DEPARTMENT OF INFORMATION TECHNOLOGY, NITK SURATHKAL

Parallel Programming

LAB 1 -5th August 2020

Note: Observe the results of each program, take the screenshot of the result and upload it in the Moodle.

Note:

parallel
Forms a team of threads and starts parallel execution.
#pragma omp parallel *[clause[[,]clause] ...]*
structured-block
clause:
if(*scalar-expression*)
num_threads(*integer-expression*)
default(**shared** | **none**)
private(*list*)
firstprivate(*list*)
shared(*list*)
copyin(*list*)
reduction(*reduction-identifier*: *list*)

---

loop Specifies that the iterations of associated loops will be executed in parallel by threads in the team in the context of their implicit tasks.
#pragma omp for *[clause[[,]clause] ...]*
for-loops
clause:
private(*list*)
firstprivate(*list*)
lastprivate(*list*)
reduction(*reduction-identifier*: *list*)
schedule(*kind[, chunk_size]*)
collapse(*n*)
ordered

nowait

kind:

• **static:** Iterations are divided into chunks of size *chunk_size* and assigned to threads in the team in round-robin fashion in order of thread number.

• **dynamic:** Each thread executes a chunk of iterations then requests another chunk until none remain.

• **guided:** Each thread executes a chunk of iterations then requests another chunk until no chunks remain to be assigned.

• **auto:** The decision regarding scheduling is delegated to the compiler and/or runtime system.

• **runtime:** The schedule and chunk size are taken from the *run-sched-var* ICV.

I. Finding number of CPU s in system

a) lscpu command

```
  $ lscpu
$ lscpu | egrep 'Model name|Socket|Thread|NUMA|CPU\(s\)'
$ lscpu -p
```

```
axebell ~ lscpu
Architecture:                    x86_64
CPU op-mode(s):                  32-bit, 64-bit
Byte Order:                      Little Endian
Address sizes:                   39 bits physical, 48 bits virtual
CPU(s):                          8
On-line CPU(s) list:             0-7
Thread(s) per core:              2
Core(s) per socket:              4
Socket(s):                       1
NUMA node(s):                    1
```

```
axebell ~ lscpu | egrep 'Model name|Socket|Thread|NUMA|CPU\(s\)'
CPU(s):                          8
On-line CPU(s) list:             0-7
Thread(s) per core:              2
Socket(s):                       1
NUMA node(s):                    1
Model name:                      Intel(R) Core(TM) i5-8300H CPU @ 2.30GHz
NUMA node0 CPU(s):               0-7
```

b)Run top ot htop command to obtain the number of CPUs/cores in linux

```
$top
```

c) Execute nproc print the number of CPUs available on Linux

```
$ nproc --all
$ echo "Threads/core: $(nproc --all)"
```

1.Write a C/C++ simple parallel program to display the *thread_id* and total number of threads.

/*simpleomp.c*/

#include<omp.h>

int main(){

int nthreads,tid;

#pragma omp parallel private(tid)

{

tid=omp_get_thread_num();

printf("Hello world from thread=%d\n",tid);

if(tid==0)

{

nthreads=omp_get_num_threads();

printf("Number of threads=%d\n",nthreads);

}

}

}

Execute the program as follows:

$gcc –o simple –fopenmp simpleomp.c

$export OMP_NUM_THREADS=2

$./simple

<u>Note down the output in your observation book.</u>

Number of threads in a parallel region is determined by the *if* clause, *num_threads(),omp_set_num_threads(), OMP_NUM_THREADS.*

Use these various methods to set the number of threads and mention the method of setting the same.

```
axebell … course.content pc301Lab programs 1 export OMP_NUM_THREADS=2
axebell … course.content pc301Lab programs ./simpleomp
Hello from thread = 1
Hello from thread = 0
Number of threads = 2
```

```c
#include<omp.h>
#include<stdio.h>
int main() {
    int nthreads, tid;
    omp_set_num_threads(3); //setting number of threads overrides OMP_NUM_THREADS
    #pragma omp parallel private(tid)
    {
        tid = omp_get_thread_num();
        printf("Hello from thread = %d\n", tid);
        if(tid == 0) {
            nthreads = omp_get_num_threads();
            printf("Number of threads = %d\n",nthreads);
        }
    }
}
```

Setting no of threads using omp_set_num_threads();

```
axebell … course.content pc301Lab programs gcc -o simpleomp -fopenmp simpleomp.c
axebell … course.content pc301Lab programs ./simpleomp
Hello from thread = 1
Hello from thread = 0
Number of threads = 3
Hello from thread = 2
```

2. Check the output of following program:

/*ifparallel.c*/

#include<omp.h>

int main(){

int val;

printf("Enter 0: for serial 1: for parallel\n");

scanf("%d",&val);

#pragma omp parallel if(val)

{

if(omp_in_parallel())

printf(“Parallel val=%d id= %d\n”,val, omp_get_thread_num());

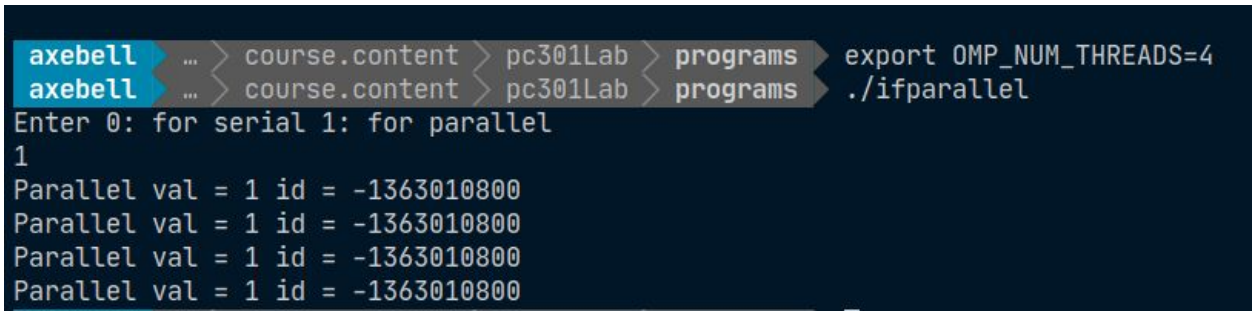
else

printf(“Serial val=%d id= %d\n”,val, omp_get_thread_num());

}

}

Note down the output in your observation book.



3.Observe and record the output of following program

/*num_threads.c*/

#include<omp.h>

int main(){

#pragma omp parallel num_threads(4)

{

int tid=omp_get_thread_num();

printf(“Hello world from thread=%d\n”,tid);

}

}

```
axebell … course.content pc301Lab programs gcc -o num_threads -fopenmp num_threads.c
axebell … course.content pc301Lab programs ./num_threads
Hello word from thread = 2
Hello word from thread = 0
Hello word from thread = 3
Hello word from thread = 1
```

4.Write a C/C++ parallel program for adding corresponding elements of two arrays.

/*addarray.c*/

#include<omp.h>

int main(){

int i,n,chunk;

int a[20],b[20],c[20];

n=20;

chunk=2;

/*initializing array*/

for(i=0;i<n;i++)

{ a[i]=i*2;

  b[i]=i*3;

}

#pragma omp parallel for default(shared) private(i) schedule(static,chunk)

{

for(i=0;i<n;i++)

{

c[i]=a[i]+b[i];

printf("Thread id= %d i=%d,c[%d]=%d\n", omp_get_thread_num(),i,i,c[i]);

}

}

Check the output by varying

1. **Chunk size**
2. **Number of threads**
   Note down the allotment of i range for each thread.

```
axebell … course.content pc301Lab programs export OMP_NUM_THREADS=4
axebell … course.content pc301Lab programs gcc -o addarray -fopenmp addarray.c
axebell … course.content pc301Lab programs ./addarray
Chunk=2Thread id = 0 i = 0, c[0] = 0
Thread id = 0 i = 1, c[1] = 5
Thread id = 0 i = 8, c[8] = 40
Thread id = 0 i = 9, c[9] = 45
Thread id = 0 i = 16, c[16] = 80
Thread id = 0 i = 17, c[17] = 85
Thread id = 2 i = 4, c[4] = 20
Thread id = 2 i = 5, c[5] = 25
Thread id = 2 i = 12, c[12] = 60
Thread id = 2 i = 13, c[13] = 65
Thread id = 3 i = 6, c[6] = 30
Thread id = 3 i = 7, c[7] = 35
Thread id = 1 i = 2, c[2] = 10
Thread id = 1 i = 3, c[3] = 15
Thread id = 1 i = 10, c[10] = 50
Thread id = 1 i = 11, c[11] = 55
Thread id = 1 i = 18, c[18] = 90
Thread id = 3 i = 14, c[14] = 70
Thread id = 3 i = 15, c[15] = 75
Thread id = 1 i = 19, c[19] = 95
```

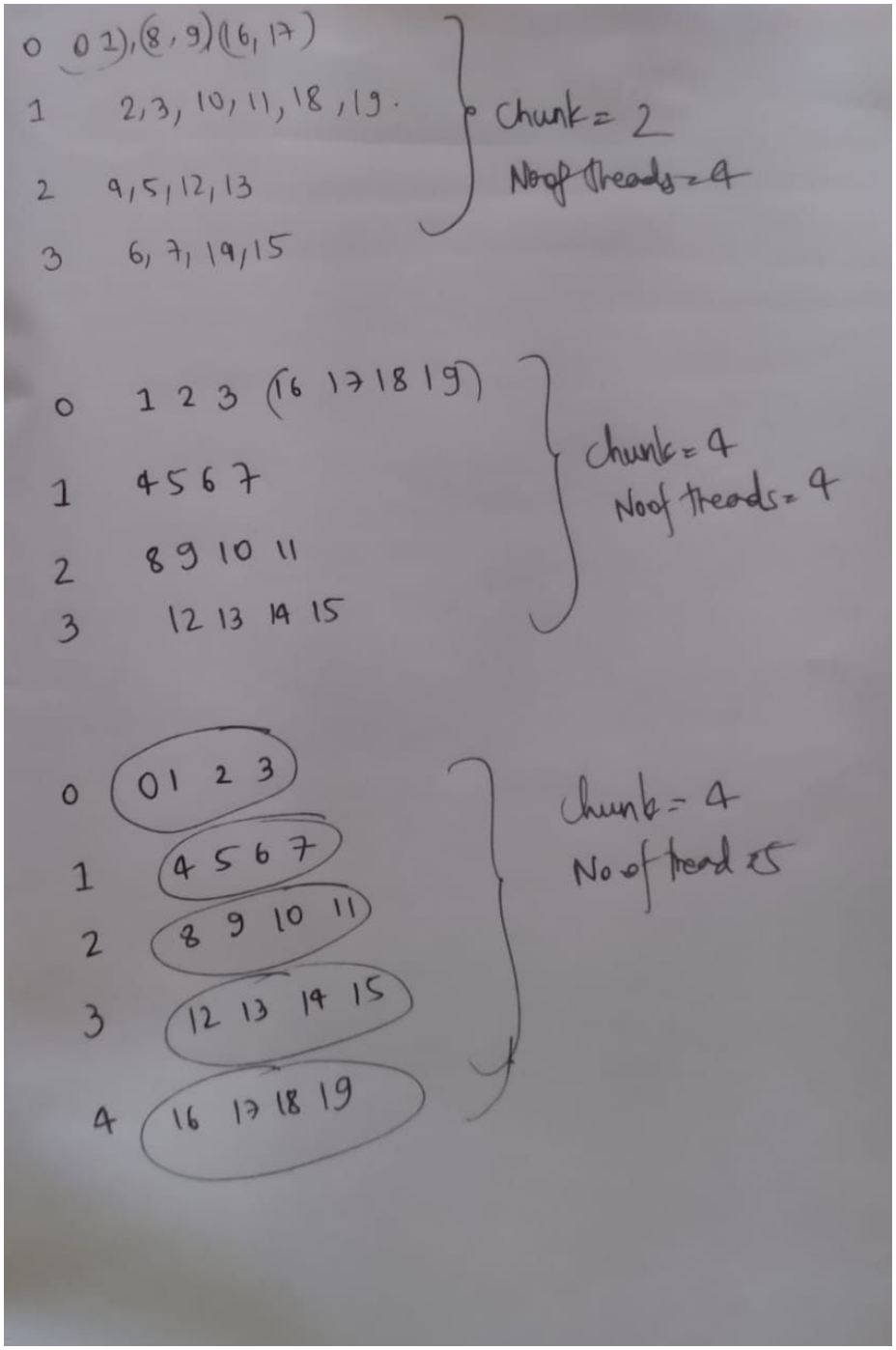
Chunk = 2 Thread = 4

```
axebell … course.content pc301Lab programs gcc -o addarray -fopenmp addarray.c
axebell … course.content pc301Lab programs ./addarray
Chunk=4Thread id = 0 i = 0, c[0] = 0
Thread id = 0 i = 1, c[1] = 5
Thread id = 0 i = 2, c[2] = 10
Thread id = 0 i = 3, c[3] = 15
Thread id = 0 i = 16, c[16] = 80
Thread id = 0 i = 17, c[17] = 85
Thread id = 3 i = 12, c[12] = 60
Thread id = 3 i = 13, c[13] = 65
Thread id = 3 i = 14, c[14] = 70
Thread id = 3 i = 15, c[15] = 75
Thread id = 0 i = 18, c[18] = 90
Thread id = 0 i = 19, c[19] = 95
Thread id = 1 i = 4, c[4] = 20
Thread id = 1 i = 5, c[5] = 25
Thread id = 2 i = 8, c[8] = 40
Thread id = 2 i = 9, c[9] = 45
Thread id = 2 i = 10, c[10] = 50
Thread id = 2 i = 11, c[11] = 55
Thread id = 1 i = 6, c[6] = 30
Thread id = 1 i = 7, c[7] = 35
time spent, 0.001034 axebell … course.content pc301Lab programs export OMP_NUM_THREADS=5
```

Chunk = 4 Threads = 4

```
time spent, 0.001034 axebell … course.content pc301Lab programs export OMP_NUM_THREADS=5
axebell … course.content pc301Lab programs gcc -o addarray -fopenmp addarray.c
axebell … course.content pc301Lab programs ./addarray
Chunk=4Thread id = 0 i = 0, c[0] = 0
Thread id = 0 i = 1, c[1] = 5
Thread id = 0 i = 2, c[2] = 10
Thread id = 0 i = 3, c[3] = 15
Thread id = 4 i = 16, c[16] = 80
Thread id = 4 i = 17, c[17] = 85
Thread id = 4 i = 18, c[18] = 90
Thread id = 4 i = 19, c[19] = 95
Thread id = 2 i = 8, c[8] = 40
Thread id = 2 i = 9, c[9] = 45
Thread id = 2 i = 10, c[10] = 50
Thread id = 2 i = 11, c[11] = 55
Thread id = 1 i = 4, c[4] = 20
Thread id = 1 i = 5, c[5] = 25
Thread id = 1 i = 6, c[6] = 30
Thread id = 1 i = 7, c[7] = 35
Thread id = 3 i = 12, c[12] = 60
Thread id = 3 i = 13, c[13] = 65
Thread id = 3 i = 14, c[14] = 70
Thread id = 3 i = 15, c[15] = 75
```

Chunk = 4 Threads = 5


0 0 1), (8, 9) ((6, 17)
1 2, 3, 10, 11, 18, 19.
2 4, 5, 12, 13
3 6, 7, 14, 15
Chunk = 2
Noof threads = 4
0 1 2 3 (16 17 18 19)
1 4 5 6 7
2 8 9 10 11
3 12 13 14 15
Chunk = 4
Noof threads = 4
0 01 2 3
1 4 5 6 7
2 8 9 10 11
3 12 13 14 15
4 16 17 18 19
Chunk = 4
No of thread = 5