CMSC 714
Lecture 5
OpenMP and UPC

Alan Sussman
Notes

• MPI project due week from Friday, Sept. 21
  • any questions about project spec, or running on deepthought2 cluster?
• Don't forget to send questions for readings
  • additional readings posted today, with who should send questions
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 process executes at a time
  • single statement
    • only one process runs this code (first thread to reach it)
  • plus a lot more
OpenMP

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

• **Example**

```c
#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 15-20 years of SMP practice

• **Implementation**
  - C compiler directives using `#pragma omp <directive>`
  - Parallelism can be specified for regions & loops
  - Data can be
    - Private – each thread has local copy
    - Shared – single copy for all threads
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 { ... }

```c
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

```c
#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);
    }
}
```

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

```c
int main() {
    int n, i;
    double w, x, sum, pi;
    printf(“Enter number of intervals: \n”);
    scanf(“%d”, &n);
    /* calculate the interval size */
    w = 1.0;
    sum = 0.0;
    #pragma omp parallel for private(x), shared(w), reduction(+: sum)
    for (i = 1; i <= n; i++) {
        x = w * (i - 0.5);
        sum = sum + f(x);
    }
    pi = w * sum;
    printf (“computed pi = %f\n”, pi);
}
/* function to integrate */
double f(double a) {
    return (2.0 / (1.0 + a*a));
}
```
program compute_pi
    integer n, i
    double precision w, x, sum, pi, f, a

    c function to integrate
    f(a) = 4.0 / (1.0 + a*a)
    print *, “Enter number 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
UPC

• Extension to C for parallel computing
  • a Partitioned Global Address Space (PGAS) language
  • others include Titanium (Java) and Co-Array Fortran (part of the current Fortran standard)

• Target Environment
  • Distributed memory machines
  • Cache Coherent multi-processors

• Features
  • Explicit control of data distribution
  • Includes parallel for statement
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
  - Cast global pointers to local pointers for efficiency
  - One-sided communication routines (memput / memget)
- **Compilers translate global pointers, generate communication**

- **Example**

```c
shared int *x, *y, z[100];

upc_forall (i = 0; i < 100; i++) { z[i] = *x++ * *y++; }
```
UPC Execution Model

• **SPMD-based**
  • One thread per process
  • 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 fourth parameter to for loop, “affinity”
  • Where code is executed is based on “affinity”
• 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