Overview

- Part 1: General multicore architecture
- Part 2: GPU architecture
Part 1: General Multicore architecture
Uniprocessor Performance (SPECint)


⇒ Sea change in chip design: multiple “cores” or processors per chip

- VAX: 25%/year 1978 to 1986
- RISC + x86: 52%/year 1986 to 2002
- RISC + x86: ??%/year 2002 to present
Conventional Wisdom (CW) in Computer Architecture

- Old CW: Chips reliable internally, errors at pins
- New CW: $\leq 65$ nm $\Rightarrow$ high soft & hard error rates
- Old CW: Demonstrate new ideas by building chips
- New CW: Mask costs, ECAD costs, GHz clock rates $\Rightarrow$ researchers can’t build believable prototypes
- Old CW: Innovate via compiler optimizations + architecture
- New: Takes $> 10$ years before new optimization at leading conference gets into production compilers
- Old: Hardware is hard to change, SW is flexible
- New: Hardware is flexible, SW is hard to change
Conventional Wisdom (CW) in Computer Architecture

- Old CW: Power is free, Transistors expensive
- New CW: “Power wall” Power expensive, Xtors free (Can put more on chip than can afford to turn on)
- Old: Multiplies are slow, Memory access is fast
- New: “Memory wall” Memory slow, multiplies fast (200 clocks to DRAM memory, 4 clocks for FP multiply)
- Old: Increasing Instruction Level Parallelism via compilers, innovation (Out-of-order, speculation, VLIW, …)
- New CW: “ILP wall” diminishing returns on more ILP
- New: Power Wall + Memory Wall + ILP Wall = Brick Wall
  - Old CW: Uniprocessor performance 2X / 1.5 yrs
  - New CW: Uniprocessor performance only 2X / 5 yrs?
The Memory Wall

- On die caches are both area intensive and power intensive
  - StrongArm dissipates more than 43% power in caches
  - Caches incur huge area costs

The Power Wall

\[ P = \alpha C V_{dd}^2 f + V_{dd} I_{st} + V_{dd} I_{leak} \]

- Power per transistor scales with frequency but also scales with \( V_{dd} \)
  - Lower \( V_{dd} \) can be compensated for with increased pipelining to keep throughput constant
  - Power per transistor is not same as power per area \( \rightarrow \) power density is the problem!
  - Multiple units can be run at lower frequencies to keep throughput constant, while saving power
The Current Power Trend

Improving Power/Perfomance

\[ P = \alpha CV_{dd}^2 f + V_{dd}I_{st} + V_{dd}I_{\text{leak}} \]

- Consider constant die size and decreasing core area each generation = more cores/chip
  - Effect of lowering voltage and frequency \( \rightarrow \) power reduction
  - Increasing cores/chip \( \rightarrow \) performance increase

Better power performance!
Increasing the number of cores increases the demanded memory bandwidth

What architectural techniques can meet this demand?
The Memory Wall

- Increasing the number of cores increases the demanded memory bandwidth
- What architectural techniques can meet this demand?
The Memory Wall

- On die caches are both area intensive and power intensive
  - StrongArm dissipates more than 43% power in caches
  - Caches incur huge area costs

The Power Wall

\[ P = \alpha CV_{dd}^2 f + V_{dd}I_{st} + V_{dd}I_{leak} \]

- Power per transistor scales with frequency but also scales with \( V_{dd} \)
  - Lower \( V_{dd} \) can be compensated for with increased pipelining to keep throughput constant
  - Power per transistor is not same as power per area \( \rightarrow \) power density is the problem!
  - Multiple units can be run at lower frequencies to keep throughput constant, while saving power
The ILP Wall

- Limiting phenomena for ILP extraction:
  - **Clock rate**: at the wall each increase in clock rate has a corresponding CPI increase (branches, other hazards)
  - **Instruction fetch and decode**: at the wall more instructions cannot be fetched and decoded per clock cycle
  - **Cache hit rate**: poor locality can limit ILP and it adversely affects memory bandwidth
  - **ILP in applications**: serial fraction on applications

- Reality:
  - Limit studies cap IPC at 100-400 (using ideal processor)
  - Current processors have IPC of only 2-8/thread?

The ILP Wall: Options

- **Increase granularity of parallelism**
  - Simultaneous Multi-threading to exploit TLP
    - TLP has to exist → otherwise poor utilization results
  - Coarse grain multithreading
  - Throughput computing

- **New languages/applications**
  - Data intensive computing in the enterprise
  - Media rich applications
Part2: GPU architecture
Beyond Programmable Shading: In Action

GPU Evolution - Hardware

1995
NV1
1 Million Transistors

1999
GeForce 256
22 Million Transistors

2002
GeForce 4
63 Million Transistors

2003
GeForce FX
130 Million Transistors

2004
GeForce 6
222 Million Transistors

2005
GeForce 7
302 Million Transistors

2006-2007
GeForce 8
754 Million Transistors

2008
GeForce GTX 200
1.4 Billion Transistors

Beyond Programmable Shading: In Action
GPU Architectures: Past/Present/Future

- 1995: Z-Buffered Triangles
- Riva 128: 1998: Textured Tris
- NV10: 1999: Fixed Function X-Formed Shaded Triangles
- NV20: 2001: FFX Triangles with Combiners at Pixels
- NV30: 2002: Programmable Vertex and Pixel Shaders (!)
- NV50: 2006: Unified shaders, CUDA
  - Global Illumination, Physics, Ray tracing, AI
  - future???: extrapolate trajectory
    - Trajectory == Extension + Unification
The Classic Graphics Hardware

Vertex Shader

Triangle Setup

Fragment Shader

Fragment Blender

Texture Maps

Combine vertices into triangle, convert to fragments

Texture map fragments

Z-cull

Alpha Blend

Frame-Buffer(s)

Transform

Project

GPU

programmable

configurable

fixed
Modern Graphics Hardware

- Pipelining
  - Number of stages

- Parallelism
  - Number of parallel processes

- Parallelism + pipelining
  - Number of parallel pipelines
Modern GPUs: Unified Design

Discrete Design

Shader A

Shader B

Shader C

Shader D

Unified Design

ibuffer  ibuffer  ibuffer  ibuffer

Shader Core

obuffer  obuffer  obuffer  obuffer

Vertex shaders, pixel shaders, etc. become *threads* running different programs on a flexible core
Why unify?

- **Heavy Geometry**
  - Vertex Shader
  - Workload Perf = 4
  - Heavy Pixel
  - Workload Perf = 8

Idle hardware
Why unify?

<table>
<thead>
<tr>
<th>Unified Shader</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Vertex Workload</td>
<td></td>
</tr>
<tr>
<td>Pixel Workload</td>
<td></td>
</tr>
</tbody>
</table>

Heavy Geometry  
Workload Perf = 11

Heavy Pixel  
Workload Perf = 11
Beyond Programmable Shading: In Action

GeForce 8: Modern GPU Architecture
The device is a set of multiprocessors.

Each multiprocessor is a set of 32-bit processors with a **Single Instruction Multiple Data** architecture.

At each clock cycle, a multiprocessor executes the same instruction on a group of threads called a **warp**.

The number of threads in a warp is the **warp size**.
Goal: Performance per millimeter

• For GPUs, performance == throughput

• Strategy: hide latency with computation not cache
  ➔ Heavy multithreading!

• Implication: need many threads to hide latency
  - Occupancy - typically prefer 128 or more threads/TPA
  - Multiple thread blocks/TPA help minimize effect of barriers

• Strategy: Single Instruction Multiple Thread (SIMT)
  - Support SPMD programming model
  - Balance performance with ease of programming
• High-level description of SIMT:
  - Launch zillions of threads
  - When they do the same thing, hardware makes them go fast
  - When they do different things, hardware handles it gracefully
SIMT Thread Execution

• Groups of 32 threads formed into **warps**
  - always executing same instruction
  - some become inactive when code path diverges
  - hardware **automatically handles divergence**

• Warps are the primitive unit of scheduling
  - pick 1 of 32 warps for each instruction slot
  - Note warps may be running different programs/shaders!

• **SIMT execution is an implementation choice**
  - sharing control logic leaves more space for ALUs
  - largely invisible to programmer
  - must understand for performance, not correctness
GPU Architecture: Trends

- Long history of ever-increasing programmability
  - Culminating today in CUDA: program GPU directly in C

- Graphics pipeline, APIs are abstractions
  - CUDA + graphics enable “replumbing” the pipeline

- Future: continue adding expressiveness, flexibility
  - CUDA, OpenCL, DX11 Compute Shader, …
  - Lower barrier further between compute and graphics
Moore’s Law gives you more and more transistors

What do you want to do with them?

CPU strategy: make the workload (one compute thread) run as fast as possible

Tactics:
- Cache (area limiting)
- Instruction/Data prefetch
- Speculative execution
  - limited by “perimeter” – communication bandwidth
  - then add task parallelism…multi-core

GPU strategy: make the workload (as many threads as possible) run as fast as possible

Tactics:
- Parallelism (1000s of threads)
- Pipelining
  - limited by “area” – compute capability
GPU Architecture

- Massively Parallel
  - 1000s of processors (today)
- Power Efficient
  - Fixed Function Hardware = area & power efficient
  - Lack of speculation. More processing, less leaky cache
- Latency Tolerant from Day 1
- Memory Bandwidth
  - Saturate 512 Bits of Exotic DRAMs All Day Long (140 GB/sec today)
  - No end in sight for Effective Memory Bandwidth
- Commercially Viable Parallelism
  - Largest installed base of Massively Parallel (N>4) Processors
    Using CUDA!!! Not just as graphics
- Not dependent on large caches for performance
  - Computing power = Freq * Transistors
  - Moore’s law ^2
GPU Architecture: Summary

- From fixed function to configurable to programmable
  - architecture now centers on flexible processor core
- Goal: performance / mm$^2$ (perf == throughput)
  - architecture uses heavy multithreading
- Goal: balance performance with ease of use
  - SIMT: hardware-managed parallel thread execution