Erika Parayno
April 15, 2020
CSS 503
Prog 1: Convex Hull

Part 1: Code

Source code: See zip file.

Execution output: See image below.

```
erika@erika:~/Desktop/CSS503/Prog1$ ls
convex  convex.cpp  convex_mp  convex_mp.cpp
erika@erika:~/Desktop/CSS503/Prog1$ g++ convex.cpp -o convex
erika@erika:~/Desktop/CSS503/Prog1$ g++ convex_mp.cpp -o convex_mp
erika@erika:~/Desktop/CSS503/Prog1$ ./convex 100
do you want to display initial data? n
elapsed time = 1702
do you want to display result data? y
point[47].x = 7739.000000 .y = 12.000000
point[72].x = 9503.000000 .y = 19.000000
point[27].x = 9956.000000 .y = 1873.000000
point[45].x = 9932.000000 .y = 5060.000000
point[74].x = 9708.000000 .y = 6715.000000
point[28].x = 6862.000000 .y = 9170.000000
point[99].x = 5928.000000 .y = 9529.000000
point[15].x = 3929.000000 .y = 9802.000000
point[64].x = 709.000000 .y = 8927.000000
point[32].x = 336.000000 .y = 6505.000000
point[85].x = 124.000000 .y = 4914.000000
point[53].x = 97.000000 .y = 2902.000000
point[33].x = 846.000000 .y = 1729.000000
point[50].x = 795.000000 .y = 570.000000
point[5].x = 2362.000000 .y = 27.000000
erika@erika:~/Desktop/CSS503/Prog1$ ./convex_mp 100 4
do you want to display initial data? n
elapsed time = 551
do you want to display result data? y
point[47].x = 7739.000000 .y = 12.000000
point[72].x = 9503.000000 .y = 19.000000
point[27].x = 9956.000000 .y = 1873.000000
point[45].x = 9932.000000 .y = 5060.000000
point[74].x = 9708.000000 .y = 6715.000000
point[28].x = 6862.000000 .y = 9170.000000
point[99].x = 5928.000000 .y = 9529.000000
point[15].x = 3929.000000 .y = 9802.000000
point[64].x = 709.000000 .y = 8927.000000
point[32].x = 336.000000 .y = 6505.000000
point[85].x = 124.000000 .y = 4914.000000
point[53].x = 97.000000 .y = 2902.000000
point[33].x = 846.000000 .y = 1729.000000
point[50].x = 795.000000 .y = 570.000000
point[5].x = 2362.000000 .y = 27.000000
erika@erika:~/Desktop/CSS503/Prog1$ ./convex 20000 1
do you want to display initial data? n
elapsed time = 71517
do you want to display result data? n
erika@erika:~/Desktop/CSS503/Prog1$ ./convex_mp 20000 1
do you want to display initial data? n
elapsed time = 94496
do you want to display result data? n
erika@erika:~/Desktop/CSS503/Prog1$ ./convex_mp 20000 2
do you want to display initial data? n
elapsed time = 46803
do you want to display result data? n
erika@erika:~/Desktop/CSS503/Prog1$ ./convex_mp 20000 3
do you want to display initial data? n
elapsed time = 45313
do you want to display result data? n
erika@erika:~/Desktop/CSS503/Prog1$ ./convex_mp 20000 4
do you want to display initial data? n
elapsed time = 38537
do you want to display result data? n
erika@erika:~/Desktop/CSS503/Prog1$
```

Part 2: Report

Design document of your parallelization strategies including explanations and illustration in one or two pages:

This program modifies the existing divide_and_conquer method to allow for parallelization of one task. This is done to take advantage of the multi-core architecture of modern-day machines.

The program will take in the two arguments, the number of points and the number of processes. In the divide_and_conquer method, the number of processes will be passed as parameter, leaves. First it will divide deque<Point> q into s1 and s2. Then it will check if it has more than 1 leaves because if it has more than 1 leaves, then you will need to fork the process. If leaves =1, as root will always equal to one leaf, then it will behave like the divide_and_conquer code from the original convex.cpp file.

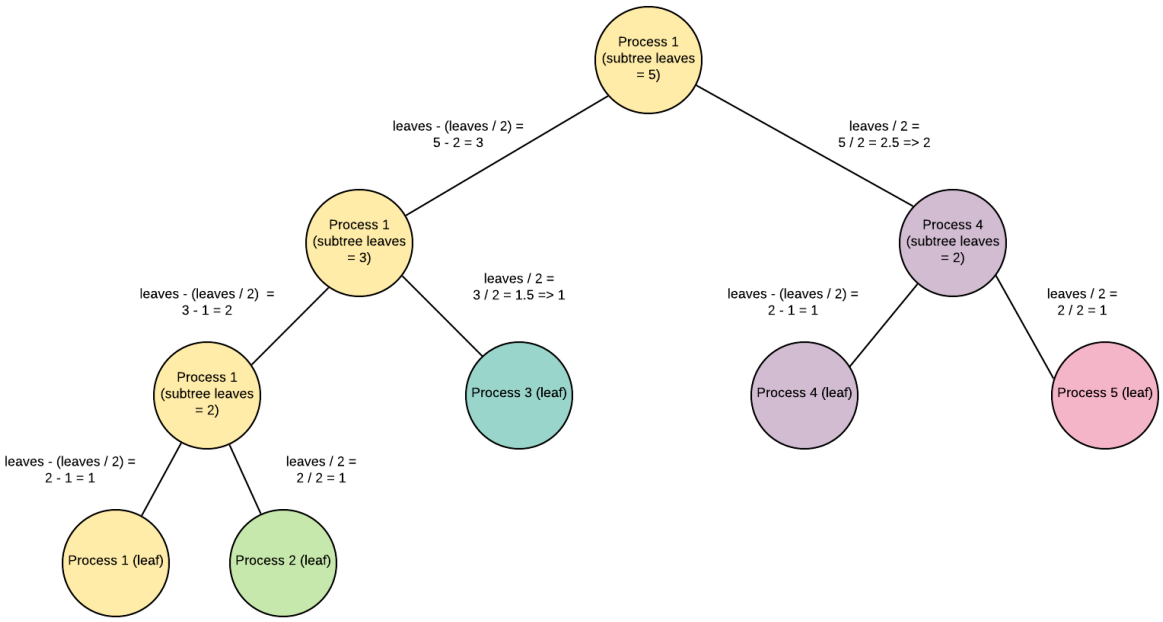

Figure 1. How leaves are calculated and used as parameters when using recursive function, divide and conquer.

Furthermore, if leaves > 1, we will fork the process. However, before forking the process, we must initialize the pipe. If creating pipe is successful, we will then proceed to fork. Then there will be another if and else statement, determined by the pid value. If the pid value is 0 then it is a child and if it is greater than 0, it is a parent.

In the child method we will work on the right side. If s2 is greater than 1, we will perform divide_and_conquer, passing in s2 and (leaves / 2) as parameters. Recall, in order to reach leaves number of processes, we must divide the leaves in half when passing at

as a parameter in the recursive function. (The exception for odd numbers will be handled in the parent divide_and_conquer call). See figure 1 to see how this was calculated in a process tree with 5 leaves. After calling the divide_and_conquer method, we will then close the read side of the pipe and call send method to write on the write side of the pipe. Once this is done, the pipe will be ready to be read by the parent.

When the program enters the parent process, it will work on the left side, read the right side of the pipe, and then proceed to perform merge and graham. As mentioned earlier, the parent will be handling the odd number of leaves. This is done by passing s1 and [leaves – (leaves/2)] as parameters into the recursive function, divide_and_conquer. This will handle the left side. In order to get the right side, we must read from the pipe, in which the child wrote on. To do this, we must first close the write side of the pipe and call recv to read from the read side of the pipe. Prior to receiving, we must clear s2. Once this is complete, we will have the updated s1 and s2, and it will be ready for the merge and graham calls.

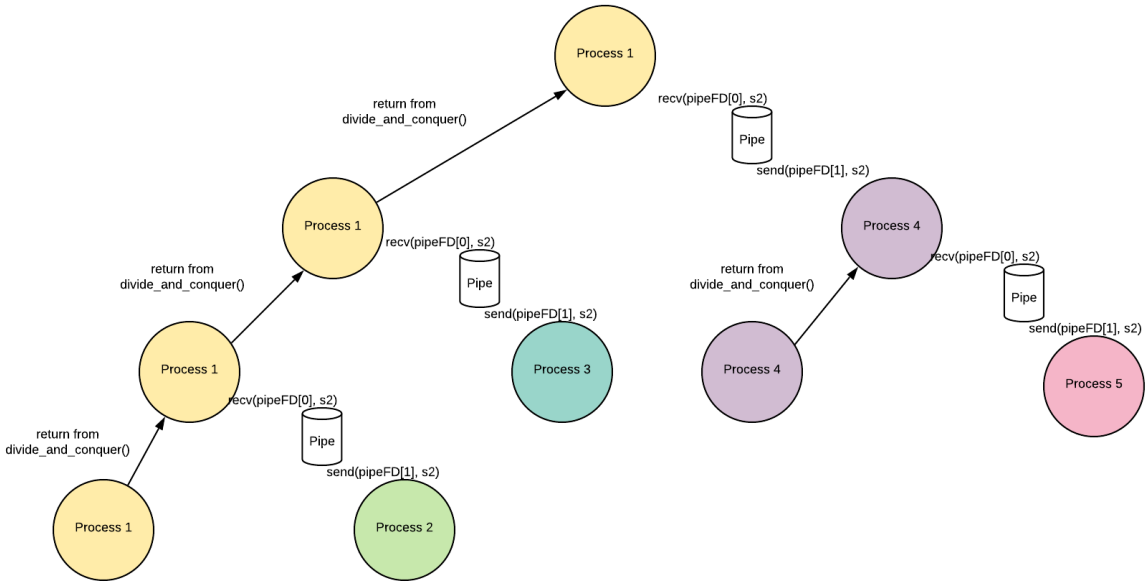

Figure 2 Returning back to the parent. A modified version of figure 6 from assignment handout.

Discuss results in one or two pages. This includes analysis of the effectiveness of your parallelization, possible performance improvement of your program, and limitations of your program:

- Analysis of the effectiveness of parallelization:

	Expected	Actual
Parallelization	Your program creates argv[2] – 1 child processes and involve all argv[2] processes in parallel computation.	My program creates argv[2] – 1 child processes and involve all argv[2] processes in parallel computation.
The performance improvement with two processes applied to 20,000 points	>1.7 times	=2.0
The performance improvement with four processes applied to 20,000 points	>2.3 times	=2.45

  My program works as expected and meets expected performance. Ensured that my points are correct by comparing it to convex.cpp results.

- Possible performance improvement of program:
  - **Increase hardware:** Use a computer with at least 4 CPUs to get near the expected results.
  - **Increase process argument**: Increase number processes to improve performance. There is an exception. See limitations of program for more information.
  - **Minimize background activity**: It is best to close all other programs to make sure you can make the most of your CPU to optimize performance.

- Limitations of program:
  - **Effect of hardware and concurrent programs**: As noted in the above section, number CPUs and programs running in the background can slow down performance of this program. This can introduce variance in performance between different machines and users.
  - **Amahl's law**: As one can speculate with testing how performance may change when increasing from 1 to 2 processes or 1 to 4, you might think that this behaves in a linear fashion. However, according to Amahl's law, this is not the case. Amahl's law predicts theoretical speedup when using multiple processes. Amahl's law is formulated as latency = 1 / [(1-p) + (p/s)]. When looking at how this looks on a graph, you'll see that different programs will have different parallelportion (some programs are more

parallelizable than others). However, all of them follow a similar pattern: the law of diminishing returns.

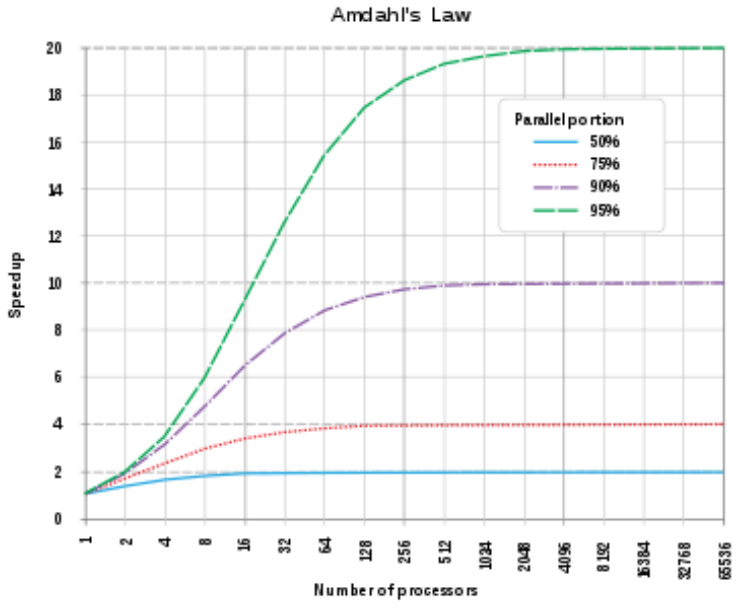

Figure 3 Amahl's law, source: https://en.wikipedia.org/wiki/Amdahl%27s_law