Image Morphology

Sullivan Lauderdale
Electrical and Computer Engineering Department
School of Engineering and Computer Science
Oakland University, Rochester, MI
e-mail: slauderdale@oakland.edu

***Abstract*—This project will be focused around the performing of image morphology of a grayscale image. The scope will include erosion, dilation, opening, closing, or boundary extraction. The operation will be selected by the user, and the operation will be performed using Thread Building blocks as well sequential implementation. These will both produce timed, and the different results will be displayed in the terminal window.**

I. INTRODUCTION

This project will cover a broad range of image morphology operations, and it will show the varying effects of TBB and sequential implementation on the timing of an operation. The project will seek to fully display the benefits of using TBB for larger operations and will seek to point out the discrepancy between TBB and sequential implementation. It will also be a challenge to attempt to implement the multiple morphology operations to a large-scale image in an effective manner.

II. METHODOLOGY

The methodology will cover the separate operations to be covered in the project. Figure 1 shows the full software flow.

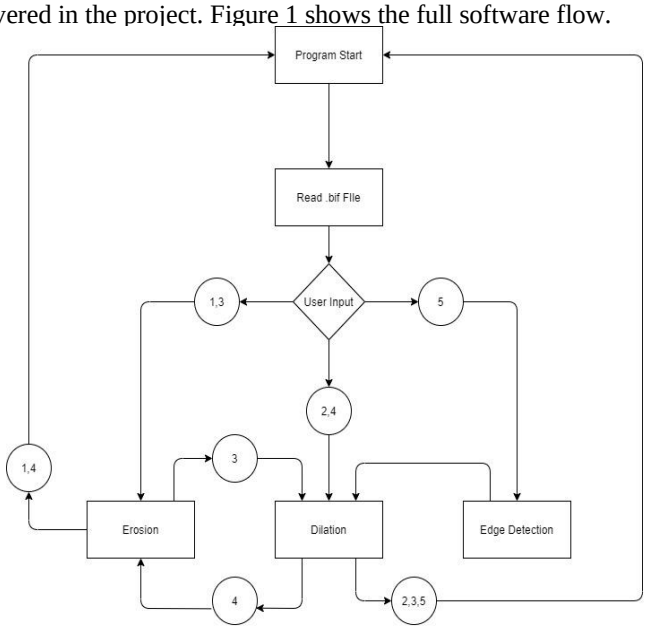


Figure 1

This project also includes a universal implementation. To run the code, the user simply needs to name the input binary file "input.bif" and place it in the same folder as the code. Then when the user runs the code, they can type "./morpho 1 940 602". The morpho is used to run the code, the one selects the operation, and the final two numbers are the image dimensions. The functional flow followed by the project is shown below in figure 2.

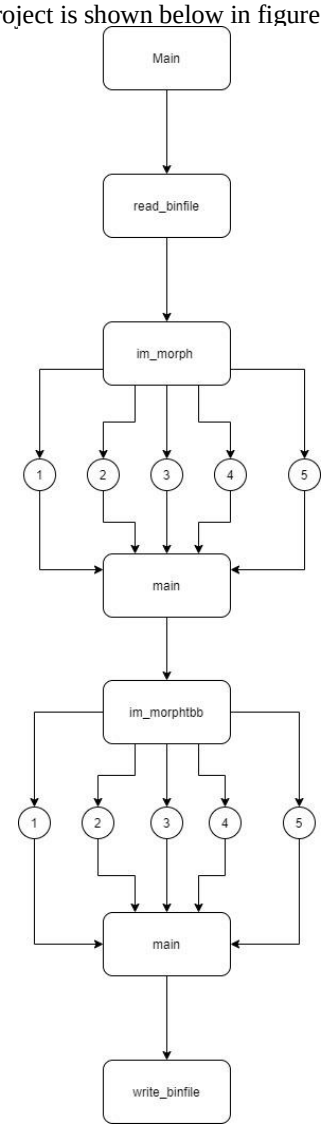


Figure 2

This figure shows the functional process that will be taken by the project. The numbers 1-5 indicate the operation specified by the user, and they will each follow a slightly different path. The path of the specific numbers is shown in figure one but was not repeated to make figure 2 more readable. The parallelization strategy in this project was the use of nested parallel for loops to perform maximum and minimum operations in a parallel manner. The convolution was also computed in parallel, and this strategy used for the functions was able to fully demonstrate the benefits of using TBB's.

A. Dilation

Dilation is the first operation that will be performed in the project. It will include TBB and sequential implementation. Dilation will use a disc of radius 2 to minimize values. To select dilation as the morphological operation, the user will need to enter 1 as their operator. Dilation seeks to increase the boundaries and contrast displayed in the image as it applies the maximum values inside the matrix. The parallel_for function will be used to send the matrices, and to determine minimum value. It will seek grow the boundaries of the image where the erosion shrinks them. It uses a kernel like the erosion above does, and it will once again be applied to a binary image, and it will produce a binary output file that can be viewed using MATLAB or octave. [2]

B. Erosion

Erosion is applied to binary images, and it will seek to erode the boundaries of an image. This operation is performed with a kernel or structuring element that is applied to the binary input. The image kernel used for this case is a disc with a radius 2. When this kernel is applied, the maximum number from the kernel is applied to the center circled number. The erosion will be performed with the use of nested parallel for loops. The parallel for loops will pass the rows and columns of the kernel into the maximum function. Lastly, this maximum number will be applied into the center of the kernel. The image in this case should vary between 0-255 for the morphology operation to be used. The images throughout the entire project will be binary input files for effective operations[1].

C. Opening

Opening is the third morphology operation that will be applied within the scope of this project. It will once again be applied to a binary image with TBB and sequential implementation. Opening will see to preserve the pieces of an image that are like the structuring element. It is a similar operation to erosion, but it does not have as many reducing properties as the erosion operation. The opening operation will perform an erosion and dilation, but it will use the same structuring element for both operations to perform the operation. [3] Figure 2 shown below shows the flow of the opening operation.

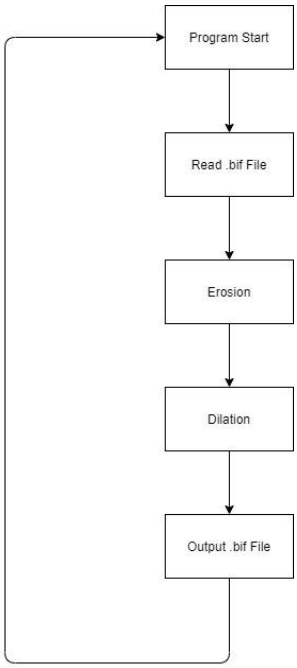


Figure 4

D. Closing

Closing is the fourth morphology that will be applied, and it inverse of the opening operation. Same as the opening operation, closing will perform dilation and erosion with the same kernel, but in this case the dilation is performed first followed by the erosion operation. This setup for opening and closing is why the erosion and dilatation operations will need to be designed and coded primarily. Once those operations are designed and verified, it will be possible to implement the opening and closing operations [4]. Figure 3 shown below shows the program flow of the closing operation.

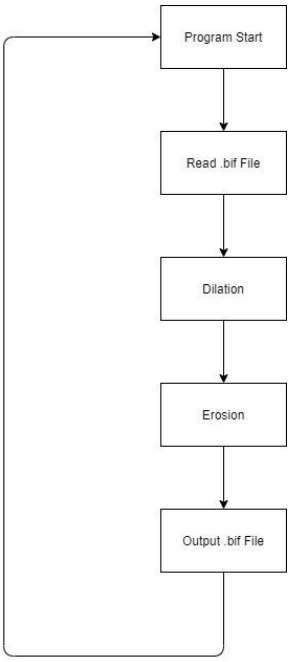


Figure 5

E. Boundary Extraction

Boundary extraction will be the final and most difficult operation to perform. It will first involve the use of edge detection to determine the edges within the image. Next, it will involve the use of dilation to fully display the boundary that surrounds the selected area. The edge detection once again will require the use of a kernel, and the dilation operation will use a separate kernel and implement the same operation discussed in section B. Figure 4 shown below shoes the boundary extraction operation

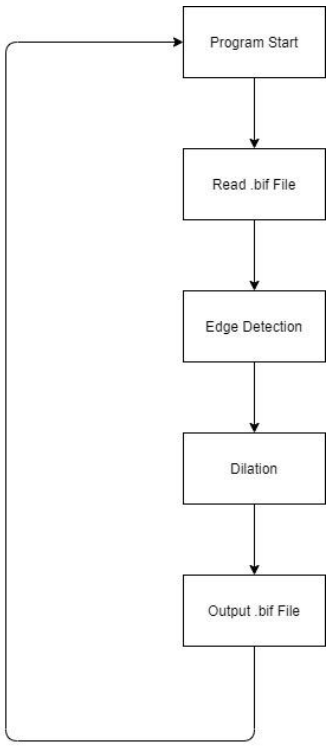


Figure 6

While the dilation disc remains the same as the one used in all the previous operations. In this case, the kernel used for convolution is separate. The kernel is shown below in figure 6.

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

Figure 6

This kernel was hard coded into the main, and it was used for all image sizes.

III. Experimental Setup and Results

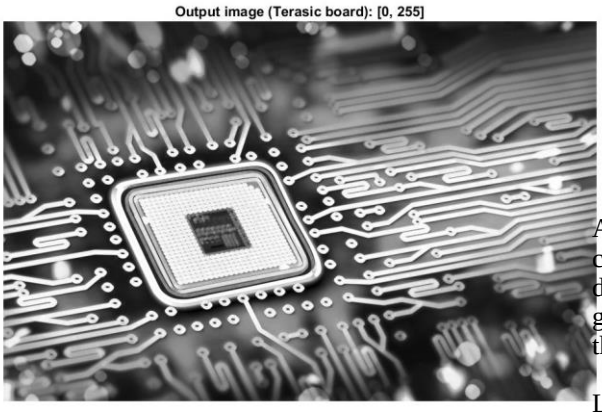
A. Experimental Setup

The experimental setup will be crucial in the design process to ensure that the experiment and program functions effectively. To test the project, I used MATLAB to run the same operations, and then compare my results. This will allow for the project to be tested to ensure that it was successful. I will also implement this experiment in stages. Because the erosion and dilation operations are crucial to the entirety of the project, I first implemented and tested these functions. Once these functions could run successfully, I could then easily move into the rest of the functions. Also, for ease of testing, I used the file given from original TBB

notes section to test and display my code results. This was only a small portion, however, because I also was able to run varying image sizes for fully display the effects of using TBB.

i) Dilation

The first image morphology operation to be tested was dilation. This operation was able to be run on the DE2i-250 board, and it could then be tested with MATLAB. The dilated image is shown below in the image.



This image could then be measured against the MATLAB computation of the same operation. MATLAB was used to calculate the difference between the image generated by the Terasic board and an image generated by MATLAB. MATLAB has built in functions to calculate the dilation of an image, and this made it a reliable source to be measured again. When the dilation is performed successfully, the difference is shown as a black image.

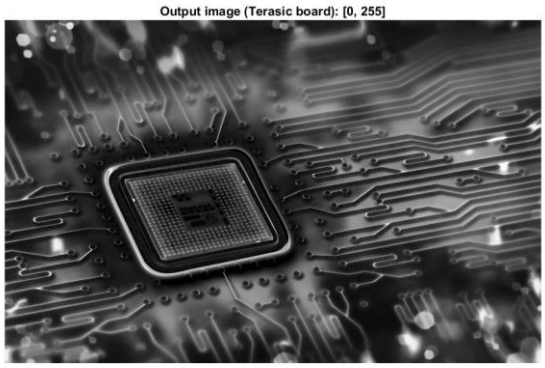
Lastly, the effects of implementing TBB is shown by the following table. This table shows the sequential time for various image sizes, followed by the TBB implementation.

Size	TBB	Sequential
640x480	65118.2	91901.3
840x486	80998.6	121091.2
940x602	108854.2	165749.9
1080x1080	215625	351569.5
3840x2160	1445532	2370123

This table fully shows the effect of implementing TBB. TBB's are able to save extensive amounts of time when implemented, especially as the images grown larger, and the operations grow more complex. Each of the values comes from the average of 10 test runs to ensure that the values are an accurate representation. The values in all the following table are also taken as the average of 10 runs so all the tables contain accurate values.

ii) Erosion

The second operation performed in the project was erosion. Erosion is the opposite method of dilation, and it was performed in the same way as dilation. To test this operation, the code was first run on the Terasic Board. The Terasic generated image is shown below.



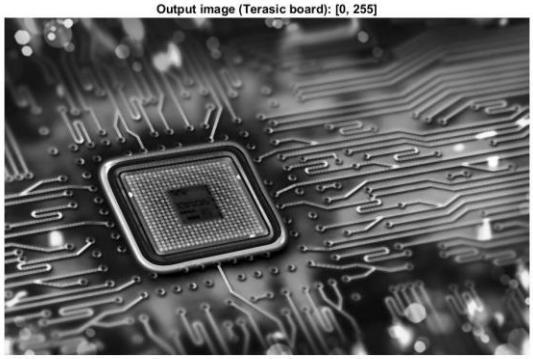
After this operation was performed on the Terasic board, it could then be tested through the use of MATLAB. The difference between the MATLAB image and the image generated by the board were calculated as a difference, and the image was black meaning there was no difference.

Lastly, the time data was measured to show the benefit of using TBB's. the data is shown in the below table, and the advantage of TBB's is clearly known.

Size	TBB	Sequential
640x480	62668.7	90263.2
840x486	79711.9	115135.4
940x602	106837.9	161462.8
1080x1080	215925.5	332736.7
3840x2160	1414877	2320940

iii) Opening

The next operation performed was opening. Opening was performed by performing the erosion operation followed by the dilation operation. The opening operation as performed on the Terasic board is shown below.



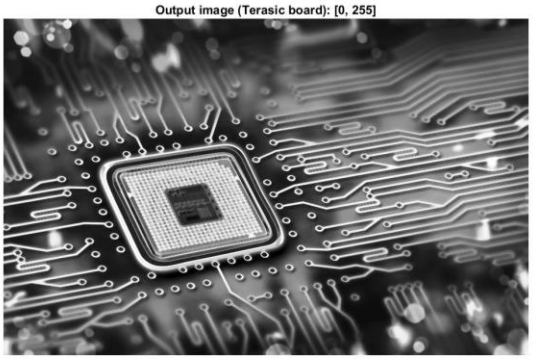
MATLAB also contains a function to calculate the opening image. Once again, the MATLAB generated image could be used to check against the Terassic board image, and it returned a black image reporting a successful program.

This operation once again aligns with the desired result. The operation was also performed while being timed to shown the effects of the TBB's verses the sequential implementation. This operation also took about double the time as the previous two operations which makes sense because it performs both erosion and dilation.

Size	TBB	Sequential
640x480	128707.5	174647.3
840x486	159361	225240.2
940x602	217368.8	316735.8
1080x1080	447179.8	694088.8
3840x2160	2945601	4702000

iv) Closing

The next operation performed is closing. Closing is performed in the opposite manner as it is dilation followed by erosion. The Terasic board implementation is shown below.



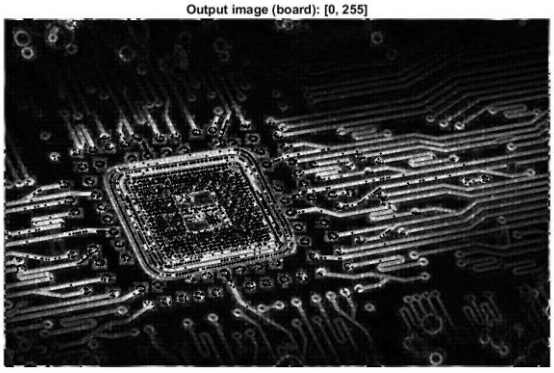
This operation was then measured against the MATLAB performed operation. This MATLAB operation once again returned the desired black image to show that the program calculations where successful.

The calculation time of the operations was measured to show the benefit of the Thread Building Blocks. This table once again clearly shows the full benefit of using thread building implementation in programming.

Size	TBB Sequential	TBB
640x480	122573.5	170861.3
840x486	166873.1	230879.9
940x602	214482.5	319985.4
1080x1080	431267.7	648047
3840x2160	2864193	4635286

v) Boundary Extraction

The final operation to be performed was boundary extraction. This operation is performed by first performing the convolution of an image with a kernel followed by the dilation of the resultant convoluted image. This was the most difficult operation to perform because the convolution caused thread race issues. However, moving the thread operations outside the function made it possible to run the operation as a thread safe operation. The image generated from the Terasic board is shown below.



This image was generated, and it needed to be tested against a MATLAB generated image to ensure a proper operation. The MATLAB code for this final portion was also more difficult to perform. To generate the proper file to compare with the Terasic image, MATLAB also had to perform the convolution followed by the dilation. If boundary extraction was run in MATLAB it would not return the same result because the formula for boundary extraction is different. Therefore, the image was tested against the same 2 operations, and once this was successfully implemented, the desired black image was the result.

This operation was once again compared against the sequential implementation and the implementation using thread building blocks. The thread building blocks once again proved to be much faster especially when the image files grew larger.

Size	TBB	Sequential
640x480	101327.7	137012.7
840x486	128350.3	184715.5
940x602	172444.7	244315.4

1080x1080	346436.1	520706.8
3840x2160	2241965	3557525

CONCLUSIONS

This project was designed to fully display the benefits of implementing thread building blocks in complicated operations. Although the operation was successful, and the thread building blocks proved to run faster in every instance, there are still some improvements that could be made.

The first improvement that could be made would be the implementation of the parallel reduce function along with the parallel for operation. This would prove to optimize the program slightly more, although the benefits of the TBB are still fully shown in the current implementation.

The next implementation that could be used to improve the program would be the separation of sequential and TBB operations. In the current operation, the program is set to perform the sequential operation followed by the TBB operation. This led to easier testing and was effective for the demonstration required in the project. However, if this program were going to be used in a stand-alone environment, it would be beneficial to separate operations and give the user the chance to choose if they would like to run the sequential or the TBB operation. In the more complex operations, the program begins to take longer to run because it had to make it through both operations. If they were performed separately, they would prove more effective if only one operation was needed.

The most difficult portion of this project was ensuring that the operations remained thread safe. In the beginning of the project, that is what led to the most errors within the project, but these were best fixed through the use or parallel_for outside of the function. This meant that the rows were sent separately, and the correct output could be calculated effectively.

The project designed for this course was programmed to show the full effect of TBB in reducing operating times. The tables shown in the report fully demonstrate the benefit that is achieved using TBB. It was effective to show that in a practical environment TBB would be ideally used in all cases to provide full benefits to operating speed.

.

REFERENCES

[1] https://homepages.inf.ed.ac.uk/rbf/HIPR2/erode.htm

[2] https://homepages.inf.ed.ac.uk/rbf/HIPR2/dilate.htm

[3] https://homepages.inf.ed.ac.uk/rbf/HIPR2/open.htm

[4] https://homepages.inf.ed.ac.uk/rbf/HIPR2/close.htm