Lecture 23: Parallelism

- Administration
  - Take QUIZ 17 over P&H 7.1-5, before 11:59pm today
  - Project: Cache Simulator, Due April 29, 2010
- Last Time
  - On chip communication
  - How I/O works
- Today
  - Where do architectures exploit parallelism?
  - What are the implications for programming models?
  - What are the implications for communication?
  - ... for caching?

Parallelism

- What type of parallelism do applications have?

- How can computer architectures exploit this parallelism?
Granularity of Parallelism

- Fine grain instruction level parallelism
- Fine grain data parallelism
- Coarse grain (data center) parallelism
- Multicore parallelism

Fine grain instruction level parallelism

- Fine grain instruction level parallelism
  - Pipelining
  - Multi-issue (dynamic scheduling & issue)
  - VLIW (static issue) - Very Long Instruction Word
    - Each VLIW instruction includes up to N RISC-like operations
    - Compiler groups up to N independent operations
    - Architecture issues fixed size VLIW instructions
VLIW Example

Let $N = 4$

Theoretically 4 ops/cycle
- $8 \text{ VLIW} = 32$ operations

In practice,
- Compiler cannot fill slots
- Memory stalls

in practice

Multithreading

- Performing multiple threads together
  - 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
- Medium-grain multithreading
  - Only switch on long stall (e.g., L2-cache miss)
  - Simplifies hardware, but doesn't hide short stalls such as data hazards
Simultaneous Multithreading

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

• Will it survive? In what form?
• Power considerations ⇒ simplified microarchitectures
  - Simpler forms of multithreading
• Tolerating cache-miss latency
  - Thread switch may be most effective
• Multiple simple cores might share resources more effectively

Granularity of parallelism

• Fine grain instruction level parallelism
  - Pipelining
  - Multi-issue (dynamic scheduling & issue)
  - VLIW (static issue)
  - Multithreading

• Fine grain data parallelism
  - Vector machines (CRAY)
  - SIMD (Single instruction multiple data)
  - Graphics cards
Example of Fine Grain Data Parallelism

Vectorization

Exploits parallelism in the data (not instructions!)
Fetch groups of data
Operate on them in parallel

```
for (i = 0; i < N; i++)        // original
  a(i) = a(i) + b(i) / 2

 // vectorized
for (i = 0; i < N; i+ vectorSize)
  VR1 = VectorLoad(a, i)
  VR2 = VectorLoad(b, i)
  VR! = VR1 + VR2 / 2
  VectorStore (a, vectorSize, VR1)
```

Granularity of Parallelism

- Fine grain instruction level parallelism
- Fine grain data parallelism
- Coarse grain (data center) parallelism
- Multicore medium grain parallelism
Coarse Grain Parallel Hardware
Data Centers & the Internet

Microsoft low power data center processors

Google: Cooling

Coarse Grain
Embarrassingly Parallel Software

• Lots and lots and lots of independent data & work
• Threads/tasks rarely or never communicate
• Map-Reduce model with Search example
  Idea: divide the Internet Search index into 1000 parts, & search each of the 1000 parts independently
  Process
  1. Type in a query
  2. Coordinator sends a message to search many independent data sources (e.g., 1000 different servers)
  3. A process searches, and sends a response message back to the coordinator
  4. Coordinator merges the results & returns answer
• Works great!
Granularity of Parallelism
Too Hot, Too Cold, or Just Right?

• Fine grain parallel architectures
  - Lots of instruction & data communication
    • Instructions/data all on the same chip
    • Communication latency is very low, order of cycles

• Coarse grain parallel architectures
  - Lots and lots of independent data & work
  - Rarely or never communicate, because communication is very, very expensive

• Multicore is betting there are applications with
  - Medium grain parallelism (i.e., threads, processes, tasks)
  - Not too much data, so it won’t swamp the memory system
  - Not too much communication (100 to 1000s of cycles across chip through the memory hierarchy)

Multicore Hardware
Intel Nehalem 4-core Processor

Per core:
32KB L1 I-cache, 32KB L1 D-cache, 512KB L2 cache
Shared across cores: 32MB L3
Shared Memory Multicore Programming Model

Example: Two threads running on two CPUs
Communicate implicitly by accessing shared global state
Example: both threads access shared variable X
Must synchronize accesses to X

```
thread 1              thread 2
withdraw(wdr) {
  lock(l)
  if X > wdr
    X = X - wdr
  bal = X
  unlock(l)
  return bal
}

deposit(dep) {
  lock(l)
  X = X + dep
  bal = X
  unlock(l)
}
```

Shared Memory Cache Coherence Problem

- Two threads share variable X
- Hardware
  - Two CPUs, write-through caches

<table>
<thead>
<tr>
<th>Time step</th>
<th>Event</th>
<th>CPU 1’s cache</th>
<th>CPU 2’s cache</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td>10</td>
</tr>
<tr>
<td>1</td>
<td>CPU 1 reads X</td>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>CPU 2 reads X</td>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>CPU 1 writes 1 to X</td>
<td>5</td>
<td>10</td>
<td>5</td>
</tr>
</tbody>
</table>

UTCS 352, Lecture 23
Coherence Defined

**Sequential Coherence**
Reads return most recently written value

**Formally**
- $P$ writes $X$; $P$ reads $X$ (no intervening writes)
  $\Rightarrow$ read returns written value
- $P_1$ writes $X$; $P_2$ reads $X$
  $\Rightarrow$ read returns written value
  - c.f. CPU 2 reads $X = 5$ after step 3 in example
- $P_1$ writes $X$, $P_2$ writes $X$
  $\Rightarrow$ all processors see writes in the same order
  - End up with the same final value for $X$

Remember Amdahl's Law

![Graph showing speedup vs. fraction of program that is parallel]

UTCS CS352 Lecture 3
Summary

• Parallelism
  - Granularities
  - Implications for programming models
  - Implications for communication
  - Implications for caches

• Next Time
  - More on multicore caches

• Reading: P&H P&H 7.6-13