Lecture 19: Shared-Memory Multiprocessors

- Topics: coherence protocols for symmetric shared-memory multiprocessors (Sections 4.1-4.2)
Procedure Solve(A)
begin
diff = done = 0;
while (!done) do
  diff = 0;
  for i ← 1 to n do
    for j ← 1 to n do
      temp = A[i,j];
      A[i,j] ← 0.2 * (A[i,j] + neighbors);
      diff += abs(A[i,j] – temp);
    end for
  end for
  if (diff < TOL) then done = 1;
end while
end procedure
procedure Solve(A)
    int i, j, pid, done=0;
    float temp, mydiff=0;
    int mymin = 1 + (pid * n/procs);
    int mymax = mymin + n/nprocs -1;
    while (!done) do
        mydiff = diff = 0;
        BARRIER(bar1,nprocs);
        for i  mymin to mymax
            for j  1 to n do
                ...
            endfor
        endfor
        LOCK(diff_lock);
        diff += mydiff;
        UNLOCK(diff_lock);
        BARRIER (bar1, nprocs);
        if (diff < TOL) then done = 1;
        BARRIER (bar1, nprocs);
    endwhile

main()
begin
    read(n); read(nprocs);
    A  G_MALLOC();
    initialize (A);
    CREATE (nprocs,Solve,A);
    WAIT_FOR_END (nprocs);
end main
**Message Passing Model**

```c
main()
read(n); read(nprocs);
CREATE (nprocs-1, Solve);
Solve();
WAIT_FOR_END (nprocs-1);

procedure Solve()
int i, j, pid, nn = n/nprocs, done=0;
float temp, tempdiff, mydiff = 0;
myA ← malloc(…)
initialize(myA);
while (!done) do
    mydiff = 0;
    if (pid != 0)
        SEND(&myA[1,0], n, pid-1, ROW);
    if (pid != nprocs-1)
        SEND(&myA[nn,0], n, pid+1, ROW);
    if (pid != 0)
        RECEIVE(&myA[0,0], n, pid-1, ROW);
    if (pid != nprocs-1)
        RECEIVE(&myA[nn+1,0], n, pid+1, ROW);
    for i ← 1 to nn do
        for j ← 1 to n do
            …
        endfor
    endfor
    if (pid != 0)
        SEND(mydiff, 1, 0, DIFF);
    RECEIVE(done, 1, 0, DONE);
    else
        for i ← 1 to nprocs-1 do
            RECEIVE(tempdiff, 1, *, DIFF);
            mydiff += tempdiff;
        endfor
    endif
    if (mydiff < TOL) done = 1;
    for i ← 1 to nprocs-1 do
        SEND(done, 1, I, DONE);
    endfor
endwhile
```
Shared-Memory Vs. Message-Passing

Shared-memory:
• Well-understood programming model
• Communication is implicit and hardware handles protection
• Hardware-controlled caching

Message-passing:
• No cache coherence → simpler hardware
• Explicit communication → easier for the programmer to restructure code
• Sender can initiate data transfer
SMPs or Centralized Shared-Memory
Distributed Memory Multiprocessors
SMPs

- Centralized main memory and many caches $\rightarrow$ many copies of the same data

- A system is cache coherent if a read returns the most recently written value for that word

<table>
<thead>
<tr>
<th>Time</th>
<th>Event</th>
<th>Value of X in Cache-A</th>
<th>Cache-B</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>-</td>
<td>-</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>CPU-A reads X</td>
<td>1</td>
<td>-</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>CPU-B reads X</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>CPU-A stores 0 in X</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Cache Coherence

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
Cache Coherence Protocols

• Directory-based: A single location (directory) keeps track of the sharing status of a block of memory

• Snooping: Every cache block is accompanied by the sharing status of that block – all cache controllers monitor the shared bus so they can update the sharing status of the block, if necessary

  ➢ Write-invalidate: a processor gains exclusive access of a block before writing by invalidating all other copies
  ➢ Write-update: when a processor writes, it updates other shared copies of that block
Design Issues

- Invalidate
- Find data
- Writeback / writethrough

- Cache block states
- Contention for tags
- Enforcing write serialization
SMP Example

Processor A
Caches

Processor B
Caches

Processor C
Caches

Processor D
Caches

Main Memory

I/O System

A: Rd X
B: Rd X
C: Rd X
A: Wr X
A: Wr X
C: Wr X
B: Rd X
A: Rd X
A: Rd Y
B: Wr X
B: Rd Y
B: Wr X
B: Wr Y
## SMP Example

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>A:</td>
<td>Rd X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B:</td>
<td>Rd X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C:</td>
<td>Rd X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A:</td>
<td>Wr X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A:</td>
<td>Wr X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C:</td>
<td>Wr X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B:</td>
<td>Rd X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A:</td>
<td>Rd X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A:</td>
<td>Rd Y</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A:</td>
<td>Rd Y</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B:</td>
<td>Wr X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B:</td>
<td>Wr X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B:</td>
<td>Wr Y</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B:</td>
<td>Wr Y</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### SMP Example

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>A: Rd X</td>
<td>S</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B: Rd X</td>
<td>S</td>
<td>S</td>
<td></td>
</tr>
<tr>
<td>C: Rd X</td>
<td>S</td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td>A: Wr X</td>
<td>E</td>
<td>I</td>
<td>I</td>
</tr>
<tr>
<td>A: Wr X</td>
<td>E</td>
<td>I</td>
<td>I</td>
</tr>
<tr>
<td>C: Wr X</td>
<td>I</td>
<td>I</td>
<td>E</td>
</tr>
<tr>
<td>B: Rd X</td>
<td>I</td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td>A: Rd X</td>
<td>S</td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td>A: Rd Y</td>
<td>S (Y)</td>
<td>S (X)</td>
<td>S (X)</td>
</tr>
<tr>
<td>B: Wr X</td>
<td>S (Y)</td>
<td>E (X)</td>
<td>I</td>
</tr>
<tr>
<td>B: Rd Y</td>
<td>S (Y)</td>
<td>S (Y)</td>
<td>I</td>
</tr>
<tr>
<td>B: Wr X</td>
<td>S (Y)</td>
<td>E (X)</td>
<td>I</td>
</tr>
<tr>
<td>B: Wr Y</td>
<td>I</td>
<td>E (Y)</td>
<td>I</td>
</tr>
</tbody>
</table>
# Example Protocol

<table>
<thead>
<tr>
<th>Request</th>
<th>Source</th>
<th>Block state</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read hit</td>
<td>Proc</td>
<td>Shared/excl</td>
<td>Read data in cache</td>
</tr>
<tr>
<td>Read miss</td>
<td>Proc</td>
<td>Invalid</td>
<td>Place read miss on bus</td>
</tr>
<tr>
<td>Read miss</td>
<td>Proc</td>
<td>Shared</td>
<td>Conflict miss: place read miss on bus</td>
</tr>
<tr>
<td>Read miss</td>
<td>Proc</td>
<td>Exclusive</td>
<td>Conflict miss: write back block, place read miss on bus</td>
</tr>
<tr>
<td>Write hit</td>
<td>Proc</td>
<td>Exclusive</td>
<td>Write data in cache</td>
</tr>
<tr>
<td>Write hit</td>
<td>Proc</td>
<td>Shared</td>
<td>Place write miss on bus</td>
</tr>
<tr>
<td>Write miss</td>
<td>Proc</td>
<td>Invalid</td>
<td>Place write miss on bus</td>
</tr>
<tr>
<td>Write miss</td>
<td>Proc</td>
<td>Shared</td>
<td>Conflict miss: place write miss on bus</td>
</tr>
<tr>
<td>Write miss</td>
<td>Proc</td>
<td>Exclusive</td>
<td>Conflict miss: write back, place write miss on bus</td>
</tr>
<tr>
<td>Read miss</td>
<td>Bus</td>
<td>Shared</td>
<td>No action; allow memory to respond</td>
</tr>
<tr>
<td>Read miss</td>
<td>Bus</td>
<td>Exclusive</td>
<td>Place block on bus; change to shared</td>
</tr>
<tr>
<td>Write miss</td>
<td>Bus</td>
<td>Shared</td>
<td>Invalidate block</td>
</tr>
<tr>
<td>Write miss</td>
<td>Bus</td>
<td>Exclusive</td>
<td>Write back block; change to invalid</td>
</tr>
</tbody>
</table>
Coherence Protocols

• Two conditions for cache coherence:
  ➢ write propagation
  ➢ write serialization

• Cache coherence protocols:
  ➢ snooping
  ➢ directory-based

  ➢ write-update
  ➢ write-invalidate
Title

• Bullet