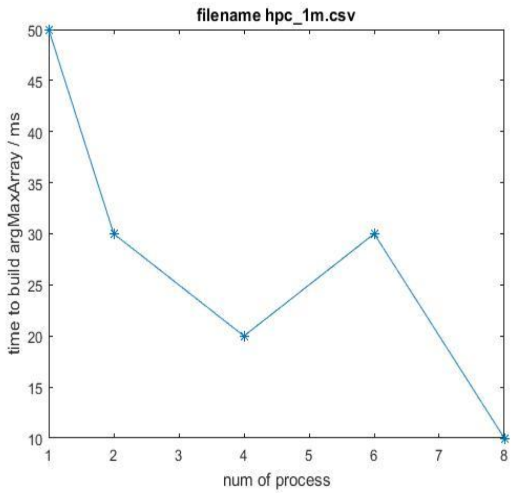
Rui Huang rhhq7

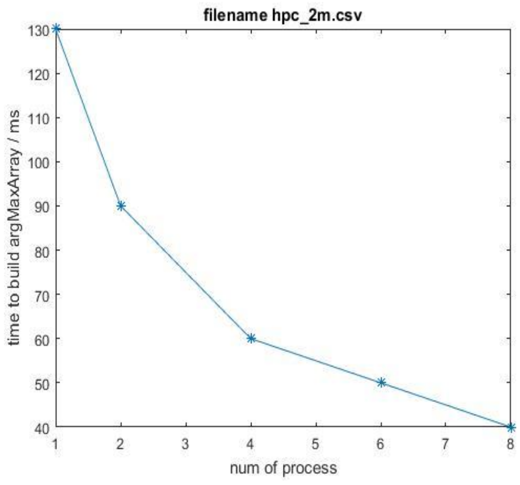
For homework two, the task is to calculate how many pairs of (X, Y) in each bucket. In order to do this, I first converted the file into a two dimensions array called data, and built a one-dimension array called argMaxArray, the ith element of which is the argmax of the ith row in data. Based on the argMaxArray, we can build a one-dimension array called bucketSize, each element is the size of the corresponding bucket. The way I did this is to go through the argMaxArray, for each element, if the value is i, I increased the value of the ith element in bucketSize by 1. In order to make the program run faster, I parallelized two parts of the program: one part is building argMaxArray, the second part is building bucketSize. The following part of this report will discuss the details about the parallelization and analysis the time trend and also what I learned from this homework.

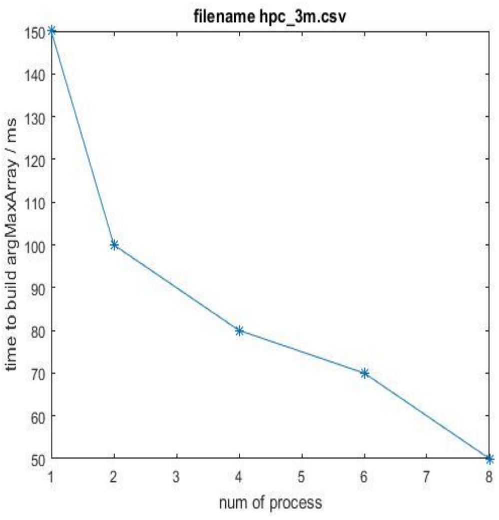
1.Build argMaxArray

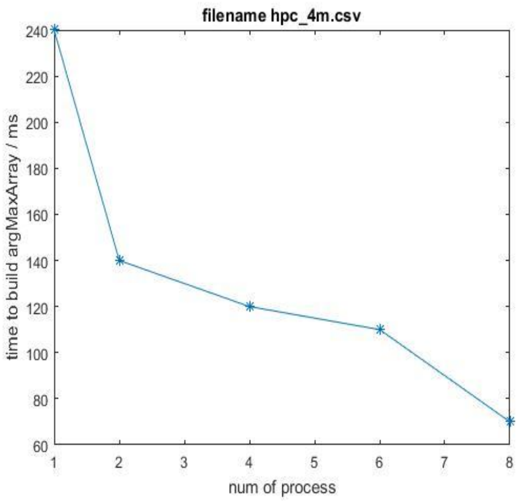
The naïve method is to go through each line of the array called data (this array is mentioned above, I will refer it to data in following context) using one process. The problem of this method is that this array is pretty big and it will cost a lot of time. There is some potentiality to parallelize. If I put data in a shared memory, many process can simultaneous work on it without affect other. The only thing I need to pay attention to is making sure which row need to be processed by which process. I schedule it in this way: suppose I have 3 processes, the parent will process the 0th row, 3th row, 6th row,9th row ……, the first child processes 1th row, 4th row, 7th row, 10th row……, the second processes 2th row, 5th row, 8th row 11th row……. In this way, if the time to build argMaxArray with only one process is T0, if I have n process, the time to build argMaxArray will be T0/n ideally. There are many reasons why it is not T0/n in practice. One of them is that it cost some time to schedule processes for operating system. Another one is that there is more cache miss if I use many process. To build the argMaxArray, I used shared memory, but there is no need to use semaphore, because each element of argMaxArray will be write only once. There is no situation like two processes read the data simultaneous and thesome operation on the data and write it back. Following are some the pictures about time it cost to build argMaxArray with different number of process and different files.

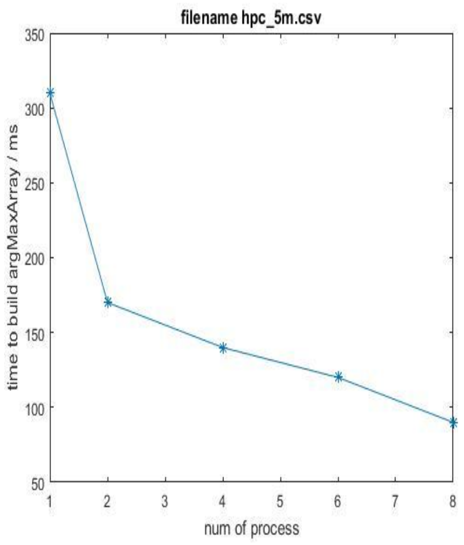
Rui Huang rhhq7

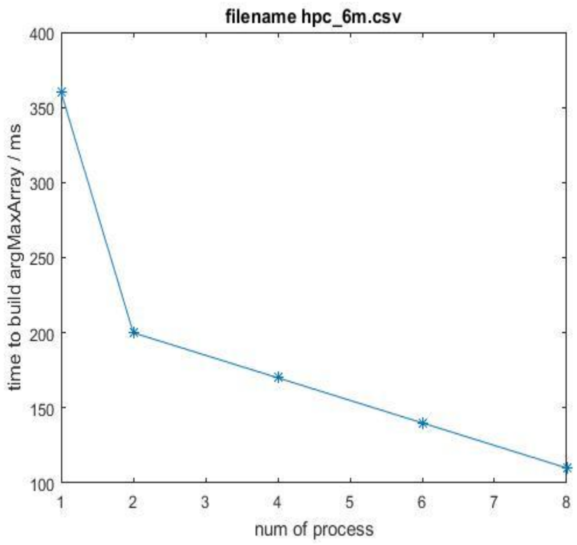




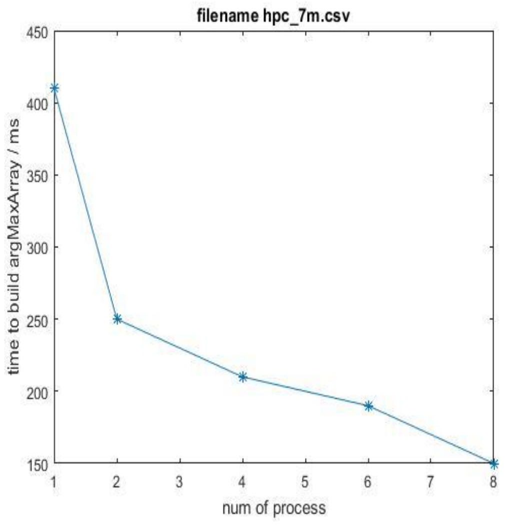


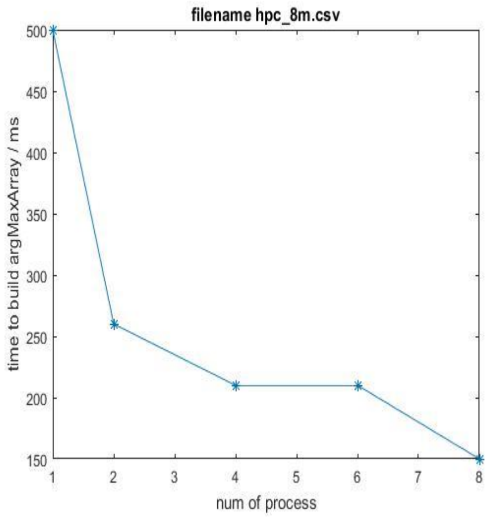


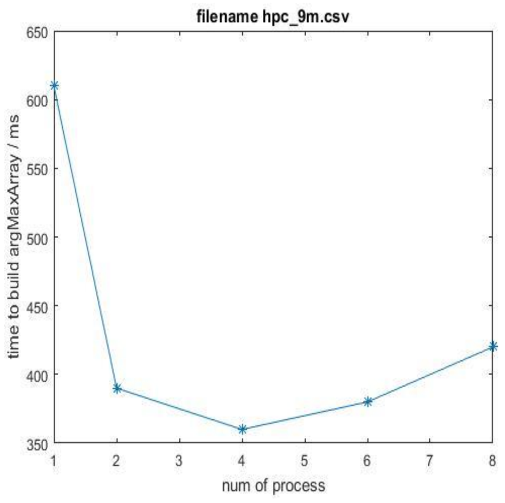




Rui Huang rhhq7



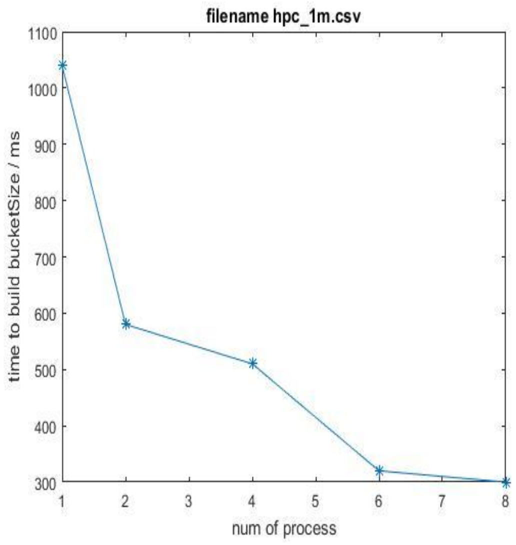


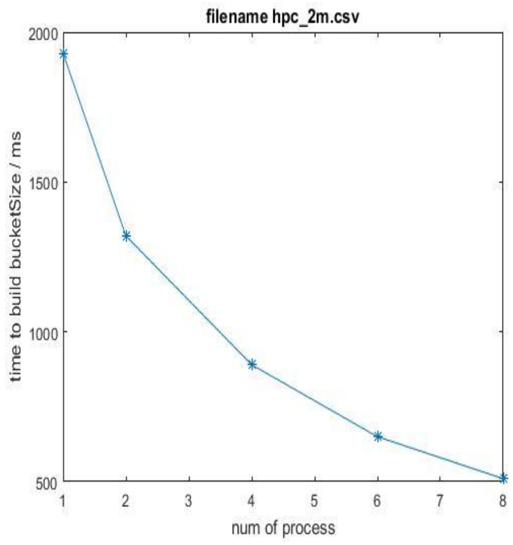


In the first picture(hpc_1m.csv) and last picture(hpc_9m.csv), I notice that the program not always run faster with a larger number of process and this situation didn't occur at other picture (other files). The reason probably is that the operating system spent more time to schedule and the cache miss more often. From these images, we can know that the bigger the file is, the more benefits I can get from multi-process.

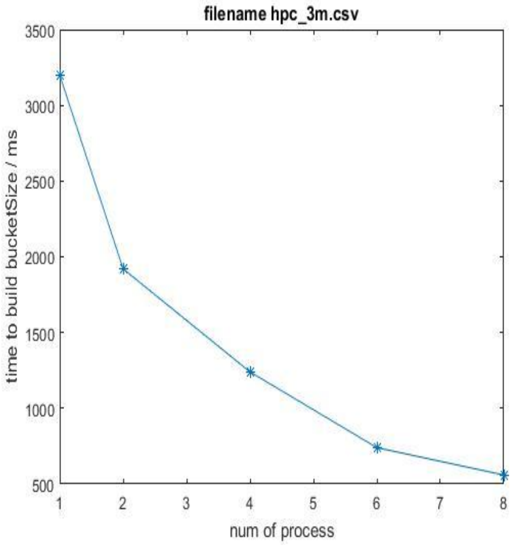
2.Build BucketSize array

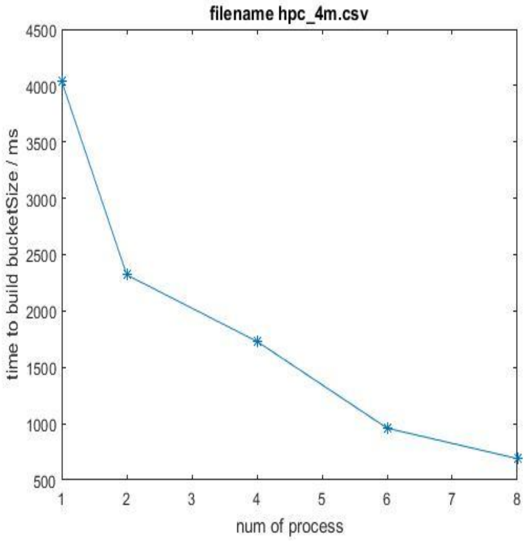
The naïve way to build the bucketSize is use one single process to go through the argMaxArray. For each element in argMaxArray, if the value is i, I just increased the ith element of bucketSize by 1. This method was slow. There was some potentiality to do parallelization. If there were many process, different processes can work on different elements of bucketSize. For example, process 1 increased the first element and simultaneous process 2 read the third element of bucketSize and increased it. There are 21 buckets, which means the length of bucketSize is 21. When the number of process is less than 21 and becoming larger, the time it cost to build the BucketSize would keep decreasing. However, if the number of process was larger than 21, there was no benefit I could get from increase the number of process. It was easy to understand that if there were more than 21 buckets, at least one of them was idle because all the bucketSize elements were occupied by other process. Another big issue of how to parallelize this part was that if one process read an element from bucketSize and increase it, and before the process wrote the value back to the bucketSize, another process read the same element. This will cause the wrong result. I use semaphore to solve this question. I create 21 semaphores for each element of bucketSize. Each time a process visited an element of bucketSize, the element was locked, after the process finished increased the element, the element was unlocked. In this way, different processes could cooperate. If there were 3 processes, first process read the 0th, 3th, 6 th …… elements of argMaxArray and increased the corresponding element in bucketSize, the same as the second,third process. The only difference is that the second read the 1th, 4 th, 7 th…… elements, and the third process read the 2th, 5 th, 8 th…… elements. Because I use semaphore to prevent different process from working on the same element simultaneous, the result should e right. The result print out by the program can prove the correctness of my code. Following are some the pictures about time it cost to build bucketSize with different number of process and different files.

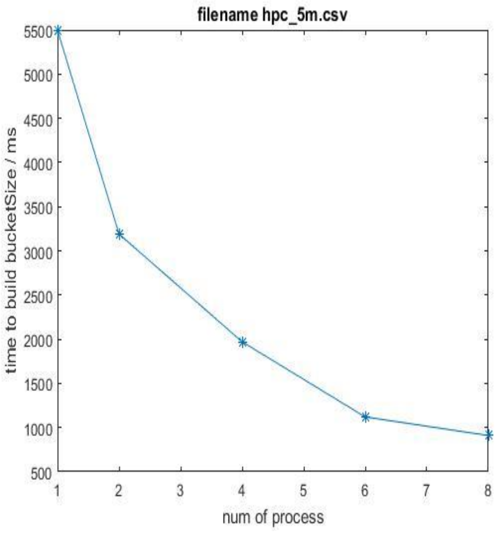


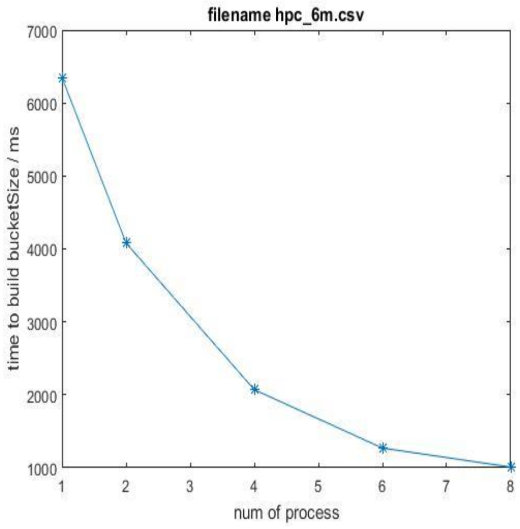


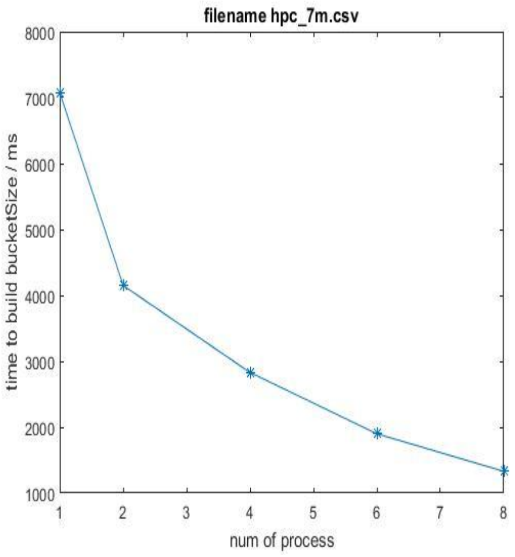
Rui Huang rhhq7

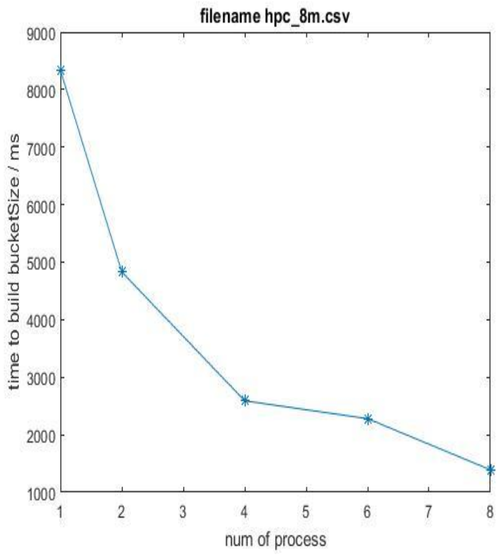


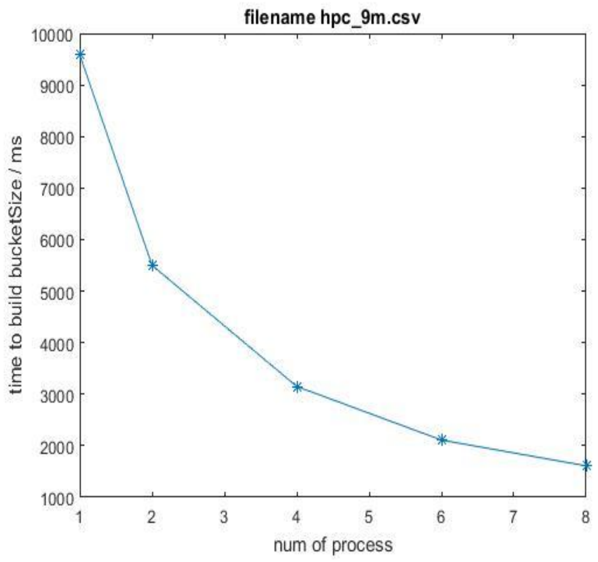












3. What I learned

Before I did this homework, I have never written program to create shared memory and semaphore and use multi-process to speed up the program. After finished this homework, I can't believe what I have done. I feel very good and I am a better programmer than I was before. Also, I have meet some bugs when doing it. For example, I didn't release the shared memory after the program end. I thought when main process end, shared memory would be released by operating system. After talking to professor, I knew I am wrong. Another thing I learned is that I learn how to use GDB from this homework.