Acknowledgement …

- Part of class notes are from
  - David Patterson
  - Electrical Engineering and Computer Sciences
  - University of California, Berkeley
  - http://www.eecs.berkeley.edu/~pattrsn
  - http://www-inst.eecs.berkeley.edu/~cs252

- And, from
  - Mary Jane Irwin
  - Computer Science and Engineering
  - Pennsylvania State University
  - http://www.cse.psu.edu/~mji
  - www.cse.psu.edu/~cg431
Outline

• Cray 1
• MP Motivation
• SISD v. SIMD v. MIMD
• Centralized vs. Distributed Memory
• Challenges to Parallel Programming
• Consistency, Coherency, Write Serialization
• Write Invalidate Protocol
• Example
• Conclusion

“The CRAY-1 computer system”

• by R.M. Russell, *Comm. of the ACM*, January 1978

• Number of functional units: 12 (today’s: 6~8)
• Clock rate: 80MHz (possible due to its small size)
• Memory size: 8 MB
• Memory latency: 50ns (faster than today’s PC)
• Cooling (4x per cubic inch of CDC 7600)
• Instruction set architecture: load/store, 3 address, 24-bit memory address
• Virtual Memory: No
“The CRAY-1 computer system”

- 138 MFLOPS, 250 MFLOPS peak (Pentium 4: 4 GFLOPS)

- The CRAY-1 weighs 10,500 lbs (10^6 more than today)
- The CRAY-1 consumes 115kW (10^3 of those today), and has a Freon cooling system
  - Power was limiting factor than as it is today

- How many Cray-1 were shipped?
  - 80 Cray-1s of all types were sold, priced from $5M to $8M

Outline

- Cray 1
- MP Motivation
- SISD v. SIMD v. MIMD
- Centralized vs. Distributed Memory
- Challenges to Parallel Programming
- Consistency, Coherency, Write Serialization
- Write Invalidate Protocol
- Example
- Conclusion
Déjà vu all over again?

“... today’s processors ... are nearing an impasse as technologies approach the speed of light...”


- Transputer had bad timing (Uniprocessor performance↑)
  ⇒ Procrastination rewarded: 2X seq. perf. / 1.5 years
- “We are dedicating all of our future product development to multicore designs. ... This is a sea change in computing”
  Paul Otellini, President, Intel (2005)
- All microprocessor companies switch to MP (2X CPUs / 2 yrs)
  ⇒ Procrastination penalized: 2X sequential perf. / 5 yrs

<table>
<thead>
<tr>
<th>Manufacturer/Year</th>
<th>AMD’05</th>
<th>Intel’06</th>
<th>IBM’04</th>
<th>Sun’05</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processors/chip</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td>Threads/Processor</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>Threads/chip</td>
<td>2</td>
<td>4</td>
<td>4</td>
<td>32</td>
</tr>
</tbody>
</table>
Applications Needing “Supercomputing”

• Energy (plasma physics (simulating fusion reactions), geophysical (petroleum) exploration)
• DoE stockpile stewardship (to ensure the safety and reliability of the nation’s stockpile of nuclear weapons)
• Earth and climate (climate and weather prediction, earthquake, tsunami prediction and mitigation of risks)
• Transportation (improving vehicles’ airflow dynamics, fuel consumption, crashworthiness, noise reduction)
• Bioinformatics and computational biology (genomics, protein folding, designer drugs)
• Societal health and safety (pollution reduction, disaster planning, terrorist action detection)

Other Factors ⇒ Multiprocessors

• Growth in data-intensive applications
  – Data bases, file servers, ...
• Growing interest in servers, server perf.
• Increasing desktop perf. less important
  – Outside of graphics
• Improved understanding in how to use multiprocessors effectively
  – Especially server where significant natural TLP
• Advantage of leveraging design investment by replication
  – Rather than unique design
Flynn’s Taxonomy

- Flynn classified by data and control streams in 1966

<table>
<thead>
<tr>
<th>Single Instruction Single Data (SISD) (Uniprocessor)</th>
<th>Single Instruction Multiple Data (SIMD) (single PC: Vector, CM-2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multiple Instruction Single Data (MISD) (????)</td>
<td>Multiple Instruction Multiple Data (MIMD) (Clusters, SMP servers)</td>
</tr>
</tbody>
</table>

- **SIMD** ⇒ Data Level Parallelism
- **MIMD** ⇒ Thread Level Parallelism
- MIMD popular because
  - Flexible: N pgms and 1 multithreaded pgm
  - Cost-effective: same MPU in desktop & MIMD

SIMD Processors

- Single control unit
- Multiple datapaths (processing elements – PEs) running in parallel
  - PEs are interconnected (usually via a mesh or torus) and exchange/share data as directed by the control unit
  - Each PE performs the same operation on its own local data
### Example SIMD Machines

<table>
<thead>
<tr>
<th>Maker</th>
<th>Year</th>
<th># PEs</th>
<th># b/PE</th>
<th>Max memory (MB)</th>
<th>PE clock (MHz)</th>
<th>System BW (MB/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Illiac IV</td>
<td>UIUC</td>
<td>1972</td>
<td>64</td>
<td>64</td>
<td>1</td>
<td>13</td>
</tr>
<tr>
<td>DAP</td>
<td>ICL</td>
<td>1980</td>
<td>4,096</td>
<td>1</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>MPP</td>
<td>Goodyear</td>
<td>1982</td>
<td>16,384</td>
<td>1</td>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>CM-2</td>
<td>Thinking Machines</td>
<td>1987</td>
<td>65,536</td>
<td>1</td>
<td>512</td>
<td>7</td>
</tr>
<tr>
<td>MP-1216</td>
<td>MasPar</td>
<td>1989</td>
<td>16,384</td>
<td>4</td>
<td>1024</td>
<td>25</td>
</tr>
</tbody>
</table>

### Centralized vs. Distributed Memory

- **Centralized Memory**
  - Symmetric MP (SMP)
  - UMA (Uniform memory access)

- **Distributed Memory**
  - NUMA (non-uniform memory access)
**2 Models for Communication and Memory Architecture**

1. Communication occurs by explicitly passing messages among the processors: **message-passing multiprocessors**

2. Communication 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

---

**Supercomputer Style Migration (Top500)**

![Graph showing migration](http://www.top500.org/lists/2005/11/)

- In the last 8 years uniprocessor and SIMDs disappeared while Clusters and Constellations grew from 3% to 80%
Challenges of Parallel Processing

• First challenge is % of program inherently sequential
• Suppose 80X speedup from 100 processors. What fraction of original program can be sequential?
  a. 10%
  b. 5%
  c. 1%
  d. <1%

Encountering Amdahl’s Law

• Speedup due to enhancement E is
  \[
  \text{Speedup w/ E} = \frac{\text{Exec time w/o E}}{\text{Exec time w/ E}}
  \]
• Suppose that enhancement E accelerates a fraction F (F <1) of the task by a factor S (S>1) and the remainder of the task is unaffected

\[
\text{ExTime w/ E} = \text{ExTime w/o E} \times \text{Speedup w/ E}
\]
**Encountering Amdahl’s Law**

- Speedup due to enhancement E is
  \[
  \text{Speedup w/ E} = \frac{\text{Exec time w/o E}}{\text{Exec time w/ E}}
  \]
- Suppose that enhancement E accelerates a fraction \( F \) (\( F < 1 \)) of the task by a factor \( S \) (\( S > 1 \)) and the remainder of the task is unaffected.
  \[
  \text{ExTime w/ E} = \text{ExTime w/o E} \times ((1-F) + \frac{F}{S})
  \]
  \[
  \text{Speedup w/ E} = \frac{1}{(1-F) + \frac{F}{S}}
  \]

**Challenges of Parallel Processing**

- First challenge is % of program inherently sequential
- Suppose 80X speedup from 100 processors. What fraction of original program can be sequential?
  a. 10%
  b. 5%
  c. 1%
  d. <1%
  
  \[
  \text{Speedup w/ E} = \frac{1}{(1-F) + \frac{F}{S}}
  \]
  
  For 80X speedup:
  \[
  80 = \frac{1}{(1-F) + \frac{F}{100}}
  \]
  \[
  F = 99.75%
  \]
  or, sequential = 0.25%
Challenges of Parallel Processing

- Second challenge is long latency to remote memory
- Suppose 32 CPU MP, 2GHz, 200 ns remote memory, all local accesses hit memory hierarchy and base CPI is 0.5. (Remote access = 200/0.5 = 400 clock cycles.)
- What is performance impact if 0.2% instructions involve remote access?
  a. 1.5X
  b. 2.0X
  c. 2.5X

CPI Equation

- CPI = Base CPI + Remote request rate x Remote request cost
- CPI = 0.5 + 0.2% x 400 = 0.5 + 0.8 = 1.3
- No communication is 1.3/0.5 or 2.6 faster than 0.2% instructions involve local access
### N/UMA Remote Memory Access Times (RMAT)

<table>
<thead>
<tr>
<th>Year</th>
<th>Type</th>
<th>Max Proc</th>
<th>Interconnection Network</th>
<th>RMA T (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1996</td>
<td>SMP</td>
<td>64</td>
<td>Address buses, data switch</td>
<td>500</td>
</tr>
<tr>
<td>1996</td>
<td>NUMA</td>
<td>2048</td>
<td>2-way 3D torus</td>
<td>300</td>
</tr>
<tr>
<td>1998</td>
<td>SMP</td>
<td>32</td>
<td>8 x 8 crossbar</td>
<td>1000</td>
</tr>
<tr>
<td>1999</td>
<td>NUMA</td>
<td>512</td>
<td>Fat tree</td>
<td>500</td>
</tr>
<tr>
<td>1999</td>
<td>SMP</td>
<td>32</td>
<td>Switched bus</td>
<td>400</td>
</tr>
<tr>
<td>2002</td>
<td>SMP</td>
<td>8</td>
<td>Switched bus</td>
<td>240</td>
</tr>
<tr>
<td>2003</td>
<td>SMP</td>
<td>64</td>
<td>Switched bus</td>
<td>275</td>
</tr>
<tr>
<td>2004</td>
<td>NUMA</td>
<td>10240</td>
<td>Fat tree</td>
<td>???</td>
</tr>
</tbody>
</table>

---

### Single Bus (Shared Address UMA) MP

- Caches are used to reduce latency and to lower bus traffic.
- Must provide hardware to ensure that caches and memory are consistent (cache coherency).
- Must provide a hardware mechanism to support process synchronization.
Example Cache Coherence Problem

- Processors see different values for $u$ after event 3
- With write back caches, value written back to memory depends on happenstance of which cache flushes or writes back value when
  - Processes accessing main memory may see very stale value
- Unacceptable for programming, and its frequent!

Example Memory Consistency Problem

- Intuition not guaranteed by coherence
- Expect memory to respect order between accesses to different locations issued by a given process
  - To preserve orders among accesses to same location by different processes
- Coherence is not enough!
  - Pertains only to single location

```
P1
/* Assume initial value of A and flag is 0*/
A = 1; while (flag == 0); /*spin idly*/
flag = 1; print A;
```

```
P2
```

```
Conceptual Picture
Mem
R1
R2
```
Intuitive Memory Model

- Too vague and simplistic; 2 issues
  1. **Coherence** defines values returned by a read
  2. **Consistency** determines when a written value will be returned by a read
- Coherence defines behavior to same location, Consistency defines behavior to other locations

---

Defining Coherent Memory System

1. **Preserve Program Order:**
   P writes A to location X \( \Rightarrow \) P reads X: returns A  
   (with no writes of X by another processor occurring between the write and the read by P)

2. **Coherent view of memory:**
   Q writes A to location X \( \Rightarrow \) P reads X: returns A  
   (if the read and write are sufficiently separated in time and no other writes to X occur between the two)

3. **Write serialization:**
   P writes A to X, Q writes B to X \( \Rightarrow \) all other processors see the same order of writes
Write Consistency

• For now assume
  1. A write does not complete (and allow the next write to occur) until all processors have seen the effect of that write
  2. The processor does not change the order of any write with respect to any other memory access

⇒ if a processor writes location A followed by location B, any processor that sees the new value of B must also see the new value of A

• These restrictions allow the processor to reorder reads, but forces the processor to finish writes in program order

* See Section 4.6

2 Classes of Cache Coherence Protocols

1. Directory based — Sharing status of a block of physical memory is kept in just one location, the directory

2. Snooping — Every cache with a copy of data also has a copy of sharing status of block, but no centralized state is kept
   • All caches are accessible via some broadcast medium (a bus or switch)
   • All cache controllers monitor or snoop on the medium to determine whether or not they have a copy of a block that is requested on a bus or switch access
Multiprocessor Cache Coherency

- Cache coherency protocols
  - Bus snooping – cache controllers monitor shared bus traffic with duplicate address tag hardware (so they don’t interfere with processor’s access to the cache)

Bus Snooping Protocols

- Multiple copies are not a problem when reading
- Processor must have exclusive access to write a word
- All other processors sharing that data must be informed of writes
Handling Writes

Ensuring that all other processors sharing data are informed of writes can be handled two ways:

1. **Write-update** (write-broadcast) – writing processor broadcasts new data over the bus, all copies are updated
   - All writes go to the bus → higher bus traffic
   - Since new values appear in caches sooner, can reduce latency

2. **Write-invalidate** – writing processor issues invalidation signal on bus, cache snoops check to see if they have a copy of the data, if so they invalidate their cache block containing the word (this allows multiple readers but only one writer)
   - Uses the bus only on the first write → lower bus traffic, so better use of bus bandwidth

Example: Write-invalidate

- Must invalidate before step 3
- Write update uses more broadcast medium BW ⇒ all recent MPUs use write invalidate
Architectural Building Blocks

- **Cache block state transition diagram**
  - FSM specifying how disposition of block changes
    - invalid, valid, dirty
- **Broadcast Medium Transactions (e.g., bus)**
  - Fundamental system design abstraction
  - Logically single set of wires connect several devices
  - Protocol: arbitration, command/addr, data
    - Every device observes every transaction
- **Broadcast medium enforces serialization of read or write accesses ⇒ Write serialization**
  - 1st processor to get medium invalidates others copies
  - Implies cannot complete write until it obtains bus
  - All coherence schemes require serializing accesses to same cache block
- Also need to find up-to-date copy of cache block

Ordering

- Writes establish a partial order
- Doesn’t constrain ordering of reads, though shared-medium (bus) will order read misses too
  - any order among reads between writes is fine, as long as in program order
• Add states and state transition in uniprocessor systems
  – They must have CPU request & block replacement
  – But they don’t have bus request

Example Write-invalidate Snoopy Protocol

• Invalidation protocol, write-back cache
  – Snoops every address on bus
  – If it has a dirty copy of requested block, provides that block in response to the read request and aborts the memory access
• Each memory block is in one state:
  – Clean in all caches and up-to-date in memory (Shared)
  – OR Dirty in exactly one cache (Exclusive)
  – OR Not in any caches
• Each cache block is in one state (track these):
  – Shared: block can be read
  – OR Exclusive: cache has only copy, its writeable, and dirty
  – OR Invalid: block contains no data (in uniprocessor cache too)
• Read misses: cause all caches to snoop bus
• Writes to clean blocks are treated as misses
State Machine – CPU Requests

- State machine for CPU requests for each cache block
- Non-resident blocks invalid

Cache Block State

- Invalid
- Shared (read/only)
- Cache Block State
- Exclusive (read/write)

CPU Read hit

CPU Read
Place read miss on bus

CPU Write
Place Write Miss on bus

CPU Read miss
Write back block, Place read miss on bus

CPU Write Miss
Write back cache block
Place write miss on bus

State Machine - Block-replacement

- State machine for CPU requests for each cache block

Cache Block State

- Invalid
- Shared (read/only)

CPU Read hit

CPU Read
Place read miss on bus

CPU Write
Place Write Miss on bus

CPU Read miss
Write back block, Place read miss on bus

CPU Write Miss
Write back cache block
Place write miss on bus
State Machine – All CPU Requests

- State machine for **CPU** requests for each cache block

  - Invalid
  - Shared (read/only)
  - Exclusive (read/write)
  - CPU Read
    - Place read miss on bus
  - CPU Read hit
  - CPU Write
    - Place Write Miss on bus
  - CPU Write Miss
    - Write back cache block
    - Place write miss on bus

  Cache Block State

State Machine - Bus Request

- State machine for **bus** requests for each cache block

  - Invalid
  - Shared (read/only)
  - Exclusive (read/write)
  - Write miss for this block
    - Write Back Block; (abort memory access)
  - Read miss for this block
    - Write Back Block; (abort memory access)
State Machine - Summary

- State machine for CPU requests for each cache block and for bus requests for each cache block

![State Machine Diagram]

Write-Invalidate CC Examples

- I = invalid, S = shared, E = exclusive (only one)

1. Read hit for A

   Proc 1
   A
   E

   Proc 2
   A
   I

   Main Mem
   A

2. Reads A

   Proc 1
   A
   E

   Proc 2
   A
   I

   Main Mem
   A
Write-Invalidate CC Examples

– I = invalid, S = shared, E = exclusive (only one)
1. Read hit for A
   
   Proc 1
   A  E
   
   Proc 2
   A  I
   
   Main Mem
   A
   
2. Reads A
3. Snoop sees read request for A, writes back A to MM
4. Read miss for A
5. Changes its state to S

2010-11-02
Write-Invalidate CC Examples

- I = invalid, S = shared, E = exclusive (only one)

1. Read hit for A
   - Proc 1: A
   - Proc 2: E
   - Main Mem: A

2. Reads A
   - Proc 1: A
   - Proc 2: I
   - Main Mem: A

3. Read request for A
   - Proc 1: A
   - Proc 2: I
   - Main Mem: A

4. Gets A from MM & changes its state to E
   - Proc 1: A
   - Proc 2: S
   - Main Mem: A

5. Changes its state to S
   - Proc 1: A
   - Proc 2: I
   - Main Mem: A

Other Coherence Protocols

- There are many variations on cache coherence protocols

- Another write-invalidate protocol used in the Pentium 4 (and many other micro’s) is MESI with four states:
  - Modified – same
  - Exclusive – only one copy of the shared data is allowed to be cached; memory has an up-to-date copy
    - Since there is only one copy of the block, write hits don’t need to send invalidate signal
  - Shared – multiple copies of the shared data may be cached (i.e., data permitted to be cached with more than one processor); memory has an up-to-date copy
  - Invalid – same
**Mes Agility Coherency Protocol**

- **Invalid (not valid block)**
  - Processor write miss
  - [Send invalidate signal]
  - Processor exclusive read miss
  - Processor exclusive read

- **Shared (clean)**
  - Processor exclusive read

- **Exclusive (clean)**
  - Processor exclusive read miss
  - [Write back block]

- **Modified (dirty)**
  - Processor write miss
  - [Write back block]

- **Processor write or read hit**
  - Processor write

- **Another processor has read/write miss for this block**

**Conclusions**

- “End” of uniprocessors speedup ⇒ Multiprocessors
- Parallelism challenges: % parallelizable, long latency to remote memory
- Centralized vs. distributed memory
  - Small MP vs. lower latency, larger BW for Larger MP
- Message Passing vs. Shared Address
  - Uniform access time vs. Non-uniform access time
- Snooping cache over shared medium for smaller MP by invalidating other cached copies on write
- Sharing cached data ⇒ Coherence (values returned by a read), Consistency (when a written value will be returned by a read)
- Shared medium serializes writes ⇒ Write consistency