Parallel Programming

LAB 2

16th August 2017

Objectives:

- To understand the working of *#pragma omp parallel* and *#pragma omp for*.
- To understand the scheduling clause with *static, dynamic and guided*.
- To learn the working of clauses such as *collapse, reduction and ordered*.

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

Objective 1 and 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.

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. hread by setting number of threads =8 and chunk size =10

Objective 3

1. **Execute following code and observe the results with and without collapse clause.**

```
#include <omp.h>
#include <stdio.h>
void main()
{
int j, k, a;
```

```
#pragma omp parallel num_threads(2)
{
#pragma omp for collapse(2) ordered private(j,k) schedule(static,3)
for (k=1; k<=3; k++)
for (j=1; j<=2; j++)
{
#pragma omp ordered
printf("%d %d %d\n", omp_get_thread_num(), k, j);
/* end ordered */
}
}
}
```

2. **Execute following code and observe the results with and without ordered clause.**

```
#include <omp.h>
#include <stdio.h>
void main()
{
int i,myval;
#pragma omp parallel for private(myval) ordered
   for(i=1; i<=10; i++)
     {
      myval = i+2;
      #pragma omp ordered
          printf("%d %d\n", i, myval);
     }
}
```

3. **Execute the following program and analyze the result with and without using reduction().**

```
#include<omp.h>
#include<stdio.h>
int main(){
int i,n,chunk;
int result=0;
int a[20],b[20],c[20];
n=20;
chunk=5;
/*initializing array*/
for(i=0;i<n;i++)
{ a[i]=i*2;
```

```
b[i]=i*3;
}
#pragma omp parallel for default(shared) num_threads(4)
private(i) schedule(static,chunk) reduction(+:result)
for(i=0;i<n;i++)
result=result+(a[i]*b[i]);

printf("Final result=%d\n",result);
}
```

4. **Write a C/C++ OpenMP program to find ROWSUM and COLUMNSUM of a matrix a[n][n].**
5. **Write a C/C++ OpenMP program to perform matrix multiplication.**
6. **The details of an employ with employ id and employ salary is stored in a two dimensional array. The company would like to raise the salary of all its employees by 6%. If the increase in salary is more than 5,000 Rs, the company would like to put tax of 2% on the increased amount more than 5,000 Rs. Calculate the total extra amount the company need to spend by increasing the salary 6%.**

Note: Write the program, results and analysis in your observation book and get signature from Teaching Assistants.