**University of Washington Bothell**

**CSS503: System Programming**

**Program 1: Parallelizing a Convex-Hull Program**

**Snehal Jogdand**

## **Design**

The convex hull program recursively divides a list of all points by two until each subset includes at least two, and thereafter merges those that make a convex. By utilizing multiprocessing this functionality can be paralyzed to gain some efficiency. At each division of list, we can spawn child processes to compute a subpart of the result. The child processes can then parallelly work on the computation and the results can be merged back to parent process.

Once the Q is divided into two smaller subsets (say left and right subset), I use fork() to spawn a new child process. The child process works on the right subset, whereas the parent takes care of the left subset. I have used pipe() to allow the child process to send its sub convex hull result back to the parent. Once the child process computes the sub convex hull, it writes it queue result to the pipe. The same result is then read by the parent process. The parent waits for the child process, then uses the two results, merges them, and apply the graham algorithm to get the convex hull.

For analysis of effectiveness of parallelization, the leaves variable decides the number of sub processes to be created. So, if leaves are 4, we would spawn 4 sub processes to compute a subset of the result during the recursion.

**Divide Phase:**



In divide phase, we fork to spawn new process. Each child process then recursively works on the subset to get the sub convex hull. The parent process waits for the child process to complete before merging the convex hull results.

Conquer Phase:

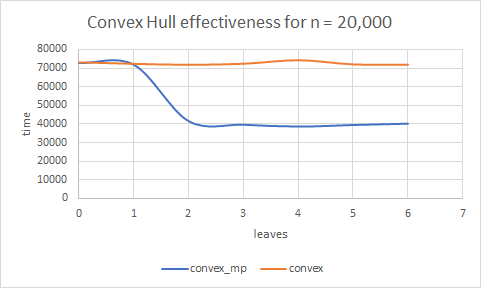


In conquer phase, we fork and compute the right subset. We wait for child process to complete. In parallel parent process processes left subset. Once all is done, the results are merged, and graham is applied to get the convex hull.

## **Results**

### Analysis of the effectiveness of your parallelization

Utilizing multiprocessing we can effectively parallelize the program and gain some efficiency. I ran both convex and convex\_mp program over a 2 CPU machine. The results show **~1.9** times increase in execution time when multiprocessing is used.



* For a smaller leaves value of 0 and 1, the convex\_mp program works same as a convex program since we don’t spawn any processes.
* As number of process increase beyond 2, we get the advantage of parallelization and multiprocessing. The execution time increase by ~1.9 times.
* As the number of leaves increase beyond 2, the program is limited by number of CPUs available. The results remain same as is. Adding more leaves isn’t helping execution time anymore.
* As the number of CPU’s would increase, we would get additional advantage of increase in leaves or number or processes.

### Possible performance improvement of your program

We currently divide the Q in just 2 subsets. One half of the subset is then processed by one child process. We do this recursively, but at each recursion, the number of child process is limited by the number of subsets to be created. If we have more than 2 CPU’s available, we are not taking the full benefit of parallelization. If we can divide the subsets into multiple subsets, say 4 subsets at each recursion for 4 CPU’s, we probably would get more benefits and could see higher performance improvements as number of CPU’s increase.

### Limitations of your program

The program is heavily limited by the number of CPU’s available. After certain leaves value, the execution time won’t improve which can be seen in our results graph above as well. In the results, we see that after 2 leaves, the execution time doesn’t vary by much.