Final Project:

OptiSleep

Jerome Andaya

Jeffrey Lin

Khai Phan

OptiSleep: EC500 E1 Final Project Report

I. Problem Statement

Throughout the class, we have completed a handful of parallel programming to study heat equations and how the temperature inside a bound space slowly converges after enough time (in our case, iterations) have passed. However, if we consider this going from point A to point B, the problem our team seeks to address is attempting to go from point B to point A. Rather than using sources to calculate a final temperature for a bound space, the team seeks to calculate the required source temperatures upon a desired convergence value.

To relate this problem to an everyday scenario, let's discuss a common situation for a Boston University student living in South Campus. The student, Khai, has a broken heater, which is constantly generating an absurd amount of heat throughout the night. Additionally, the temperature outside is *freezing*. Now, if he opens one (or both) of his windows too widely, the temperature inside his room will decrease to an uncomfortable low over night; this increases the probability of him falling ill in the morning with a runny nose and/or cough. On the contrary, if he does not open his window(s) enough, the temperature in the room increases to an uncomfortable high, causing breathing difficulties.

Thus, we want to know: given the outside temperature and the heater temperature, how wide should the window(s) be open in order for the room to converge to a comfortable range of 68 to 72 degrees Fahrenheit. We plan to use GPUs (writing in Cuda) and Jacobi relaxation to perform our calculations.

II. Prior-Implementation Discussion

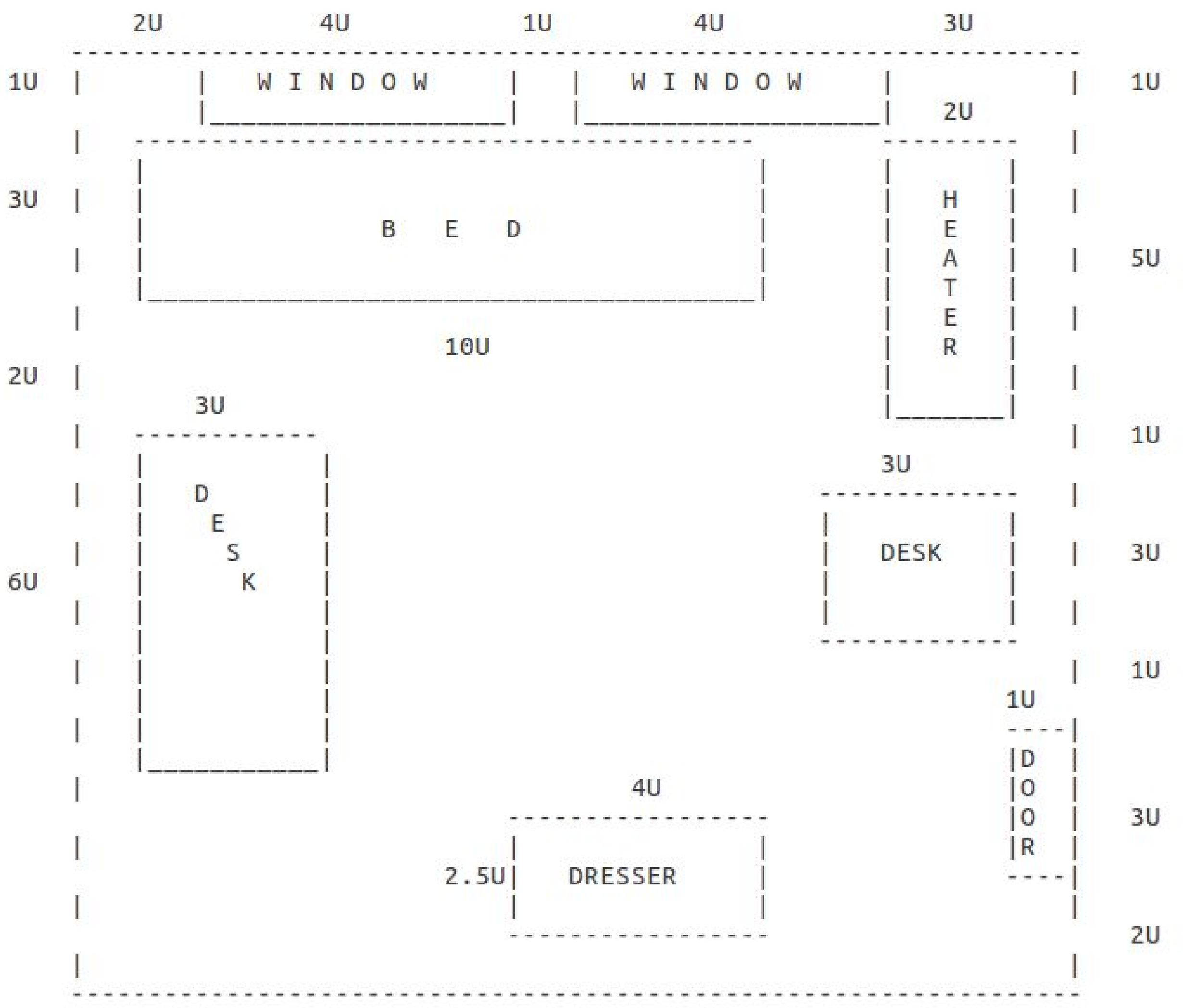
We suspect that we will face some of the following obstacles during design and implementation:

1. We lack computational power. Instead of going from point A to point B, in which there is only a need to compute the relaxation algorithm once, we will need to calculate a large sample pool of *point A to point B* problems in order to go from point B to point A, considering that this is very similar to a guess-and-check problem.
2. Designing a good model will be difficult. The real world has so many laws and behaviors occurring at any given time. It's not symmetrical. It's not 2-dimensional. Our model can't

oversimplify or else the results will be skewed; however, creating an accurate model may be out of reach, given the timeline of this project.

III. Design and Implementation

Firstly, we need to generate a model. Originally, our model's dimensions were 2048 by 2048 to more accurately capture the room; this translates to approximately 1/10th of a centimeter per square foot in my room. However, after attempting many preliminary calculations, these dimensions were not a viable choice given our calculation power; we needed to iterate through over 100 window combinations, with each combination having over 20000 iterations of relaxation. Thus, the final model's dimensions were lowered to 512 by 512. The room is modeled as follows:



In this diagram, U represents the unit, which is 32 array spaces. The objects are labeled appropriately.

After creating this model, our runtime drastically decreased. Since we lowered each axis's dimension by a factor of 4, the entire runtime was increased by a factor of 16, since each iteration of relaxation needed to only iterate through 1/16th the number of array spaces.

In addition to decreasing the dimensions of our model, we introduced some new design choices:

1. The heat sources cannot be relaxed; we created logic to ignore the heater and the windows during relaxation iterations.
2. The opened portion of the window is measured as a percentage (0 to 1). If the window is completely open, the temperature is treated as the outside temperature. If the window is completely closed, the temperature is treated as room temperature. The temperature of the range in-between is a linear relationship between these two values.
3. The walls are completely insulating. Thus, the walls also cannot be relaxed in order to preserve the temperature inside the room; they are initialized to 68 degrees Fahrenheit and remain constant for the duration of the experiment.
4. There is a drift parameter which increases as the windows are opened more widely (this is based on the window's percentage mentioned in #2). The drift parameter dictates how swiftly the temperature will relax from the wall with the windows to the opposite wall.

These rules would enhance our model: it makes sense that the heater never stops emitting heat and that the outside world's temperature cannot be affected by the temperature of the room.

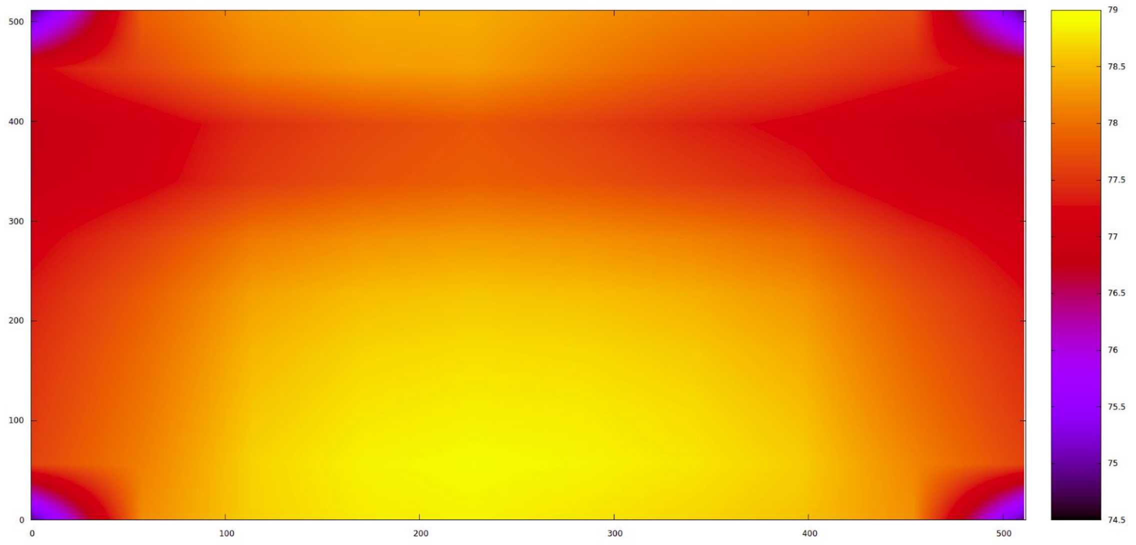
Secondly, we created the code and algorithm to run our relaxation. As stated before, we use Jacobi relaxation. We also introduce GPU computational power into our problem as the number of iterations are still very large despite the decrease in dimensions. Our design is to call Jacobi relaxation on a temperature grid, generated using window parameters. The average temperature around the upper region of the bed (where the face would be) is recorded once the temperature changes inside the room converges. This process is then iterated for each combination of the windows' opening size. For example, our program would run Jacobi relaxation for *window1=0.0, window2=0.0, window1=0.0, window2=0.1*, until *window1=1.0,* *window2=1.0*. If any of these combinations yielded the desired temperature range, the program returns the window parameters.

IV. Experiments and Discussion of Experiments

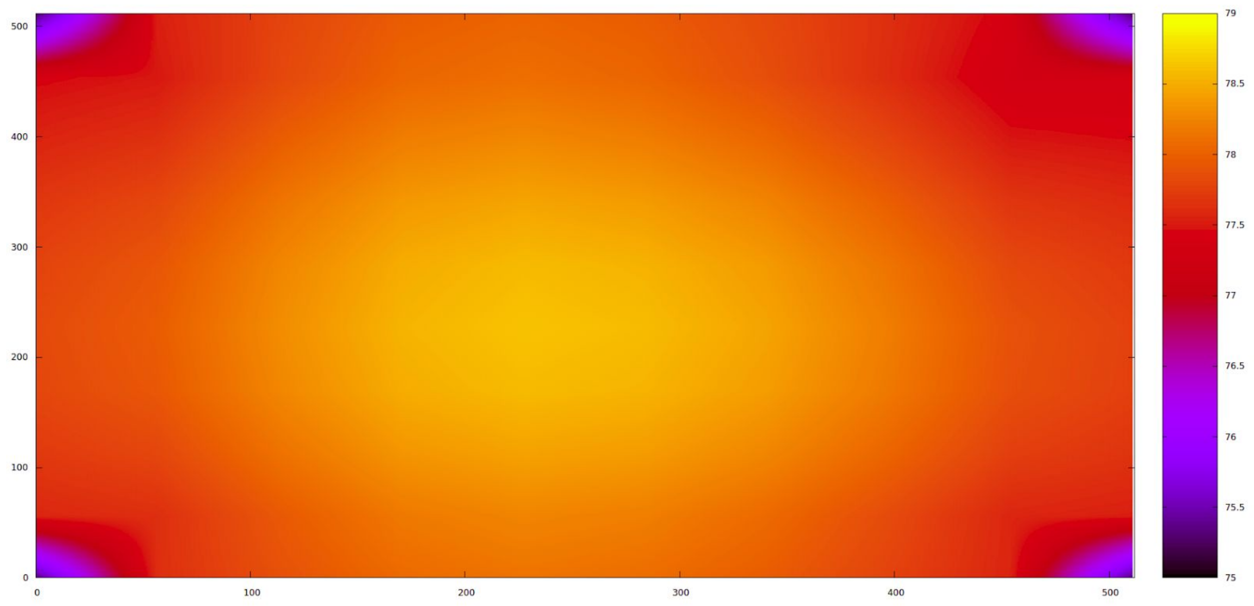
Our experiments consisted of running the program for different initialization values, different room temperature values, and then repeating with a window fixated at closed (a closed window has a value of 0).

During our experiments, we noticed that the room's convergence temperature was heavily influenced by our initial room temperature. For example, if we created a scenario where the windows were closed and the room temperature was heated to 85 degrees Fahrenheit, the upper region of the bed converges to approximately 78 degrees Fahrenheit. However, if we created a scenario where the windows were open and the room converged to 50 degrees Fahrenheit, the room would later converge to approximately 58 degrees Fahrenheit.

In another subset of experiments, in which we fixed values for both windows and attempted to relax it in order to visualize the output, we discovered that the drift parameter is quickly smoothing the temperature differences in the room. The initial room temperature for this experiment is set to 80 degrees Fahrenheit. An example of the room can be seen below:



One aspect to note about this diagram is the axes; the windows are located at the bottom of the diagram rather than the top. Observing the above diagram, at just 3000 iterations of Jacobi, the temperature of the room is already beginning to disperse evenly in a large portion of the room. The 12000 iteration mark can be observed below:

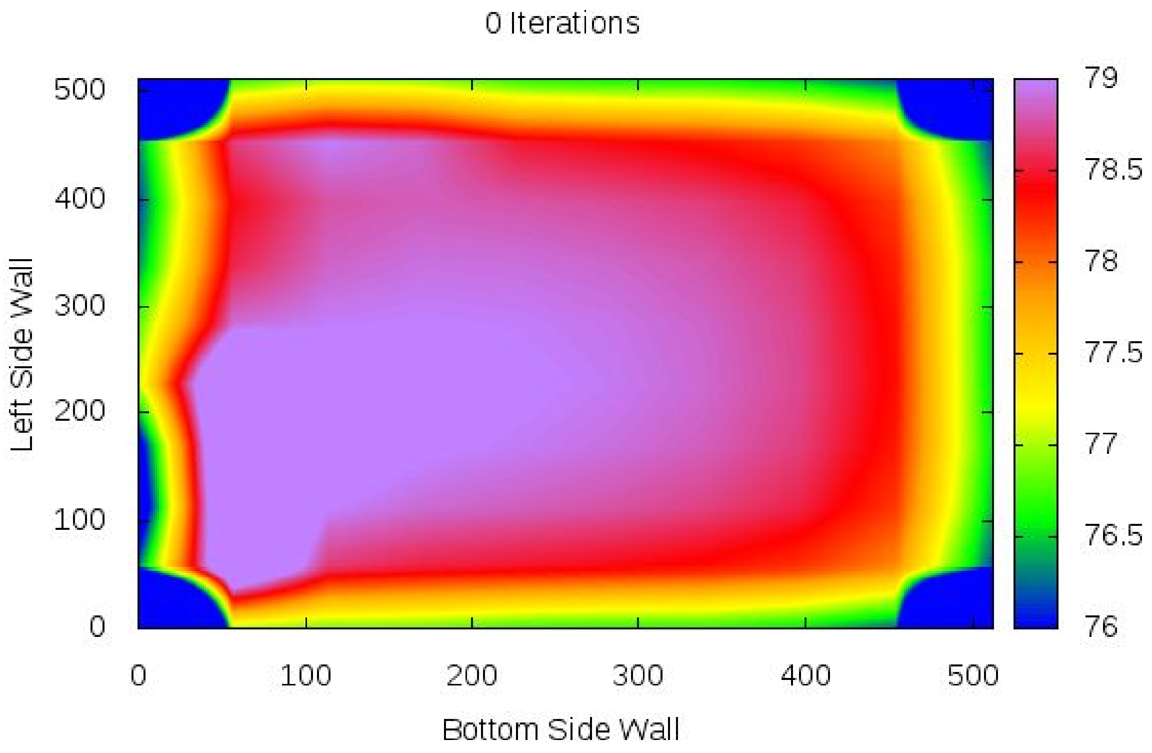


Just 9000 iterations later, the temperature inside the room is already mostly distributed at approximately 78 degrees Fahrenheit.

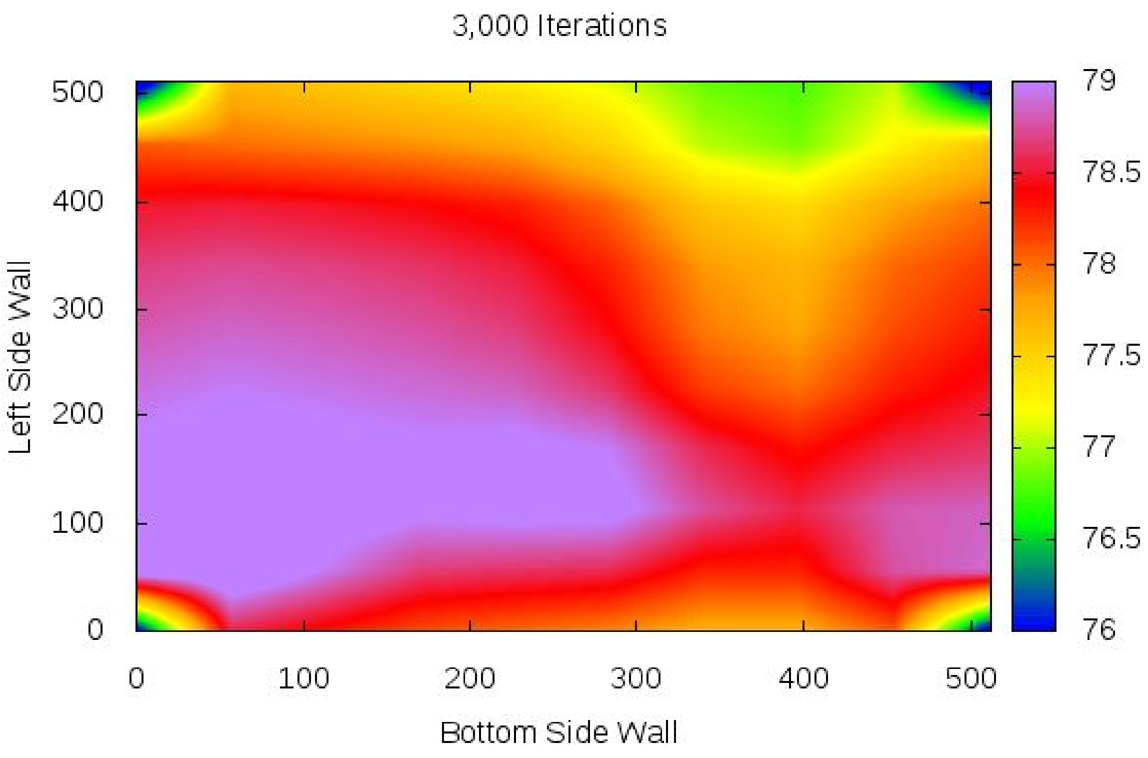
We suspect that this is because the temperatures inside the room greatly outweigh the effect of the windows and the heater. Because of this, even after relaxation, the impact from the initial temperature is still apparent. It makes sense that the room would eventually converge at a temperature near the average of all initial temperatures.

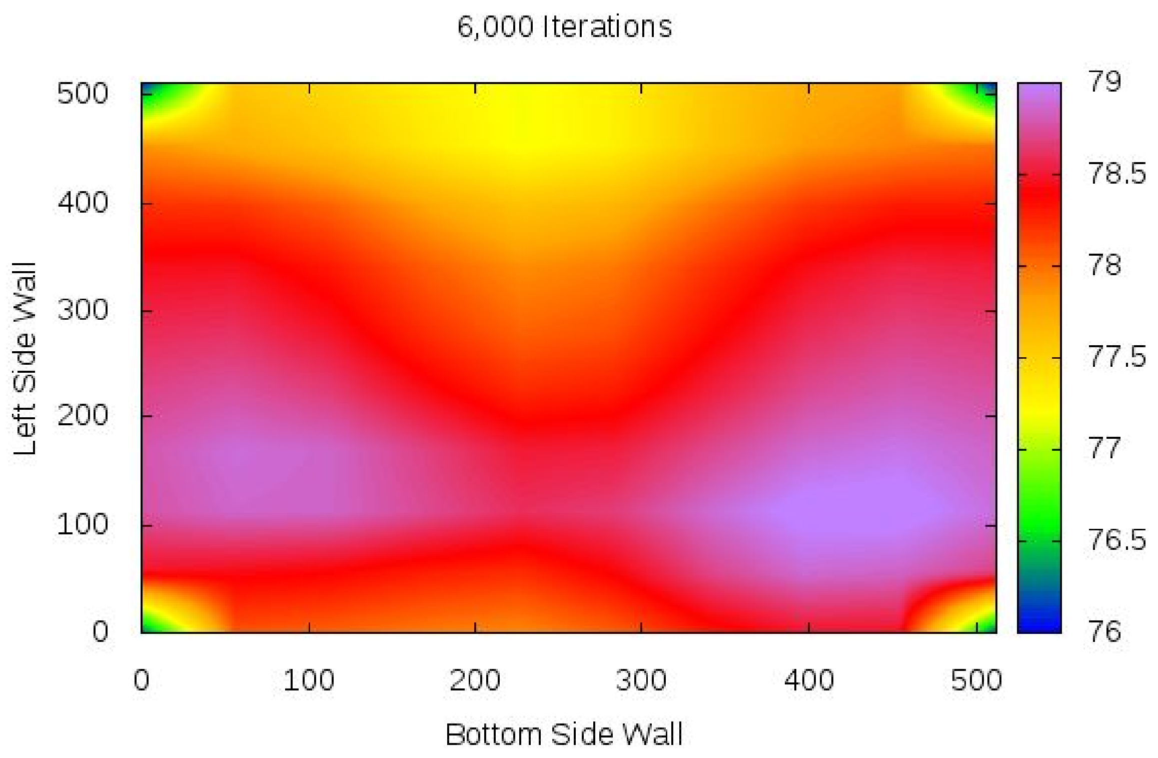
A second set of experiments were also conducted using a model that tried to include air flow. This model was created upon discovering that adjusting relaxation weights only changed conductivity, and not convection. Thus, we had a wind blow from one side of the room to the other; from a window out to a vent. This was modeled by copying blocks downward every 1000th iteration if the block was aligned vertically with the window. The 31st block (bottom block) was not touched. This models the air “disappearing” out through the vent.

This is our initial layout once again. Take note of the heater at the bottom left-hand side. Our windows can be found at the top of the heatmap. In general, this is a better model than the previous set of experiments. We see the cold enter the room far more quickly, and then disperse throughout the room slowly.



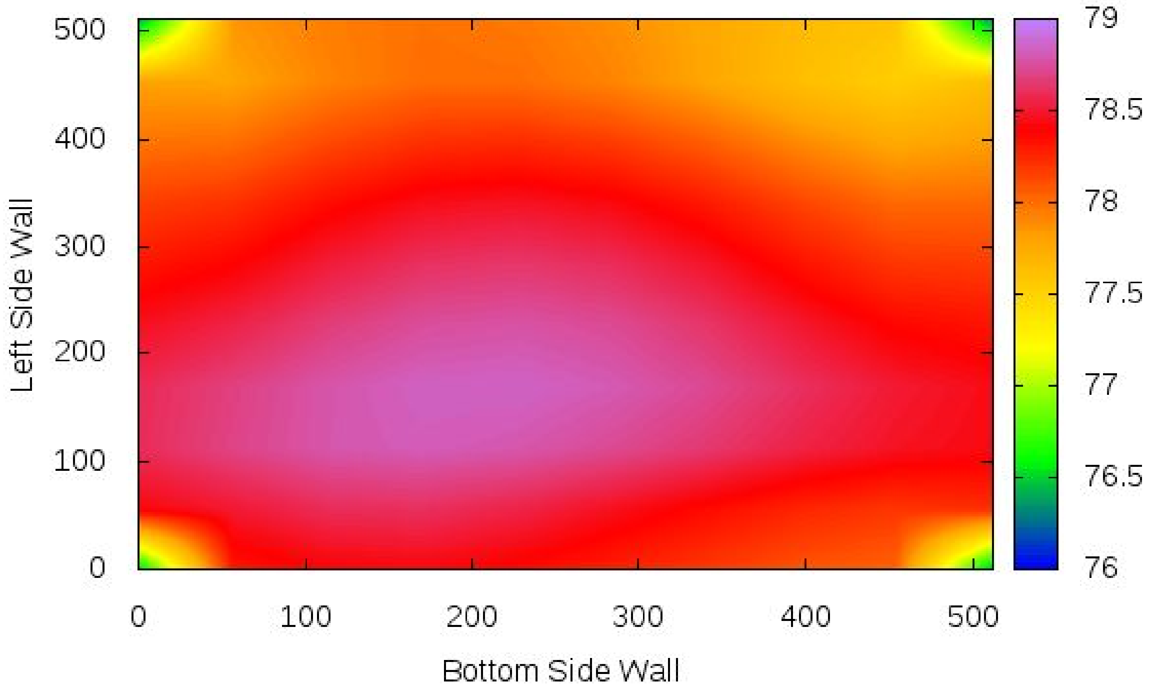
This is after 3,000 iterations. You can see the window leak cold air inwards. The entire heat column is shifted downwards. We see the heat disperse like normal. The 85 degree heat source disperses heat and the heat dissipates to surrounding areas.


3,000 Iterations
Left Side Wall
Bottom Side Wall
0
100
200
300
400
500
76
76.5
77
77.5
78
78.5
79

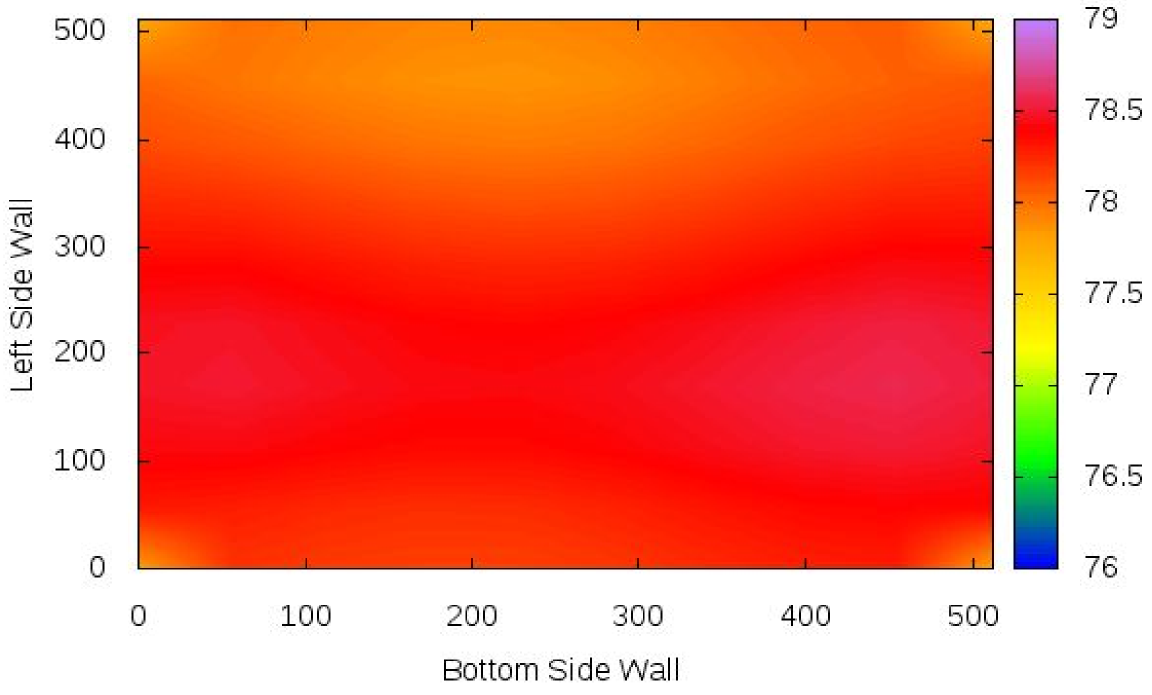


From 6,000 iterations, we can see that heat has dissipated from the heat source and has begun to even out. The cold air coming from the window seems to not affect the room as substantially as before. With 12,000 Iterations, the remainder of the heat seems to be focused near the heat source and the remainder of the room seems to even out. As our iteration count rises to 25,000, everything begins to converge towards one temperature and the differences between our 50,000th iteration and our 100,000 iteration are negligible.

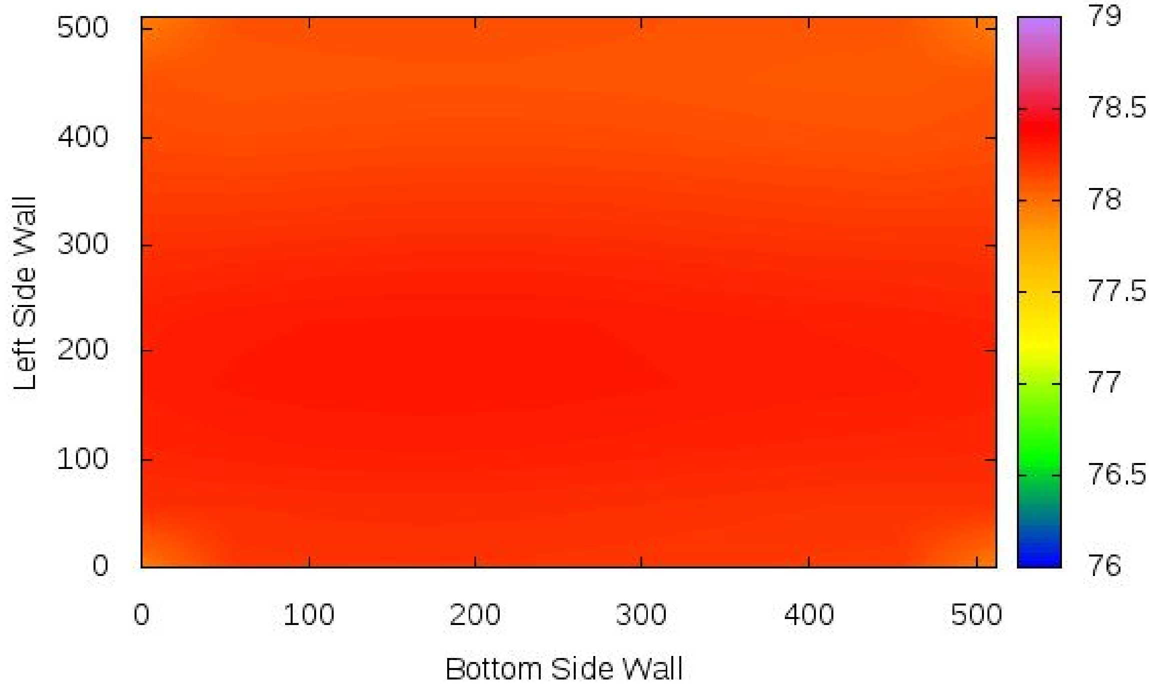
12,000 Iterations



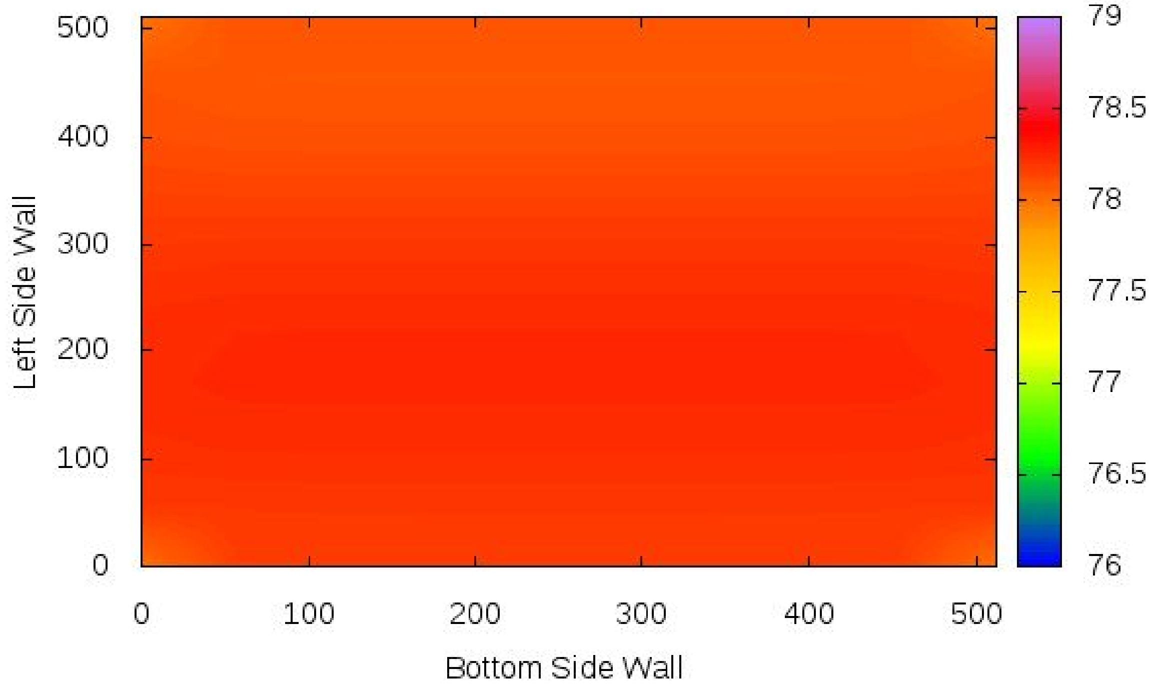
25,000 Iterations



50,000 Iterations



100,000 Iterations



Although our second set of experiments, in which we include a convection model to more accurately capture the room, displays signs of transfer of heat in a more expected way, the results are ultimately the same. The temperature for the room ultimately converges to approximately one temperature, notably a hot temperature (one greatly above our desired). This occurs even with the windows set to be widely opened.

This might have been caused by mixing models. Our first model of the windows/convection was implemented in such a way that the temperature of the windows was determined by its opened width; this obviously does not make as much sense in the real world. Combining this with our new drift model, we were simply having warm air pulled into the room, which may have distorted our results. A better experiment would be to keep the window temperature static depending on the outside temperature, and model the rate at which the air moves into the room by the width of the window. Rather, we kept the rate of the movement static and varied the window temperature (again, this is caused by the combination of the models).

V. Conclusion

Overall, the entire project was successful in that we learned many things. Although we did not succeed in designing a program which would allow us to accurately choose how open a window should be in order to achieve a specific convergence temperature, we discovered which models work better than others. We discovered that changing heat conductivity for the air is a poor model, while adding convection is a better model. We discovered that modeling a window's temperature based on its open width is a poor choice, and that using the opened width to model the velocity of the air moving into the room is a much better choice. Perhaps, for future improvement, we can run experiments inside the room to refine our parameters for how fast air should move into the room. Including a wind parameter would be wise as well to model this feature.

Additionally, we learned more things about GPU programming, which is a success. We learned what makes a program faster and what makes a program slower. Additionally, we also discussed optimization methods which we didn't get a chance to use. For example, we discussed that if different window parameters converged to a variety of values, we could use a form of dictionary search (where we estimate and then check) parameters depending on their output values. However, because the temperatures always converged to similar values despite the value of the input parameters, this wasn't implemented (it was still good we thought about it though!).

VI. Contributions and Team Assessment

Because the entire team was eager to learn, every task was actually touched by each team member. There was a lot of debugging and testing to do, so each task had plenty of subtasks to perform. A list of the task is as follows:

1. Writing GPU Baseline Code for SOR
2. Integrating the Cuda GPU Code with C INIT File
3. Integrating a Second GPU Kernel Code into the Model
4. Gathering and Analyzing Data
5. Organizing GitHub REPO
6. Designing a Model of the Room and Nature's Behaviors (temperature, wind)

Overall, our team was always on the same page and functioned smoothly. We rotated tasks whenever we needed a new pair of eyes to troubleshoot an issue.