Min, Max, Sum - IPC Project

13:332:434 - Spring 2017

Richard Ahn - 157004402 - zac.blanco@rutgers.edu
Zachary Blanco - 158007117 - richahn2@gmail.com

Project Information:
All function for parts A-D are contained within **minsmax.c**. The functions are documented within **minsmax.h** and available by #include "**minsmax.h**"

Some utility functions are found within **util.h** as well. Mainly for reading the file.

The tests which we ran on our code are in **test.c**.

Tests Description
We used a function to generate random test files, but we also had two static files for our tests, **t1.txt** and **t2.txt** (Please find them in our attached submission)

They had min, max and sum
- t1 → {5, 55, 123}
- t2 → {-814, 40, 0}

For each of the 4 parts we individually tested each method with the number of processes for parts B-D ranging from 1-10 for each test. We made sure these numbers were correct by using the **assert.h** system library (See assert(3) of the linux man pages)

Our other set of tests will generate a file containing a specified amount random numbers. As the file is generated, the min, max, and sum are calculated. Then after generating and writing the file we test all four methods on the generated file using varying amounts of processes to ensure that the methods work as intended.

During the process of generating random files we also time each run of the algorithm (See *Extra Credit Information* section below)

Makefile

- Run all tests
  - make test
- Compile library and test file
  - Make

The compiled executable is named **a_test.out**

Extra Credit Information:
To record the execution times of the various implementations, we used the gettimeofday() function provided by the standard library **time.h**. To measure the execution time, we called the function before and after the program to be tested, and we found the difference in microseconds.

Our initial belief for the runtimes of these various functions was that the multiprocessor programs would mostly outpace the single processor speeds, and that the recursive and iterative variations would run about the same. When we tested our code, however, we noticed that on average, the single processor program ran faster than the others. We believe that this may be due to the OS copying the entire heap and stack space that includes the large array, which is a very time consuming task. This observation then tells us that for tests with a higher number of processes, the iterative and recursive version timings should increase as the number of processes increase, assuming that the time to copy the array outweighs the smaller time each process must take to run the **minsmax** algorithm on the smaller portion of the array that they have. We found this to be true when we compared the timings of the iterative algorithm with 10 and 20 processes on a test file of size 100k. In one instance, the time for the program with 10 processes took 14.8 milliseconds, but the time for 20 took 20.6 milliseconds.

We designed our test to be given the choice of the number of processes to use, since it wasn't mathematically clear what number would be the most optimal. We chose values ranging from 1 and 21, and we found that the results were that the higher the number of processes, the longer the programs took, particularly for the recursive, iterative, and combined versions.

See **test.txt** within our submission to view the full results of our timing test cases.

*Note: running "make test" will generate the same output that is in **test.txt***