Review

- Last Class:
  - Advanced translation optimizations
  - Main memory
- Today's class:
  - Thread-level parallelism
  - Multicore/manycore processors
- Announcement and reminder
  - HW3 will be distributed today
  - HW2 grades will be posted today
  - Midterm grades will be posted tomorrow
  - Correction on syllabus: We will have class on March 23rd

1

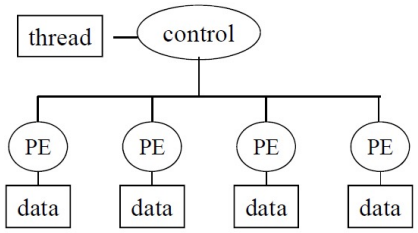
Flynn's Taxonomy of Computers

- Mike Flynn, "Very High-Speed Computing Systems," Proc. of IEEE, 1966
- SISD: Single instruction operates on single data element
- SIMD: Single instruction operates on multiple data elements
  - Array processor
  - Vector processor
- MISD: Multiple instructions operate on single data element
  - Closest form: systolic array processor, data driven streaming processor
- MIMD: Multiple instructions operate on multiple data elements (multiple instruction streams)
  - Multiprocessor
  - Multithreaded processor

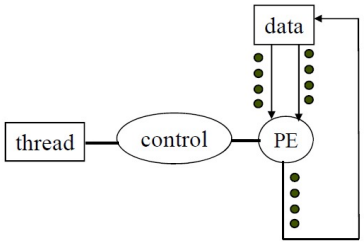
2

SIMD

- Synchronous, lockstep execution
  - All PEs execute the same instructions on different data
  - Also called array processor

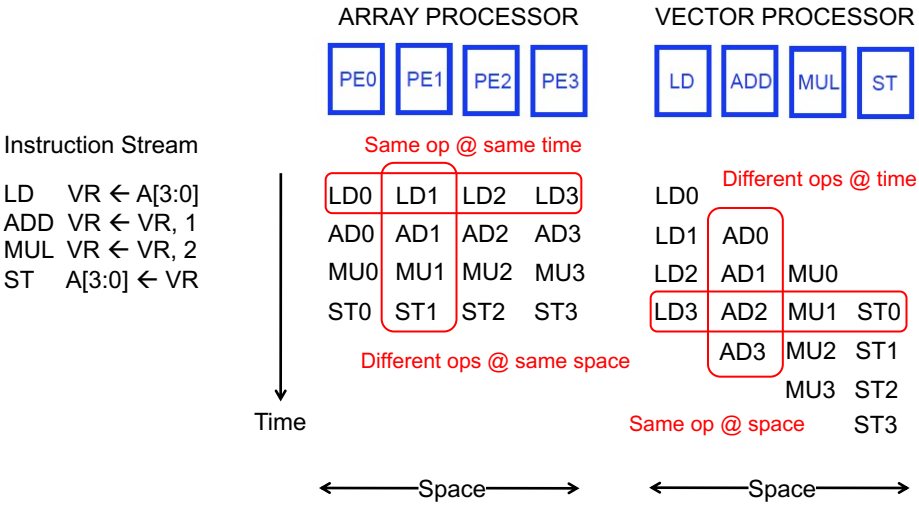


- Vector processing
  - The same instruction is repeatedly executed on different data



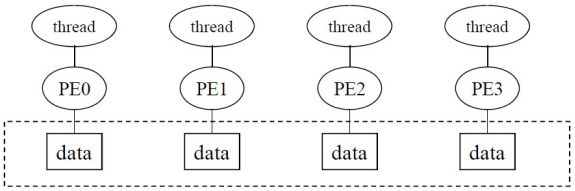
3

Array vs. Vector Processors



4

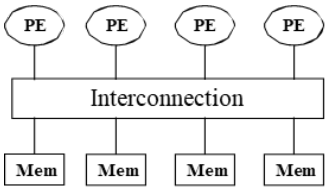
MIMD



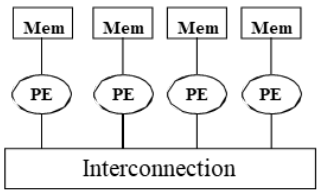
- Multiple threads executing on different data – However, if the threads are to cooperate to solve a problem (as opposed to solving different problems), there should be interaction between the programs and/or the data.
- Many flavors depending on the physical memory architecture and the virtual address space of each processing element (PE).
  - Address space = the range of memory addresses that the PE can access.

5

Physical Memory Architectures



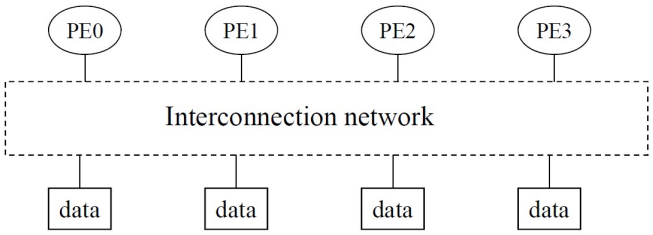
Global, shared memory (Symmetric Multi-Processors – SMP)



Distributed memory

6

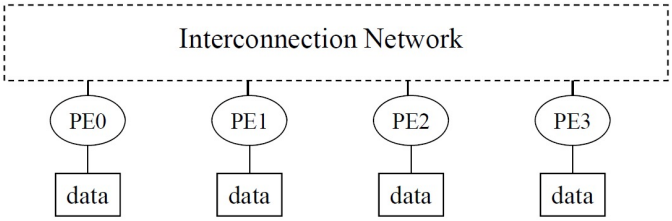
Shared Address Space



- Each of PE0, PE1, PE2 and PE3 can address (directly access) the same locations (shared virtual address space)
- No need for message passing – communicate through shared memory locations. E.g., OpenMP

7

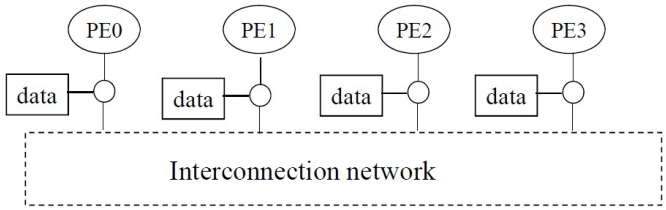
Distributed Address Space



- Each of PE0, PE1, PE2 and PE3 have separate virtual address spaces.
- In order for PEi to access data in the address space of PEj, the two processes have to communicate through explicit messages. E.g. MPI

8

Distributed Shared Memory Systems



- Shared address space, but physically distributed memory.
  - No need for message passing – communicate through shared memory locations.
- Data is physically distributed, but a runtime system is responsible to access data that do not reside in the local memory.
- Results in the so called "Non Uniform Memory Access" – NUMA (as opposed to UMA, "Uniform Memory Access")

Shared Memory multiProcessor (SMP)

- SMPs come in two styles
  1. Uniform memory access (UMA) multiprocessors

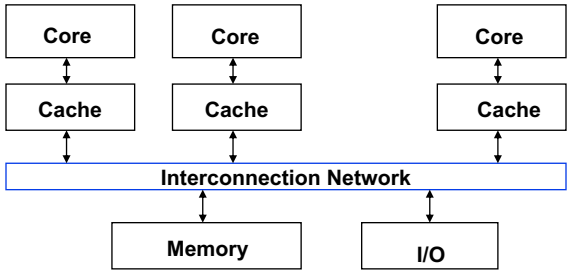
     Accesses to main memory take about the same amount of time no matter which processor requests the access and no matter which word is asked for
  2. Nonuniform memory access (NUMA) multiprocessors

     some memory accesses are much faster than other depending on which processor asks for which word

     Programming NUMA is harder than programming UMA
- Q1 – Single physical address space shared by all cores
- Q2 – Threads on cores coordinate/communicate through shared variables in memory (via loads and stores)
  1. Use of shared data must be coordinated via synchronization primitives (locks) that allow access to data to only one core at a time (used to implement critical sections)
  2. Caches must be kept coherent (read of a data item returns the most recently written value of that data item)

The Big Picture: Where are We Now?

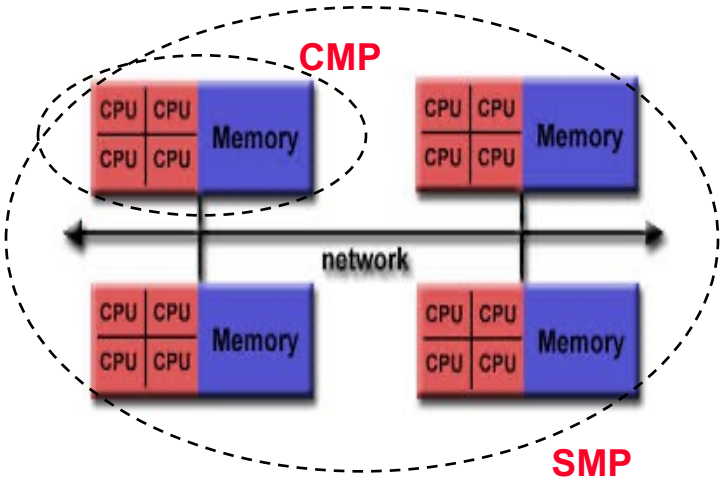
- Multiprocessor – a computer system with at least 2 processors
- Multicore (also known as CMP) – a chip (processor) with at least 2 cores



- Can deliver high throughput for multiple independent jobs – multiprogramming – via task-level or process-level parallelism
- Can improve the run time of a single program that has been specially crafted to run on a multiprocessor – a multithreaded (parallel) program

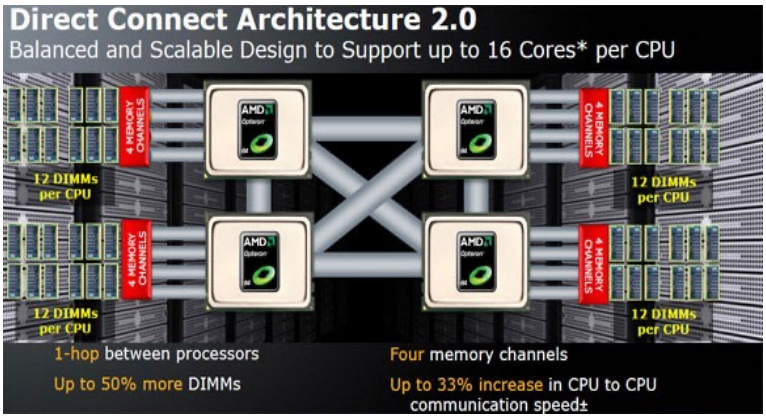
11

SMP (Shared Memory Multiprocessor) vs CMP (Multicore)



12

AMD SMP

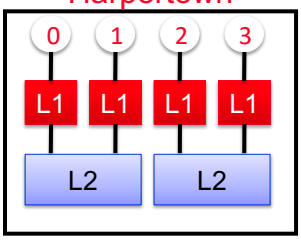


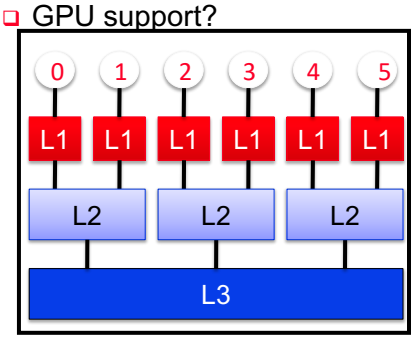
13

Multicore = Platform Variety

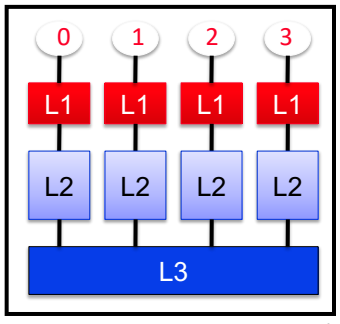
- Number/type of cores
- Levels and structure of the cache hierarchy
- Number of memory controllers (MCs) on-chip
- Type of on-chip interconnect
- GPU support?

Harpertown





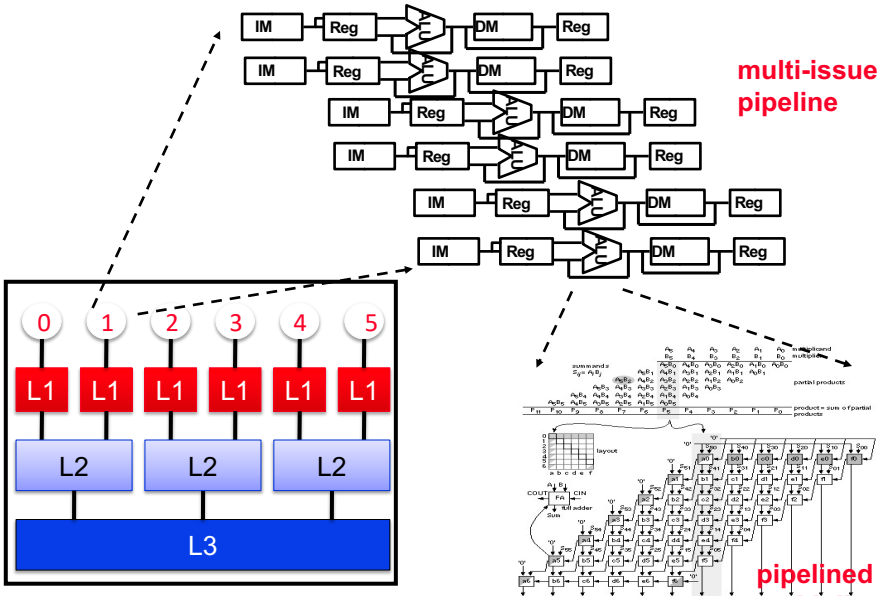
Dunnington



Nehalem

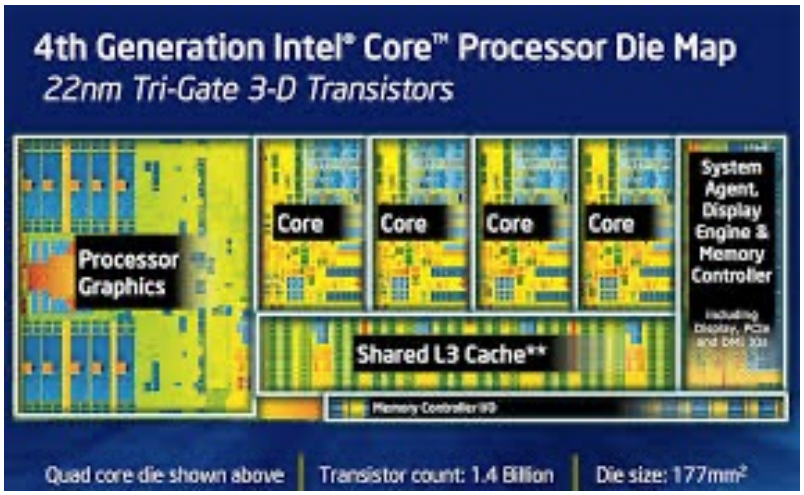
15

Hierarchical Parallelism



Intel's Haswell Multicore

- 4 cores + graphics co-processor, shared LLC, 2 MCs



17

The Trend

- Why not all cores on a single die?
  - Packaging difficulties
  - Scalability difficulties
  - Heating dissipation difficulties
  - Exceed reticle die area ~800 mm^2 (technology challenge)
- Why not many processor chips
  - Communication overheads
  - NUMA overheads
  - Programming difficulties
- Chiplet Design
  - Pioneering Chiplet Technology and Design for the AMD EPYC™ and Ryzen™ Processor Families. AMD Inc. (ISCA 2021)

18

Key Multiprocessor Design Questions

- Q1 – How do they share data?
- Q2 – How do they coordinate?
- Q3 – How scalable is the architecture? How many cores can be supported?

19

Shared Memory multiProcessor (SMP)

- SMPs come in two styles
  1. Uniform memory access (UMA) multiprocessors

     Accesses to main memory take about the same amount of time no matter which processor requests the access and no matter which word is asked for
  2. Nonuniform memory access (NUMA) multiprocessors

     some memory accesses are much faster than other depending on which processor asks for which word

     Programming NUMA is harder than programming UMA
- Q1 – Single physical address space shared by all cores
- Q2 – Threads on cores coordinate/communicate through shared variables in memory (via loads and stores)
  1. Use of shared data must be coordinated via synchronization primitives (locks) that allow access to data to only one core at a time (used to implement critical sections)
  2. Caches must be kept coherent (read of a data item returns the most recently written value of that data item)

20

SMP Multithreaded Programming Model

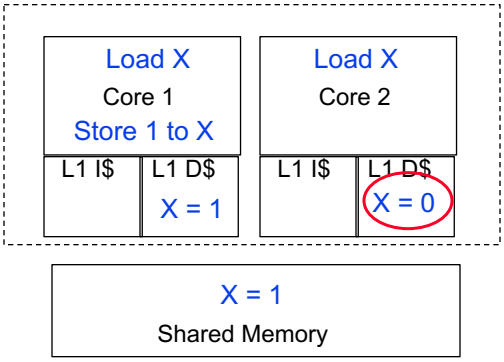
- Creating multithreaded programs for multicores ?
- Programmer explicitly creates multiple software threads
  - Java has thread support built in, C/C++ supports P-threads library
  - Other tools such as OpenMP, Thread Blocks, Cilk Plus
  - Speedup via Thread-Level Parallelism (TLP)
- All loads & stores are to a single **shared memory** space
- A “thread switch” can occur at any time
  - SMT, FGMT
  - Pre-emptive multithreading by OS
    - Hardware timer interrupt occasionally triggers OS
    - Quickly swapping threads gives illusion of concurrent execution

21

Cache Coherence in Multicores

- Illustration of the cache coherence problem in a two core processor with private L1I$ and L1D$ caches and a (common) shared memory.

1. Core 1 does Load X
2. Core 2 does Load X
3. Core 1 does Store 1 to X
   …
4. Core 1 does WB of X



22

A Coherent Memory System

- Any read of a data item should return the most recently written value of the data item
  - Coherence – defines what values can be returned by a read
    - Stores to the same location are serialized (two stores to the same location must be seen in the same order by all cores)
  - Consistency – determines when a written value will be returned by a read

Critical for performance

- To enforce coherence, caches must provide
  - Replication of shared data items in multiple cores' caches
    - Replication reduces both latency and contention for a read shared data item
  - Migration of shared data items to a core's local cache
    - Migration reduces the latency of the access the data and the bandwidth demand on the shared memory

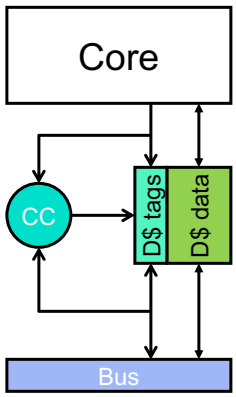
23

Cache Coherence Protocols

- Need a hardware mechanism to ensure cache coherence, the most popular of which is snooping
  - The cache controllers monitor (snoop) on the broadcast medium (e.g., bus) with duplicate address tag hardware (so they don't interfere with core's access to the cache) to determine if their cache has a copy of a block that is requested
- Write invalidate protocol – writes require exclusive access and invalidate all other existing copies
  - Exclusive access ensure that no other readable or writable copies of an item exists
- If two processors attempt to write the same data at the same time, one of them wins the race, causing the other core's copy to be invalidated. For the other core to complete, it must obtain the new data from the first core's cache – thus enforcing write serialization

24

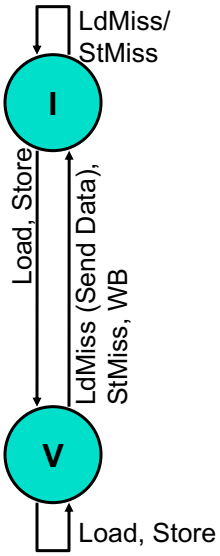
Hardware Cache Coherence



- Write-back caches (rather than write-through) for performance reasons
- Coherence Controller (CC)
  - Monitors ("snoops") bus traffic (addresses and data)
  - Executes the **coherence protocol**
    - What to do the with local copy when you see different things happening on bus
- Three core-initiated events
  - **Ld**: load **St**: store **WB**: write-back
- Two remote-initiated (bus) events
  - **LdMiss**: read miss from ***another*** core
  - **StMiss**: write miss from ***another*** core

25

Review: VI (MI) coherence protocol



- **Each cacheline has its own state machine**
- **VI (valid-invalid) protocol**:
  - Two states (per block in cache)
    - **V (valid)**: have block
    - **I (invalid)**: don't have block
    + Can implement with one bit (the valid bit)
- Protocol diagram (left)
  - If anyone **else** wants to read/write block
    - Give it up: transition to **I** state
  - WriteBack (WB) if your own copy is dirty
- This is an **write-invalidate** protocol
- **Write-update** protocol: copy data (write-through), don't invalidate
  - Sounds good, but wastes a lot of bus bandwidth

26

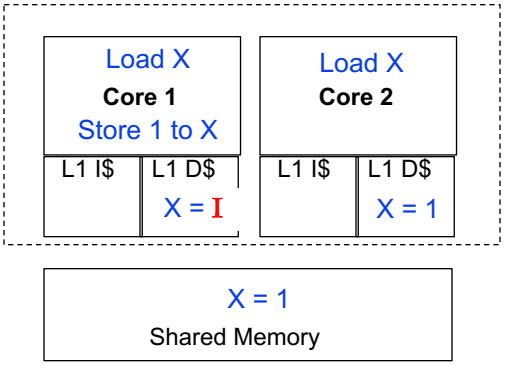
VI Protocol State Transition Table

State	*From This Core*		*From Other Cores*	
	Load	Store	Load Miss	Store Miss
Invalid (I)	Miss → V	Miss → V	---	---
Valid (V)	Hit	Hit	Send Data → I	→ I

- Rows are "states" (**I** vs **V**) of data blocks in the caches
- Columns are "events" in the cores
  - WriteBack events not shown
  - **V → I** on a LdMiss also updates memory (Send Data) **if** block is Dirty
- Memory sends data when no core responds

27

Example of VI Snooping Protocol



LdMiss seen by Core 1, so Core 1 invalidates its copy

StMiss seen by Core 2, so Core 2 invalidates its copy

LdMiss seen by Core 1, so Core 1 does a Send Data and then invalidates its copy

- When the second miss by Core 2 occurs, Core 1 responds with the data value for Core 2 (canceling the response from the shared memory), updating the value of X in the shared memory, and invalidating its own copy

28