruction Processors (2)

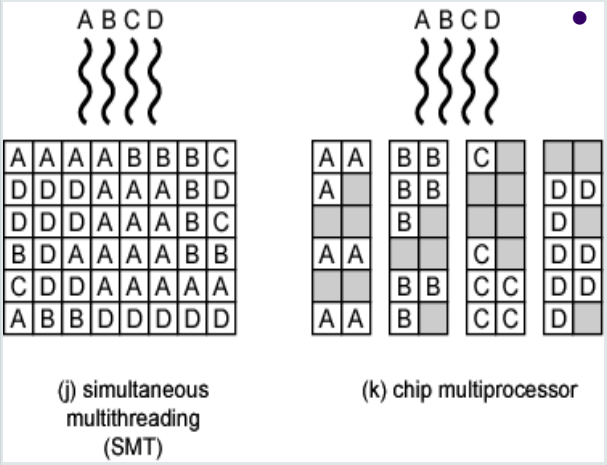
- Very long instruction word (VLIW)
  - Multiple instructions in single word.
  - Constructed by at compilation time.
  - Operations executed in parallel give in the same word.
  - May pad with no-ops.
- Interleaved multithreading VLIW
  - Similar efficiencies to interleaved multithreading on superscalar architecture.
- Blocked multithreaded VLIW
  - Similar efficiencies to blocked multithreading on superscalar architecture.

SMT and Chip Multiprocessor



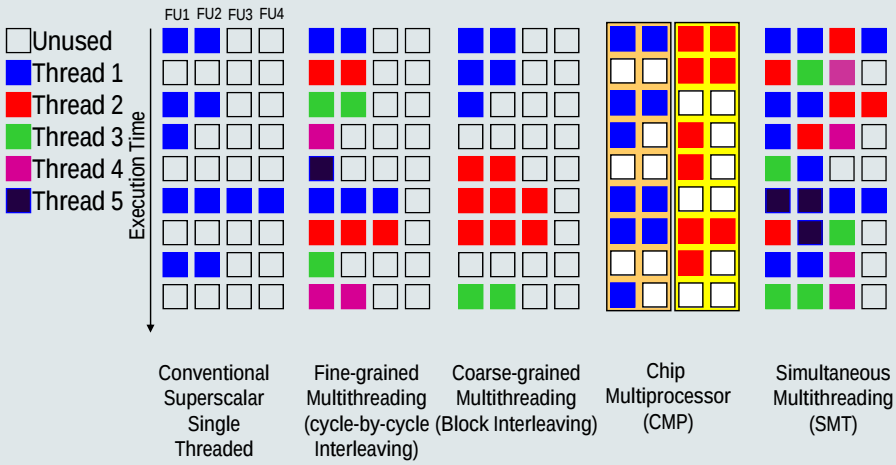
- Simultaneous multithreading
  - Issue multiple instructions at a time.
  - One thread may fill all horizontal slots.
  - Instructions from two or more threads may be issued to fill all horizontal slots.
  - With enough threads, it can issue maximum number of instructions on each cycle.
    - Best efficiency!

SMT and Chip Multiprocessor



- Chip multiprocessor
  - Multiple processors.
  - Each can be itself a superscalar processor, with multiple instruction issue.
  - Each processor is assigned a thread (or a process).
    - Control is simplified!

Multithreading Paradigms



Programming for Multi-Core

- There must be many threads or processes:
  - Multiple applications running on the same machine.
    - Multi-tasking is becoming very common.
    - OS software tends to run many threads as a part of its normal operation.
  - An application may also have multiple threads.
    - In most cases, it must be specifically written.
- OS scheduler should map the threads to different cores, in order to balance the work load or to avoid hot spots due to heat generation.

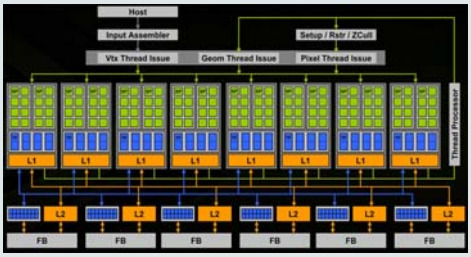
Lecture 11: Multi-Core and GPU

- **Multi-core computers**
- **Multithreading**
- **GPUs**
- **General Purpose GPUs**

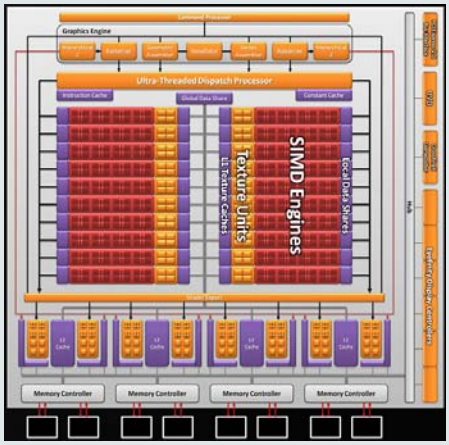
Graphics Processing Unit — Why?

- Graphics applications are:
  - Rendering of 2D or 3D images with complex optical effects;
  - Highly computation intensive;
  - Massively parallel;
  - Data stream based.
- General purpose processors (CPU) are:
  - Designed to handle huge data volumes;
  - Serial, one operation at a time;
  - Control flow based;
  - High flexible, but not very well adapted to graphics.
- GPUs are the solutions!

Example GPUs Today



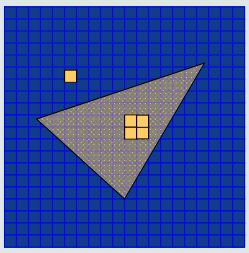
Nvidia G80



AMD 5870

GPU Design

1) Process pixels in parallel



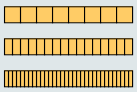
- Data-parallel:
  - 2.3M pixels per frame
    => lots of work
  - All pixels are independent
    => no synchronization
  - Lots of spatial locality
    => regular memory access
- Great speedups:
  - Limited only by the amount of hardware

GPU Design

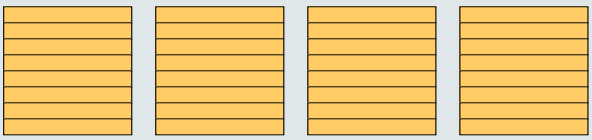
2) Focus on throughput, not latency

- Each pixel can take a long time…
  …as long as we process many at the same time.
- Great scalability
  - Lots of simple parallel processors
  - Low clock speed

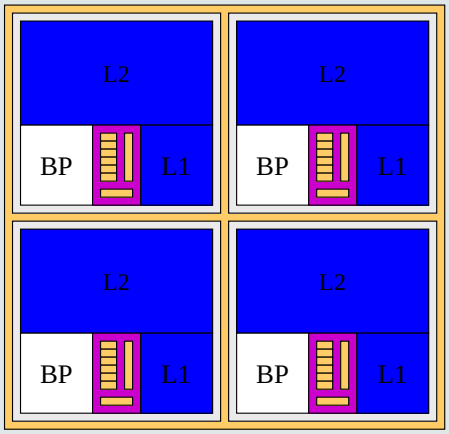
Latency-optimized (fast, serial)

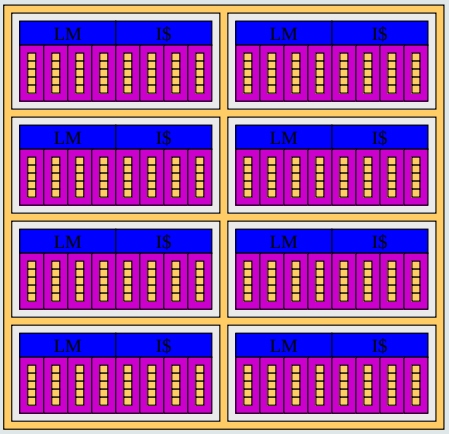


Throughput-optimized (slow, parallel)



CPU vs. GPU Philosophy





4 CPU Massive Cores: Big caches, branch predictors, out-of-order, multiple-issue, speculative execution…

~2 IPC per core, 8 IPC total @3GHz

8*8 Simple GPU Cores: No caches, in-order, single-issue, …

~ 1 IPC per core, 64 IPC total @1.5GHz

Source: David Black-Schaffer

Lecture 11: Multi-Core and GPU

- Multi-core computers
- Multithreading
- GPUs
- General Purpose GPUs

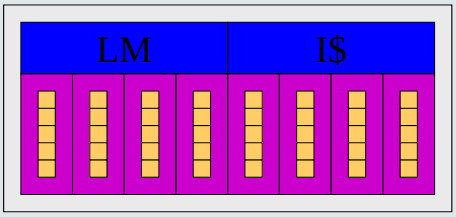
General Purpose GPUs

Question: Can we use GPUs for non-graphics tasks?

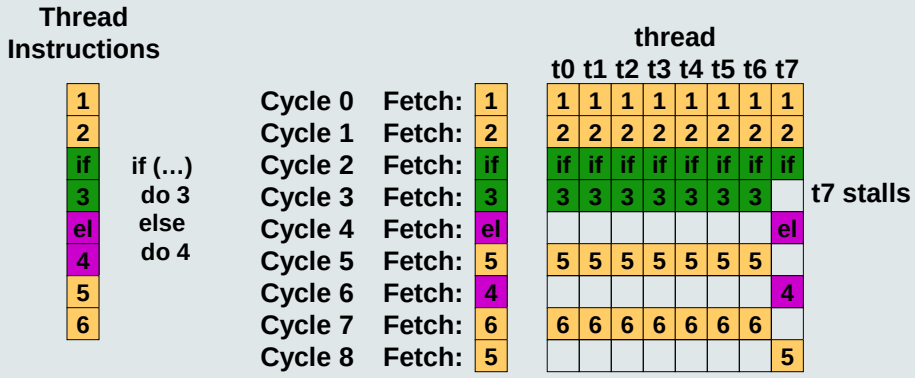
- Answer: Yes!
  - They're incredibly fast and awesome.
- Answer: Maybe
  - They're fast, but hard to program.
- Answer: Not really
  - My algorithm runs slower on the GPU than on the CPU.
- Answer: No
  - I need more precision/memory/synchronization/other.

GPU Instruction Bandwidth

- GPU compute units fetch 1 instruction per cycle, and share it with 8 processor cores.
  - SIMD (AMD GPU = 4-way; Intel's Larabee = 16-way).
- What if they don't all want the same instruction?
  - Divergent execution.



Divergent Execution



Divergent execution can dramatically hurt performance!

Source: David Black-Schaffer

CPU vs. GPU Architecture

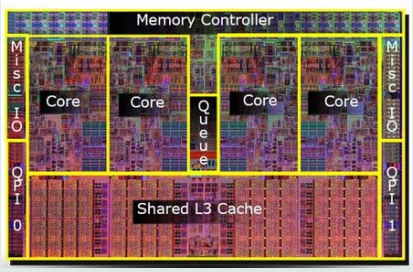
- **GPUs** are throughput-optimized
  - Each thread may take a long time, but thousands of threads
- **CPUs** are latency-optimized
  - Each thread runs as fast as possible, but only a few threads

- **GPUs** have hundreds of simple cores
- **CPUs** have a few massive cores

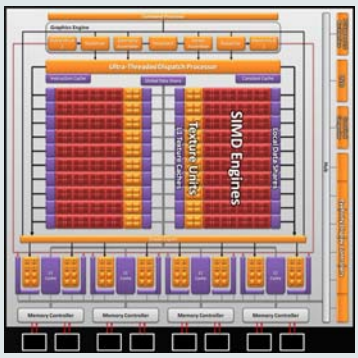
- **GPUs** excel at regular math-intensive work
  - Lots of ALUs for math, little hardware for control
- **CPUs** excel at irregular control-intensive work
  - Lots of hardware for control, few ALUs

Why Should We Care?

Intel Nehalem 4-core



AMD Radeon 5870



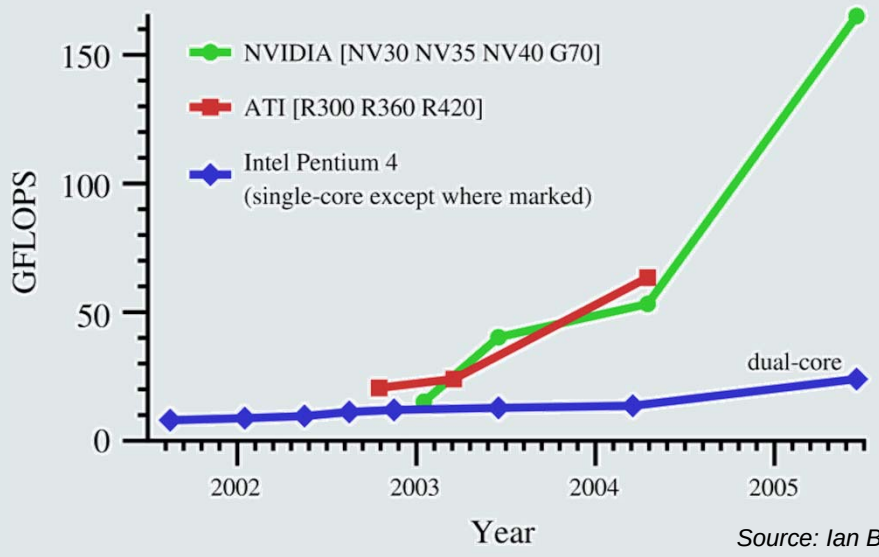
130W, 263mm^2	188W, 334mm^2
106 GFLOPs (SP)	2720 GFLOPs (SP)
Big caches (8MB)	Small caches (<1MB)
Out-of-order	Hardware thread scheduling
0.8 GFLOPs/W	14.5 GFLOPs/W

Source: David Black-Schaffer

GPU Architecture Features

- Massively Parallel, 1000s of processors (today).
- Power Efficient:
  - Fixed Function Hardware = area & power efficient.
  - Lack of speculation. More processing, less leaky cache.
- Memory Bandwidth:
  - Memory Bandwidth is limited in CPU.
  - GPU is not dependent on large caches for performance.
- Computing power = Frequency * Transistors
  - (Moore's law) ^{2}.
  - GPUs: 1.7X (pixels) to 2.3X (vertices) annual growth.
  - CPUs: 1.4X annual growth.

Computational Power



Source: Ian Buck

CUDA

- CUDA = Computer Unified Device Architecture.
- A scalable parallel programming model and a software environment for parallel computing.
  - Minimal extensions to familiar C/C++ environment;
  - Heterogeneous serial-parallel programming model.
- It enables general-purpose GPU computing
  - It also maps well to multi-core CPUs.
- It enables heterogeneous systems (i.e., CPU+GPU)
  - CPU good at serial computation, GPU at parallel.
- GPU with CUDA bring parallel computing to the masses.
  - Over 85,000,000 CUDA-capable GPUs sold.

CUDA Features

- CUDA is a powerful parallel programming model
  - Scalable — hierarchical thread execution model.
- Threads:
  - GPU threads are extremely light-weight, with little creation overhead.
  - GPU needs 1000s of threads for full efficiency.
  - Multi-core CPU needed only a few.
- GPU/CUDA address all three levels of parallelism:
  - Thread parallelism;
  - Data parallelism;
  - Task parallelism.

Summary

- All computers are now parallel computers!
- Multi-core processors represent an important new trend in computer architecture.
  - Decreased power consumption and heat generation.
  - Minimized wire lengths and interconnect latencies.
- They enable true thread-level parallelism with great energy efficiency and scalability.
- Graphics requires a lot of computation and a huge amount of bandwidth — GPUs.
- GPUs are coming to general purpose computing, since they deliver huge performance with small power.
- Massively parallel computing are becoming a commodity technology.