Brendan Pho
Brock McNerney
Nick Albright
Wesley Llamas

BubbleSort - Mid-Project Deliverable Discussion

SOURCE: https://github.com/nickalbright97/BubbleSort
Note: Everything in the repo is our mid-project deliverable

What have you accomplished?

Image import and array construction:

We were able to find out the structure of TIFF files/stacks of TIFF files. We are currently able to retrieve the dimensions of TIFF files in main.c, and are able to traverse through TIFF "directories." The directories represent the stack of TIFF images we are trying to analyze. Furthermore, after implementing directory traversal for TIFF files, we were also able to read in TIFF images and retrieve the grayscale values of each pixel. We were able to store those pixels in an array which will be used for calculations. We've also created a utility method for array access, so our distance algorithms can easily address the array. Main.c is also able to output other meaningful information such as the current directory being processed and the number of directories (image slices) in a TIFF stack. In addition, while import.c builds on the functionality of main.c and allows TIFF header reading to determine the properties of the file, such as little/big endian byte ordering.

Calculations/Other Project Aspects:

We currently have a C serial version of the IDL code that calculates distance. The distance calculation portion of IDL code uses the euclidean distance algorithm and takes about an hour to run. The serial version we created uses a worker queue and a reverse flood fill algorithm and takes about 10 seconds to run.

What discoveries have you made?

Image import and array construction:

We were able to find out the TIFF file structure. We also discovered that image import and array construction will not need to be parallelized, since the serial versions execute almost instantaneously.

Calculations/Other Project Aspects:

Originally we were working with an inefficient algorithm, searching each individual element, then searching each individual element again. We parallelized this using OpenMP and saw almost linear speedup (with a 50x50x50 image, the serial version took 30 seconds and the parallel version took as low as 4 seconds descending from 2-64 threads), but our inefficient algorithm was just "embarrassingly" parallel and the program didn't see nearly enough speedup to reach our goal of a 512x512x512 image running in under 1 hour.

After realizing that parallelizing the inefficient algorithm wasn't giving us nearly the speedup we required, we moved to attempting to use CUDA code in order to have the program run on a GPU. We were able to accomplish this and it ran faster than the parallel version, but still was not fast enough to reach our goal (a 256x256x256 image took 27 minutes).

The biggest discovery that we came across was changing the inefficient algorithm (searching each individual element, then searching each individual element again) to a reverse flood-fill algorithm using a worker queue. This change in code saw tremendous speedup in the serial version, 64x64x64 took 124s with inefficient and 64x64x64 takes less than 1s with new algorithm implementation.

What roadblocks have you hit?

We hit many of the roadblocks that we previously stated in the original project proposal, as well as some other unforeseen ones.

Image import and array construction:

We first ran into the roadblock of the difficulty of converting IDL code into C. We were able to convert much of the original IDL code. We also ran into the issue of figuring out which external libraries we would have to use in order to implement image analysis. We were able to remedy this problem and implement file I/O for TIFF files after requesting Libtiff/tiffio library to be added to the cluster.

Calculations/Other Project Aspects:

The first roadblock we hit was with serializing the IDL code. The IDL was able to search in a radius with a library that is not available in C, so we needed to create our own algorithm. The algorithm we began with was an inefficient algorithm that used our c version of the euclidean

algorithm and our firsts tests showed that getting the distances from a 512x512x512 image was not feasible (would take upwards of 24 hours).

We continued along with the inefficient solution while trying to parallelize to reach our speedup goal, and while we did see good speedup, it wasn't close to our goal. In the end, we ended up revising the serial code using the reverse flood fill algorithm and a worker queue. While this shows tremendous speedup and is well within our goal, it will not be easily parallelizable like the previous algorithm because it does a lot of queueing and dequeuing with the worker queue, creating a lot of critical sections.

What do you still have to do?

Image import and array construction:

We still have to implement noise reduction for the TIFF images. We also have to use the queue algorithm on the actual image using actual pixel values by combining both parts of the code.

Calculations/Other Project Aspects:

Some of the serial code is still a little messy and repetitive due to a lot of testing and just figuring it out. The serial code needs cleaned up and documented. Then, we still need to attempt to parallelize the distance algorithm code which will be difficult (we've decided to first try pthreads and use the distance function as a worker function). We can try different alternatives until we see speedup from the serial version.

Are you on track to finish the project by the final due date? If not, what do you plan to do to get back on track?

We are currently on track to finish the project by the final due date. Figuring out how to implement noise reduction will be a tricky and challenging task but we believe that we are currently in a great position to finish on time and to make any necessary revisions by the deadline.