Lecture: Memory Technology Innovations

- Topics: state-of-the-art and upcoming changes: buffer chips, 3D stacking, non-volatile cells, photonics
- Multiprocessor intro
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
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
• 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

Processor & Caches
Memory
I/O
Memory
I/O
Memory
I/O
Memory
I/O

Interconnection network
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
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
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;
  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 (mydiff < TOL)  done = 1;
  for i ← 1 to nprocs-1 do
    RECEIVE(tempdiff, 1, *, DIFF);
    mydiff += tempdiff;
  endfor
  if (pid != 0)
    SEND(mydiff, 1, 0, DIFF);
    RECEIVE(done, 1, 0, DONE);
  else
    for i ← 1 to nprocs-1 do
      RECEIVE(done, 1, I, DONE);
    endfor
  endif
  endwhile
Title

• Bullet