Introduction

- Reading
  - Today UPC & OpenMP
  - Thursday HPF paper

Software DSM

- Software abstraction to create illusion of shared memory
  - Use message passing to move data around
  - Let's programmers think they have shared memory
- Simple idea, problem is making it work fast
  - Too much data movement
  - Too much synchronization
- Definitions
  - Coherence
    - Ensure modifications propagate to all copies
    - Preserve program order
    - Serialize write
  - Consistency
    - Defines when and what order updates are seen
    - Defines behavior of reads and writes with respect to **other** memory locations

Sequential Consistency

- A system is sequentially consistent if the result of any execution is the same as if the operations of all of the processors were executed in some sequential order, and the operations of each individual processor appear in this sequence in the order specified by its program – Lamport'79
- In practice
  - Every write must be seen on all processors before any succeeding read or write can be issued

```
A= 0;              B= 0;
A= 1;              B= 1;
If (B== 0)         If (A== 0)
   ....               ....
```

Sequential Consistency Problems

- False Sharing
  - Two un-related items get moved because they are being updated at the same time
- Slow Performance
  - Too much communication
  - To much latency

Release Consistency

- Observation: Program need synchronization to get right results
- Distinguish ordinary accesses and synchronization
  - Read the last value written by a processors you synchronized with
- If synchronization is correct, RC behaves like SC
- Lazy Release Consistency
  - Rather than push updates, pull them on next access

Page-based DSM

- Use hardware pages as unit of sharing
  - Hardware page fault handlers can provide support
- Issues
  - How to keep illusion of **shared** memory
  - How to manage communication volume

- Use Release (or Lazy Release Consistency)
  - Reduces the overhead

Multiple Writers

- Allow multiple processes to write a single page
- Synchronization should ensure correctness
- Need to track **what** has changed
  - Diff page between acquire and release of locks
    - Store copy on first update
  - Send diffs between nodes
    - Merge diffs
      - Must be disjoint or synchronization is wrong
  - Aquire lock pulls modifications in

Programming Assignment Notes

- Assume that memory is limited
  - don't replicate the board on all nodes
- Need to provide load balancing
  - goal is to speed computation
  - must trade off
    - communication costs of load balancing
    - computation costs of making choices
    - benefit of having similar amounts of work for each processor
- Consider "back of the envelop" calculations
  - how fast can pvm move data?
  - what is the update time for local cells?
  - how big does the board need to be to see speedups?

OpenMP

- Support Parallelism for SMPs
  - provide a simple portable model
  - allows both shared and private data
  - provides parallel do loops
- Includes
  - automatic support for fork/join parallelism
  - reduction variables
  - atomic statement
    - one processes executes at a time
  - single statement
    - only one process runs this code (first thread to reach it)

Sample Code

```
program compute_pi
    integer n, i
    double precision w, x, sum, pi, f, a
c function to integrate
    f(a) = 4.d0 / (1.d0 + a*a)
    print *, \021Enter number of intervals: \021
    read *,n
c calculate the interval size
    w = 1.0d0/n
    sum = 0.0d0
!$OMP PARALLEL DO PRIVATE(x), SHARED(w)
!$OMP& REDUCTION(+: sum)
    do i = 1, n
       x = w * (i - 0.5d0)
       sum = sum + f(x)
    enddo
    pi = w * sum
    print *, \021computed pi = \021, pi
    stop
    end
```

UPC

- Extension to C for parallel computing
- Target Environment
  - Distributed memory machines
  - Cache Coherent multi-processors
- Features
  - Explicit control of data distribution
  - Includes parallel for statement

UPC Execution Model

- SPMD-based
  - One thread per processor
  - Each thread starts with same entry to main
- Different consistency models possible
  - “strict” model is based on sequential consistency
  - “relaxed” based on release consistency

Forall Loop

- Forms basis of parallelism
- Add forth parameter to for loop “affinity”
  - Where code is executed is based on “affinity”
- Lacks explict barrier before/after execution
  - Differs from openMP
- Supports nested forall loops

Split-phase Barriers

- Traditional Barriers
  - Once enter barriers, busy-wait until everyone arrives
- Split-phase
  - Announce intention to enter barrier (upc_notify)
  - Perform some **local** operations
  - Wait for everyone else (upc_wait)
- Advantage
  - Allows work while waiting for processes to arrive
- Disadvantage
  - Must find work to do
  - Takes time to communicate both notify and wait