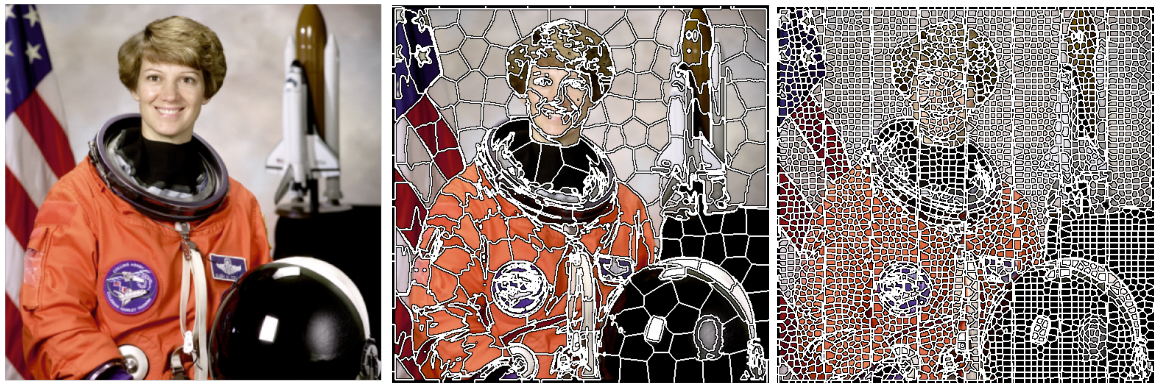
CS 7080 High Performance Computing
Homework 4
OpenMPI - Message Passing Interface

Haoxiang Zhang
PowPrint: hzny2

Due: November 3, 2021

1 Homework Description

homework 4, we need to use OpenMPI (Message Passing Interface) on Superpixel algorithm from homework 2 first part. In the Fig.1, it shows how the SLIC Superpixels alogrithm works. Once the code could run successfully, we will make the code parallelized with *mpi.h*.



(a) Input image (b) Output image - 256 superpixels (c) Output image - 4096 superpixels

Figure 1: SLIC Superpixels Algorithm Example

2 Homework Details

For homework 4, we used the same code from homework2. In Fig.1b and Fig.1c, it is easily to see the how good results that the superpixel algorithm did.

In this homework, I used OpenMPI to implement the parallel part. MPI is paralleling the code with multi nodes. These nodes are from different computer or the nodes from a computing server. When doing MPI, all of the nodes will do the same thing as the master node. Therefore, in this homework, I have to manage the works for nodes well. Because the kseeds and klabels are updating every iteration, all of the nodes need to get updated the variables at the end of each iteration. For MPI, we don't need to give the parallel level. MPI will figure out how many nodes are available.

In Algorithm 1, it shows the details about the whole program. In the PerformSuperpixel() function, the code about calculating distances and labeling are using multi-nested-loop, which cost a lot extra time. Therefore I decided to parallelize this part with MPI. However, when finish the distance calculation part, it should goes to recalculate centroid part, then go to next iteration. I collected the results from each worker node and summarize them to klabels before recalculate centroid in Master node's work. After that, broadcasting kseeds of l, a, b, x, and y to all the nodes make sure all the nodes have the updated kseeds for next iteration. In the parallel work, I initialize the labels array to -1. Then, when the master node receive the result, it knows which pixels that the worker is working on, and which pixels are not the worker's job.

Algorithm 1 SLIC SuperPixels with OpenMPI Parallelization

```
Input
    $F$      the input file path
    $K$      the integer count of target superpixels
    $O$      the output filename
Output
    $img$    saved output image file
procedure HOMEWORK4($F, K, O$)
    Initialize MPI

    read image
    convert RGB to Lab format

    Master: prepare the l, a, and b channel's data for the algorithm
    MPI_Bcast the data
    Master: getLABXYSeeds
    MPI_Bcast the kseeds

    for all iteration do
        MPI_Barrier                              ▷ All the nodes should be ready HERE
        if Workers then
            do parallel work, and send Klabels and the distances to Master node
        else                                     ▷ Master node doing work
            reset distance vector to the max value
            for all Worker nodes do
                receive klabels and distances
                save the klabels has lowest distance

                recalculate the centroid
            end for
        end if
        MPI_Barrier                              ▷ Make sure all the nodes has done the work
        broadcast kseeds to all the nodes
    end for

    Master: DrawContoursAroundSegments()         ▷ Draw the contour boundary
    Master: save the image

    MPI_Finalize()
end procedure
```

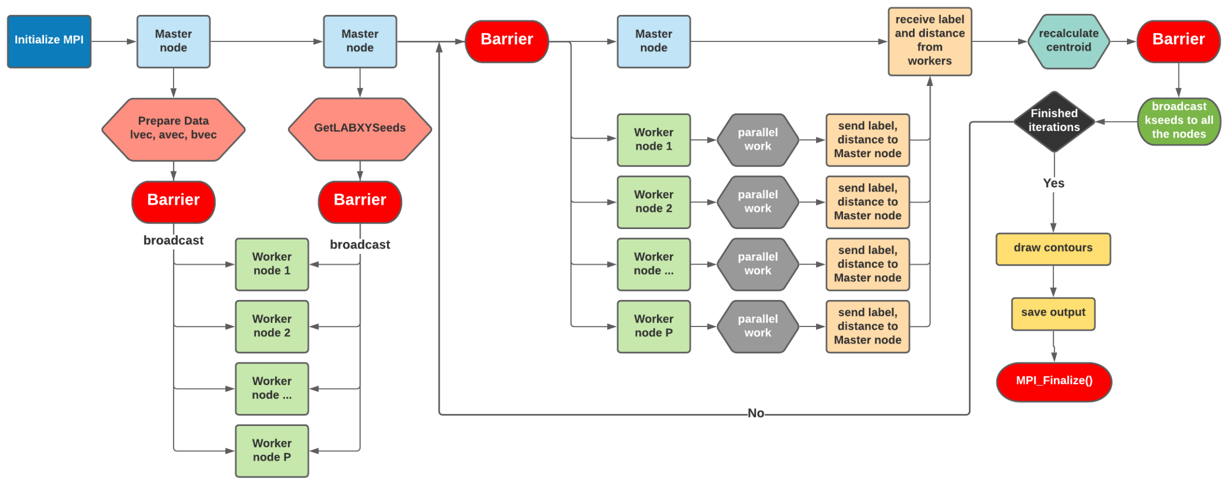


Figure 2: Parallelization Pipeline - Initialize MPI, synchronize data, Master node receive the data from workers, then recalculate centroid

3 Problems & Solutions

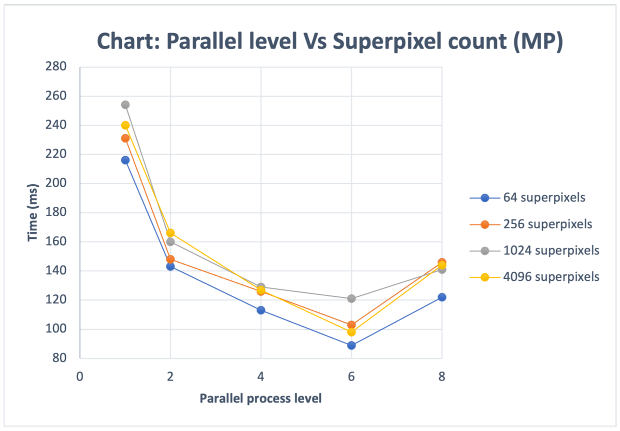
I mixed master's work and workers' work together, only assign the work for master node. The worker nodes are doing all the work left in the iterations, basically, computing distance. However, this way, master node will do the workers' work as well, which is not clear to manage them. Therefore, I separate the Master node from workers' job. Master node will only do the summarize the results from each worker and recalculate the centroid. After this change, the code in the iterations will be much clear.

When the workers send the data to the master, I only sent the labels. If the label is -1, then this pixel is not include to the worker, but the in the result, each superpixel is like a square. The picture is not segmented. I checked the label for each pixel. The label numbers are messed up. After I double check the part that computing the distance. It will continuously update the label with lower distance compare to other superpixels. This reminded me that I have to also pass the distance to the master node, because there is no connection between any of the nodes. In the master node, receive both labels result and distance result. Update the label with lowest distance.

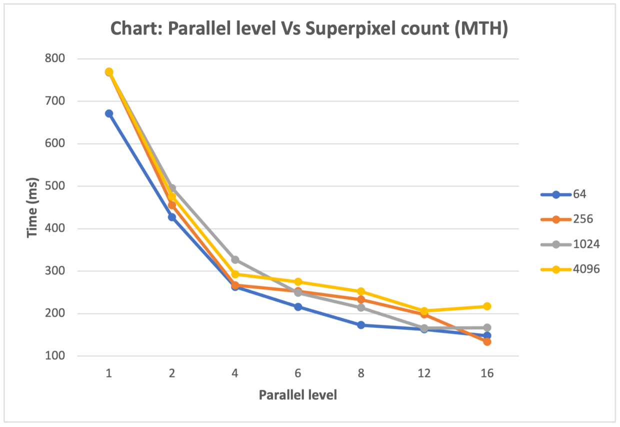
4 Conclusion

In the Fig.3, there are the results from three different parallel programming methods. In the Fig.3c, it shows the comparison with different number of superpixel and different parallel level. We can found that usually, larger data cause longer time. Based on the figure, when parallel level at 4 or 6, the time that used is lowest. The time data in the figure is not beautiful because I am running the test cases on Lewis Server, which causes the result is not that accurate. Sometimes, the run-time is about 1000ms, but it would go back to 400ms when run it again.

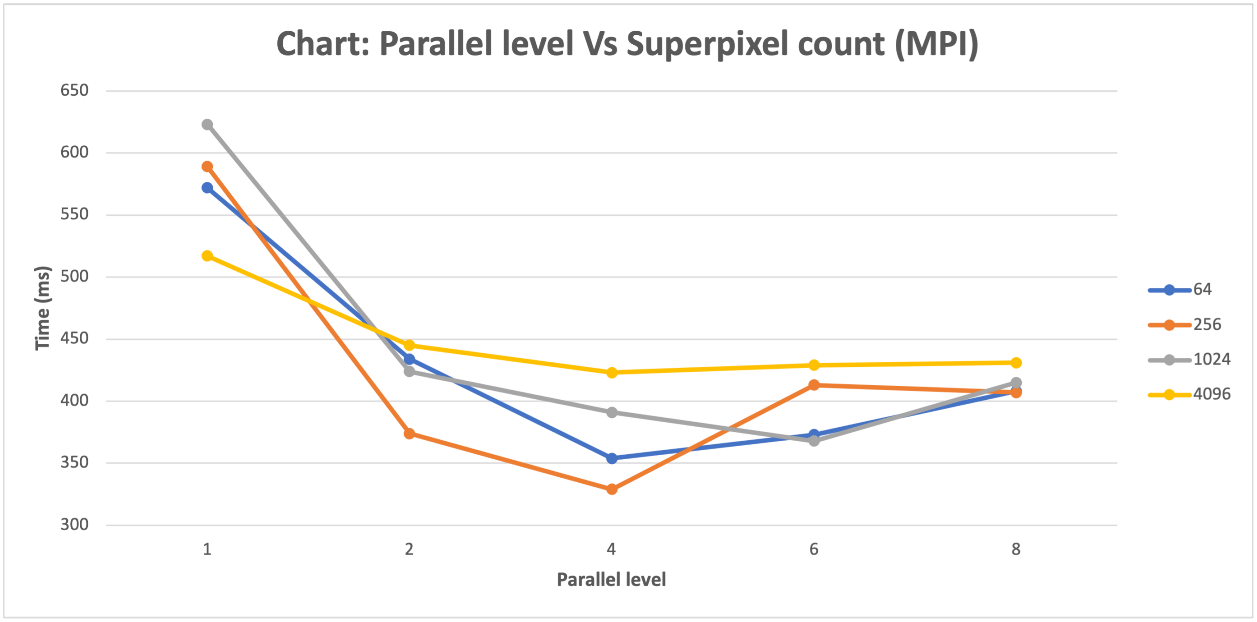
Compare these three methods, first two, MP and MTH, they were running in the same machine, I guess there is nothing to affect the running time. MPI is using multiple nodes in a server or different computers in the local host network. The message passing or the different power of the machine may cause the result sometimes faster and sometimes slower. Anyways, MPI still following the Amdahl's Law.



(a) Multi-Processing



(b) Multi-threading



(c) OpenMPI (Message Passing Interface)

Figure 3: Compare three parallel methods from HW2, HW3, and HW4

5 Lesson Learned

After this assignment, I am clear about how does OpenMPI work. Based on the results with different parallel level, it clearly shows the effect that speed up the processing time.

After initialize MPI, all of the nodes will do the same thing in the main function. Therefore, for some parts that are only need one node to execute, I will ask the master node to do it and broadcast the data to all the nodes. Also, it's better to set a barrier after broadcasting in case that workers are starting the work that needs updated data from broadcasting. Because the worker's job may only need to small part of work, the master node should do all of the thing else to avoid the all of the nodes do the same thing, which may cause longer to finish the work.

There is no shared memory or local variables that nodes could communicate with each other. MPI has function send and receive to pass the message. When use these functions, it is better to use array as a container but not a vector. When use send function, **MPI_Send()**, it needs the data, the size of the data, the type of the data, the destination, the tag which let other nodes receive the right message, and the communicator. Same, for receive function, **MPI_Recv**, it needs almost everything like send function besides use the source instead of the destination to receive the message from the specific node, and the status to save some information through this message passsing.

Compare to multi-processing and multi-threading, MPI is using the way like socket which is used to send and receive the message from other device. All of the nodes will be terminated by function **MPI_Finalize()**.

6 Useful Links

- MPI Tutorial, see: `https://mpitutorial.com/`
- MPI functions explanation, see: `https://www.imsc.res.in/~kabru/parapp/mpi2.pdf`