Performance Optimizations for Parallel Modeling of Solidification with Dynamic Intensity of Computation

Kamil Halbiniak[1(✉)], Lukasz Szustak[1], Adam Kulawik[1], and Pawel Gepner[2]

[1] Czestochowa University of Technology, Dabrowskiego 69, 42-201 Czestochowa, Poland
{khalbiniak,lszustak,adam.kulawik}@icis.pcz.pl
[2] Intel Corporation, Santa Clara, USA
pawel.gepner@intel.com

Abstract. In our previous works, a parallel application dedicated to the numerical modeling of alloy solidification was developed and tested using various programming environments on hybrid shared-memory platforms with multicore CPUs and manycore Intel Xeon Phi accelerators. While this solution allows obtaining a reasonable good performance in the case of the static intensity of computations, the performance results achieved for the dynamic intensity of computations indicates pretty large room for further optimizations.

In this work, we focus on improving the overall performance of the application with the dynamic computational intensity. For this aim, we propose to modify the application code significantly using the *loop fusion* technique. The proposed method permits us to execute all kernels in a single nested loop, as well as reduce the number of conditional operators performed within a single time step. As a result, the proposed optimizations allows increasing the application performance for all tested configurations of computing resources. The highest performance gain is achieved for a single Intel Xeon SP CPU, where the new code yields the speedup of up to 1.78 times against the original version.

The developed method is vital for further optimizations of the application performance. It allows introducing an algorithm for the dynamic workload prediction and load balancing in successive time steps of simulation. In this work, we propose the workload prediction algorithm with 1D computational map.

Keywords: Numerical modeling of solidification · Phase-field method · Parallel programming · OpenMP · Workload prediction · Load balancing · Intel Xeon Phi · Intel Xeon Scalable processors

1 Introduction

The phase-field method is a powerful tool for solving interfacial problems in materials science [9]. It has been not only used for solidification dynamics [7,8],

R. Wyrzykowski et al. (Eds.): PPAM 2019, LNCS 12043, pp. 370–381, 2020.
https://doi.org/10.1007/978-3-030-43229-4_32

but has been also applied to other phenomena such as viscous fingering [3], fracture dynamics [5], and vesicle dynamics [9]. In our previous works [4,10,12], a parallel application dedicated to the numerical modeling of alloy solidification, which is based on the phase-field method, was developed and tested using various programming environments on hybrid shared-memory platforms with multicore CPUs and manycore Intel Xeon Phi accelerators. In these works, two different versions of application were considered: with the static and dynamic intensity of computations. The results of experiments shown that in the second case the performance and scalability drop significantly. In particular, for the static computational intensity, the usage of 4 KNL processors permits us to accelerate the application about 3.1 times against the configuration with a single KNL processor. At the same time, the performance results achieved in the case of the dynamic intensity of computations indicated pretty large room for further optimizations, since for example the utilization of 4 KNL devices gives only the speedup of 1.89 times.

In this work, we focus on improving the overall performance of the solidification application with the dynamic computational intensity. We propose an optimization method which is based on the *loop fusion* technique and assumes a significant modification of the application source code. This method permits us to compress the main application workload into a single nested loop, as well as reduce the number of conditional operators which have to be performed within a single time step. At the same time, the developed method is vital for further optimizations of the application performance. It allows introducing an algorithm for the dynamic workload prediction and load balancing in successive time steps of simulation. In this work, we propose the workload prediction algorithm with 1D computational map.

This paper is organized as follows. Section 2 outlines the numerical model, as well as presents the basic version of the solidification application with the dynamic intensity of computations. Section 3 outlines the approach for performance optimization of the application. The next section presents performance evaluation of the proposed method on platforms equipped with Intel Xeon Scalable CPUs and Intel KNL accelerators. Section 5 outlines the algorithm for the workload prediction and load balancing, which is based on 1D computational map. Section 6 concludes the paper.

2 Parallelization of Numerical Modeling of Solidification with Dynamic Intensity of Computation

2.1 Overview of Numerical Model

In the modeling problem studied in this paper, a binary alloy of Ni-Cu is considered as a system of the ideal metal mixture in liquid and solid phases. The numerical model refers to the dendritic solidification process [1,13] in the isothermal conditions with constant diffusivity coefficients for both phases. It allows us to use the field-phase method defined by Warren and Boettinger [14]. In the

model, the growth of microstructure during the solidification is determined by solving a system of two PDEs. The first equation defines the phase content ϕ:

$$\frac{1}{M_\phi}\frac{\partial \phi}{\partial t} = \varepsilon^2 \left[\nabla \cdot \left(\eta^2 \nabla \phi\right) + \eta\eta' \left(sin(2\theta)\left(\frac{\partial^2 \phi}{\partial y^2} - \frac{\partial^2 \phi}{\partial x^2}\right) + 2cos(2\theta)\frac{\partial^2 \phi}{\partial x \partial y}\right)\right]$$
$$-\frac{1}{2}\left(\eta'^2 + \eta\eta''\right)\left(-cos\left(2\theta\right)\left(\frac{\partial^2 \phi}{\partial y^2} - \frac{\partial^2 \phi}{\partial x^2}\right) + 2sin\left(2\theta\right)\frac{\partial^2 \phi}{\partial x \partial y} - \frac{\partial^2 \phi}{\partial x^2} - \frac{\partial^2 \phi}{\partial y^2}\right) \quad (1)$$
$$-cH_B - \left(1 - c\right)H_A - cor\ ,$$

where: M_ϕ is defined as the solid/liquid interface mobility, ε is a parameter related to the interface width, η is the anisotropy factor, H_A and H_B denotes the free energy of both components, *cor* is the stochastic factor which models thermodynamic fluctuations near the dendrite tip. The coefficient θ is calculated as follows:

$$\theta = \frac{\partial \phi}{\partial y} \Big/ \frac{\partial \phi}{\partial x}\,. \quad (2)$$

The second equation defines the concentration c of the alloy dopant, which is one of the components of the alloy:

$$\frac{\partial c}{\partial t} = \nabla \cdot D_c \left[\nabla c + \frac{V_m}{R} c\left(1 - c\right)\left(H_B\left(\phi, T\right) - H_A\left(\phi, T\right)\right)\nabla \phi\right], \quad (3)$$

where: D_c is the diffusion coefficient, V_m is the specific volume, R is the gas constant.

In this model, the generalized finite difference method [2,6] is used to obtain the values of partial derivatives in Eqs. (1) and (2). In order to parallelize computations with a desired accuracy, the explicit scheme is applied with a small time step $\Delta t = 1e-7s$.

The resulting computations [12] belong to the group of forward-in-time, iterative algorithms since all the calculations performed in the current time step k depend on results determined in the previous step $k-1$. The application code consists of two main blocks of computations, which are responsible for determining either the phase content ϕ or the dopant concentration c. In the model, the values of ϕ and c are determined for nodes distributed across a considered domain (Fig. 1). For this aim, the values of derivatives in all nodes have to be calculated at every time step.

In our previous work [12], two different cases were introduced – with the static and dynamic intensity of computations. In the first case, the workload of computing resources is constant during the application execution, since a constant number of equations is solved. This assumption corresponds to modeling problems in which the variability of solidification phenomena in the whole domain has to be considered. In the second case, the model is able to solve differential equations only in part of nodes, which is changing during the simulation following the growth of microstructure. The use of a suitable selection criterion allows reducing significantly the amount of computations. The consequence is a significant workload imbalance, since the selection criterion is calculated after the static partitioning of the grid nodes across computing resources.

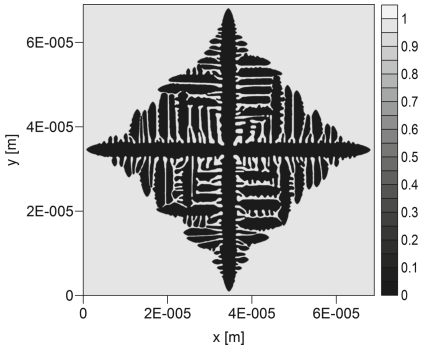

Fig. 1. Phase content for the simulated time $t_S = 2.75 \times 10^{-3} s$

2.2 Basic Version of Solidification Application with the Dynamic Intensity of Computations

Figure 2 illustrates the computational core of the solidification application with the dynamic intensity of computations, for a single time step. All the computations in the application are organized as five loops that iterate over all nodes of the grid. Two of them, with kernels K_1 and K_3, execute calculations in the boundary nodes, while the other two loops, with kernels K_2 and K_4, perform computations for the internal part of the grid. The last loop completes the execution of a single time step. The selection of the boundary and internal nodes of the grid is implemented using four conditional statements, which have to be executed within a single time step.

All kernels of the application are organized as two nested loops, where the outer and inner loops iterate over the grid nodes and neighbors of each node, respectively. The inner loop corresponds to stencil computations used for determination of partial derivatives. Figures 3 and 4 depict the code snippets corresponding to kernels K_1 and K_2 that are responsible for determining the dopant concentration c for the boundary and internal nodes of the grid, respectively. The structure of kernels K_3 and K_4 that calculate the phase content ϕ is similar. For a given node i, the indices `node_e[offset+j]` of its neighbours are kept in the configuration file describing the whole grid. In consequence, the patterns of all 22 stencils are determined at runtime. The structure of kernels allows their parallelization using **omp parallel for** directives of OpenMP for the outer loops.

The application studied in this paper uses the SOA (structure of arrays) layout of memory, where computations are performed using one-dimensional arrays. For instance, `node_conc0[i]` contains value of the dopant concentration for the *i-th* node, while `node_Fi0[i]` corresponds to value of the phase content for this node. The transformation to the SoA organization of memory from the original AoS (array of structures) layout, which was used in the original code, are in line with the first step of the methodology proposed in work [4].

```
#pragma omp parallel for
for(int i=0; i<gridSize; ++i) {
   if(boundary[i]) {
      Kernel K1
   }
}
#pragma omp parallel for
for(int i=0; i<gridSize; ++i) {
   if(!boundary[i]) {
      Selection criterion
      Kernel K2
   }
}
#pragma omp parallel for
for(int i=0; i<gridSize; ++i) {
   if(boundary[i]) {
      Kernel K3
   }
}
#pragma omp parallel for
for(int i=0; i<gridSize; ++i) {
   if(!boundary[i]) {
      Selection criterion
      Kernel K4
   }
}
#pragma omp parallel for
for(int i=0; i<gridSize; ++i) {
    Kernel K5
}
```

Fig. 2. General scheme of basic version of solidification application with the dynamic intensity of computations

The criterion for selecting grid nodes plays a vital role for the overall performance of the application. In the basic version of the solidification application with the dynamic computational intensity, the selection criterion (see Fig. 2) is checked during execution of kernels K_2 and K_4. The execution of this criterion involves two additional conditional operators. As a result, six conditional statements have to be executed within a single time step. Moreover, the execution of the selection criterion leads to the analysis of practically all nodes of the grid (excluding boundary ones), not just nodes within the area of grain growth.

```
#pragma omp parallel for
for(int i=0; i<grid_size; i++) {
   const int offset = i*max_neighbors;
   double d0(0.0), d2(0.0);
   double z[max_neighbors];
   /.../
   for(int j=0; j<neighbors_count[i]; ++j) {
      // Stencil computations used to determine partial derivatives
      z[j] = (node_g2[offset+0]*node_cosAlf[i]+
             node_g2[offset+2]*node_cosBet[i])*node_hx[offset+j]+
             (node_g2[offset+1]*node_cosAlf[i]+
             node_g2[offset+3]*node_cosBet[i])*node_hy[offset+j]
      const int idx = node_e[offset+j];
      d2 += node_conc0[idx]*z[j];
      /.../
   }
   // Computations performed within nodes
}
```

Fig. 3. Kernel K_1

```
#pragma omp parallel for
for(int i=0; i<grid_size; i++) {
   const int offset = i*max_neighbors;
   const int gOffset = i*25;
   double d1xCj(0.0), d1xDcj(0.0), d1xFj(0.0);
   double zx[max_neighbors];
   /.../
   for(int j=0; j<neighbors_count[i]; ++j) {
      // Stencil computations
      // 3 of all 15 stencils used in kernel K2
      zx[j] = 1/pow(node_h[offset+j],2*m)*
              (node_g[gOffset+0]*node_hx[offset+j]+
              node_g[gOffset+1]*node_hy[offset+j]+
              0.5*node_g[gOffset+2]*node_hx[offset+j]*node_hx[offset+j]+
              0.5*node_g[gOffset+3]*node_hy[offset+j]*node_hy[offset+j]+
              node_g[gOffset+4]*node_hx[offset+j]*node_hy[offset+j]);
      const int idx = node_e[offset+j];
      d1xCj += zx[j]*node_conc0[idx];
      d1xDcj += zx[j]*node_Dc[idx];
      d1xFj += zx[j]*node_Fi0[idx];
      /.../
   }
   // Computations performed within nodes
}
```

Fig. 4. Kernel K_2

```
#pragma omp parallel for
for(int i=0; i<gridSize; ++i) {
    Selection criterion
    if(boundary[i]) {
        Kernel K1
        Kernel K3
    }
    else {
        Kernel K2
        Kernel K4
    }
    Kernel K5
}
```

Fig. 5. General scheme of modified version of solidification application with the dynamic intensity of computations

3 Performance Optimization of the Application Using *Loop Fusion*

In this section, we present a method for optimizing the performance of solidification application with the dynamic computational intensity. This method assumes a significant modification of the application source code using the *loop fusing* technique. This optimization technique assumes merging selected loops, in order to reduce the loop overheads, as well as increase the instruction parallelism, improve the data locality, and even reduce data transfers [11].

Figure 5 presents the general scheme of executing a single time step of the application after applying the *loop fusion* technique. In contrast to the basic version of code (Fig. 2), all workloads of the modified version are executed in a single nested loop. Such a solution allows us to reduce the number of conditional statements used to selecting the boundary and internal nodes of the grid, as well as to decrease the number of conditional statements required for checking the selection criterion. In practice, the execution of a single time step requires now to perform only two conditional statements, instead of six ones in the basic version.

Implementing the *loop fusion* requires also a suitable modification of the selection criteria. The resulting criterion is obtained by merging the selection criteria used for the kernels K_2 and K_4 in the basic code. Moreover, due to removing the loop completing each time step, some additional calculations have to be performed in the new selection criterion.

In the modified scheme of the application execution (Fig. 5), the selection criterion is still calculated for all nodes of the grid. However, this scheme allows us to introduce further optimization of the application performance by providing a method for the efficient workload prediction and load balancing across resources of a computing platform (see Sect. 5).

Table 1. Specification of tested platforms

	Intel Xeon Platinum 8180 (SKL)	Intel Xeon Phi 7250F (KNL)
Number of devices	2	1
Number of cores per device	28	68
Number of threads per device	56	272
Base frequency [GHz] (AVX frequency)	2.5 (1.7)	1.4 (1.2)
SIMD width [bits]	512	512
AVX peak for DP [GFlop/s]	3046.4	2611.2
Size of last level cache [MB]	38.5	34
Memory size	512 GB DDR4	16 GB MCDRAM 96GB DDR4
Memory bandwidth [GB/s]	119,2	MCDRAM: 400+ DDR4: 115.2

4 Experimental Results

In this section, we present performance results obtained for the approach proposed in the previous section, assuming the double precision floating point format. The experiments are performed for two platforms (Table 1):

1. SMP server equipped with two Intel Xeon Platinum 8180 CPUs (first generation of Intel Xeon Scalable Processor architecture);
2. single Intel Xeon Phi 7250 F processor (KNL architecture).

The KNL accelerator is utilized in the *quadrant* clustering mode with the MCDRAM memory configured in the *flat* mode. All the tests are compiled using the Intel `icpc` compiler (version 19.0.1) with `-O3` and `-xMIC-AVX512` flags for Intel KNL, and `-xCore-AVX512 -qopt-zmm-usage=high` flags for Intel Xeon Platinum CPUs. In order to ensure the reliability of performance results, the measurements of the execution time are repeated $r = 10$ times, and the median value is used finally.

Table 2 presents the total execution times and speedups achieved for the basic and optimized versions of the solidification application with the dynamic computational intensity. The tests are executed for 110 000 time steps, and two grid sizes: 2000×2000 and 3000×3000, using three configurations of computing resources:

1. single Intel Xeon Platinum 8180 CPU;
2. two Intel Xeon Platinum 8180 CPUs;
3. single Intel KNL processor.

The analysis of Table 2 permits us to conclude that the proposed method allows increasing significantly the performance of computations for all configurations of computing resources. for the grid of size 2000×2000, the highest performance gain is achieved for configuration with a single Intel Xeon Platinum CPU, when the new code yields the speedup of about 1.78 times against the basic

Table 2. The execution times and speedups achieved for the basic and optimized versions of the solidification application with the dynamic intensity of computational

Size of grid	Computing resources	Execution time [s]		Speedup
		Basic T_B	Optimized T_{Op}	$S = T_B/T_{Op}$
2000×2000	$1 \times$ SKL	1785	1001	1.78
	$2 \times$ SKL	1456	837	1.74
	$1 \times$ KNL	1661	1078	1.54
3000×3000	$1 \times$ SKL	4061	2359	1.72
	$2 \times$ SKL	3266	2005	1.63
	$1 \times$ KNL	3869	2540	1.52

version. At the same time, the lowest speedup equal to 1.54 times is obtained on Intel KNL processor. For configuration with two Intel Xeon CPUs, the developed implementation allows accelerating the simulation by 1.74 times. For the second grid, the performance gains achieved by the proposed optimization method are similar.

5 Toward Dynamic Workload Prediction and Load Balancing: 1D Map Approach

Although the proposed optimization method (see Sect. 3) allows increasing the performance of computations, it does not ensure an efficient workload distribution across computing resources. Thus, the next step of performance optimization of the solidification application with the dynamic computational intensity will focus on resolving this issue. To achieve this goal, we propose an algorithm for the dynamic workload prediction.

This algorithm is responsible for predicting the computational workload in successive time steps of the simulation. In practice, it permits us to adjust the computational domain to the domain of simulation. The computational domain refers to the grid area wherein the primary computations are performed and the selection criterion is checked. The prediction of the workload for the next time step $k + 1$ is based on results of computations performed in the current step k. In practice, if values of variables in a grid node are computed in a given time step, this node and its neighbours are taken into consideration when predicting the computational domain for the next time step.

The workload prediction algorithm proposed in this paper is illustrated in Fig. 6. It is based on representing the grid nodes using an 1D array. The computational domain predicted for the next time step is defined by two coordinates referring respectively to the beginning (`minNode`) and end (`maxNode`) of the area wherein the computations are performed and the selection criterion is checked. For the first time step ($k = 1$), the area of checking the selection criterion includes

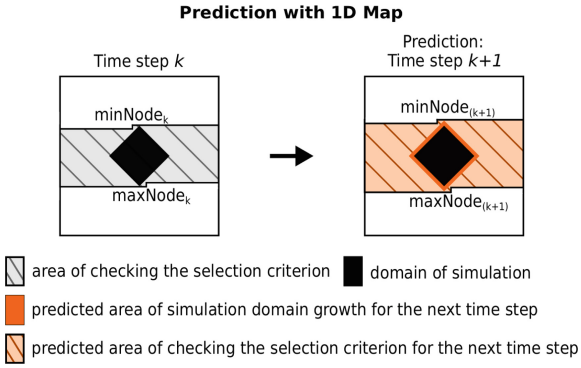

Fig. 6. Predicting domain of computation in successive time steps with 1D map

the whole grid. In this case, `minNode = 0` and `maxNode = grid_size`. Then, starting from the second time step ($k = 2$), the selection criterion is checked only within the area adjusted to the domain of simulation.

Predicting the computational domain plays a significant role in ensuring the efficient load balancing across computing resources (cores). In the basic version of the application, the selection criterion is checked for all nodes of the gird. It

```
int minTemp = minNode;
int maxTemp = maxNode;
#pragma omp parallel for \
   reduction(min:minTemp) reduction(max:maxTemp)
for(int i=minNode; i<maxNode; ++i) {
   Selection criterion
   minTemp = min(i, minTemp);
   maxTemp = max(i, maxTemp);
   if(boundary[i]) {
      Kernel K1
      Kernel K3
   }                                  Primary
   else {                          computations
      Kernel K2
      Kernel K4
   }
      Kernel K5
}
Workload prediction: determining new values of
minNode and maxNode using minTemp and maxTemp
```

Fig. 7. General scheme of a single time step in solidification application with workload prediction using 1D map

leads to undesirable situations when only a part of cores perform primary computations, while the rest of cores are responsible only for checking the selection criterion. The usage of the algorithm for workload prediction permits resolving this problem. As a result, the selection criterion is not checked for the entire grid, but only for the predicted area (Fig. 7). This ensures a more efficient workload distribution across cores, since all cores will perform primary computations within the domain of simulation in successive time steps.

6 Conclusions and Future Works

The main challenge of this work is the performance optimization of the solidification application with the dynamic computational intensity. For this aim, we propose to modify the application code significantly using the *loop fusion* technique. The proposed approach permits us to execute all kernels of the application in a single nested loop, as well as reduce the number of conditional operators that have to be performed within a single time step.

The achieved performance results show that the proposed optimization method allows increasing the application performance for all tested configurations of computing resources. The highest performance gain is achieved for a single Intel Xeon CPU, where the new code yields the speedups of about 1.78 and 1.72 times against the basic version, respectively for grids of size 2000×2000 and 3000×3000. At the same time, the usage of the proposed method for a single Intel KNL accelerator permits to reduce the execution time of about 1.54 and 1.52 times, respectively.

The aim of our future work is to investigate the possibility of accelerating the studied application using the dynamic workload prediction and workload balancing. In particular, we are planning to incorporate the algorithm presented in this paper, which is based on 1D computational map, into the modified code which uses the *loop fusion* technique. Also, it is expected to develop another algorithm for workload prediction and load balancing which uses 2D computational map. This solution should allow adjusting the domain of computations to the domain of simulation more accurately.

Acknowledgments. This research was conducted with the financial support of the National Science Centre (Poland) under grants no. UMO-2017/26/D/ST6/00687. The authors are grateful to: (i) Intel Technology Poland and (ii) Czestochowa University of Technology (MICLAB project no. POIG.02.03.00.24-093/13) for granting access to HPC platforms.

References

1. Adrian, H., Spiradek-Hahn, K.: The simulation of dendritic growth in Ni-Cu alloy using the phase field model. Arch. Mater. Sci. Eng. **40**(2), 89–93 (2009)
2. Benito, J.J., Ureñ, F., Gavete, L.: The generalized finite difference method. In: Àlvarez, M.P. (ed.) Leading-Edge Applied Mathematical Modeling Research, pp. 251–293. Nova Science Publishers, New York (2008)

3. Folch, R., Casademunt, J., Hernandez-Machado, A., Ramirez-Piscina, L.: Phase-field model for Hele-Shaw flows with arbitrary viscosity contrast. II. Numerical study. Phys. Rev. E **60**(2), 1734–1740 (1999)
4. Halbiniak, K., Wyrzykowski, R., Szustak, L., Olas, T.: Assessment of offload-based programming environments for hybrid CPU-MIC platforms in numerical modeling of solidification. Simul. Model. Pract. Theory **87**, 48–72 (2018)
5. Karma, A., Kessler, D., Levine, H.: Phase-field model of mode III dynamic fracture. Phys. Rev. Lett. **87**(4), 045501 (2001). https://doi.org/10.1103/PhysRevLett.87.045501
6. Kulawik, A.: The modeling of the phenomena of the heat treatment of the medium carbon steel. Wydawnictwo Politechnki Czestochowskiej, (281) (2013). (in Polish)
7. Provatas, N., Elder, K.: Phase-Field Methods in Materials Science and Engineering. Wiley, Weinheim (2010)
8. Shimokawabe, T., et al.: Peta-scale phase-field simulation for dendritic solidification on the TSUBAME 2.0 supercomputer. In: Proceedings of the 2011 ACM/IEEE International Conference on High Performance Computing, Networking, Storage and Analysis, SC 2011. IEEE Computer Society (2011). https://doi.org/10.1145/2063384.2063388
9. Steinbach, I.: Phase-field models in materials science. Model. Simul. Mater. Sci. Eng. **17**(7), 073001 (2009). https://doi.org/10.1088/0965-0393/17/7/073001
10. Szustak, L., Halbiniak, K., Kulawik, A., Wrobel, J., Gepner, P.: Toward parallel modeling of solidification based on the generalized finite difference method using Intel Xeon Phi. In: Wyrzykowski, R., Deelman, E., Dongarra, J., Karczewski, K., Kitowski, J., Wiatr, K. (eds.) PPAM 2015. LNCS, vol. 9573, pp. 411–422. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-32149-3_39
11. Szustak, L., Rojek, K., Olas, T., Kuczynski, L., Halbiniak, K., Gepner, P.: Adaptation of MPDATA heterogeneous stencil computation to Intel Xeon Phi coprocessor. Sci. Prog. (2015). https://doi.org/10.1155/2015/642705
12. Szustak, L., Halbiniak, K., Kuczynski, L., Wrobel, J., Kulawik, A.: Porting and optimization of solidification application for CPU-MIC hybrid platforms. Int. J. High Perform. Comput. Appl. **32**(4), 523–539 (2018)
13. Takaki, T.: Phase-field modeling and simulations of dendrite growth. ISIJ Int. **54**(2), 437–444 (2014)
14. Warren, J.A., Boettinger, W.J.: Prediction of dendritic growth and microsegregation patterns in a binary alloy using the phase-field method. Acta Metall. Mater. **43**(2), 689–703 (1995)