Administrivia

- HW0 scores, HW1 peer-review assignments out.
- HW2 out, due Nov. 2.
- If you’re having Cython trouble with HW2, let us know.
- Review on Wednesday: Post questions on Piazza
Introduction to GPUs

With many slides from Kayvon Fatahalian
Single Core CPU

- Fetch/Decode
- ALU (Execute)
- Execution Context
- Data cache (a big one)
- Out-of-order control logic
- Fancy branch predictor
- Memory pre-fetcher

Majority of chip transistors used to perform operations that help a single instruction stream run fast.

More transistors → larger cache, smarter out-of-order logic, smarter branch predictor, etc.

(Also: more transistors → smaller transistors → higher clock frequencies)
Most of this logic is to help serial programs run quickly.

- **Fetch/Decode**
- **ALU (Execute)**
- **Execution Context**
- **Data cache** *(a big one)*
- **Out-of-order control logic**
- **Fancy branch predictor**
- **Memory pre-fetcher**

Majority of chip transistors used to perform operations that help a single instruction stream run fast. More transistors = larger cache, smarter out-of-order logic, smarter branch predictor, etc. (Also: more transistors → smaller transistors → higher clock frequencies)
How do we speed this up?

- Fetch/Decode
- ALU (Execute)
- Execution Context
- Data cache (a big one)
- Out-of-order control logic
- Fancy branch predictor
- Memory pre-fetcher

Majority of chip transistors used to perform operations that help a single instruction stream run fast.

More transistors = larger cache, smarter out-of-order logic, smarter branch predictor, etc.

(Also: more transistors → smaller transistors → higher clock frequencies)
Multiple Cores

Caches reduce length of stalls (reduce latency)

Processors run efficiently when data is resident in caches (caches reduce memory access latency, also provide high bandwidth to CPU)

- L1 cache (32 KB)
- L2 cache (256 KB)
- L3 cache (8 MB)
SIMD Extensions

Recall original compiled program:
Instruction stream processes one array element at a time using scalar instructions on scalar registers (e.g., 32-bit floats).

Add ALUs to increase compute capability.
Add lots of Cache

Hides latency, lets multiple threads interleave.

Core 1

Core N

L1 cache (32 KB)

L2 cache (256 KB)

L3 cache (8 MB)

Memory

DDR3 DRAM

(Gigabytes)

25 GB/sec
How does SIMD interact with control logic & cache?

- Fetch/Decode
- ALU (Execute)
- Execution Context
- Data cache (a big one)
- Out-of-order control logic
- Fancy branch predictor
- Memory pre-fetcher

Majority of chip transistors used to perform operations that help a single instruction stream run fast.

More transistors → larger cache, smarter out-of-order logic, smarter branch predictor, etc.

(Also: more transistors → smaller transistors → higher clock frequencies)
GPU idea:
Throw out most of this.

Fetch/Decode
ALU (Execute)
Execution Context

Data cache (a big one)
Out-of-order control logic
Fancy branch predictor
Memory pre-fetcher

Majority of chip transistors used to perform operations that help a single instruction stream run fast.

More transistors = larger cache, smarter out-of-order logic, smarter branch predictor, etc.

(Also: more transistors → smaller transistors → higher clock frequencies)
Core i7

Example: Intel Core i7 (Sandy Bridge)

- 4 cores
- 8 SIMD ALUs per core

On campus:

- New GHC machines: 4 cores, 8 SIMD ALUs per core
- Machines in GHC 5207 (old GHC 3000 machines): 6 cores, 4 SIMD ALUs per core

4 Cores
8 SIMD ALUs / core
Example: NVIDIA GTX 480

15 cores
32 SIMD ALUs per core
1.3 TFLOPS
GTX-480 in more detail

NVIDIA GTX 480 core

- Fetch/Decode
- Fetch/Decode
- Execution contexts (128 KB)
- “Shared” memory (16+48 KB)

= SIMD function unit, control shared across 16 units (1 MUL-ADD per clock)

- This process occurs on another set of 16 ALUs as well
- So there are 32 ALUs per core
- \(15 \times 32 = 480\) ALUs per chip

Source: Fermi Compute Architecture Whitepaper, CUDA Programming Guide 3.1, Appendix G

NVIDIA GTX 480: more detail (just for the curious)
**CPU:**
Big caches, few threads, modest memory BW
Rely mainly on caches and prefetching

**GPU:**
Small caches, many threads, huge memory BW
Rely mainly on multi-threading
More Threads

NVIDIA GTX 480 core

- Fetch/Decode
- Execution contexts (128 KB)
- “Shared” memory (16+48 KB)

128 KB for “Contexts”

Registers, program state, etc.

Each core can have as many threads as it can hold their contexts.

Fast switching between threads.
Many small contexts

good latency hiding
Few large contexts
poor latency hiding

Diagram of a processor with:
- Fetch/Decode
- ALUs numbered 1 to 8
- Four cores numbered 1 to 4

The layout suggests a design with few large contexts and poor latency hiding ability.
Sounds great, what’s the catch?
Sounds great, what’s the catch?

Every instruction is SIMD.

All ALUs are doing exactly the same thing in lockstep.
a = b + c

d = e \times a

result = 3 \times d
After branch: continue at full performance

\[ a = b + c \]
\[ d = e \times a \]
\[ \text{result} = 3 \times d \]
After branch: continue at full performance

Time (clocks)

ALU 1 ALU 2 ... ... ALU 8

\[
\begin{align*}
\text{a} &= \text{b} + \text{c} \\
\text{d} &= \text{e} \times \text{a} \\
\text{result} &= 3 \times \text{d}
\end{align*}
\]
After branch: continue at full performance

ALU 1
ALU 2
... 
ALU 8

Time (clocks)

`if(x > 0) {
  <unconditional code>
} else {
  <resume unconditional code>
`
What about Branches?

if x > 0:
    tmp = x ** 5.0
else:
    tmp = 2 * tmp

result = tmp + 1
What about Branches?

```python
if x > 0:
    tmp = x ** 5.0
else:
    tmp = 2 * tmp
result = tmp + 1
```
What about Branches?

if \( x > 0 \):
    \( \text{tmp} = x \times 5.0 \)
else:
    \( \text{tmp} = 2 \times \text{tmp} \)

result = \text{tmp} + 1

Not all ALUs do useful work!

Worst case: 1/8 peak performance
What about Branches?

if x > 0:
    tmp = x ** 5.0
else:
    tmp = 2 * tmp

result = tmp + 1

Back to peak performance
Caches

NVIDIA GTX 480 core

Instead of large caches, hide latency with many more threads, and high memory bandwidth.
**CPU:**
- Big caches, few threads, modest memory BW
- Rely mainly on caches and prefetching

**GPU:**
- Small caches, many threads, huge memory BW
- Rely mainly on multi-threading

**L1 cache**
- (32 KB)
- (64 KB)

**L2 cache**
- (256 KB)
- (768 KB)

**L3 cache**
- (8 MB)
- (256 KB)

**Memory**
- DDR3 DRAM
- DDR5 DRAM

**Execution contexts**
- (128 KB)
- (128 KB)

**Scratchpad L1 cache**
- (12 KB)
- (64 KB)

**GFX texture cache**
- (12 KB)
- (768 KB)
GPUs - Summary

• Many, many Arithmetic Logic Units (ALUs).

• Many threads per core (efficiency & latency hiding).

• High memory bandwidth (for bandwidth-bound applications)

• **Every** instruction is SIMD within a core.

• Memory bandwidth has to be managed for peak performance (more on this later).
Incoming

• Wednesday
  • Review (post questions!)

• Friday
  • Intro to Odyssey (and OpenCL).