Multiprocessors

Key questions:
• How do parallel processors share data?
• How do parallel processors coordinate?
• How many processors?

Two main approaches to sharing data:
• Shared memory (single address space)
  • Synchronization
  • Uniform/nonuniform memory access
• Message passing
• Clusters (processors connected via LAN)

Two types of connection:
• Single bus
• Network

<table>
<thead>
<tr>
<th>Category</th>
<th>Choice</th>
<th>Number of processors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sharing data</td>
<td>Shared memory</td>
<td>UMA 2 – 64</td>
</tr>
<tr>
<td></td>
<td></td>
<td>NUMA 8 – 256</td>
</tr>
<tr>
<td></td>
<td>Message passing</td>
<td>8 – 256</td>
</tr>
<tr>
<td>Connection</td>
<td>Network</td>
<td>8 – 256</td>
</tr>
<tr>
<td></td>
<td>Bus</td>
<td>2 – 32</td>
</tr>
</tbody>
</table>
Programming multiprocessors

Multiprogramming is difficult:
  • Communication problems
  • Requires knowledge about the hardware
  • All parts of the program should be parallelized
Multiprocessors connected by a single bus

- Each processor is smaller than a multichip processor
- The use of caches can reduce the bus traffic
- There exists mechanisms to keep caches and memory consistent
Parallel program

Shared data: A[100000], sum[10]
Private data: i, half

sum[Pn]=0;
for (I=10000*Pn; I<10000*(Pn+1); I++)
    sum[Pn]=sum[Pn]+A[I];

half=10
repeat
    sync(); /* wait for partial sum completion - barrier synchronization */
    if (half%2!=0 && Pn==0)
        sum[0]=sum[0]+sum[half-1];
    half=half/2;
    if (Pn<half) sum[Pn]=sum[Pn]+sum[Pn+half];
until (half==1);
Multiprocessor cache coherency

Snooping (monitoring) protocols: locate all caches that share a block to be written. Then:

- **Write-invalid**: The writing processor causes all copies in other caches to be invalidated before changing its local copy. Similar to write-back.
- **Write-update (broadcast)**: The write processor sends the new data (the word) over the bus. Similar to write-through.
- The role of the size of the block (broadcasting only a word, false sharing).
Write-invalidate cache coherency protocol based on a write-back policy

Each cache block is in one of the following states:
- **Read only**: the block is not written and may be shared
- **Read/Write**: the block is written (dirty) and may not be shared
- **Invalid**: the block does not have valid data
Synchronization using coherency

- Using locks (semaphores)
- Atomic swap operation

<table>
<thead>
<tr>
<th>Step</th>
<th>Processor P0</th>
<th>Processor P1</th>
<th>Processor P2</th>
<th>Bus activity</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Has lock</td>
<td>Spins, testing if lock = 0</td>
<td>Spins, testing if lock = 0</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Sets lock to 0; sends invalidate over bus</td>
<td>Spins, testing if lock = 0</td>
<td>Spins, testing if lock = 0</td>
<td>Write-invalidate of lock variable sent from P0</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Cache miss</td>
<td>Cache miss</td>
<td></td>
<td>Bus services P2’s cache miss</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Responds to P2’s cache miss; sends lock = 0</td>
<td>(waits for cache miss)</td>
<td>(waits for cache miss)</td>
<td>Response to P2’s cache miss</td>
<td>Update memory with block from P0</td>
</tr>
<tr>
<td>5</td>
<td>(waits for cache miss)</td>
<td>Tests lock = 0; succeeds</td>
<td></td>
<td>Bus services P1’s cache miss</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Tests lock = 0; succeeds</td>
<td>Attempt swap; needs write permission</td>
<td>Response to P1’s cache miss</td>
<td>Responds to P1’s cache miss</td>
<td>sends lock variable</td>
</tr>
<tr>
<td>7</td>
<td>Attempt swap; needs write permission</td>
<td>Send invalidate to gain write permission</td>
<td>Bus services P2’s invalidate</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Cache miss</td>
<td>Swap; reads lock = 0 and sets to 1</td>
<td>Bus services P1’s cache miss</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Swap; read lock = 1 sets to 1; go back to spin</td>
<td>Responds to P1’s cache miss, sends lock = 1</td>
<td>Response to P2’s cache miss</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**FIGURE 9.3.6 Cache coherence steps and bus traffic for three processors, P0, P1, and P2.** This figure assumes write-invalidate coherency. P0 starts with the lock (step 1). P0 exits and unlocks the lock (step 2). P1 and P2 race to see which reads the unlocked value during the swap (steps 3–5). P2 wins and enters the critical section (steps 6 and 7), while P1 spins and waits (steps 7 and 8). The “critical section” is the name for the code between the lock and the unlock. When P2 exits the critical section, it sets the lock to 0, which will invalidate the copy in P1’s cache, restarting the process.