CMSC 714
Lecture 4
OpenMP and UPC

Chau-Wen Tseng
(from A. Sussman)

Programming Model Overview

- Message passing (MPI, PVM)
  - Separate address spaces
  - Explicit messages to access shared data
    - Send / receive (MPI 1.0), put / get (MPI 2.0)
- Multithreading (Java threads, pthreads)
  - Shared address space
    - Only local variables on thread stack are private
  - Explicit thread creation, synchronization
- Shared-memory programming (OpenMP, UPC)
  - Mixed shared / separate address spaces
  - Implicit threads & synchronization

Shared Memory Programming Model

- Attempts to ease task of parallel programming
  - Hide details
    - Thread creation, messages, synchronization
  - Compiler generate parallel code
    - Based on user annotations
- Possibly lower performance
  - Less control over
    - Synchronization
    - Locality
    - Message granularity
- My inadvertently introduce data races
  - Read & write same shared memory location in parallel loop

OpenMP

- Support parallelism for SMPs, multi-core
  - Provide a simple portable model
  - Allows both shared and private data
  - Provides parallel for/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)

OpenMP

- Characteristics
  - Both local & shared memory (depending on directives)
  - Parallelism directives for parallel loops & functions
  - Compilers convert into multi-threaded programs (i.e. pthreads)
  - Not supported on clusters

Parallel for indicates loop iterations may be executed in parallel

- Example

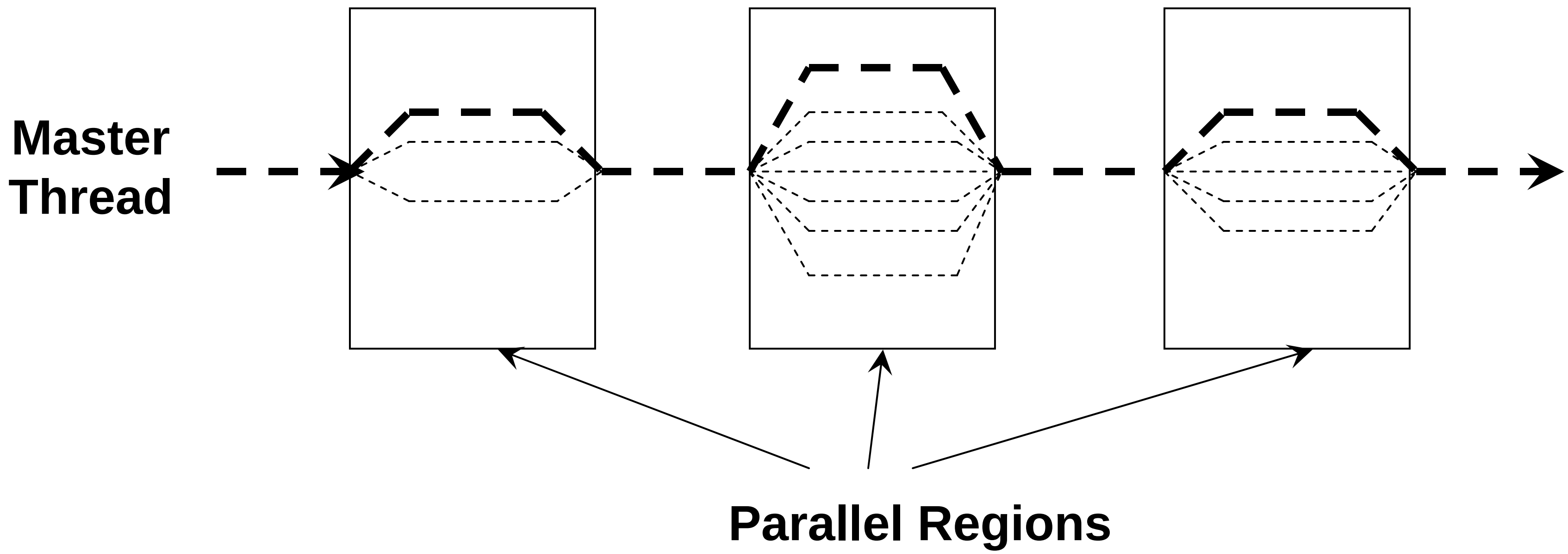
```
#pragma omp parallel for private(i)
for (i=0; i<NUPDATE; i++) {
        int ran = random();
        table[ ran & (TABSIZE-1) ] ^= stable[ ran >> (64-LSTSIZE) ];
}
```

More on OpenMP

- Characteristics
  - Not a full parallel language, but a language extension
  - A set of standard compiler directives and library routines
  - Used to create parallel Fortran, C and C++ programs
  - Usually used to parallelize loops
  - Standardizes last 15 years of SMP practice
- Implementation
  - Compiler directives using #pragma omp <directive>
  - Parallelism can be specified for regions & loops
  - Data can be
    - Private – each processor has local copy
    - Shared – single copy for all processors

OpenMP – Programming Model

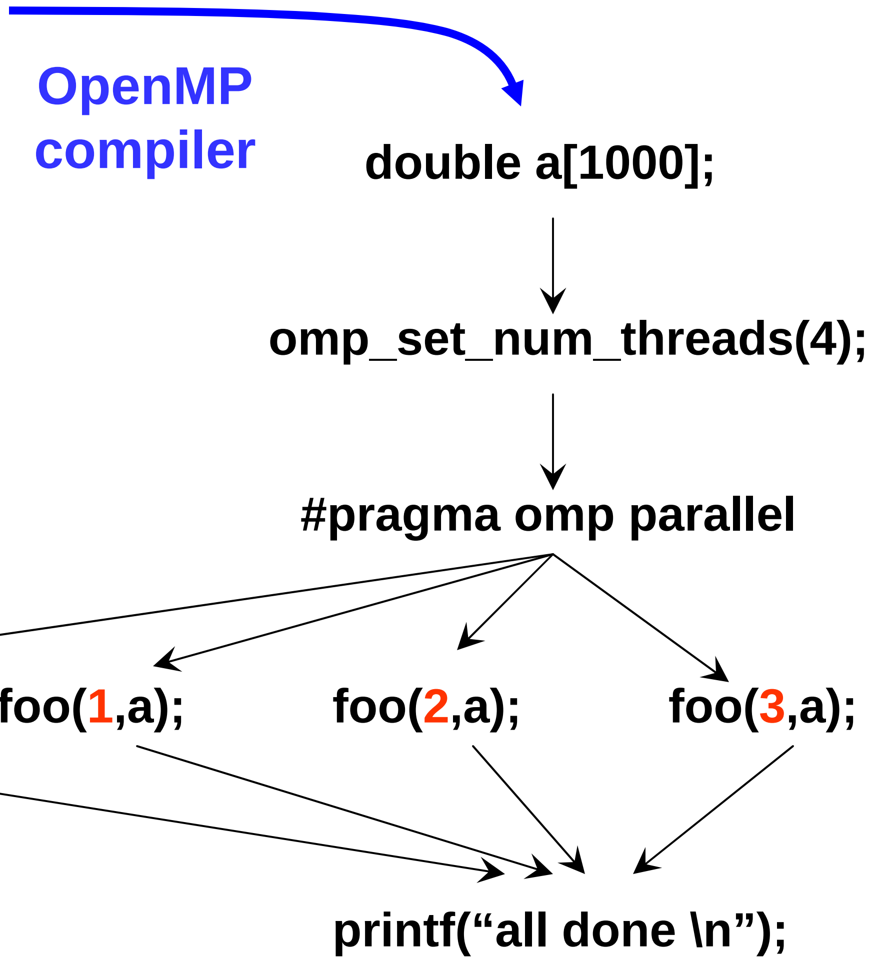
- Fork-join parallelism (restricted form of MIMD)
  - Normally single thread of control (master)
  - Worker threads spawned when parallel region encountered
  - Barrier synchronization required at end of parallel region



OpenMP – Example Parallel Region

- Task level parallelism – #pragma omp parallel { … }

```
double a[1000];
omp_set_num_threads(4);
#pragma omp parallel
{
  int id = omp_thread_num();
  foo(id,a);
}
printf(“all done \n”);
```



OpenMP – Example Parallel Loop

- Loop level parallelism – #pragma omp parallel for
  - Loop iterations are assigned to threads, invoked as functions

OpenMP compiler

```
#pragma omp parallel for
for (i=0;i<N;i++) {
    foo(i);
}
```

```
#pragma omp parallel
{
    int id, i, nthreads,start, end;
    id = omp_get_thread_num();
    nthreads = omp_get_num_threads();
    start = id * N / nthreads ;         // assigning
    end = (id+1) * N / nthreads ;       // work
    for (i=start; i<end; i++) {
        foo(i);
    }
}
```

Loop iterations scheduled in blocks

Iteration Scheduling

- Parallel for loop
  - Simply specifies loop iterations may be executed in parallel
  - Actual processor assignment is up to compiler / run-time system
- Scheduling goals
  - Reduce load imbalance
  - Reduce synchronization overhead
  - Improve data location
- Scheduling approaches
  - Block (chunks of contiguous iterations)
  - Cyclic (round-robin)
  - Dynamic (threads request additional iterations when done)

Parallelism May Cause Data Races

- Data race
  - Multiple accesses to shared data in parallel
  - At least one access is a write
  - Result dependent on order of shared accesses
- May be introduced by parallel loop
  - If data dependence exists between loop iterations
  - Result depend on order loop iterations are executed
  - Example

```
#pragma omp parallel for
for (i=1;i<N-1;i++) {
   a[i] = ( a[i-1] + a[i+1] ) / 2;
}
```

Sample Fortran77 OpenMP 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 *, “Enter # of intervals: “
   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 *, “computed pi = “, pi
   stop
   end
```

Reductions

- Specialized computations that
  - Partial results may be computed in parallel
  - Combine partial results into final result
  - Examples
    - Addition, multiplication, minimum, maximum, count
- OpenMP reduction variable
  - Compiler inserts code to
    - Compute partial result locally
    - Use synchronization / communication to combine results

UPC

- Extension to C for parallel computing
- Target Environment
  - Distributed memory machines
  - Cache coherent multi-processors
  - Multi-core processors
- Features
  - Explicit control of data distribution
  - Includes parallel for statement
  - MPI-like run-time library support

UPC

- Characteristics
  - Local memory, shared arrays accessed by global pointers
  - Parallelism : single program on multiple nodes (SPMD)
  - Provides illusion of shared one-dimensional arrays
  - Features
    - Data distribution declarations for arrays
    - One-sided communication routines (memput / memget)
    - Compilers translate shared pointers & generate communication
    - Can cast shared pointers to local pointers for efficiency
- Example

```
shared int *x, *y, z[100];
upc_forall (i = 0; i < 100; j++) {   z[i] = *x++ × *y++;  }
```

More UPC

- Shared pointer
  - Key feature of UPC
    - Enables support for distributed memory architectures
  - Local (private) pointer pointing to shared array
  - Consists of two parts
    - Processor number
    - Local address on processor
  - Read operations on shared pointer
    - If for nonlocal data, compiler translates into memget( )
  - Write operations on shared pointer
    - If for nonlocal data, compiler translates into memput( )
  - Cast into local private pointer
    - Accesses local portion of shared array w/o communication

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
    - Results must match some sequential execution order
  - “Relaxed” based on release consistency
    - Writes visible only after release synchronization
      - Increased freedom to reorder operations
      - Reduced need to communicate results
  - Consistency models are tricky
    - Avoid data races altogether

Forall Loop

- Forms basis of parallelism
- Add fourth parameter to for loop, “affinity”
  - Where code is executed is based on “affinity”
  - Attempt to assign loop iteration to processor with shared data
    - To reduce communication
- Lacks explicit barrier before / after execution
  - Differs from OpenMP
- Supports nested forall loops

Split-phase Barriers

- Traditional Barriers
  - Once enter barrier, busy-wait until all threads arrive
- Split-phase
  - Announce intention to enter barrier (upc_notify)
  - Perform some **local** operations
  - Wait for other threads (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