1 Hands-on; Shared Memory II; OpenMP

1. **Hello world:** code18.c

   - In this example, the master thread forks a parallel region.
   - All threads in the team obtain their unique thread number and print it.
   - The master thread only prints the total number of threads.
   - Two OpenMP library routines are used to obtain the number of threads and each thread's number.

   Follow the steps below for executing OpenMP code;

```
export OMP_NUM_THREADS=8
gcc -o code18 code18.c -fopenmp
./code18
```

```c
/******************************************************************
 * FILE: omp_hello.c
 * DESCRIPTION:
 *  OpenMP Example - Hello World - C/C++ Version
 *  In this simple example, the master thread forks a parallel region.
 *  All threads in the team obtain their unique thread number and
 *  print it. The master thread only prints the total number of
    threads.
 *  Two OpenMP library routines are used to obtain the number of
 *  threads and each thread's number.
 * AUTHOR: Blaise Barney  5/99
 * LAST REVISED: 04/06/05
 ******************************************************************
    */
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>

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

  int nthreads, tid;
/* Fork a team of threads giving them their own copies of variables */
#pragma omp parallel private(nthreads, tid)
  {
    tid = omp_get_thread_num();   /* Obtain thread number */
    printf("Hello World from thread : %d\n", tid);

    /* Only master thread does this */
    if (tid == 0)
      {
      nthreads = omp_get_num_threads();
      printf("Number of threads = %d\n", nthreads);
      }

  }  /* All threads join master thread and disband */
  return 0;
}
```

2. **Shared Variables:** code19.c

   - OpenMP default is shared variables.
   - To make private, need to declare with pragma:

   Follow the steps below for executing OpenMP code;

```
export OMP_NUM_THREADS=8
gcc -o code19 code19.c -fopenmp
./code19
```

```c
#include <stdio.h>
#include <omp.h>
#include <unistd.h>

int a,b,x,y,num_threads,thread_num;
int main()
{
  printf("I am in sequential part.\n");
#pragma omp parallel num_threads (8)  private (a) shared (b)
  {
    num_threads=omp_get_num_threads();
    thread_num=omp_get_thread_num();
    x=thread_num;
    //sleep(1);
    y=x+1;
    printf("I am openMP parellized part and thread %d. \n X and Y
    values are %d and %d. \n",omp_get_thread_num(),x,y);
  }
  printf("I am in sequential part again.\n");
  return 0;
}
```

3. **Loop work-sharing:** code20.c

   - The iterations of a loop are scheduled dynamically across the team of threads.
   - A thread will perform CHUNK iterations at a time before being scheduled for the next CHUNK of work.

   Follow the steps below for executing OpenMP code;

```
gcc -o code20 code20.c -fopenmp
./code20
```

```c
/******************************************************************
 * FILE: omp_workshare1.c
 * DESCRIPTION:
 * OpenMP Example - Loop Work-sharing - C/C++ Version
 * In this example, the iterations of a loop are scheduled dynamically
 * across the team of threads.  A thread will perform CHUNK iterations
 * at a time before being scheduled for the next CHUNK of work.
 * AUTHOR: Blaise Barney  5/99
 * LAST REVISED: 04/06/05
 ******************************************************************
    */
#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++) /* Some initializations */
    a[i] = b[i] = i * 1.0;
  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)
#pragma omp for schedule(dynamic,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 */
  return 0;
}
```