Parallel Computing for Science & Engineering CS395T

02/06/2018

Instructors:
Lars Koesterke, TACC
Charlie Dey, TACC

Example code

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.

```
PROGRAM EXAMPLE_OMP

!$ USE OMP_LIB

INTEGER, PARAMETER :: M = 40
REAL, DIMENSION(M) :: X, Y

!***  Preset MTS; Inquiry of the number of Threads
MTS = -1
!$ MTS = OMP_GET_MAX_THREADS()

!***  Alternative formulation with conditional compilation
!***  requires compilation with -fpp flag
MTS_A = -1
#ifdef _OPENMP
MTS_A = OMP_GET_MAX_THREADS()
#endif

!***  Serial or parallel mode
WRITE (0,*)
IF (MTS .LT. 0) THEN
  MTS          = 1
  WRITE (0,'(A)') 'You are in serial Mode!'
  WRITE (0,*)
ELSE
  WRITE (0,'(A,I4)') 'Number of Threads is : ', MTS
  WRITE (0,*)
ENDIF

!***  Setup of array, not (easily) parallelizable
X(1) = 0.1
DO I=2, M
  X(I) = X(I-1) + 0.1
ENDDO

!***  Setup of array, with parallelization
!$OMP PARALLEL DO SCHEDULE(DYNAMIC,2)
DO I=1, M
  X(I) = REAL(I) / 10.
ENDDO
```

```
!***  Calculating the Sum and the Product of the Array
!***  Uses implicit declaration of shared and private variab
!***  The loop index is private
!***  The Variable X is shared
SUM  = 0.
PROD = 1.
!$OMP PARALLEL DO REDUCTION(+:SUM) REDUCTION(*:PROD)
DO I=1, M
  SUM  = SUM  + X(I)
  PROD = PROD * X(I)
ENDDO
WRITE (0,'(A,ES15.8,4X,ES15.8)') 'Sum/Product = ’,SUM,PROD

!***  Calculation with private variables
!***  All variables are declared either shared or private
!$OMP PARALLEL DO DEFAULT(NONE) SHARED(X, Y) &
                  PRIVATE(I, T1, T2, T3)
DO I=2, M
  T1   = X(I-1) * X(I-1)
  T2   = X(I)   * X(I)
  T3   = X(I+1) * X(I+1)
  Y(I) = SQRT(T1 + T2 + T3)
ENDDO

!***  Loop with Load-imbalance, use of guided schedule
!***  count the number of updates in a critical region
ICOUNT = 0
!$OMP PARALLEL DO SCHEDULE(GUIDED)
DO I=1, M
  Y(I) = 0.
  DO J=I, M              ! The amount of work in this loop
    Y(I)   = Y(I) + X(J) !   decreases over time
!$OMP CRITICAL
    ICOUNT = ICOUNT + 1
!$OMP END CRITICAL
  ENDDO
ENDDO
WRITE (0,'(A,I6)') 'Number of updates = ', ICOUNT

WRITE (0,*)
END
```

```
ifort –openmp …
ifort –fpp …
```

```
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>

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

  const int m = 40;
  int  mts, i, j, icount;
  float x[m], y[m], sum, prod, t1, t2, t3;

  // Preset mts, Inquire the number of Threads
  mts = -1;
#ifdef _OPENMP
  mts = omp_get_max_threads();
#endif
  printf("\n");
  if (mts == -1)
    {
      mts = 1;
      printf("You are in serial Mode!\n");
    }
  else
    {
      printf("Number of Threads is %i\n", mts);
    }

  // Setup of array, not (easily) parallelizable
  x[0] = 0.1;
  for (i=1; i<m; i++)
    {
      x[i] = x[i-1] + 0.1;
    }

  // Setup of array, with parallelization
#pragma omp parallel for
  for (i=0; i<m; i++)
    {
      x[i] = (float)(i+1) / 10.;
    }
```

icc –openmp …

```
 // Calculating the Sum and the Product of the Array
  // Uses implicit declaration of shared and private variables
  // The loop index is private
  // The Variable X is shared
  sum  = 0.;
  prod = 1.;
#pragma omp parallel for reduction(+:sum) reduction(*:prod)
  for (i=0; i<m; i++)
    {
      sum  = sum + x[i];
      prod = prod * x[i];
    }
  printf("Sum/Product = %15.8e, %15.8e\n", sum, prod);

  // Calculation with private variables
  // All variables are declared either shared or private
#pragma omp parallel for default(none) shared(x, y) \
                         private(i, t1, t2, t3)
  for (i=1; i<m-1; i++)
    {
      t1   = x[i-1] * x[i-1];
      t2   = x[i]   * x[i];
      t3   = x[i+1] * x[i+1];
      y[i] = sqrt(t1 + t2 + t3);
    }
  // Loop with Load-imbalance, use of guided schedule
  // count the number of updates in a critical region
  icount = 0;
#pragma omp parallel for schedule(guided) shared(icount) private(j)
  for (i=0; i<m; i++)
    {
      y[i] = 0.;
      for (j=i; j<m; j++)
            {
              y[i] = y[i] + x[j];
#pragma omp critical
              {
                icount = icount + 1;
              }
            }
    }
  printf("Number of updates = %6i\n", icount);
  return 0;
}
```

icc –Wno-unknown-pragmas …