Homework 6

CSC 140 – Advanced Algorithm Design and Analysis

Follow the "homework procedures" found on Piazza to submit files to DBInbox. If anything in this assignment does not make sense, please ask for help.

Quiz: We will have a closed-note 20-30 minute quiz on this homework in class Mon Apr 15.

Non-submitted work:

Read: CLRS Chapter 27 except Section 27.2 and also, as needed, Victor Eijkhout's book on parallel programming at `http://pages.tacc.utexas.edu/~eijkhout/istc/istc.html` (look for "Check out my MPI/OpenMP book").

Written Problems:

There are three steps to follow for the written problems. Step 1: Do them as if they were homework assigned to be graded (ie, take them seriously and try to do a good job). Step 2: Self-assess your work by comparing your solutions to the solutions provided by me. Step 3: Revise your original attempts as little as possible to make them correct. Submit both your original attempt and your revision as separate files.

Submit two files to DBInbox following the procedure documented on Piazza. The first file should be named exactly **hw6.pdf** and should be your homework solutions before looking at my solutions. The second file should be named exactly **hw6revised.pdf** and should be your homework solutions after looking at my solutions and revising your answers.

NOTE: If you wish to handwrite your solutions you may, but only if your handwriting is very *easy to read and the file you submit is less than 1MB in size. Also, part of your homework grade may be based on my subjective opinion of how seriously you take this process.*

1) Do CLRS Exercises 27.1-1, 27.1-2.

2) In a divide and conquer algorithm following recurrence relation $T(n) = aT(n/b) + f(n)$ is $T_\infty(n)$ always asymptotically faster than $T_1(n)$? Explain.

3) Discuss the results of your programming problems. Include a description of your strategy to maximize speed, the amount of parallelism available in the algorithm (ie, T_1/T_∞) and how close you were to achieving this speedup. Describe the system you tested on and what influence that may have had on your success.

Programming:

Due: The program(s) may be first graded sometime – maybe hours, maybe days – after the quiz.

A) Implement a serial version of P-Merge from CLRS Chapter 27 (ie, don't use any OpenMP) using the following header. Submit to DBInbox one file named **hw6_A.c**. See below for more requirements about C code.

```
void pmerge(int t[], int p1, int r1, int p2, int r2, int a[], int p3)
```

B) Implement a parallel version of P-Merge from CLRS Chapter 27 (ie, do use OpenMP) using the same header. Make it as fast as possible using strategies seen in class. Submit to DBInbox one file named **hw6_B.c**. See below for more requirements about C code.

C) Below is an implementation of mergesort. Because mergesort requires spare space proportional to the array being sorted, this implementation uses the strategy of having the client pass two equal size

arrays and having the result of the sort placed in the secondary array. At each level of recursion it sorts from one array to the other, with the end result being placed into `dst`. Duplicate hw6_B.c into a file hw6_C.c and add this code. Make it parallel and as fast as possible using strategies seen in class. Submit to DBInbox one file named **hw6_C.c**. See below for more requirements about C code.

```
void ms(int a[], int b[], int p, int r, int a_to_b) {
    if (p < r) {
        int q = (p+r)/2;
        ms(a,b,p,q,!a_to_b);
        ms(a,b,q+1,r,!a_to_b);
        if (a_to_b)
            pmerge(a,p,q,q+1,r,b,p);
        else
            pmerge(b,p,q,q+1,r,a,p);
    }
}

void mergesort140(int src[], int dst[], int n) {
    memcpy(dst,src,n*sizeof(int));
    ms(src,dst,0,n-1,1);
}
```

C Requirements

For each of the C files you submit, do not include a main program. You should write your own test programs in a separate file and compile them like `gcc -O2 -fopenmp -Wextra -Wall -std=c99 hw6_A.c main.c`. But, once your testing is over, submit only the required file and not your tests. You will get no credit if compiling emits any warnings or you #include any files other than the standard C headers (`https://en.cppreference.com/w/c/header`).

You may find it useful to test your programs using Clang's address sanitizer which exposes buffer overflows. To do this, you'd compile like this (without OpenMP): `clang -O0 -g -fsanitize=address -Wextra -Wall -std=c99 hw6_A.c main.c`. Clang can be found on Macs, Athena, and Apollo.

All submitted code should include a comment at the beginning with your name, 4-digit code, and brief explanation of the what's in the file. Furthermore, any hard-to-understand code should have local comments to help the reader understand.