# Parallel Programming

## LAB 3 - 20th August 2018

**Note: Write all programs in your observation book and record the results. Get the signature of faculty /teaching assistance.**

**Objective: To learn the concept of schedule in *for* Directive**

1. The number of threads set can be checked at command prompt using
   **echo $OMP_NUM_THREADS**
   The number of threads can be set at command prompt using
   **export OMP_NUM_THREADS=4**

2. The format for *for* directive is as follows.

```
#pragma omp for [clause ...]  newline
            schedule (type [,chunk])
            ordered
            private (list)
            firstprivate (list)
            lastprivate (list)
            shared (list)
            reduction (operator: list)
            collapse (n)
            nowait

   for_loop
```

**Schedule(type [,chunk]**

**SCHEDULE**: Describes how iterations of the loop are divided among the threads in the team. The default schedule is implementation dependent.

STATIC

Loop iterations are divided into pieces of size *chunk* and then statically assigned to threads. If chunk is not specified, the iterations are evenly (if possible) divided contiguously among the threads.

DYNAMIC

Loop iterations are divided into pieces of size *chunk*, and dynamically scheduled among the threads; when a thread finishes one chunk, it is dynamically assigned another. The default chunk size is 1.

GUIDED

Iterations are dynamically assigned to threads in blocks as threads request them until no blocks remain to be assigned. Similar to DYNAMIC except that the block size decreases each time a parcel of work is given to a thread. The size of the initial block is proportional to:

**number_of_iterations / number_of_threads**

Subsequent blocks are proportional to
**number_of_iterations_remaining / number_of_threads**
The chunk parameter defines the minimum block size. The default chunk size is 1.

RUNTIME

The scheduling decision is deferred until runtime by the environment variable OMP_SCHEDULE. It is illegal to specify a chunk size for this clause.

AUTO

The scheduling decision is delegated to the compiler and/or runtime system.

For simplicity, we assume that we have a loop of 16 iterations, which has been parallelized by OpenMP, and that we are about to execute that loop using 2 threads.

In **default scheduling**

- thread 1 is assigned to do iterations 1 to 8;
- thread 2 is assigned to do iterations 9 to 16.

In **static scheduling**, using a "chunksize" of 4:

- thread 1 is assigned to do iterations 1 to 4 and 9 to 12.
- thread 2 is assigned to do iterations 5 to 8 and 13 to 16.

In **dynamic scheduling**, using a "chunksize" of 3:

- thread 1 is assigned to do iterations 1 to 3.
- thread 2 is assigned to do iterations 4 to 6.

The next chunk is iterations 7 to 9, and will be assigned to whichever thread finishes its current work first, and so on until all work is completed.

**Consider following example program:**

This program explores the use of the for work-sharing construct. The program provided here adds two vectors together using a work-sharing approach to assign work to threads:

```
#include<omp.h>
#include<stdio.h>
#include<stdlib.h>
#define CHUNKSIZE 10
 #define N 100
int main (int argc, char *argv[]) {
int nthreads, tid, i, chunk;
 float a[N], b[N], c[N];
for (i=0; i < N; i++)
```

```
    a[i] = b[i] = i * 1.0; // initialize arrays
chunk = CHUNKSIZE;
#pragma omp parallel shared(a,b,c,nthreads,chunk) private(i,tid) {
     tid = omp_get_thread_num();
    if (tid == 0)
    {
        nthreads = omp_get_num_threads();
        printf("Number of threads = %d\n", nthreads);

     }

        printf("Thread %d starting...\n",tid);
        #pragma omp for schedule(static,chunk)
        for (i=0; i<N;i++)
        {
              c[i]=a[i]+b[i];
            printf(“Thread %d: c[%d]=%f\n”,tid,i,c[i]);
        }
    } /*end of parallel section*/
    }
```

i) Observe following results from executing above program with **schedule(static,chunk)**.

a) Note down the range of data elements provided for each thread by setting number of threads =5 and chunk size=10

b) Note down the range of data elements provided for each thread by setting number of threads =5 and chunk size =25

ii) Observe following results from executing above program with **schedule(dynamic,chunk)**.

a) Note down the range of data elements provided for each thread by setting number of threads =5 and chunk size=10

b) Note down the range of data elements provided for each thread by setting number of threads =8 and chunk size =10

iii) Observe following results from executing above program with **schedule(guided,chunk)**.

a) Note down the range of data elements provided for each thread by setting number of threads =5 and chunk size=10

b) Note down the range of data elements provided for each thread by setting number of threads =5 and chunk size=5

c) Note down the range of data elements provided for each thread by setting number of threads =8 and chunk size =5

**Briefly explain what you have understood by running the programs with various scheduling methods. Also mention, when each type of scheduling is useful in programming.**