Parallelization Design

Purpose: Report Design document of parallelization strategies including explanations and illustration in one or two pages. (No more than two)

As the figure 5(divide phase) and figure 6(conquering phase) shown in the program1 instruction, we're expected to halve the current task and let parent and child take on its half part respectively, this action is done by calling fork() system call. When the number of children goes up to the expected limit (ie, leaves - 1), parent stops spawning, instead, parents and children go back, send back information to the parent who is at the higher level.

Every time before a parent forks a new child, a pipe is created in advance so that the parent and child process can share the same pipe.

```
//declare file descriptor and create a pipe
int pipeFD[2];
pipe(pipeFD);
```

In the conquer phase where a parent receives a sub convex hull from a child, the pipe I created allows the child to send its sub convex hull back to its parent using recv().

```
//parent receive sub convex hull from child through pipe
close(pipeFD[1]);
recv(pipeFD[0], s2);
```

On the other side, in the divide phase when sending a sub convex hull from a child to its parent, send() is used to store information in the pipe.

```
//send child's sub convex hull back to pipe
close(pipeFD[0]);
send(pipeFD[1], s2);
```

Worth to mention, fork a child process and let it work on the right subset, whereas have the original parent take care of the left subset. Parent should wait until children process completed, using wait() system call.

When child process is executing,

```
divide_and_conquer(s1, leaves/2);  // work on the left queue
```

When parent process is executing,

```
divide_and_conquer(s2, leaves - leaves/2);  // work on the right queue
```