CSCI-GA.3033-009
Multicore Processors:
Architecture & Programming

Lecture 3: Know Your Hardware...
You Cannot Ignore it!

Mohamed Zahran (aka Z)
mzahran@cs.nyu.edu
http://www.mzahran.com
Computer Technology

- **Memory**
  - DRAM capacity: 2x / 2 years (since '96)
    64x size improvement in last decade.

- **Processor**
  - Speed 2x / 1.5 years (since '85)
    100X performance in last decade

- **Traditional Disk Drive**
  - Capacity: 2x / 1 year (since '97)
    250X size in last decade
Memory Wall

“Moore’s Law”

Processor-Memory Performance Gap:
(grows 50% / year)

Most of the single core performance loss is on the memory system!
Memory

Von-Neumann Bottleneck

Processor

$
Two Main Program Characteristics

- **Temporal locality**
  - I used X
  - Most probably I will use it again soon

- **Spatial locality**
  - I used item number M
  - Most probably I will need item M+1 soon
Cache Analogy

- Hungry! must eat!
  - Option 1: go to refrigerator
    * Found → eat!
    * Latency = 1 minute
  - Option 2: go to store
    * Found → purchase, take home, eat!
    * Latency = 20-30 minutes
  - Option 3: grow food!
    * Plant, wait ... wait ... wait ..., harvest, eat!
    * Latency = ~250,000 minutes (~ 6 months)
Storage Hierarchy Technology

- Processor
  - SUPER FAST
  - SUPER EXPENSIVE
  - TINY CAPACITY

- CPU Cache
  - LEVEL 1 (L1) CACHE
  - LEVEL 2 (L2) CACHE
  - LEVEL 3 (L3) CACHE
  - FASTER
  - EXPENSIVE
  - SMALL CAPACITY

- Physical Memory
  - RANDOM ACCESS MEMORY (RAM)
  - FAST
  - PRICED REASONABLY
  - AVERAGE CAPACITY

- Solid State Memory
  - NON-VOLATILE FLASH-BASED MEMORY
  - AVERAGE SPEED
  - PRICED REASONABLY
  - AVERAGE CAPACITY

- Virtual Memory
  - FILE-BASED MEMORY
  - SLOW
  - CHEAP
  - LARGE CAPACITY

Simplified Computer Memory Hierarchy Illustration: Ryan J. Leng
Why Memory Wall?

• DRAMs not optimized for speed but for density (till now at least!)
• Off-chip bandwidth
• Increasing number of on-chip cores
  – Need to be fed with instructions and data
  – Big pressure on buses, memory ports, ...
Cache Memory: Yesterday

- Processor-Memory gap not very wide
- Simple cache (one or two levels)
- Inclusive
- Small size and associativity
Cache Memory: Today

- Wider Processor-Memory gap
- Two or three levels of cache hierarchy
- Larger size and associativity
- Inclusion property revisited
- Coherency
- Many optimizations
  - Dealing with static power
  - Dealing with soft-errors
  - Prefetching
  - ...

Cache Memory: Tomorrow

• Very wide processor-memory gap
• Multiple cache hierarchies (multi-core)
• On/Off chip bandwidths become bottleneck
• Scalability problem
• Technological constraints
  – Power
  – Variability
  – …
100s On-Chip Cores

• Technologically possible

• Near-future usage:
  – Massively parallel applications
    • Multithreading

• In the long run
  – Day to day use
    • Hybrid multithreading + multiprogramming
From Single Core to Multicore

• Currently mostly shared memory
  – This can change in the future
  – The “sharing” can be logical only (i.e. distributed shared memory)

• A new set of complications, in addition to what we already have 😞
  – Coherence
  – Consistency
Shared Memory Multicore

- Uniform
  - Uniform Cache Access
  - Uniform Memory Access

- Non-Uniform
  - Non-Uniform Cache Access
  - Non-Uniform Memory Access
Memory Model

• **Intuitive**: Reading from an address returns the most recent write to that address.
• This is what we find in uniprocessors
• For multicore, we call this: **sequential consistency**
  – Much harder and tricky to achieve
  – This is why we need **coherence**
Sequential Consistency Model

• Example:
  – P1 writes data=1, then writes flag=1
  – P2 waits until flag=1, then reads data

<table>
<thead>
<tr>
<th>If P2 reads flag</th>
<th>Then P2 may read data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Ensuring Consistency: Coherence Protocol

• Cache coherence needed in multicore processors to ensure consistency

• A memory system is coherent if:
  – P writes to X; no other processor writes to X; P reads X and receives the value previously written by P
  
  – P1 writes to X; no other processor writes to X; sufficient time elapses; P2 reads X and receives value written by P1
  
  – Two writes to the same location by two processors are seen in the same order by all processors - write serialization
Cache coherence

y0 privately owned by Core 0
y1 and z1 privately owned by Core 1

\[ x = 2; \quad /* \text{shared variable} */ \]

<table>
<thead>
<tr>
<th>Time</th>
<th>Core 0</th>
<th>Core 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>[ y0 = x; ]</td>
<td>[ y1 = 3x; ]</td>
</tr>
<tr>
<td>1</td>
<td>[ x = 7; ]</td>
<td>Statement(s) not involving ( x )</td>
</tr>
<tr>
<td>2</td>
<td>Statement(s) not involving ( x )</td>
<td>[ z1 = 4x; ]</td>
</tr>
</tbody>
</table>

y0 eventually ends up = 2
y1 eventually ends up = 6
z1 = ???
Snooping Cache Coherence

• The cores share a bus.

• Any signal transmitted on the bus can be “seen” by all cores connected to the bus.

• When core 0 updates the copy of x stored in its cache it also broadcasts this information across the bus.

• If core 1 is “snooping” the bus, it will see that x has been updated and it can mark its copy of x as invalid.
Directory Based Cache Coherence

- Uses a data structure called a directory that stores the status of each cache line.
- When a variable is updated, the directory is consulted, and the cache controllers of the cores that have that variable’s cache line in their caches are invalidated.
Cache Coherence Protocols

- Snoopy protocols
- Directory-Based protocols
  - Write invalidate
  - Write update
Example: MESI Protocol

PR = processor read
PW = processor write
BR = observed bus read
BW = observed bus write
S/~S = shared/NOT shared
The Future In Technology

• **Traditional**
  - SRAM
  - DRAM
  - Hard drives

• **New**
  - eDRAM
  - Flash
  - Solid-State Drive

• **Even Newer**
  (disruptive technology?)
  - M-RAM
  - STT-RAM
  - PCM
  - ... 

+ • 3D Stacking
  • Photonic interconnection
As A Programmer

• A parallel programmer is also a performance programmer: know your hardware.
• Your program does not execute on a vacuum.
• In theory, compilers understand memory hierarchy and can optimize your program;
  – In practice they don’t!!
• Even if compiler optimizes one algorithm, it won’t know about a different algorithm that might be a much better match to the processor
As A Programmer

• You don’t see the cache
  – But you feel it
• You see the disk and memory
  – So you can explicitly manage them
As A Programmer: Tools In Your Box

- Tiling
- Number of threads you spawn at any given time
- Thread granularity
- User thread scheduling
- Locality (both types)
- What is your performance metric?
  - Throughput
  - Latency
  - Bandwidth-delay product
- Best performance for a specific configuration

Vs Scalability
The Rest of This Lecture

• Get to know the design of some state-of-the-art processors
• Think about ways to exploit this hardware in your programs
• Compare how your program will look like if you did not know about the hardware
Data movement costs more than computation.
Your Parallel Program

- Threads
  - Granularity
  - How many?
- Thread types
  - Processing bound
  - Memory bound
- What to run? When? Where?
- Communication
- Degree of interaction
Processors We Will Look at

SPARC T4

IBM Power 7

Sandy Bridge die photo (courtesy of ISSCC).

Intel Sandy Bridge
SPARC T4

- The next generation of Oracle multicore
- 855M transistors
- Supports up to 64 threads
  - 8 cores
  - 8 threads per core
  - Cannot be deactivated by software
- Private L1 and L2 and shared L3
- Shared L3
  - Shared among 8 cores
  - Banked
  - 4MB
  - 16-way set associative
  - Line size of 64 bytes
PEU: PCI-Express unit;  DMU: Data management unit;  NIU: Network interface unit;  NCU: Non-cacheable unit;  SIU: System interface unit;  BoB: Buffer on Board
The Cores in SPARC T4
The Cores in SPARC T4

• Supports up to 8 threads
• DL1 and IL1:
  – 16KB
  – 4-way set associative
  – 32 bytes cache line
  – Shared by all 8 threads
• IL1 has 3 line prefetch on-miss
• DL1 has stride-based and next-line prefetchers
What to Do About Prefetching?

- Use arrays as much as possible. Lists, trees, and graphs have complex traversals which can confuse the prefetcher.
- Avoid long strides. Prefetchers detect strides only in a certain range because detecting longer strides requires a lot more hardware storage.
- If you must use a linked data structure, pre-allocate contiguous memory blocks for its elements and serve future insertions from this pool.
- Can you re-use nodes from your linked-list?
Questions

• Suppose that you have 8 threads that are processing bound and another 8 memory bound... how will you assign them to cores on T4?
• What if all threads are computation bound?
• What if they are all memory bound?
• T4 gives the software the ability to pause a thread for few cycles. When will you use this feature?
POWER7 Processors: The Beat Goes On

Approaching 20 Years of POWER Processors

Source: Slides from Joel M. Tendler from IBM
IBM Power 7

- Supports global shared memory space for POWER7 clusters
  - So you can program a cluster as if it were a single system
- Designed for power-efficiency, unlike Power 6
- ~1.2B Transistors
- Up to 8 cores and 4-way SMT
- TurboCore mode that can turn off half of the cores from an eight-core processor, but those 4 cores have access to all the memory controllers and L3 cache at increased clock speeds.
- 3.0 – 4.25 GHz clock speed
IBM Power 7: Cache Hierarchy

• 32KB DL1 and IL1 per core
• 256KB L2 per core
• eDRAM L3 4MB per core (total of 32MB)
  – Very flexible design for L3
Source: Slides from Joel M. Tendler from IBM
Questions

• If you are writing the same programs for T4 and Power 7, will you change anything?

• As a programmer, how can you make use of the cache hierarchy?
Intel Sandy Bridge

- New microarchitecture
- 22nm

Sandy Bridge, source: Intel
Intel Sandy Bridge

- New microarchitecture
- 22nm

Heterogeneous Multicore

Sandy Bridge, source: Intel
Intel Sandy Bridge

- Improvement over its predecessor Nehalem
- Targeting multimedia applications
  - Introduced Advanced Vector Extensions (AVX)
Features for You to Use

• Sandy bridge processors have 256bit wide vector units per core

• As a programmer you can:
  – Using AVX instructions
  – Use the compiler to vectorize your code
    • http://ispc.github.com/
TILERA: Many-core chips

- Released in August 2007.
- TILE64 offered 64 cores arranged in a 2-D grid.
- TILE-Gx8072 has 72 cores with a 2-D grid of communication channels called the iMesh Interconnect.
  - iMesh comes with five independent mesh networks that offer an aggregate bandwidth exceeding 110 Tbps.
- Each core has 32KB data and 32KB instruction L1 caches and 256KB L2 cache. A 18MB L3 coherent cache is shared between the cores. Access to the main RAM is done via four DDR3 controllers.
TILE-Gx8072 Block Diagram
Many-core : Intel's Xeon Phi

- Introduced in 2012.
- Offered as a co-processor.
- Runs in 2 of the top 10 supercomputers in the world, including the #1 machine, China's Tianhe-2.
- Equipped with 61 heavily customized Pentium cores.
- Each core:
  - Can handle four threads.
  - Includes a special 512-bit wide, Vector Processing Unit (VPU) that operates in SIMD mode to process 16 single precision or 8 double precision floating point numbers per clock cycle.
  - Comes equipped with 32KB data and 32KB instruction L1 caches and 512KB L2 coherent cache.
Intel's Xeon Phi, cont.

- Cores communicate over a bi-directional ring.
- Ring is made of six individual rings, three in each direction.
- Each direction has one 64-bytes wide data ring, and two narrower rings, an address ring (AD) and an acknowledgment ring (AK).
  - The AD ring is used to send read/write commands and memory addresses.
  - The AK ring is used for L2 cache coherence.
- The coherency is managed by distributed Tag Directories (TD), that contain information about every L2 cache line on the chip.
Intel's Xeon Phi Block Diagram

Source: Intel
Question: Can you design your program with different type of parallelism?
More Questions: Not Related to a Specific Processor

• Your code does not execute alone. Can you do something about it to avoid interference?

• As a programmer, what can you do about power?
Conclusions

• You need to know the big picture, at least
  – number of cores and SMT capability
  – Interconnection
  – Memory hierarchy
  – What is available to software and what is not

• The memory is a major bottleneck of performance.

• Interconnection is another bottleneck.

• Actual performance of program can be a complicated function of the architecture
  – Slight changes in the architecture or program change the performance significantly

• The art of delegation
  – What to do at user level and what to leave for the compiler, OS, and runtime