Chapter 6

Parallel Processors from Client to Cloud

Fall 2018

Soontae Kim
School of Computing, KAIST

Announcements

- HW#3 posted
  - Due on Final exam but not collected and graded
- Final exam
  - Dec. 11 (Tues) 4:00PM ~ 6:00PM
  - All class materials covered after midterm exam

Introduction

- Goal of architects : connecting multiple computers to get higher performance
  - Multiprocessors
  - Scalability, availability, power efficiency
- Task-level (process-level) parallelism(TLP)
  - High throughput for independent jobs
- Parallel processing program
  - Single program run on multiple processors
- Multicore microprocessors
  - Chips with multiple processors (cores)

Hardware and Software

- Hardware
  - Serial: e.g., Pentium 4
  - Parallel: e.g., quad-core Xeon e5345
- Software
  - Sequential: e.g., compilers
  - Concurrent: e.g., operating system
- Sequential/concurrent software can run on serial/parallel hardware
  - Challenge: making effective use of parallel hardware

Parallel Programming

- Parallel software is the problem
- Need to get significant performance improvement
  - Otherwise, just use a faster uniprocessor, since it's easier!
- Difficulties
  - Partitioning
  - Coordination
  - Communications overhead

Amdahl's Law

- Sequential part can limit speedup
- Example: 100 processors, 90× speedup?
  - $T_{new} = T_{parallelizable}/100 + T_{sequential}$
  - $$\text{Speedup} = \frac{1}{(1 - F_{parallelizable}) + F_{parallelizable}/100} = 90$$
  - Solving: $F_{parallelizable} = 0.999$
- Need sequential part to be 0.1% of original time

Instruction and Data Streams

- An alternate classification

		Data Streams	
		Single	Multiple
Instruction Streams	Single	**SISD**: Intel Pentium 4	**SIMD**: SSE instructions of x86
	Multiple	**MISD**: No examples today	**MIMD**: Intel Xeon e5345

- SPMD: Single Program Multiple Data
  - A single program running on all processors of a MIMD computer
  - Conditional code for different processors

Vector Processors

- Use highly pipelined function units
- Stream data to vector registers and the units
  - Data collected from memory into registers
  - Results stored from registers to memory
- Example: Vector extension to MIPS
  - 32 vector registers, each has 64 64-bit registers
  - Vector instructions
    - `lv`, `sv`: load/store vector
    - `addv.d`: add vectors of double
    - `addvs.d`: add scalar to each element of vector of double
- Significantly reduces instruction-fetch bandwidth

Example: DAXPY ($Y = a \times X + Y$)

- Conventional MIPS code

```
      l.d    $f0,a($sp)      ;load scalar a
      addiu  r4,$s0,#512     ;upper bound of what to load
loop: l.d    $f2,0($s0)      ;load x(i)
      mul.d  $f2,$f2,$f0     ;a × x(i)
      l.d    $f4,0($s1)      ;load y(i)
      add.d  $f4,$f4,$f2     ;a × x(i) + y(i)
      s.d    $f4,0($s1)      ;store into y(i)
      addiu  $s0,$s0,#8      ;increment index to x
      addiu  $s1,$s1,#8      ;increment index to y
      subu   $t0,r4,$s0      ;compute bound
      bne    $t0,$zero,loop  ;check if done
```

- Vector MIPS code

```
      l.d      $f0,a($sp)    ;load scalar a
      lv       $v1,0($s0)    ;load vector x
      mulvs.d  $v2,$v1,$f0   ;vector-scalar multiply
      lv       $v3,0($s1)    ;load vector y
      addv.d   $v4,$v2,$v3   ;add y to product
      sv       $v4,0($s1)    ;store the result
```

Vector vs. Scalar

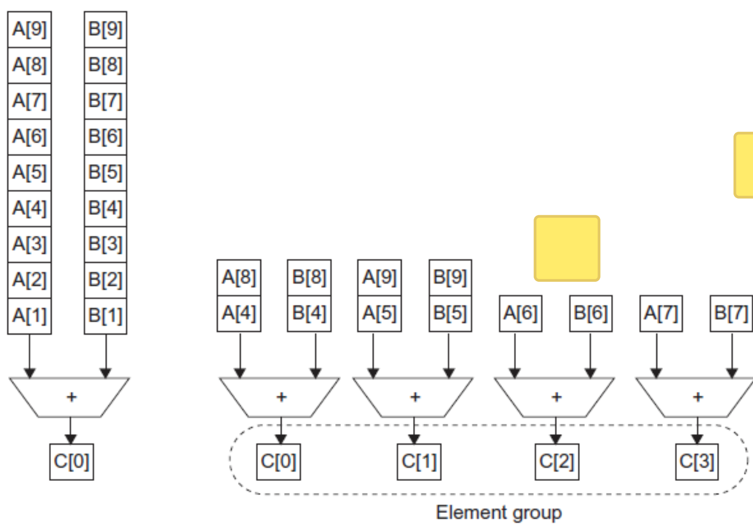
- Vector architectures and compilers
  - Simplify data-parallel programming
  - Explicit statement of absence of loop-carried dependences
    - Reduced checking in hardware
  - Regular access patterns benefit from interleaved and burst memory access
  - Avoid control hazards by avoiding loops
- More general than ad-hoc media extensions (such as MMX, SSE)
  - Better match with compiler technology

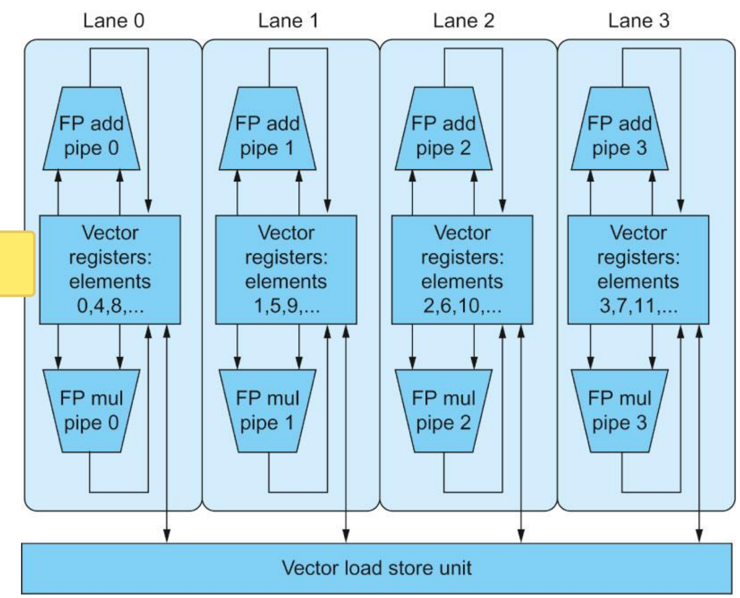
SIMD

- Operate elementwise on vectors of data
  - E.g., MMX and SSE instructions in x86
    - Multiple data elements in 128-bit wide registers
- All processors execute the same instruction at the same time
  - Each with different data address, etc.
- Simplifies synchronization
- Reduced instruction control hardware
- Works best for highly data-parallel applications

Vector vs. Multimedia Extensions

- Vector instructions have a variable vector width, multimedia extensions have a fixed width
- Vector instructions support strided access, multimedia extensions do not
- Vector units can be combination of pipelined and arrayed functional units:





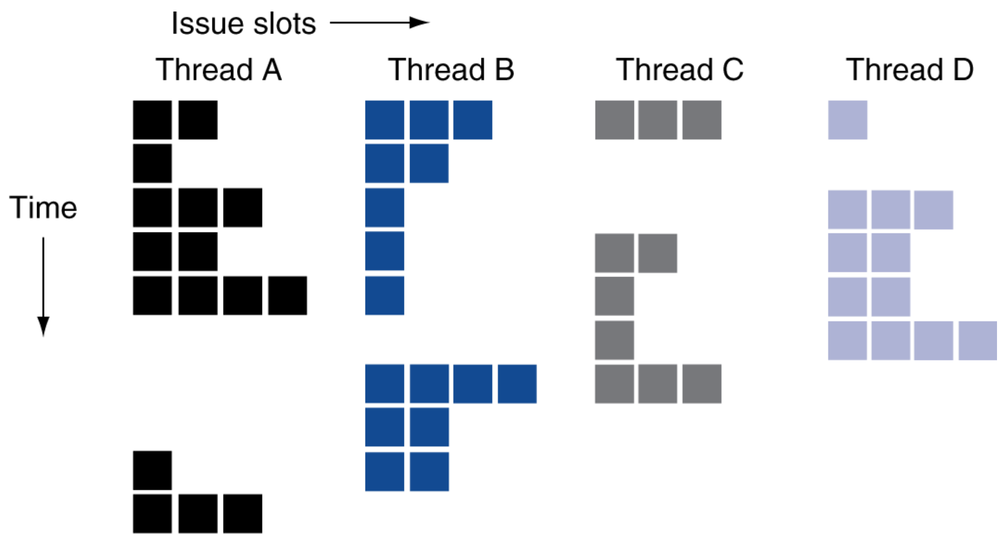
Multithreading

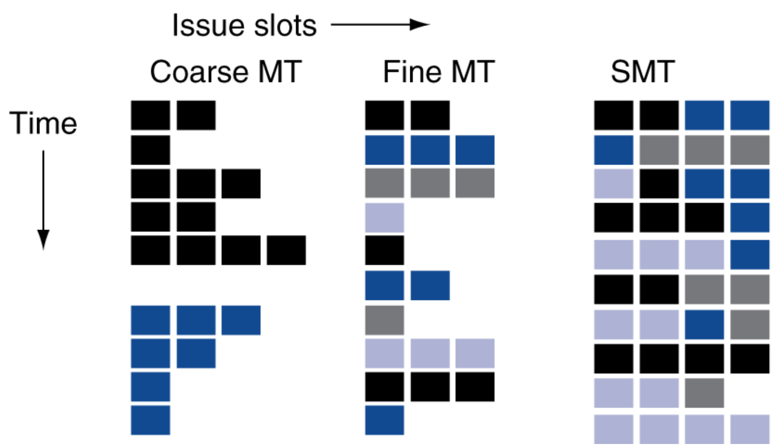
- Executing multiple threads in parallel using one CPU
  - Replicate registers, PC, etc.
  - Fast switching between threads
- Fine-grain multithreading
  - Switch threads after each cycle
  - Interleave instruction execution
  - If one thread stalls, others are executed
- Coarse-grain multithreading
  - Only switch on long stall (e.g., L2 cache miss)
  - Simplifies hardware, but doesn't hide short stalls (eg, data hazards)

Simultaneous Multithreading

- In multiple-issue dynamically scheduled processor
  - Schedule instructions from multiple threads
  - Instructions from independent threads execute when functional units are available
  - Within threads, dependencies are handled by scheduling and register renaming
- Example: Intel Pentium-4 Hyper Threading
  - Two threads: duplicated registers, shared functional units and caches

Multithreading Example

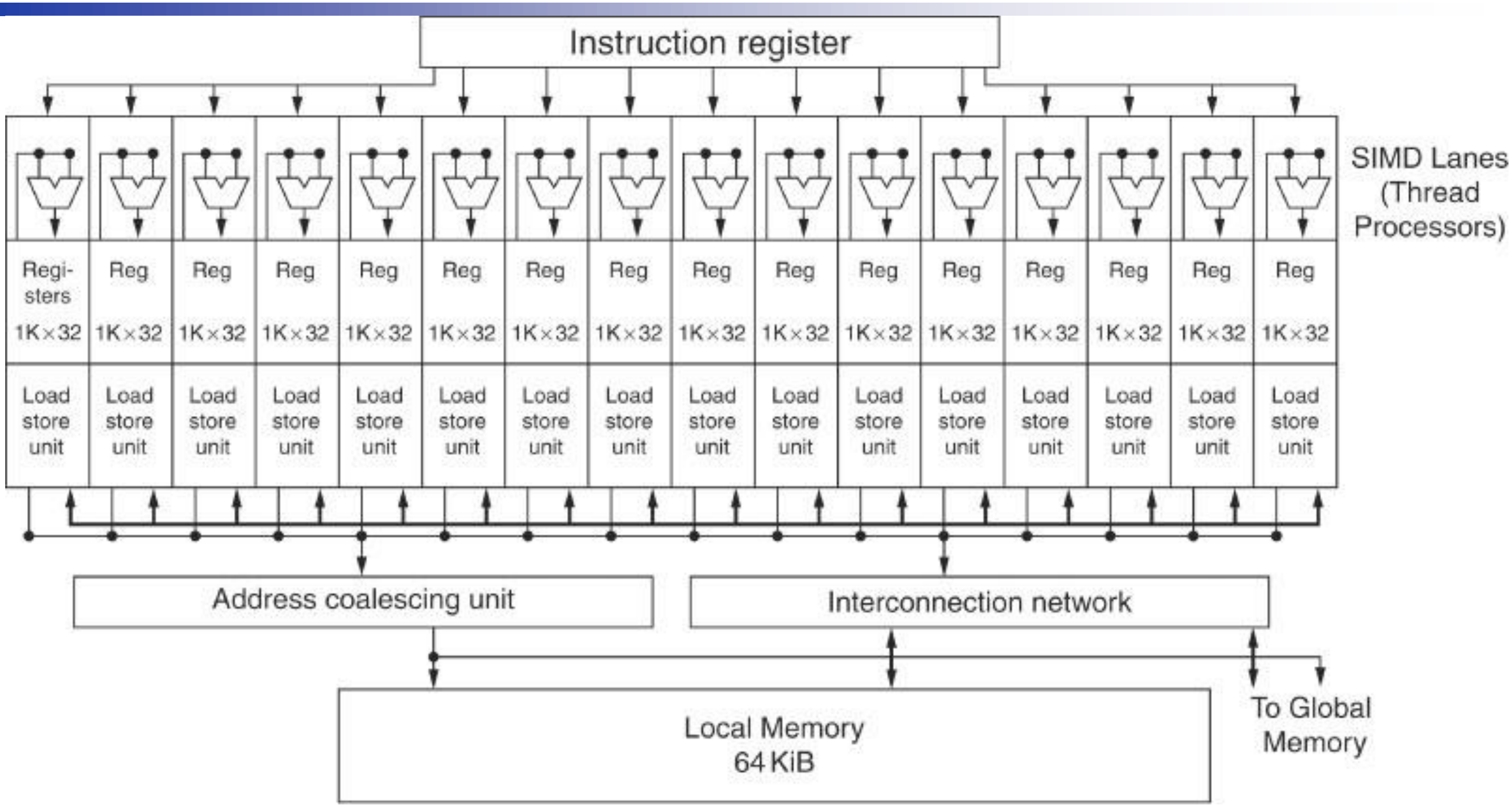




GPU Architectures

- 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
  - OpenCL
  - Compute Unified Device Architecture (CUDA)

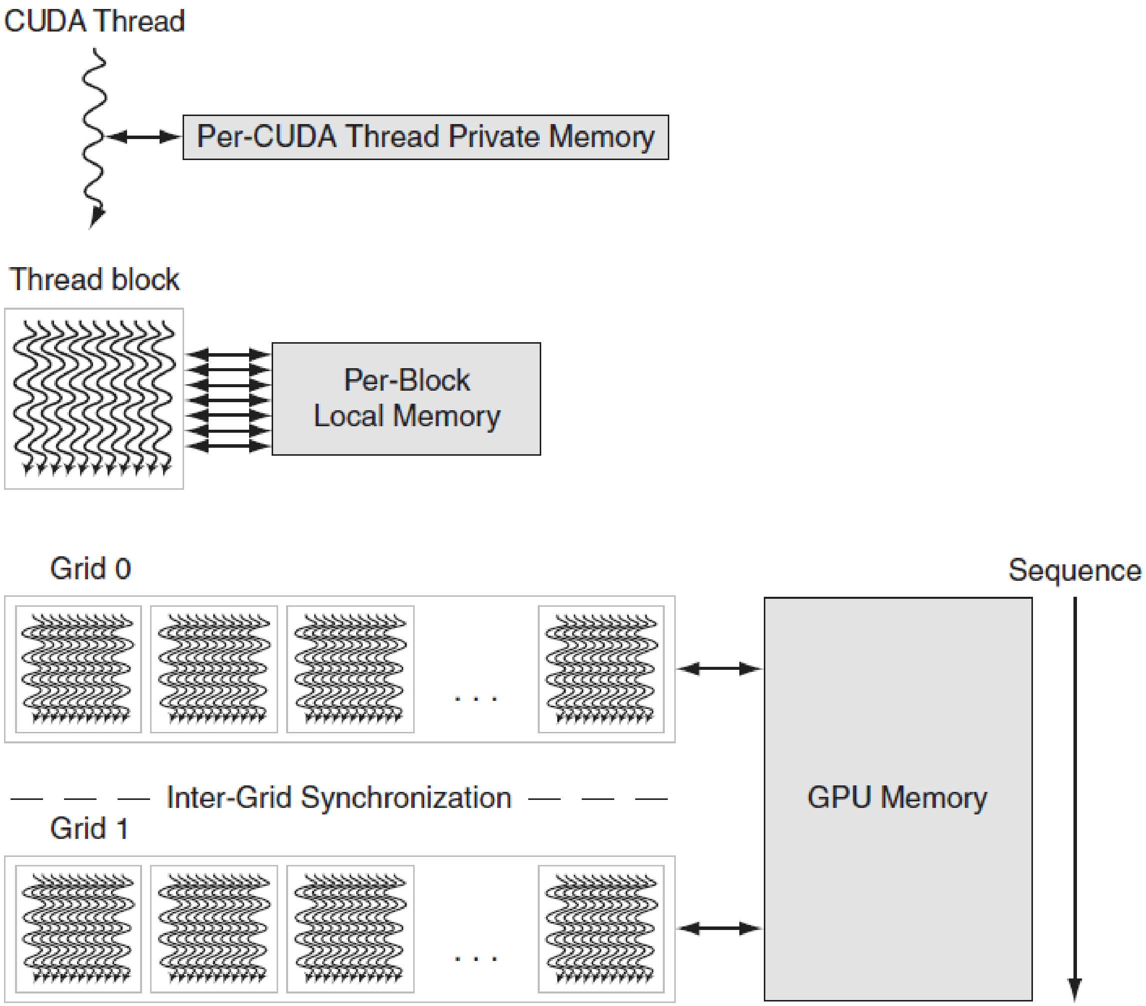
NVIDIA GPU Architectures



* GPU has many of multithreaded SIMD processors

FIGURE 6.9 Simplified block diagram of the datapath of a multithreaded SIMD Processor. It has 16 SIMD lanes. The SIMD Thread Scheduler has many independent SIMD threads that it chooses from to run on this processor.

GPU Memory Structures



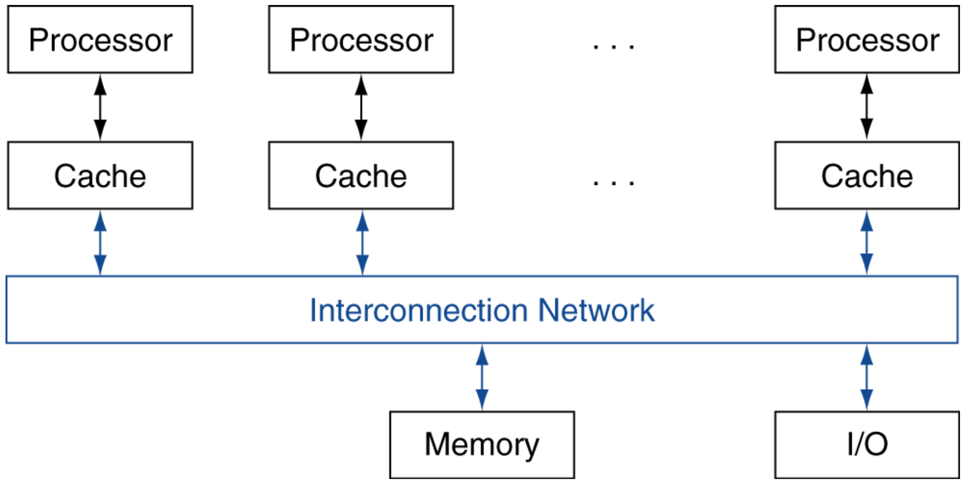
Classifying GPUs

- Don't fit nicely into SIMD/MIMD model
  - Conditional execution in a thread allows an illusion of MIMD
    - But with performance degredation
    - Need to write general purpose code with care

	Static: Discovered at Compile Time	Dynamic: Discovered at Runtime
Instruction-Level Parallelism	VLIW	Superscalar
Data-Level Parallelism	SIMD or Vector	**Tesla Multiprocessor**

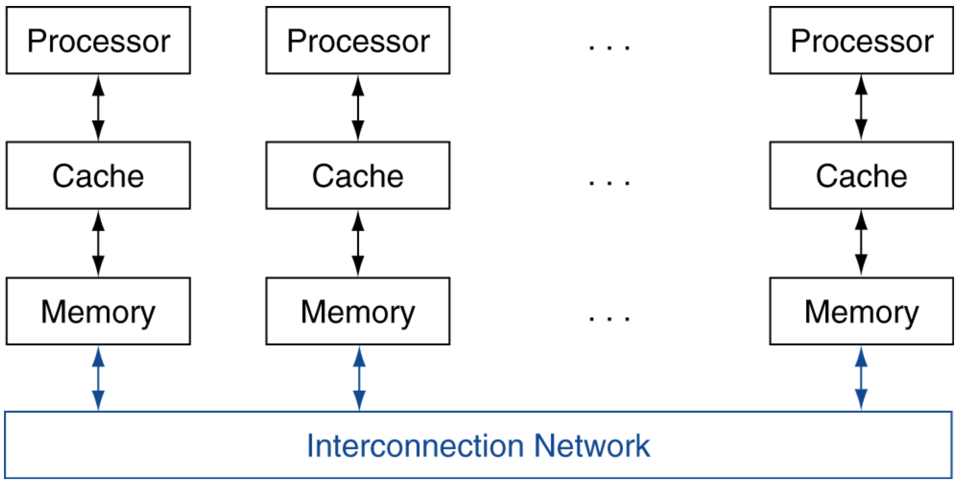
Shared Memory Multiprocessor

- SMP: shared memory multiprocessor
  - Hardware provides single physical address space for all processors
  - Synchronize shared variables using locks
  - Memory access time
    - UMA (uniform) vs. NUMA (nonuniform)



Message Passing Multiprocessor

- Each processor has private physical address space
- Hardware sends/receives messages between processors thru interconnection network



Loosely Coupled Clusters

- Network of independent computers
  - Each has private memory and OS
  - Connected using I/O system
    - E.g., Ethernet/switch, Internet
- Suitable for applications with independent tasks
  - Web servers, databases, simulations, …
- High availability, scalable, affordable
- Problems
  - Administration cost (prefer virtual machines)
  - Low interconnect bandwidth
    - c.f. processor/memory bandwidth on an SMP