Announcements

Project 3 Cache Race Games night Monday, May 4th, 5pm
- Come, eat, drink, have fun and be merry!
- Location: B17 Upson Hall

Prelim 2: Thursday, April 30th in evening
- Time and Location: **7:30pm sharp** in Statler Auditorium
- Old prelims are online in CMS
- Prelim Review Session:
  - **TODAY**, Tuesday, April 28, 7-9pm in B14 Hollister Hall

Project 4:
- Design Doc due May 5th, bring design doc to mtg May 4-6
- **Demos**: May 12 and 13
- **Will not be able to use slip days**
Announcements

Prelim2 Topics

- Lecture: Lectures 10 to 24
- Data and Control Hazards (Chapters 4.7-4.8)
- RISC/CISC (Chapters 2.16-2.18, 2.21)
- Calling conventions and linkers (Chapters 2.8, 2.12, Appendix A.1-6)
- Caching and Virtual Memory (Chapter 5)
- Multicore/parallelism (Chapter 6)
- Synchronization (Chapter 2.11)
- Traps, Exceptions, OS (Chapter 4.9, Appendix A.7, pp 445-452)

- HW2, Labs 3/4, C-Labs 2/3, PA2/3

- Topics from Prelim1 (not the focus, but some possible questions)
Survey

Are you a gamer?

a) PC Games (NVIDIA card, etc)
b) Xbox One/PlayStation 4/etc
c) Mobile phone
d) Online games (minecraft, etc)
e) You do not play games!
GPUs
The supercomputer in your laptop

GPU: Graphics processing unit

Very basic till about 1999
   Specialized device to accelerate display
Then started changing into a full processor

2000-....: Frontier times
Parallelism

CPU: Central Processing Unit
GPU: Graphics Processing Unit
Parallelism

GPU parallelism is similar to multicore parallelism

Key: How to gang schedule thousands of threads on thousands of cores?

Hardware multithreading with thousands of register sets

GPU Hardware multithreads (like multicore Hyperthreads)

• Multithread + extra PCs and registers – dependency logic
• Illusion of thousands of cores
• Fine grain hardware multithreading - Easier to keep pipelines full
GPU Architectures

Processing is highly data-parallel
- GPUs are highly multithreaded
- Use thread switching to hide memory latency
  - Less reliance on multi-level caches
- Graphics memory is wide and high-bandwidth

Trend toward general purpose GPUs
- Heterogeneous CPU/GPU systems
- CPU for sequential code, GPU for parallel code

Programming languages/APIs
- DirectX, OpenGL
- C for Graphics (Cg), High Level Shader Language (HLSL)
- Compute Unified Device Architecture (CUDA)
GPU-type computation offers higher GFlops

(Source: Sam Naffziger, AMD)
GPUs: Faster than Moore’s Law

Moore’s Law is for Wimps?!

One-pixel polygons (~10M polygons @ 30Hz)

Slope ~2.4x/year
(Moore’s Law ~ 1.7x/year)

Peak Performance (polygon s/sec)

Graph courtesy of Professor John Poulton (from Eric Haines)
Programmable Hardware

• Started in 1999
• Flexible, programmable
  – Vertex, Geometry, Fragment Shaders
• And much faster, of course
  • 1999 GeForce256: 0.35 Gigapixel peak fill rate
  • 2001 GeForce3: 0.8 Gigapixel peak fill rate
  • 2003 GeForceFX Ultra: 2.0 Gigapixel peak fill rate
  • ATI Radeon 9800 Pro: 3.0 Gigapixel peak fill rate
  • 2006 NV60: ... Gigapixel peak fill rate
  • 2009 GeForce GTX 285: 10 Gigapixel peak fill rate
  • 2011
    – GeForce GTC 590: 56 Gigapixel peak fill rate
    – Radeon HD 6990: 2x26.5
  • 2012
    – GeForce GTC 690: 62 Gigapixel/s peak fill rate
Evolution of GPU
Around 2000

Fixed function pipeline
Around 2005

Programmable vertex and pixel processors

G70 (Based on NV40): 2005
Post 2006: Unified Architecture

[Diagram showing the architecture with GigaThread Engine and L2 Cache components]
Why?

- Parallelism: thousands of cores
- Pipelining
- Hardware multithreading
- Not multiscale caching
  - Streaming caches
- Throughput, not latency
# Flynn’s Taxonomy

<table>
<thead>
<tr>
<th>Single Instruction Single Data (SISD)</th>
<th>Multiple Instruction Single Data (MISD)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single Instruction Multiple Data (SIMD)</td>
<td>Multiple Instruction Multiple Data (MIMD)</td>
</tr>
</tbody>
</table>
MIMD array of SIMD procs
Grids, Blocks, and Threads

![Diagram of Grid, Blocks, and Threads](image)
CUDA Memory

- Faster, per-block
- Fastest, per-thread
- Slower, global
- Read-only, cached
Heterogeneous Computing

Host: the CPU and its memory

Device: the GPU and its memory

Shuang Zhao, Cornell University, 2014
Programming using CUDA

Compute Unified Device Architecture

do_something_on_host();
kernel<<<nBlk, nTid>>>(args);
cudaDeviceSynchronize();
do_something_else_on_host();

Highly parallel

Shuang Zhao, Cornell University, 2014
Threads in a block are partitioned into warps

- All threads in a warp execute in a Single Instruction Multiple Data, or SIMD, fashion
- All paths of conditional branches will be taken
- Warp size varies, many graphics cards have 32

No guaranteed execution ordering between warps
Branch Divergence

Threads in one warp execute very different branches
Significantly harms the performance!

Simple solution:

• Reordering the threads so that all threads in each block are more likely to take the same branch
• Not always possible
Example: NVIDIA Tesla
Example: NVIDIA Tesla

Streaming Processors

• Single-precision FP and integer units
• Each SP is fine-grained multithreaded

Warp: group of 32 threads

• Executed in parallel, SIMD style
  – 8 SPs × 4 clock cycles
• Hardware contexts for 24 warps
  – Registers, PCs, ...
single-threaded free lunch
The diagram illustrates the progression of computing technology from the 1970s to the 2010s, with a focus on the era of single-threaded free lunch. It highlights the shift towards multicore and cloud-core technologies, with a warning that 'the free lunch is so over'. It also mentions the entry into the jungle of heterogeneous computing, marked by the need for 'exit moore'. The timeline extends beyond 2010, indicating ongoing developments and innovations.
Historical Perspective
Goal for Today: Back to the Future
Can multiple entities interact with processor at once?
Back to the Future: Timesharing
Can multiple entities interact with processor at once?

Instruction Fetch
- New PC
- Instruction
- PC

Instruction Decode
- Extend
- Control
- Detect hazard

Instruction Fetch/ID
- Memory
- Inst

ID/EX
- Register file
- ALU
- Address

EX/MEM
- Forward unit
- Execute
- Memory

MEM/WB
- Write-back
- Ctrl
Fernando J. Corbató (MIT)

- Known for pioneering time sharing systems and MULTICS operating system (later influence UNIX)
- Influences: Turing Award Recipient (1990).
  - "for his pioneering work in organizing the concepts and leading the development of the general-purpose, large-scale, time-sharing and resource-sharing computer systems"
  - Corbató's Law: "Regardless of whether one is dealing with assembly language or compiler language, the number of debugged lines of source code per day is about the same!"
1963 Timesharing: A Solution to Computer Bottlenecks

http://www.youtube.com/watch?v=Q07PhW5sCEk&feature=youtu.be

- Reporter John Fitch at the MIT Computation Center in an extended interview with MIT professor of computer science Fernando J. Corbato
- The prime focus of the film is timesharing, one of the most important developments in computing