# Mesoscale Microstructure Simulation and Parallel File I/O Using AMOS

## Parallel Programming and Computing
## Final Project Report, Spring 2017

*Chengjian* Zheng[1], *Feifei* Pan[2,3], *Guohui* Liu[3], and *Liyu* Pan[2,3]

[1] *Rensselaer Polytechnic Institute, Mechanical, Aerospace and Nuclear Engineering Department, Troy, New York, USA*
[2] *Rensselaer Polytechnic Institute, Computer Science Department, Troy, New York, USA*
[3] *Rensselaer Polytechnic Institute, Information Technology and Web Science Department, Troy, New York, USA*

**Abstract.** Parallel computing is widely applied to simulate various mesoscale microstructure models in physical system. For this group project, our group surveys a wide range of papers and projects in related research area and we plan to improve, extend, and re-implement the existing project on applying parallel computing techniques to simulate two physics models that we are particularly interested in — crystal grain evolution, also known as the grain growth model. The implementation of our simulations will be based on Rensselaer Polytechnic Institute's petascale supercomputing system AMOS, a Blue Gene/Q system.

## Keywords

**Mesoscale Microstructure simulation; multithreading; pthreads; Crystal grain evolution; Blue Gene/Q; Parallel file I/O; Supercomputing.**

## 1 Introduction

One major application of parallel techniques in the world of physics and material science is performing various simulations. The existing Mesoscale Microstructure Simulation Project aims to provide "a simple, consistent, and extensible programming interface for all grid and mesh based microstructure evolution methods". This project is written in C++ with example solutions to over 22 model simulations in both 2 dimensional (2D) and 3 dimensional(3D), with or without parallelism. We intend to make improvement and extend the existing methods in the project, especially Monte-Carlo Potts model, to address the problem of a 2D crystal grain evolution, also known as the grain growth simulation. For visualization, we use Matlab to generate stage diagrams and experiment result plots.

In this project report, we first provide some basic background information of AMOS, the supercomputer, and Blue Gene/Q system in Section 2. Following that, we briefly discuss the mesoscale microstructure model–2D crystal grain evolution and how to solve this problem with parallel computing techniques. In Section 4, we focus on the algorithms involved in our group project. Experiment settings and result analysis are presented in Section 5. We finalize our report with Section 6, Conclusion and Future Work.

## 2 AMOS and Blue Gene/Q

### 2.1 General Information

Blue Gene is a series of supercomputers designed by IBM started from year 1999, while Blue Gene/Q is the third generation in this series. The system of Blue Gene/Q is scaled to 256 racks (increased from the 72 racks in its previous generation Blue Gene/P) and beyond. It is the highest capability machine in the world with 20- 100FP+ peak. It is very reliable however takes low maintenance effort. Blue Gene/Q has high power efficiency with low latency, high bandwidth inter processor communication and memory system. On the software side, it posses open source programming environment. The architecture extends application reach.

The AMOS at Rensselaer Polytechnic Institute is a 5-rack IBM Blue Gene/Q system supercomputer. AMOS has 5120 nodes where each node has a 16-core 1.6 GHz A2 processor and a 16GB DDR3 memory. With the capability of performing over one quadrillion calculations per second, AMOS is the most powerful academia-based supercomputer in the Northeast of the United States.

### 2.2 Compile Environment

For our group project, we use the IBM XL C/C++ for BlueGene v 12.1 (mpi-xl) compiler for C and C++ codes. AMOS can be accessed using ssh q or ssh amos. On AMOS, we can use the **modules** commands to locate available compilers and change compile environment settings.

The following are some commonly used modules commands:

- module avail: list the currently available modules.
- module list: list loaded modules.
- module load xl: load the IBM XL compiler set.
- module load gnu: load the GNU compilers.
- module unload or module clear:Remove all modules.

In addition, **Slurm** is used on AMOS for job-scheduling and **sbatch** can be used to submit batch jobs.

## 3 Mesoscale Microstructure Model

In this section, we comprehensively discuss the general concepts behind the mesoscale microstructure model we're interested in. Moreover, we present the reasons why we believe the simulation of this particular mesoscale microstructure model is a perfect parallel programming and computing topic.

We will first start with the definitions of crystal gain evolution models.

### 3.1 Crystal grain evolution

Crystal grain evolution is also known as grain growth, that is, at high temperature, the size of grains (crystallites) will increase in material. Grain growth problem has long been studied and the rules for grain growth are well-established. Grain growth occurs because of the movement of grain boundaries. Starting from the initial state, an arbitrary grain follows a set of rules to change states, during which the size of grains will change simultaneously. Grains are particularly small. Therefore, a material usually contains a great number of grains and each grain will need to be calculate or evaluate before changing states.

Similar models include Lattice gas dynamics, which is a type of cellular automaton. As a cellular automaton, each model contains a lattice which contains plenty of sites that can take a certain number of particles with certain velocities. Starting from the initial state, the states of the particles keep changing. The states are always boolean value, which means at a given site at a particular time step, there is either a particle moving in each direction or not. The state of an arbitrary particle is defined by the state of itself and its neighbors. Generally speaking, at each time step, every particles requires a calculation for its next state.

In this project, we focus on crystal grain evolution.

### 3.2 Solve the Problem with Parallel Computing

In nature of lattice gas dynamics and crystal grain evolution, both models start with an initial status and the status will keep changing with a series of steps. In each step, massive calculation or evaluation is required before status change. Use crystal grain evolution as example, our initial size of the crystal is around 10um, which contains about 10,000 grains and is partitioned into $4096 \times 4096$ grid points. In each state, each of the 10,000 grains will need to be evaluated based on a set of rules. By using brute force algorithm, it would take a tremendous amount of time to complete the calculation. However, with parallel computing techniques, we can greatly reduce running time.

```
pseudo code for Voronoi Tesselation

Tesselation(GRID &grid, int numSeeds){
        if (rank==0)
            Randomly generate N seeds in the global domain\
                store them in the array seeds[N][2];
        Broadcast all N seeds from rank 0 to all the ranks;

        for all the lattice points in my local grid{
            int minDistance = INT_MAX;
            int minID = 0;
            for (int i=0; i<N; i++){
               Compute the Distance from seed[i] to the
               current lattice point;
                   if (Distance < minDistance){
                       minDistance = Distance;
                       minID = i;
                    }
            }
            Modify the grain ID in current lattice point as
            minID;
        }
    }
```

**Figure 1.** pseudo code for Voronoi Tessellation

## 4 Algorithms

In Section 4, we present the major algorithms we used in our project. First, we use Voronoi diagram to randomly generate a Step 0 crystal. The most important algorithm for our simulation is the Monte-Carlo Potts Model, which is used to calculate/ evaluate the status change in every step. Parallel File I/O using MPI functions serves to further improve the performance. In addition, we implement multithreading to increase parallelism.

### 4.1 Voronoi Tessellation

Voronoi Tessellation, also known as Voronoi decomposition and Voronoi partition, is the very first step for our Mesoscale Microstructure models Simulation project. We use Voronoi Tessellation to generate the Step 0 crystal, with 10,000 grains, which we use in our future grain growth process.

The theory behind Voronoi Tessellation can be described as follows: assuming we have a space $X$ and a site (also known as lattice or grid) set $G = g_1, g_2, ..., g_n$ with $g_i$ represents a sites. A Voronoi Tessellation aims to partition the space $X$ into $n$ cells, one for each site in set $G$, while there exists a point $p$ lies in the cell corresponding to a site $g_i$ in $G$ satisfying that the distance between the site $g_i$ and point $p$ is smaller than the distance between $g_j$ and $p$, where $g_j$ is an arbitrary site in $S$ other than $g_i$:

$$d(g_i, p) < d(g_j, p) \tag{1}$$

We implement Voronoi Tessellation to create the Stage 0 crystal. In addition, we use a series of MPI functions in the existing MPI libraries on AMOS to add parallelism to Voronoi Tessellation algorithm. The pseudo code for this implementation is show in Figure 1. The result of our code for Voronoi Tessellation, a 2D diagram, is presented in Figure 2.

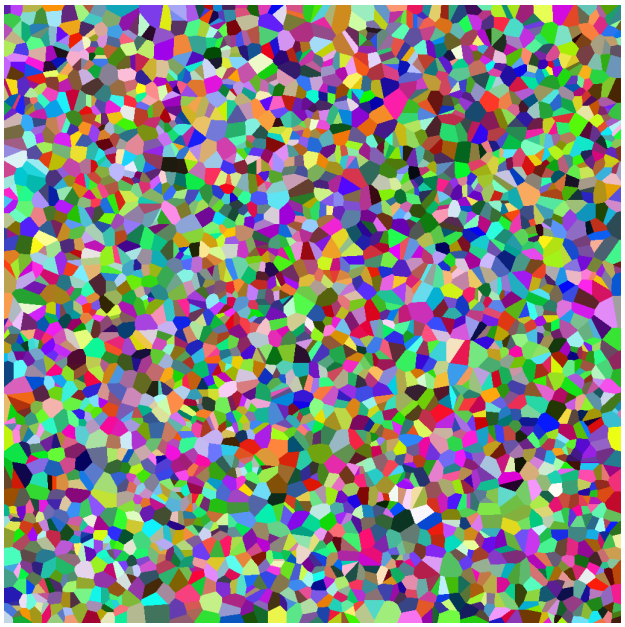

**Figure 2.** The initial stage crystal diagram created using our code on AMOS.

## 4.2 Monte-Carlo Potts model

### 4.2.1 General Monte-Carlo Potts model

The Monte-Carlo Potts model simulation maps the microstructure of a material onto a 2-dimensional sites where each site of the lattice or grid has a spin, $s_i$. $s_i$ represents the orientation of the grain. The total energy in this system is calculated by Hamiltonian, the operator corresponding to the total energy in quantum mechanics.

$$H = {}^{J}/_{2} \sum_{i=1}^{N} \sum_{j=1}^{N} (1 - \delta(S_i, S_j)) \tag{2}$$

where the first sum over $i$ is the sum over all sites in the system and the sum over $j$ is the sum only on $z$ nearest neighbors. $\delta(S_i, S_j)$ is the Kronecker delta function. It equals 1 if $S_i = S_j$, else equals 0.

Monte-Carlo Potts Model randomly selects a site and then decides if the spin of this selected site will change based on the change of total system energy. The process for a N lattice site is listed as below:

- Randomly select a lattice site $i$;
- Randomly select a spin $S_j$ of site $j$;
- Change spin of site $i$ to $S_j$ following probability equation(3):

$$P_i(\triangle E) = \begin{cases} 0 & \text{if } \triangle E > 0 \\ M_i & \text{if } \triangle E \leq 0 \end{cases} \tag{3}$$

  where $M_i$ represents a scaling function and $\triangle E$ represents the energy change with new spin value $S_j$ assigned to site $i$.
- Time increase by 1 after each attempted spin flip.

```
pseudo code for Monte-Carlo Potts model

For all the lattice points{
       If (at Grain Boundary) {
            - Randomly select a neighbor grain ID
            - Compute energy difference:
              Original Energy - Replaced Energy ==> \Delta E
            - Make decision: accept new grain ID or not?
                    \Delta E < 0 => accept
            - Modify grain ID
       }
   }
```

**Figure 3.** pseudo code for Monte Carlo Potts model

- Each time step of this system requires N attempted spin flips.

The pseudo code for Monte Carlo Potts is shown in Figure 3.

### 4.2.2 Moving to Parallel with Pthread

As we can see from our previous algorithm analysis of Monte-Carlo Potts model, 10,000 grain IDs require 10,000 attempted spin flips, which means a lot of massive calculations. Therefore, we add parallel techniques to this algorithm to improve the program performance.

The $4096 \times 4096$ grid points are split into N ranks, such that each rank contains a grid composed of a contiguous subset of the grid points. The grid has dimensions of $(4096/\sqrt{N})$ by $(4096/\sqrt{N})$. A grid is further divided into sub-grids by m threads. Each thread handles a sub-grid with $(4096/(\sqrt{N} * m))$ rows of a grid. Special consideration is needed for the last thread if the remainder of this division is not zero. That's to say, we need to cover all remaining cells in the sub-grid of the very last thread.

From grain growth point of view, we need both ghost rows (upper and lower) and ghost columns (left and right) for grids in all ranks. In addition to the ghost rows and columns, ghost grid points for the four corners of each grid is also needed for grain calculation. We introduce Upper Left Ghost Point, Upper Right Ghost Point, Lower Left Ghost Point, and Lower Right Ghost Point to each grid. In this way, all the corner grid points can have correct grain values for growth calculation, and all scenarios can easily be covered. In each rank, MPI_ISend is used to send grain IDs of the ghost grid points to 8 neighbor ranks, and MPI_IRecv is used to receive the grain IDs of the ghost grid points from 8 neighbor ranks. The grids have both row and column wrap around.

Global pointer variables are created to store the addresses of the grain IDs for inter-process information exchanging. A dedicated function is written to take care of each rank's sending and receiving processes which are performed only once in each iteration. With such solution, the update function can focus on the grain growth steps and multi-thread logic. For grid points in the boundary rows and columns, update function holds its steps until MPI_IRecv returns the necessary ghost grid points. The

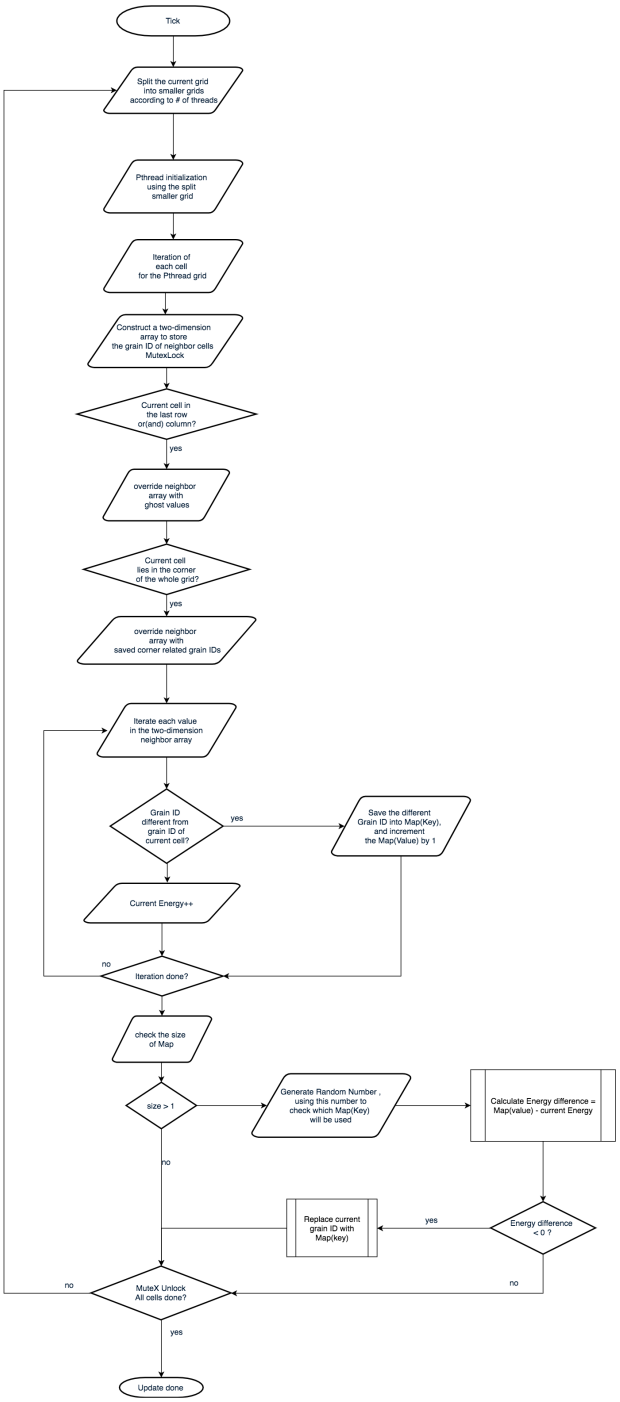


**Figure 4.** Work flow for each tick with pthreads

sending process is performed as soon as the update function is done to eliminate possible communication overhead.

This whole process is shown as a work flow in Figure 4.

### 4.3 Parallel File I/O

The output of all the ranks is written to the output file simultaneously using MPI_write_at_all(). To determine where in the file each line should be written, let origin(x,y) be the global coordinates of the rank's first grid point, i be

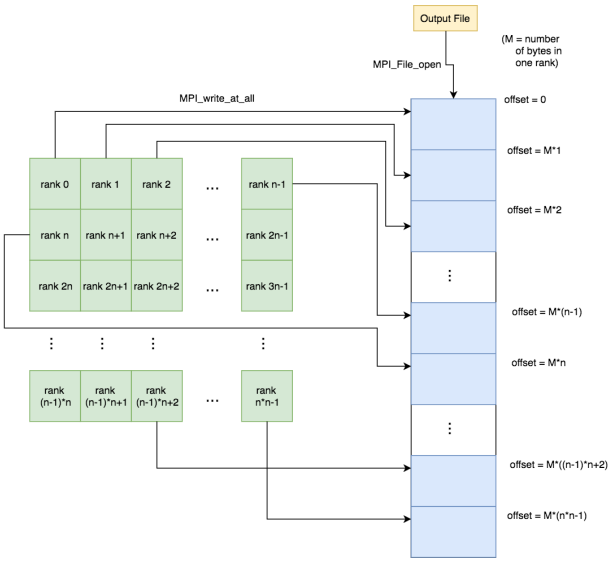


**Figure 5.** Parallel file output schema

```
pseudo code for Output

For each line in grid{
        compute the offset of the line{
        offset = (grid.x + line_index) * numColumns *
                    (sqrt(numRanks)) + grid.y
           write the line to the output file with the offset
     }
     }
After all ranks completed writing output, cut the one-line
output into multiple lines, with each line having the right
number of integers.
```

**Figure 6.** pseudo code for file output

the line index number, numRows be the number of rows in the rank, numCols be the number of columns in the rank, and numRanks be the total number of ranks, the offset is calculated as:

$$offset = (origin.x + i) \times numCols \times \sqrt{numRanks} + origin.y$$

After all ranks completed writing output to file, the one-line binary output is cut into multiple lines and converted to decimal integers using hexdump. This parallel file output schema is visualized in Figure 5 and the pseudo code is shown in Figure 6.

## 5 Experiments

In this section, we discuss the most significant part of our group project, which is the experiments. We start with the experiment settings subsection. We specify the execution conditions we use in each experiment, including the number of Blue Gene/Q nodes, MPI ranks per node and threads per rank. Next, we describe the results in details, show grain growth diagrams for different stages and compare the results from various execution conditions. Detail result analysis and discussion can be found in Section 5 as well.

### 5.1 Experiment settings

We design a set of strong scaling experiments. With the total degree of parallelization fixed at 4096, 3 settings for Blue Gene/Q nodes are used, 64 nodes, 128 nodes and 256 nodes. The corresponding combinations of execution conditions can be found in Table 1.

**Table 1.** Strong scaling Experimental Settings for Parallel Monte Carlo Simulation

| BGQ Node | MPI Rank / Node | pthread / Rank |
|---|---|---|
| 64 | 64 | 1 |
| 64 | 16 | 4 |
| 64 | 4 | 16 |
| 64 | 2 | 32 |
| 64 | 1 | 64 |
| 128 | 32 | 1 |
| 128 | 8 | 4 |
| 128 | 2 | 16 |
| 128 | 1 | 32 |
| 256 | 16 | 1 |
| 256 | 4 | 4 |
| 256 | 1 | 16 |

### 5.2 Experiment results & analysis

Within the total 5,000 iterations, we display one diagram for every 1,000 iterations, as shown in Figure 8 to Figure 12, with the initial stage of the crystal shown in Figure 2. The performance study of the strong scaling experimental settings in Table 1 is plotted in Figure.7.We can see the overhead of pthread creation through the trend of computation time with respect to pthread per rank settings. At the same degree of parallelization, computation time increases almost linearly as the pthread per rank increases for the 64 Node setting. The overhead is low moving from 1 pthread per rank to 16 pthread per rank, but gets significantly higher after that, especially for the 64 BGQ Node setting. The highest parallel efficiency is achieved at the case with 128 BGQ Nodes, 8 MPI Rank / Node and 4 pthread / Rank.

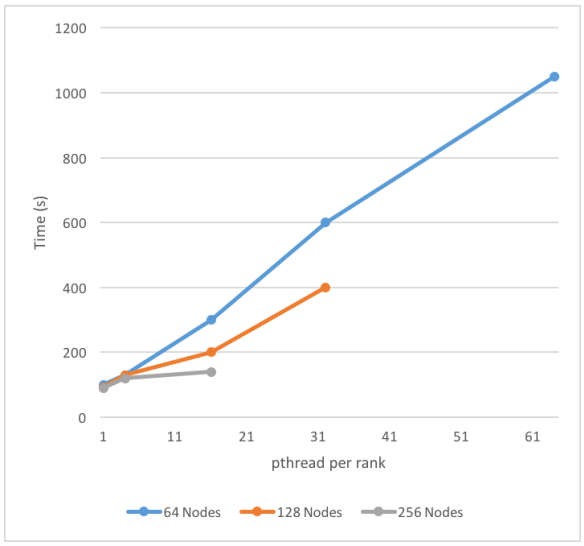


**Figure 7.** The computational time under different rank and pthread settings. (Total Rank × pthread per rank is kept fixed as 4096).

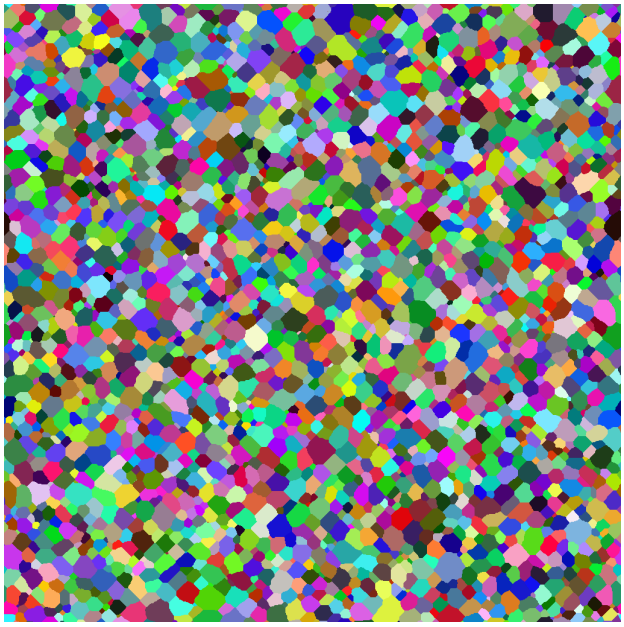

**Figure 8.** Grain Growth Diagram after 1,000 iterations

## 6 Conclusions and Future Work

### 6.1 Conclusions

The major contributions of our group project can be categorized as: (1) successfully implement the parallel file I/O using MPI ranks; (2) add parallel feature-pthreads to the existing Mesoscale Microstructure Models Simulation project framework. We are able to achieve parallelism during the implementation of Voronoi tessellation with MPI ranks, Monte-Carlo Potts model with both pthreads and MPI ranks, and the file output with MPI ranks. Our implementations demonstrate that by adapting parallel techniques we learned from CSCI-4320/6320, the system performance can be strongly improved.

The experimental results demonstrates the overhead of pthread operations. We believe it will be more efficient in less saturated conditions (lower than 16 pthread per rank) for high parallel efficiency.

### 6.2 Future Work

In one recent paper on optimizing the simulation of grain growth *SPOCK: Exact Parallel Kinetic Monte-Carlo on 1.5 Million Tasks*, a new scalable implementation of Monte-Carlo method is proposed. SPOCK, Scalable Parallel Optimistic Crystal Kinetics, is accomplished using the Time Warp paradigm. Time Warp is a optimistic synchronization protocol which is both robust and scalable for parallel discrete event simulation. We plan to extend our

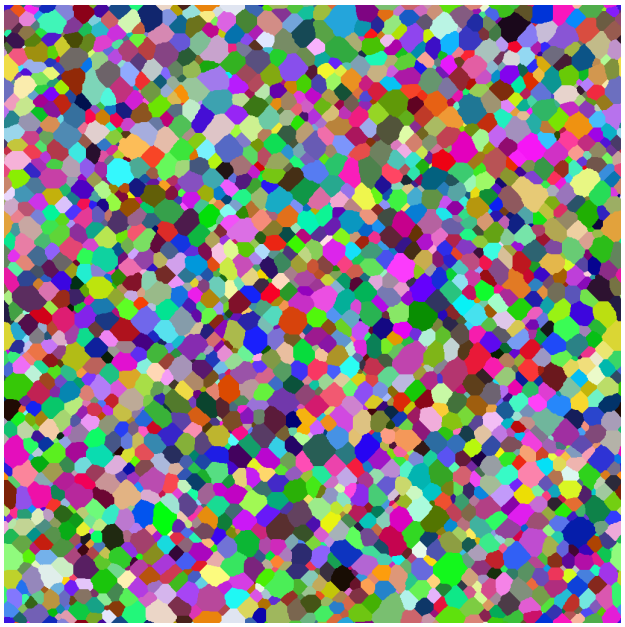

**Figure 9.** Grain Growth Diagram after 2,000 iterations

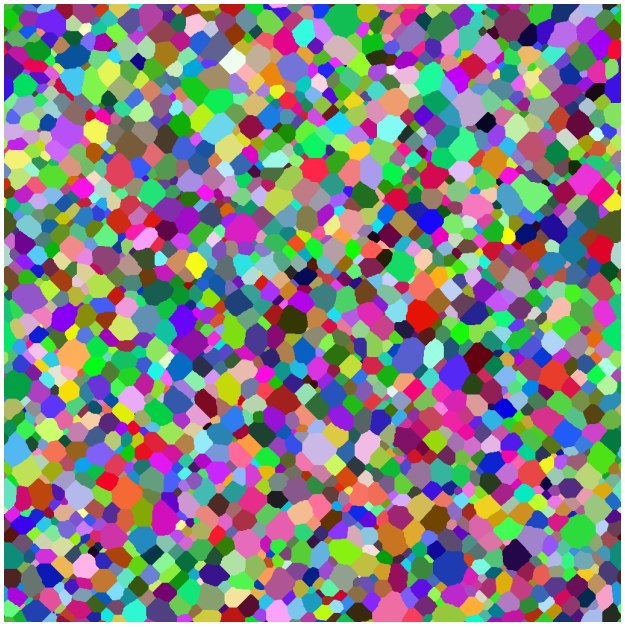

**Figure 10.** Grain Growth Diagram after 3,000 iterations

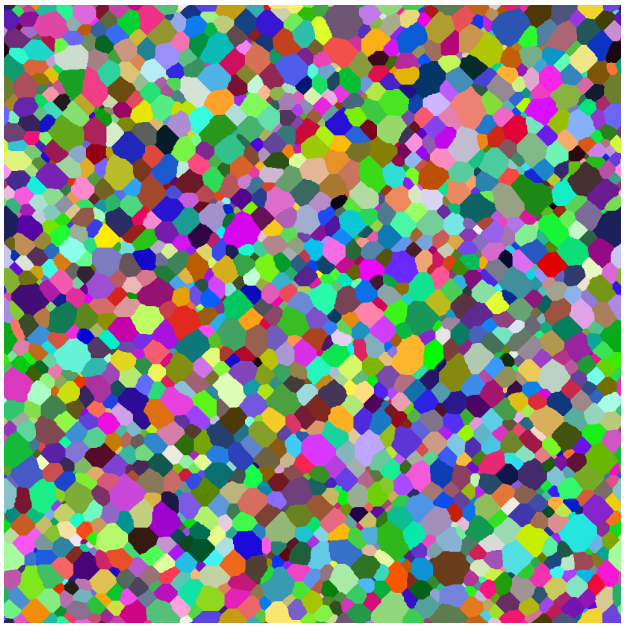

**Figure 11.** Grain Growth Diagram after 4,000 iterations

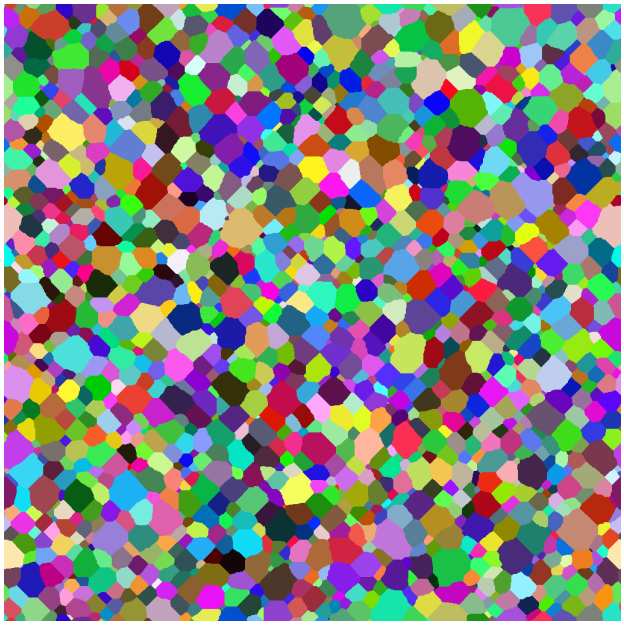

**Figure 12.** Grain Growth Diagram after 5,000 iterations (Final stage)

current simulation method further to ROSS simulator running Time Warp paradigm to achieve even larger scale and make improvements in system performance.

## 7 Contribution

- **Chengjian Zheng**: Chengjian is the functional and technical designer of this solution. Chengjian also led the implementation of this project in terms of coding, testing, running, and analysis of performance results.
- **Feifei Pan**: Feifei led the whole project from the very beginning, and involved in each development phase. Feifei and Chengjian are the functional designers of this project. Feifei is a key contributor of this paper.
- **Guohui Liu**: Guohui acted as a technical designer and a programmer in this whole development, and Guohui contributed to the technical parts of this document.
- **Liyu Pan**: Liyu acted as a technical designer and a programmer in this whole development, and Liyu also contributed to the technical parts of this document.

## References


[1] S.A. Wright, S.J. Plimpton, T.P. Swiler, R.M. Fye, M.F. Young, E.A. Holm, Potts-model Grain Growth Simulations:Parallel Algorithms and Applications pp. 11–18 (1997)

[2] J. Gruber, T. Keller, Mesoscale Microstructure Simulation Project (https://github.com/mesoscale/mmsp, 2014)

[3] T. Oppelstrup, D.R. Jefferson, V.V. Bulatov, L.A. Zepeda-Ruiz, SPOCK: Exact Parallel Kinetic Monte-Carlo on 1.5 Million Tasks (2016)

[4] S.Plimpton, C.Battaile, M.Chandross, L.Holm, A.Thompson, V.Tikare, G.Wagner, E.Webb, X.Zhou, C. Cardona et al., *Crossing the Mesoscale No-Man's Land via Parallel Kinetic Monte Carlo* (Sandia report SAND2009-6226, 2009)

[5] J.Fu, N.Liu, O.Sahni, K.Jansen, M.Shephard, C.D.Carothers, *Scalable parallel I/O alternatives for massively parallel partitioned solver systems* (IEEE International Symposium on Parallel Distributed Processing, 2010), pp. 1–8


.