Purdue Northwest, Hammond, CS 404
Homework 1
Due 2/4/2020 11:59 pm

1 Matrix Multiplication

- Write a parallel C program, parallelized using OpenMP, that reads from the command line one two or three input arguments that decide the dimensions of the input matrices.
- With one command line argument, the program should generate two square matrices, product of which should be computed and written in a file named `output`.
- With two arguments, the two matrices should be such that their dimensions are compatible for their product to be computed. Likewise for three dimensions, the last argument would be the second dimension (columns) of the second matrix.
- Write serialized code to do the job too
  For two input matrices A and B of dimensions $m \times p$ and $p \times n$ respectively

```
double [][] C = new double [ m ][ n ];
for ( int i = 0; i < m ; i ++) {
  for ( int j = 0; j < n ; j ++) {
    for ( int k = 0; k < p ; k ++) {
      C [ i ][ j ] += A [ i ][ k ] * B [ k ][ j ];
    }
  }
}
```

2 OpenMp

- The pre-processor directive `#include<omp.h>` has to be included in the preamble of the program
- The level of parallelization that can be achieved is different for each system. Run the `Hello World` program in the HW package to know your system configuration.

  For example, a quad-core machine with two processors on each core, will launch 8 threads by default for each parallel section

- A program section to be run in parallel has to have its own scope that is preceded by `#pragma omp parallel` for example

```
#pragma omp parallel
#pragma omp for
for(i = 0; i < N; i++ ){
```

```
    a[i] += 3;
    ...
}
```

- Collapsing nested loops:

```
#pragma omp for collapse(2)
for ( i=0; i<N; i++ )
   for ( j=0; j<N; j++ )
      A[i][j] = B[i][j] + C[i][j]
```

- `#pragma omp oredered` can be used to enforce an order among certain statements in a parallel loop
- **Reduction**: Used to avoid the *race* condition among various threads

```
double result = 0;
#pragma omp parallel
{
 double local_result;
 #pragma omp for
 for (i=0; i<N; i++) {
   local_result = f(x,i);
 #pragma omp critical
 result += local_result;
 }
}
```

3 Miscellaneous

- Use functions `srand48`, `drand48()` for generating random real numbers between 1 and 100
- The function `gettimeofday()` can be used to compute how long it takes for the serialized code and the paralellized version. Ultimately report the speed-up you achieved using OpenMp.
- `omp_get_num_threads()` can be used to get the number of a thread
- `set OMP_NUM_THREADS` can be used to set the number of threads in a parallel portion
- Compile your code with: `gcc -fopenmp hw1.c -o hw1`
- Run by doing: `./hw1 arg1` (or more arguments depending on what you are testing)
- Write as many comments as you can to explain your code

4 Files to Submit

Write a `Report.txt` that discusses what you found difficult about this assignment, how you planned your approach to it, and what you learned completing it. Also discuss the time complexity of the serialized and the parallel version of your code. ***Cite all your references***.
Send me only (via email: arpitamhow@gmail.com):
1. hw1.c – 50 points
2. Report.txt – 50 points