CMPE110 Lecture 25
Security and GPUs

Heiner Litz

https://canvas.ucsc.edu/courses/12652

Announcements

- Next Friday Review Session 2
  - Post your questions on Piazza

- Final Exam
  - Tuesday, June 12th
  - 12 – 3pm

Security

OOO Processors

Modern processors optimize for ILP

Characteristics

- Fetch multiple instructions per cycle (2-4, superscalar)
- Execute instructions out-of-order
  - Execution order dictated by true dependencies
  - Memory/registers updated in order due to precise exception
- Speculative
  - Branch prediction
  - Also speculate no exceptions (page faults, etc)
  - Since memory/registers updated in order this is ok
- Deep cache hierarchies (2 to 4 layers)

OOO Execution – Dynamic Scheduling

Execute instructions out-of-order

Fetch multiple instructions per cycle using branch prediction

Figure out which are independent and execute them in parallel

Example

```
add   $t0, $t1, $t2
or  $t3, $t0, $t2
sub $t0, $t1, $t2
and $t5, $t0, $t2
```

Superscalar + Dynamic scheduling

```
add   $t0, $t1, $t2      sub $t0, $t1, $t2
or  $t3, $t0, $t2        and $t5, $t0, $t2
```

Requires register renaming to make it work

Recent Security Attacks to OOO Processors

Based on slides from Mark Hill

Full set at https://goo.gl/QcwqeP

Key issues

Our ISAs specify functionality but not timing

Our processors have a lot of internal state

Known as microarchitecture state

Attacks exploit timing behavior and internal state to bypass security checks

Side-Channel Attack: PRIME Secret in Micro-Arch

1. Prime micro-architectural state
   a. Repeatedly access array **`train[]`** to train branch predictor to expect access **`< bound`**
   b. Access all of array **`save[]`** to put it completely in a cache of size **`SIZE`**

Side-Channel Attack: SAVE Secret in Micro-Arch

2. Coerce processor into speculatively executing instructions that will be nullified to (a) find a secret & (b) save it in micro-architecture

```
branch (R1 >= bound) goto error ; Speculate not taken even if R1 >= bound
load R2 ← memory[train+R1]      ; Speculate to find SECRET outside of train[]
and R3 ← R2 && 0xffff           ; Speculate to convert SECRET bits into index
load R4 ← memory[save+SIZE+R3]  ; Speculate to save SECRET by victimizing
memory[save+R3] since it aliases in cache with new access memory[save+SIZE+R3]
```

3. HW detects mis-speculation

   Undoes architectural changes

   Leaves cache (micro-architecture) changes (correct by Architecture 1.0)

Side-Channel Attack: RECALL Secret from Micro-Arch


Spy vs. Spy, Mad Magazine, 1960

4: Probe time to access each element of **`save[]`** -- micro-architectural property;
If accessing **`save[foo]`** slow due to cache miss, then SECRET is **`foo`**. A leak!

5: Repeat many times to obtain secret information at some bandwidth. (More shifting/masking needed to get all SECRET bits victimizing 64B cache lines)

Well-known as covert timing channel (1983)

Meltdown v. Spectre

	Meltdown	Spectre
Architecture	Intel, Apple	Intel, Apple, ARM, AMD
Entry	Must have code execution on the system	Must have code execution on the system
Method	Intel Privilege Escalation + Speculative Execution	Branch prediction + Speculative Execution
Impact	Read kernel memory from user space	Read contents of memory from other users' running programs
Action	Software patching	Software patching (more nuanced)

Daniel Miessler 2018

Miessler Blog (https://danielmiessler.com/blog/simple-explanation-difference-meltdown-spectre/)

Meltdown

Can leak the contents of kernel memory at up to 500KB/s


```
; rcx = kernel address
; rbx = probe array
retry:
mov al, byte [rcx]
shl rax, 0xc
jz retry
mov rbx, qword [rbx + rax]
```

TRAP!! (not branch)
Under mis-speculation

Listing 2: The core instruction sequence of Meltdown. An inaccessible kernel address is moved to a register, raising an exception. The subsequent instructions are already executed out of order before the exception is raised, leaking the content of the kernel address through the indirect memory access.

Meltdown & Software

Bad: Meltdown operates with bug-free OS software

Good: Major commercial OSs patched for Meltdown ~January 2018

Idea: Don’t map (much) of protected kernel address space in user process

Spectre

Classic side-channel attack w/ deep micro-arch info

- Many ways to prime prime micro-architecture
  - Branch predictor
  - Branch target buffer
- Multiple covert channels
  - Timing (cache, register file, functional unit contention)
  - Power consumption
  - EMI

Spectre Applicability

Exploit branch mis-prediction to let Javascript steal from Chrome browser

- Demonstrated Intel Haswell/Skylake, AMD Ryzen, & several ARM cores
- Many other existing designs vulnerable

Spectre Mitigation

Branch prediction

- SW: Suppress branch prediction “when important” with **`mfence`**, etc.
- Insert mfence manually into all software?
- What if we miss one?
- HW could auto-magically suppress branch prediction when appropriate (???)

Spectre Mitigation

Branch Target Buffer

- SW: Not clear. Disable hyper-threading, etc.?
- HW: Make micro-architecture state private to thread (not core or processor)

More generally: Hard to mitigate threats NOT YET DEFINED. We don't have a good answer to this yet!

GPUs

18

History of GPUs

- Early video cards
  - Frame buffer memory with address generation for video output
- 3D graphics processing
  - Originally high-end computers (e.g., SGI)
  - Moore's Law $\Rightarrow$ lower cost, higher density
  - 3D graphics cards for PCs and game consoles
- Graphics Processing Units
  - Processors oriented to 3D graphics tasks
  - Vertex/pixel processing, shading, texture mapping, rasterization

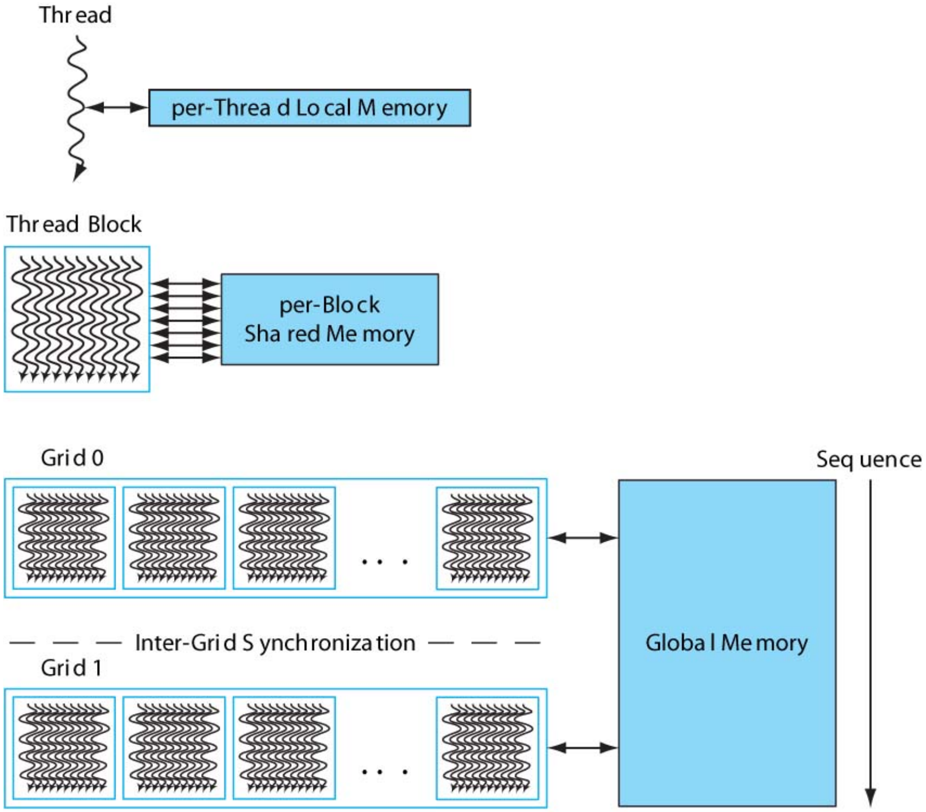
GPU Architectures

- Processing is highly data-parallel and highly multi-threaded
  - 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 Thread Model
Software View



Single instruction multiple threads (SIMT)

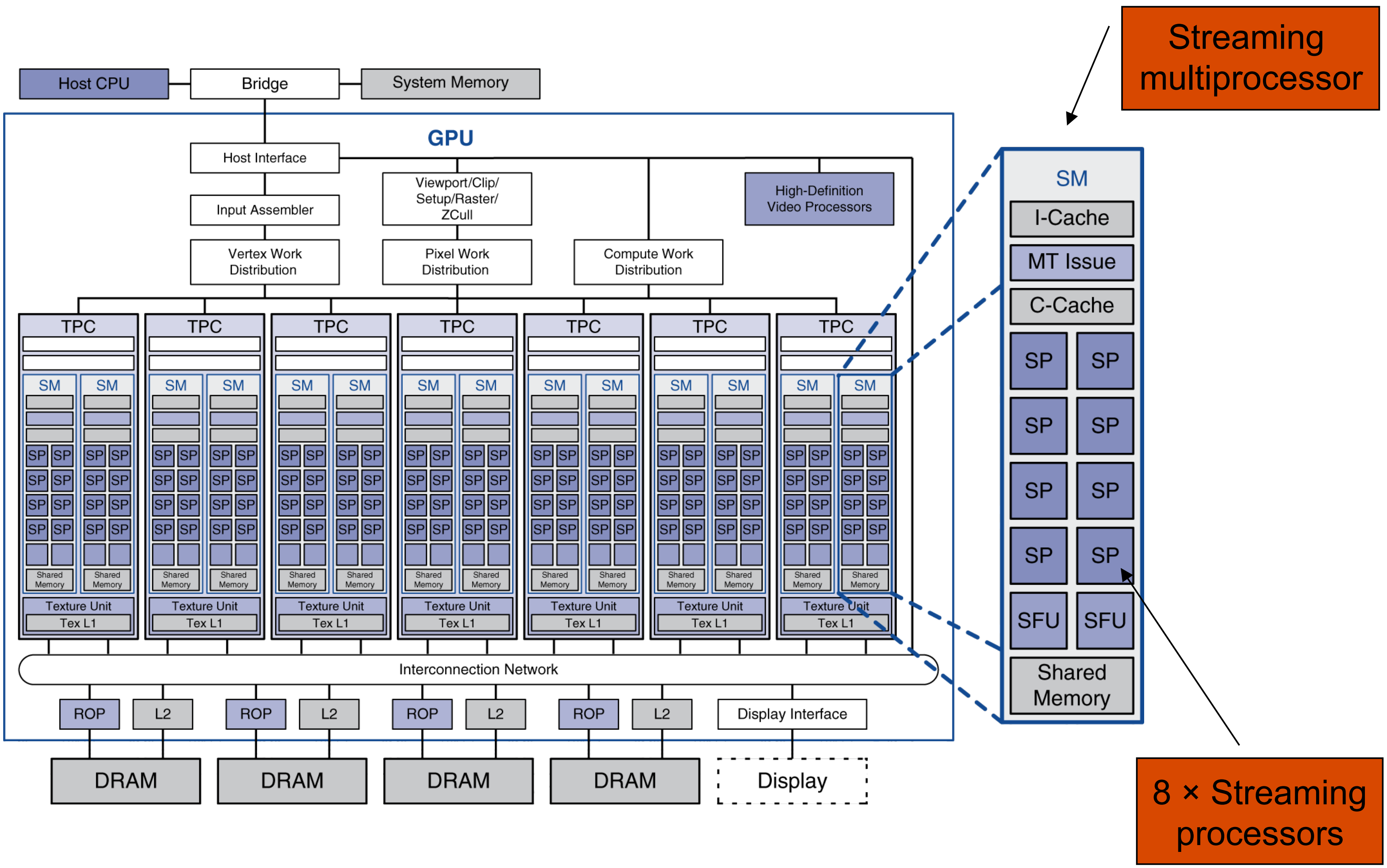
Parallel threads packed in blocks

- Also called Warps
- All threads execute in lockstep
- All threads perform same instruction
- Access to per-block shared memory
- Control flow divergence within a block with predication
- Switch thread block each cycle within a grid
- Can synchronize with barrier

GPU Memory Hierarchy

- Registers
  - Needed to support large number of threads without spilling to main memory: 64K (vs 32 of RISC-V)
- Shared Memory
  - Statically shared among threads
  - Explicitly managed by programmer
- Hardware managed cache
  - Only in recent GPGPUs, mainly for power reduction
- DRAM
  - High Bandwidth, small, soldered on PCB

Example: NVIDIA Tesla



Putting GPUs into Perspective

Feature	Multicore with SIMD	GPU
SIMD processors	4 to 8	8 to 16
SIMD lanes/processor	2 to 4	8 to 16
Multithreading hardware support for SIMD threads	2 to 4	16 to 32
Typical ratio of single precision to double-precision performance	2:1	2:1
Largest cache size	8 MB	0.75 MB
Size of memory address	64-bit	64-bit
Size of main memory	8 GB to 256 GB	4 GB to 6 GB
Memory protection at level of page	Yes	Yes
Demand paging	Yes	No
Integrated scalar processor/SIMD processor	Yes	No
Cache coherent	Yes	No