Domain Decomposition and Large Scale Scientific Computing

Homework 2

By

Domingo Eugenio Cattoni Correa

Master in numerical method in engineering

1. Scope

The scope of this homework is to implement a matrix – vector product code in Fortran 95 in order to perform the following task:

1. Matrix – vector y = y+A*x, where $A \in R^n x R^n$ is dense matrix and x, y $\in R^n$.
2. Parallelize the code developed in the previous step using OpenMP.
3. Use SCHEDULE strategies and several chunk sizes.

1.1. Code

Figure 1 shows the part of the code where matrix-vector product was implemented.

```
do i = 1,size(x)
      do j = 1,size(A,2)
      y(i) = y(i)+A(j,i)*x(i)
     end do
 end do
```

Figure 1: Matrix – vector product.

It can be seen that was used two nested loops accessing the matrix A by the row.

1.2. Parallelization

It was used OpenMP in order to parallelise the code developed previously. Next figure shows the parallelized region.

```
!$OMP  PARALLEL DEFAULT(NONE) SHARED(A,x,y) PRIVATE(i,j)
!$OMP  DO
 do i = 1,size(x)
       do j = 1,size(A,2)
       y(i) = y(i)+A(j,i)*x(i)
      end do
  end do
!$OMP  END DO
!$OMP  END PARALLEL
```

Figure 2: Parallelized region.

When several nested do-loops are present, it is always convenient to parallelize the outer most one, since then the amount of work distributed over the different threads is maximal.

According to the paragraph written above can be observed that only the row loop was parallelized.

1.3. Schedule strategies

```
!$OMP  PARALLEL DEFAULT(NONE) SHARED(A,x,y) PRIVATE(i,j)
!$OMP  DO SCHEDULE (STATIC,500)
 do i = 1,size(x)
       do j = 1,size(A,2)
       y(i) = y(i)+A(j,i)*x(i)
      end do
  end do
!$OMP  END DO
!$OMP  END PARALLEL
```

a)

```
!$OMP  PARALLEL DEFAULT(NONE) SHARED(A,x,y) PRIVATE(i,j)
!$OMP  DO SCHEDULE (STATIC,1000)
 do i = 1,size(x)
       do j = 1,size(A,2)
       y(i) = y(i)+A(j,i)*x(i)
      end do
  end do
!$OMP  END DO
!$OMP  END PARALLEL
```

b)

```
!$OMP  PARALLEL DEFAULT(NONE) SHARED(A,x,y) PRIVATE(i,j)
!$OMP  DO SCHEDULE (DYNAMIC)
 do i = 1,size(x)
       do j = 1,size(A,2)
       y(i) = y(i)+A(j,i)*x(i)
      end do
  end do
!$OMP  END DO
!$OMP  END PARALLEL
```

c)

Figure 3: Different schedule strategies. a) Schedule Static with 500 chunks, b) Schedule Static with 1000 chunks, c) Schedule Dynamic.

Figure above shows different schedule strategies. Two of them were defined as static type and 500 and 1000 chunks, the last one was defined as dynamic type.

As a comparation, Table 1shows the elapsed time of different numbers of threads and different strategies.

Table 1: Elapsed time.

	Elapsed time [seg.]					
Type	**1 Thread**	**2 Threads**	**4 Threads**	**8 Threads**	**16 Threads**	**24 Threads**
Serial	1,30E-01	1,20E-01	1,60E-01	1,50E-01	1,20E-01	1,20E-01
Parallel Std	1,30E-01	1,10E-01	1,00E-01	9,50E-02	9,7E-02	1,00E-01
Parallel static (chunk = 500)	1,30E-01	1,20E-01	1,02E-01	9,80E-02	9,90E-02	1,00E-01
Parallel static (chunk = 1250)	1,40E-01	1,10E-02	1,00E-01	1,00E-01	1,00E-01	1,00E-01
Parallel Dynamic	1,00E-01	1,00E-01	1,00E-01	1,00E-01	1,00E-01	1,00E-01

It can be seen that the different strategies of parallelization (Parallel, schedule static with different numbers of chunk, schedule dynamic) have shown less elapsed time than serial type. But, it can be observed that these differences were not significate as it expected. The problem was that this code was run on login node.