Welcome!
Today’s Agenda:

- Grand Recap
- Exam
- Now What
Today’s Agenda:

- Grand Recap
- Exam
- Now What
Recap
Instruction Cost

What is the ‘cost’ of a multiply?

```
float x = 0, y = 0.1f;
unsigned int i = 0, j = 0x28929227;
for( int k = 0; k < ITERATIONS; k++ )
{
    // ...
    x += y, y *= 1.01f;
    // ...
    i += j, j ^= 0x17737352, i >>= 1, j /= 28763;
    // ...
    if (with) x *= y;
    // ...
    i += j, j ^= 0x17737352, i >>= 1, j /= 28763;
}
```

dummy = x + (float)i;

```
fldz
xor ecx, ecx
fld dword ptr ds:[405290h]
add ecx, edx
mov eax, 91D2A969h
xor edx, 17737352h
shr ecx, 1
mul eax, edx
fld st(1)
faddp st(3), st
fld st(1)
fmul st(1), st
mov eax, 91D2A969h
shr edx, 0
add ecx, edx
jne tobetimed<0>+1Fh
```
Instruction Cost

What is the ‘cost’ of a single instruction?

Cost is highly dependent on the surrounding instructions, and many other factors. However, there is a ‘cost ranking’:

- **bit shifts**
- simple arithmetic, logical operands
- multiplication
- division
- sin, cos, tan, pow, exp

This ranking is generally true for any processor (including GPUs).
Instruction Pipeline

CPU Instruction Pipeline

Instruction execution is typically divided in four phases:

1. **Instruction Fetch**
   
   Retrieve the instruction from RAM

2. **Instruction Decode**
   
   Decode the byte-code

3. **Execute**
   
   Execute the instruction

4. **Writeback**
   
   The result is written to RAM/registers

Clock cycle

Cycles per instruction (CPI): 4
Instruction Pipeline

CPU Instruction Pipeline

This leads to a ‘superscalar’ pipeline, allowing for ‘instruction level parallelism’. This type of pipeline allows for the execution of multiple instructions simultaneously, assuming that not all instructions belong to the same class.

<table>
<thead>
<tr>
<th>i</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>fp₀</td>
<td>EX</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>int₀</td>
<td>EX</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>m₀</td>
<td>EX</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>fp₁</td>
<td></td>
<td>EX</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>int₁</td>
<td>EX</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>m₁</td>
<td></td>
<td>EX</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>fp₂</td>
<td></td>
<td></td>
<td>EX</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>int₂</td>
<td>EX</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>m₂</td>
<td></td>
<td></td>
<td></td>
<td>EX</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Clock cycle

From 4 CPI to 3 IPC
Out of Order execution:

The compiler and the CPU re-organizes code to maximize throughput/latencies.

Q: When is re-ordering instructions most lucrative?

A: When we feed the compiler with enough mixed integer/floating point instructions
Speculative execution

The CPU uses a hardware technique called branch prediction to predict the direction of the branching.

Q: Which piece of hardware is responsible for predicting the branching path?

A: Re-order buffer (ROB) maintains the status of all instruction execution and retires in-order.
Rules of Engagement

Common Opportunities in Low-level Optimization

1. Avoid Costly Operations
2. Precalculate
3. Pick the Right Data Type
4. Avoid Conditional Branches
5. Early Out
6. Use the Power of Two
7. Do Things Simultaneously
Consistent Approach

Consistent approach in software optimization

1. **Determine optimization requirements**
2. **Profile: determine hotspots**
3. **Analyze hotspots: determine scalability**
4. **Apply high-level optimizations to hotspots**
5. **Profile again**
6. **Parallelize / vectorize / use GPGPU**
7. **Profile again**
8. **Apply low level optimizations to hotspots**
9. **Repeat steps 7 and 8 until time runs out**
10. **Report**
Q: Profiling is hard due to measurement noise. What are possible sources of noise?

- Dynamic frequency scaling
- (Filesystem) cache
- Cloud environment
- Shared resources

Q: How do we handle this noise in our measures?

- Multiple runs
- Metric choice
- Comparison of metrics
Q: Why is time not always the best profiling metric?

A: Because it does not tell us where the optimization opportunities lie.

Terminology and Metrics in performance Analysis we discussed

- Retired vs. executed instruction
- CPU utilization
- CPI and IPC
- Micro-operations
- Pipeline slot
- Core vs. reference cycles
- Cache misses
- Branch mispredictions

Many more!
Recap – lecture 4 & 5

SIMD Basics

Other instructions:

- \_m128 c4 = \_mm\_div\_ps(a4, b4); // component-wise division
- \_m128 d4 = \_mm\_sqrt\_ps(a4); // four square roots
- \_m128 d4 = \_mm\_rcp\_ps(a4); // four reciprocals
- \_m128 d4 = \_mm\_rsqrt\_ps(a4); // four reciprocal square roots (!)
- \_m128 d4 = \_mm\_max\_ps(a4, b4);
- \_m128 d4 = \_mm\_min\_ps(a4, b4);

Keep the assembler-like syntax in mind:

- \_m128 d4 = dx4 * dx4 + dy4 * dy4;

Agner Fog:

“Automatic vectorization is the easiest way of generating SIMD code, and I would recommend to use this method when it works. Automatic vectorization may fail or produce suboptimal code in the following cases:

- when the algorithm is too complex.
- when data has to be re-arranged in order to fit into vectors and it is not obvious to the compiler how to do this or when other parts of the code needs to be changed to handle the re-arranged data.
- when it is not known to the compiler which data sets are bigger or smaller than the vector size.
- when it is not known to the compiler whether the size of a data set is a multiple of the vector size or not.
- when the algorithm is required to functions that are defined elsewhere or cannot be inlined which are not readily available in vector versions.
- when the algorithm involves many branches that are not easily vectorized.
- when floating point operations have to be reordered or transformed and it is unknown to the compiler whether these transformations are permissible with respect to precision, overflow, etc.
- when functions are implemented with lookup tables.
Recap – lecture 7

INFOMOV – Lecture 14 – “Digest & Recap”

槽0

槽1

槽2

槽3

0000
0001
0002
0003
0004
0005
0006
0007
0008
0009
000A
000B
000C
000D
000E
000F

T0

T1

L1 I-$

L1 D-$

L2 $

L3 $

T0

T1

L1 I-$

L1 D-$

L2 $

T0

T1

L1 I-$

L1 D-$

L2 $

T0

T1

L1 I-$

L1 D-$

L2 $

 Survive = SurvivalProbability; 

diameter = doing it properly, weight off; 

Radiance = SampleLightBand; 

x + radiance.y + radiance.z) > 0 

E = \left(\text{weight} \times \cos(\theta)\right) / \text{radius}

window walk done properly, closest to driven;

\text{weight} \times \cos(\theta) / \text{radius}

\text{weight} \times \cos(\theta) / \text{radius}
Recap – lecture 9
Recap – lecture 10
GPGPU Architecture

**CPU**
- 4 cores
- L1 and L2 cache on chip
- L3 cache shared
- RAM separate
- Integrated graphics chip

**GPU**
- Hundreds to thousands of cores!
- Level 1 and 2 cache on chip
- RAM separate
GPU Architecture

Hierarchical order

TPC

PolyMorph Engine
GPU Architecture

Streaming Multiprocessors (SMs)

A SM contains the logical cores for performing operations on data.

SMs on modern NVidia GPUs contain:
- 4 processing blocks containing:
  - 1x 16 FP32 units
  - 1x 16 FP32 and 16 INT32 units
  - 4 Tensor Cores
  - L0 i-cache, warp scheduler, instruction dispatch
- 1 Ray Tracing Core
- L1 Data cache/Shared Memory
- 4 Texture Units

In total, each SM contains $16 \times 2 \times 4 = 128$ cores that can operate simultaneously on FP data.
Thread Execution

Quick overview of thread execution

**Thread or work-item**

Block or work-group

**Scalar processor**

**SMs**

A grid of blocks is always executed on the same device

**Threads are executed by a scalar processor**

**Blocks are executed on SMs**

They are always executed on the same SM
**GPGPU Memory**

We have seen several memory locations on a GPU:

Q: What is the accessibility level per memory location?

<table>
<thead>
<tr>
<th>Location</th>
<th>Accessibility</th>
<th>Size*</th>
<th>Latency (cycles)</th>
<th>Cached</th>
<th>Constant</th>
<th>Location (on/off chip)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Register</td>
<td>Thread</td>
<td>64k 32-bit registers per SM</td>
<td>0~1</td>
<td>-</td>
<td>No</td>
<td>On</td>
</tr>
<tr>
<td>Local</td>
<td>Thread</td>
<td>?</td>
<td>400~600</td>
<td>Yes</td>
<td>No</td>
<td>Off</td>
</tr>
<tr>
<td>Shared</td>
<td>Block</td>
<td>0KB – 100KB per SM</td>
<td>1~32</td>
<td>-</td>
<td>No</td>
<td>On (L1 data-cache)</td>
</tr>
<tr>
<td>Global</td>
<td>All threads + host</td>
<td>Multiple GB</td>
<td>400~600</td>
<td>No</td>
<td>No</td>
<td>Off</td>
</tr>
<tr>
<td>Texture</td>
<td>All threads + host</td>
<td>Multiple GB</td>
<td>400~600</td>
<td>Yes</td>
<td>Yes</td>
<td>Off</td>
</tr>
<tr>
<td>Constant</td>
<td>All threads + host</td>
<td>64KB</td>
<td>400~600</td>
<td>Yes</td>
<td>Yes</td>
<td>Off</td>
</tr>
</tbody>
</table>
GPGPU SIMT

- Threads are executed in groups of 32 called 'warps' (wavefronts on AMD)
- Warps execute in lockstep
- At each cycle, all threads in a warp must execute the same instruction with different data

Q: How does the GPU handle conditional code?

<table>
<thead>
<tr>
<th>Thread</th>
<th>cycle 0</th>
<th>cycle 1</th>
<th>cycle 2</th>
<th>cycle 3</th>
<th>cycle 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>i0</td>
<td>i1</td>
<td>?</td>
<td>?</td>
<td>i4</td>
</tr>
<tr>
<td>1</td>
<td>i0</td>
<td>i1</td>
<td>?</td>
<td>?</td>
<td>i4</td>
</tr>
<tr>
<td>2</td>
<td>i0</td>
<td>i1</td>
<td>?</td>
<td>?</td>
<td>i4</td>
</tr>
<tr>
<td>...</td>
<td>i0</td>
<td>i1</td>
<td>?</td>
<td>?</td>
<td>i4</td>
</tr>
<tr>
<td>31</td>
<td>i0</td>
<td>i1</td>
<td>?</td>
<td>?</td>
<td>i4</td>
</tr>
</tbody>
</table>

```c
int a = get_local_id(0); // i0
if (a % 2 == 0) // i1
    a = 0;
else a = 1; // i2
return a; // i4
```
### GPGPU SIMT

- Threads are executed in groups of 32 called 'warps' (wavefronts on AMD)
- Warps execute in lockstep
- At each cycle, all threads in a warp must execute the same instruction with different data

**Q: How does the GPU handle conditional code?**

<table>
<thead>
<tr>
<th>Thread</th>
<th>cycle 0</th>
<th>cycle 1</th>
<th>cycle 2</th>
<th>cycle 3</th>
<th>cycle 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>i0</td>
<td>i1</td>
<td>i2</td>
<td></td>
<td>i4</td>
</tr>
<tr>
<td>1</td>
<td>i0</td>
<td>i1</td>
<td></td>
<td>i3</td>
<td>i4</td>
</tr>
<tr>
<td>2</td>
<td>i0</td>
<td>i1</td>
<td>i2</td>
<td></td>
<td>i4</td>
</tr>
<tr>
<td>...</td>
<td>i0</td>
<td>i1</td>
<td>...</td>
<td>...</td>
<td>i4</td>
</tr>
<tr>
<td>31</td>
<td>i0</td>
<td>i1</td>
<td>i3</td>
<td></td>
<td>i4</td>
</tr>
</tbody>
</table>

```cpp
int a = get_local_id(0); // i0
if (a % 2 == 0) // i1
    a = 0; // i2
else a = 1; // i3
return a; // i4
```
GPGPU Programming Model

We have seen that when we launch a GPU kernel, that we must specify a local work-size, a global work-size, and the working dimensions.

Q: What is the trade-off that we must consider when choosing a local work-size?
A: We must consider resource-sharing vs. warp swapping

Q: Does it matter if we pick 1D, 2D or 3D for our working dimensions for performance?
A: No, we can always achieve the same functionality with various working dimensions.
When profiling GPGPU applications, there are very little performance metrics available compared to the CPU.

Q: Why is this the case?

CPU vs. GPU

CPUs hide latencies via
- Super-scalar execution
- Out-of-order execution
- Branch prediction
- Cache hierarchy
- Speculative prefetching

GPUs hide latencies via
- Swapping blocked warps
In the third GPGPU lecture we showed a conversion from a CPU fluid simulation to a GPGPU version. After converting the code to GPGPU, we did some simple optimizations which made the application run about 6 times faster.

Q: Where lie many optimization opportunities in GPGPU?
Recap – Lecture 14
"Dear Charles,

We will talk a lot about the O(N).

In other words:

- You will be able to do this to virtually any program.
- You will be able to apply these principles to new/already
  applied to.
- You will have a more intimate relationship with your
  computer.

At the end of this course:

You will know how to speed up critical code by a factor 2x to 10x (and more).

[Disclaimer: That has not been patented by an expert.]

INFOMOV - Lecture 14 - "Digest & Recap"
Today's Agenda:

- Grand Recap
- Exam
- Now What
Exam

What to Study

1. Slides

2. Literature on the website and in the slides:
   - Modern Microprocessors: a 90 minute guide, see lecture 2 slides or click here
   - What Every Programmer Should Know About Memory (just the yellow bits)
   - Gallery of Processor Cache Effects (link)
   - Game Programming Patterns - Data Locality
   - Data-Oriented Design (Or Why You Might Be Shooting Yourself in the Foot With OOP)
   - The Neglected Art of Fixed Point Arithmetic


4. Skills you picked up with the practical assignments
Example Questions

CPUs and GPUs have fundamentally different core strategies for dealing with latencies such as memory access time. What are these strategies?
Example Questions

Why is the theoretical peak performance of a GPU typically much higher than that of a CPU?
Example Questions

Explain the concept of streaming processing.
Exam

Example Questions

What or who is NUMA?

You may bring a dictionary to the exam.
You may answer in Dutch, if you wish.
You may not bring notes to the exam.
You may bring pizza to the exam.
Example Questions

Explain what false sharing is.
Exam

Example Questions

How does a GPU handle conditional code?
Exam

Example Questions

Why does OpenCL have a native `sqrt` as well as an `sqrtf`?
Exam

Example Questions

Do modern systems still use SRAM? Why / why not?

You may bring a dictionary to the exam.
You may answer in Dutch, if you wish.
You may not bring notes to the exam.
You may bring pizza to the exam.
Example Questions

How many tag bits are needed for a 128KB 8-way set associative cache, assuming a cache line size of 128 bytes?
Example Questions

Is self-modifying code possible on a modern processor? Under what conditions?
Today’s Agenda:

- Grand Recap
- Exam
- Now What
Now What
Now What
Now What

COMPUTER PROGRAMMER

What my friends think I do

What my mom thinks I do

What society thinks I do

What my spouse thinks I do

What I think I do

What I actually do
"That's all Folks!"

/INFOMOV2022/