COEN-4710 Computer Hardware
Lecture 8

From ILP to Parallel Processors (Ch.6)

Cristinel Ababei
Marquette University
Department of Electrical and Computer Engineering

1

Outline

- **Limits of Instruction Level Parallelism (ILP)**
- **TLP – Hardware Multithreading**
- **Multiprocessors and Multicores (+ Networks-on-Chip)**
- **Graphics Processor Units (GPUs)**
- **Domain Specific Architectures**

2

2

Getting CPI less than 1

Superscalar and VLIW processors

- Trade-off between static and dynamic instruction scheduling
  - Static scheduling places burden on software
  - Dynamic scheduling places burden on hardware
- Superscalar processors issue a variable number of instructions per cycle, up to a maximum, using static (compiler) or dynamic (hardware) scheduling
- VLIW processors issue a fixed number of instructions per cycle, seen as a packet (potentially with empty slots), and created and scheduled statically by the compiler

3

3

Superscalar Execution

- A superscalar architecture is one in which several instructions can be initiated simultaneously and executed independently.
- Pipelining allows several instructions to be executed at the same time, but they have to be in different pipeline stages at a given moment.
- Superscalar architectures include all features of pipelining but, in addition, there can be several instructions executing simultaneously in the same pipeline stage.

4

4

Superscalar Execution

- Higher instruction fetch bandwidth
- INT instructions, loads and stores occupy one slot
- FP instructions occupy second slot
  - Need pipelined FP datapath, otherwise FP datapath becomes bottleneck

Instruction type	Pipe stages						
INT instruction	IF	ID	EX	MEM	WB		
FP instruction	IF	ID	EX	MEM	WB		
INT instruction		IF	ID	EX	MEM	WB	
FP instruction		IF	ID	EX	MEM	WB	
INT instruction			IF	ID	EX	MEM	WB
FP instruction			IF	ID	EX	MEM	WB

5

5

Superscalar processor with dynamic scheduling

- Extend Tomasulo's algorithm to handle multiple issue
- Instructions issued in program order
- Cannot issue multiple dependent instructions in the same cycle. Two ways to deal with that:
  1. Issue stage split in halves to issue dependent instructions in the same cycle (one in half cycle, another in the second half); not scalable
  2. Add logic to be able to handle two or more instructions at the same time; this done at Issue stage to process instructions and find dependencies

6

6

Limitations of multiple-issue processors

Software and hardware implications

- Inherent limitations of ILP in programs (control and data dependencies)
  - Need as many independent instructions as pipeline depth
  - Deep unrolling, register pressure
- Hardware complexity increasing rapidly with instructions issued per cycle
  - Adding FUs scales complexity linearly
  - Higher memory and register-file bandwidth, more ports
  - Memory system implications, interleaving
  - Dynamic scheduling expensive
  - Issue logic of dynamic scheduled processors expensive
- Middle ground: combination of static (compiler) and dynamic scheduling

The Future?

- **Moving away from ILP enhancements**

	Power4	Power5	Power6	Power7
Introduced	2001	2004	2007	2010
Initial clock rate (GHz)	1.3	1.9	4.7	3.6
Transistor count (M)	174	276	790	1200
Issues per clock	5	5	7	6
Functional units	8	8	9	12
Cores/chip	2	2	2	8
SMT threads	0	2	2	4
Total on-chip cache (MB)	1.5	2	4.1	32.3

Figure 3.47 Characteristics of four IBM Power processors. All except the Power6 were dynamically scheduled, which is static, and in-order, and all the processors support two load/store pipelines. The Power6 has the same functional units as the Power5 except for a decimal unit. Power7 uses DRAM for the L3 cache.

Outline

- **Limits of Instruction Level Parallelism (ILP)**
- **TLP – Hardware Multithreading**
- **Multiprocessors and Multicores (+ Networks-on-Chip)**
- **Graphics Processor Units (GPUs)**
- **Domain Specific Architectures**

9

Performance Beyond Traditional ILP

- There can be much higher natural parallelism in some applications (e.g., Database or Scientific codes)
- Explicit
  - Thread Level Parallelism (TLP) or
  - Data Level Parallelism (DLP)
- Thread: light-weight process with own instructions and data
  - Each thread has all the state (instructions, data, PC, register state, and so on) necessary to allow it to execute
- Data Level Parallelism: Perform identical operations on data, and lots of data

10

Thread Level Parallelism (TLP)

- Goal: Improve Uniprocessor Throughput
- ILP exploits implicit parallel operations within a loop or straight-line code segment
- TLP explicitly represented by the use of multiple threads of execution that are inherently parallel
- Goal: Use multiple instruction streams to improve
  1. Throughput of computers that run many programs
  2. Execution time of multithreaded programs
- TLP could be more cost-effective to exploit than ILP

11

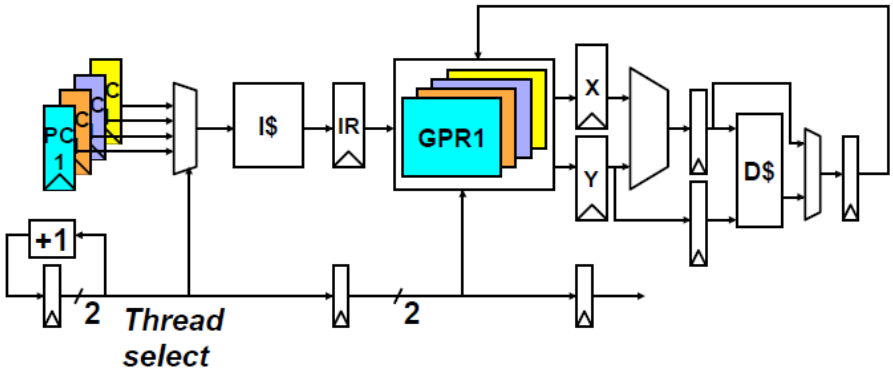
Multithreaded Execution

- Hardware Multithreading: multiple threads to share the functional units of 1 processor via overlapping
  - processor must duplicate independent state of each thread e.g., a separate copy of register file, a separate PC, and for running independent programs, a separate page table
  - memory shared through the virtual memory mechanisms, which already support multiple processes
  - HW for fast thread switch; much faster than full process switch (which can take 100s to 1000s of clocks)
    - » single process might contain multiple threads; all threads within a process share the same memory space, and can communicate with each other directly, because they share the same variables
- When to switch between threads?
  1. Alternate instruction per thread (fine grain)
  2. When a thread is stalled, perhaps for a cache miss, another thread can be executed (coarse grain)

12

12

Simple Multithreaded Pipeline



- Have to carry thread select down pipeline to ensure correct state bits read/ written at each pipe stage
- Appears to software (including OS) as multiple, albeit slower, CPUs

13

13

1. Fine-Grained Multithreading

- Switches between threads on each cycle, causing the execution of multiples threads to be interleaved
- Usually done in a round-robin fashion, skipping any stalled threads
- CPU must be able to switch threads every clock
- Advantage is it can hide both short and long stalls, since instructions from other threads executed when one thread stalls
- Disadvantage is it slows down execution of individual threads, since a thread ready to execute without stalls will be delayed by instructions from other threads
- Used on Sun's T1 (2005) and T2 (Niagara, 2007)

14

14

Fine-Grained Multithreading on the Sun T1

- **Circa 2005; first major processor to focus on TLP rather than ILP**

Characteristic	Sun T1
Multiprocessor and multithreading support	Eight cores per chip; four threads per core. Fine-grained thread scheduling. One shared floating-point unit for eight cores. Supports only on-chip multiprocessing.
Pipeline structure	Simple, in-order, six-deep pipeline with three-cycle delays for loads and branches.
L1 caches	16 KB instructions; 8 KB data. 64-byte block size. Miss to L2 is 23 cycles, assuming no contention.
L2 caches	Four separate L2 caches, each 750 KB and associated with a memory bank. 64-byte block size. Miss to main memory is 110 clock cycles assuming no contention.
Initial implementation	90 nm process; maximum clock rate of 1.2 GHz; power 79 W; 300 M transistors; 379 mm^2 die.

15

15

CPI on Sun T1

Benchmark	Per-thread CPI	Per-core CPI
TPC-C	7.2	1.80
SPECJBB	5.6	1.40
SPECWeb99	6.6	1.65

- T1 has 8 cores; with 4 threads/core
- Ideal effective CPI per thread? (4)
- Ideal per-core CPI? (1)
- In 2005 when it was introduced, the Sun T1 processor had the best performance on integer applications with extensive TLP and demanding memory performance, such as SPECJBB and transaction processing workloads

16

16

2. Course-Grained Multithreading

- Switches threads only on costly stalls, such as L2 cache misses
- Advantages
  - Relieves need to have very fast thread-switching
  - Doesn't slow down thread, since instructions from other threads issued only when the thread encounters a costly stall
- Disadvantage is hard to overcome throughput losses from shorter stalls, due to pipeline start-up costs
  - Since CPU issues instructions from 1 thread, when a stall occurs, the pipeline must be emptied or frozen
  - New thread must fill pipeline before instructions can complete
- Because of this start-up overhead, coarse-grained multithreading is better for reducing penalty of high cost stalls, where pipeline refill time << stall time
- Used in IBM AS/400, but not in modern processors

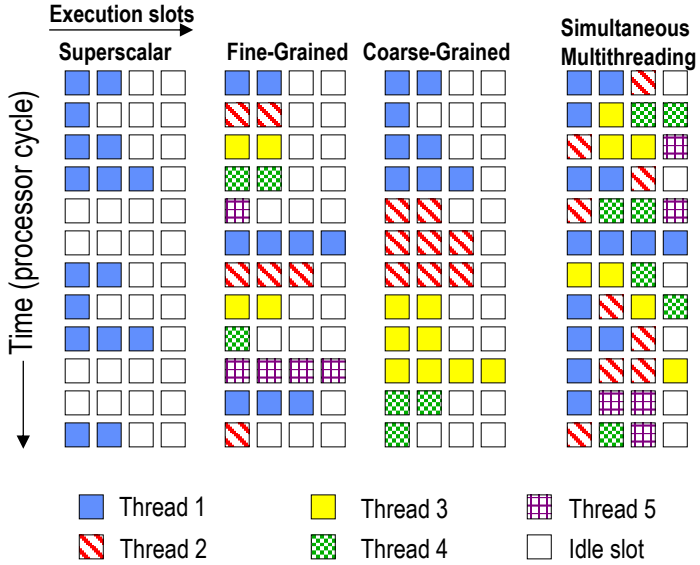
3. Simultaneous Multithreading (SMT) for OOO superscalars

- Techniques presented so far have all been "vertical" multithreading where each pipeline stage works on one thread at a time
- SMT uses fine-grain control already present inside an OOO superscalar to allow instructions from multiple threads to enter execution on same clock cycle. Gives better utilization of machine resources.

Simultaneous Multithreading (SMT)

- Simultaneous multithreading (SMT): a variation of fine-grained multithreading implemented on top of a multiple-issue, dynamically scheduled processor
- Add multiple contexts and fetch engines and allow instructions fetched from different threads to issue simultaneously
- Utilize wide out-of-order superscalar processor issue queue to find instructions to issue from multiple threads
- OOO instruction window already has most of the circuitry required to schedule from multiple threads
- Any single thread can utilize whole machine
- Used in Core i7 (2008) and IBM Power 7 (2010)

Illustration of Multithreading Categories



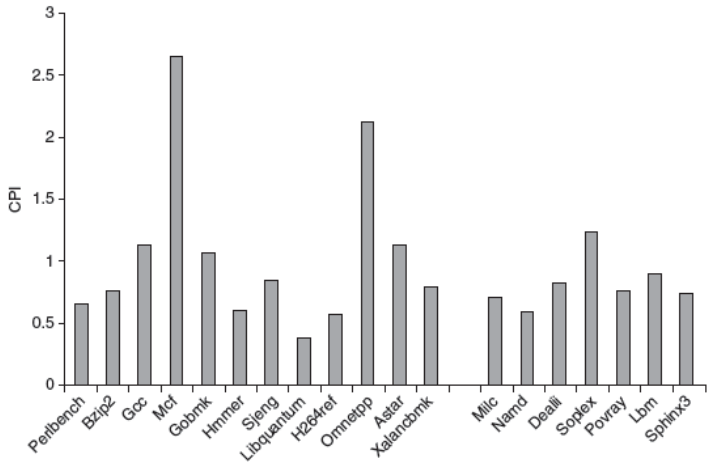
Example 1: Intel Core i7

- Aggressive out-of-order speculative microarchitecture
- Total pipeline depth is 14 stages; branch mispredictions cost 17 cycles
- 48 load and 32 store buffers
- Six independent functional units can each begin execution of a ready micro-op in the same cycle
- Uses a 36-entry centralized reservation station shared by six functional units, with ROB
  - Up to six micro-ops may be dispatched to functional units every cycle

21

21

Intel Core i7 Performance: CPI



The CPI for the 19 SPECCPU2006 benchmarks shows an average CPI of 0.83 for both the FP and integer benchmarks. SMT further improves performance by 15-30% (according to Intel)

22

22

Outline

- **Limits of Instruction Level Parallelism (ILP)**
- **TLP – Hardware Multithreading**
- **Multiprocessors and Multicores (+ Networks-on-Chip)**
- **Graphics Processor Units (GPUs)**
- **Domain Specific Architectures**

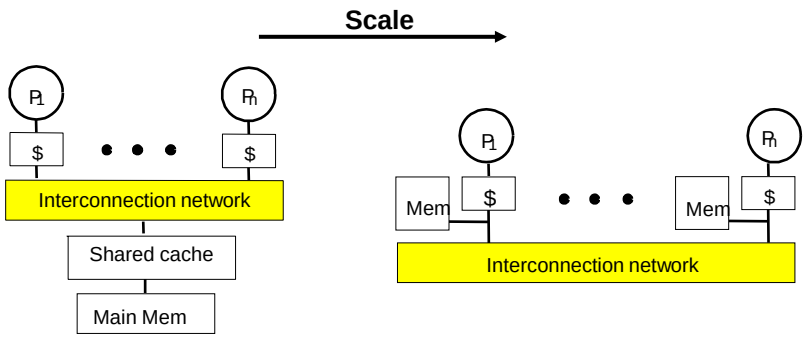
Back to Basics

- "A parallel computer is a collection of processing elements that cooperate and communicate to solve large problems fast."

Parallel Architecture =
Computer Architecture + Communication Architecture

- Two classes of multiprocessors WRT memory:
  1. Centralized Memory Multiprocessor
     - < few dozen processor chips (and < 100 cores)
     - Small enough to share single, centralized memory
  2. Physically Distributed-Memory multiprocessor
     - Larger number chips and cores than 1
     - BW demands ⇒ Memory distributed among processors

Centralized vs. Distributed Memory



Centralized Shared-memory Multiprocessor

Distributed-memory Multiprocessor

25

25

1. Centralized Memory Multiprocessor

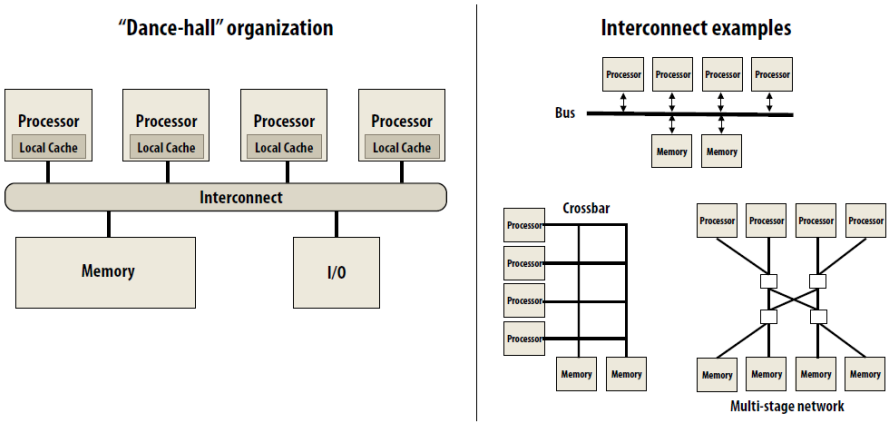
- Also called symmetric multiprocessors (SMPs) because single main memory has a symmetric relationship to all processors
- Large caches ⇒ single memory can satisfy memory demands of small number of processors
- Can scale to a few dozen processors by using a switch and by using many memory banks
- Although scaling beyond that is technically conceivable, it becomes less attractive as the number of processors sharing centralized memory increases

26

26

Centralized Memory Multiprocessor

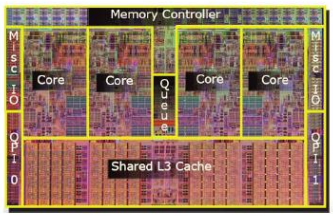
Any processor can directly reference any memory location



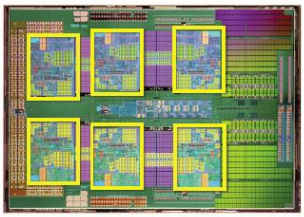
- Symmetric (shared-memory) multi-processor (SMP):
  - Uniform memory access time: cost of accessing an uncached* memory address is the same for all processors

27

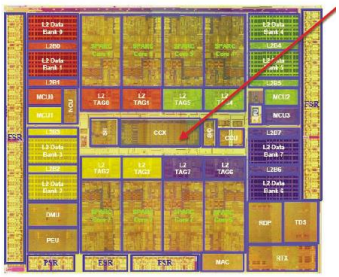
SMP Examples: Commodity processors



Intel Core i7 (quad core)
(network is a ring)



AMD Phenom II (six core)



Eight cores

Note size of crossbar: about die area of one core

28

28

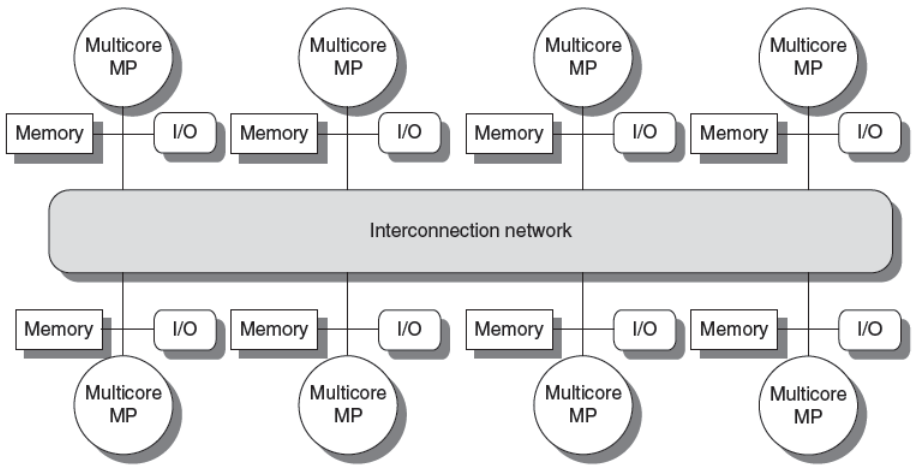
2. Distributed Shared Memory (DSM) Multiprocessor

- Also called non uniform memory access (NUMA) since the access time depends on the location of a data word in memory
- Pros:
  - Cost-effective way to scale memory bandwidth
    - If most accesses are to local memory
  - Reduces latency of local memory accesses
- Cons:
  - Communicating data between processors more complex
  - Must change software to take advantage of increased memory BW

29

29

Distributed Shared Memory *Multiprocessor*

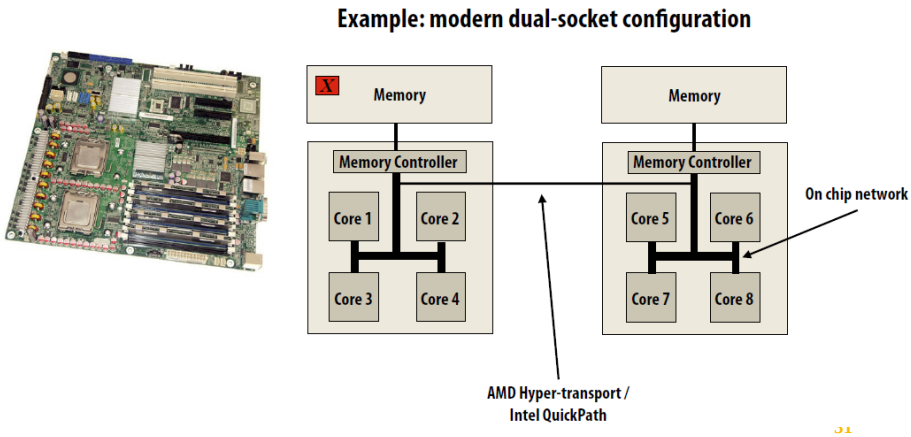


30

30

NUMA Example

Example: latency to access location X is higher from cores 5-8 than cores 1-4



31

31

Two Models for Communication and Memory Architecture

1. Communication in DSM and SMP occurs through a shared address space (via loads and stores): shared memory multiprocessors either
   - UMA (Uniform Memory Access time) for shared address, centralized memory MP
   - NUMA (Non Uniform Memory Access time multiprocessor) for shared address, distributed memory MP
2. Communication occurs by explicitly passing messages among the processors: message-passing multiprocessors
   - Mostly clusters and warehouse scale systems

32

32

Networks-on-Chip (NoC)

- See separate PPT presentation dedicated to this topic only!

33

33

Outline

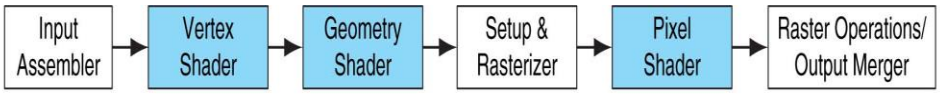
- Limits of Instruction Level Parallelism (ILP)
- TLP – Hardware Multithreading
- Multiprocessors and Multicores (+ Networks-on-Chip)
- Graphics Processor Units (GPUs)
- Domain Specific Architectures

34

34

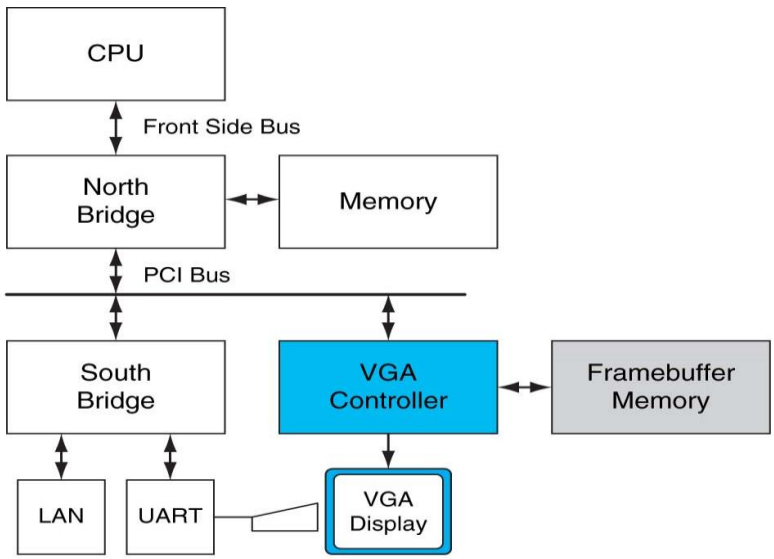
Graphics Processing Units (GPUs)

- Original GPUs were dedicated fixed-function devices for generating 3D graphics (mid-late 1990s) including high-performance floating-point units
  - Provide workstation-like graphics for PCs
  - Programmability was an afterthought
- Over time, more programmability added (2001-2005)
  - E.g., New language Cg (Nvidia) for writing small programs run on each vertex or each pixel, also Windows DirectX variants
  - Massively parallel (millions of vertices or pixels per frame) but very constrained programming model
- Graphics logical pipeline:



35

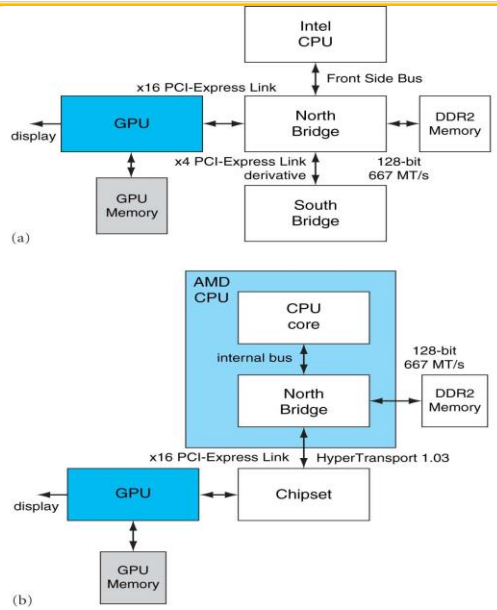
Historical PC



36

36

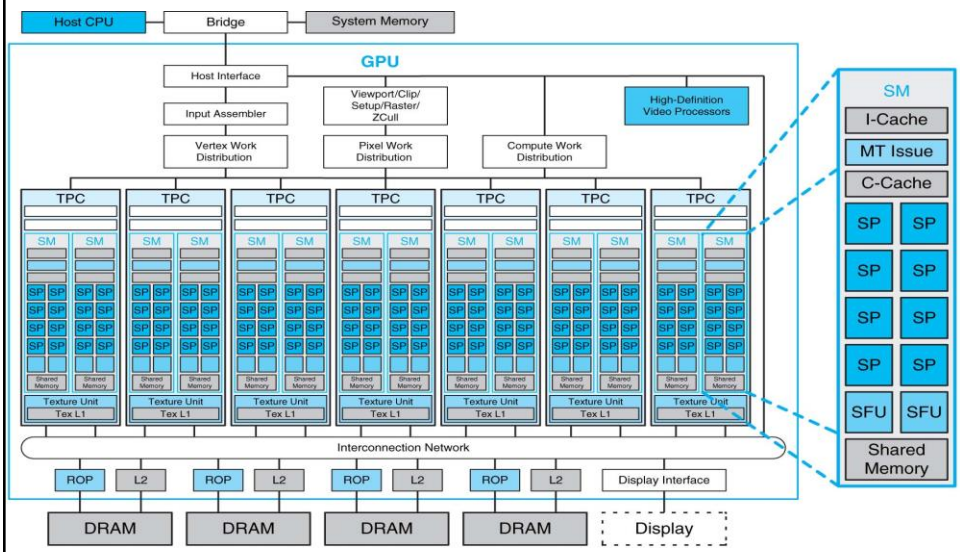
Contemporary: Intel, AMD



37

37

Basic unified GPU architecture



Example GPU with 112 streaming processor (SP) cores organized in 14 streaming multiprocessors (SMs); the cores are highly multithreaded. It has the basic Tesla architecture of an NVIDIA GeForce 8800. The processors connect with four 64-bit-wide DRAM partitions via an interconnection network. Each SM has eight SP cores, two special function units (SFUs), instruction and constant caches, a multithreaded instruction unit, and a shared memory.

38

38

A Shift in the GPU Landscape

- Some users noticed they could do general-purpose computation by mapping input and output data to images, and computation to vertex and pixel shading computations
- Referred to as general-purpose computing on graphics processing units (GP-GPU)
- Incredibly difficult programming model as had to use graphics pipeline model for general computation
  - A programming revolution was needed!

39

39

General-Purpose GPUs (GP-GPUs)

- In 2006, Nvidia introduced GeForce 8800 GPU supporting a new programming language: CUDA
  - "Compute Unified Device Architecture"
  - Subsequently, broader industry pushing for OpenCL, a vendor-neutral version of same ideas.
- Idea: Take advantage of GPU computational performance and memory bandwidth to accelerate some kernels for general-purpose computing
- Attached processor model: Host CPU issues data-parallel kernels to GP-GPU for execution
- This lecture has a simplified version of Nvidia CUDA-style model and only considers GPU execution for computational kernels, not graphics
  - Would need another course to describe graphics processing

40

CUDA Revolution

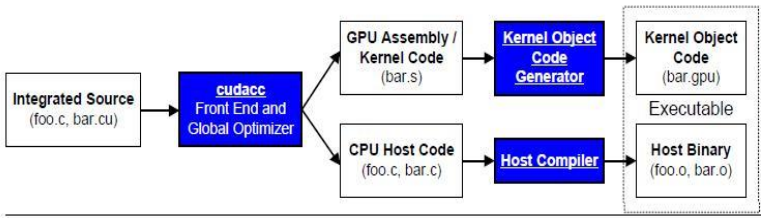
- CUDA Community Showcase
  - http://www.nvidia.com/object/gpu-applications.html
  - Computational fluid dynamics, EDA, finance, life sciences, signal processing, …
  - Speed-up's of >300x for some applications
- GPU Technology Conference
  - http://www.gputechconf.com/page/home.html
  - Include archive of previous editions
- General-Purpose Computation on Graphics Hardware
  - http://gpgpu.org/
  - Catalog the current and historical use of GPUs for general-purpose computation
- Download CUDA
  - https://developer.nvidia.com/cuda-downloads
  - And start using it!
- Many universities have already courses dedicated to teaching and using CUDA

41

41

CUDA Compilation

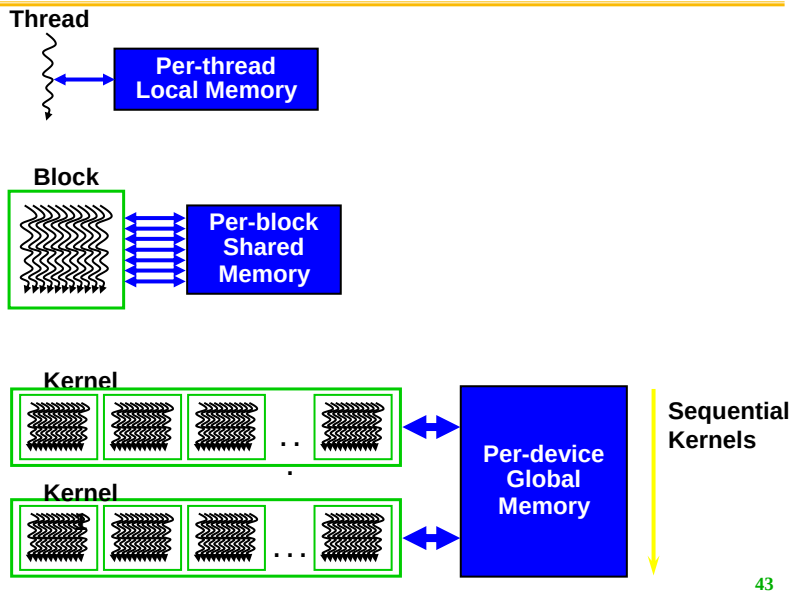
- As a programming model, CUDA is a set of extensions to ANSI C
- CPU code is compiled by the host C compiler and the GPU code (kernel) is compiled by the CUDA compiler. Separate binaries are produced



42

42

GPU Memory Hierarchy

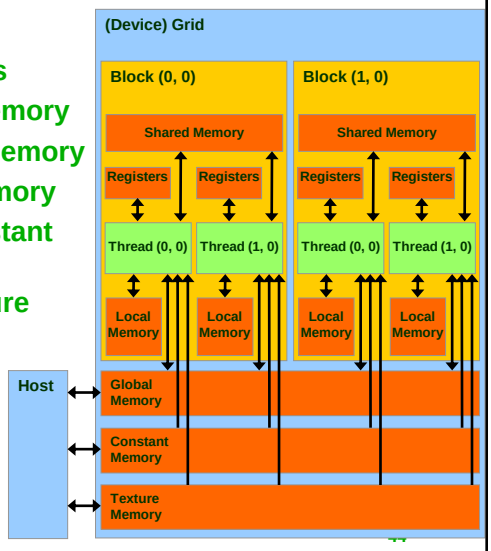


43

43

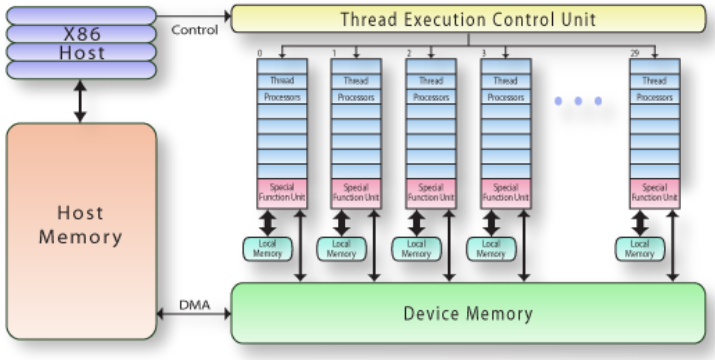
CUDA Device Memory Space Overview

- Each thread can:
  - R/W per-thread registers
  - R/W per-thread local memory
  - R/W per-block shared memory
  - R/W per-grid global memory
  - Read only per-grid constant memory
  - Read only per-grid texture memory
- The host can R/W global, constant, and texture memories



44

Example: Tesla Architecture

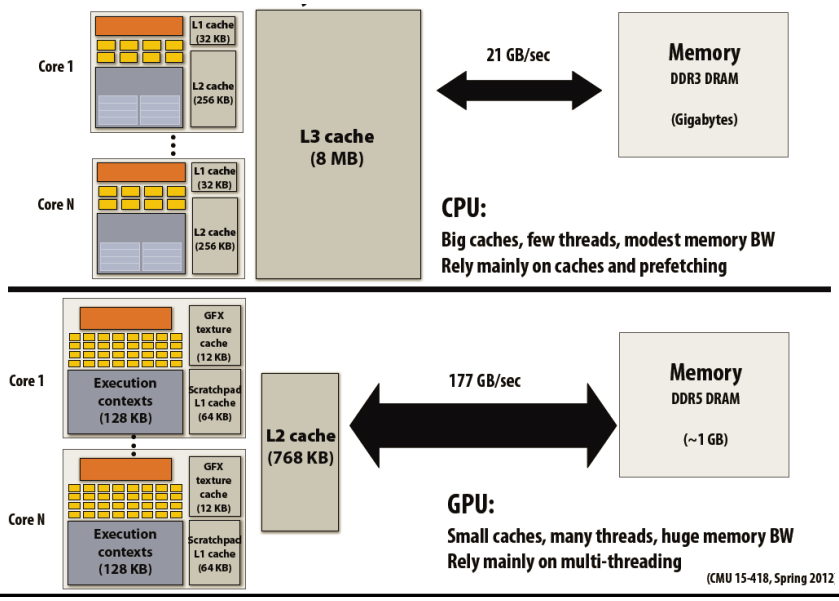


- Used for Technical and Scientific Computing
- L1/L2 Data Cache
  - Allows for caching of global and local data
  - Same on-chip memory used for Shared and L1
  - Configurable at kernel invocation

45

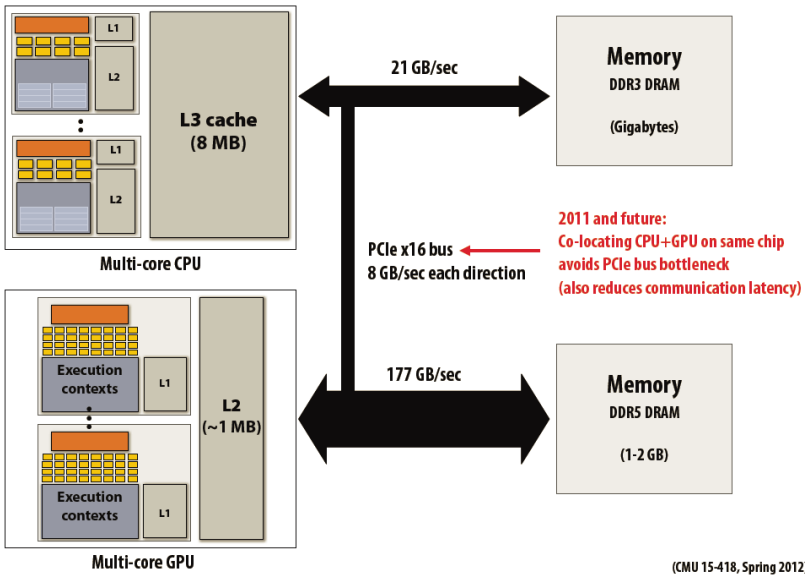
45

CPU vs. GPU memory hierarchies



46

Entire system view: CPU + discrete GPU



(CMU 15-418, Spring 2012)

47

Type	More descriptive name	Closest old term outside of GPUs	Official CUDA/ NVIDIA GPU term	Book definition
Program Abstractions	Vectorizable Loop	Vectorizable Loop	Grid	A vectorizable loop, executed on the GPU, made up of one or more Thread Blocks (bodies of vectorized loop) that can execute in parallel.
	Body of Vectorized Loop	Body of a (Strip-Mined) Vectorized Loop	Thread Block	A vectorized loop executed on a multithreaded SIMD Processor, made up of one or more threads of SIMD instructions. They can communicate via Local Memory.
	Sequence of SIMD Lane Operations	One iteration of a Scalar Loop	CUDA Thread	A vertical cut of a thread of SIMD instructions corresponding to one element executed by one SIMD Lane. Result is stored depending on mask and predicate register.
Machine Object	A Thread of SIMD Instructions	Thread of Vector Instructions	Warp	A traditional thread, but it contains just SIMD instructions that are executed on a multithreaded SIMD Processor. Results stored depending on a per-element mask.
	SIMD Instruction	Vector Instruction	PTX Instruction	A single SIMD instruction executed across SIMD Lanes.
Processing Hardware	Multithreaded SIMD Processor	(Multithreaded) Vector Processor	Streaming Multiprocessor	A multithreaded SIMD Processor executes threads of SIMD instructions, independent of other SIMD Processors.
	Thread Block Scheduler	Scalar Processor	Giga Thread Engine	Assigns multiple Thread Blocks (bodies of vectorized loop) to multithreaded SIMD Processors.
	SIMD Thread Scheduler	Thread scheduler in a Multithreaded CPU	Warp Scheduler	Hardware unit that schedules and issues threads of SIMD instructions when they are ready to execute; includes a scoreboard to track SIMD Thread execution.
	SIMD Lane	Vector lane	Thread Processor	A SIMD Lane executes the operations in a thread of SIMD instructions on a single element. Results stored depending on mask.
Memory Hardware	GPU Memory	Main Memory	Global Memory	DRAM memory accessible by all multithreaded SIMD Processors in a GPU.
	Local Memory	Local Memory	Shared Memory	Fast local SRAM for one multithreaded SIMD Processor, unavailable to other SIMD Processors.
	SIMD Lane Registers	Vector Lane Registers	Thread Processor Registers	Registers in a single SIMD Lane allocated across a full thread block (body of vectorized loop).

Figure 6.12 Quick guide to GPU terms. We use the first column for hardware terms. Four groups cluster these 12 terms. From top to bottom: Program Abstractions, Machine Objects, Processing Hardware, and Memory Hardware. 48

48

Outline

- Limits of Instruction Level Parallelism (ILP)
- TLP – Hardware Multithreading
- Multiprocessors and Multicores (+ Networks-on-Chip)
- Graphics Processor Units (GPUs)
- Domain Specific Architectures

Domain Specific Architectures (DSA)

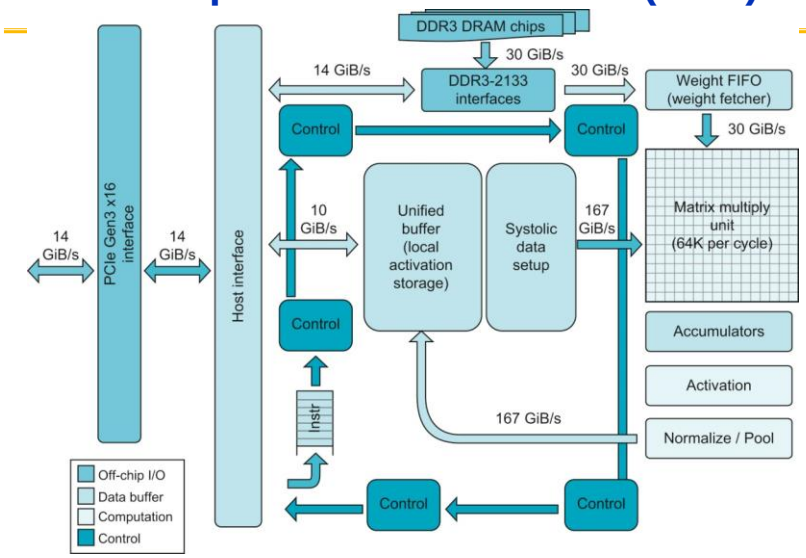


Figure 6.13 TPUv1 Block Diagram. The main computation part is the Matrix Multiply Unit (MXU) in the upper-right corner. Its inputs are the Weight FIFO and the Unified Buffer and its output is the Accumulators. The 24 MiB Unified Buffer is almost a third of the TPUv1 die, and the MXU with 65,536 multiple-accumulate ALUs is a quarter, so the datapath is nearly two-thirds of the TPUv1 die. For CPUs, Multilevel caches are often two-thirds of the die. (Adapted from Hennessy JL, Patterson DA.

DSA Five Principles

1. Use dedicated memories to minimize distance over which data are moved
2. Invest resources saved from dropping advanced microarchitectural optimizations into more arithmetic units or bigger memories
3. Use the easiest form or parallelism that matches the domain
4. Reduce data size and type to the simplest needed for the domain
5. Use a domain-specific programming language to port code to DSAs

51

51

Conclusion

- ILP ran out of steam
- Parallelism (thread and core) took over
- GPUs have become general purpose, thousands of cores
- DSAs at the beginning
- Cloud – datacenters/warehouse scale computers

52

52