Korean Journal of Chemical Engineering (2024) 41:2423–2432
https://doi.org/10.1007/s11814-024-00146-w

ORIGINAL ARTICLE

Performance Benchmark of Cahn–Hilliard Equation Solver with Implementation of Semi-implicit Fourier Spectral Method

Ilhyun Cho[1] · Jeonghwan Lee[1] · Kunok Chang[1]

Received: 15 January 2024 / Revised: 15 January 2024 / Accepted: 1 March 2024 / Published online: 12 March 2024

Abstract
The performance scaling issue of phase-field simulation is one that must be overcome to perform realistic large-scale three-dimensional prediction. The CUDA (Compute Unified Device Architecture) parallel acceleration method developed over a decade ago showed very good performance in terms of calculation speed, but was limited by the small size of memory on the GPU. Recently, Apple Inc. has announced a GPU–CPU hybrid architecture, Apple silicon (M1 or later), and we examine the advantages of this architecture for performing realistic large-scale phase-field simulations and compare it to existing CUDA architecture. When solving the Cahn–Hilliard equation using the FFT (Fast Fourier Transform) with CUDA architecture developed by Nvidia and Apple silicon architecture developed by Apple Inc., we compared performance across hardware, as well as other considerations such as form factor and heat dissipation of the workstation.

Keywords Parallel computing · Performance scaling · Phase-field modeling

Introduction

Predicting the microstructure of a specific domain is a very important engineering task, and many important properties depend on the microstructure [1–3]. For example, the spacing between dendrite arms on the electrode surface of a Li-ion battery is a key variable that determines battery performance [3]. The phase-field method has been developed and popularized by leaps and bounds in the last few decades due to its ability to predict microstructures by taking into account the thermodynamic and kinetic properties of materials [4, 5], as well as many other multiphysical interactions, such as elastic effects [6] and electrostatic effects [7]. Phase-field methods have dramatically improved computational performance and tractability by implicitly storing information about the interface, whereas many numerical methods have to explicitly track the boundaries of the domain [8–11].

Many numerical techniques are applied to solve the Allen–Cahn (Ginzburg–Landau) and Cahn–Hilliard equations in the phase-field method, such as the Finite Difference (FD), Finite Element (FE), Finite Volume (FV), and Semi-Implicit Fourier Spectral (SIFS) method. Especially, the SIFS method is a popular method for solving the Cahn–Hilliard equation because it is known to dramatically increase the numerical stability of the fourth-order Cahn–Hilliard equation [12, 13].

Despite the numerically superior stability of the SIFS method, the need to repeatedly perform the Discrete Fourier Transform (DFT) and inverse-DFT (iDFT) at each iteration was a significant burden. As the need to improve the performance of numerical computations has grown, various parallelization techniques have evolved, including CPU parallelism techniques such as MPI [14] and OpenMP [15], and GPU parallelism techniques such as CUDA [16] have become very popular since the 2010s. In particular, cuFFT, a DFT library developed for CUDA [17], and vkFFT, a library developed for Apple silicon that supports CPU-GPU hybrid parallelization [18], are expected to be important breakthroughs for SIFS, whose scalability has traditionally been limited by the time and resources required for DFT.

In the case of GPU parallelization, tens or hundreds of thousands of cores are used, which is very advantageous in terms of computational speed, but it has the limitation that memory is relatively limited [19]. Also, since there are many cores, the amount of heat generated is very large, so the size of the machine becomes larger and the intensive cooling system is needed.

Kunok Chang
kunok.chang@khu.ac.kr

[1] Department of Nuclear Engineering, Kyung Hee University, Yongin, Korea

There were clear limitations to scaling the physical memory of the GPU, therefore, optimizations were needed to reduce the latency of CPU-GPU memory communication. This can be achieved by reducing communication overhead, such as optimizing load distribution and ensuring parallel scalability [20]. To this end, research on the optimization of parallelization is being conducted in the direction of solving memory size limitations or reducing memory latency. The asynchronous method, which can interpret larger memory sizes more than the actual GPU memory has been implemented [19]. Additionally, CUDA unified memory which allows for accessing both CPU and GPU memory as a single memory using pointer has been applied to increase compute density [21], optimizing GPU memory access pattern to interpret the ultrafast magnetization dynamics model [22], and utilizing the low-latency shared memory within GPU as a user-managed cache to analyze the dendrite growth of nickel-based superalloys [19]. These methods alleviate memory size limitations and memory overhead, but they either result in reduced performance or require complex algorithms to address data latency.

Recently, a Heterogeneous architecture has emerged, integrating CPU, GPU and memories of each. Both CPU and GPU access the physically same memory, there is no need for separate data transfer processes. With these advantages, the Apple silicon architecture is reported to perform notably efficiently [23, 24]. While Nvidia GPUs leverage CUDA for parallelization, Metal framework is used for Apple silicon GPU parallel implementation. Metal framework offers functionalities with high-performance features. To utilize the full capability of the Metal framework, an optimization method is considered.

Herein, we developed a GPU parallel solver for the semi-implicit scheme of the Cahn–Hilliard equation. Performance benchmarks were conducted on Nvidia GPU using CUDA and compared the SoC (System on a Chip) architecture developed by Apple Inc., utilizing the Metal framework.

We compared the performance of parallelization using the Metal Framework incorporated into SoC of Apple with that of Nvidia GPU parallelization. We presented various benchmarks that can serve as a reference for future development of high-performance software for engineering including computational materials science.

Methods and Details

Semi-implicit Fourier Spectral Method

We simulated the spinodal decomposition, an example can be described by Cahn–Hilliard equation [8] in Eq. (1).

$$\frac{\partial c(\mathbf{r},t)}{\partial t} = \nabla \cdot \left[M(\mathbf{r},t) \cdot \nabla \left(\frac{\delta F(\mathbf{r},t)}{\delta c} \right) \right] \tag{1}$$

where the free energy of the system is expressed as Eq. (2).

$$F(\mathbf{r},t) = \int_V \left\{ \frac{1}{V_m} \left[f(c) + \frac{1}{2}\kappa (\nabla c)^2 \right] \right\} dV \tag{2}$$

Where c is the concentration of the solute, $F(\mathbf{r},t)$ is the free energy of the system, and $f(c)$ is the chemical free energy, κ is the gradient energy coefficient, and M is the mobility.

$$\frac{\partial c(\mathbf{r},t)}{\partial t} = M\nabla^2 \left[\frac{\delta f(c)}{\delta c} - \kappa \nabla^2 c(\mathbf{r},t) \right] \tag{3}$$

Implementing Fourier spectral scheme [25],

$$\frac{\partial \tilde{c}(\mathbf{k},t)}{\partial t} = -Mk^2 \left[\left\{ \frac{\delta f(c)}{\delta c} \right\}'_k + \kappa k^2 \tilde{c}(\mathbf{k},t) \right] \tag{4}$$

Where $\mathbf{k} = (k_l, k_2, k_3)$ denotes the reciprocal vector in Fourier space, $\tilde{c}(\mathbf{k},t)$ and $\{\frac{\delta f(c)}{\delta c}\}'_k$ are the Fourier transforms of $c(\mathbf{r},t)$ and $\frac{\delta f(c)}{\delta c}$, respectively. Implementing Semi-implicit scheme [25],

$$\frac{\tilde{c}^{n+1}(\mathbf{k},t) - \tilde{c}^{n}(\mathbf{k},t)}{\Delta t} = -Mk^2 \left[\left\{ \frac{\delta f(c)^n}{\delta c} \right\}'_k + \kappa k^2 \tilde{c}^{n+1}(\mathbf{k},t) \right] \tag{5}$$

$$\tilde{c}^{n+1}(\mathbf{k},t) = \tilde{c}^{n}(\mathbf{k},t) - \frac{Mk^2 \Delta t \frac{\delta f(c)^n}{\delta c}{}'_k}{1 + \kappa k^4 \Delta t} \tag{6}$$

Benchmark problems were proposed by Jokisaari et al. [26] for phase-field simulation, and a simple polynomial form of the free energy is applied. We applied a double-well potential to $f(c)$, which is a second-order polynomial function, where c has a concentration range between 0 and 1, and the coefficients for mobility and boundary energy were set to $M = 1$ and $\kappa = 1$, respectively.

$$f(c) = c^2(c-1)^2 \tag{7}$$

$$\tilde{c}^{n+1}(\mathbf{k},t) = \tilde{c}^{n}(\mathbf{k},t) - \frac{Mk^2 \Delta t \{2(c-1)(2c-1)\}'_k}{1 + \kappa k^4 \Delta t} \tag{8}$$

Metal API

GPU architecture in parallel programming is characterized by its ability to leverage a multitude of cores for simultaneous operations. However, it is less efficient in executing sequential tasks. To overcome this limitation, CPU intervenes and invokes parallel operations through kernel functions or shader functions executed on the GPU. Rather than executing multiple kernels concurrently, CPU schedules them one at a time in a sequential manner. This ensures orderly execution while harnessing the power of parallel computing. When utilizing Apple silicon for parallel computing purposes, developers must employ Metal API (Application Programming Interface), which allows for hybrid GPU–CPU parallelism [27]. To write efficient kernel functions with Metal API, programmers are required to utilize the C++-style Metal Shading Language (MSL) [28]. It should be noted that Metal API exclusively supports Objective-C and Swift languages; however, it does offer support for interfacing with C++ language as well. In addition to facilitating effective utilization of GPUs via Metal API, Apple Inc. provides MPS (Metal Performance Shader) Library-a valuable resource containing preeminent mathematical operations that are highly optimized for enhanced performance during computation processes.

Figure 1 shows the schematic execution model of Metal API.

Parallel operations are executed with a unit work called Grid, which divided into subsets called threadgroups. Threadgroup consists of threads. The threads in a threadgroup are also composing SIMD (single-instruction, multiple data) groups. Metal API concurrently executes kernels through SIMD threads, which resembles warp of CUDA. For all Apple devices, each SIMD consists of 32 threads. The number of threadgroups affects the efficiency and performance of the code. However, the optimal threadgroup sizes depends on many factors, such as the size of system and the complexity of the code. In this study, we implemented optimization techniques to efficiently solve the problem using the SIFS scheme. Specifically, we employed a thread group size adjustment strategy, which involves adjusting the number of threads and thread groups.

Threadgroup Size Adjustment Strategy

To fully utilize potential of modern GPU, maintaining workload balance is considered important between heterogeneous cores [29]. Fang et al. [30] suggested changing sizes of GPU kernel threadblock improves computational performance. They achieved at least 20% reduced execution time at their benchmarks.

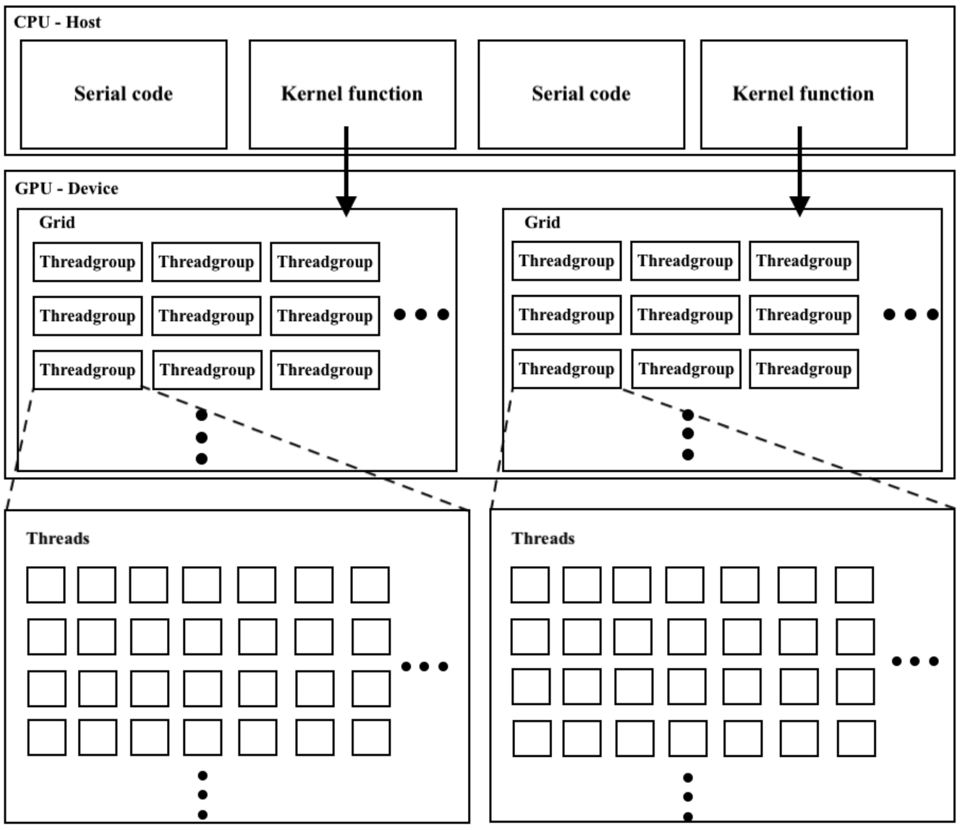


Fig. 1 Metal API execution model for CPU–GPU hybrid architecture

Similar with CUDA, Metal kernels are executed through designated threadgroup numbers during kernel's execution. Therefore, we adjust threadgroup sizes. For Metal API, the number of threadgroup size is up to 1024. Given that

$$\text{grid} = \text{threadgroup} \times \text{thread}$$

increasing the number of threads will enhance performance because there will be more concurrent operations. However, since grid is constrained, the number of threadgroups will decrease. Since threadgroups are distributed to GPU cores to process tasks, It is important that threadgroups are evenly distributed across cores. A small number of threadgroups can somewhat decrease performance due to imbalance of cores.

In this study, the system dimension was divided into 1^2, 2^2, 4^2, 8^2, 16^2, and 32^2 to explore the optimal threadgroup size for the system dimensions of 1024^2, 2048^2, and 4096^2, and the performance depending on the threadgroup size is shown in Fig. 2.

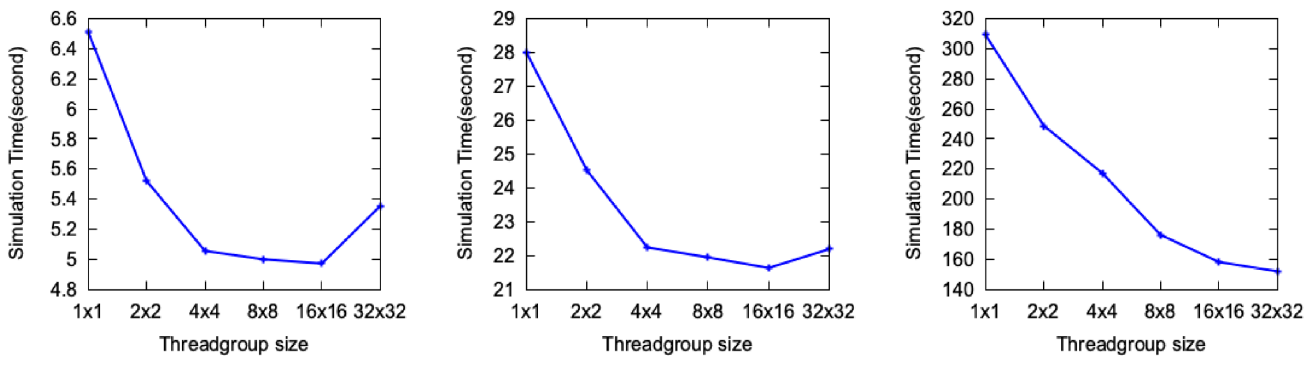
We also investigated optimal threadgroup size and performance improvement percentage(%) for each system dimension shown Table. 1. The parallel performance was observed to degrade for thread group sizes of 1^2 and 2^2 due to their small sizes. Therefore, we focused on performance improvement over 4^2 threadgroup sizes. The result shows that the strategy greatly affects runtime performance over 2048^2 system dimension. The underlying reason lies in the fact that the Cahn–Hilliard equation, implemented with the SIFS method, involves both FFT (Fast Fourier Transform, an optimized Discrete Fourier Transform) operations and non-FFT operations. In terms of performance improvement, the efficiency of the FFT function call is determined by the external library. However, The efficiency of non-FFT operations can be optimized by adjusting the thread group size. The vkFFT library utilized in this study exhibits peak efficiency at a system dimension of 2048^2, and performance decreases thereafter. Therefore, the performance improvement in non-FFT operations becomes more important for system dimensions larger than 2048^2, leading to a 36.5% performance improvement at 4096^2 system dimension.

Table 1 Optimal threadgroup size and execution time reduction percentage(%) compared with 1^2 threadsize

System dimension	1024^2	2048^2	4096^2	8192^2
Optimal threadgroup size	16^2	16^2	32^2	32^2
Performance improvement (%)	7.1	2.7	36.5	30.0

Results & Discussion

We developed code to solve three versions of three-dimensional Cahn-Hillard equations with SIFS using cuFFT for the CUDA-accelerated version (hereafter CUDA-version code) and vkFFT for the Metals-accelerated version (hereafter METAL-version code). We also developed additional set of code of serial version (hereafter SERIAL-version code). For METAL-version code, a set of simulations were carried out on Apple silicon, and for CUDA-version code, we ran the calculations on a custom workstation utilizing an Intel CPU and NVIDIA GPU built by a professional builder, and compared computational performance and power consumption. For the performance benchmark, we measured the physical time it took to perform the calculations on each workstation. Also, execution time including data transfer between CPU and GPU is recorded and compared for CUDA-version code and METAL-version code. We have verified that the METAL-version code and CUDA-version code produce



(a) 1024^2 system dimension (b) 2048^2 system dimension (c) 4096^2 system dimension

Fig. 2 The required computation time for simulating the given system dimensions, namely **a** 1024^2 system dimension, **b** 2048^2 system dimension, and **c** 4096^2 system dimension, is examined as a function of the threadgroup size in the Metal kernel

the same results based on the time-tested accuracy of the SERIAL-version code.

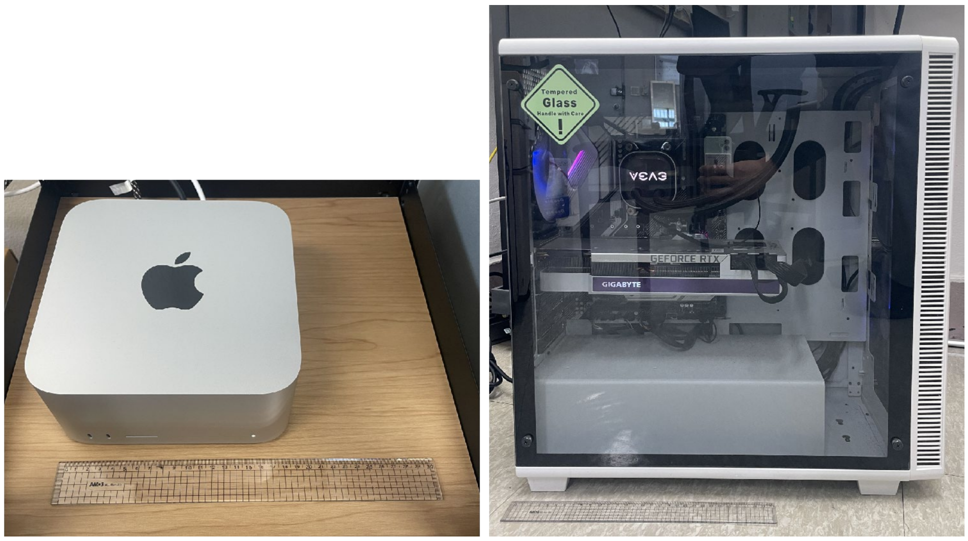
Physical Size and Power Consumption of the Workstation

We utilized workstations of various form factors for this study. Workstations with Nvidia GPUs are typically very large and heavy, and the size of the GPU is getting bigger and bigger for cooling efficiency. For convenience, we'll refer a to workstation with Nvidia GPU as a NVIDIA-workstation. However, with an integrated and optimized design, Apple silicon achieves not only small in size but also powerful performance. In Fig. 3, we can see the difference in size between a high-performance computer with an Apple M2-Ultra and a workstation with an RTX-3090Ti. Mac Studio (installed M2-Ultra chipset), is much smaller in size compared with NVIDIA-workstation. Moreover, compared with about 20 kg weighted workstations, Mac studio only weighs 3.6 kg. Details of the hardware used in each implementation can be found in the Table 2. Comparing TDP, which is power consumption at maximum theoretical load, the Mac studio (M2 Ultra, 60W) requires about 1/8 power of the Workstation (RTX3090Ti, 450W). Less power consumption means a smaller cooling system and less heat generation, which has many advantages for workstation operation.

Affordable System Size

Performing large three-dimensional simulations requires a lot of computational resources, and GPU memory is usually not enough for this task. Of course, multi-GPUs can be utilized to expand the available memory size, but there is a problem of scaling due to transmission between GPUs, and there are limitations that make it difficult to implement technically. For example, based on previous methods such as Multi-GPU or asynchronous method, are not satisfactory for high bandwidth compared with VRAM of single GPU. Also, technical implementation of multi-GPU strategy is relatively challenging because it costs a lot and asynchronous method follows an additional algorithm which could deteriorates runtime performance. Apple silicon applied unified memory which either CPU and GPU can occupy physically same memory. Therefore, there is no discrete VRAM for Apple silicon and GPU memory could be very large.

Instead, Apple silicon GPU have maximum allocatable memory, without affecting its runtime performance. Each GPU allocatable memory (VRAM) for Apple silicon and Nvidia GPU is investigated in Tables 3 and 4. Result shows that VRAM of M2 Ultra (96 GB) is 4 times bigger than VRAM of single RTX-3090Ti (24 GB), about 6 times bigger than VRAM of Tesla V100 (16 GB). Therefore,



(a) Mac Studio(M2 Ultra) (b) Nvidia GPU Workstation(RTX3090)

Fig. 3 Comparison of workstation dimension with **a** Mac studio (M2 Ultra) and **b** Nvidia GPU workstation (RTX3090)

Table 2 Hardware features of the workstation utilized in this study

	M1 Pro	M2 ultra	RTX2070	RTX3090Ti	RTX4090	Tesla V100
Memory [GB]	16 (unified)	128 (unified)	8	24	24	16
Memory bandwidth [GB/s]	200	800	448	1008	1008	900
TDP [W]	30	60	175	450	450	300

Table 3 Maximum affordable GPU memory for Apple GPUs

Apple Inc	M1 pro	M2 Ultra
RAM [GB]	16	128
VRAM [GB]	10.9	96
Percentage [%]	68	75

Table 4 Maximum affordable GPU memory for Nvidia GPUs

Nvidia	RTX 2070	RTX 3090Ti	Tesla V100
VRAM [GB]	8	24	16

Table 5 Memory required for two-dimensional systems operating Cahn–Hilliard equation solver

Two-dimensional	4096^2	8192^2	16384^2	32768^2
Buffer size	64 MB	256 MB	1.02 GB	4.09 GB
Metal	466 MB	1.77 GB	7.01 GB	28.10 GB
CUDA	828 MB	2.40 GB	8.77 GB	–

Apple silicon have an advantage for larger systems, without those GPU memory running out problems.

Solving the Cahn–Hilliard equation requires the initial value of conserved order parameter. We investigate systems that sized power of two at each axis. Therefore, initial data file sizes could be substantial. For example, the 4096^2 system size has about 64 MB size, and the 32768^2 system size has about 4 GB size at a single precision accuracy in two-dimensional system. These sizes vary depending on the extension of the file or the accuracy required for corresponding research.

However, GPU parallelism requires much more memory than the initial data file size. The data for parallel computation is stored in a component called a buffer, which GPU is able to access. However, it is recommended to use separate buffers for different operations. This is because in parallel operations, different operations modify the same data at the same time, creating a race condition. Therefore, multiple buffers are required for parallel computing in terms of data integrity. The amount of memory required can vary depending on the environment, such as operating system and programming language. Memory in GPU required for solving the Cahn–Hilliard equation is investigated respectively: two-dimensional system for Table 5, three-dimensional system for Table 6.

Maximum affordable system sizes for each devices are decided by VRAM and Memory requirements. For example, 16384^2 system requires 8.77 GB VRAM for CUDA. Consequently, RTX2070 cannot run CUDA code at 16384^2 system because VRAM (8 GB) is not sufficient. The maximum

Table 6 Memory required for three-dimensional systems operating Cahn–Hilliard equation solver

Three-dimensional	128^3	256^3	512^3	1024^3
Buffer size	8 MB	64 MB	512 MB	4.09 GB
Metal	81 MB	418 MB	3.04 GB	28.10 GB
CUDA	89 MB	826 MB	–	–

affordable system sizes of each hardware are shown in Table 7. Comparing affordable maximum system sizes, M2 Ultra (96 GB VRAM) can afford 32768^2 and 1024^3 system sizes, while RTX4090 (24 GB VRAM) only afford 16384^2 and 256^3 system size. Also, M1 Pro (10.9 GB VRAM) can afford same comparable sizes with RTX4090, RTX3090Ti. However, despite having sufficient memory (24 GB VRAM) on both RTX4090 and RTX3090Ti, the CUDA code fails to run at a system size of 512^3. This discrepancy is notable since the code operates successfully on a system size of 16384^2, which has smaller dataset, suggesting that the issue may not be related to memory size or VRAM. Two discrete error message are emerge—one indicating a memory access issue and the other signaling with insufficient memory. Specifically, at a system size of 512^3, the error message is as follows: 0: DEV_MKDESC: allocate FAILED:700(an illegal memory access was encountered). Conversely, at a system size of 1024^3, the error message is as follows: 0: ALLOCATE: 4303355904 bytes requested; status = 2 (out of memory) While dividing the data for computation might offer a solution, this may require additional algorithms and result in communication overhead. The operating system and its version are Ubuntu 20.04.6 LTS.

Performance Benchmark

The performance benchmark of CPU and GPU codes implemented to each device is evaluated.

In Fig. 4, we compared the binary execution time between two devices (METAL-workstation with M1 Pro and NVIDIA-workstation with RTX2070). For the M1 Pro, we saw an overall lower performance gain compared to the RTX2070. This is likely due to lower bandwidth (200 GB/s) compared to that of RTX2070 (448 GB/s). It is shown that performance gain of CUDA steadily increases along the system size, while Metal's performance gains are somewhat stagnated after 2048^2 system size. However at particular 2048^2 system, Metal's simulation time were faster than that of CUDA's.

In Fig. 5, we displayed the binary execution time between on workstation with M2-Ultra and workstations with RTX4090 and RTX 3090Ti, respectively. At $256^2, 512^2$ system dimensions, two CUDA-workstations result in faster simulation time. However, at $1024^2, 2048^2, 4096^2$ system

Table 7 Affordable maximum system size for Cahn–Hilliard equation solver for each devices

	M2 Ultra	M1 Pro	RTX 4090	RTX 3090Ti	RTX 2070
VRAM [GB]	96	10.9	24	24	8
Three-dimensional system	1024^3	512^3	256^3	256^3	256^3
Two-dimensional system	32768^2	16384^2	16384^2	16384^2	8192^2

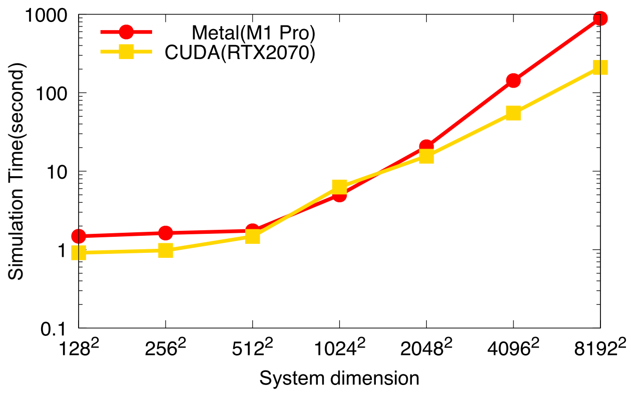


Fig. 4 Simulation time elapsed at each system dimension for Cahn–Hilliard equation solver in M1 Pro, RTX 2070

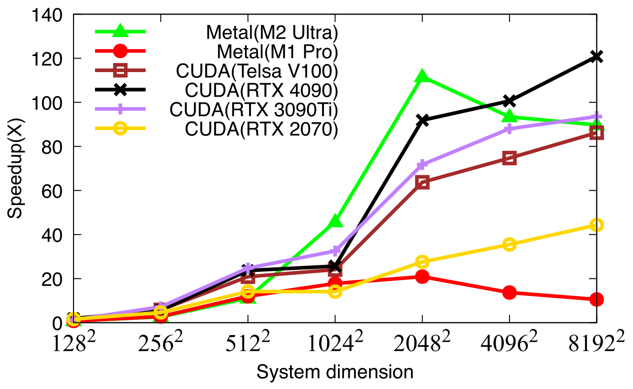


Fig. 6 Computational performance for two-dimensional simulations at each system dimension with different hardware configurations

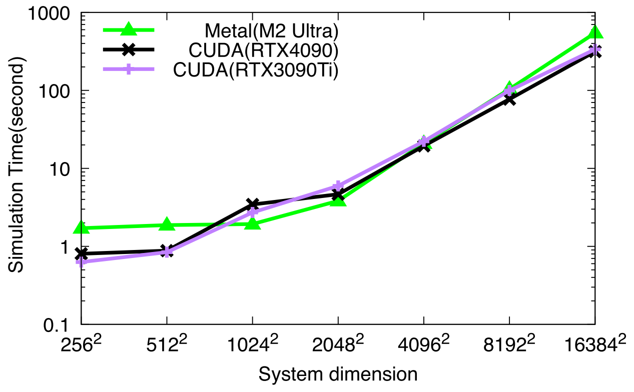


Fig. 5 Simulation time elapsed at each system dimension for Cahn–Hilliard equation solver in M2 Ultra, RTX 4090, RTX 3090Ti

dimensions, the METAL-workstation shows superior computational efficiency. However, 8192^2, 16384^2 system dimension result in CUDA-workstation is again superior. Overall, the performance of the phase-field simulation solver with SIFS on the M2-Ultra and RTX-3090Ti is similar. However, the trend of which architecture is more dominant changes slightly depending on the size of the system dimension.

In Fig. 6, the speedup of every device is plotted. Speedup on the y axis represents the ratio of GPU parallel code execution time to CPU code execution time. This means the larger the speedup, the greater the performance of each GPU. Theoretically, GPU parallelization achieves scalability with increasing data size compared to CPU.

However, GPU performance is decided by number of factors. There are two limiting factors for performance depending on the conditions: bandwidth and number of cores. Bandwidth decides how much data is transferred every second. Number of cores implies the amount of data a device can work at once. At Fig. 6, M1 Pro(200 GB/s), RTX2070(448 GB/s) shows lower performance increase over 1024^2 dimension, compared with larger bandwidth devices(i.e., M2 Ultra(800 GB/s), RTX3090Ti(1008 GB/s), RTX4090(1008 GB/s))

Nevertheless, higher bandwidth does not always translate to performance gains. This is because, when a sufficient amount of data is transferred, the GPU's ability to process it all at once becomes crucial. In addition, even though the RTX 4090 and the RTX 3090Ti have the same bandwidth, significant performance gains are observed on the RTX 4090 (16,384 CUDA cores) compared to the RTX 3090Ti (10752 CUDA cores).

Moreover, Metal code shows decreasing speedup after 2048^2 dimension for M1 Pro and M2 Ultra. It is thought to be the inefficiency of Fourier transform, which takes a large part of Cahn–Hilliard solver. Therefore, a larger speedup is expected for larger system sizes if an optimized FFT (Fast Fourier Transformation) function is provided in Metal. Despite those drawbacks, it is remarkable that speedup of M2 Ultra is comparable to or better than latest GPUs of Nvidia.

Data Transfer

Data transfer between the CPU and GPU is essential for GPU computation. At the beginning of parallel operation, CPU data is sent to the GPU and copied. To check the result, it must be sent back to the CPU and copied. Therefore, there are problems with memory management due to copying and performance degradation due to low bandwidth between CPU and GPU. Research has been done to solve these problems software-wise, but in the end, CPU-GPU data transfer is unavoidable. However, using unified memory architecture eliminates the need for memory copying and also reduces data latency.

We investigated the execution time of Cahn–Hilliard equation SIFS solver, including data transfer between CPU and GPU with CUDA-version code and METAL-version code. Simulations were performed on 2048^2 system dimension until 10,000 time step(iterations). Every 10 time step, we wrote the result file as the text file.

Table 8 compares the time spent transferring data at each stage of the computation. We are looking at performance degradation due to CPU-to-GPU data transfer. Therefore, we separately measured data output time, which is determined by CPU performance. Since write time is dependent on CPU performance, the AMD Ryzen9 3900X (3.8 GHz) with a higher base clock took less time than the M1 Pro (2.0 GHz). We expected the CPU-GPU data transfer time in CUDA (RTX2070 + AMD Ryzen9 3900X) to be high, but it only took 1.24 s for 1000 times data outputs. Metal (M1 Pro) did not require any data transfer time. However, to output the data during the continuous parallelization process, we used the method of synchronization at the end of each iteration to read the result directly. This resulted in a time delay of about 7 s compared to the original execution time of M1 Pro (20 s). These results are puzzling, as Apple silicon has a hardware advantage. Presumably, despite using slower PCIe, we believe that many optimized memory management methods of CUDA has some advantage.

Morphological Assessment

All set of codes used the same initial concentration. Since we applied Eq. 7 as a double-well potential, as shown in Fig. 7, the second-order derivative is negative in the concentration range from about 0.23 to 0.78. Figure 8 illustrates spinodal decomposition with an initial concentration of 0.23, while Fig. 9 depicts the coarsening phenomenon when the initial concentration is set to 0.5.

Table 8 Comparing execution time including data transfer between CPU-GPU at RTX2070 and M1 Pro

	Run time [sec]	Write time [sec]	Data transfer time [sec]	Total time [sec]
RTX 2070	15.4	387.9	1.24	403.4
M1 Pro	27.0	609.9	0	637.0

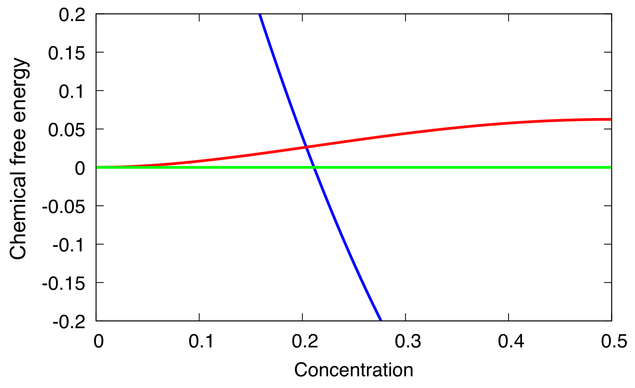


Fig. 7 Plot of the chemical free energy function and it's second derivative where spinodal decomposition appears

Also, three-dimensional systems are simulated and results are visualized in Fig. 10. As expected, spinodal decomposition is shown above 0.23 concentration. Results confirm that CPU, GPU-CUDA, and GPU-Metal based codes exhibit the same morphology.

Conclusions

Using Apple silicon architecture and CUDA architecture, we simulated the spinodal decomposition phenomenon based on a semi-implicit Fourier spectral method that intensively utilizes the Fast Fourier Transform. All set of codes implemented with CUDA and Metal parallelization technique produced the same result in terms of physical outcomes; however the time taken to obtain results varied. In terms of computational efficiency, the M2 Ultra architecture was comparable to the RTX 3090Ti, but the maximum affordable memory of the GPU was approximately four times larger than the RTX 3090Ti. In terms of the manageability of the workstation, the Mac Studio from Apple Inc. has a significant advantage over a typical workstation with the RTX 3090Ti, as it is very small in its form factor and consumes only 1/8 of the power.

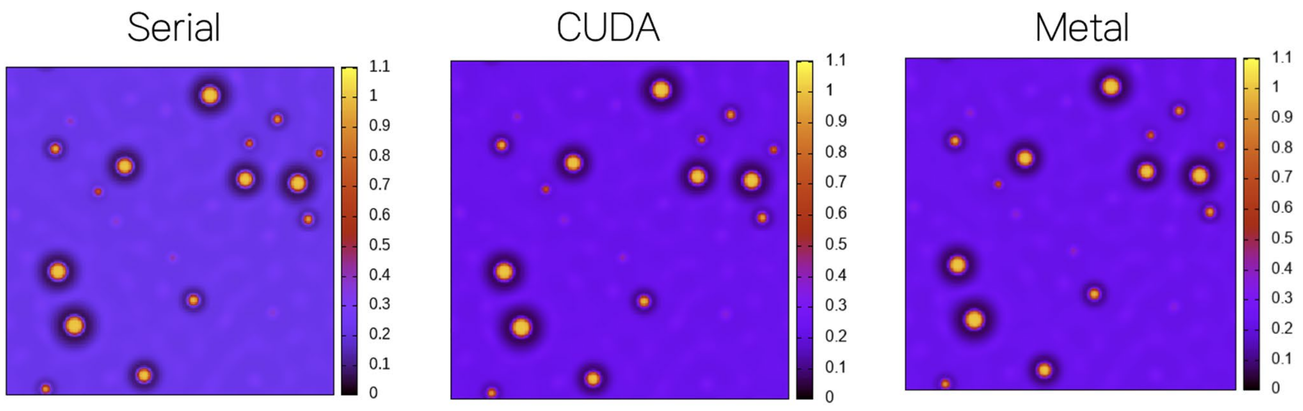


Fig. 8 Result of Cahn–Hilliard equation of 128^2 system dimension at initial concentration of $c_0 = 0.23$, after 10,000 time step

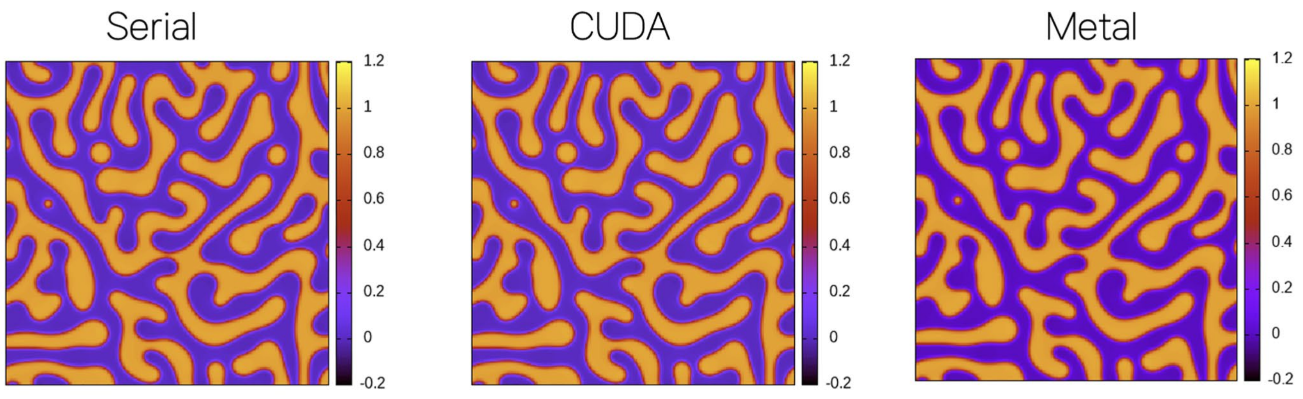
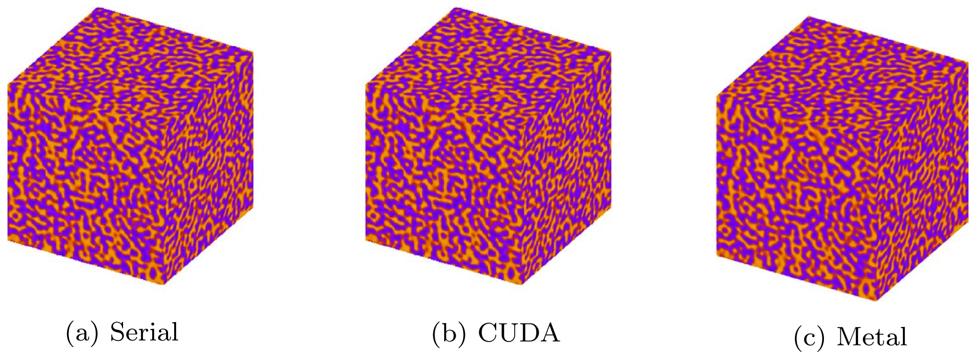


Fig. 9 Morphological result of Cahn–Hilliard equation of 128^2 system dimension at initial concentration of $c_0 = 0.5$, after 10,000 time step

Fig. 10 Result of the Cahn–Hilliard equation solver of 128^3 system dimension at initial concentration of $c_0 = 0.23$, after 10,000 time step from each three-dimensional simulation code: **a** Serial-version code, **b** CUDA-version code and **c** Metal-version code



Acknowledgements This work was supported by Korea Institute of Energy Technology Evaluation and Planning(KETEP) grant funded by the Korea government (MOTIE) (NO. 20224000000550, Training simulation experts for the development of structural materials for molten salt reactors). This work was also supported by the "Human Resources Program in Energy Technology" of the Korea Institute of Energy Technology Evaluation and Planning (KETEP), granted financial resources from the Ministry of Trade, Industry Energy, Republic of Korea (No. 20214000000070).

Funding This article is funded by Korea Institute of Energy Technology Evaluation and Planning (20224000000550, 20214000000070) Kunok Chang.

Data Availability The processed data required to reproduce these findings would be available upon the request.

References

1. C. Luo, Y. Zheng, Y. Xu, H. Ding, C. Zheng, C. Qin, B. Feng, Cyclic co 2 capture characteristics of a pellet derived from sol-gel cao powder with ca 12 al 14 o 33 support. Korean J. Chem. Eng. **32**, 934–938 (2015)
2. Y.T. Lim, O.O. Park, Microstructure and rheological behavior of block copolymer/clay nanocomposites. Korean J. Chem. Eng. **18**, 21–25 (2001)
3. D. Son, W.-G. Lim, J. Lee, A short review of the recent developments in functional separators for lithium-sulfur batteries. Korean J. Chem. Eng. **40**(3), 473–487 (2023)
4. J.W. Cahn, J.E. Hilliard, Free energy of a nonuniform system. i. interfacial free energy. J. Chem. Phys. **28**(2), 258–267 (1958)
5. S.M. Allen, J.W. Cahn, A microscopic theory for antiphase boundary motion and its application to antiphase domain coarsening. Acta Metall. **27**(6), 1085–1095 (1979)
6. S. Hu, L. Chen, A phase-field model for evolving microstructures with strong elastic inhomogeneity. Acta Mater. **49**(11), 1879–1890 (2001)
7. Q. Sherman, P. Voorhees, Phase-field model of oxidation: equilibrium. Phys. Rev. E **95**(3), 032801 (2017)
8. J.W. Cahn, On spinodal decomposition. Acta Metall. **9**(9), 795–801 (1961)
9. L.-Q. Chen, Phase-field models for microstructure evolution. Annu. Rev. Mater. Res. **32**(1), 113–140 (2002)
10. N. Moelans, B. Blanpain, P. Wollants, An introduction to phase-field modeling of microstructure evolution. Calphad **32**(2), 268–294 (2008)
11. I. Steinbach, Phase-field models in materials science. Modell. Simul. Mater. Sci. Eng. **17**(7), 073001 (2009)
12. J. Zhu, L.-Q. Chen, J. Shen, V. Tikare, Coarsening kinetics from a variable-mobility Cahn–Hilliard equation: application of a semi-implicit fourier spectral method. Phys. Rev. E **60**(4), 3564 (1999)
13. D. Li, Z. Qiao, On second order semi-implicit Fourier spectral methods for 2d Cahn–Hilliard equations. J. Sci. Comput. **70**, 301–341 (2017)
14. W. Gropp, E. Lusk, N. Doss, A. Skjellum, A high-performance, portable implementation of the mpi message passing interface standard. Parallel Comput. **22**(6), 789–828 (1996)
15. L. Dagum, R. Menon, Openmp: an industry standard api for shared-memory programming. IEEE Comput. Sci. Eng. **5**(1), 46–55 (1998)
16. J. Sanders, E. Kandrot, CUDA by example: an introduction to general-purpose GPU programming, Addison-Wesley Professional, (2010)
17. Nvidia, CUFFT Library. https://developer.nvidia.com/cufft.
18. D. Tolmachev, Vkfft-a performant, cross-platform and open-source gpu fft library. IEEE Access **11**, 12039–12058 (2023)
19. C. Yang, Q. Xu, B. Liu, Gpu-accelerated three-dimensional phase-field simulation of dendrite growth in a nickel-based superalloy. Comput. Mater. Sci. **136**, 133–143 (2017)
20. A. Zhang, Z. Guo, B. Jiang, S. Xiong, F. Pan, Numerical solution to phase-field model of solidification: a review. Comput. Mater. Sci. **228**, 112366 (2023)
21. J. Glaser, P.S. Schwendeman, J.A. Anderson, S.C. Glotzer, Unified memory in hoomd-blue improves node-level strong scaling. Comput. Mater. Sci. **173**, 109359 (2020)
22. J. Lu, S. Gao, W. Xiong, C. Xu, Optimization of gpu parallel scheme for simulating ultrafast magnetization dynamics model. Comput. Mater. Sci. **184**, 109924 (2020). https://doi.org/10.1016/j.commatsci.2020.109924
23. C. Kenyon, C. Capano, Apple silicon performance in scientific computing, in, IEEE High Performance Extreme Computing Conference (HPEC). IEEE **2022**, 1–10 (2022)
24. L. Gebraad, A. Fichtner, Seamless gpu acceleration for c++-based physics with the metal shading language on apple's m series unified chips. Seismol. Soc. Am. **94**(3), 1670–1675 (2023)
25. L.Q. Chen, J. Shen, Applications of semi-implicit Fourier-spectral method to phase field equations. Comput. Phys. Commun. **108**(2–3), 147–158 (1998)
26. A.M. Jokisaari, P. Voorhees, J.E. Guyer, J. Warren, O. Heinonen, Benchmark problems for numerical implementations of phase field models. Comput. Mater. Sci. **126**, 139–151 (2017)
27. Apple Inc., Apple Metal API version 3.1. https://developer.apple.com/documentation/metal/.
28. Apple Inc., Metal Shading Language Specification Version 3.1. https://developer.apple.com/metal/Metal-Shading-Language-Specification.pdf.
29. L. Chen, O. Villa, S. Krishnamoorthy, G.R. Gao, Dynamic load balancing on single-and multi-gpu systems, in, IEEE International Symposium on Parallel & Distributed Processing (IPDPS). IEEE **2010**, 1–12 (2010)
30. J. Fang, K. Zhou, C. Tan, H. Zhao, Dynamic block size adjustment and workload balancing strategy based on cpu-gpu heterogeneous platform, in, IEEE International Conference on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom). IEEE **2019**, 999–1006 (2019)