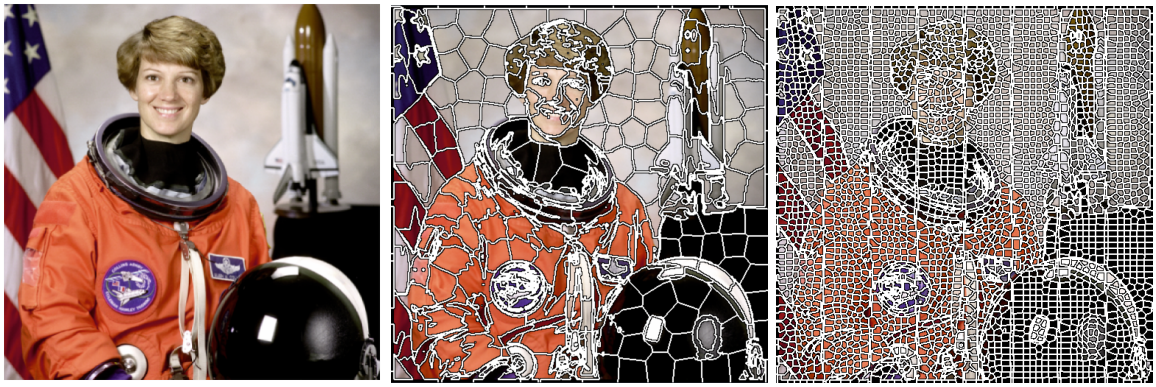
CS 7080 High Performance Computing
Homework 3
Multi-threading parallel programming

Haoxiang Zhang
PowPrint: hzny2

Due: October 10, 2021

1 Homework Description

Homework 3 did similar with homework 2. In homework 3, we need to use multi-threading instead of multi-processing 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 boost threads API.



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

Figure 1: SLIC Superpixels Algorithm Example

2 Homework Details

For homework 3, 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 boost thread to implement the parallel part. When do multi-threading, all of the variables can be accessed by every thread. In this assignment, all of the kseeds, distances, and labels will be passed and accessed by pointers. Because we need to pass the vectors and other variables to the threads. I made a threader class and pass the variables via object.

In Algorithm 1, it shows the details about the whole program. In the PerformSuperpixel() function, the code about calculate distances is using multi-nested-loop, which cost a lot extra time. Therefore I decided to parallelize this part with threads. However, when finish the distance calculation part, it should goes to recalculate centroid part, then go to next iteration. I made a barrier to block all the threads before centroid recalculation part using $join_all()$.

In Algorithm 2, it is a threader class which is doing the each thread's work. Because kseeds, distcec, klabels, and etc. will be used in each thread, I pass them to the object with pointers for updating the vectors and other variables directly. In the operator (a functor), I passed a thread ID for controlling each thread's work. From superpixel 0 to $numk$, p threads will alternatively go through them. For example, these superpixels will be taken from thread 0, 1, 2, ..., p, 0, 1, 2, and etc..

Algorithm 1 SLIC SuperPixels with Multi-Threading Parallelization

Input
  F the input file path
  P the integer level of parallelism
  K the integer count of target superpixels
  O the output filename
Output
  img saved output image file
procedure HOMEWORK3(F, P, K, O)
  read image
  convert RGB to Lab format
  getLABXYSeeds

  create thread object T ▷ The thread object is from a Threader class (Alg.2)
  declare a thread group

  for all *iteration* **do**
    reset distance vector to the max value of double type
    for all p threads **do**
      create a new thread t referenced by T, with given thread id.
      add thread t to the thread group
    end for
    join all threads ▷ Make sure all the threads has done the work
    recalculate the centroid
  end for

  DrawContoursAroundSegments() ▷ Draw the contour boundary
  save the image
end procedure

Algorithm 2 DistanceThreader Class

function DISTANCETHREADER(*kseeds*, *distvec*, *klabels*, *lvec*, *avec*, *bvec*, $offset$, *invwt*, *width*, *height*, *p*, *numk*)
    assign the parameters to member private variables
    declare a boost::mutex()
end function

function $\sim$ *DistanceThreader*
    delete mutex
end function

function OPERATOR(int myIdx)
    for all $n \rightarrow 0 - numk$ **do**
        if $n\%p == myIdx$ **then**
            calculate distances with n^{th} superpixel
        end if
    end for
end function

Every time called *join* function, the thread will be terminated. Therefore, I add threads in each iteration. The UML class diagram in **Fig.2** shows the detail member variables and functions in the *DistanceThreader* Class.

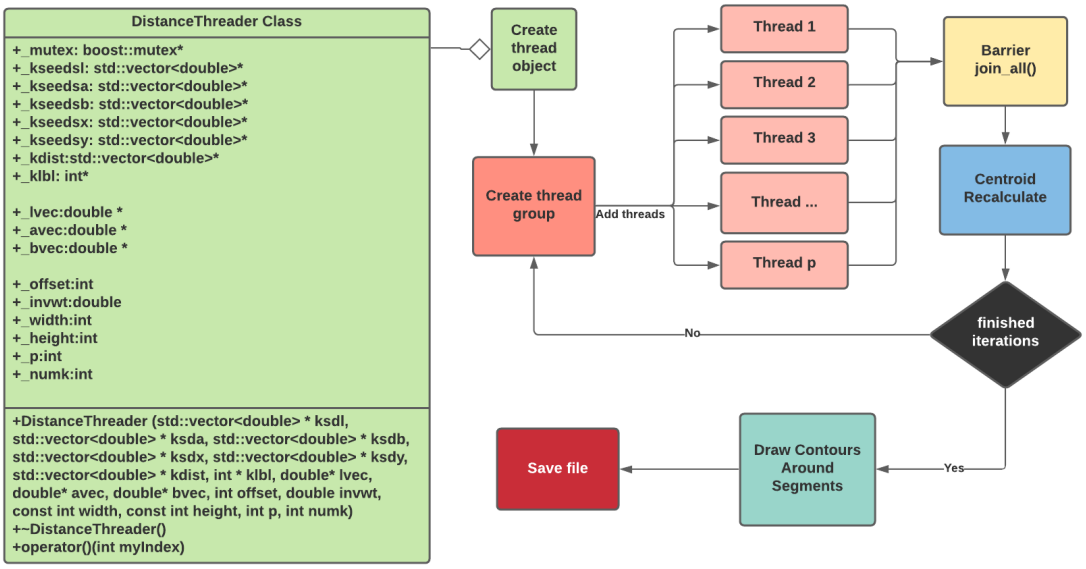


Figure 2: Parallelization Pipeline - create a thread object, add threads in each iteration. Join all threads before do centroid calculation

3 Problems & Solutions

I tried to pass parameters to the threads in *add_thread*() with the thread, but it not allows me to add more than three parameters. Then, based on the given multi-threading example. I found I can make a class and make an object to passing all the required parameters to the threads.

I found in the Threader class object, there is no way to let each thread know their own id. I looked up the *boost* :: *thread* document. I found I can pass one variable to the thread, and the operator function could receive the variable.

4 Conclusion

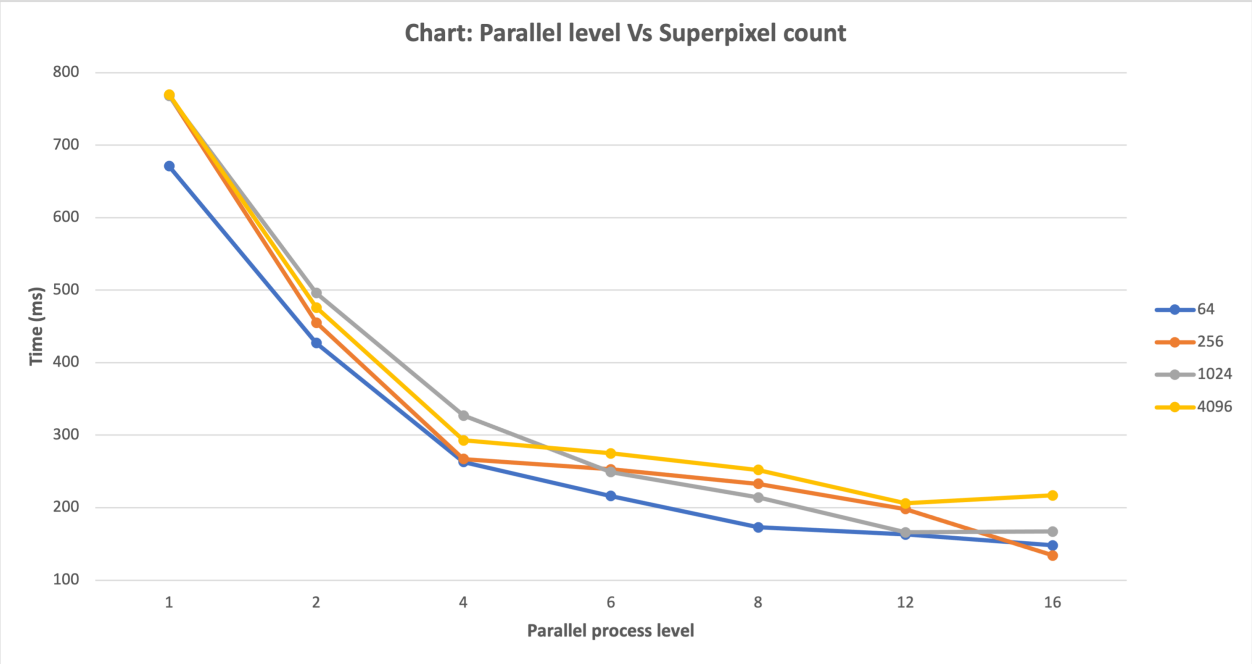


Figure 3: Caption

Fig.3 shows the comparison with different number of superpixel and different parallel level. We can easily find that with more paralleled processes the speed is faster, because the distance and labeling part is paralleled which saves a lot of time. We can found that the accelerate decreased after four threads paralleling. And for 16 threading, it may cost more time than 12 threading. The reason I guess is because that the core of the computer may not have 16 cores, which cause that the threads that has no resources to do the work will wait other threads release the resource then start to work.

5 Lesson Learned

After this assignment, I am clear about how to setup multi-threading and how to create a threader class. Based on the results with different parallel level, it clearly shows the effect that speed up the processing time.

When do multi-threading, threads can access to variables. Therefore, we need to pass all the variables to the threads as pointers. The threader class could take all the pointers from the main function. Based on this part, I have learned how to use pointer appropriately. I can pass the references of each variable to the threader object. In the object, I can declare the pointers to store the references from main function. Therefore, we don't have to worry about the data would not update to the variables. Using pointers is much faster than just copying the data to the function, which could waste a lot of memory and slow down.

Compare to multi-processing, multi-threading is more easier to set up. Every time the thread will be terminated after it finish the work. And using a barrier to block them all. The threads will back to work again once adding them back to thread group. ‘