OpenMP, Part 2

EAS 520
High Performance Scientific Computing

University of Massachusetts Dartmouth

Spring 2015

References

This presentation is almost an exact copy of Dartmouth College's openMP tutorial. The link can be found in:

`http://www.dartmouth.edu/~rc/classes/intro_openmp/`

Changes from the original document are related to compilers and job submissions for UMass Dartmouth clusters.

How to Compile and run an OpenMP program

On the UMD cluster, the default gcc and gfortran compilers support OpenMP.
Altneratively, you could load a different compiler via a module command:

```
$ module load eas520/compilers/gcc-4.8.2
```

Then you can compile the source code

```
$ gfortran -fopenmp -o hello-f hello-f.f90
```

and run the program

```
$ ./hello-f
```

The output should look like (thread calls order can be different)

```
Hello from thread 1, nthreads 4
Hello from thread 3, nthreads 4
Hello from thread 2, nthreads 4
Hello from thread 0, nthreads 4
```

Approaches to Parallelism Using OpenMP

Two main approaches:

- loop-level
- parallel regions

Loop-Level Parallelism:

- sometimes call *fine-grained parallelism*
- individual loops parallelized
- each thread assigned a unique range of the loop index
- execution starts on a single serial thread
- multiple threads are spawned inside a parallel loop
- after parallel loop execution is serial
- relatively easy to implement

(We gave simplistic examples of this last time.)

Continued..

Parallel Regions Parallelism:

- sometimes called *coarse-grained parallelism*
- any sections of codes can be parallelized (not just loops)
- using the thread identifier to distribute the work
- execution starts on a single serial thread
- multiple threads are started for parallel regions (not necessarily at a loop)
- ends on a single serial thread

Shared vs Private Variables

- By default all variables in a loop share the same address space
- All threads can modify and access all variables (except the loop index)
- Can result in incorrect results
- Can use shared and private clauses with parallel for or parallel do

Example of Code with No Data Dependencies

Fortran Example

```
!$omp parallel do private(temp) shared(n,a,b,c)
do i = 1, n
   temp = 2.0*a(i)
   a(i) = temp
   b(i) = c(i)/temp
enddo
```

C/C++ Example

```
#pragma omp parallel for private(temp) shared(n,a,b,c)
{
   for(i=1; i<=n; i++) {
      temp = 2.0*a[i];
      a[i] = temp;
      b[i] = c[i]/temp;
   }
}
```

Example of Parallelizing a Loop:

Fortran Example: workshare.f90

```
program workshare
  use omp_lib
  implicit none
  integer :: nthreads, tid, ncores, i
  integer, parameter :: n = 100
  real, dimension(n) :: a, b, c
  character(LEN=50), parameter :: fmt = '(A, I2, A ,I3, A , F8.2)'

  ! some initializations
  do i = 1, n
     a(i) = i
     b(i) = a(i)
  end do

  ncores = 8
  call omp_set_num_threads(ncores)

  !$omp parallel shared(a,b,c) private(i,tid)
        tid = omp_get_thread_num()
        if (tid .eq. 0) then
          nthreads = omp_get_num_threads()
          write(*,*) 'number of threads =', nthreads
        end if

        write(*,*) 'thread',tid,' starting...'

        !$omp do
        do i = 1, n
           c(i) = a(i) + b(i)
           write(*,fmt) ' thread', tid, ': c(', i ,')=', c(i)
        end do
        !$omp end do nowait

        write(*,*) 'thread',tid,' done.'
   !$omp end parallel
end program workshare
```

Example of Parallelizing a Loop

C/C++ Example

```
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>
int main (int argc, char *argv[]) {

int nthreads, tid;

/* Fork a team of threads giving them their own copies of variables */
#pragma omp parallel private(nthreads, tid)
 {

 /* Obtain thread number */
 tid = omp_get_thread_num();
 printf("Hello World from thread = %d\n", tid);

 /* Only master thread does this */
 if (tid == 0)
 {
  nthreads = omp_get_num_threads();
 printf("Number of threads = %d\n", nthreads);
 }

 } /* All threads join master thread and disband */

}
```

Basic OpenMP Functions

- **omp_get_num_threads()** - get the number of threads used in a parallel region
- **omp_get_thread_num()** - get the thread rank in a parallel region (0 to **omp_get_num_threads()** -1)
- **omp_set_num_threads(nthreads)** - set the number of threads used in a parallel region

Fortran Example

```
!$omp parallel
      write(*,*) ' Thread rank: ', omp_get_thread_num()
!$omp end parallel
```

C/C++ Example

```
# pragma omp parallel
{
  printf("Thread rank: %d\n", omp_get_thread_num());
}
```

Other OpenMP Clauses (firstprivate, lastprivate, ordered)

firstprivate

- initialize a variable from the serial part of the code
- private clause doesn't initialize the variable

Fortran Example

```fortran
j = jstart
!$omp parallel do firstprivate(j)
do i = 1, n
   if(i == 1  .or. i == n) then
      j = j + 1
   endif
   a(i) = a(i) + j
end do
```

C/C++ Example

```c
j = jstart;
#pragma omp parallel for firstprivate(j)
{
   for(i=1; i<=n; i++){
      if(i == 1 || i == n)
         j = j + 1;
      a[i] = a[i] + j;
   }
}
```

Continued...

lastprivate

- thread that executes the ending loop index copies its value to the master (serial) thread
- this gives the same result as serial execution

Fortran Example

```
!$omp parallel do lastprivate(x)
do i = 1, n
   x = sin(pi*dx*real(i))
   a(i) = exp(x)
end do
lastx = x
```

C/C++ Example

```
#pragma omp parallel for lastprivate(x)
{
   for(i=1; i<=n; i++){
    x = sin( pi * dx * (float)i );
      a[i] = exp(x);
   }
}
lastx = x;
```

Continued...

ordered

- used when part of the loop must execute in serial order
- ordered clause plus an ordered directive

Fortran Example

```
!$omp parallel do private(myval) ordered
do i = 1, n
   myval = do_lots_of_work(i)
   !$omp ordered
   write(*,*)  i, myval
   !$omp end ordered
end do
lastx = x
```

Continued....

C/C++ Example

```
#pragma omp parallel for private(myval) ordered
{
   for(i=1; i<=n; i++){
      myval = do_lots_of_work(i);
      #pragma omp ordered
      {
         printf("%d %d\n", i, myval);
      }
   }
}
```

Reduction Operations

An example of a reduction operation is a summation:

Fortran Example

```
do i = 1, n
   sum = sum + a(i)
end do
```

C/C++ Example

```
for(i=1; i<=n; i++){
   sum = sum + a[i];
}
```

How reduction works:

- sum is the reduction variable
- cannot be declared shared
    - threads would overwrite the value of sum
- cannot be declared private
    - private variables don't persist outside of parallel region
- specified reduction operation performed on individual values from each thread

Example of reduction clause

Fortran Example

```
!$omp parallel do reduction(+:sum)
do i = 1, n
   sum = sum + a(i)
end do
```

Fortran Reduction Operands

Operator	Initial Value
+	0
*	1
-	0
.AND.	.true.
.OR.	.false.
.IEOR.	0
.IOR.	0
.IAND.	All bits on
.EQV.	.true.
MIN	Largest positive #
MAX	Most negative #

C/C++ Example

```
#pragma omp parallel for reduction(+:sum)
{
   for(i=1; i<=n; i++){
      sum = sum + a[i];
   }
}
```

C/C++ Reduction Operands

Operator	Initial Value
+	0
*	1
-	0
&	~0
|	0
^	0
&&	1
||	0