Lecture: Memory, Coherence Protocols

- Topics: wrap-up of memory systems, intro to multi-thread programming models

Refresh

- Every DRAM cell must be refreshed within a 64 ms window
- A row read/write automatically refreshes the row
- Every refresh command performs refresh on a number of rows, the memory system is unavailable during that time
- A refresh command is issued by the memory controller once every 7.8us on average

Problem 5

- Consider a single 4 GB memory rank that has 8 banks. Each row in a bank has a capacity of 8KB. On average, it takes 40ns to refresh one row. Assume that all 8 banks can be refreshed in parallel. For what fraction of time will this rank be unavailable? How many rows are refreshed with every refresh command?

Problem 5

- Consider a single 4 GB memory rank that has 8 banks. Each row in a bank has a capacity of 8KB. On average, it takes 40ns to refresh one row. Assume that all 8 banks can be refreshed in parallel. For what fraction of time will this rank be unavailable? How many rows are refreshed with every refresh command?

  The memory has 4GB/8KB = 512K rows
  There are 8K refresh operations in one 64ms interval.
  Each refresh operation must handle 512K/8K = 64 rows
  Each bank must handle 8 rows
  One refresh operation is issued every 7.8us and the memory is unavailable for 320ns, i.e., for 4% of time.

Address Mapping Policies

- Consecutive cache lines can be placed in the same row to boost row buffer hit rates
- Consecutive cache lines can be placed in different ranks to boost parallelism
- Example address mapping policies:
  row:rank:bank:channel:column:blkoffset

  row:column:rank:bank:channel:blkoffset

Reads and Writes

- A single bus is used for reads and writes
- The bus direction must be reversed when switching between reads and writes; this takes time and leads to bus idling
- Hence, writes are performed in bursts; a write buffer stores pending writes until a high water mark is reached
- Writes are drained until a low water mark is reached

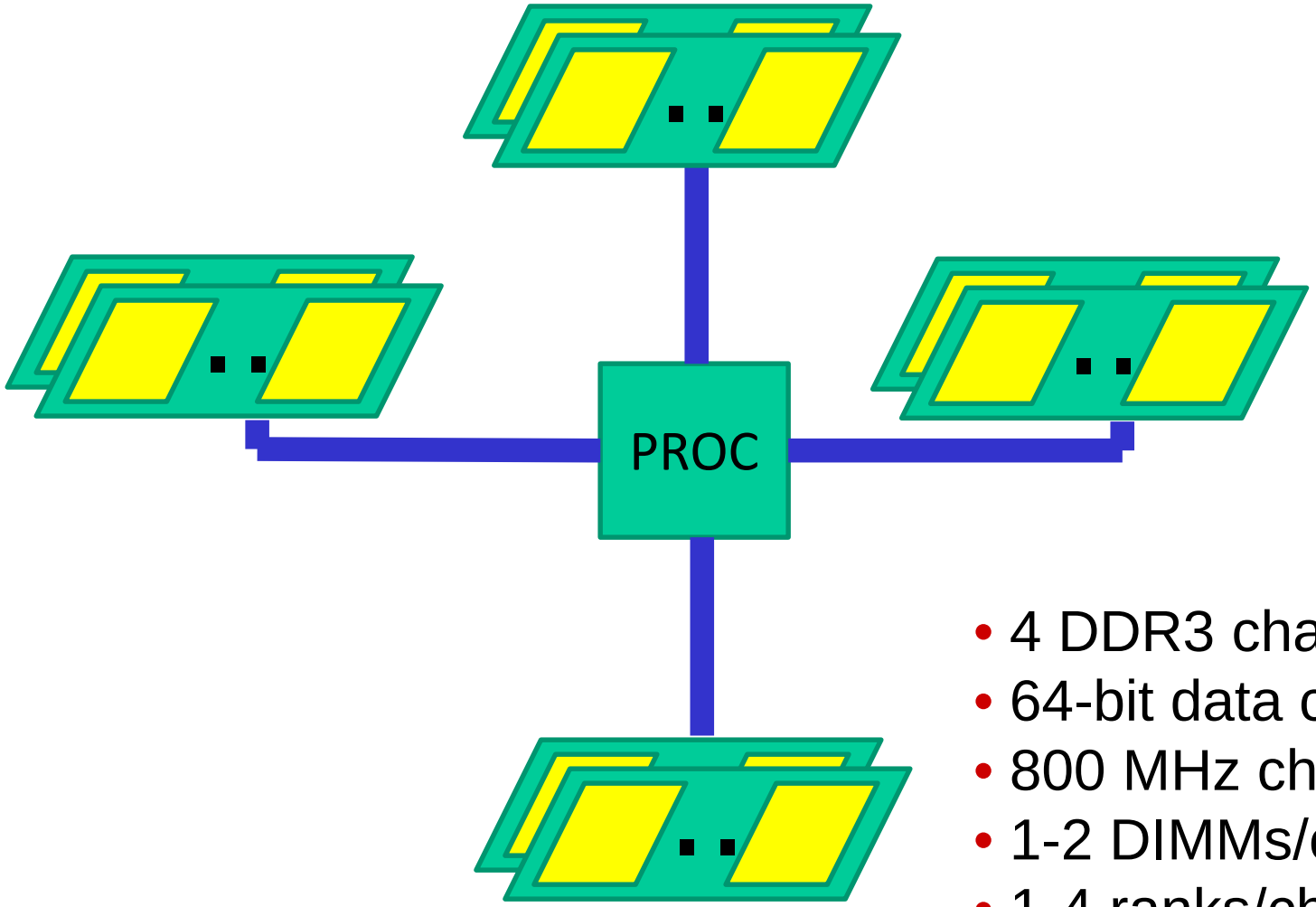
Scheduling Policies

- FCFS: Issue the first read or write in the queue that is ready for issue
- First Ready - FCFS: First issue row buffer hits if you can
- Close page -- early precharge
- Stall Time Fair: First issue row buffer hits, unless other threads are being neglected

Error Correction

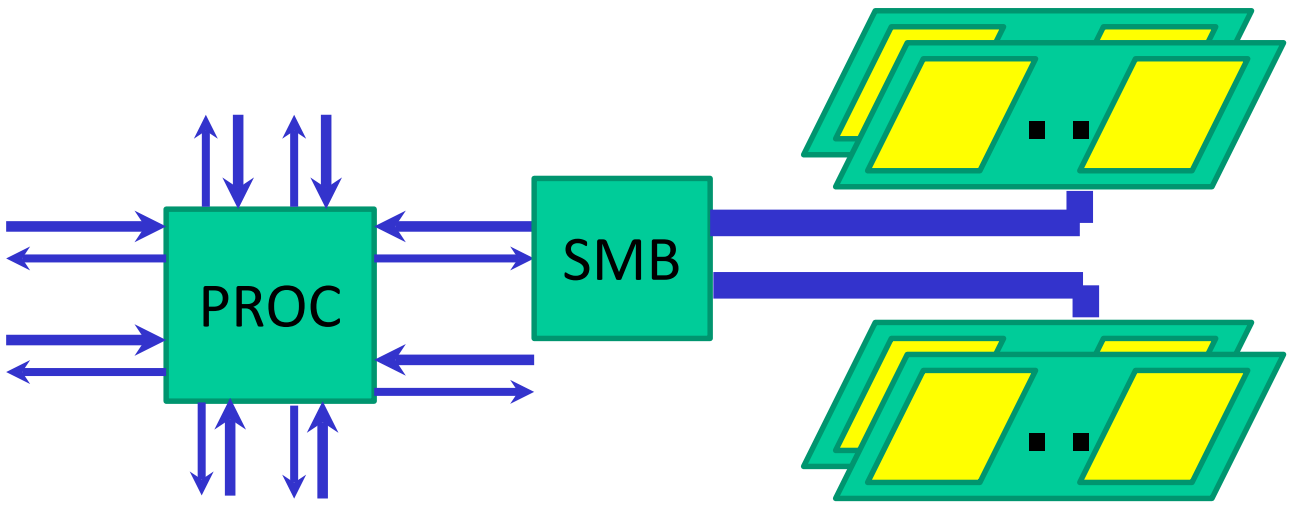
- For every 64-bit word, can add an 8-bit code that can detect two errors and correct one error; referred to as SECDED – single error correct double error detect
- A rank is now made up of 9 x8 chips, instead of 8 x8 chips
- Stronger forms of error protection exist: a system is chipkill correct if it can handle an entire DRAM chip failure

Modern Memory System



- 4 DDR3 channels
- 64-bit data channels
- 800 MHz channels
- 1-2 DIMMs/channel
- 1-4 ranks/channel

Cutting-Edge Systems



- The link into the processor is narrow and high frequency
- The Scalable Memory Buffer chip is a “router” that connects to multiple DDR3 channels (wide and slow)
- Boosts processor pin bandwidth and memory capacity
- More expensive, high power

Problem 6

- What is the boost in capacity and bandwidth provided by using an SMB? Assume that a DDR3 channel requires 64 data wires, 32 addr/cmd wires, and runs at a frequency of 800 MHz (DDR). Assume that the SMB connects to the processor with two 16-bit links that run at frequencies of 6.4 GHz (no DDR). Assume that two DDR3 channels can connect to an SMB. Assume that 50% of the downstream link's bandwidth is used for commands and addresses.

Problem 6

- What is the boost in capacity and bandwidth provided by using an SMB? Assume that a DDR3 channel requires 64 data wires, 32 addr/cmd wires, and runs at a frequency of 800 MHz (DDR). Assume that the SMB connects to the processor with two 16-bit links that run at frequencies of 6.4 GHz (no DDR). Assume that two DDR3 channels can connect to an SMB. Assume that 50% of the downstream link's bandwidth is used for commands and addresses.

  The increase in processor read/write bw =
  (6.4GHz x 72) /(800MHz x 2 x 64) = 4.5x

  (for every 96 wires used by DDR3, you can have 3 32-bit links;
  each 32-bit link supports effectively 24 bits of read/write data)

  Increase in per-pin capacity = 4 DIMMs-per-32-pins /
  2 DIMMs-per-96-pins = 6x

Future Memory Trends

- Processor pin count is not increasing
- High memory bandwidth requires high pin frequency
- High memory capacity requires narrow channels per “DIMM”
- 3D stacking can enable high memory capacity and high channel frequency (e.g., Micron HMC)

Future Memory Cells

- DRAM cell scaling is expected to slow down
- Emerging memory cells are expected to have better scaling properties and eventually higher density: phase change memory (PCM), spin torque transfer (STT-RAM), etc.
- PCM: heat and cool a material with elec pulses – the rate of heat/cool determines if the material is crystalline/amorphous; amorphous has higher resistance (i.e., no longer using capacitive charge to store a bit)
- Advantages: non-volatile, high density, faster than Flash/disk
- Disadvantages: poor write latency/energy, low endurance

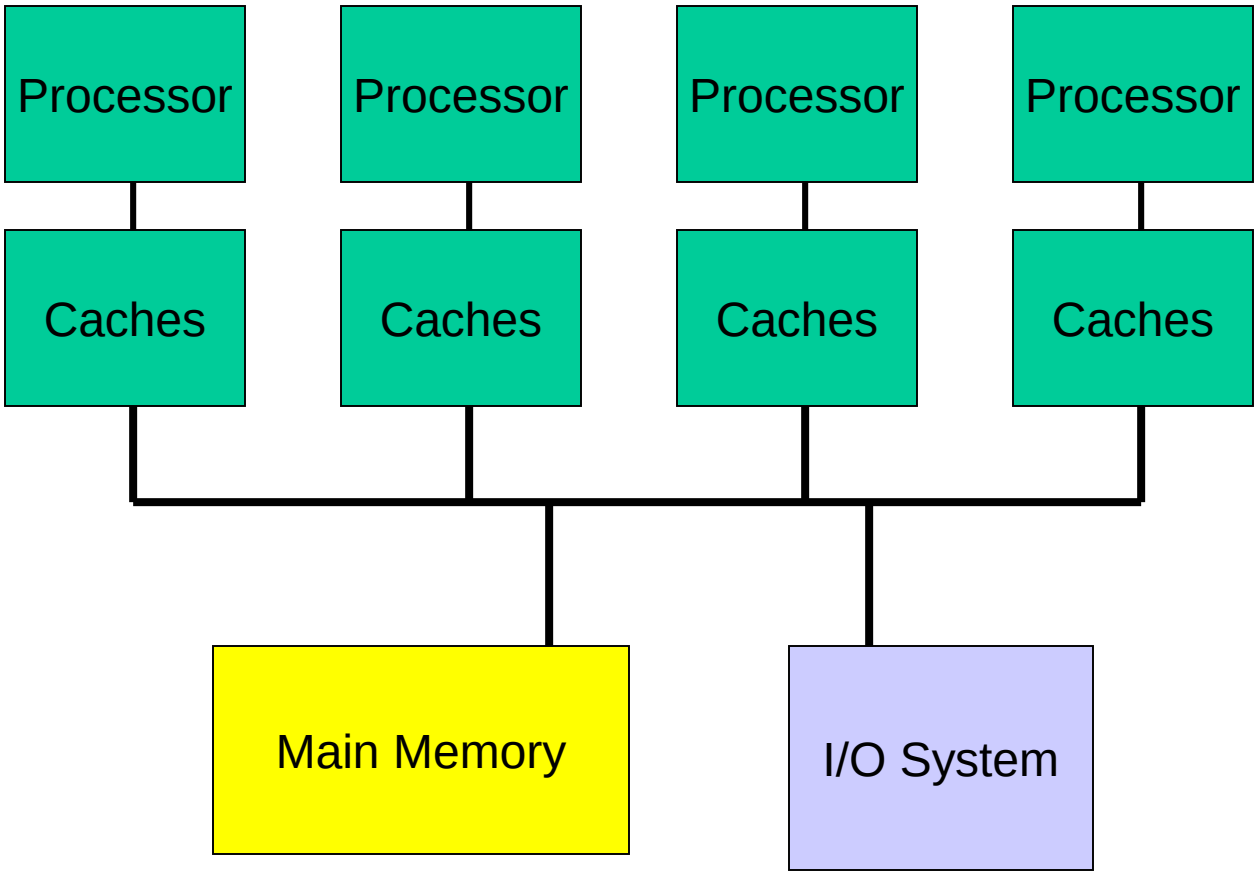
Silicon Photonics

- Game-changing technology that uses light waves for communication; not mature yet and high cost likely
- No longer relies on pins; a few waveguides can emerge from a processor
- Each waveguide carries (say) 64 wavelengths of light (dense wave division multiplexing – DWDM)
- The signal on a wavelength can be modulated at high frequency – gives very high bandwidth per waveguide

Multiprocs -- 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

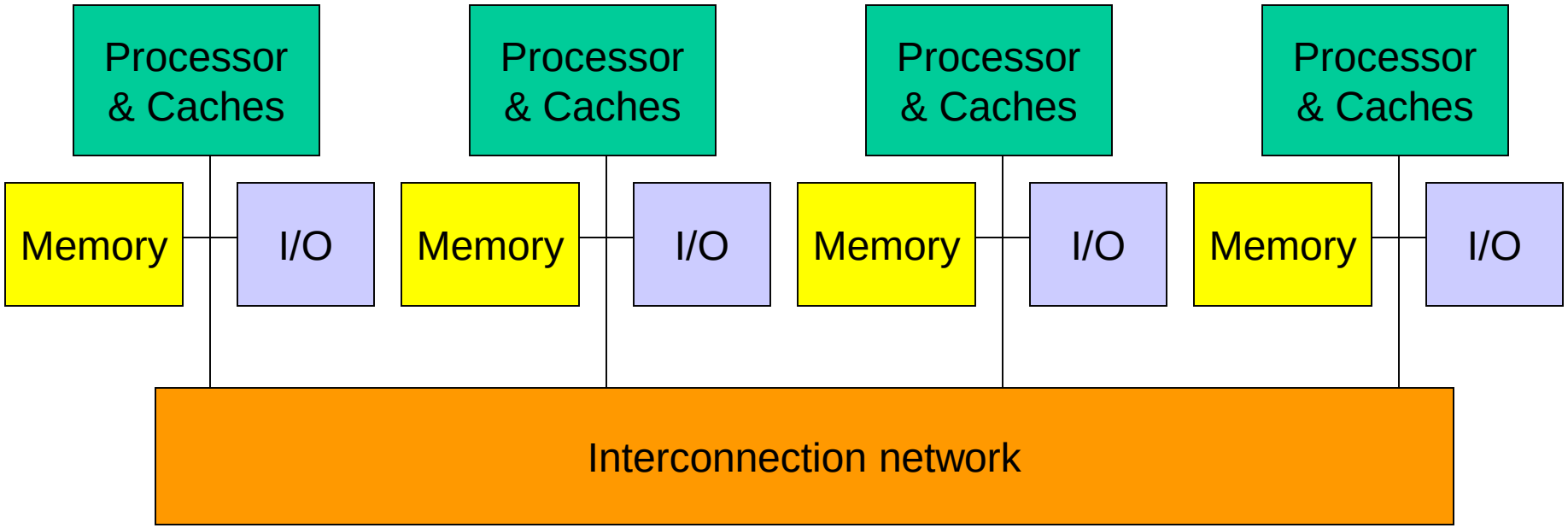
SMPs or Centralized Shared-Memory



Multiprocs -- 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 → simpler hardware
- Explicit communication → easier for the programmer to restructure code
- Sender can initiate data transfer

Ocean Kernel

```
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
```

Shared Address Space Model

```
int  n, nprocs;
float  **A, diff;
LOCKDEC(diff_lock);
BARDEC(bar1);


main()
begin
   read(n); read(nprocs);
   A ← G_MALLOC();
   initialize (A);
   CREATE (nprocs,Solve,A);
   WAIT_FOR_END (nprocs);
end main
```

```
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
```

Title

- Bullet