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>UMA</td>
<td>2 – 64</td>
</tr>
<tr>
<td></td>
<td>NUMA</td>
<td>8 – 256</td>
</tr>
<tr>
<td>Message passing</td>
<td></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-invalidate**: 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>P1</th>
<th>P2</th>
<th>P3</th>
<th>Bus activity</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Has lock</td>
<td>Spins, testing lock=0</td>
<td>Spins, testing lock=0</td>
<td>none</td>
</tr>
<tr>
<td>2</td>
<td>Sets lock to 0 and 0 sent over bus</td>
<td>Spins, testing lock=0</td>
<td>Spins, testing lock=0</td>
<td>Write-invalidate of lock variable sent from P0</td>
</tr>
<tr>
<td>3</td>
<td>Cache miss</td>
<td>Cache miss</td>
<td></td>
<td>Bus decides to service P2 cache miss</td>
</tr>
<tr>
<td>4</td>
<td>Waits (bus busy)</td>
<td>Lock=0</td>
<td></td>
<td>Cache miss for P2 satisfied</td>
</tr>
<tr>
<td>5</td>
<td>Lock =0</td>
<td>Swap: reads locks and sets to 1</td>
<td></td>
<td>Cache miss for P1 satisfied</td>
</tr>
<tr>
<td>6</td>
<td>Swap: reads locks and sets to 1</td>
<td>Value from swap=0 and 1 sent over bus</td>
<td></td>
<td>Write-invalidate of lock variable sent from P2</td>
</tr>
<tr>
<td>7</td>
<td>Value from swap=1 and 1 sent over bus</td>
<td>Owns the locks and updates the shared data</td>
<td></td>
<td>Write-invalidate of lock variable sent from P1</td>
</tr>
<tr>
<td>8</td>
<td>Spins, testing lock=0</td>
<td></td>
<td></td>
<td>None</td>
</tr>
</tbody>
</table>