CSCI-GA.3033-010
Multicore Processors: Architecture & Programming

Lecture 3: Know Your Hardware

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.

- **Disk**
  - 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!
I am really sorry. Didn’t know that it would be that serious.
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
  - Processor Register
  - CPU Cache
    - Level 1 (L1) Cache
    - Level 2 (L2) Cache
    - Level 3 (L3) Cache
- Physical Memory
  - Random Access Memory (RAM)
- Solid State Memory
  - Non-volatile Flash-based Memory
- Virtual Memory
  - File-based Memory
- EDO, SD-RAM, DDR-SDRAM, RD-RAM and More...
- SSD, Flash Drive
- Mechanical Hard Drives

▲ 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:** The reading and 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
  – The memory consistency model defines “time elapsed” before the effect of a processor is seen by others
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

Required concurrency = Bandwidth * Latency

(over-simplification though)
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 the 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
- AMD 15h (Bulldozer)
- 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 computation intensive 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?
IBM Power 7

- Supports global shared memory space for POWER7 clusters
  - So you can program a cluster as if it were a single system
- Design 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
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
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)
• The GPU can access the large L3 cache
• Intel’s team totally re-designed the GPU
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/
Question: Can you design your program with different type of parallelism?
AMD 15h (Bulldozer)

- Designed from scratch
- 32nm technology
- combine the functions of what would normally be two discrete cores into a single package ("module" in AMD literature)
- The aim was to hit higher frequencies
- Performance worse than expected!
• According to AMD, Windows 7 doesn’t understand Bulldozer’s resource allocation very well.
  – Windows 7 sees 8 discrete cores
• Question: What can you do about this?
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.
• 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