National Institute of Technology Karnataka Surathkal
Department of Information Technology

IT 301 Parallel Computing

Shared Memory Programming Technique (4)

OpenMP : *for –schedule*

Dr. Geetha V
Assistant Professor
Dept of Information Technology
NITK Surathkal

Index

Course Outline

Course Plan: Theory:

Part A: Parallel Computer Architectures

Week 1,2,3: ***Introduction to Parallel Computer Architecture:*** Parallel Computing, Parallel architecture, bit level, instruction level , data level and task level parallelism. Instruction level parelllelism: pipelining(Data and control instructions), scalar and superscalar processors, vector processors. Parallel computers and computation.

Week 4,5: Memory Models: UMA, NUMA and COMA. Flynns classification, Cache coherence,

Week 6,7: Amdahl's Law. Performance evaluation, Designing parallel algorithms : Divide and conquer, Load balancing, Pipelining.

Week 8 -11: ***Parallel Programming techniques like Task Parallelism using TBB, TL2, Cilk++ etc. and software transactional memory techniques.***

Course Outline

Part B: OpenMP/MPI/CUDA

Week 1,2,3 : ***Shared Memory Programing Techniques:*** Introduction to OpenMP : Directives: *parallel, for, sections, task, master, single, critical, barrier, taskwait, atomic. Clauses: private, shared, firstprivate, lastprivate, reduction, nowait, ordered, schedule, collapse, num_threads, if().*

Week 4,5: ***Distributed Memory programming Techniques:*** MPI: Blocking, Non-blocking.

Week 6,7 : CUDA : OpenCL, Execution models, GPU memory, GPU libraries.

Week 10,11,: ***Introduction to accelerator programming using CUDA/OpenCL and Xeon-phi. Concepts of Heterogeneous programming techniques.***

Practical:

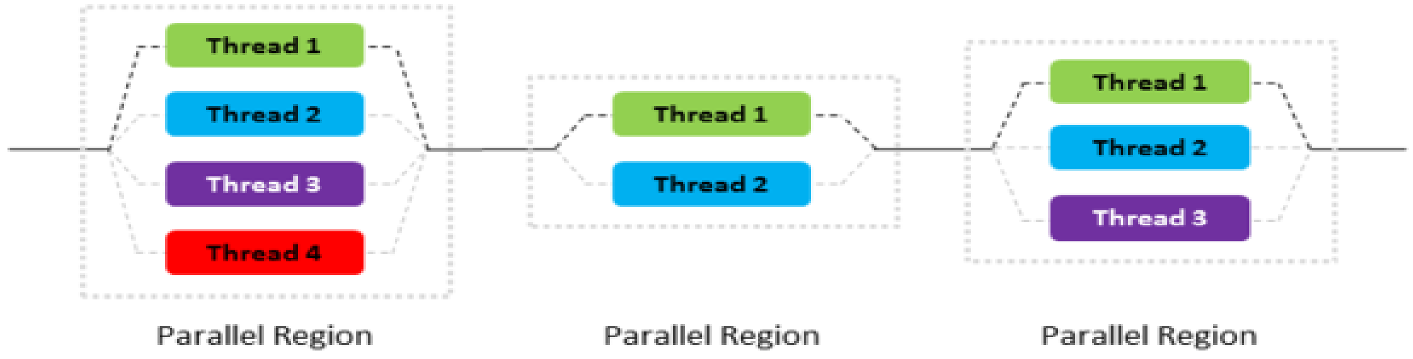
Implementation of parallel programs using OpenMP/MPI/CUDA.

Assignment: Performance evaluation of parallel algorithms (in group of 2 or 3 members)

1. OpenMP

FORK – JOIN Parallelism

- OpenMP program begin as a single process: the master thread. The master thread executes sequentially until the first parallel region construct is encountered.
- When a parallel region is encountered, master thread
  - Create a group of threads by FORK.
  - Becomes the master of this group of threads and is assigned the thread id 0 within the group.
- The statement in the program that are enclosed by the parallel region construct are then executed in parallel among these threads.
- JOIN: When the threads complete executing the statement in the parallel region construct, they synchronize and terminate, leaving only the master thread.



2. OpenMP Programming: Directives : Parallel, For

#pragma omp parallel *[clause[,]clause…] new-line*

Structured-block

Clause: **if(***scalar-expression***)**

num_threads(*integer-expression***)**

default(*shared|none***)**

private(*list***)**

firstprivate(*list)*

shared(*list)*

copyin(*list)*

reduction(*operator:list)*

#pragma omp for *[clause[,]clause…] new-line*

for-loops

Clause: **private(***list***)**

firstprivate(*list)*

lastprivate(*list)*

reduction(*operator:list)*

schedule(*kind[,chunk_size]***)**

collapse(*n)*

ordered

nowait

2. OpenMP Programming: Clauses : Schedule

#pragma omp for *[clause[,]clause…] new-line*

for-loops

Clause: **private**(*list*)

firstprivate(*list)*

lastprivate(*list)*

reduction(*operator:list)*

schedule(*kind[,chunk_size]*)

collapse(*n)*

ordered

nowait

Schedule(kind[,chunksize]) Clause

- Schedule clause specifies how iteration of the loop are divided into contiguous non-empty subsets, called chunks, and how these chunks are assigned among threads of the team.
- Kind: It has following kind.
  - Static
  - Dynamic
  - Guided
  - runtime

2. OpenMP Programming: *schedule(static, chunk_size)*

#pragma omp for *[clause[,]clause...] new-line*

for-loops

Clause: **private**(*list*)

firstprivate(*list)*

lastprivate(*list)*

reduction(*operator:list)*

schedule(*kind[,chunk_size]*)

collapse(*n)*

ordered

nowait

Schedule(static, chunksize]) Clause

- Iterations are divided into chunk of size chunk_size.
- Chunks are statically assigned to threads in round robin fashion in the order of thread number
- Last chunk to be assigned may have smaller number of iterations.
- When no chunk size is specified, iterations/threads
- Example: 28 iteration, threads= 4
- Schedule(static, 5)

thread0	thread1	thread2	Thread3	thread0	thread1
0-4	5-9	10-14	15-19	20-24	25-27

2. OpenMP Programming: *schedule(dynamic, chunk_size)*

#pragma omp for *[clause[,]clause…] new-line*

for-loops

Clause: **private**(*list*)

firstprivate(*list)*

lastprivate(*list)*

reduction(*operator:list)*

schedule(*kind[,chunk_size]*)

collapse(*n)*

ordered

nowait

Schedule(Dynamic, chunksize]) Clause

- Iterations are assigned to threads in chunksize as the threads request them.
- Thread executes the chunk of iteration and then requests another chunk, until all iterations are complete.
- Each chunk contains chunksize except for the last chunk assigned.
- Example: 28 iteration, threads= 4
- Schedule(dynamic, 5)

thread1	thread3	thread0	thread2	thread1	thread2
0-4	5-9	10-14	15-19	20-24	25-27

2. OpenMP Programming: *schedule(guided, chunk_size)*

Schedule(Guided, chunksize]) Clause

- Iterations are assigned to threads of chunksize as the threads request them.
- Thread executes the chunk of iteration and then requests another chunk, until all iterations are complete.
- Chunk = remaining iterations / #threads
- Chunk size determines the minimum size of chunk , except lst chunk.
- Default value of chunk_size =1

- Chunk = remaining iterations / #threads
- Example: 28 iteration, threads= 4
- Schedule(guided,3)
- 28/4 = 7 [remaining = 28-7=21]
- 21/4=5.2 => 6 [remaining =21-6 = 15]
- 15/4=3.7=>4 [remaining = 15-4 = 11]
- 11/4=2.7 =>3 [remaining =11-3=8]
- 8/4 = 2 [min is chunk size 3 . So assign 3:]
- [remaining =8-3 = 5]
- 5/4 = 1 [min = 3: remaining : 2]
- 2<=3 , so last chunk = 2

thread2	thread1	thread0	thread3	thread2	thread2	thread2
0-6 (7)	7-12(6)	13-16(4)	17-19(3)	20-22 (3)	23-25(3)	26-27(2)

2. OpenMP Programming: s*chedule(runtime)*

#pragma omp for *[clause[,]clause...] new-line*

for-loops

Clause: **private**(*list*)

firstprivate(*list)*

lastprivate(*list)*

reduction(*operator:list)*

schedule(*kind[,chunk_size]*)

collapse(*n)*

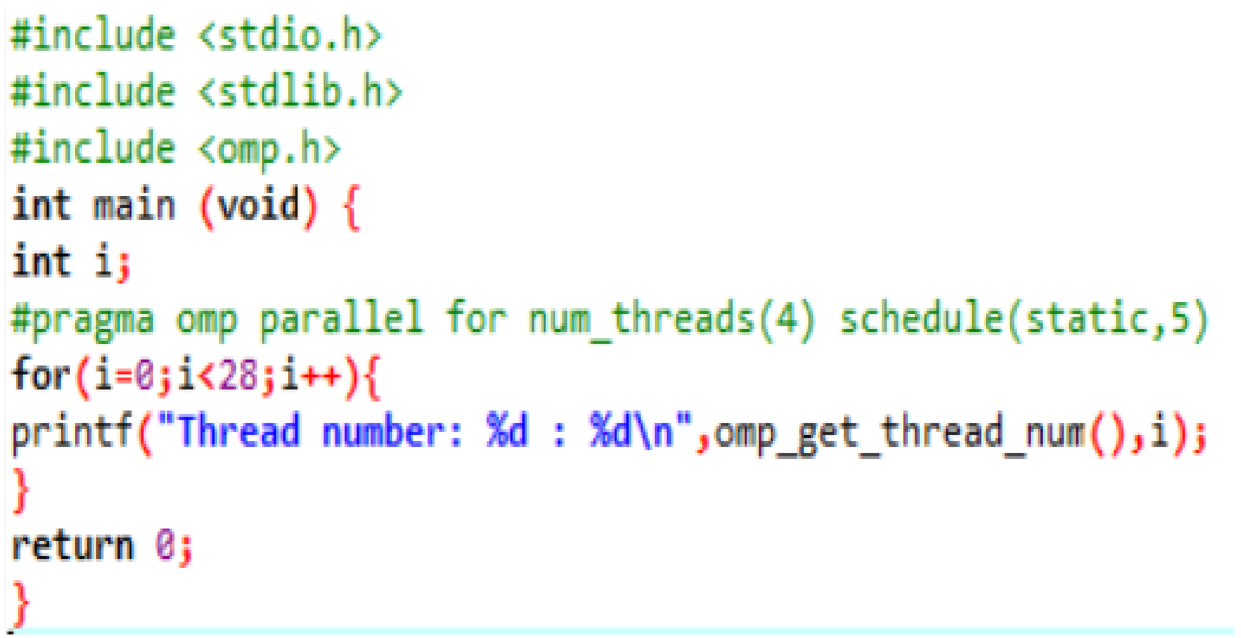
ordered

nowait

Schedule(runtime) Clause

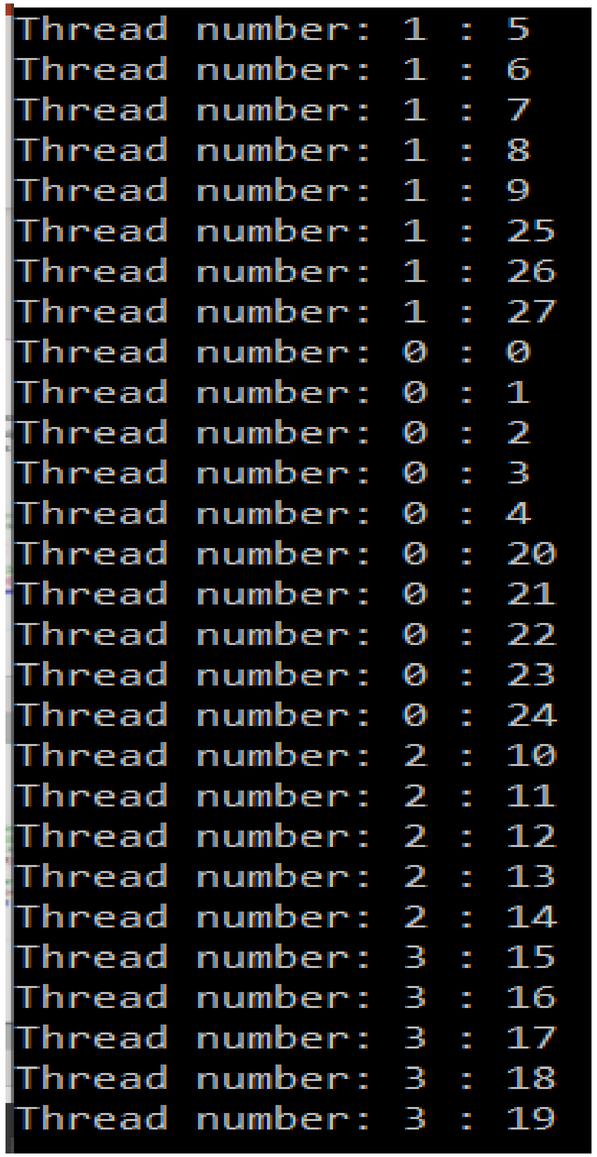
- The decision regarding scheduling is defered until run time, and the schedule and chunk size are taken from the *run-sched-var* control variable.

2. OpenMP Programming: *schedule(static, chunk_size)*



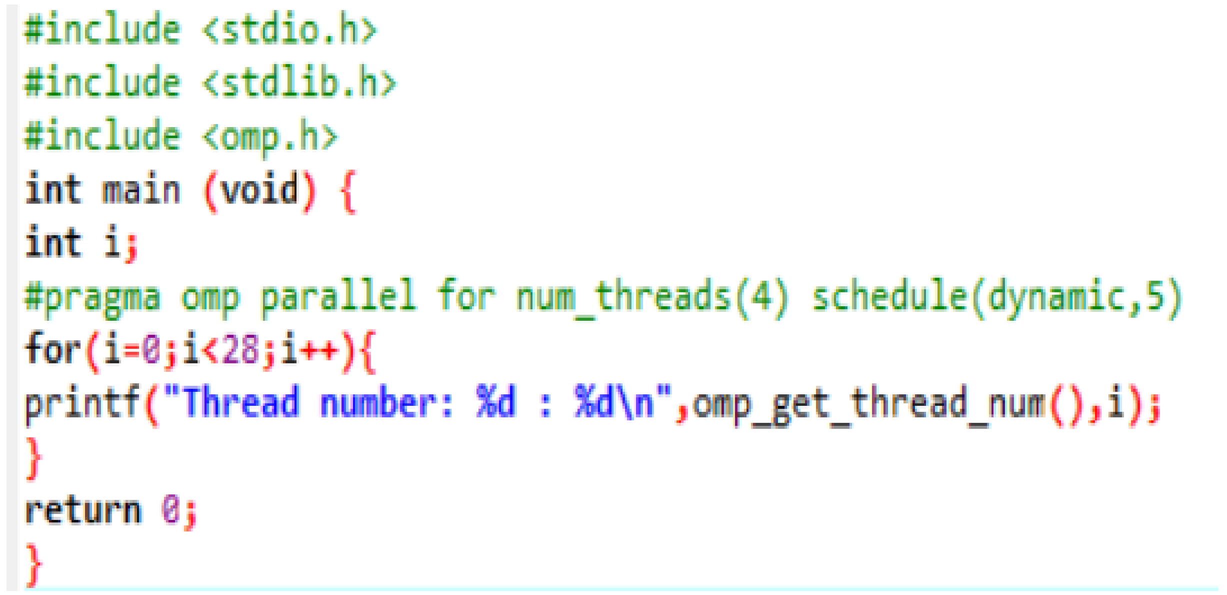
```
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
int main (void) {
int i;
#pragma omp parallel for num_threads(4) schedule(static,5)
for(i=0;i<28;i++){
printf("Thread number: %d : %d\n",omp_get_thread_num(),i);
}
return 0;
}
```

thread0	thread1	thread2	Thread3	thread0	thread1
0-4	5-9	10-14	15-19	20-24	25-27



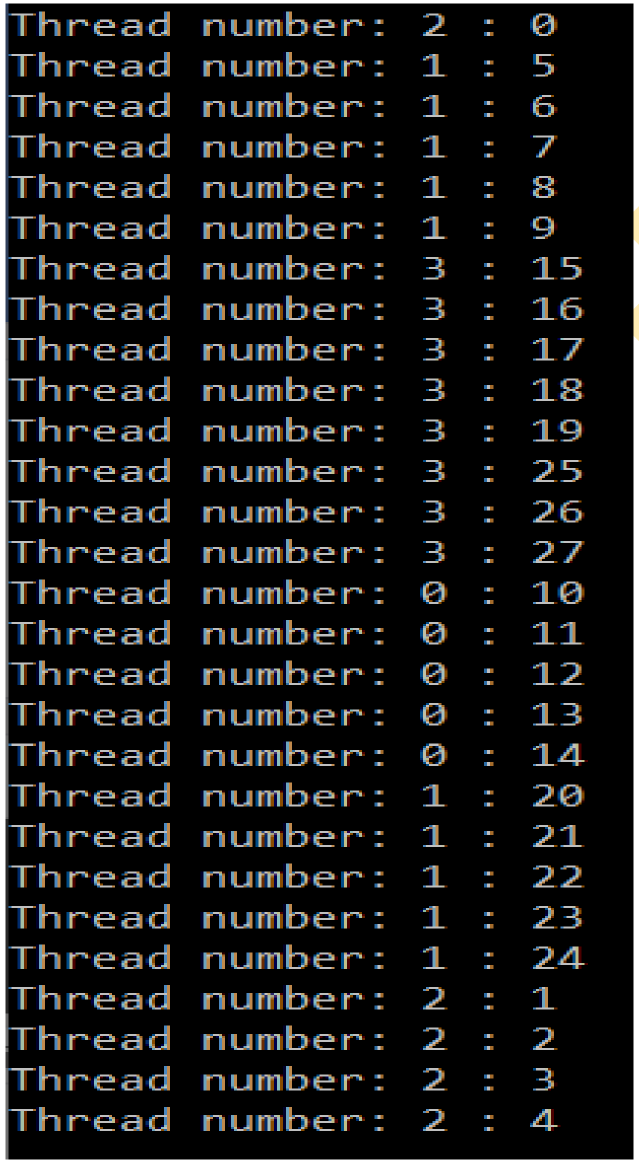
```
Thread number: 1 : 5
Thread number: 1 : 6
Thread number: 1 : 7
Thread number: 1 : 8
Thread number: 1 : 9
Thread number: 1 : 25
Thread number: 1 : 26
Thread number: 1 : 27
Thread number: 0 : 0
Thread number: 0 : 1
Thread number: 0 : 2
Thread number: 0 : 3
Thread number: 0 : 4
Thread number: 0 : 20
Thread number: 0 : 21
Thread number: 0 : 22
Thread number: 0 : 23
Thread number: 0 : 24
Thread number: 2 : 10
Thread number: 2 : 11
Thread number: 2 : 12
Thread number: 2 : 13
Thread number: 2 : 14
Thread number: 3 : 15
Thread number: 3 : 16
Thread number: 3 : 17
Thread number: 3 : 18
Thread number: 3 : 19
```

2. OpenMP Programming: *schedule(dynamic, chunk_size)*



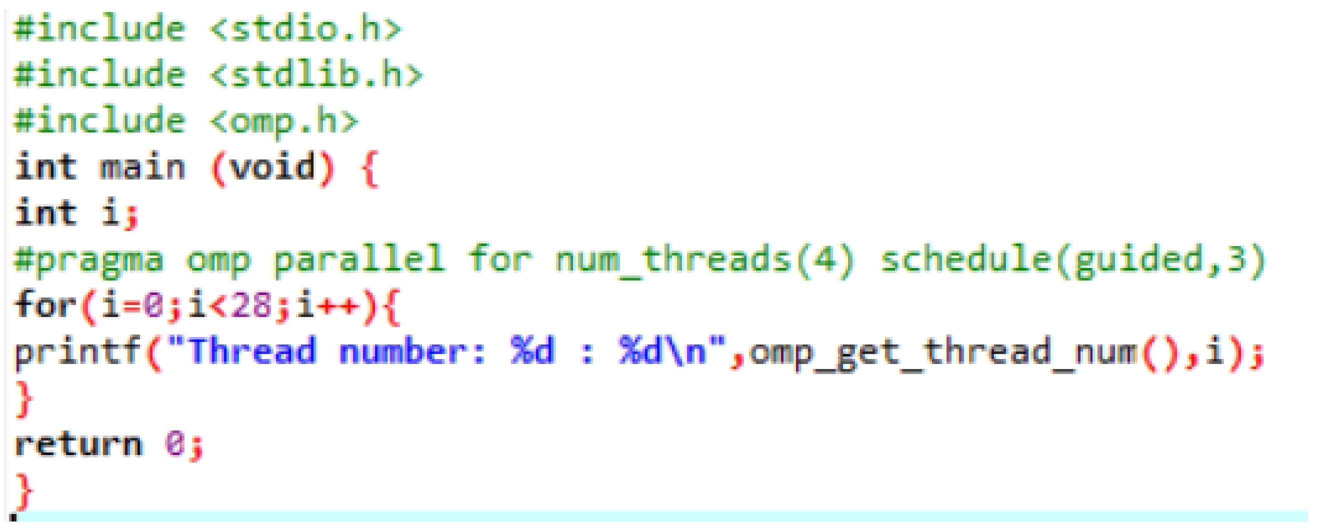
```c
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
int main (void) {
int i;
#pragma omp parallel for num_threads(4) schedule(dynamic,5)
for(i=0;i<28;i++){
printf("Thread number: %d : %d\n",omp_get_thread_num(),i);
}
return 0;
}
```

thread2	thread1	thread0	thread3	thread1	thread3
0-4	5-9	10-14	15-19	20-24	25-27

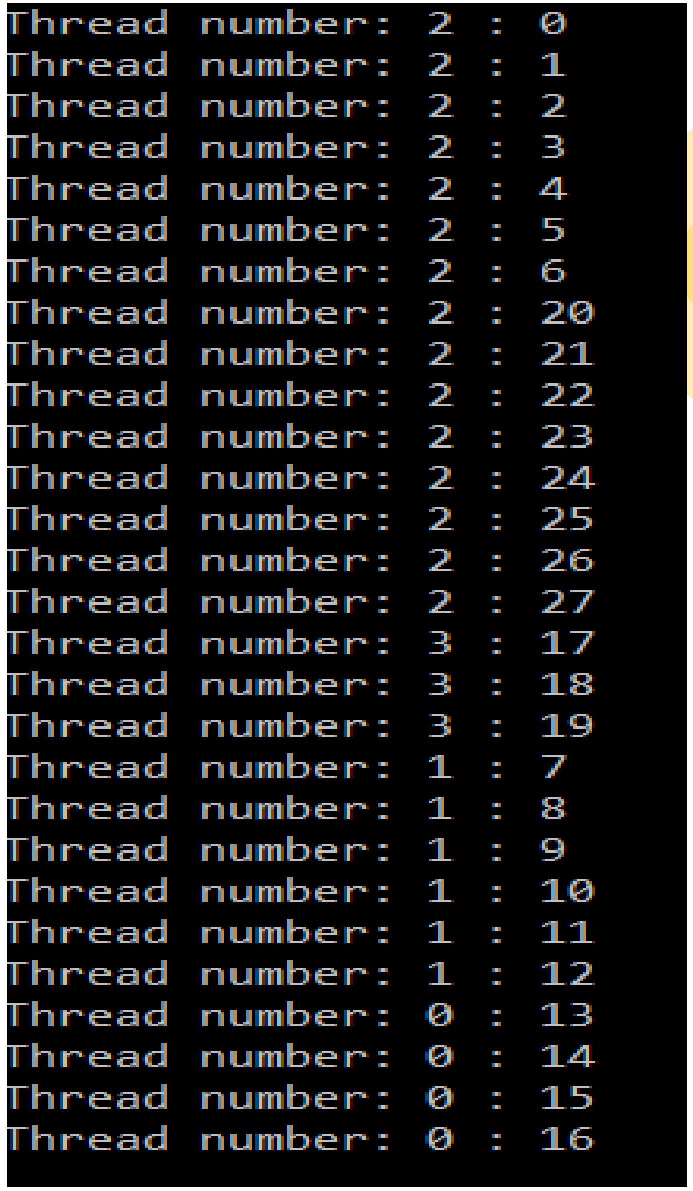


```
Thread number: 2 : 0
Thread number: 1 : 5
Thread number: 1 : 6
Thread number: 1 : 7
Thread number: 1 : 8
Thread number: 1 : 9
Thread number: 3 : 15
Thread number: 3 : 16
Thread number: 3 : 17
Thread number: 3 : 18
Thread number: 3 : 19
Thread number: 3 : 25
Thread number: 3 : 26
Thread number: 3 : 27
Thread number: 0 : 10
Thread number: 0 : 11
Thread number: 0 : 12
Thread number: 0 : 13
Thread number: 0 : 14
Thread number: 1 : 20
Thread number: 1 : 21
Thread number: 1 : 22
Thread number: 1 : 23
Thread number: 1 : 24
Thread number: 2 : 1
Thread number: 2 : 2
Thread number: 2 : 3
Thread number: 2 : 4
```

2. OpenMP Programming: *schedule(guided, chunk_size)*



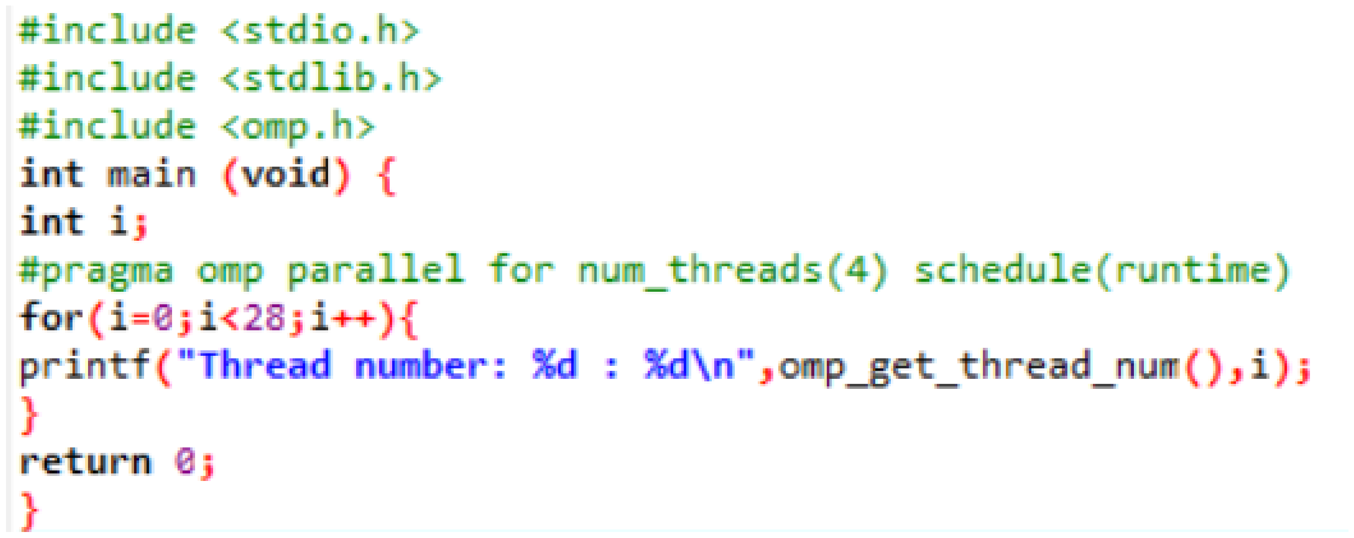
```c
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
int main (void) {
int i;
#pragma omp parallel for num_threads(4) schedule(guided,3)
for(i=0;i<28;i++){
printf("Thread number: %d : %d\n",omp_get_thread_num(),i);
}
return 0;
}
```



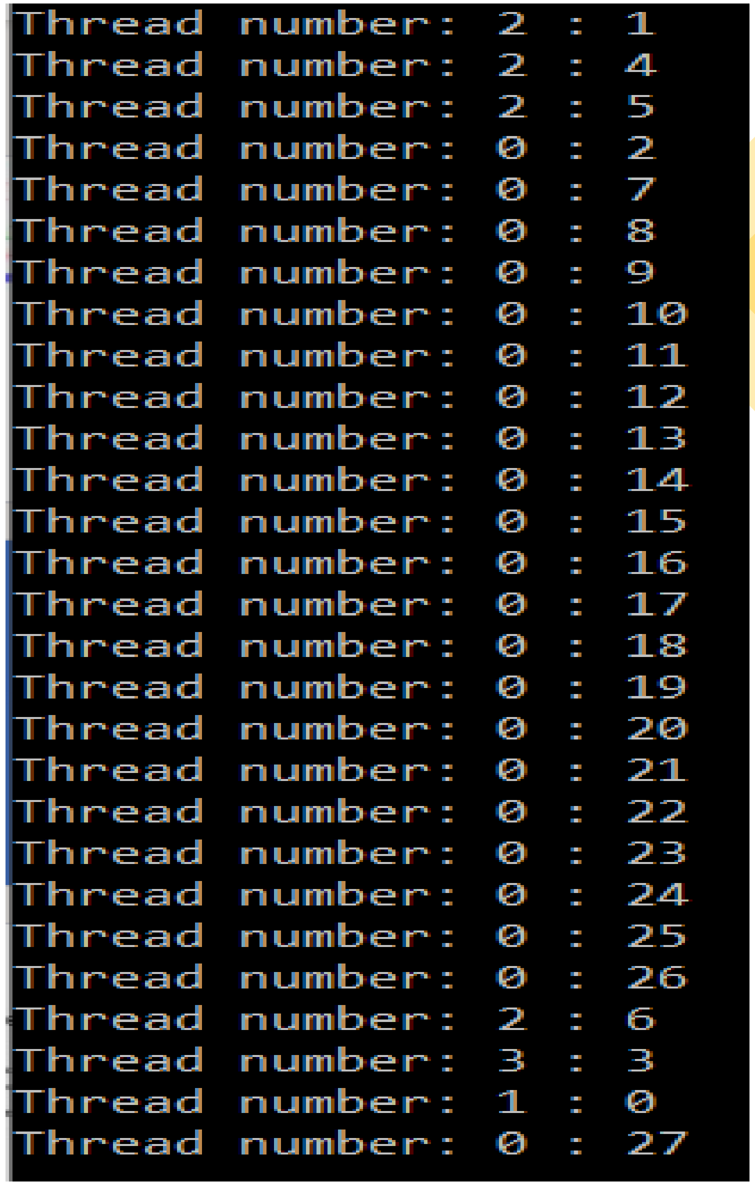
```
Thread number: 2 : 0
Thread number: 2 : 1
Thread number: 2 : 2
Thread number: 2 : 3
Thread number: 2 : 4
Thread number: 2 : 5
Thread number: 2 : 6
Thread number: 2 : 20
Thread number: 2 : 21
Thread number: 2 : 22
Thread number: 2 : 23
Thread number: 2 : 24
Thread number: 2 : 25
Thread number: 2 : 26
Thread number: 2 : 27
Thread number: 3 : 17
Thread number: 3 : 18
Thread number: 3 : 19
Thread number: 1 : 7
Thread number: 1 : 8
Thread number: 1 : 9
Thread number: 1 : 10
Thread number: 1 : 11
Thread number: 1 : 12
Thread number: 0 : 13
Thread number: 0 : 14
Thread number: 0 : 15
Thread number: 0 : 16
```

thread2	thread1	thread0	thread3	thread2	thread2	thread2
0-6 (7)	7-12(6)	13-16(4)	17-19(3)	20-22 (3)	23-25(3)	26-27(2)

2. OpenMP Programming: *schedule(runtime)*



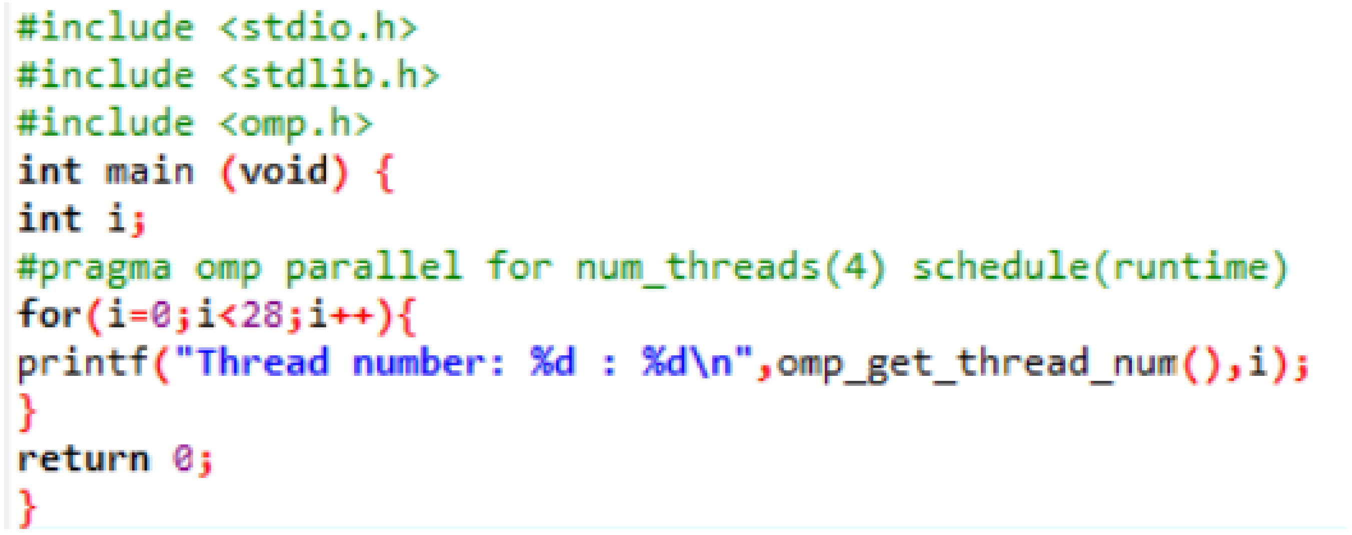
```c
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
int main (void) {
int i;
#pragma omp parallel for num_threads(4) schedule(runtime)
for(i=0;i<28;i++){
printf("Thread number: %d : %d\n",omp_get_thread_num(),i);
}
return 0;
}
```



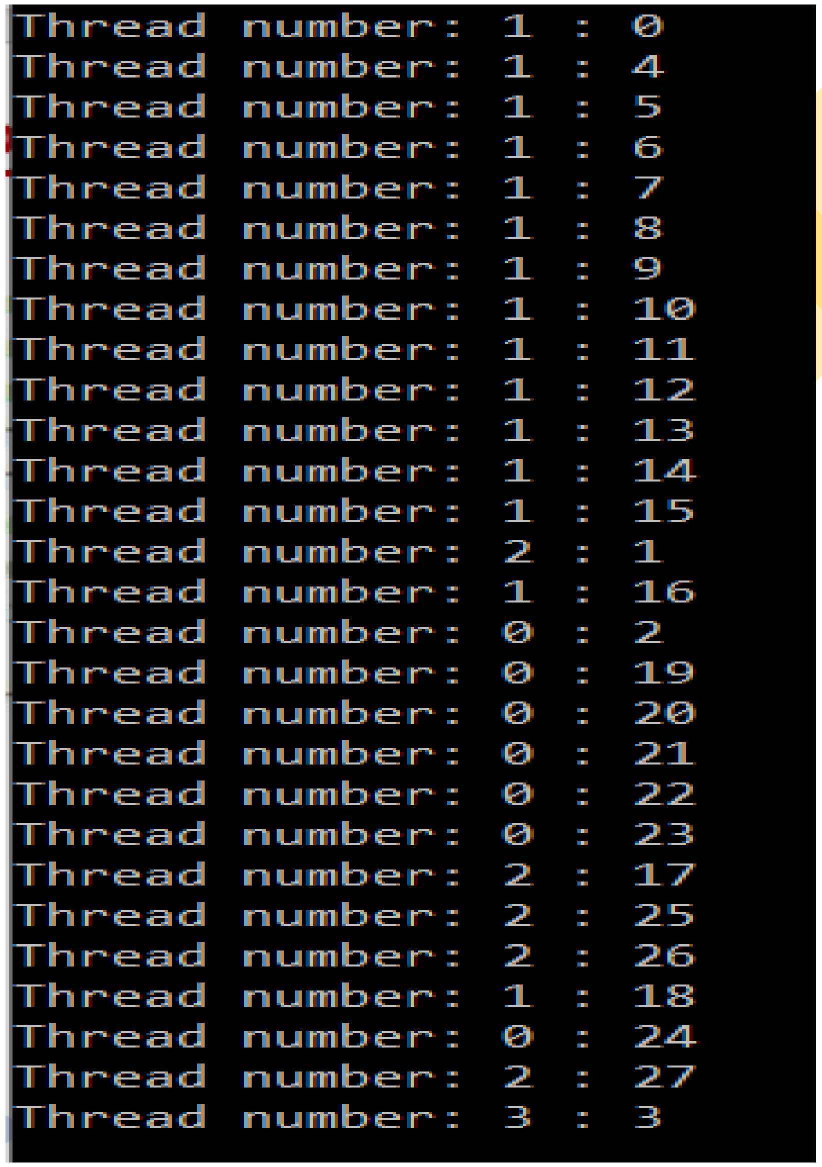
```
Thread number: 2 : 1
Thread number: 2 : 4
Thread number: 2 : 5
Thread number: 0 : 2
Thread number: 0 : 7
Thread number: 0 : 8
Thread number: 0 : 9
Thread number: 0 : 10
Thread number: 0 : 11
Thread number: 0 : 12
Thread number: 0 : 13
Thread number: 0 : 14
Thread number: 0 : 15
Thread number: 0 : 16
Thread number: 0 : 17
Thread number: 0 : 18
Thread number: 0 : 19
Thread number: 0 : 20
Thread number: 0 : 21
Thread number: 0 : 22
Thread number: 0 : 23
Thread number: 0 : 24
Thread number: 0 : 25
Thread number: 0 : 26
Thread number: 2 : 6
Thread number: 3 : 3
Thread number: 1 : 0
Thread number: 0 : 27
```

thread1	thread0	thread3	Thread2	thread0
0	2	3	4-6	7-27

2. OpenMP Programming: s*chedule(runtime)*



```c
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
int main (void) {
int i;
#pragma omp parallel for num_threads(4) schedule(runtime)
for(i=0;i<28;i++){
printf("Thread number: %d : %d\n",omp_get_thread_num(),i);
}
return 0;
}
```



```
Thread number: 1 : 0
Thread number: 1 : 4
Thread number: 1 : 5
Thread number: 1 : 6
Thread number: 1 : 7
Thread number: 1 : 8
Thread number: 1 : 9
Thread number: 1 : 10
Thread number: 1 : 11
Thread number: 1 : 12
Thread number: 1 : 13
Thread number: 1 : 14
Thread number: 1 : 15
Thread number: 2 : 1
Thread number: 1 : 16
Thread number: 0 : 2
Thread number: 0 : 19
Thread number: 0 : 20
Thread number: 0 : 21
Thread number: 0 : 22
Thread number: 0 : 23
Thread number: 2 : 17
Thread number: 2 : 25
Thread number: 2 : 26
Thread number: 1 : 18
Thread number: 0 : 24
Thread number: 2 : 27
Thread number: 3 : 3
```

Index

Reference

Text Books and/or Reference Books:

1. Professional CUDA C Programming – John Cheng, Max Grossman, Ty McKercher, 2014

2. B.Wilkinson, M.Allen, ”Parallel Programming: Techniques and Applications Using Networked Workstations and Parallel Computers”, Pearson Education, 1999

3. I.Foster, ”Designing and building parallel programs”, 2003

4. Parallel Programming in C using OpenMP and MPI – Micheal J Quinn, 2004

5. Introduction to Parallel Programming – Peter S Pacheco, Morgan Kaufmann Publishers, 2011

6. Advanced Computer Architectures: A design approach, Dezso Sima, Terence Fountain, Peter Kacsuk, 2002

7. Parallel Computer Architecture : A hardware/Software Approach, David E Culler, Jaswinder Pal Singh Anoop Gupta, 2011 8. Introduction to Parallel Computing, Ananth Grama, Anshul Gupta, George Karypis, Vipin Kumar, Pearson, 2011

Reference

Acknowledgements

1. Introduction to OpenMP https://www3.nd.edu/~zxu2/acms60212-40212/Lec-12-OpenMP.pdf
2. Introduction to parallel programming for shared memory Machines https://www.youtube.com/watch?v=LL3TAHpxOig
3. OpenMP Application Program Interface Version 2.5 May 2005
4. OpenMP Application Program Interface Version 5.0 November 2018

Thank You