Lecture 17: Multiprocessors

- Topics: multiprocessor intro and taxonomy, symmetric shared-memory multiprocessors (Sections 4.1-4.2)
Taxonomy

• SISD: single instruction and single data stream: uniprocessor

• MISD: no commercial multiprocessor: imagine data going through a pipeline of execution engines

• SIMD: vector architectures: lower flexibility

• MIMD: most multiprocessors today: easy to construct with off-the-shelf computers, most flexibility
Memory Organization - I

- Centralized shared-memory multiprocessor or Symmetric shared-memory multiprocessor (SMP)

- Multiple processors connected to a single centralized memory – since all processors see the same memory organization → uniform memory access (UMA)

- Shared-memory because all processors can access the entire memory address space

- Can centralized memory emerge as a bandwidth bottleneck? – not if you have large caches and employ fewer than a dozen processors
SMPs or Centralized Shared-Memory
Memory Organization - II

• For higher scalability, memory is distributed among processors → distributed memory multiprocessors.

• If one processor can directly address the memory local to another processor, the address space is shared → distributed shared-memory (DSM) multiprocessor.

• If memories are strictly local, we need messages to communicate data → cluster of computers or multicomputers.

• Non-uniform memory architecture (NUMA) since local memory has lower latency than remote memory.
Distributed Memory Multiprocessors
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 $\rightarrow$ simpler hardware
- Explicit communication $\rightarrow$ easier for the programmer to restructure code
- Sender can initiate data transfer
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
Message Passing Model

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
      if (mydiff < TOL) done = 1;
      for i ← 1 to nprocs-1 do
         SEND(done, 1, I, DONE);
      endfor
   endif
endwhile
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
## 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>
Title

• Bullet