Heat Diffusion Simulation with Parallel Programing

1. Introduction

Diffusion is the movement of anything, atoms, energy, heat for example, from regions of higher concentrations to the lower ones. We can observe this phenomenon frequently in real life, from making a cup of tea, turning on air conditioner, putting a drop of ink to the water to the nutrient exchange process in plants and animals. The flowing figure demonstrates a diffusion process.

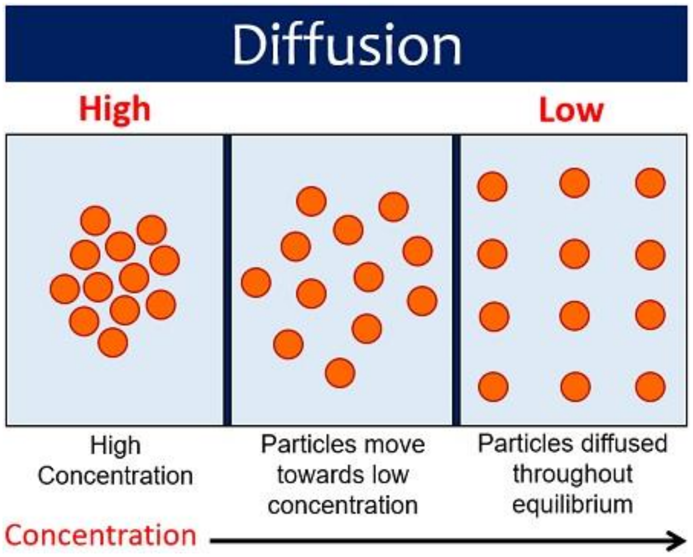
Diffusion
High
Low
High Concentration
Particles move towards low concentration
Particles diffused throughout equilibrium
Concentration

Figure 1 Illustration of diffusion phenomenon

Since diffusion appears in so many phenomena in nature, the need of simulating this process is highly demanded. Engineers and scientists need to know the final state of the system, given initial conditions. And in many cases, they are only interested in the final steady state of concentration field, because the transient process occurs only in a short period of time. In this project, we examine a method to simulate diffusion process, the Gauss-Seidel, and using parallel algorithms to reduce computational time.

2. Mathematical Background

2.1. Laplacian Equation

To find the equilibrium state, we need to solve the Laplacian equation. Let's consider the two-dimensional grid, i.e., the computational field is a flat region. Each position in grid corresponds to one concentration value. Setting all the time derivatives to zero in the diffusion equation, we derive the time independent diffusion equation

$$\nabla^2 c = 0$$

This famous Laplacian Equation is crucial in many fields of sciences and engineering. Denoting the concentration of the point (i, j) is $c_{i,j}$ and substituting Laplacian Equation to diffusion equation, we get the system of equations

$$c_{l,m} = \frac{1}{4}\left(c_{l+1,m} + c_{l-1,m} + c_{l,m+1} + c_{l,m-1}\right), \qquad \forall\ (l, m)$$

Solving this system of equations, we will get the final steady state of the system. To be more convenient, we can rewrite these equations in the form on matrix $\boldsymbol{A} \times \boldsymbol{x} = \boldsymbol{b}$, where $\boldsymbol{A}$ is $(n \times n)$ matrix and $\boldsymbol{x}, \boldsymbol{b}$ are $(n \times 1)$ vectors.

There are two approaches to solve this problem, the first is direct method, which consists of mathematical steps lead to direct result. The other is iterative method, in which we seek to approximation solutions by updating the values step by step, until reaching the pre-defined stopping condition. Direct method requires higher computational complexity, compared to iterative one. Especially in this digital era, we usually encounter large scale simulation problems need to be handled in a short period of time. In this project, we will examine an iterative methos, the Gauss-Seidel and implementation parallel algorithms in Graphical Processing Unit.

2.2. Gauss-Seidel Iterative Method.

Before moving to Gauss-Seidel, we would like to introduce a little bit about Jacobi, which is also an iterative method. The value of a point in grid at a certain iteration is updated based on the old value of four neighboring point. The updating formular is

$$c_{l,m}^{n+1} = \frac{1}{4}\left(c_{l,m+1}^{n} + c_{l+1,m}^{n} + c_{l-1,m}^{n} + c_{l,m-1}^{n}\right)$$

.

A disadvantage of Jacobi method is that it only uses the old values to update the new one, although new values are available. It might slow down the process, i.e. Jacobi method might require more iterations than other methods. Moreover, Jacobi method does not update in-place, old values still need to be stored.

The Gauss-Seidel Method can be considered as an improvement of Jacobi, it uses new values as soon as they are computed. The first version of Gauss-Seidel is Row-Wise Ordering, in which the values of point are updated from left to right, top to bottom. We can formulate this as the following schema

$$c_{l,m}^{n+1} = \frac{1}{4}\left(c_{l,m+1}^{n} + c_{l+1,m}^{n} + c_{l-1,m}^{n+1} + c_{l,m-1}^{n+1}\right)$$

Figure is the demonstration of row-wise ordering schema. The top-left point is updated first, and the bottom-right is the final changed in the grid. A derivation of row-wise ordering is that we cannot apply parallel algorithm for this strategy, since the point still depends on its neighbors. That is the reason red-black ordering was invented.

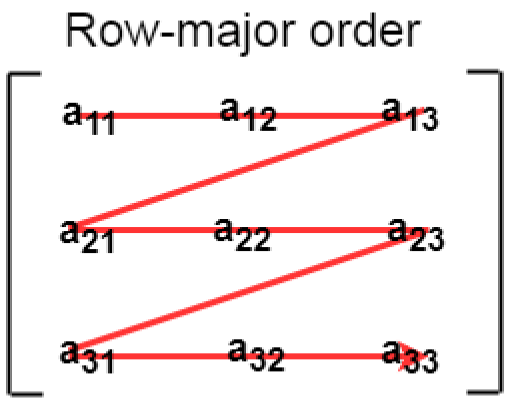

Figure 2 Gauss-Seidel Row-Wise Ordering

Instead of updating value in a row-wise manner, red-black ordering divide grid into red and black points, like a checkerboard. The flowing figure is an example of coloring grid by red-black ordering. By this strategy, we observe that the red points only depend black one, and vice versa. This means that we can update red points in parallel, and so as for the black points. Following is updating schema of Gauss-Seidel Red-Black Ordering.

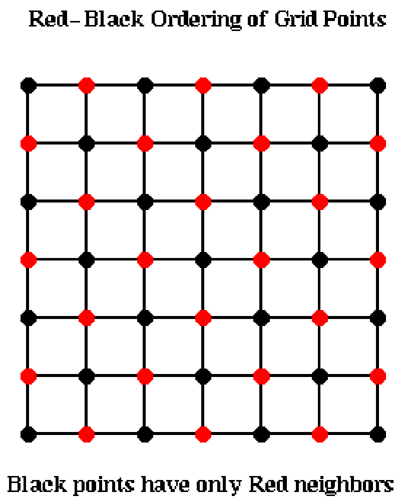
Red–Black Ordering of Grid Points
Black points have only Red neighbors

Figure 3 Gauss-Seidel Reb-Black Ordering

$$c_{l,m} = \frac{1}{4}\left(c_{l,m+1} + c_{l+1,m} + c_{l-1,m} + c_{l,m-1}\right)$$

Because Gauss-Seidel is an iterative method, the stopping condition is required. In our case, we stop the updating process when the maximum changing is smaller than a threshold. Formally, we demand that

$$\max_{i,j} |c_{i,j}^{n+1} - c_{i,j}^{n}| < \epsilon\,.$$

3. Algorithms

3.1. Gauss-Seidel Parallel Algorithm

Since our computational filed is two dimensional, we divide the Graphical Processing Unit, GPU for short, by two dimensions. Each thread is responsible for a 2D region, which has equal size comparing to other. It will update both red and black point in its own region. For simple implementation, our convention is that the size of computational gird is divided by number of total threads by each dimension. Let take an example when the grid has size 256 by 256, the dimension of blocks is 2 by each direction, and shape of threads is 2 by 2 in each block. In this condition, the thread at position $(0, 0)$ of block (0, 1) is responsible for the region with x-index from 0 to 63 and y-index from 128 to 191.

The pseudo-code for parallel Gauss-Seidel is described as follows. Firstly, the red points are updated in parallel, while the maximum local changing of them in each thread are updated. The following is the black points updating without any difference, except computational field. When all the points are updated, we need to examine the stopping condition, to decide whether to continue the process or nor. This task requires global maximum changing, among the local one, computed by each thread. For obtaining the global maximum, we use parallel reduction, which is described more detail in next section.

```
do {
        for each thread {
            update its red points;
            compute max local changing $\delta_{i,j}$; }
        for each thread {
            update its black points;
            compute max local changing $\delta_{i,j}$; }
        compute global maximum changing $\delta$ ;
    } while ($\delta > threshold$)
```

3.2. Parallel Reduction Algorithm

As mentioned in the previous section, the stopping condition required global maximum. Because GPU uses shared memory, if we use only an address to store global maximum, it will raise synchronization issue when more than one threads write to this position. Address for each thread to store local maximum is compulsory. As a results, we need to implement mechanism to obtain the maximum, with a low computation cost. Parallel reduction is our proposal approach.

We introduce the parallel reduction for one-dimensional array first, and will extend it to two-dimensional. The idea of parallel reduction is quite simple. At the first step each thread will be assigned to find maximum of two consecutive values, and store it in the first position, among two. Next, each thread will still be required to find maximum among two values, which is maximum of two local values obtained in previous step, and store in first position among two. When the second step is done, we obtain local maximum of four consecutive values. The number of active threads is reduced two by each step. Continuing this process, until the total number of threads is one, we obtain the global maximum in the 1D array.

.

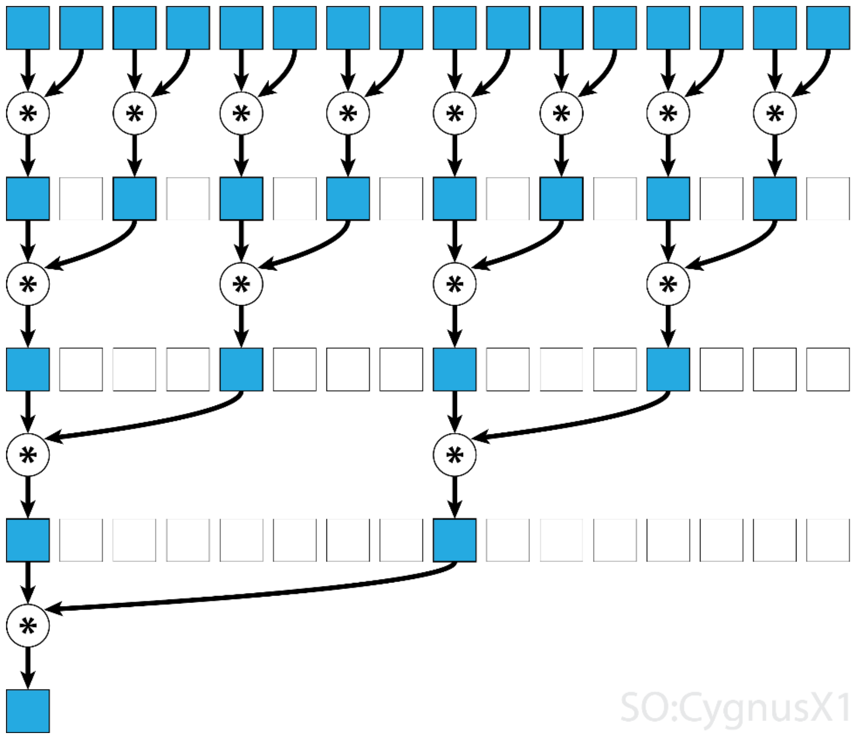

Figure 4 Parallel Reduction for one-dimensional array

Extending this strategy to two-dimensional field, we perform parallel reduction in each row, by which we can get the row-wise maximum at the first position. The first column now contains all maximum of each row. Formally, $\epsilon_{i,1}$ is the maximum value of i^{th} row. The remaining step is quite trivial, performing one-dimensional parallel reduction to the first column will get the global maximum of all matric.

4. Results

In this section, we will demonstrate the results of our simulation. We initialize the grid with size 256 by 256, the blocks dimension is 16 in both x and y direction. Each block has shape 8 by 8 threads. One convention is that the shape of computational grid must be divided by size of threads by both dimensions. For grid points value, the square center region is initially 80, and the other is 25. The boundary condition in this case is 100 at top-right side and 0 at bottom-left. These values can be initiated with any strategy. With this initialization, the system is considered at equilibrium state with threshold 0.001 at iteration 11947.

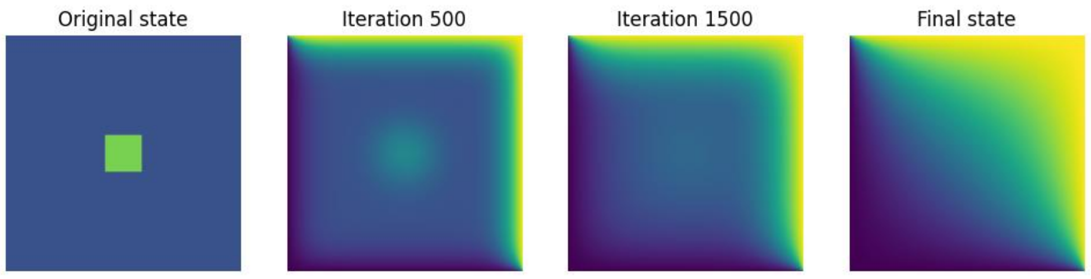

Figure 5 Diffusion process with values 100 at top-right edges

We can change the boundary conditions and obtain other results. Instead of values 100 at top-right side, we change to 100 at top and down, while values at left and right edges are 0. The system converts after 11090 iterations, following is the demonstration of this process.

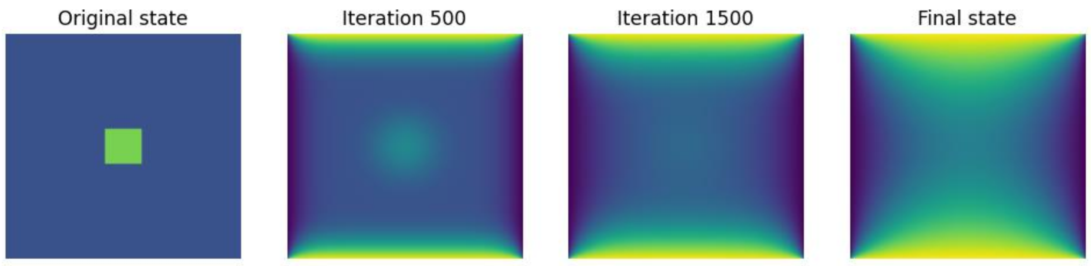

Figure 6 Diffusion process with values 100 at top and down edges

5. Conclusion and Further Development

In this project, we have implemented Gauss-Seidel with parallel algorithms on Graphical Processing Unit. Simulation of these phenomena in sciences and engineering is highly demanded, especially with fast and effective computation cost.

We implemented a two-dimensional grid, while it can easily extend to three dimensions, such as the diffusion of molecules in space, or any number of dimensions. Another development of our work can be examining the effectiveness of Gauss-Seidel method, compared to others iterative method, like Jacobi or Successive Over Relaxation. We can also evaluate the overhead of parallelism with the linear algorithm, with specific computational size.