Lecture 7
OpenMP III
Synchronization & Data Environment

John Cavazos
Dept of Computer & Information Sciences
University of Delaware
Lecture Overview

- Reduction Clause
- Basic Synchronization
  - master, single, critical, atomic
  - Barrier
- Data Environment
- Lab
main()
{
    #pragma omp parallel
    {
        cout << "Hello World!";
    }
}

CISC 879: Advanced Parallel Programming
Reduction Clause Review

reduction(<operator>: <variable list>)

+ , *, -, / , &, |, ^, &&, ||

20    long arraysum(int n, long * a) {
21        int i;
22        long sum=0;
23        #pragma omp parallel for reduction(+:sum)
24            for (i = 0; i < n; i++)
25                sum += a[i];
26        return sum;
27    }
Reduction clause can also be used with `#pragma omp parallel`
Constructs:
n. the block following master will only be executed by the master thread

```c
#pragma omp parallel
{
    do_many_things();
    #pragma omp master
    {
        exchange_boundaries();
    }
    #pragma omp barrier
    do_many_other_things();
}
```
OpenMP Synchronization

- Constructs:
  b. the block following *single* will only be executed by one thread of the *team*

```c
#pragma omp parallel
{
  do_many_things();
  #pragma omp single
  {
    exchange_boundaries();
  }
  #pragma omp barrier
do_many_other_things();
}
```

Whichever thread makes it here first
OpenMP Synchronization

- Constructs:
  - c. only one thread of a team can be in a **critical** block at anytime

```c
float res;
#pragma omp parallel
{
  float B; int i, id, nthrds;
  id = omp_get_thread_num();
  nthrds = omp_get_num_threads();
  for(i=id;i<niters;i+=nthrds){
    B = big_job(i);
  }
  #pragma omp critical
  res += consume(B);
}
```
OpenMP Synchronization

- Constructs:
  - **d. Atomic** provides mutual exclusion to a memory location

```c
#pragma omp parallel
{
    double tmp, B;
    B = DOIT();
    tmp = big_ugly(B);
    #pragma omp atomic
    X += tmp;
}
```
OpenMP Synchronization

- Constructs:
  - **barrier** is a synchronization point for all threads in the team

```
#pragma omp parallel
{
  int id=omp_get_thread_num();
  A[id] = big_calc1(id);
  #pragma omp barrier
  B[id] = big_calc2(id, A);
}
```
OpenMP Barrier

1) explicit barrier
   #pragma omp barrier

2) implicit barrier
   e.g. at the end of
   #pragma omp parallel
   #pragma omp parallel for
   #pragma omp single
   and more ....
```c
#pragma omp parallel shared (A, B, C) private(id)
{
    id = omp_get_thread_num();
    A[id] = big_calc1(id);
    #pragma omp barrier
    #pragma omp for
    for (i = 0; i < N; i++) { C[i] = big_calc3(i, A); }
    #pragma omp for nowait
    for (i = 0; i < N; i++) { B[i] = big_calc2(C, i); } 
    A[id] = big_calc4(id);
}
```

- implicit barrier at the end of a parallel region
- implicit barrier at the end of a for worksharing construct
- no implicit barrier due to nowait
int main() {

    double startTime;

    #pragma omp parallel private (startTime) num_threads(4)
    {
        startTime = omp_get_wtime();
        // Each thread sleep ID second (master thread sleep 0 s)
        while( (omp_get_wtime() - startTime) < (double)(omp_get_thread_num()));
        printf("I (%d) finish to count\n", omp_get_thread_num());
        // Each thread will wait other
        #pragma omp barrier
        printf("I (%d) pass the Barrier\n", omp_get_thread_num());

        #pragma omp single
        {
            printf("I (%d) am the only one executing this code\n", omp_get_thread_num());
        }

        #pragma omp master
        {
            printf("I (%d) am the Master\n", omp_get_thread_num());
        }
    }

    return 0;
}
barrier-omp.c in Lecture-05.zip

http://www.cis.udel.edu/~wwang/cisc879/Lecture-05.zip

tristan@tristan-laptop:~/classes/hpc-lecture/lecture2$ ./barrier-omp
I (0) finish to count
I (1) finish to count
I (2) finish to count
I (3) finish to count
I (3) pass the Barrier
I (3) am the only one executing this code
I (2) pass the Barrier
I (0) pass the Barrier
I (1) pass the Barrier
I (0) am the Master
OpenMP Data Environment

- SHARED
- PRIVATE
- FIRSTPRIVATE
- LASTPRIVATE
- DEFAULT(SHARED|NONE)
incr = 0;
#pragma omp parallel for firstprivate(incr)
for (i=0; i <= Max; i++) {
    if (i%2 == 0) incr++;
    A[i] = incr;
}

Private variable incr gets initialized!
void sq2(int n, double *lastitem)
{
    double x; int i;
    #pragma omp parallel for lastprivate(x)
    for (i=0; i < n; i++) {
        x = a[i] * a[i] + b[i] * b[i];
        b[i] = sqrt(x);
    }
    *lastitem = x;
}

Value of x in the last iteration (i==n-1) retained!
- **DEFAULT(SHARED)**
  Variable attributes not specified are shared by default

- **DEFAULT(NONE)**
  Manually specify each variable’s private/shared attribute
Debug the Mandelbrot Code

http://www.cis.udel.edu/~wwang/cisc879/Lecture-07.zip
Mandelbrot-NeedsCorrection.c
Mandelbrot-Corrected.c