

Parallel Computing for Science & Engineering CS395T

02/06/2018

Instructors:

Lars Koesterke, TACC

Charlie Dey, TACC

Example code



THE UNIVERSITY OF TEXAS AT AUSTIN
Texas Advanced Computing Center

Why OpenMP? — How to learn OpenMP?

- Why?

Execute faster in Parallel

OpenMP is easy to learn

Works well on SMP platforms, i.e.

Supercomputers and PCs

- How does it work?

Threads, shared & private memory

Parallel regions embedded in serial code

Work-sharing in parallel regions

Loops & Sections

- What are the basics?

How do I get started?

Example code

- What features are available?

OpenMP is a “rich” language

It provides tools for your needs

- synchronization (barrier, critical region)
- serial segments in parallel regions (single, ...)
- Reductions
- Interaction with the environment (Runtime API)
- etc.

```

!*** Preset mts; Number of Threads has to be 4
mts = 4
!$ call OMP_SET_NUM_THREADS(mts)
write (*,(A,i2)) 'Number of Threads is set to ', MTS
write (*,*)

!*** Parallel section and worksharing in one statement
!$OMP PARALLEL DO
do i=0, mts-1
  write (*,(a,i2)) 'A : This is thread # ', i
enddo
write (*,*)

!*** Worksharing inside a parallel section
!$OMP PARALLEL
!$OMP DO
do i=0, mts-1
  write (*,(a,i2)) 'B : This is thread # ', i
enddo
!$OMP END PARALLEL
WRITE (*,*)

!*** Worksharing twice inside a parallel section
!$OMP PARALLEL
!$OMP DO
do i=0, mts-1
  write (*,(a,i2)) 'C1: This is thread # ', i
enddo
!$OMP DO
do i=0, mts-1
  write (*,(a,i2)) 'C2: This is thread # ', i
enddo
!$OMP END PARALLEL
write (*,*)

!*** Explicit Worksharing inside a parallel section
!$OMP PARALLEL DEFAULT(NONE) PRIVATE(its, is, ie) SHARED(x)
!$ its = OMP_GET_THREAD_NUM()
if (its == 0) then; is=1; ie=25; endif
if (its == 1) then; is=26; ie=50; endif
if (its == 2) then; is=51; ie=75; endif
if (its == 3) then; is=76; ie=100; endif
write (*,'(a,i2,2x,a,1x,i4,1x,i4)') &
'DE: This is thread # ', its, 'Loop bounds : ', is, ie
do i=is, ie
  x(i) = x(i) + 1.
enddo
!$OMP END PARALLEL
write (*,*)

```

```

!*** Parallel section and worksharing in one statement with ordered output
!$OMP PARALLEL DO ORDERED DEFAULT(NONE) PRIVATE(i,j) SHARED(mts,y)
do i=0, mts-1
  do j=1, 100
    y(j,i) = y(j,i) + 1.
  enddo
!$OMP ORDERED
  write (*,'(a,i2)') 'OR: This is thread # ', i
!$OMP END ORDERED
enddo
write (*,*)

!*** Orphaned Worksharing
write (*,'(a)') 'Orphaned work sharing within parallel region'
!$OMP PARALLEL
call orphan(mts)
!$OMP END PARALLEL
write (*,*)

!*** Orphaned Worksharing, called outside of a parallel region
write (*,'(a)') 'Orphaned outside of parallel region'
call orphan(mts)
write (*,*)

!*** Orphaned replicated work
write (*,'(a)') 'Orphaned inside of parallel region'
!$OMP PARALLEL
call orphan_replicated(mts)
!$OMP END PARALLEL
write (*,*)
END

SUBROUTINE orphan(mts)
!$ USE OMP_LIB
!$ its = OMP_GET_THREAD_NUM()
!$OMP DO
do i=0, mts-1
  write (*,'(a,i2)') 'O : This is thread # ', its
enddo
RETURN
END

SUBROUTINE orphan_replicated(mts)
!$ USE OMP_LIB
!$ its = OMP_GET_THREAD_NUM()
do i=0, mts-1
  write (*,'(a,i2)') 'O : This is thread # ', its
enddo
RETURN
END

```




```
PROGRAM EXAMPLE_OMP_04
```

```
!$ USE OMP_LIB
```

```
!*** Scratch Array
```

```
INTEGER, PARAMETER :: M = 100 ! M has to be 100
REAL, DIMENSION(M) :: X
REAL, DIMENSION(M,4) :: Y
```

```
!*** Preset mts; Number of Threads has to be 4
mts = 4
```

```
!$ call OMP_SET_NUM_THREADS(mts)
write (*,'(A,I2)') 'Number of Threads is set to ', MTS
write (*,*)
```

```
!*** Critical
```

```
icount = 0
!$OMP PARALLEL DO DEFAULT(NONE) PRIVATE(i,j) SHARED(y,icount)
do i=1, 4
  do j=1, m
    y(j,i) = y(j,i) + 1.
  enddo
!$OMP CRITICAL
  icount = icount + 1
!$OMP END CRITICAL
enddo
write (*,'(a,i6)') 'CRITICAL      :: icount = ', icount
write (*,*)
```

```
!*** Critical with if construct
```

```
icount = 0
!$OMP PARALLEL DO DEFAULT(NONE) PRIVATE(i,j) SHARED(y,icount)
do i=1, 4
  do j=1, m
    y(j,i) = y(j,i) + 1.
  enddo
  if (i <= 2 ) then
!$OMP CRITICAL
  icount = icount + 1
!$OMP END CRITICAL
  endif
enddo
write (*,'(a,i6)') 'CRITICAL + IF :: icount = ', icount
write (*,*)
```

```
!*** Single :: worksharing construct,
!***          every thread has to encounter it
!$OMP PARALLEL DEFAULT(NONE) PRIVATE(i,j,its) SHARED(y,icount)
!$ its = OMP_GET_THREAD_NUM()
!$OMP SINGLE
write (*,'(a,i4)') &
'Calculation has started, SINGLE, this is thread ', its
!$OMP END SINGLE
!$OMP DO
do i=1, 4
  do j=1, m
    y(j,i) = y(j,i) + 1.
  enddo
enddo
!$OMP END PARALLEL
write (*,*)
```

```
!*** Master :: NOT a worksharing construct,
```

```
!***          not every thread has to encounter it
!$OMP PARALLEL DEFAULT(NONE) PRIVATE(i,j,its) SHARED(y,icount)
!$ its = OMP_GET_THREAD_NUM()
!$OMP MASTER
write (*,'(a,i4)') &
'Calculation has started, MASTER, this is thread ', its
!$OMP END MASTER
!$OMP DO
do i=1, 4
  do j=1, m
    y(j,i) = y(j,i) + 1.
  enddo
enddo
!$OMP END PARALLEL
write (*,*)
```

```
!*** Master :: NOT a worksharing construct,
```

```
!***          not every thread has to encounter it
!$OMP PARALLEL DEFAULT(NONE) PRIVATE(i,j,its) SHARED(y,icount)
!$ its = OMP_GET_THREAD_NUM()
if (its <= 1) then
!$OMP MASTER
write (*,'(a,i4)') &
'Calculation has started, MASTER, this is thread ', its
!$OMP END MASTER
write (*,'(a,i4)') &
'Hi, this is thread ', its
endif
```

```
!$OMP DO
do i=1, 4
  do j=1, m
    y(j,i) = y(j,i) + 1.
  enddo
enddo
!$OMP END PARALLEL
write (*,*)
END
```




```

int main (int argc, char* argv[])
{

    const int m = 100;
    int its, mts, i, j, icount, is, ie;
    float x[m], y[4][m], sum, prod, t1, t2, t3;

    // Preset mts, Inquire the number of Threads
    mts = 4;
#ifdef _OPENMP
    omp_set_num_threads(mts);
#endif
    printf("Number of Threads is %i\n", mts);
    printf("\n");

    // Parallel section and worksharing in one statement
#pragma omp parallel for
    for (i=0; i<mts; i++)
    {
        printf("A : This is thead # %i\n", i);
    }
    printf("\n");

    // Worksharing inside a parallel section
#pragma omp parallel
    {
#pragma omp for
        for (i=0; i<mts; i++)
        {
            printf("B : This is thead # %i\n", i);
        }
    }
    printf("\n");

    // Worksharing twice inside a parallel section
#pragma omp parallel
    {
#pragma omp for
        for (i=0; i<mts; i++)
        {
            printf("C1: This is thead # %i\n", i);
        }
#pragma omp for
        for (i=0; i<mts; i++)
        {
            printf("C2: This is thead # %i\n", i);
        }
    }
    printf("\n");

```

```

// Explicit Worksharing inside a parallel section
#pragma omp parallel
{
    its = omp_get_thread_num();
    if (its == 0) {is=0; ie=24; }
    if (its == 1) {is=25; ie=49; }
    if (its == 2) {is=50; ie=74; }
    if (its == 3) {is=75; ie=100;}
    printf("DE: This is thead # %i,   Loop bounds : %i,%i\n", its, is, ie);
    for (i=is; i<ie; i++)
    {
        x[i] = x[i] + 1.;
    }
}
printf("\n");

// Parallel section and worksharing in one statement with ordered output
#pragma omp parallel for ordered default(none) private(i,j) shared(mts,y)
    for (i=0; i<mts; i++)
    {
        for (j=0; j<100; j++)
        {
            y[i][j] = y[i][j] + 1.;
        }
    }
#pragma omp ordered
    {
        printf("OR: This is thread # %i\n", i);
    }
    printf("\n");

    // Orphaned Worksharing
    printf("Orphaned work sharing within parallel region\n");
#pragma omp parallel
    {
        orphan(mts);
    }
    printf("\n");

    // Orphaned Worksharing, called outside of a parallel region
    printf("Orphaned outside of parallel region\n");
    orphan(mts);
    printf("\n");

    // Orphaned Worksharing
    printf("Orphaned inside of parallel region\n");
#pragma omp parallel
    {
        orphan_replicated(mts);
    }
    printf("\n");
}

```



```
void orphan(int mts)
{
    int i, its;
    its = omp_get_thread_num();
#pragma omp for
    for (i=0; i<mts; i++)
    {
        printf("O : This is thread #%i\n", its);
    }
}

void orphan_replicated(int mts)
{
    int i, its;
    its = omp_get_thread_num();
    for (i=0; i<mts; i++)
    {
        printf("O : This is thread #%i\n", its);
    }
}
```



```

nt main (int argc, char* argv[])

const int m = 100;
int  its, mts, i, j, icount, is, ie;
float x[m], y[4][m], sum, prod, t1, t2, t3;

// Preset mts, Inquire the number of Threads
mts = 4;
#ifdef _OPENMP
omp_set_num_threads(mts);
#endif
printf("Number of Threads is %i\n", mts);
printf("\n");

// Critical
icount = 0;
#pragma omp parallel for default(none) private(i,j) shared(y,icount)
for (i=0; i<4; i++)
{
    for (j=0; j<m; j++)
    {
        y[i][j] = y[i][j] + 1.;
    }
#pragma omp critical
    {
        icount = icount + 1;
    }
}
printf("CRITICAL      :: icount = %i\n", icount);
printf("\n");

// Critical with if construct
icount = 0;
#pragma omp parallel for default(none) private(i,j) shared(y,icount)
for (i=0; i<4; i++)
{
    for (j=0; j<m; j++)
    {
        y[i][j] = y[i][j] + 1.;
    }
    if (i < 2)
    {
#pragma omp critical
        {
            icount = icount + 1;
        }
    }
}
printf("CRITICAL + IF :: icount = %i\n", icount);
printf("\n");

```

```

// Single :: worksharing construct,
//          every thread has to encounter it
#pragma omp parallel default(none) private(i,j,its) shared(y,icount)
{
    its = omp_get_thread_num();
#pragma omp single
    {
        printf("Calculation has started, SINGLE, this is thread %i\n", its);
    }
#pragma omp for
    for (i=0; i<4; i++)
    {
        for (j=0; j<m; j++)
        {
            y[i][j] = y[i][j] + 1.;
        }
    }
    printf("\n");

// Master :: NOT a worksharing construct,
//          not every thread has to encounter it
#pragma omp parallel default(none) private(i,j,its) shared(y,icount)
{
    its = omp_get_thread_num();
#pragma omp master
    {
        printf("Calculation has started, MASTER, this is thread %i\n", its);
    }
#pragma omp for
    for (i=0; i<4; i++)
    {
        for (j=0; j<m; j++)
        {
            y[i][j] = y[i][j] + 1.;
        }
    }
}
printf("\n");

```



```

// Master :: NOT a worksharing construct,
//          not every thread has to encounter it
#pragma omp parallel default(none) private(i,j,its) shared(y,icount)
{
    its = omp_get_thread_num();
    if (its <= 1)
    {
#pragma omp master
    {
        printf("Calculation has started, MASTER, this is thread %i\n", it
    }
        printf("
                                Hi, this is thread %i\n", it
    }
}
#pragma omp for
for (i=0; i<4; i++)
{
    for (j=0; j<m; j++)
    {
        y[i][j] = y[i][j] + 1.;
    }
}
printf("\n");
}

```

PROGRAM EXAMPLE_OMP_05

!\$ USE OMP_LIB

!*** Scratch Array

INTEGER, PARAMETER :: M = 100 ! M has to be 100

REAL, DIMENSION(M) :: X

REAL, DIMENSION(M,4) :: Y

!*** Preset mts; Number of Threads has to be 4

mts = 4

!\$ call OMP_SET_NUM_THREADS(mts)

write (*,'(A,I2)') 'Number of Threads is set to ', MTS

write (*,*)

!*** Worksharing with NOWAIT: Independent variables

!\$OMP PARALLEL

!\$OMP DO

do i=0, M

 x(i) = x(i) + 1.

enddo

!\$OMP ENDDO NOWAIT

!\$OMP DO

do i=0, M

 y(i,1) = y(i,1) + 1.

enddo

!\$OMP END PARALLEL

!*** Worksharing with NOWAIT: Same variable

!\$OMP PARALLEL

!\$OMP DO

do i=0, M

 x(i) = x(i) + 1.

enddo

!\$OMP ENDDO NOWAIT

!\$OMP DO SCHEDULE(DYNAMIC,5)

do i=0, M

 x(i) = x(i) + 1.

enddo

!\$OMP END PARALLEL

END

