Phase-Field modeling of dendritic solidification with MPI(C++)

Chirantandip Mahanta
MM19B029

Term Project
AM5035 : High Performance Computing Lab

Indian Instutute of Technology, Madras.

Abstract

We will numerically solve the isotropic and anisotropic dendritic growth during solidification using the Phase-Field method. The mathematical model of the phenomenon is derived from the work of Kobayashi[1]. The PDEs will be solved within the finite-difference approximation on a 5 point stencil. The program will be written in C++ and MPI(Message Passing Interface) will be used for parallel data and functional processing.

1. Introduction

The phase-field is a thermodynamics-based method mostly used to model phase changes and microstructure evolution in materials. It is a mesoscopic method, meaning intermediate size. It deals with material systems between the nanoscale and a few micrometers. The variables used can be abstract non-conserved quantities representing a phase or anything (for example, a variable x such that $x = 0$ is a solid, $x = 1$ is a liquid and $x \in (0, 1)$ is an interface between the two), or they can be conserved measurable quantity such as concentration.

The principle variable in such a model, that determines the state of a system is called and order parameter.

1.0.1 Order parameter

The state of a system is captured in a variable, which is most often a scalar field, a continuous function of position and time, and that represents a property of the system such as concentration or phase identity. Such a variable is called an Order parameter, and let's denote it with $\phi(r, t)$. The value of ϕ at each position and time is determined from the free energy of the system. If in addition of energy conservation, $\phi(r, t)$ must itself remain conserved throughout its time evolution, it is called a conserved order parameter, for example the concentration field is conserved by the law of conservation of mass. Parameters like phase identity, grain orientation etc. that do not obey any conservation law are called non-conserved order parameters.

1.1 The Phase Field Model

The model used is the famous work of Kobayashi [1] which is one of the earliest phase-field models for dendritic solidification.

The model includes two order parameters. One is a non-conserved order parameter $\phi(r, t)$ which takes the value of 1 in solid phase and 0 in liquid phase. The other parameter is the temperature field $T(r, t)$. Here r is the spatial position and t is time.

The free energy functional chosen for this model is the Ginzburg-Landau type free energy including m as a parameter :

$$F(\phi, m) = \int_V \frac{1}{2}\epsilon^2 |\nabla\phi|^2 + f(\phi, m)dv; \tag{1.1}$$

where ϵ is a small parameter which determines the thickness of the layer and also controls the mobility of the interface. f is a double well potential (free energy function) that has local minimas at $\phi = 0$ and $\phi = 1$ for each value of m. The specific form of f taken in this model is :

$$f(\phi, m) = \frac{1}{4}\phi^4 + (\frac{1}{2} - \frac{1}{3}m)\phi^3 + (\frac{1}{4} - \frac{1}{2}m)\phi^2 \tag{1.2}$$

where $|m| < \frac{1}{2}$. Anisotropy is accounted for by assuming that ϵ depends on the direction of the outer normal vector at the interface. Its value is calculated as :

$$\epsilon = \bar{\epsilon}\sigma(\theta) \tag{1.3}$$

where $\bar{\epsilon}$ is the mean value and $\sigma(\theta)$ represents anisotropy in the form :

$$\sigma(\theta) = 1 + \delta \cos j(\theta - \theta_0) \tag{1.4}$$

where δ is the strength of anisotropy and j is the mode number of anisotropy which takes the value of 4 for cubic lattices and 6 for hexagonal lattices. θ_0 is the initial offset angle taken as a constant. The angle θ is defined as :

$$\theta = \arctan\left(\frac{\partial\phi/\partial y}{\partial\phi/\partial x}\right) \tag{1.5}$$

The parameter m is assumed to be dependent on the degree of supercooling an the temperature. The dependency is expressed as :

$$m(T) = \frac{\alpha}{\pi}\arctan\left[\gamma(T_{eq} - T)\right] \tag{1.6}$$

where α is a positive constant and T_{eq} is the equilibrium temperature. The Allen–Cahn equation for the evolution of the non-conserved order parameter ϕ takes the respective forms as expressed in equations 1.8 and 1.9.

The temperature field evolution equation is derived from enthalpy conservation and is expressed as :

$$\frac{\partial T}{\partial t} = \nabla^2 T + \kappa\frac{\partial\phi}{\partial t} \tag{1.7}$$

The equation is non-dimentionalized so that the characteristic cooling temperature is 0 and the equilibrium temperature is 1.

The isotropic phase field evolution equation is :

$$\tau\frac{\partial\phi}{\partial t} = \epsilon^2\nabla^2\phi + \phi(1-\phi)(\phi - \frac{1}{2} + m) \tag{1.8}$$

Periodic boundary conditions are assumed on the x axis. On the Y axis Dirichlet boundary conditions are assumed with $\phi = 1$ on top and $\phi = 0$ on bottom.

The phase field ϕ evolution equation for anisotropic solidification is expressed as:

$$\tau\frac{\partial\phi}{\partial t} = \frac{\partial}{\partial y}\left(\epsilon\frac{\partial\epsilon}{\partial\theta}\frac{\partial\phi}{\partial x}\right) - \frac{\partial}{\partial y}\left(\epsilon\frac{\partial\epsilon}{\partial\theta}\frac{\partial\phi}{\partial y}\right) + \nabla.(\epsilon^2\nabla\phi) + \phi(1-\phi)(\phi - \frac{1}{2} + m) \tag{1.9}$$

Dirichlet boundary conditions are applied here.

2. Parallelization

2.1 Need for Parallelization

Parallelization is essential for simulating any material phenomenon. In this case, because solidification is a macroscopic phenomenon and our simulations run in a microscopic scale, parallelization is crucial in scaling up the simulation domain while maintaining computable time. The following points show the need for parallelization in detail:

1. **Improved Computational Efficiency**: Parallelization allows for distributing the computational workload across multiple processors or computing resources, thereby significantly reducing the overall computation time. Solving complex differential equations often involves iterative methods and extensive calculations, and parallelization enables the simultaneous execution of these computations, leading to faster simulations.

2. **Scalability**: Parallelization provides scalability, allowing simulations to be performed on increasingly larger scales. As the complexity of the differential equations and the physical phenomena being simulated grow, parallel computing enables the use of more computing resources to handle the increased computational demand. This scalability is particularly crucial for simulating complex physical phenomena accurately.

3. **Handling High-Dimensional Problems**: Many physical phenomena involve systems with high-dimensional state spaces, resulting in a large number of differential equations to solve simultaneously. Parallelization allows for dividing these equations into smaller subsets that can be solved concurrently, reducing the computational burden associated with high-dimensional problems. It enables efficient utilization of computational resources and enables the simulation of complex systems that would be otherwise computationally infeasible.

4. **Parameter Exploration and Sensitivity Analysis**: Parallelization facilitates conducting parameter explorations and sensitivity analyses efficiently. By running multiple simulations simultaneously with different parameter values, researchers can study the behavior and sensitivity of the physical system to various factors. This capability is particularly valuable in scientific research, engineering design, and optimization, enabling the identification of critical parameters and understanding system behavior across a range of conditions.

2.2 Implementation

We have found that data parallelization is the best way to solve the Solidification Equation 1.9. A functionally parallel implementation would have had synchronization and data-sharing overheads, thus reducing computational efficiency.

Data parallelization is implemented as such :

1. **Domain Decomposition** : The computational domain is divided into subdomains of equal size (domain_size/no_of_processors) along the Y-axis.

2. **Assignment** : Each subdomain is assigned to only one processor(rank). Multiple subdomains assigned to a single processor would just have increase the communication overhead.

3. **Orchestration** : The subdomain boundaries along the division axis are synced after every iteration to maintain the virtual continuity in boundary conditions.

4. **Mapping** : Subdomains are linearly mapped to the processors(ranks) and the compute instructions for each processor is the same.

Overall, the implementation is similar to, Single-instruction-multiple-data.

2.3 Pseudo-code

The generalized pseudo-code is presented here

```
DOMAIN_SIZE_X = S
DOMAIN_SIZE_Y = S

SUBDOMAIN_SIZE_X = S + 2
SUBDOMAIN_SIZE_Y = S/size + 2
## size = no. of Processors
## +2 to add two extra rows/columns for boundary conditions

Initialize(SUBDOMAIN_SIZE_X, SUBDOMAIN_SIZE_Y)
MPI_Barrier()

time_start = time()
for iter < TOTAL_ITERATIONS :

    for x = 1 to SUBDOMAIN_SIZE_X - 2 :
    for y = 1 to SUBDOMAIN_SIZE_Y - 2 :
        solve_PDE(x,y)

    Sync_boundaries_of_subdomains()

    MPI_Barrier()
time_end = time()
output(time_end-time_start)
Write_output()
```

3. Speedup & Efficiency

The simulations have been run on a 64-core machine for varying domain sizes and the corresponding plots are shown below.

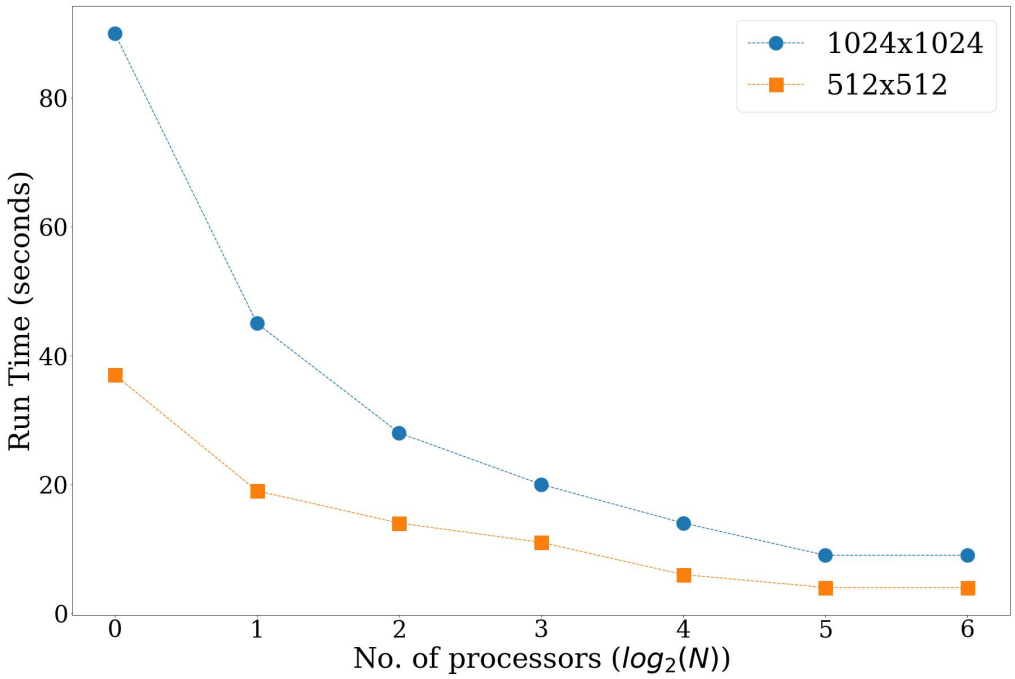

Figure 3.1: A plot of runtime v/s no. of processors for varying domain size.

3.1 Speedup

Speedup is a metric used in high-performance computing (HPC) to evaluate the improvement in computational performance achieved by parallelizing a program. It measures how much faster a parallel execution of a task is compared to a sequential execution on a single processor or a computing cluster.

Mathematically, speedup is defined as the ratio of the execution time of the sequential version ($T(1)$) to the execution time of the parallel version ($T(N)$) for the same task:

$$S(N) = \frac{T(1)}{T(N)}$$

A speedup value greater than 1 indicates that the parallel version is faster than the sequential version, while a value less than 1 implies that the parallel version is slower. The higher the speedup value, the more efficiently the parallel execution utilizes the available computing resources.

The speedup achieved by parallelization depends on various factors, including the domain size, the level of parallelism, the communication and synchronization overhead, and the efficiency of the parallel algorithm. Ideally, a well-optimized parallel program should exhibit near-linear speedup, meaning that the execution time decreases linearly with the increase in the number of processors.

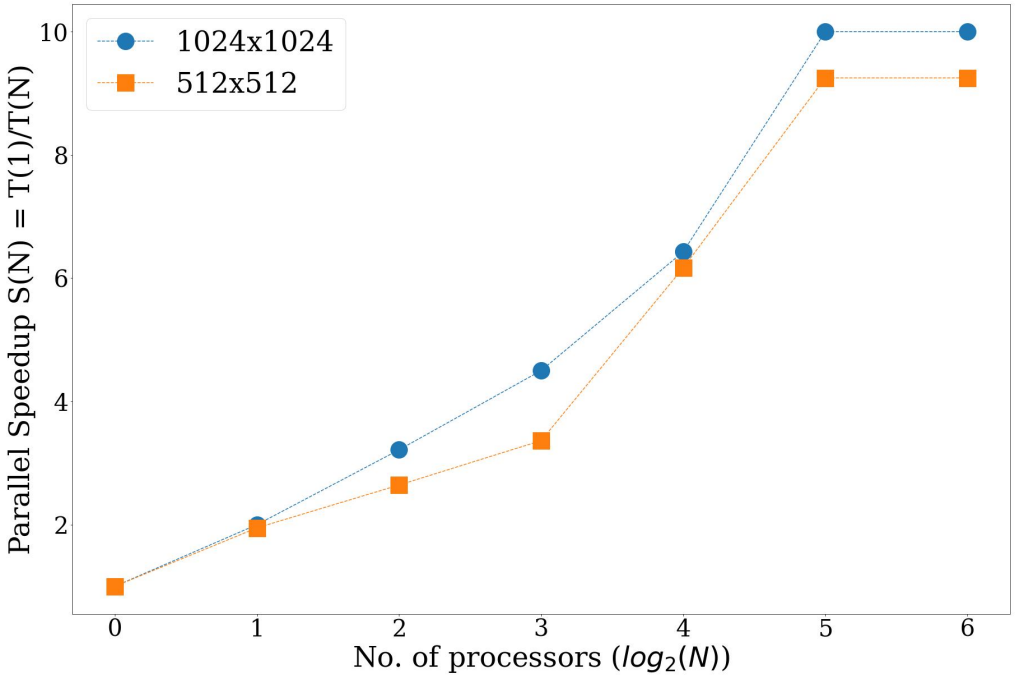

Figure 3.2: A plot of Speedup ($S(N) = \frac{T(1)}{T(N)}$) v/s no. of processors for varying domain size.

3.2 Efficiency

Efficiency refers to the effectiveness with which computational resources are utilized to solve a given problem or perform a specific task.

Efficiency is often expressed as a percentage and is calculated using the speedup ($S(N)$) and the number of processors (N) involved in the parallel execution. The formula for efficiency (ϵ) is as follows:

$$\epsilon(N) = \frac{S(N)}{N} * 100$$

Efficiency quantifies the degree to which the parallel execution of a program scales with the number of processors used. Higher efficiency values indicate that a parallel program effectively utilizes the available resources, while lower values imply that there is a significant amount of overhead or underutilization of the computing power.

Several factors can impact the efficiency of parallel computing, including load balancing, communication overhead, synchronization costs, and the inherent parallelism of the problem being solved. Achieving high efficiency requires careful consideration of these factors during the design and implementation of parallel algorithms and architectures.

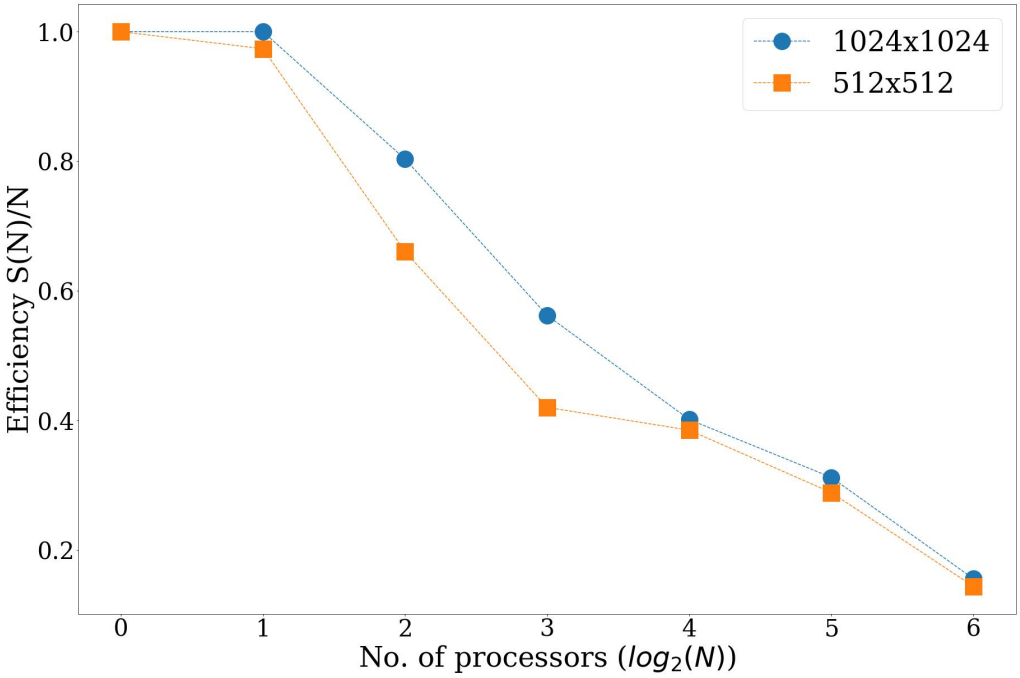

Figure 3.3: A plot of Eficiency ($\epsilon(N) = \frac{S(N)}{N}$) v/s no. of processors for varying domain size.

4. Results

The following figure shows a 2D plot of the phase field run with a domain size of 1024x1024 cells for a total of 2000 iterations. The physical parameters have the corresponding mentioned values. $\epsilon = 0.01$, $\alpha = 0.92$, $\gamma = 0.92$, $\delta = 0.6$, $J = 6$, $\tau = 3.0 \cdot 10^{-4}$, $\kappa = 1.32$.

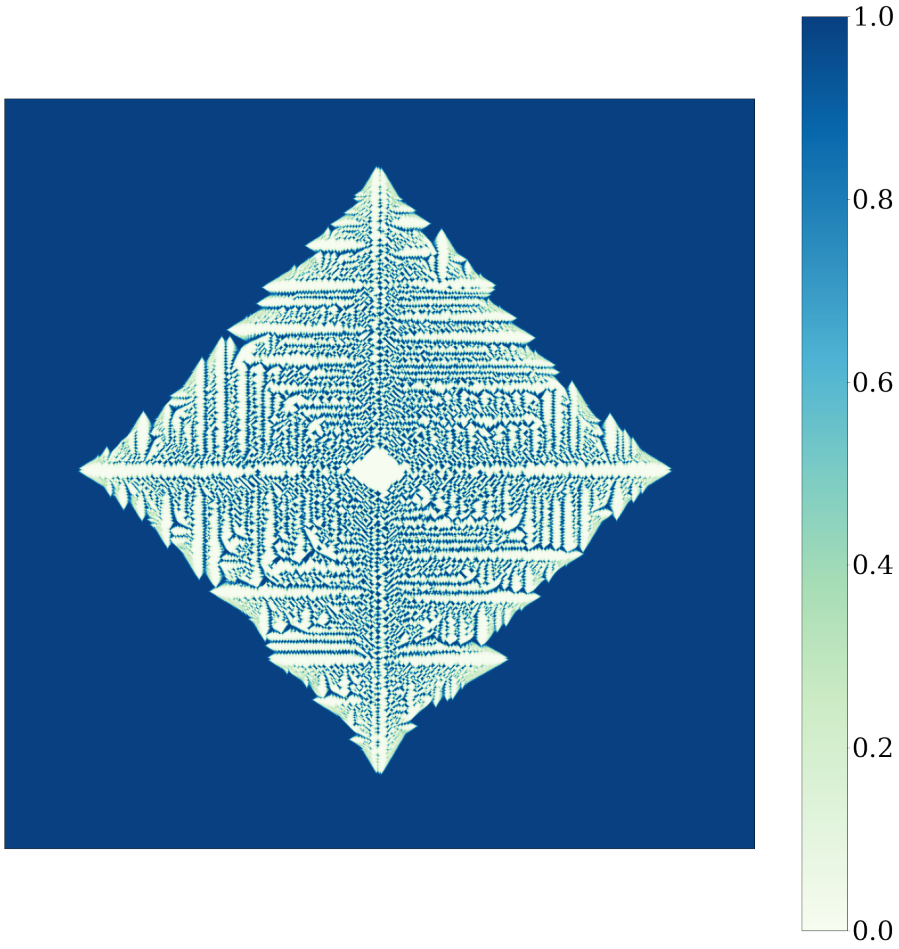

Figure 4.1: A 1024x1024 simulation of anisotropic solidification for 2000 iterations. The dendritic structure is clearly visible.

5. Conclusion

The following conclusion can be drawn from the speedup and efficiency plots :

- **Good Scalability**: The near linear speedup indicates that the program scales well with the increasing number of processors. As N increases, the run-time decreases almost proportionally. This suggests that the program effectively harnesses the available resources and efficiently distributes the workload among the processor
- **Minimal Communication Overhead**: The communication overhead is well-managed. Even while running on larger number of processors, decreasing subdomain size has not increased the run-time via communication overheads along the subdomain boundaries.
- **Inherent Sequential Dependencies**: The entire program is not completely parallelized. Because we only can parallelize each iteration individually. The barriers between subsequent iterations still cause a hindrance.

Several conclusions can be drawn from the figure 4.1.

- **Dendritic Growth**: The simulation confirms that dendritic growth occurs during solidification. Dendrites are tree-like structures that form as a result of crystal growth from the solid-liquid interface. The simulation demonstrates the characteristic branching and elongation of dendrites, indicating the growth of preferred crystallographic orientations.
- **Anisotropic Solidification**: The anisotropic nature of the simulation implies that the solidification process is not uniform in all directions. The growth of dendritic structures is influenced by the local conditions, such as temperature gradients and solute diffusion, resulting in directional variations in crystal growth. This anisotropy is evident in the shape and orientation of the dendritic arms.
- **Microstructural Evolution**: The simulation provides insights into the evolution of microstructures during solidification. Dendrites start as small seed crystals and grow by the preferential advancement of certain crystallographic planes. As the solidification progresses, neighboring dendrites interact and compete for growth space, leading to changes in their shapes and orientations. The simulation captures the complex interplay between dendritic growth, branching, coarsening, and coalescence.
- **Interface Morphology**: The simulation reveals the morphology of the solid-liquid interface during dendritic solidification. The interface is not smooth but rather rough and irregular due to the dendritic growth. The anisotropic nature of dendritic growth results in varied interface curvatures and local solidification rates. This unevenness contributes to the formation of complex microstructures and the presence of dendritic arms of different sizes and orientations.
- **Solidification Conditions**: The simulation provides information about the solidification conditions necessary for dendritic growth. Factors such as undercooling

(the temperature difference between the liquidus and the actual solidification temperature) and solute concentration gradients play a significant role in influencing the dendritic morphology and growth kinetics. The simulation outcomes can aid in understanding the relationship between processing parameters and microstructural features.

Bibliography

[1] Ryo Kobayashi. Modeling and numerical simulations of dendritic crystal growth. *Physica D: Nonlinear Phenomena*, 63(3-4):410–423, 1993.