Abstract
The multiphase field (MPF) method is recognized as a powerful numerical method to simulate microstructural evolutions, such as solidification, grain growth, recrystallization, and phase transformation, in various materials. However, because we need to solve the time evolution equations for multiple-order parameters derived from the total Gibbs free energy, MPF simulations are very computationally expensive. In this paper, we use a graphics processing unit (GPU) to accelerate the two-dimensional MPF simulation of austenite-to-ferrite transformation in a Fe–C alloy. This is an important phenomenon for predicting the morphology of multiphase microstructures in steel. To perform the MPF simulation on an NVIDIA GPU, the program code is developed in CUDA Fortran. Using this code, the acceleration performance of the GPU implementation is evaluated, and our results demonstrate that the GPU computation can powerfully accelerate the MPF simulation by introducing an active parameter tracking (APT) method, which is used to reduce the computational load and memory consumption. The performance of the GPU computation with APT achieves a speedup factor of 5 compared with the GPU computation without APT and a speedup factor of 15 compared with the basic CPU computation without the APT method.

Key words: Multiphase Field Simulation, Graphics Processing Unit, Numerical Simulation, Iron and Steel, Phase Transformation

1. Introduction
The multiphase field (MPF) method has been actively studied as the most promising numerical method for simulating microstructural evolution in materials\(^{(1)}\)\(^{(3)}\). One of the most attractive advantages of the MPF method is that it enables us to predict not only the variation of volume fraction, but also the morphological change in the simulated microstructure. In fact, the MPF method has been widely used to simulate various microstructural evolutions in solidification\(^{(4)}\)\(^{(5)}\), recrystallization\(^{(6)}\)\(^{(8)}\), and phase transformations\(^{(9)}\)\(^{(11)}\). However, on the basis of sharp interface limit analysis, the simulation result from phase-field modeling depends on the thickness of the diffuse interface\(^{(12)}\). Therefore, in order to obtain quantitative simulation results from phase-field simulations, the interface thickness (which is related to the computational grid spacing and a time increment) should be quite small. These facts imply that a large computational cost is required to perform realistic simulations on the typical scales of microstructure patterns. In particular, because we use multiple-order parameters, the MPF simulation often has a larger computational cost compared with phase-field simulations.
using a single phase-field variable. Thus, it is essential to reduce the computational cost of MPF simulations to allow a quantitative comparison between simulated and experimentally-observed microstructural morphologies.

To overcome this computational cost and enable us to perform quantitative simulations efficiently, several techniques using many CPU cores have been developed. In previous studies, cluster computing using the message passing interface (MPI) and the OpenMP algorithms have often been adopted. George et al.\textsuperscript{(13)} parallelized the phase-field simulation of alloy solidification with the MPI and successfully simulated a full three-dimensional morphology of the dendritic microstructure. Recently, Altenfeld et al.\textsuperscript{(14)} applied the OpenMP parallelization to solve the MPF simulation. In addition, the OpenMP algorithm was implemented into MICRESS\textsuperscript{(15)}, which is software based on the MPF method. This enabled a speedup factor of 5.57 using eight threads on an eight-core system.

In contrast to parallel computing techniques with many CPU cores, high-performance computing with a graphics processing unit (GPU) has also attracted much attention. Although GPUs were developed for rendering computer graphics, since the Compute Unified Device Architecture (CUDA)\textsuperscript{(16)} was released by NVIDIA as a general purpose GPU programming framework in 2006, the programming of GPUs for scientific computing has become possible. Therefore, GPU computing has successfully accelerated scientific simulations in a variety of fields, such as numerical weather prediction\textsuperscript{(17)}, computational fluid dynamics\textsuperscript{(18)}, and molecular dynamic simulations\textsuperscript{(19), (20)}.

For phase-field simulations, the authors have successfully accelerated the formation of dendrite during solidification of a pure material\textsuperscript{(21)}. In a previous study\textsuperscript{(22)}, we presented a phase-field simulation for dendrite formation during solidification in a binary alloy using a single GPU, and achieved a performance of 170 GFlops for a 576\textsuperscript{3} computational mesh. Furthermore, Shimokawabe et al.\textsuperscript{(23)} performed an extreme large-scale three-dimensional phase-field simulation of dendritic solidification in Al–Si alloy using the TSUBAME 2.0 supercomputer at the Tokyo Institute of Technology. Using 4000 GPUs with 16000 CPU cores, the performance reached 2.0 PFlops for a 4096 × 6480 × 13000 computational mesh. According to the above achievements, we have recently implemented GPU computing into the MPF simulation of grain growth in polycrystalline metallic material\textsuperscript{(25)}. This demonstrated that GPU computing is also effective for performing efficient MPF simulations. However, in order to present the effectiveness of GPU computing for the practical MPF simulation of microstructural evolution, it is essential to investigate the performance of MPF simulations for various microstructural evolutions, such as phase transformation.

In this study, we present the GPU acceleration of an MPF simulation for the austenite-to-ferrite (\(\gamma \rightarrow \alpha\)) transformation in a Fe–C alloy. This diffusional solid phase transformation, which occurs in the steel-making process, is important for predicting the complex morphology of multiphase microstructures in steel. This paper is organized as follows. In section 2, the MPF model of the \(\gamma \rightarrow \alpha\) transformation is explained. To simulate the \(\gamma \rightarrow \alpha\) transformation in a Fe–C alloy, we derive two differential equations, which are used to describe the motion of an interface and carbon diffusion during the \(\gamma \rightarrow \alpha\) transformation. In section 3, the discretization method of the differential equations derived from the MPF model is presented. Then, the active parameter tracking (APT) method, which has been used to reduce the computational load and memory consumption\textsuperscript{(24)}, is explained in section 4. In section 5, we present implementation details of the MPF and APT methods for single GPU computing. After explaining the simulation model, we investigate the acceleration performance of our implementations in section 6. Finally, our conclusions are summarized in section 7.

2. Multiphase Field model

In order to simulate the \(\gamma \rightarrow \alpha\) transformation in a Fe–C alloy, we employ the generalized MPF (GMPF) model proposed by Steinbach et al.\textsuperscript{(2)}. Using this MPF model, we can describe multiple-junction points precisely by introducing an additional interface field. Furthermore,
because the $\gamma \rightarrow \alpha$ transformation in steel generally accompanies the diffusion of carbon atoms between the $\alpha$ and $\gamma$ phases, the carbon diffusion equation of the multi-phase system is solved. Although Eiken et al.\cite{26} proposed an MPF model with the diffusion equation derived from total free energy, we employ the diffusion equation with a linearized phase diagram proposed by Tiaden et al.\cite{27} for simplicity. Hereafter, we summarize the main features of the GMPF model used in this study.

By considering a system of $N$ crystal grains, we use $N$ phase-field variables $\phi_i$ ($i=1,2,\ldots,N$). $\phi_i$ describes the fraction of the $i$-th grain and varies smoothly across an interface from $\phi_i=1$ in the $i$-th grain to $\phi_i=0$ in another grain. Thus, all phase-field variables satisfy the constraint $\sum_{i=1}^{N} \phi_i = 1$ at all points. Using the phase-field variables, the total Gibbs free energy of the system $G$ is defined as the Ginzburg–Landau-type Gibbs free energy function, which is given as\cite{28}:

$$G = \int_V \left( \sum_{i=1}^{N} \sum_{j=1}^{N} \left( -\frac{\alpha_i^2}{2} \nabla \phi_i \cdot \nabla \phi_j + W_{ij}(\phi_i) \right) + g \right) dV \quad (1)$$

where the first term of the right-hand side of Eq. (1) describes the gradient energy which is the sum of the interfacial energy. The second term is the potential energy which corresponds to the energy barrier between each grain, $a_{ij}$, $W_{ij}$, and $g$ are the gradient coefficient, the height of the energy barrier, and the bulk free energy density, respectively. These parameters are related to the interfacial energy between grain $i$ and grain $j$, $\sigma_{ij}$, and interfacial thickness $\delta$ as $a_{ij} = 2\sqrt{2\pi\sigma_{ij}}/\pi$ and $W_{ij} = 4\sigma_{ij}/\delta$. By considering $\sigma_{ij} = \sigma_{ji}$ and $\sigma_{ii} = 0$, we find $a_{ij} = a_{ji}$, $W_{ij} = W_{ji}$, $a_{ii} = 0$ and $W_{ii} = 0$.

Assuming that the total free energy decreases monotonically with time, the evolution equation of the phase-field variables can be derived as:

$$\frac{\partial \phi_i}{\partial t} = -\frac{2}{n} \sum_{j=1}^{N} \left[ \frac{\delta G}{\partial \phi_i} - \frac{\delta G}{\partial \phi_j} \right] \quad (2)$$

where $n$ is the number of phase fields at an arbitrary point and is given by $n = \sum_{i=1}^{N} \xi_i$. Here, $\xi_i$ is a step function, which is expressed as $\xi_i = 1$ in the region $0 < \phi_i \leq 1$ and $\xi_i = 0$ elsewhere. Next, we substitute the following functional derivative of $\delta G/\delta \phi_i$ to Eq.(2):

$$\frac{\delta G}{\delta \phi_i} = \sum_{k=1}^{n} \left( W_{ik} \phi_k + \frac{a_{ik}^2}{2} \nabla^2 \phi_k \right) + \frac{\partial g}{\partial \phi_i} \quad (3)$$

Then, we obtain the evolution equation of the phase-field variables as follow:

$$\frac{\partial \phi_i}{\partial t} = -\frac{2}{n} \sum_{j=1}^{N} \left[ \frac{\delta G}{\delta \phi_j} \right] \sum_{k=1}^{n} \left( W_{ik} - W_{jk} \right) \phi_k + \frac{1}{2} \left( a_{ik}^2 - a_{jk}^2 \right) \nabla^2 \phi_k \right) + \frac{8}{\pi} \sqrt{\phi_i \phi_j} \Delta G_{ij} \quad (4)$$

Here, the third term on the right-hand side of Eq. (4) describes the phenomenological thermodynamic driving force of the transformation. The magnitude of the driving force $\Delta G_{ij}$ is given by $\Delta G_{ij} = \Delta S \Delta T$ at the $\alpha/\gamma$ interface, where $\Delta S$ and $\Delta T$ are the entropy difference between the $\alpha$ and $\gamma$ phases and the undercooling, respectively\cite{28}. The mobility of the phase field $M_{ij}^0$ is given by $M_{ij}^0 = n^2 M/8\delta$, which is set to some non-zero value at the $\alpha/\gamma$ interface. $M$ is the mobility of the $\alpha/\gamma$ interface, expressed as $M = M_0 e^{Q_0/RT}$.

To simulate the diffusion of carbon atoms during the $\gamma \rightarrow \alpha$ transformation in a multiphase system, the total carbon concentration $C$ is defined as a linear function of the local carbon concentration $c_i$ weighted by the corresponding phase-field variables $\phi_i$. The local carbon concentration is given by $c_i = k_i C / \sum_{j=1}^{N} k_j \phi_j$. Here, $k_i$ is the partition coefficient of the carbon atoms, defined as the ratio of the equilibrium carbon concentration in the $i$-th grain to that in the $\gamma$ phase\cite{27}. Now, we consider an $\alpha + \gamma$ two-grain system ($N = n = 2$) for simplicity. When $\phi_1$ and $\phi_2$ correspond to the $\alpha$ and $\gamma$ phases, respectively, the total carbon concentration is written as:

$$C = \sum_{i=1}^{n} \phi_i c_i = \phi_1 c_1 + \phi_2 c_2 \quad (5)$$
Using Eq. (5), the diffusion equation for the total carbon concentration can be expressed as the sum of the diffusion fluxes of carbon atoms $J_i$ in individual grains as:

$$
\frac{\partial C}{\partial t} = \nabla \cdot \left( \sum_{i=1}^{n} \phi_i J_i \right)
= \nabla \cdot \left( \sum_{i=1}^{n} \phi_i D_\text{ff}^C \nabla c_i \right)
= \nabla \cdot \left( \phi_1 D_1^C \nabla c_1 + \phi_2 D_2^C \nabla c_2 \right).
$$

(6)

Here, $D_\text{ff}^C$ denotes the diffusion coefficient of carbon atoms in the $i$-th grain. Because $\phi_1$ and $\phi_2$ are the phase field variables for the $\alpha$ and $\gamma$ phases, $D_1^C$ and $D_2^C$ correspond to the diffusion coefficients for carbon atoms in the $\alpha$ and $\gamma$ phases. The carbon diffusion coefficients in the $\gamma$ and $\alpha$ phases are as follows:

$$
D_1^C = D_\text{ff}^C \exp\left(-\frac{Q_\alpha}{RT}\right)
$$

(7)

$$
D_2^C = D_\text{ff}^C \exp\left(-\frac{Q_\alpha}{RT}\right)
$$

(8)

where $D_\text{ff}^C$, $Q_\alpha$, and $Q_\gamma$ are the pre-exponential factor and the activation energy of the diffusion coefficient in the $\alpha$ and $\gamma$ phases. The partition coefficient $k_i$ and the undercooling $\Delta T$ are derived from a linearized phase diagram.(29).

3. Discretization method

In a numerical simulation, the calculation of Eqs. (4) and (6) are carried out by a second-order finite difference scheme for space with a first-order forward Euler-type finite difference method for time on a regular two-dimensional computational grid. In this study, we confirm that no fatal error does not occur during a numerical simulation by using above finite difference schemes.

In order to simply describe the discretization method for the MPF model used in this study, let us consider the $\alpha + \gamma$ two-grain system ($N = n = 2$) again. On this assumption, Eq.(4) for the phase field $\phi_1$ can be written as:

$$
\frac{\partial \phi_1}{\partial t} = -M^\phi_{12} \left\{ (W_{11} - W_{21}) \phi_1 + \frac{1}{2} \left( a_{11}^2 - a_{21}^2 \right) \nabla^2 \phi_1 + (W_{12} - W_{22}) \phi_2 + \frac{1}{2} \left( a_{12}^2 - a_{22}^2 \right) \nabla^2 \phi_2 - \frac{8}{\pi} \sqrt{\phi_1 \phi_2} \Delta G_{12} \right\}
$$

(9)

Here, by considering $\phi_2 = 1 - \phi_1$, $W_{11} = W_{22} = 0$, $W_{12} = W_{21}$, $a_{11} = a_{22} = 0$ and $a_{12} = a_{21}$, Eq.(9) is further reduced to the following equation:

$$
\frac{\partial \phi_1}{\partial t} = M^\phi_{12} \left\{ a_{12}^2 \nabla^2 \phi_1 + W_{12} (2\phi_1 - 1) + \frac{8}{\pi} \sqrt{\phi_1 (1 - \phi_1)} \Delta G_{12} \right\}
$$

(10)

Therefore, the time-derivative term in the left-hand sides of the evolution equations (Eqs.(10) and (6)) are calculated at the time $t$ on a grid point $(l, m)$ by using the following finite difference equations:

$$
\frac{\partial \phi_1}{\partial t} \bigg|_{l(l,m)} = \frac{\phi_1^{l+\Delta t}(l,m) - \phi_1^{l}(l,m)}{\Delta t} + O(\Delta t)
$$

(11)

$$
\frac{\partial C}{\partial t} \bigg|_{l(l,m)} = \frac{C^{l+\Delta t}(l,m) - C^{l}(l,m)}{\Delta t} + O(\Delta t)
$$

(12)

where $\Delta t$ is an increment for each time step. $O$ represents the discretization error.
On the other hand, the second order differential equation with respect to the phase field variable $\phi_1$ shown in the right-hand side of Eq.(10) is calculated on a grid point $(l, m)$ as:

$$\nabla^2 \phi_1 \big|_{(l,m)} = \frac{\phi_1 (l - 1, m) - 2\phi_1 (l, m) + \phi_1 (l + 1, m)}{\Delta x^2} + \frac{\phi_1 (l, m - 1) - 2\phi_1 (l, m) + \phi_1 (l, m + 1)}{\Delta y^2} + O(\Delta x^2)$$  (13)

Furthermore, the first term in the right-hand side of Eq.(6) is calculated as:

$$\nabla \cdot \left( \phi_1 D^c_1 c_1 \nabla c_1 \right) \big|_{(l,m)} = \frac{1}{\Delta x} \left\{ \frac{\phi_1 (l + 1, m) + \phi_1 (l, m) - 2\phi_1 (l, m) + \phi_1 (l - 1, m)}{2} \right\} D^c_1 \frac{c_1 (l + 1, m) - c_1 (l, m)}{\Delta x} + \frac{1}{\Delta y} \left\{ \frac{\phi_1 (l + 1, m) + \phi_1 (l, m) - 2\phi_1 (l, m) + \phi_1 (l - 1, m)}{2} \right\} D^c_1 \frac{c_1 (l + 1, m) - c_1 (l, m)}{\Delta y} + O(\Delta x^2)$$  (14)

where the local carbon concentration of the grain $i$ at the grid point $(l, m)$, $c_i^j (l, m)$, is calculated by using the total carbon concentration $C_i (l, m)$ and the partition coefficient of the carbon atoms $k_i$ as:

$$c_i^j (l, m) = \frac{k_i C_i (l, m)}{\sum_{j=1}^{n} k_i \phi_1^j (l, m)}$$  (15)

Note that the space derivative with respect to $c_2$ in the second term in the right-hand side of Eq.(6) is also calculated by a similar equation as Eq.(14).

4. Active parameter tracking method

Generally, the MPF method\(^{(1), (3)}\) has a heavy computational load, because values of $N$ phase-field variables must be saved in memory and $N$ time evolution equations must be solved at all gridpoints. As mentioned in the previous section, to reduce the high computational cost, we employ the GMPF method. The merit of the GMPF method is that we do not have to solve Eq. (4) on the computational grid with $n = 1$, and it is sufficient to save only $n$ phase-field variables and the total carbon concentration across the computational grid. In addition to adopting the GMPF method, we employ an active parameter tracking (APT) method to reduce the computational load and the memory consumption.

The APT method used in this study\(^{(7)}\) is slightly modified from the original algorithm proposed by Kim et al.\(^{(24), (30)}\). In our APT method, as shown in Fig. 1, only phase-field

---

**Fig. 1** Schematic explanation of the APT method.
variables with non-zero values are saved in memory as an active phase-field variable. In this case, the number of active phase-field variables corresponds to the number of grains \( n \) in a computational grid. On the other hand, the phase-field variables that are below a critical value \( \phi_{cri} \) are taken into account as non-active variables. Non-active phase-field variables do not need to be saved in memory. In this study, we set \( \phi_{cri} = 1.0 \times 10^{-40} \). Using this APT method, it is sufficient to prepare only five or six arrays to record the active phase-field variables on each computational grid. The procedure of the APT method is as follows:

**Step 1:** Solve Eq. (4) for \( n \) phase-field variables, and calculate the increment of the phase-field variable \( \Delta \phi_i \) and the phase-field variable in the next step \( \phi_{new}^i \).

**Step 2:** In Step 1, if the phase-field variable at the previous time step \( \phi_{old}^i \) is zero and the calculated increment \( \Delta \phi_i \) has a negative value, the phase field variable \( \phi_{new}^i \) is removed from the group of active phase-field variables. Then, we decrease \( n \) to \( n - 1 \).

**Step 3:** Steps 1 and 2 are repeated until all phase-field variables satisfying the above condition have been removed.

**Step 4:** The \( n \) active phase-field variables are rearranged in descending size order. The phase-field variables from the largest to the \( n_{phi} \)-th largest are recorded in memory. Here, \( n_{phi} \) is a predefined maximum number of recorded phase-field variables. In this study, \( n_{phi} = 8 \). At this time, the name of the active phase-field variables must also be recorded in memory.

**Step 5:** The phase-field variables are replaced by \( \phi_i^* = \phi_i / \sum_{j=1}^{n_{phi}} \phi_j \) so as to satisfy the constraint \( \sum_{i=1}^{n} \phi_i = 1 \).

**Step 6:** If the value of \( \phi_{new}^i \) at gridpoint \( (l \pm 1, m \pm 1) \) is not zero and \( \phi_{new}^i \) has not been saved at grid point \( (l, m) \), the phase-field variable at \( (l \pm 1, m \pm 1) \) is added to the active phase-field variable group at the gridpoint \( (l, m) \).

In Step 6, the number of active phase-field variables sometimes exceeds \( n_{phi} \). However, because we can expect the number of active phase-field variables to be less than 5 on all computational grids, it is sufficient to record the name and value of \( n_{phi} \) active phase-field variables. In a numerical simulation, Steps 1 to 6 are repeated for each time step.

### 5. GPU implementation

Table 1 shows the main specifications used for the CPU and the GPU computations. We use a single NVIDIA Tesla C2050 GPU with 3.0 GByte GDDR5 SDRAM device memory. In the case of the two-dimensional simulation with a square computational domain, the maximum available number of the finite difference grid is \( 3726 \times 3726 \). The GPU contains 448 CUDA cores, 14 streaming multiprocessors (SMs), each of which consists of 32 CUDA cores as single instruction, multiple data stream units. The on-board device memory, which is also known as the *global memory* in CUDA, is shared by all SMs in a GPU and provides 148 GB/s peak bandwidth. On the GPU, the total capacity of the shared memory and the L1 cache per SM is 64 KB and the amount of the L2 cache per SM is 768 KB. Because these L1 and L2 caches are loaded on the GPU, the shared memory is not utilized as a software-managed cache(22) in this study. For the comparison between the CPU and GPU computations, we perform the CPU computation with an Intel Xeon E3-1270 CPU.

The program code is developed using the CUDA Fortran language released by the Portland Group, Inc(31). This programming language includes a Fortran 2003 compiler and tool.

<table>
<thead>
<tr>
<th>Table 1</th>
<th>Technical specifications.</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>Intel Xeon E3-1270, 4 cores, 3.4 GHz</td>
</tr>
<tr>
<td>Main memory</td>
<td>16 GByte (DDR3)</td>
</tr>
<tr>
<td>GPU</td>
<td>NVIDIA Tesla C2050</td>
</tr>
<tr>
<td>Device memory</td>
<td>3.0 GByte (GDDR5)</td>
</tr>
<tr>
<td>Shared memory + L1 Cache per SM</td>
<td>64 KByte</td>
</tr>
<tr>
<td>L2 Cache per SM</td>
<td>768 KByte</td>
</tr>
<tr>
<td>Operating System</td>
<td>Cent OS 5.6</td>
</tr>
<tr>
<td>Compiler</td>
<td>PGI Accelerated Fortran compiler version 11.10</td>
</tr>
</tbody>
</table>
chain for programming NVIDIA GPUs using Fortran. Similar to the original CUDA C language, CUDA Fortran also enables the distribution of thousands of threads across a large number of CUDA cores. Program code written in CUDA Fortran mainly consists of CPU codes and GPU codes, which are called **host codes** and **device codes**, respectively. Functions that run on a GPU are known as **kernels**. Kernels run in parallel using a large number of threads, which are bundled into a number of **blocks** to compose a **grid**. When a kernel is called by a CPU code, the number of threads per block and number of blocks per grid is specified. In order to perform the phase-field simulation efficiently using a GPU, we should invoke a larger number of threads than there are physical CUDA cores, in order to hide the latency of memory accesses. To fulfill this requirement, we calculate the finite difference grids of a computational domain of size \((n_x, n_y, 1)\) as follows: Our kernel is invoked on a GPU with \((n_x/32, n_y/8, 1)\) blocks, each of which has \((32, 8, 1)\) threads. Therefore, each thread calculates the evolution equations for a finite difference grid. In this study, we use the above number of threads and blocks so that the coalesced global memory access by a half-warp of threads is performed as much as possible.

A flowchart of the program code developed to calculate the MPF simulation with the APT method is shown in Fig. 2. Before the execution of kernels, the initial distribution of the phase-field variables and the carbon concentration at all gridpoints must be stored in both the host and device memories. Therefore, the initial data stored in the host memory is copied to the device memory using **Host-to-Device** communication. After this process, the evolution equations of the phase-field variables, Eq. (4), are solved in the kernel named kernel 1. In kernel 1, steps 1 to 3 of the APT method (explained in the previous section) are performed. At step 2 of the APT method, some phase-field variable are removed from the group of active phase-field variables. If a phase-field variable is removed when the time evolution equation of the same phase-field variable is calculated, data dependency occurs. Thus, in order to avoid

---

**Fig. 2** Flowchart of kernels and communication in the developed program code.
Table 2 Material parameters and physical values.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interfacial energy, $\sigma_{ij}$</td>
<td>$0.5 \text{ [J/m}^2\text{]}$</td>
</tr>
<tr>
<td>Pre-exponential factor of $M, M_0$</td>
<td>$3.5 \times 10^{-7} \text{ [m}^4\text{/J/s]}$</td>
</tr>
<tr>
<td>Activation energy of $M, Q_0$</td>
<td>$-1.4 \times 10^7 \text{ [J/mol]}$</td>
</tr>
<tr>
<td>Difference in entropy, $\Delta S$</td>
<td>$3.46 \times 10^5 \text{ [J/(K} \cdot \text{m}^3\text{)]}$</td>
</tr>
<tr>
<td>Pre-exponential factor of $D_\gamma, D_\gamma^0$</td>
<td>$4.70 \times 10^{-5} \text{ [m}^2\text{/s]}$</td>
</tr>
<tr>
<td>Pre-exponential factor of $D_\alpha, D_\alpha^0$</td>
<td>$1.24 \times 10^{-5} \text{ [m}^2\text{/s]}$</td>
</tr>
<tr>
<td>Activation energy of $D_\gamma, Q_\gamma$</td>
<td>$1.55 \times 10^5 \text{ [J/mol]}$</td>
</tr>
<tr>
<td>Activation energy of $D_\alpha, Q_\alpha$</td>
<td>$9.95 \times 10^4 \text{ [J/mol]}$</td>
</tr>
</tbody>
</table>

Fig. 3 Initial distribution of $\gamma$ grains.

such a data dependency, we prepare two arrays for the phase-field variable, which we name P1 and P2. The P1 array is used for solving Eq. (4) and P2 is used for the calculation in step 2 of the APT method. After kernel 1, the phase-field variables stored in P1 are overwritten with the data in P2 using Device-to-Device communication. Using the updated active phase-field variables, the carbon diffusion equation, Eq. (6), is solved by calling the second kernel (kernel 2). Following the calculation of the phase-field variables and the carbon concentration, steps 4 and 5 of the APT method are conducted in a third kernel (kernel 3). Furthermore, step 6 of the APT method is performed in a fourth kernel (kernel 4). In order to update the calculated phase-field variables and carbon concentration, these variables are overwritten with the data on the device memory using Device-to-Device communication. When we read out the calculated data to a file as necessary, the data on the device memory is copied to the host memory by Device-to-Host communication. The time integration of the time evolution equations is performed until a predefined time step by repeating the above procedure. It should be noted that synchronization is used to ensure the accomplishment of the computation and the communication of all phase-field and concentration variables.

6. Results and discussion

6.1. Simulation model for performance evaluation

To investigate the acceleration performance of the GPU computation, we perform a two-dimensional MPF simulation of the isothermal $\gamma \rightarrow \alpha$ transformation in a polycrystalline Fe–C alloy on a single GPU. The temperature is kept constant at $T = 1000 \text{ K}$. The size of the computational domain is $DX \times DY = 128 \times 128 \ \mu\text{m}^2$, which is discretized into a $256 \times 256$ finite difference grid. The size of the grid spacing is $\Delta x = \Delta y = 0.5 \ \mu\text{m}$. The increment for each time step is $\Delta t = 5.0 \times 10^{-6} \text{ s}$.

Figure 3 shows the initial distributions of the parent $\gamma$ phase. In Fig. 3, black solid lines represent grain boundaries, which are defined by the profile of $\sum_{i=1}^{n} \phi_i = 0.6$. The initial configuration of the $\gamma$ grains is produced by an ordinary MPF simulation of grain growth. Because of the periodic boundary condition, 10 $\gamma$ grains are included in the computational
domain. The number of α nuclei is varied from 10 to 40 in order to study its effect on the acceleration performance. We assume that all α nuclei are nucleated at the first time step. The initial radius of the α nuclei is set to $r = 7\Delta x = 3.5 \mu m$. The initial carbon concentration is set to 0.15 wt.% in the γ phase and equilibrium in the α phase. The material and computational parameters used in this study are listed in Table 2.

It should be noted that, in this study, all calculations are performed in single precision, similar to our previous phase-field simulations using a GPU. Because the value ranges of the phase-field variables and the carbon concentration are small and their time evolutions are integrated explicitly, we have confirmed that the single precision results are sufficiently accurate in comparison with those of double precision. To evaluate the acceleration performance, we measure the elapsed time of each kernel using the CUDA visual profiler(32) for the GPU computation and the PGI profiler(33) for the CPU computation.

In the next section, we compare the performance of the GPU computation with that of the CPU computation. In this study, we investigate the performance of the GPU and CPU computations both with and without the APT method. Further, we compare the CPU computation with the APT method and an auto-parallelization (-Mconcur) option which is an function of the PGI compiler.(34) Using the auto-parallelization option, the compiler automatically parallelizes the calculation loop which does not contain any cross-iteration data dependencies and generates an parallel program code for multi-core CPU.(35) In the CPU computation with the auto-parallelization technique performed in this study, the MPF simulation is performed using four threads on an four-core CPU. For these different implementations, the performance of the MPF simulation with a large number of time steps and crystal grains is investigated. On the basis of the obtained results, we discuss the acceleration of the MPF simulation with GPU computations.

6.2. Performance analysis of CPU and GPU implementations

Figure 4 shows the total elapsed time for 10000 time steps with 10 α nuclei for five different implementations. Comparing the three different CPU implementations, the elapsed time can be reduced by over 50% just by introducing the APT method. The auto-parallelization technique further reduces the elapsed time by almost 50% of that without the parallelization technique. The performance of the GPU computation without the APT method shows a slight improvement over the CPU computation with the APT method. However, the CPU computation with the APT method and the parallelization technique is faster than the GPU
computation without the APT method. We can see that the GPU computation with the APT method enables us to powerfully accelerate the MPF simulation, resulting in the fastest computation. The GPU computation with the APT method exhibits an acceleration factor of 15.3 compared with the CPU computation without the APT method.

To further discuss the performance of each implementation, a breakdown of the computational times and communication times of the different implementations is shown in Fig. 5. In this figure, the execution time of the kernel, Device-to-Host, Host-to-Device, and Device-to-Device communication, as well as other CPU functions, is compared for each implementation. In all cases, the longest amount of time is spent solving the time evolution equations for the phase-field variables, which corresponds to kernel 1. Therefore, we find that the speedup factor shown in Fig. 4 is determined by the elapsed time of kernel 1. By comparing the result of the CPU computation without the APT method to that of the GPU computation, we find that the elapsed time for kernel 1 can be reduced by almost half by simply using the GPU computation. When we introduce the APT method to the GPU computation, although the execution time of kernels for the APT method is needed, we can further reduce the calculation time for kernel 1. This is because the global memory access in kernel 1 is greatly decreased by employing the APT method. These results demonstrate that the acceleration of the kernel 1 computation is the essential factor in performing efficient MPF simulations on the GPU.

6.3. Performance of the MPF simulation including a large number of crystal grains

In order to obtain statistically meaningful results to evaluate the morphological character of the simulated microstructure, we need to perform the MPF simulation using a large number of crystal grains in a large computational region. In this section, we investigate the effect of the number of crystal grains, which corresponds to the number of phase-field variables, on the computational performance.
Fig. 6 Elapsed time of the MPF simulation as a function of the number of crystal grains for different implementations.

Fig. 7  Elapsed time of the MPF simulation as a function of the time step for different implementations.
Figure 6 shows the elapsed time of the MPF simulation as a function of the number of crystal grains for different implementations. Here, the number of calculation steps is fixed at 10000. For the CPU and GPU computations without the APT method, because \( n \) time evolution equations for the phase-field variables must be solved, the elapsed time increases in proportion to the number of crystal grains. Comparing the GPU computation without the APT method with that of the CPU computation, it is found that a speedup factor of between 3 and 4 can be obtained. Therefore, when we do not employ the APT method, the MPF simulation of the transformation behavior in a thousand crystal grains still requires significant computational time, even on the GPU.

On the other hand, when the APT method is used, we find that the rate of increase with respect to the grain number is very low. If we apply the APT method to the CPU computation, the elapsed time decreases by almost half. Moreover, the parallelization technique enables us to further reduce the elapsed time. Interestingly, our results indicate that the GPU computation with the APT method can reduce the computational time to less than 25% of the GPU computation without the APT method. By focusing on the case of 40 crystal grains, it is demonstrated that the performance acceleration of the GPU computation with the APT method is from 7 to 15 times. This speedup factor is due to the acceleration of the kernel 1 calculations, as mentioned in the previous section. These results clearly indicate that the GPU computation with the APT method is the best way to perform efficient MPF simulations with a large number of crystal grains.

6.4. Performance of the MPF simulation with a large number of time steps

The realistic heat treatment process of steel is often conducted on a hundred-second time scale. Therefore, in order to compare the MPF simulation result with experimental results, we have to perform the MPF simulation with millions of time steps. For this reason, it is important to accelerate the MPF simulation with a large number of time steps using a GPU. In this section, we study the effect of the number of time steps on performance.

Figure 7 shows the elapsed time of the MPF simulation as a function of the number of time steps for different implementations. In these simulations, the number of \( \alpha \) grains is fixed at 40. The simulated evolution of the carbon concentration during the \( \gamma \rightarrow \alpha \) transformation is
shown in Fig. 8. In Fig. 9, the calculated distributions of the carbon concentration along line A-A' shown in Fig. 8 are compared with respect to the CPU and the GPU computations with the APT method. The error in the carbon concentration for the CPU and GPU computations is also imposed in Fig. 9. Here, the error is evaluated as $\left| C_{\text{gpu}} - C_{\text{cpu}} \right| / C_{\text{cpu}}$, where $C_{\text{cpu}}$ and $C_{\text{gpu}}$ represent the carbon concentration calculated by the CPU and the GPU, respectively. Although small computational errors are observed, we can confirm that the distribution of the carbon concentration calculated by the GPU is very close agreement with that by the CPU.

According to Fig. 8, the $\alpha$ grains grow as the time step increases. Because the $\gamma \rightarrow \alpha$ transformation accompanies the carbon diffusion from the $\gamma$ phase to the $\alpha$ phase, the enrichment of the carbon concentration in the $\gamma$ phase occurs, as shown in Fig. 9. Furthermore, since the calculation time for kernel 1 increases with the number of time steps, the elapsed time increases with respect to the number of time steps for both the CPU and GPU implementations, as shown in Fig. 7. It is worth mentioning that the rate of increase in the time for the GPU computation with the APT method is somewhat less than that for other implementations. With $1.0 \times 10^7$ time steps, the GPU computation with the APT method achieves a speedup factor of approximately 4 with respect to the GPU computation without the APT method and the CPU computation with the APT method and the parallelization technique. From this result, it can be expected that even if we carry out the MPF simulation with a large number of time steps, the GPU computation with the APT method will enable us to perform very efficient MPF simulations.

7. Conclusion and future issues

This paper presented the GPU acceleration of MPF simulations, which are a useful and powerful numerical method of simulating microstructural evolution in materials. In this study, we developed a program code using the CUDA Fortran language to perform efficient MPF simulations with the APT method on a NVIDIA TESLA C2050 GPU. Using the developed code, a two-dimensional MPF simulation of the isothermal $\gamma \rightarrow \alpha$ transformation in a Fe–C alloy was performed. To evaluate the acceleration performance of our GPU implementation, we studied the performance of the MPF simulation on both the GPU and an Intel Xeon CPU. The results demonstrated that the GPU computation with the APT method enables us to significantly accelerate the MPF simulation by using just a single GPU. For the GPU computation with the APT method, the acceleration performance achieved a 15-times speedup compared to the CPU computation and a 5-times speedup compared to the GPU computation without the
APT method. The key point of this acceleration is to introduce the APT method into the GPU computation and reduce the computational load, memory consumption, and global memory access. Therefore, we have demonstrated that, as the global memory on a GPU is limited, the APT method is an essential algorithm for performing efficient, large-scale MPF simulations on the GPU. The performance evaluation demonstrated that the GPU computation with the APT method was the most effective way of performing realistic MPF simulations with a large number of crystal grains.

In the future, the GPU implementation presented in this paper will be compared to a multi-CPU computation with more than four CPU cores using the OpenMP and MPI algorithms. Furthermore, the tuning of the program code that enables the effective utilization of the shared memory and the L1/L2 caches on the GPU will be performed to further accelerate the GPU computation. From the aspect of engineering applications, the acceleration performance of three-dimensional MPF simulations by our GPU implementation will be investigated. However, the amount of the global memory on the GPU used in this study is not sufficient to perform realistic simulations on the typical scales of microstructure patterns in three-dimensions. To obtain such realistic simulation results that can be directly compared to experimentally observed microstructures, multi-GPU computations with a GPU-rich cluster, such as the TSUBAME 2.0 supercomputer at the Tokyo Institute of Technology, will be quite useful.

Acknowledgements

This research was supported in part by an ISIJ Research Promotion Grant from the Iron and Steel Institute of Japan and by a Grant-in-Aid for Scientific Research for Young Scientists (B) from the Japan Society of the Promotion of Science (Project No. 23760088).

References


