Shared Memory Parallel Programming

Worksharing in OpenMP

OpenMP Directives
OpenMP Directives

• We now look at OpenMP directives to create parallel code
• Main tasks
  – create and terminate threads (parallel regions)
  – share out work in parallel regions to threads
  – synchronize threads
Some Rules for Directives

- They apply to next statement, or set of statements
- May need to mark end of set of statements
- Structured block of statements
- Examples
  - !OMP$ …
    statement
    !OMP$ END …
  - #pragma omp …
    { statement1; statement2; statement3; }
OpenMP Parallel Region

#pragma omp parallel
 !$OMP PARALLEL

• Master creates team of threads at entry
• Each thread executes the same code
• Each thread terminates at the end
• Very similar to a number of create/join’s with the same function in Pthreads
Parallel Regions

- Sequential region executed by master thread
- Parallel region executed by team of threads
- Sequential region executed by master thread
Getting Threads to do Different Things

• Through explicit thread identification (as in Pthreads)
  – threads in team are numbered consecutively

• Through work-sharing directives
Thread Identification

```c
int omp_get_thread_num();
ext int omp_get_num_threads();
```

- Gets the thread id
- Gets the total number of threads
- id of master thread is 0
Example

```c
#pragma omp parallel
{
    if( !omp_get_thread_num() )
        master();
    else
        slave();
}
```
Example

!$OMP PARALLEL PRIVATE (iam, np, ipoints)
  iam = omp_get_thread_num()
  np = omp_get_num_threads()
  ipoints = npoints / np
  call domywork (x, iam, ipoints)
!$OMP END PARALLEL
Work Sharing Directives

• Share work within a parallel region
• Scope of parallel region is dynamic, so may be in different procedure
• Main ones are
  – OpenMP for and DO
  – OpenMP sections and section
  – OpenMP workshare
OpenMP For

```c
#pragma omp parallel
#pragma omp for
for( ... ) { ... }
```

- Each thread executes a subset of the iterations
- All threads wait at the end of the OpenMP for
Multiple Work Sharing Directives

• May occur within a single parallel region

    #pragma omp parallel
    {
        #pragma omp for
        for( ; ; ) { … }
        #pragma omp for
        for( ; ; ) { … }
    }

• All threads wait at the end of each for
The NoWait Qualifier

```c
#pragma omp parallel
{
#pragma omp for nowait
for( ; ; ) { … }
#pragma omp for
for( ; ; ) { … }
}
```

- Nowait: no synchronization at end of loop
- Threads proceed immediately to second for
OpenMP DO

 !$OMP PARALLEL
 !$OMP DO
  do loop
 !$OMP END DO
 !$OMP END PARALLEL

• Fortran parallel DO same as parallel for in C
• End directive is optional but a good idea
• All threads wait at the end of the OpenMP do
The NoWait Qualifier

!$OMP PARALLEL
!$OMP DO
  do loop
!$OMP END DO NOWAIT
!$OMP DO
  do loop
!$OMP END DO
!$OMP END PARALLEL

- Nowait: no synchronization at end of first loop
- Threads proceed immediately to second loop
Fortran90 OpenMP Workshare

REAL, DIMENSION (100,100) :: A, B, C, D

. . . . .

!$OMP PARALLEL WORKSHARE
   A = B
   C = D

!$OMP END PARALLEL WORKSHARE

• Threads share work in array statements
• End directive is optional when scope is clear, but still a good idea
• All threads wait at end of parallel workshare unless NOWAIT is specified
Parallel Sections Directive

• Divides different sections of code among threads
• Each section is executed once by a thread
Example

#pragma omp parallel
{
#pragma omp sections
{
#pragma omp section { doxdir(); … }
#pragma omp section { doydir(); … }
#pragma omp section { dozdir(); … }
...
}
}

Useful Shorthand

#pragma omp parallel
#pragma omp for
for (; ; ) { ... }

is equivalent to

#pragma omp parallel for
for (; ; ) { ... }

(Same for parallel sections)
Note the Difference between ...

```c
#pragma omp parallel
{
#pragma omp for
for( ; ; ) { … } 
f();
#pragma omp for
for( ; ; ) { … } 
}
```
... and ...

```c
#pragma omp parallel for
for( ; ; ) { ... }
f();
#pragma omp parallel for
for( ; ; ) { ... }
```
OpenMP Matrix Multiply

#pragma omp parallel for
for( i=0; i<n; i++ )
    for( j=0; j<n; j++ ) {
        c[i][j] = 0.0;
        for( k=0; k<n; k++ )
            c[i][j] += a[i][k]*b[k][j];
    }
for some number of timesteps/iterations {
    #pragma omp parallel for
    for (i=0; i<n; i++)
        for (j=0, j<n, j++)
            temp[i][j] = 0.25 *
                ( grid[i-1][j] + grid[i+1][j]
                    grid[i][j-1] + grid[i][j+1] );

    #pragma omp parallel for
    for (i=0; i<n; i++)
        for (j=0; j<n; j++)
            grid[i][j] = temp[i][j];
}
for some number of timesteps/iterations {
#pragma omp parallel
{
    #pragma omp for
    for (i=0; i<n; i++)
    for( j=0, j<n, j++ )
        temp[i][j] = 0.25 *
            ( grid[i-1][j] + grid[i+1][j]
                grid[i][j-1] + grid[i][j+1] );

    #pragma omp for
    for( i=0; i<n; i++)
    for( j=0; j<n; j++ )
        grid[i][j] = temp[i][j]
}
}
Equivalent OpenMP Jacobi

for some number of timesteps/iterations {
#pragma omp parallel
{
    updategrid1();
    updategrid2();
}
}

updategrid1 ()
{
    #pragma omp for
    for (i=0; i<n; i++ )
    for( j=0, j<n, j++ )
        temp[i][j] = 0.25 *
                        ( grid[i-1][j] + grid[i+1][j]
                          grid[i][j-1] + grid[i][j+1] );
}

......
Gaussian Elimination (1 of 2)

for( i=0; i<n; i++ )
  for( j=i+1; j<n; j++ )
    for( k=i+1; k<n; k++ )
      a[j][k] = a[j][k] - a[i][k]*a[i][j] / a[j][j]

The j loop is outermost parallelizable loop.
Gaussian Elimination (2 of 2)

for( i=0; i<n; i++ )
    #pragma omp parallel for
    for( j=i+1; j<n; j++ )
        for( k=i+1; k<n; k++ )
            a[j][k] = a[j][k] - a[i][k]*a[i][j] / a[j][j]
Single Directive

!$OMP PARALLEL

......

!$OMP SINGLE

structured block of code

!$OMP END SINGLE

......

!$OMP END PARALLEL

• Executed by only one thread
• Other threads wait for it unless NOWAIT is specified
Example

for( i=0; i<100; i++ ) a[i] = f(i);

x = g(a);

for( i=0; i<100; i++ ) b[i] = x + h( a[i] );

- First loop can be run in parallel
- Middle statement is sequential
- Second loop can be run in parallel
#pragma omp parallel
{
    #pragma omp for
    for( i=0; i<100; i++ ) a[i] = f(i);
    #pragma omp single
    x = g(a);
    #pragma omp for
    for( i=0; i<100; i++ ) b[i] = x + h( a[i] );
}
Conditional Parallelism

- OpenMP constructs have an implementation cost
- Parallelism is only useful if there is sufficient work to amortize the overhead
- For small amount of work, overhead of parallelization may exceed benefit
Conditional Parallelism

```c
#pragma omp parallel if( expression )
#pragma omp for if( expression )
#pragma omp parallel for if( expression )
```

- Expression must be scalar logical
- Execute in parallel if expression is true, otherwise execute sequentially
- Similar in OpenMP Fortran
Conditional Parallelism: Example

for( i=0; i<n; i++ )
    #pragma omp parallel for if ( n-i > 100 )
    for( j=i+1; j<n; j++ )
        for( k=i+1; k<n; k++ )
            a[j][k] = a[j][k] - a[i][k]*a[i][j] / a[j][j]
Scheduling of Iterations: Issue

• User may assign specific loop iterations or individual parallel sections to a thread
• The mapping is called a schedule
• OpenMP allows for a variety of scheduling strategies
  – default is block
  – chunks (cyclic, block-cyclic)
  – self-scheduling, guided self-scheduling
Scheduling of Iterations: Specification

```c
#pragma omp parallel for schedule(<sched>)
```

- `<sched>` can be one of
  - block (default)
  - cyclic
  - gss
Scheduling of Iterations: Specification

!$OMP PARALLEL DO SCHEDULE (<sched>)

• <sched> can be one of
  – STATIC : block, default
  – STATIC (chunk) : blockcyclic, chunk is size
  – DYNAMIC (chunk) : self-scheduled
  – GUIDED (chunk) : size of pieces decreases down to chunk
Example

- Multiplication of two matrices $C = A \times B$, where the A matrix is upper-triangular (all elements below diagonal are 0).
OpenMP Matrix Multiply

#pragma omp parallel for schedule ( cyclic )
for( i=0; i<n; i++ )
  for( j=i; j<n; j++ ) {
    c[i][j] = 0.0;
    for( k=i; k<n; k++ )
      c[i][j] += a[i][k]*b[k][j];
  }
OpenMP Directives

• Parallelization directives:
  – parallel region
  – parallel for, do, worksharing, sections, section
  – single, ordered, master

• We have seen directives for distribution of work
OpenMP Directives

• More directives exist for creating local data and synchronizing threads

• Data environment directives:
  – shared, private, firstprivate, threadprivate, reduction, etc.

• Synchronization directives:
  – barrier, critical, atomic, flush
  – nowait