# Programming with OpenMP

Megha K R

# Agenda

Introduction

Programming Model

General code structure

Compilation

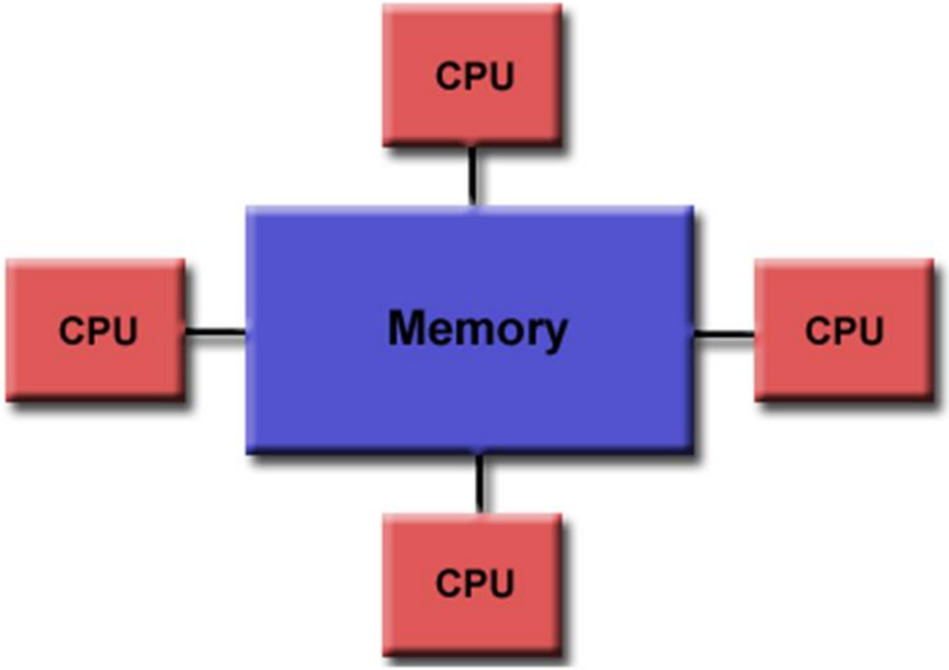
Components

- Compiler Directives
- Runtime library routines
- Environmental variables

# Introduction

- Shared Memory Model
  - Symmetric Multiprocessing
  - Single address space for all processors
    - If one processor sets x = 2 , x will also equal 2 on other processors

      (unless specified otherwise)



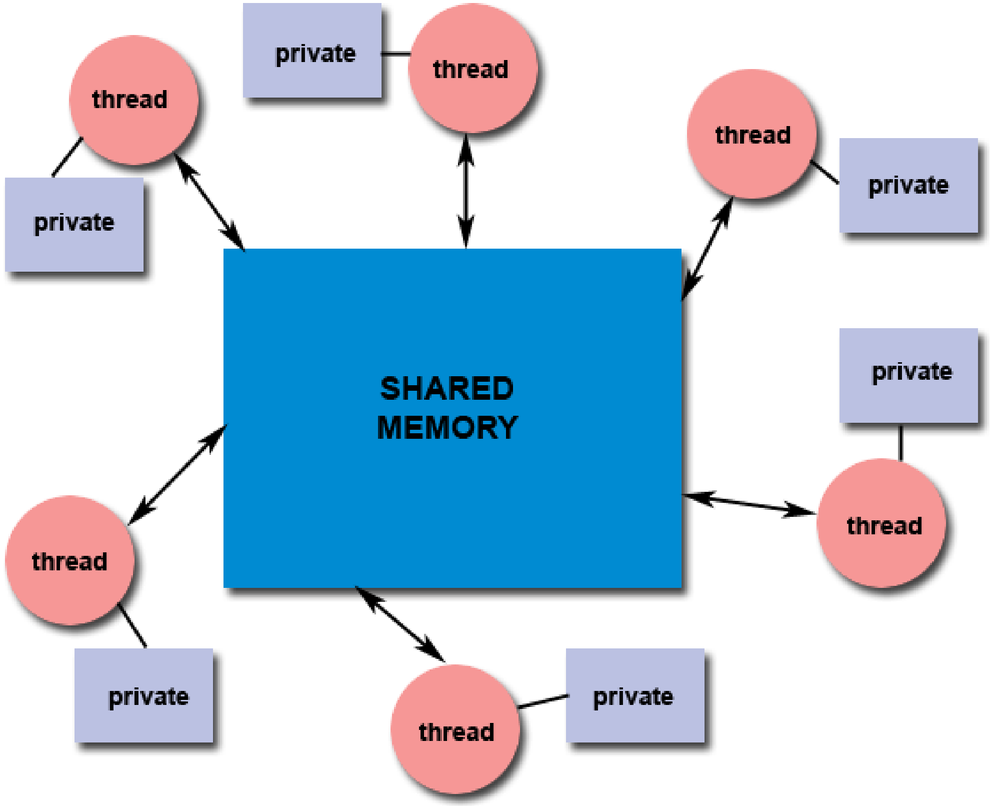
# Introduction *Contd..*

- OpenMP – Open MultiProcessing

- An Application Program Interface used for multi-threaded, shared memory parallelism

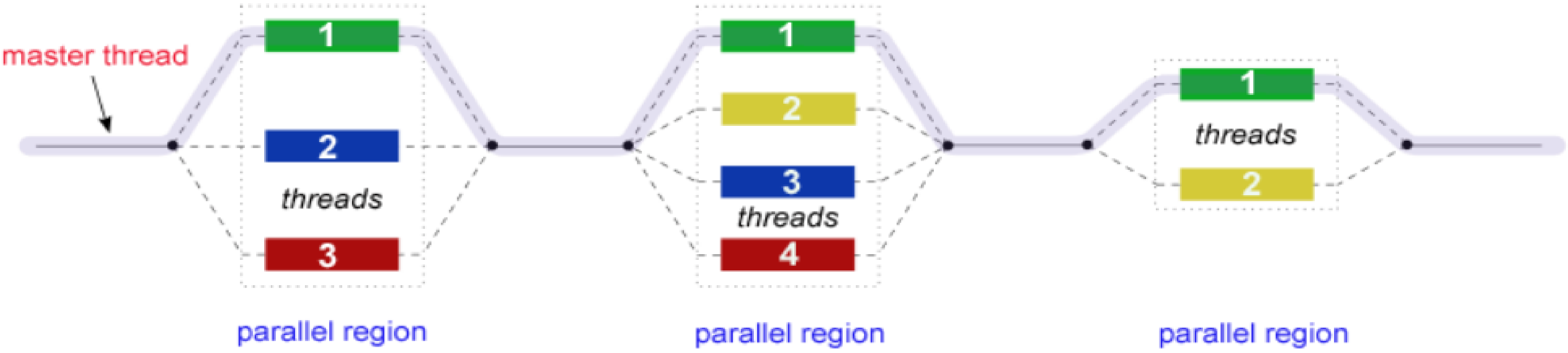
- C,C++,Fortran

- OpenMP Architecture Review Board



# Programming Model

- Fork-join model of parallel execution
- Begin as a single process, the master thread
- The master thread executes sequentially until the first parallel region construct is encountered



# Components

## Compiler Directives

- Parallel Constructs
- Work-sharing constructs
- Synchronization
- Data sharing attributes
  - Private
  - Firstprivate
  - Lastprivate
  - Reduction
  - shared

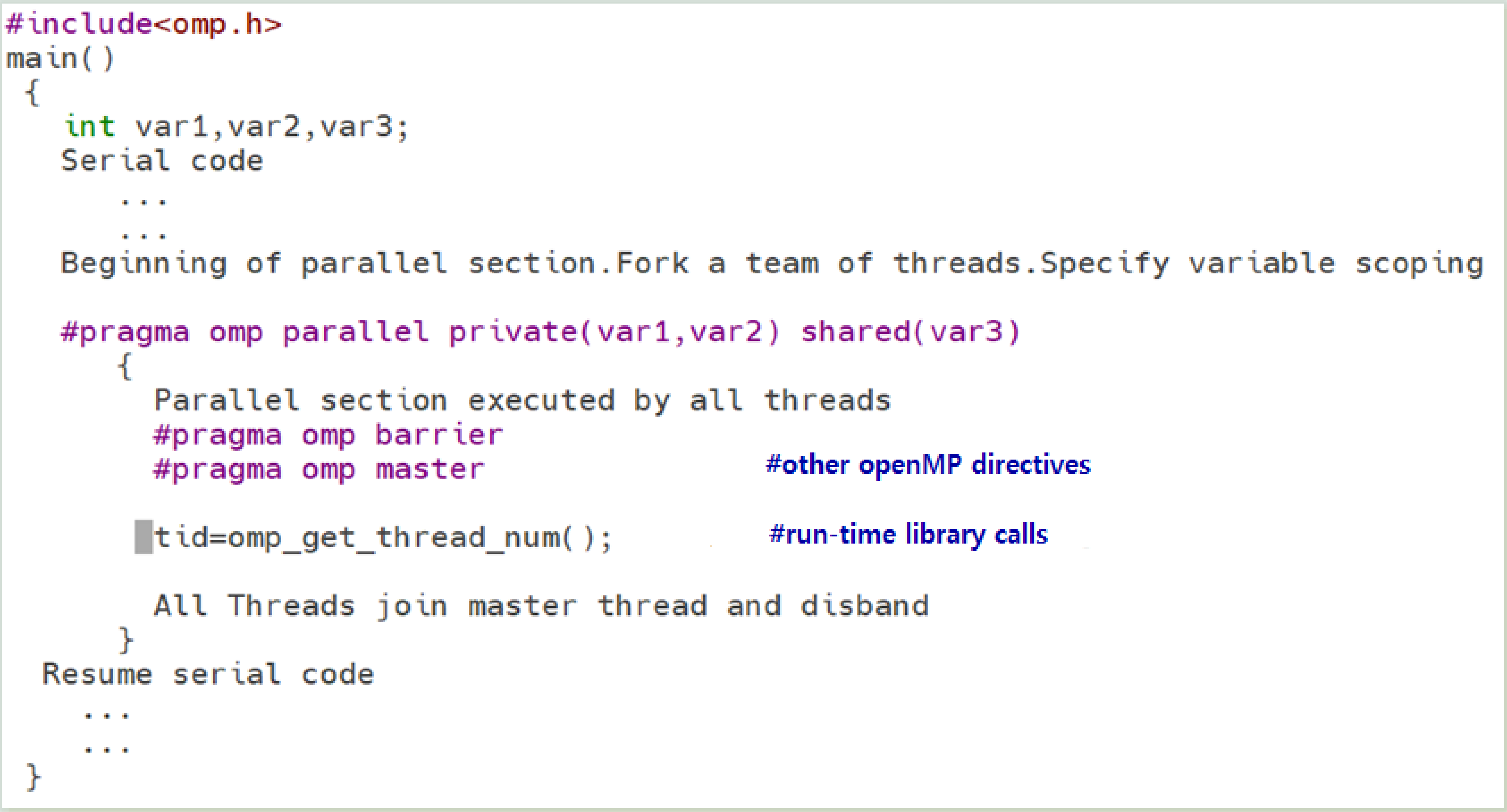
## Runtime Library Routines

- Number of threads
- Thread ID
- Dynamic thread adjustment
- Nested Parallelism
- Schedule

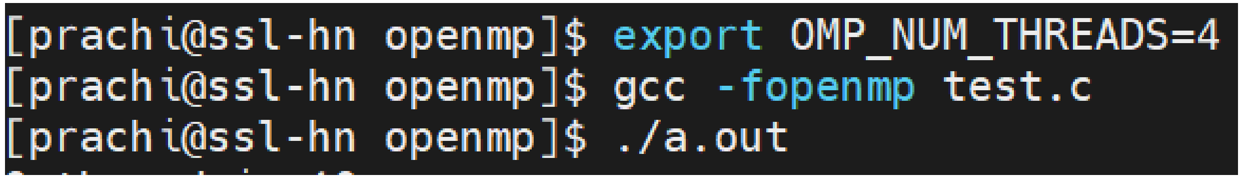
## Environment Variables

- Number of threads
- Scheduling type
- Dynamic Thread adjustment
- Nested parallelism

# General Code Structure



```
#include<omp.h>
main()
 {
   int var1,var2,var3;
   Serial code
      ...
      ...
   Beginning of parallel section.Fork a team of threads.Specify variable scoping

   #pragma omp parallel private(var1,var2) shared(var3)
      {
        Parallel section executed by all threads
        #pragma omp barrier
        #pragma omp master                    #other openMP directives

        tid=omp_get_thread_num();             #run-time library calls

        All Threads join master thread and disband
      }
  Resume serial code
    ...
    ...
 }
```



```
[prachi@ssl-hn openmp]$ export OMP_NUM_THREADS=4
[prachi@ssl-hn openmp]$ gcc -fopenmp test.c
[prachi@ssl-hn openmp]$ ./a.out
```

# Compiling OpenMP Programs

| Platform | Compiler | Flags |
|---|---|---|
| Intel<br>Linux Opteron/Xeon | icc ; icpc ; ifort | -qopenmp |
| PGI<br>Linux Opteron/Xeon | pgcc ; pgCC ; pgf77 ; pgf90 | -mp |
| GNU<br>Linux Opteron/Xeon<br>IBM Blue Gene | gcc ; g++ ; g77 ; gfortran | -fopenmp |
| IBM<br>Coral Systems | xlc_r ; cc_r xlC_r ; xlc++_r<br>xlc89_r xlc99_r | -qsmp=omp |

# Compiler Directives

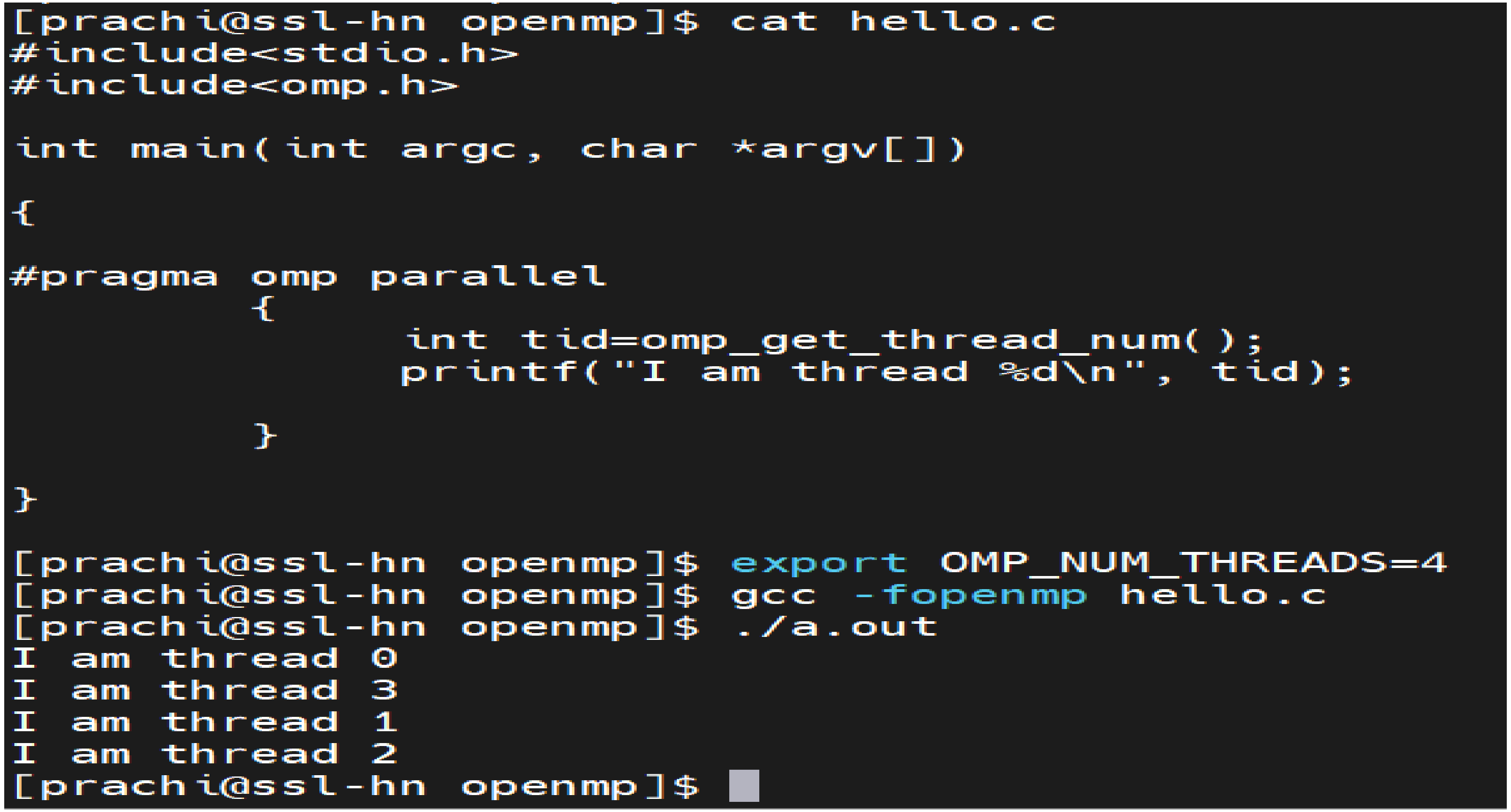
- Compiler directives appear as comments in source code
- Syntax

  *#pragma omp directive-name [clause,..]*

- They are used for:
  - Spawning a parallel region
  - Dividing blocks of code among threads
  - Distributing loop iterations between threads
  - Serializing sections of code
  - Synchronization of work among threads

# Parallel Directive

A block of codes executed by multiple threads.



```
[prachi@ssl-hn openmp]$ cat hello.c
#include<stdio.h>
#include<omp.h>

int main(int argc, char *argv[])

{

#pragma omp parallel
        {
                int tid=omp_get_thread_num();
                printf("I am thread %d\n", tid);

        }

}

[prachi@ssl-hn openmp]$ export OMP_NUM_THREADS=4
[prachi@ssl-hn openmp]$ gcc -fopenmp hello.c
[prachi@ssl-hn openmp]$ ./a.out
I am thread 0
I am thread 3
I am thread 1
I am thread 2
[prachi@ssl-hn openmp]$
```

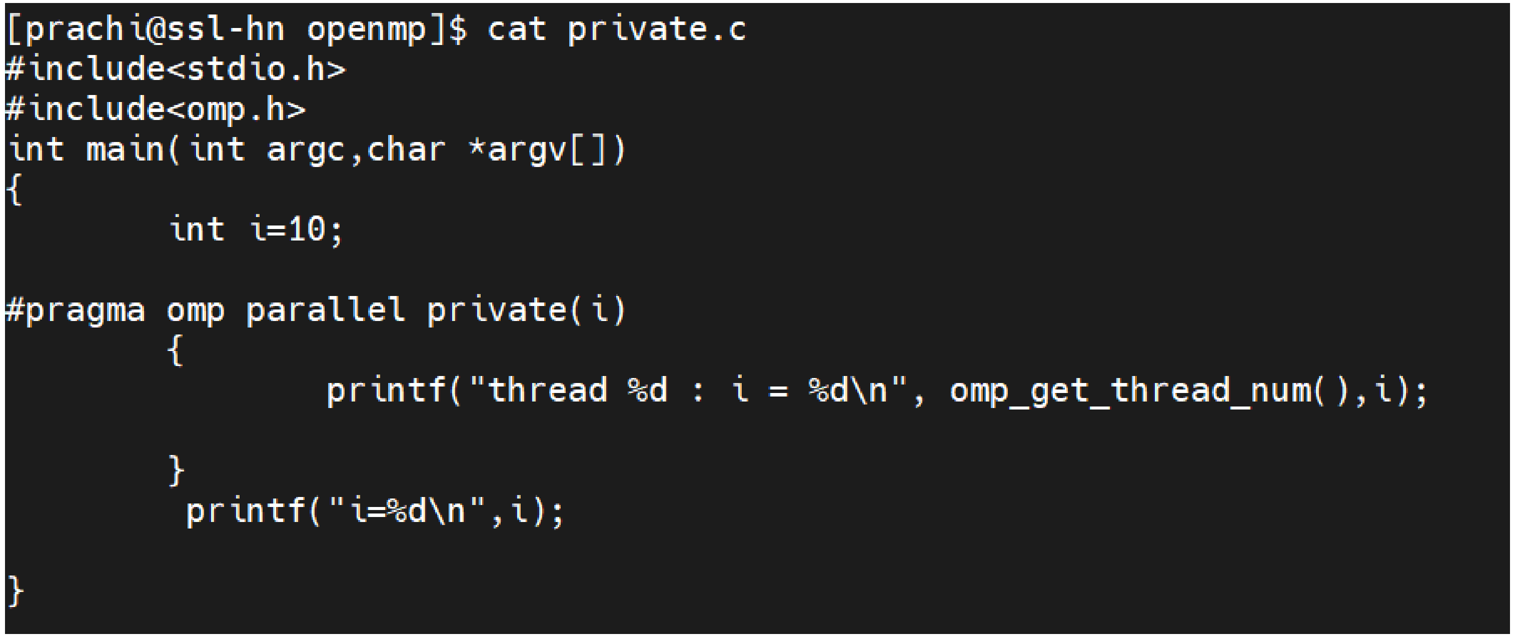
# Parallel Directive *Contd..*

```
#pragma omp parallel [clause ...]  newline
                    if (scalar_expression)
                    private (list)
                    shared (list)
                    default (shared | none)
                    firstprivate (list)
                   reduction (operator: list)
                  NUM_THREADS(scalar-integer-expression)



   structured_block
```

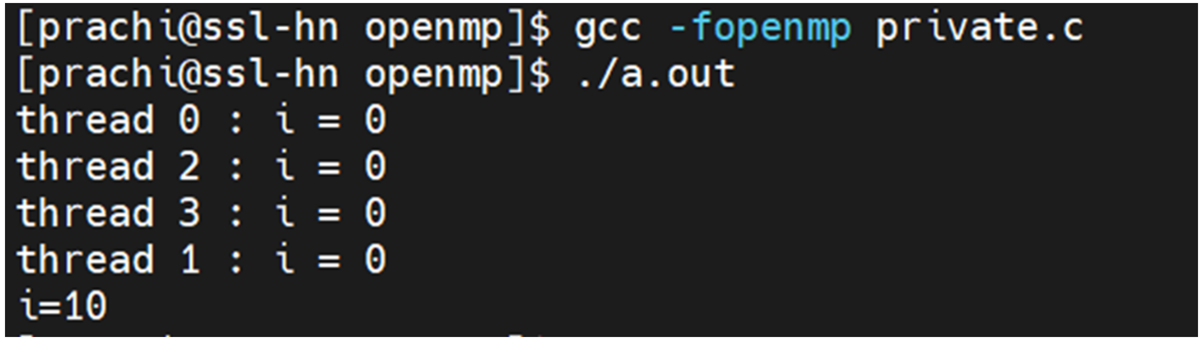
# Data Sharing Attribute Clauses

## Private

- Declares variables in its list to be private to each thread
- *private(list)*



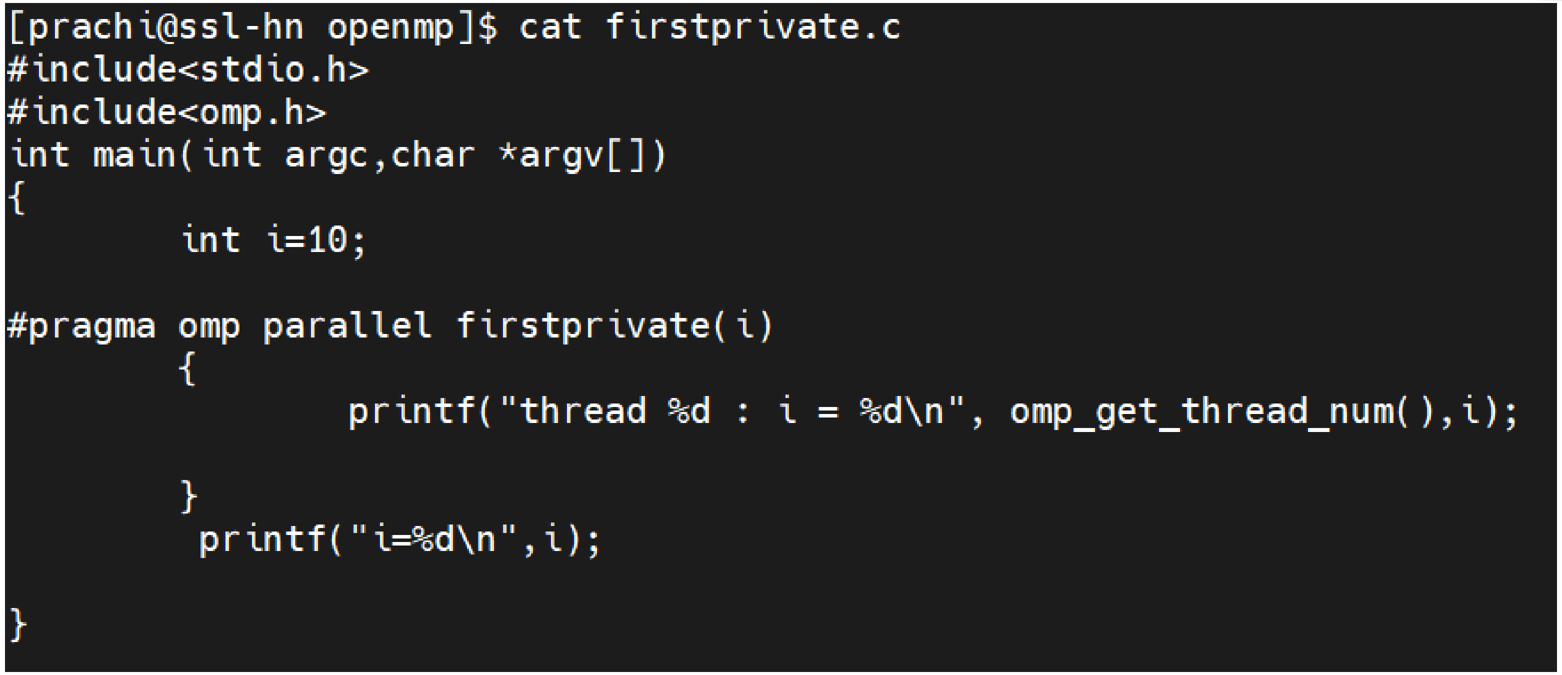
```
[prachi@ssl-hn openmp]$ cat private.c
#include<stdio.h>
#include<omp.h>
int main(int argc,char *argv[])
{
        int i=10;

#pragma omp parallel private(i)
        {
                printf("thread %d : i = %d\n", omp_get_thread_num(),i);

        }
         printf("i=%d\n",i);

}
```



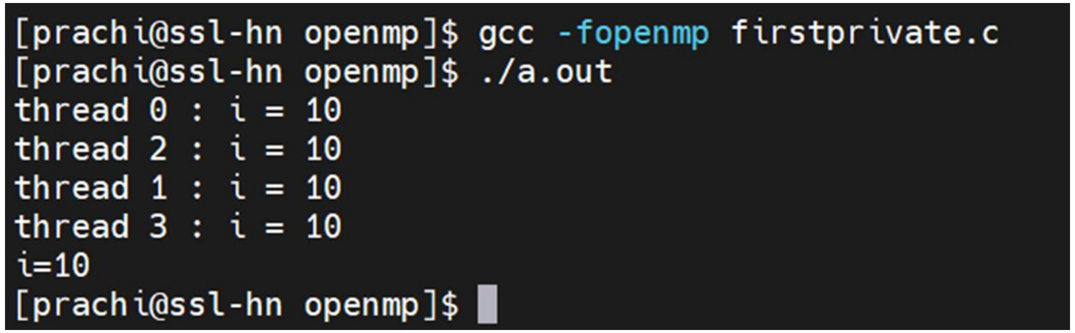
```
[prachi@ssl-hn openmp]$ gcc -fopenmp private.c
[prachi@ssl-hn openmp]$ ./a.out
thread 0 : i = 0
thread 2 : i = 0
thread 3 : i = 0
thread 1 : i = 0
i=10
```

# Firstprivate

- Private clause with automatic initialization of variables in the list
- *firstprivate (list)*



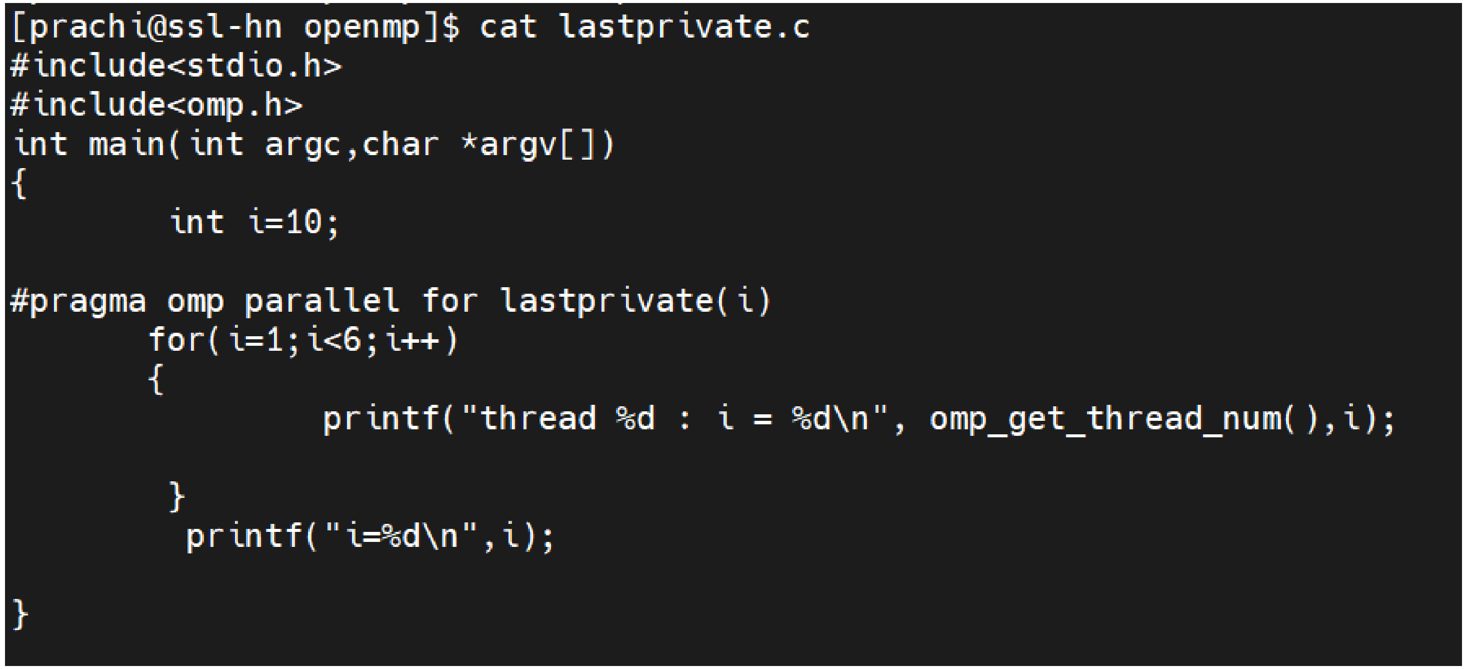
```
[prachi@ssl-hn openmp]$ cat firstprivate.c
#include<stdio.h>
#include<omp.h>
int main( int argc,char *argv[])
{
        int i=10;

#pragma omp parallel firstprivate( i)
        {
                printf("thread %d : i = %d\n", omp_get_thread_num( ),i);

        }
         printf("i=%d\n",i);

}
```



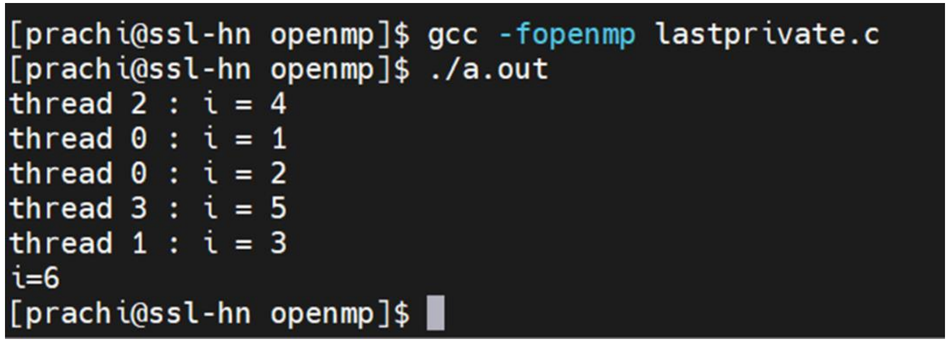
```
[prachi@ssl-hn openmp]$ gcc -fopenmp firstprivate.c
[prachi@ssl-hn openmp]$ ./a.out
thread 0 : i = 10
thread 2 : i = 10
thread 1 : i = 10
thread 3 : i = 10
i=10
[prachi@ssl-hn openmp]$
```

# Lastprivate

- Private clause with a copy from the last loop iteration
  *lastprivate (list)*

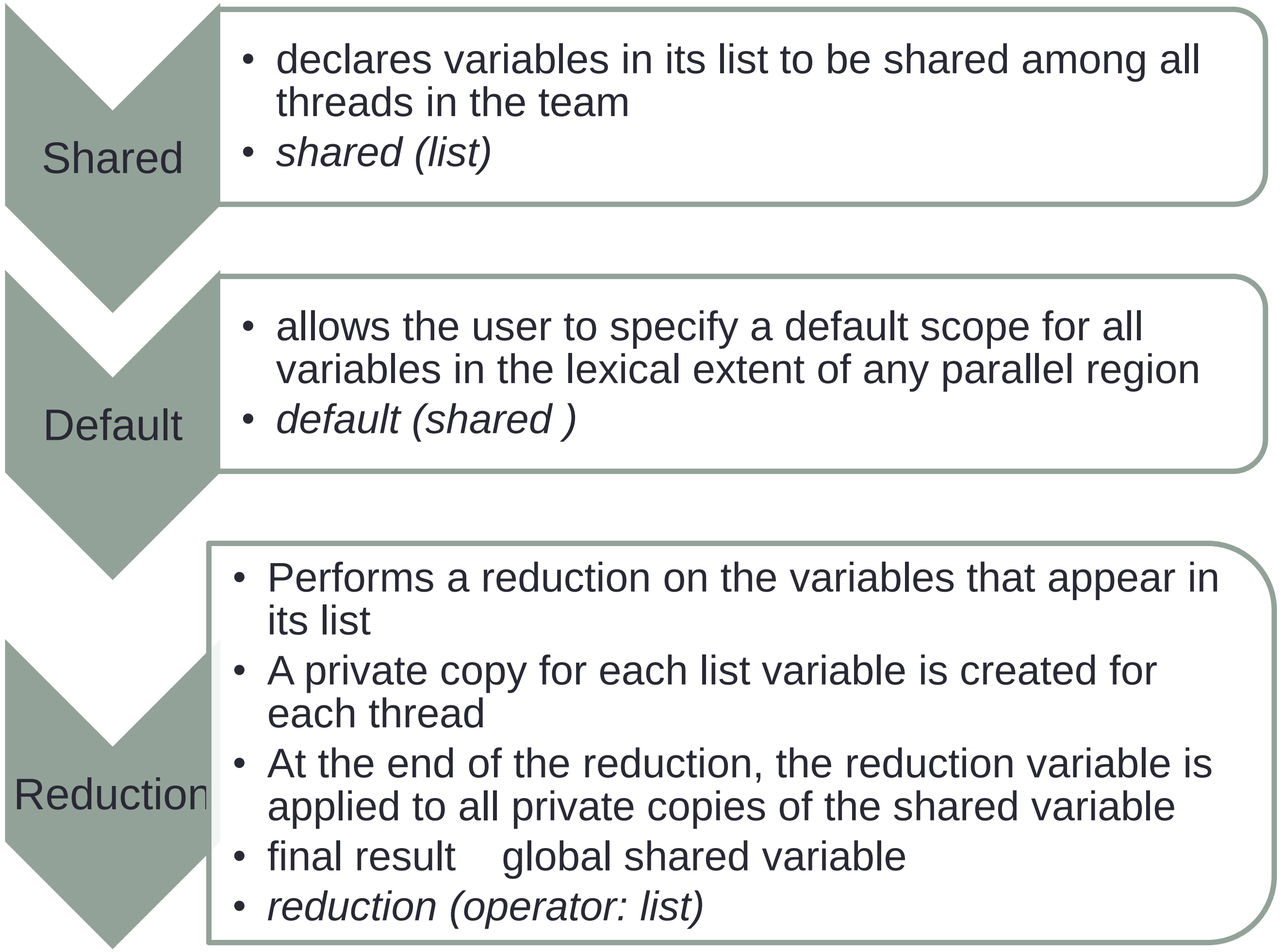


```
[prachi@ssl-hn openmp]$ cat lastprivate.c
#include<stdio.h>
#include<omp.h>
int main(int argc,char *argv[])
{
        int i=10;

#pragma omp parallel for lastprivate(i)
       for(i=1;i<6;i++)
       {
                printf("thread %d : i = %d\n", omp_get_thread_num(),i);

        }
         printf("i=%d\n",i);

}
```



```
[prachi@ssl-hn openmp]$ gcc -fopenmp lastprivate.c
[prachi@ssl-hn openmp]$ ./a.out
thread 2 : i = 4
thread 0 : i = 1
thread 0 : i = 2
thread 3 : i = 5
thread 1 : i = 3
i=6
[prachi@ssl-hn openmp]$ 
```

# Data sharing attribute clauses *Contd..*



Shared

- declares variables in its list to be shared among all threads in the team
- *shared (list)*

Default

- allows the user to specify a default scope for all variables in the lexical extent of any parallel region
- *default (shared )*

Reduction

- Performs a reduction on the variables that appear in its list
- A private copy for each list variable is created for each thread
- At the end of the reduction, the reduction variable is applied to all private copies of the shared variable
- final result    global shared variable
- *reduction (operator: list)*

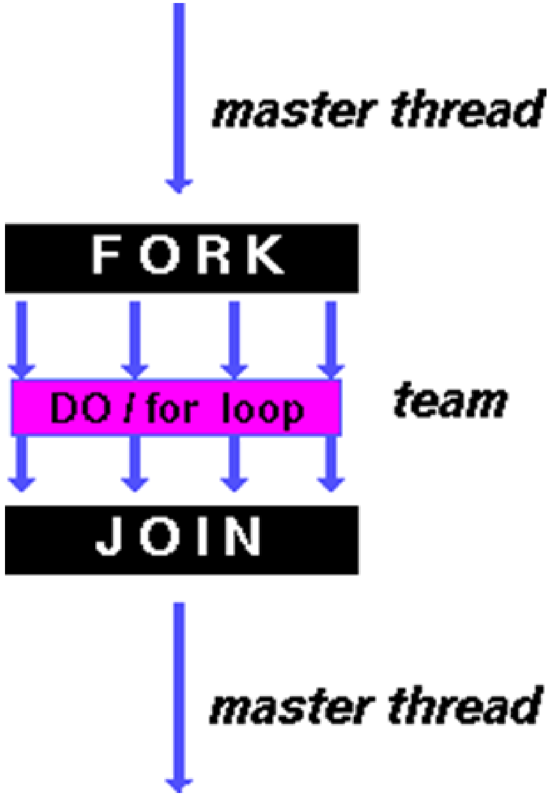
| Symbol | Meaning |
|---|---|
| + | Summation |
| - | Subtraction |
| * | Product |
| & | Bitwise AND |
| \| | Bitwise OR |
| ^ | Shift |
| && | Logical AND |
| \|\| | Logical OR |

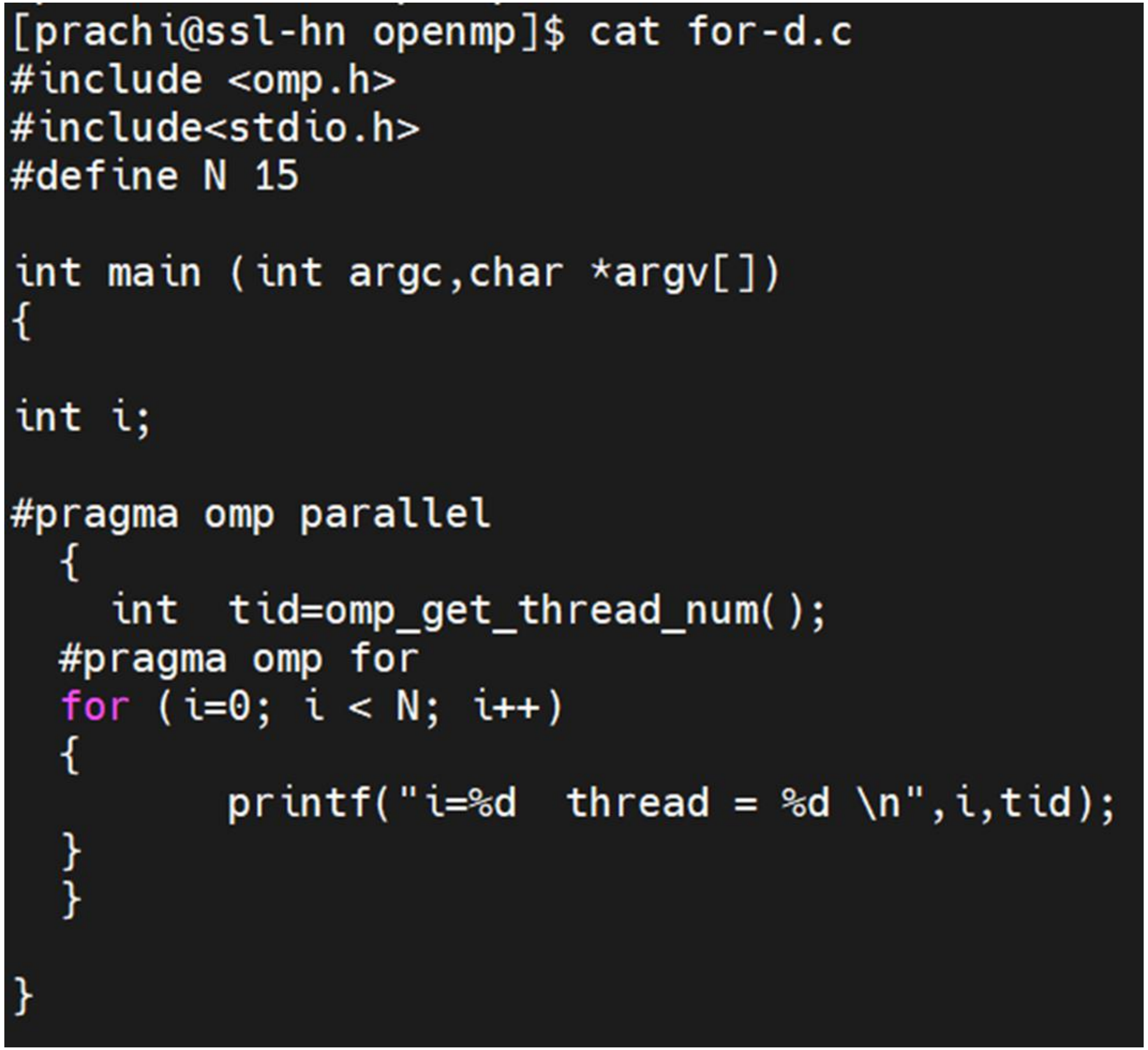
# Work Sharing Constructs

- It divides the execution of the enclosed code region among the members of the team that encounter it
- New threads are not launched
- Implied barrier at the end of a work sharing construct
- The Constructs are
  - for – data parallelism
  - section – functional parallelism
  - single – serializes a section of code

# for Construct

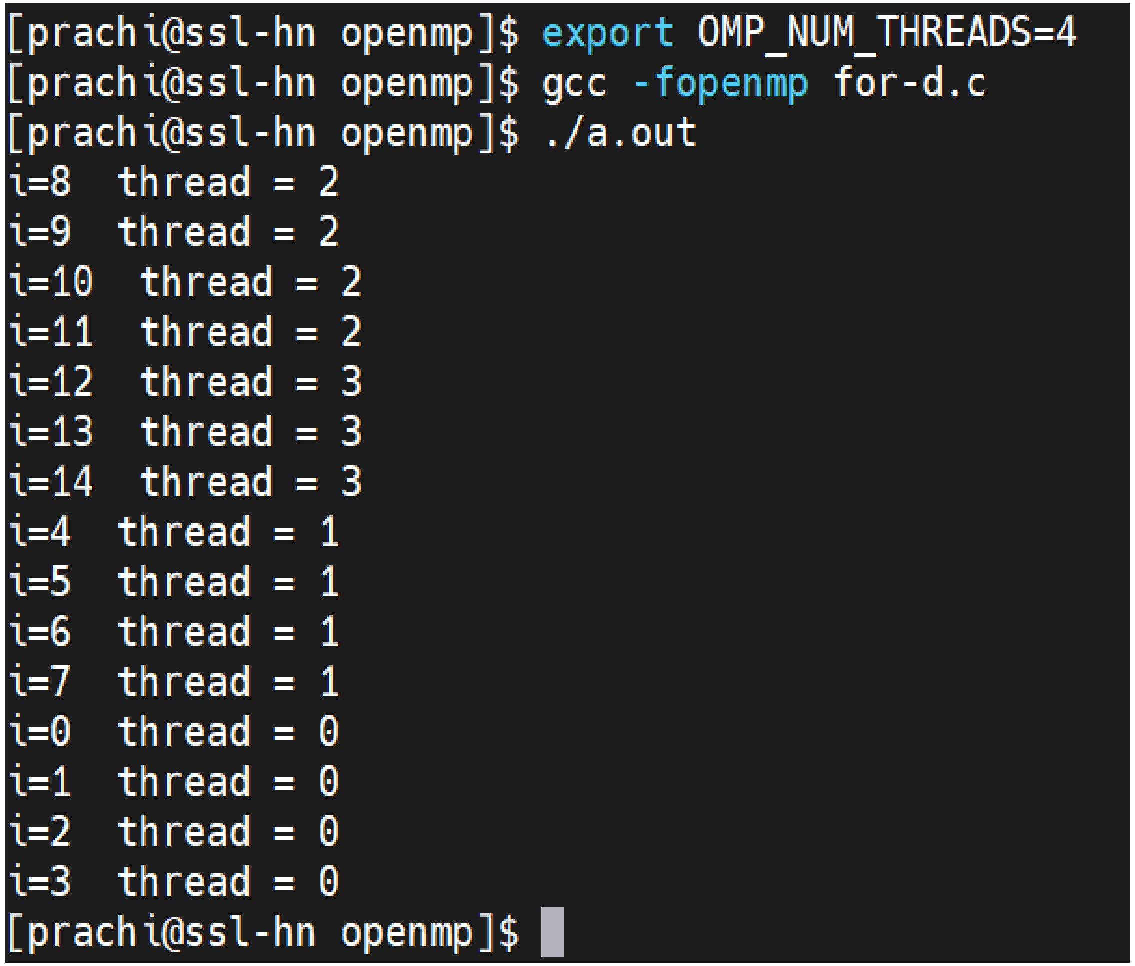
- Shares iterations of a loop across the team





```
[prachi@ssl-hn openmp]$ cat for-d.c
#include <omp.h>
#include<stdio.h>
#define N 15

int main (int argc,char *argv[])
{

int i;

#pragma omp parallel
  {
    int  tid=omp_get_thread_num();
  #pragma omp for
  for (i=0; i < N; i++)
  {
           printf("i=%d  thread = %d \n",i,tid);
  }
  }

}
```

# for Construct *Contd..*



```
[prachi@ssl-hn openmp]$ export OMP_NUM_THREADS=4
[prachi@ssl-hn openmp]$ gcc -fopenmp for-d.c
[prachi@ssl-hn openmp]$ ./a.out
i=8  thread = 2
i=9  thread = 2
i=10  thread = 2
i=11  thread = 2
i=12  thread = 3
i=13  thread = 3
i=14  thread = 3
i=4  thread = 1
i=5  thread = 1
i=6  thread = 1
i=7  thread = 1
i=0  thread = 0
i=1  thread = 0
i=2  thread = 0
i=3  thread = 0
[prachi@ssl-hn openmp]$ 
```

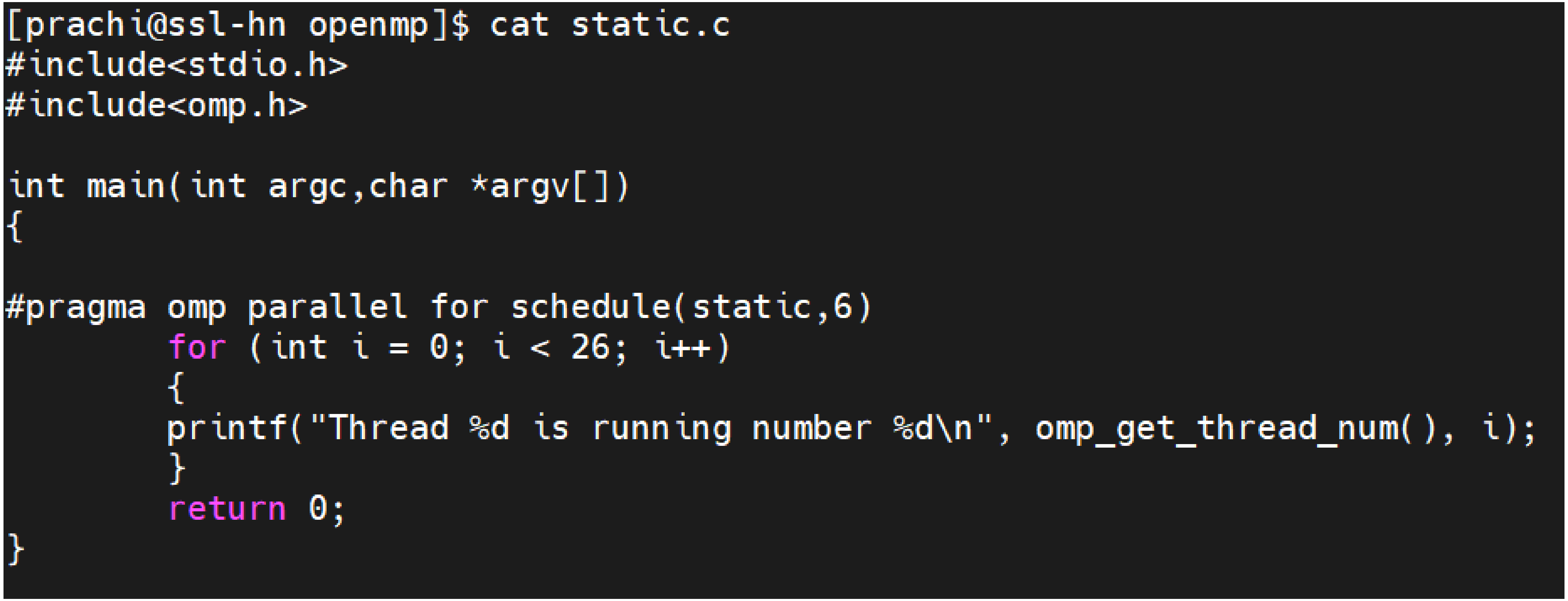
```
#pragma omp for [clause ...]  newline
            schedule (type [,chunk])
                  private (list)
                firstprivate (list)
                lastprivate (list)
                  shared (list)
            reduction (operator: list)

                   for_loop
```

# Schedule clauses

- *schedule (type [,chunk])*
- Static
  - Loop iterations are statically assigned to threads
  - If chunk is not specified, the iterations are evenly (if possible) divided contiguously among the threads

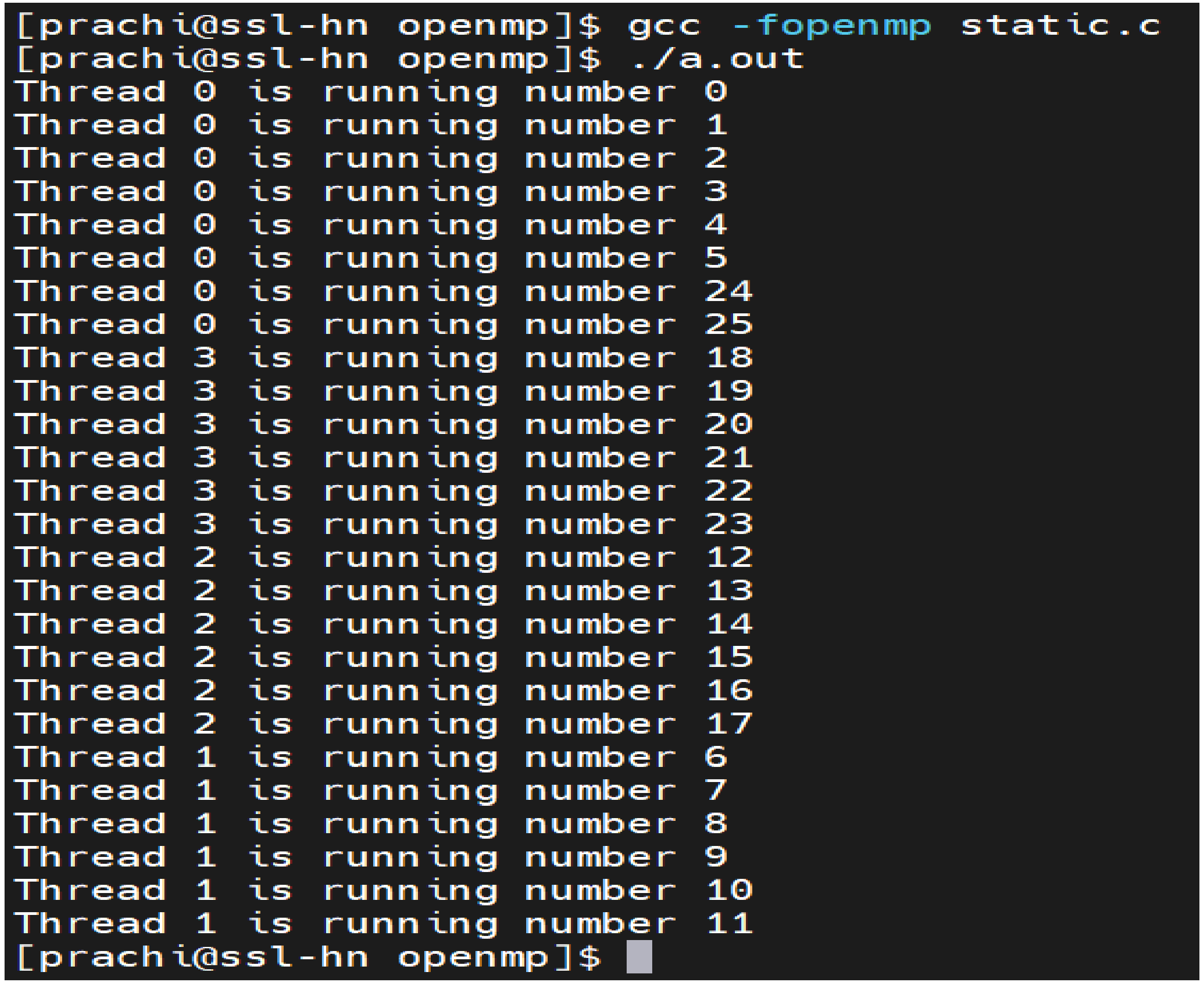
| Thread no | Chunk 1 Indices | Chunk 2 Indices | No of iterations assigned |
|---|---|---|---|
| 0 | 0-5 | 24-25 | 8 |
| 1 | 6-11 | - | 6 |
| 2 | 12-17 | - | 6 |
| 3 | 18-23 | - | 6 |

# Program (static)



```
[prachi@ssl-hn openmp]$ cat static.c
#include<stdio.h>
#include<omp.h>

int main( int argc,char *argv[])
{

#pragma omp parallel for schedule(static,6)
        for (int i = 0; i < 26; i++)
        {
        printf("Thread %d is running number %d\n", omp_get_thread_num(), i);
        }
        return 0;
}
```

# Output(static)



```
[prachi@ssl-hn openmp]$ gcc -fopenmp static.c
[prachi@ssl-hn openmp]$ ./a.out
Thread 0 is running number 0
Thread 0 is running number 1
Thread 0 is running number 2
Thread 0 is running number 3
Thread 0 is running number 4
Thread 0 is running number 5
Thread 0 is running number 24
Thread 0 is running number 25
Thread 3 is running number 18
Thread 3 is running number 19
Thread 3 is running number 20
Thread 3 is running number 21
Thread 3 is running number 22
Thread 3 is running number 23
Thread 2 is running number 12
Thread 2 is running number 13
Thread 2 is running number 14
Thread 2 is running number 15
Thread 2 is running number 16
Thread 2 is running number 17
Thread 1 is running number 6
Thread 1 is running number 7
Thread 1 is running number 8
Thread 1 is running number 9
Thread 1 is running number 10
Thread 1 is running number 11
[prachi@ssl-hn openmp]$
```

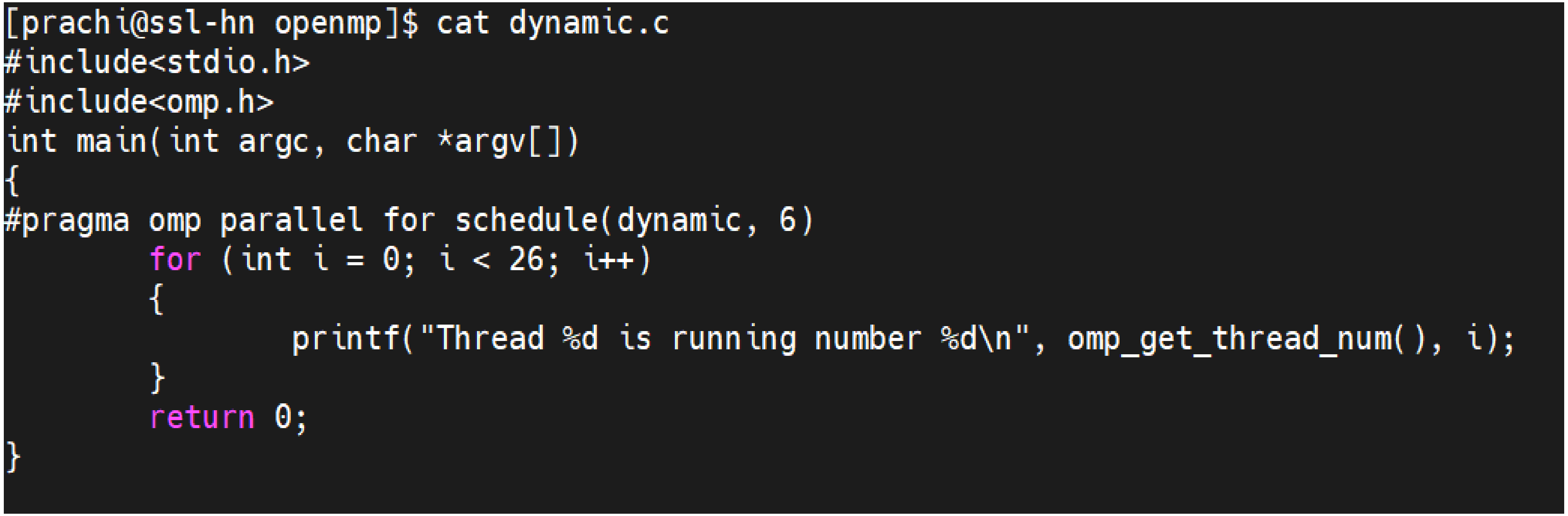
# Schedule Clauses *Contd..*

Dynamic

- Loop iterations are dynamically scheduled among the threads
- when a thread finishes one chunk , it is dynamically assigned another

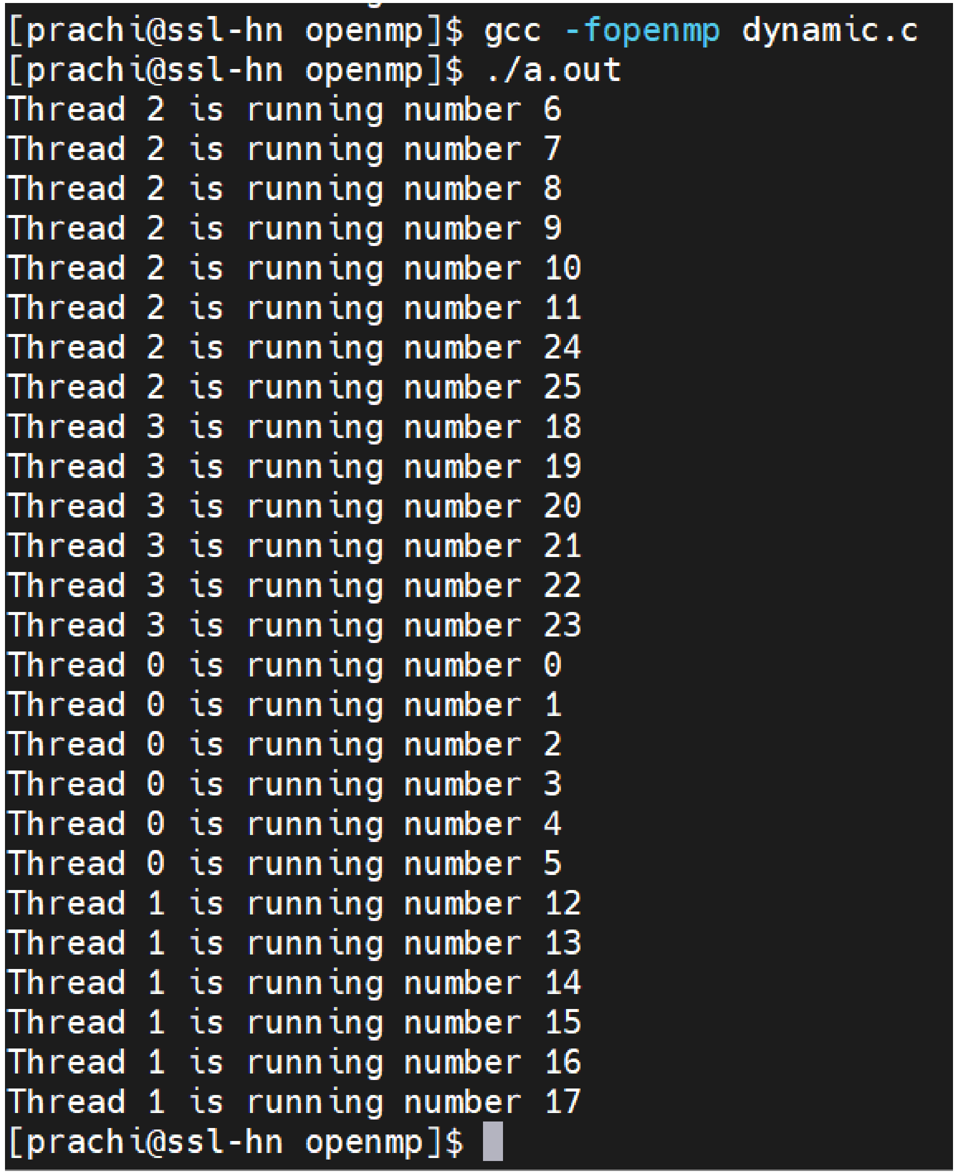
| Thread No | Chunk 1 indices | Chunk 2 indices | No of iterations assigned |
|---|---|---|---|
| 0 | 0-5 | - | 6 |
| 1 | 12-17 | - | 6 |
| 2 | 6-11 | 24-25 | 8 |
| 3 | 18-23 | - | 6 |

# Program (dynamic)



```
[prachi@ssl-hn openmp]$ cat dynamic.c
#include<stdio.h>
#include<omp.h>
int main( int argc, char *argv[])
{
#pragma omp parallel for schedule(dynamic, 6)
        for (int i = 0; i < 26; i++)
        {
                printf("Thread %d is running number %d\n", omp_get_thread_num(), i);
        }
        return 0;
}
```

# Output (dynamic)



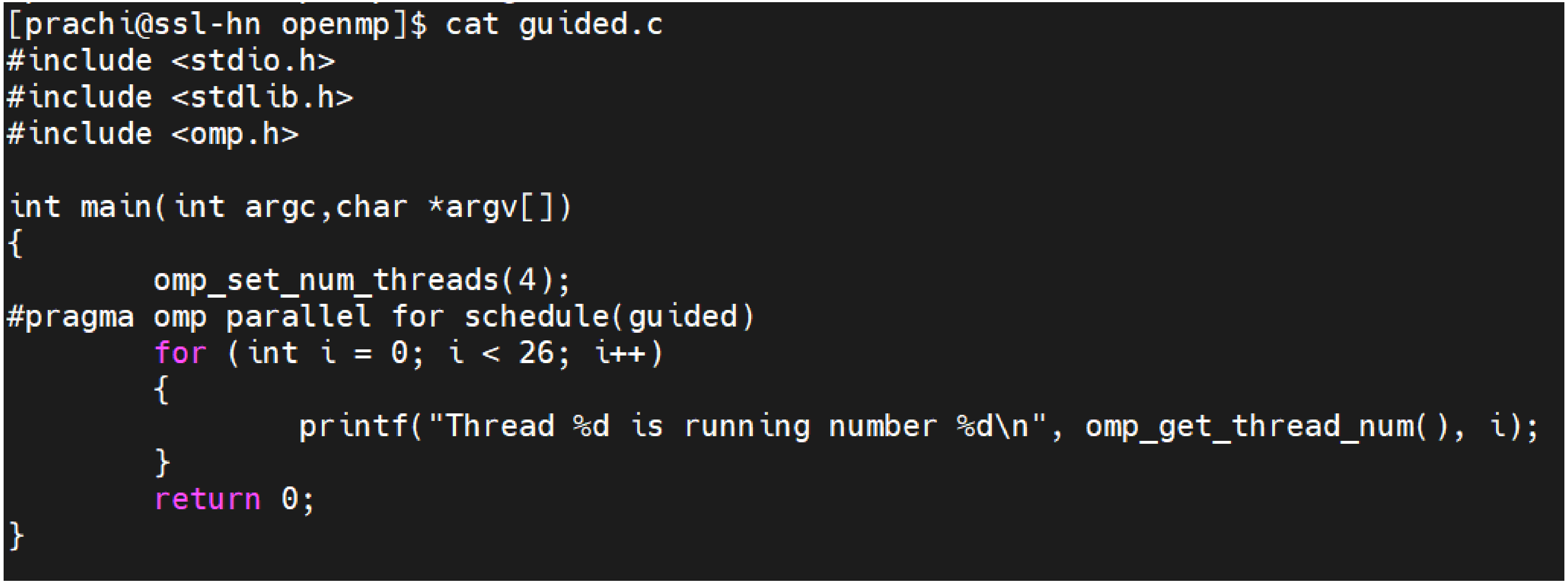
```
[prachi@ssl-hn openmp]$ gcc -fopenmp dynamic.c
[prachi@ssl-hn openmp]$ ./a.out
Thread 2 is running number 6
Thread 2 is running number 7
Thread 2 is running number 8
Thread 2 is running number 9
Thread 2 is running number 10
Thread 2 is running number 11
Thread 2 is running number 24
Thread 2 is running number 25
Thread 3 is running number 18
Thread 3 is running number 19
Thread 3 is running number 20
Thread 3 is running number 21
Thread 3 is running number 22
Thread 3 is running number 23
Thread 0 is running number 0
Thread 0 is running number 1
Thread 0 is running number 2
Thread 0 is running number 3
Thread 0 is running number 4
Thread 0 is running number 5
Thread 1 is running number 12
Thread 1 is running number 13
Thread 1 is running number 14
Thread 1 is running number 15
Thread 1 is running number 16
Thread 1 is running number 17
[prachi@ssl-hn openmp]$
```

# Schedule clauses *Contd…*

- guided
  - Iterations are dynamically assigned to threads in blocks as threads request them until no blocks remain to be assigned
  - The block size decreases each time a parcel of work is given to a thread

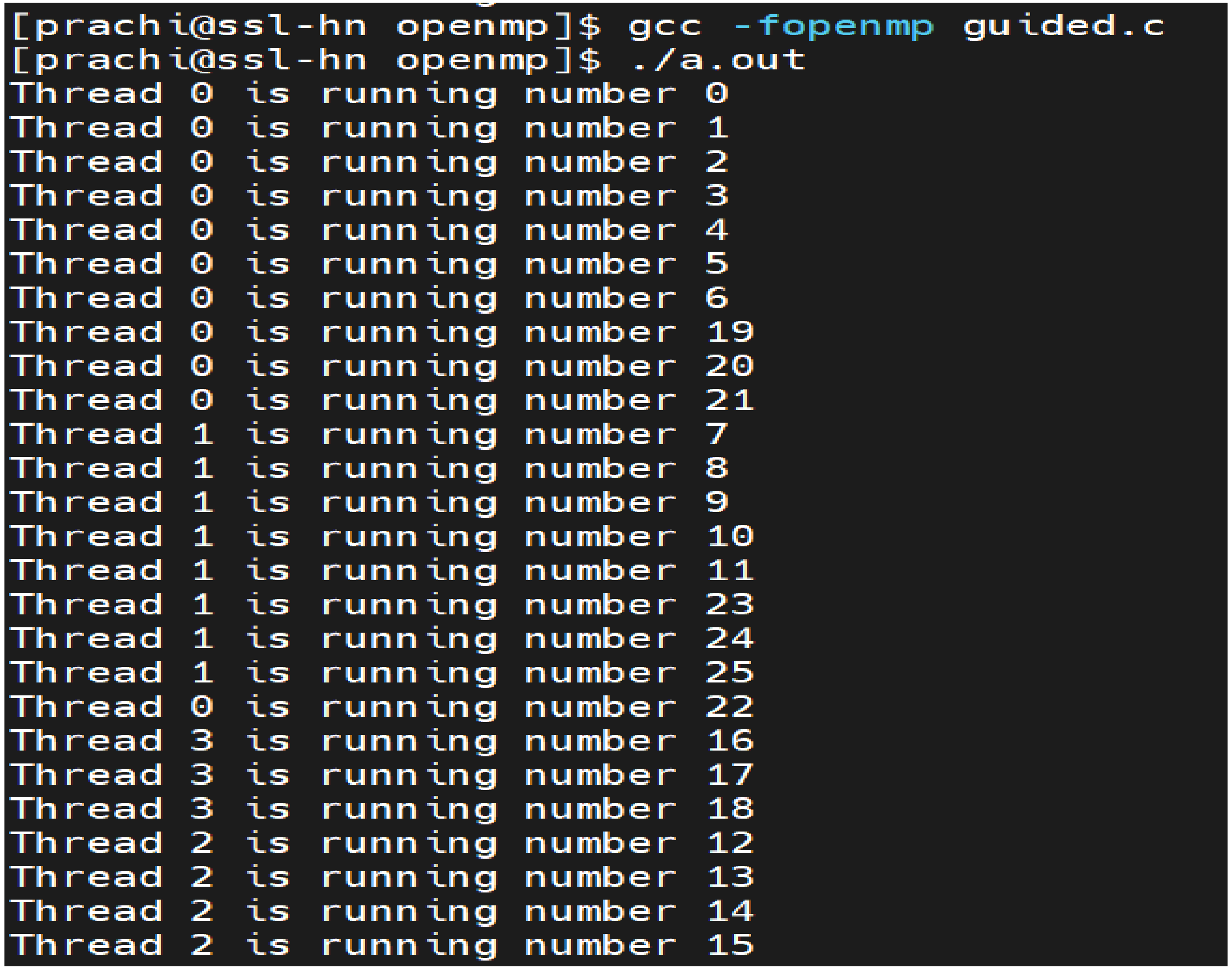
| Thread No | Chunk 1 indices | Chunk 2 indices | No of iterations assigned |
|---|---|---|---|
| 0 | 0-6 | 19-20,21-22 | 11 |
| 1 | 7-11 | 23-25 | 8 |
| 2 | 12-15 | - | 4 |
| 3 | 16-18 | - | 3 |

# Program (guided)



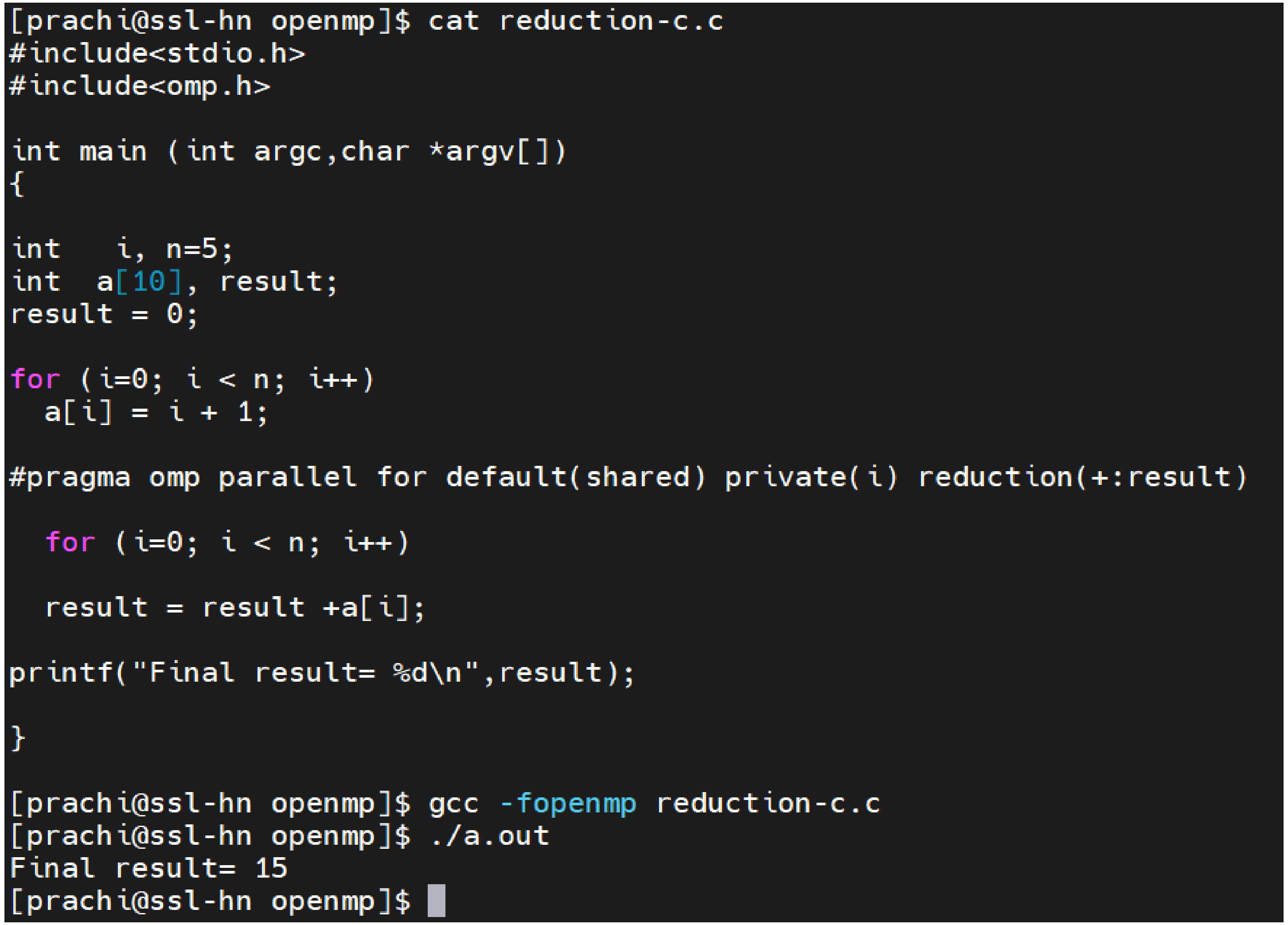
```
[prachi@ssl-hn openmp]$ cat guided.c
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>

int main( int argc,char *argv[])
{
        omp_set_num_threads(4);
#pragma omp parallel for schedule(guided)
        for (int i = 0; i < 26; i++)
        {
                printf("Thread %d is running number %d\n", omp_get_thread_num(), i);
        }
        return 0;
}
```

# Output



```
[prachi@ssl-hn openmp]$ gcc -fopenmp guided.c
[prachi@ssl-hn openmp]$ ./a.out
Thread 0 is running number 0
Thread 0 is running number 1
Thread 0 is running number 2
Thread 0 is running number 3
Thread 0 is running number 4
Thread 0 is running number 5
Thread 0 is running number 6
Thread 0 is running number 19
Thread 0 is running number 20
Thread 0 is running number 21
Thread 1 is running number 7
Thread 1 is running number 8
Thread 1 is running number 9
Thread 1 is running number 10
Thread 1 is running number 11
Thread 1 is running number 23
Thread 1 is running number 24
Thread 1 is running number 25
Thread 0 is running number 22
Thread 3 is running number 16
Thread 3 is running number 17
Thread 3 is running number 18
Thread 2 is running number 12
Thread 2 is running number 13
Thread 2 is running number 14
Thread 2 is running number 15
```

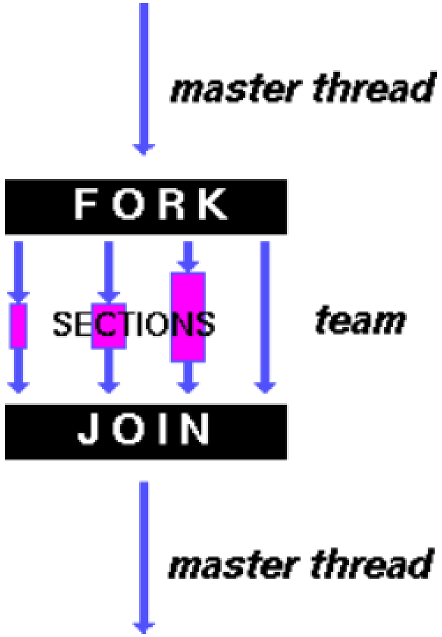
# Program and Output

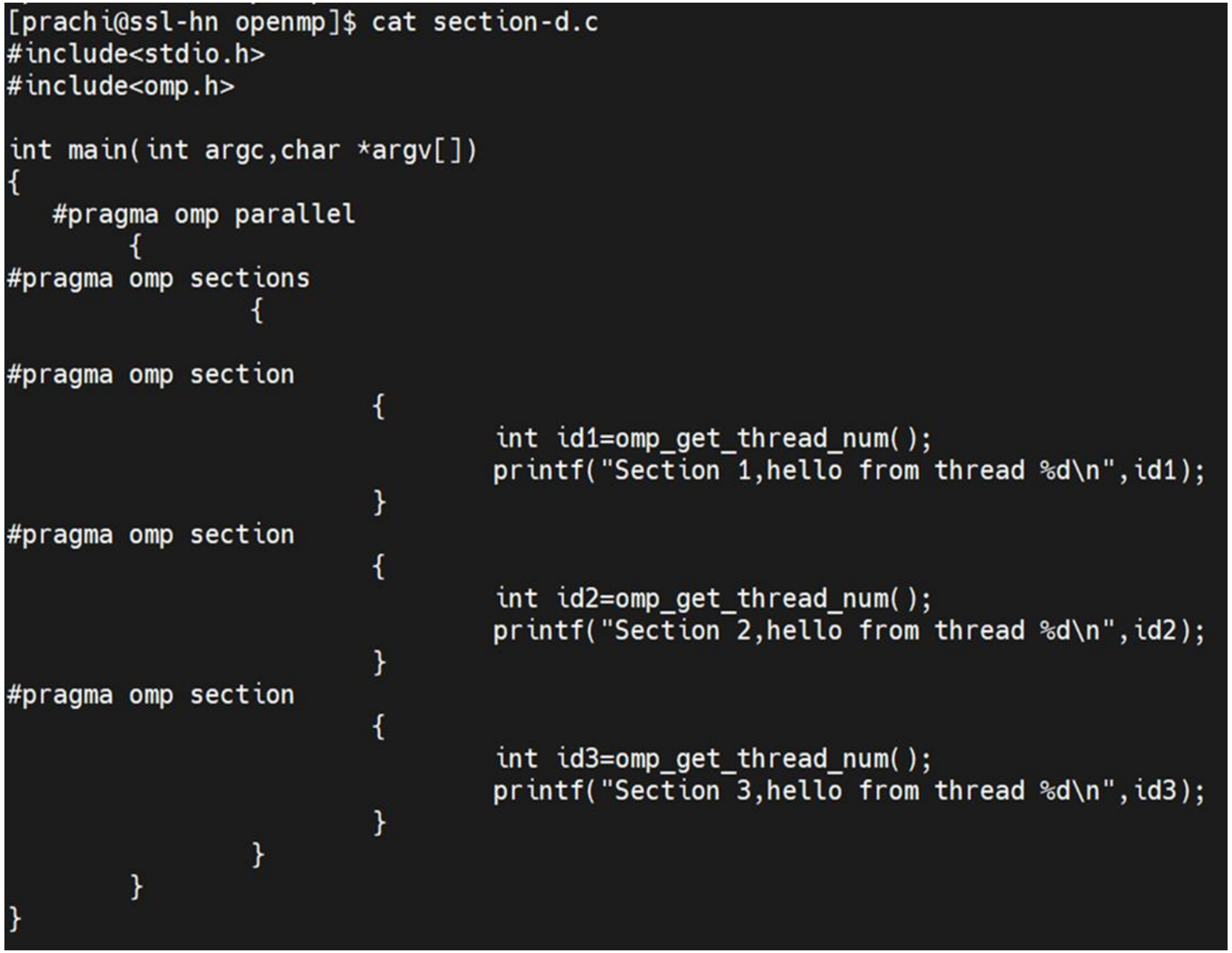


```
[prachi@ssl-hn openmp]$ cat reduction-c.c
#include<stdio.h>
#include<omp.h>

int main (int argc,char *argv[])
{

int   i, n=5;
int  a[10], result;
result = 0;

for (i=0; i < n; i++)
  a[i] = i + 1;

#pragma omp parallel for default(shared) private(i) reduction(+:result)

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

  result = result +a[i];

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

}

[prachi@ssl-hn openmp]$ gcc -fopenmp reduction-c.c
[prachi@ssl-hn openmp]$ ./a.out
Final result= 15
[prachi@ssl-hn openmp]$
```

# section Directive

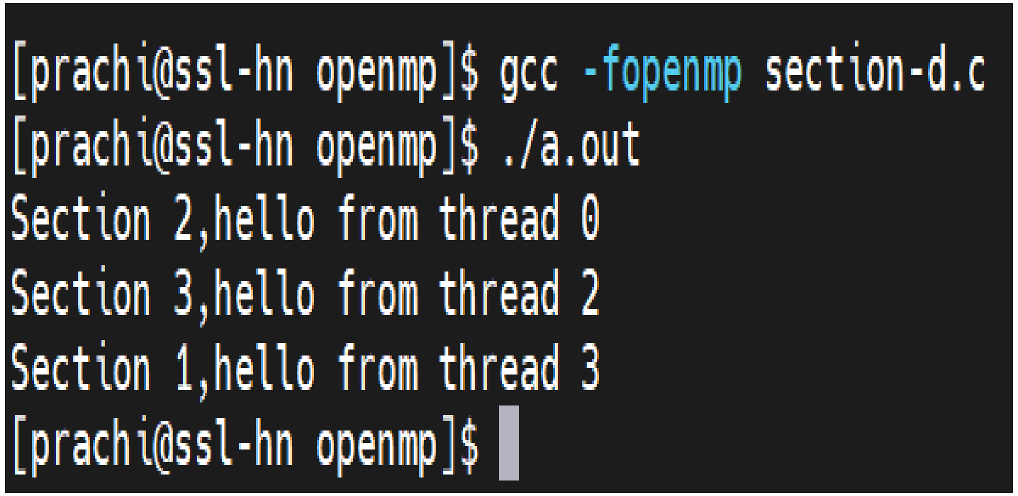
- It breaks work into separate, discrete sections
- Each section is executed by a thread





```
[prachi@ssl-hn openmp]$ cat section-d.c
#include<stdio.h>
#include<omp.h>

int main(int argc,char *argv[])
{
   #pragma omp parallel
        {
#pragma omp sections
                {

#pragma omp section
                        {
                                int id1=omp_get_thread_num();
                                printf("Section 1,hello from thread %d\n",id1);
                        }
#pragma omp section
                        {
                                int id2=omp_get_thread_num();
                                printf("Section 2,hello from thread %d\n",id2);
                        }
#pragma omp section
                        {
                                int id3=omp_get_thread_num();
                                printf("Section 3,hello from thread %d\n",id3);
                        }
                }
        }
}
```

# section Directive *Contd..*

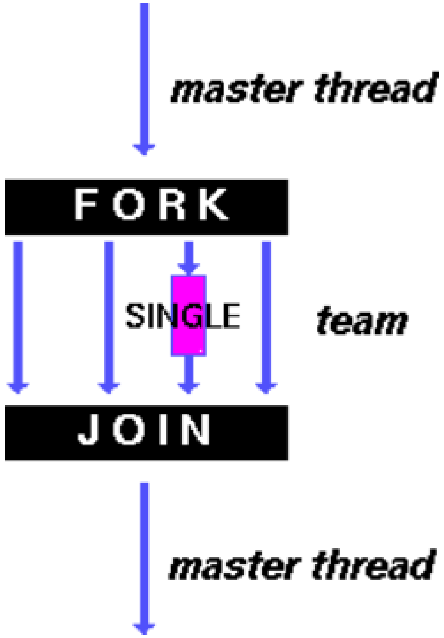


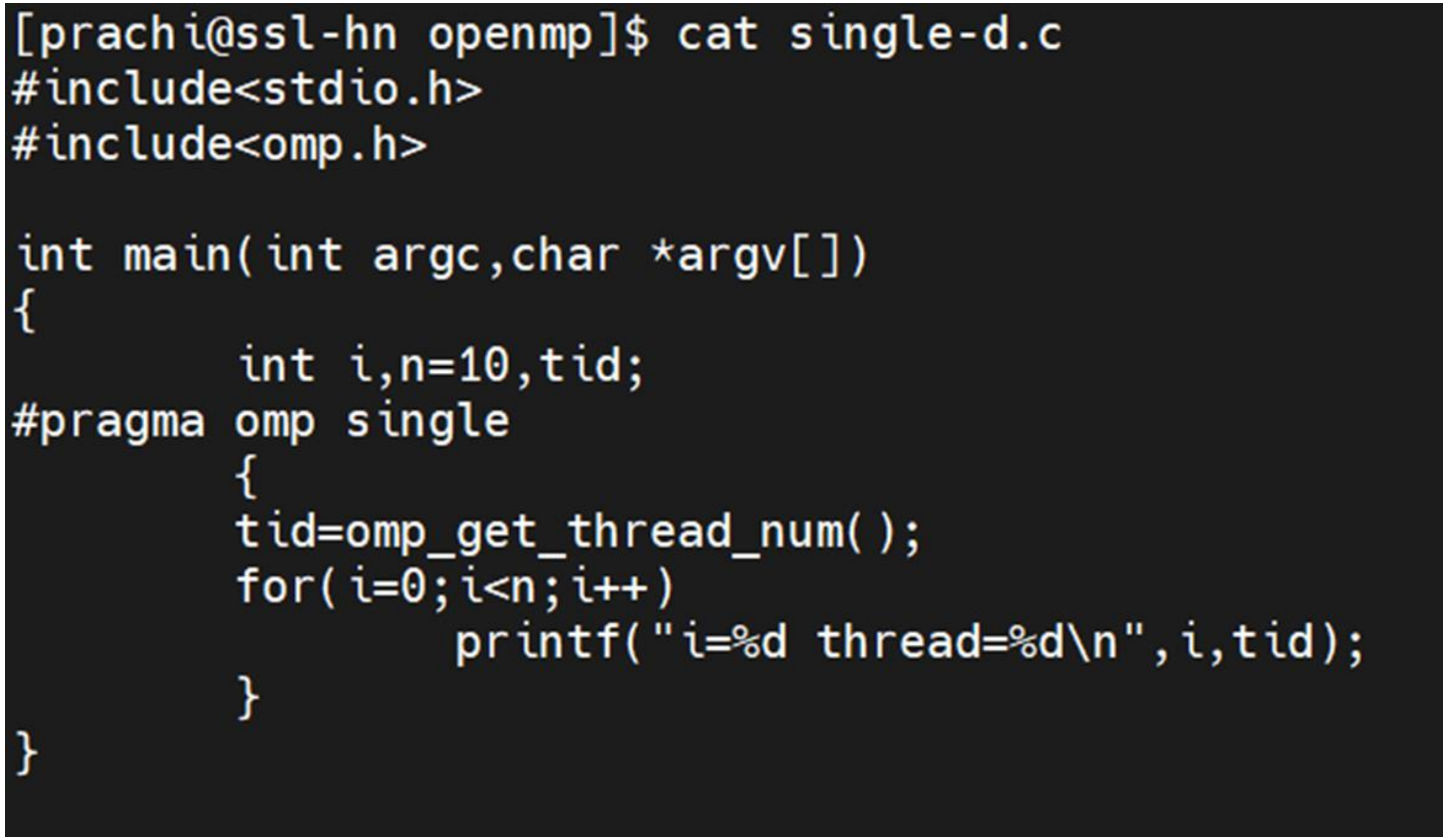
```
[prachi@ssl-hn openmp]$ gcc -fopenmp section-d.c
[prachi@ssl-hn openmp]$ ./a.out
Section 2,hello from thread 0
Section 3,hello from thread 2
Section 1,hello from thread 3
[prachi@ssl-hn openmp]$ 
```

```
• #pragma omp sections [clause ...]  newline
                 private (list)
                 firstprivate (list)
                 lastprivate (list)
                 reduction (operator: list)
                 nowait
{
#pragma omp section   newline

   structured_block

#pragma omp section   newline

   structured_block
}
```

# single Directive

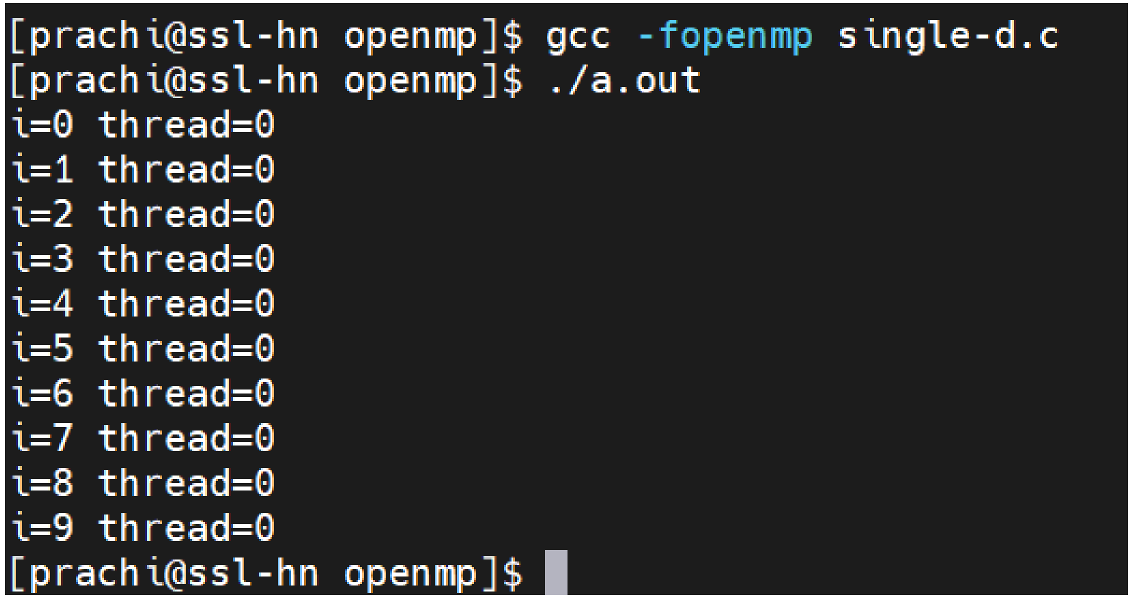
- specifies that the enclosed code is to be executed by only one thread in the team





```
[prachi@ssl-hn openmp]$ cat single-d.c
#include<stdio.h>
#include<omp.h>

int main(int argc,char *argv[])
{
        int i,n=10,tid;
#pragma omp single
        {
        tid=omp_get_thread_num();
        for(i=0;i<n;i++)
                printf("i=%d thread=%d\n",i,tid);
        }
}
```

# Single Directive *Contd..*



```
[prachi@ssl-hn openmp]$ gcc -fopenmp single-d.c
[prachi@ssl-hn openmp]$ ./a.out
i=0 thread=0
i=1 thread=0
i=2 thread=0
i=3 thread=0
i=4 thread=0
i=5 thread=0
i=6 thread=0
i=7 thread=0
i=8 thread=0
i=9 thread=0
[prachi@ssl-hn openmp]$
```

- *#pragma omp single [clause ...]  newline*

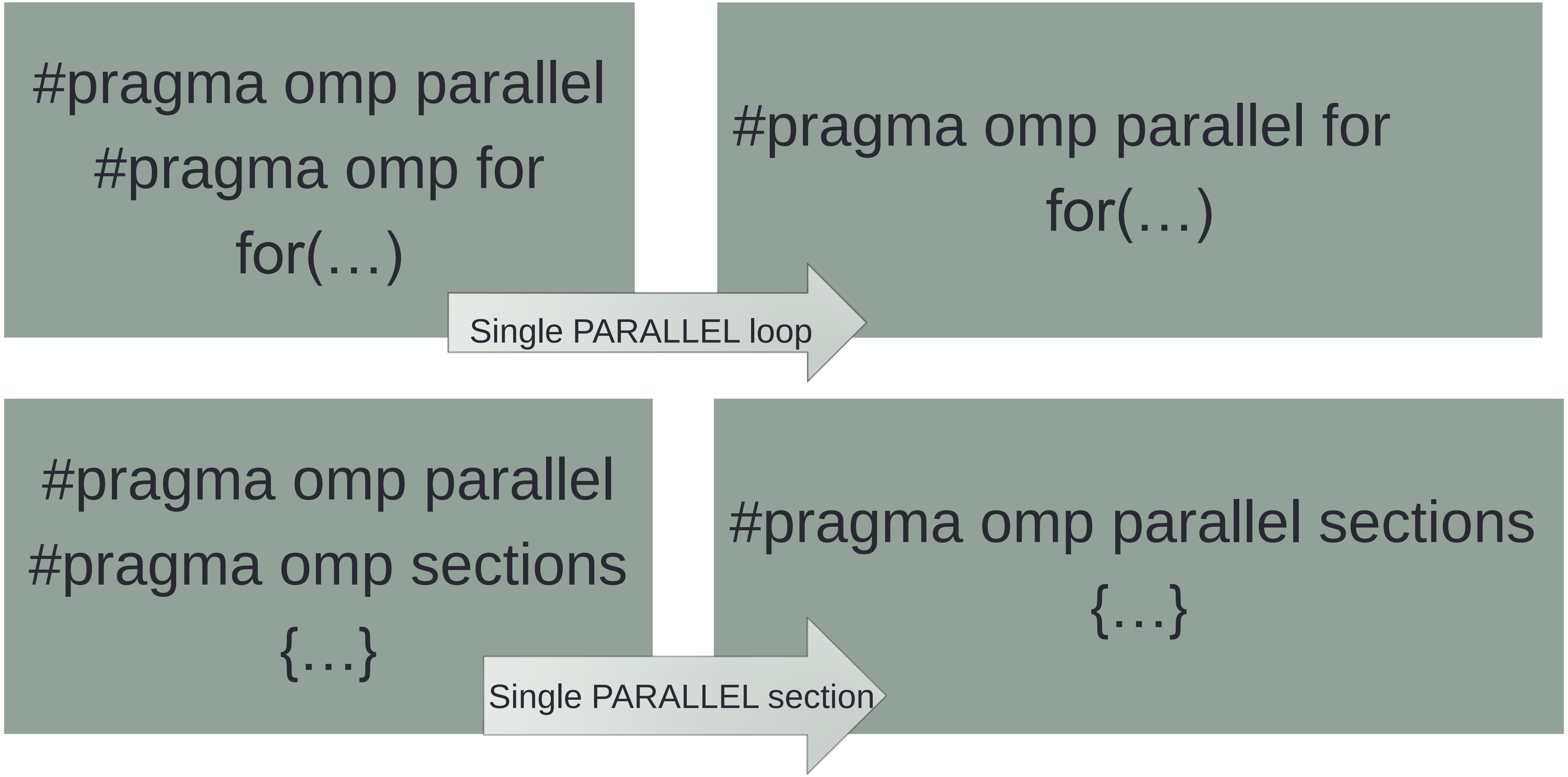
  *private (list)*

  *firstprivate (list)*

  *nowait*

  *structured_block*

# Combined Workshare Directives



```
#pragma omp parallel
#pragma omp for
for(…)
```

Single PARALLEL loop

```
#pragma omp parallel for
for(…)
```

```
#pragma omp parallel
#pragma omp sections
{…}
```

Single PARALLEL section

```
#pragma omp parallel sections
{…}
```

# Synchronization Clauses *Contd..*

## Master

- Specifies a region that is to be executed only by the master thread of the team
- *#pragma omp master*

## Critical

- Specifies a region of code that must be executed by only one thread at a time
- *#pragma omp critical*

## Barrier

- Synchronizes all threads in the team
- A thread will wait at that point until all other threads have reached that barrier
- *#pragma omp barrier*

## Atomic

- Specifies that a specific memory location must be updated atomically, rather than letting multiple threads attempt to write to it
- *#pragma omp atomic*

# Run-Time Library Routines

**omp_set_num_threads**

- Sets the number of threads that will be used in the next parallel region

**omp_get_num_threads**

- Returns the number of threads that are currently in the team executing the parallel region from which it is called

**omp_get_thread_num**

- Returns the thread number of the thread, within the team, making this call

**omp_get_num_procs**

- Returns the number of processors that are available to the program

**omp_in_parallel**

- Used to determine if the section of code which is executing is parallel or not

# Environment Variables

## OMP_NUM_THREADS

- *export OMP_NUM_THREADS[=num]*

## OMP_SCHEDULE

- *set OMP_SCHEDULE[=type[,size]]*

## OMP_NESTED

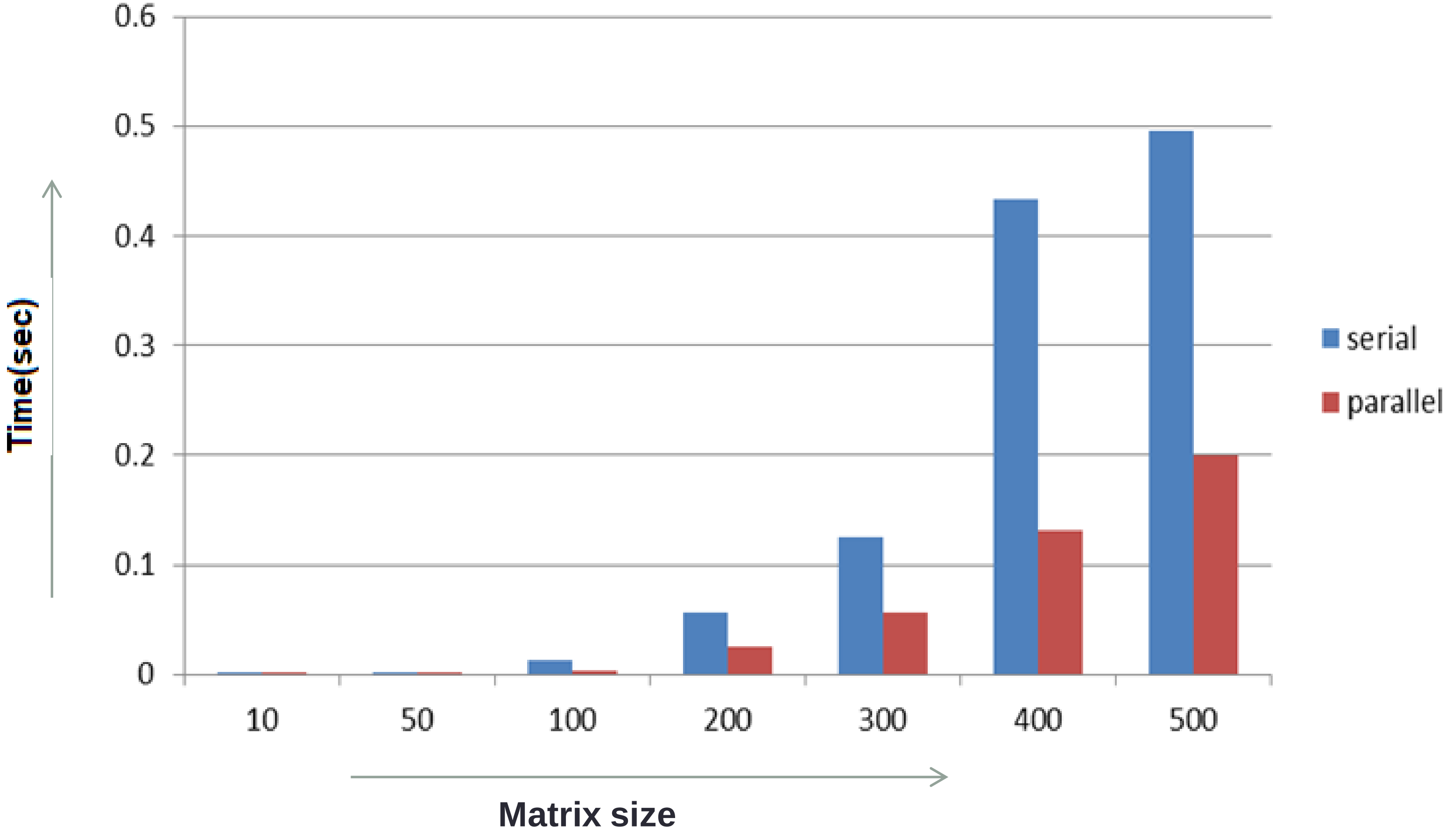
- *set OMP_NESTED[=TRUE | =FALSE]*

## OMP_DYNAMIC

- *set OMP_DYNAMIC[=TRUE | =FALSE]*

# Matrix Multiplication

- No of Threads = 4
- Computation time for parallel code is less compared to serial code
- Computation time for parallel code reduces only for higher number of iterations
- Computation time for lower number of iterations in parallel is more because of the time taken to parallelize code acts as an added overhead



# THANK YOU