CS 213 — Multiprocessor Architecture and Programming

Instructor: Elaheh Sadredini
Programming Assignment 1 - OpenMP

Released: Feb 3
Due Date: Feb 13 - 11:59 PM
Last day to submit: Feb 18 - 11:59 PM (10% penalty per day)

1 Overview

The purpose of this part is to become familiar with OpenMP constructs and programs, using your own computer. The assignment provides basic practice in coding, compiling and running OpenMP programs, covering hello world program, timing, using work sharing for, and sections directives. The OpenMP code is given. You are also asked to parallelize matrix multiplication using the work sharing for directive and draw conclusions. Code for sequential matrix multiplication is given.

1.1 "hello world" program (20 points)

An OpenMP hello world program is given below:

```
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>

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

// Fork a team of threads with 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);

    if (tid == 0) { // Only master thread does this
        nthreads = omp_get_num_threads();
    printf("Number of threads = %d\n", nthreads);
    }
    } // All threads join master thread and disband
    return(0);
}
```

This program has the basic parallel construct for defining a single parallel region for multiple threads. It also has a private clause for defining a variable local to each thread.

Note: OpenMP constructs such as parallel have their opening braces on the next line and not on the same line.

Compile the program on your own computer. Execute the program. Execute the program at least four times. Explain your output. Why does the thread order change?

Alter the number of threads to 32. Here just try adding the following function before the parallel region *pragma*. Re-execute the program.

```
omp_set_num_threads(32);
```

What to Submit:

(a) (10 points) Screenshot from running the hello world program on your computer and explanation of output and thread order.

(b) (10 points) Screenshots of the output from running the program with 32 threads and explanation of what you have observed.

1.2 Work Sharing (20 points)

This task explores the use of the *for* work-sharing construct. The following program adds two vectors together using a work-sharing approach to assign work to threads:

```
#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++)
        a[i] = b[i] = i * 1.0; // initialize arrays
    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(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);
}
```

This program has an overall *parallel* region within which there is a work-sharing *for* construct. Compile and execute the program. Depending upon the scheduling of work different threads might add elements of the vector. It may be that one thread does all the work. Execute the program several times to see any different thread scheduling. In the case that multiple threads are being used, observe how they may interleave.

Time of execution Measure the execution time by instrumenting the MPI code with the OpenMP routine *omp_get_wtime*() at the beginning and end of the program and finding the elapsed in time. The function *omp_get_wtime*() returns a double.

Experimenting with Scheduling Alter the code from *dynamic* scheduling to *static* scheduling and repeat. What are your conclusions? Alter the code from *static* scheduling to *guided* scheduling (chunk size is irrelevant) and repeat. What are your conclusions?

What to Submit:

(a) (5 points) A copy of the source program with timing added.

(b) (5 points) Screenshot from compiling and running the program with the original dynamic scheduling.

(c) (5 points) Screenshots from running the program with static and with guided scheduling

(d) (5 points) Your conclusions about the different scheduling approaches.

1.3 Work-sharing with the sections construct (20 points)

This task explores the use of the sections construction. The program below adds elements of two vectors to form a third and also multiplies the elements of the arrays to produce a fourth vector.

```
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>
#define N 50

int main (int argc, char *argv[]) {
    int i, nthreads, tid;
    float a[N], b[N], c[N], d[N];

    for (i=0; i<N; i++) { // Some initializations, arbitrary values
        a[i] = i * 1.5;
        b[i] = i + 22.35;
        c[i] = d[i] = 0.0;
    }

    #pragma omp parallel shared(a,b,c,d,nthreads) 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 sections nowait
        {
            #pragma omp section
            {
                printf("Thread %d doing section 1\n",tid);
                for (i=0; i<N; i++) {
                    c[i] = a[i] + b[i];
                    printf("Thread %d: c[%d]= %f\n",tid,i,c[i]);
                }
            }

            #pragma omp section
            {
                printf("Thread %d doing section 2\n",tid);
                for (i=0; i<N; i++) {
                    d[i] = a[i] * b[i];
                    printf("Thread %d: d[%d]= %f\n",tid,i,d[i]);
                }
            }
        } // end of sections

        printf("Thread %d done.\n",tid);

    } // end of parallel section
    return(0);
}
```

This program has a parallel region but now with variables declared as shared among thethreads as well as private variables. Also there is a sections work sharing construct. Within the sections construct, there are individual section blocks that are to be executed once by one member of the team of threads.

Compile and execute the program and make conclusions on its execution.

What to Submit:

(a) (10 points) Screenshot from compiling and running the program.

(b) (10 points) Your conclusions.

1.4 Matrix Multiplication (40 points)

A sequential program for matrix multiplication given here:

```
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>
#define N 100

int main(int argc, char *argv) {
    omp_set_num_threads(8);//set number of threads here
    int i, j, k;
    double sum;
    double start, end; // used for timing
    double A[N][N], B[N][N], C[N][N];

    for (i = 0; i < N; i++) {
        for (j = 0; j < N; j++) {
            A[i][j] = j*1;
            B[i][j] = i*j+2;
            C[i][j] = j-i*2;
        }
    }

    start = omp_get_wtime(); //start time measurement

    for (i = 0; i < N; i++) {
        for (j = 0; j < N; j++) {
            sum = 0;
            for (k=0; k < N; k++) {
                sum += A[i][k]*B[k][j];
            }
            C[i][j] = sum;
        }
    }

    end = omp_get_wtime(); //end time measurement
    printf("Time of computation: %f seconds\n", end-start);
    return(0);
}
```

The size of the $N \times N$ matrices in the program is set to 100×100. Change this to the maximum size you can use on your system without getting a segmentation fault.

You are to parallelize this matrix multiplication program in two different ways:

- Add the necessary *pragma* to parallelize the outer *for* loop in the matrix multiplication;
- Remove the *pragma* for the outer for loop and add the necessary *pragma* to parallelize the middle *for* loop in the matrix multiplication;

In both cases, collect timing data with 1 thread, 4 threads, 8 threads, and 16 threads. You will find that when you run the same program several times, the timing values can vary significantly. Therefore add a loop in the code to execute the program 10 times and display the average time.

What to Submit:

(a) For the outer matrix multiplication loop
   - (5 points) Source program listing
   - (5 points) One screenshot from compiling and running the program
   - (5 points) Graphical results of the average timings

(b) For the middle matrix multiplication loop parallelized
    - (5 points) Source program listing
    - (5 points) One screenshot from compiling and running the program
    - (5 points) Graphical results of the average timings

(c) (10 points) Your conclusions and explanations of the results.

2 Assignment Submission

Produce a single pdf document that shows that you successfully followed the instructions and perform all tasks by taking screenshots and include these screenshots in the document. Submit a single zip file for your pdf document and source codes. The name of zip file should include your first and last name.