Toward Parallel Modeling of Solidification Based on the Generalized Finite Difference Method Using Intel Xeon Phi

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

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

Abstract. Modern heterogeneous computing platforms have become powerful HPC solutions, which could be applied for a wide range of applications. In particular, the hybrid platforms equipped with Intel Xeon Phi coprocessors offers performance advantages over conventional homogeneous solutions based on CPUs, while supporting practically the same parallel programming model. However, there is still an open issue how scientific applications can utilize efficiently the hybrid platforms equipped with Intel coprocessors.

In this paper we propose a method for porting a real-life scientific application to computing platforms with Intel Xeon Phi. We focus on the parallel implementation of a numerical model of solidification, which is based on the generalized finite difference method. We develop a sequence of steps that are necessary for porting this application to platforms with accelerators, assuming no significant modifications of the code. The proposed method considers not only efficient data transfers that allow for overlapping computations with data movements, but also takes into account an adequate utilization of cores/threads and vector units. The developed approach allows us to execute the whole application 3.45 times faster than the original parallel version running on two CPUs.

Keywords: Intel Xeon Phi · Numerical model of solidification · Application porting · Optimization of data movements

1 Introduction

In the last years, it becomes evident [7,16] that future designs of microprocessors and HPC systems will be hybrid and heterogeneous in nature. An example of this trend are hybrid platforms equipped with the Intel Xeon Phi coprocessors [9,10]. These heterogeneous solutions rely on integration of two major types of components in various proportion to speed up computation intensive applications: (i) multicores CPU technology, as well as (ii) special-purpose hardware and massively parallel accelerators.

R. Wyrzykowski et al. (Eds.): PPAM 2015, Part I, LNCS 9573, pp. 411–422, 2016.
DOI: 10.1007/978-3-319-32149-3_39

The Intel Xeon Phi coprocessor [4,9] is the first product based on Intel Many Integrated Core (Intel MIC) architecture. It includes a large number of cores with wide vector processing units. It offers notable performance advantages over conventional homogeneous solutions based on CPUs, and supports practically the same parallel programming model. Although this architecture is designed for massively parallel applications, there is still an open issue how scientific applications can utilize efficiently the hybrid platforms equipped with Intel coprocessors.

In this study, we present an example of solving this problem by proposing a method for porting a real-life scientific application to computing platforms with Intel Xeon Phi. We focus on the parallel implementation of a numerical model of the dendritic solidification process in the isothermal conditions [1,14]. In this model, the growth of microstructure during the solidification process is determined by solving a system of two partial differential equations (PDEs). These equations define the phase content, and concentration of components in an alloy. The solutions of PDEs is obtained using the Meshless Finite Difference Method (with 2D geometry) and an explicit scheme of calculations.

In this paper, we present a sequence of steps that are necessary for porting application to platforms with accelerators, assuming no significant modifications of the code. In the proposed adaptation, the coprocessor is responsible for executing the major parallel workloads, while the CPU host is used only to execute the remaining part of the application, that do not require massively parallel resources. The main challenges here include not only providing efficient data transfers that overlap computations with data movements, but also ensuring an adequate utilization of cores/threads and vector units. The proposed method allows us to execute computations 3.45 faster than the original parallel code running on two CPUs.

This paper is organized as follows. Section 2 overviews the Intel Xeon Phi architecture, while Sect. 3 shows details of a target platform. Section 4 introduces the numerical model of solidification, which is based on the generalized finite difference method. Section 5 describes the idea of adaptation of the solidification algorithm to computing platforms with Intel Xeon Phi accelerators. Section 6 outlines a sequence of steps that are necessary for porting the main workloads of the application to the Intel Xeon Phi coprocessor, following the idea proposed in the previous section. Section 7 presents performance results obtained for the proposed method, while Sect. 8 concludes the paper.

2 Intel Xeon Phi Coprocessor Overview

The Intel Xeon Phi coprocessor is the first product based on the Intel Many Integrated Core Architecture (Intel MIC Architecture). Tt targets a variety of HPC segments [9,10,17] such as scientific research, physics, chemistry, biology, and climate simulation [12,13,17].

The coprocessor is equipped with more than 50 cores, caches, memory controllers, and PCIe client logic [4,5,11]. All these components are connected together by the bidirectional ring interconnect. Cores are clocked at about 1 GHz,

and allow for running up to 4 hardware threads per each core. An integral part of every core is the vector processing unit, that supports a new 512-bit SIMD instruction set called Intel Initial Many-Core Instructions. Each core has 128 vector registers 512-bit wide, and comes complete with a private L1 and L2 caches that are kept fully coherent by the ring interconnect. The coprocessor has over 6 GB of own on-board GDDR5 main memory (maximum 16 GB). The access to the main memory is realized by 6 or 8 memory controllers, that are evenly distributed on the bidirectional bus. The Intel Xeon Phi coprocessors are delivered in form factor of a PCIe additional device.

The Intel MIC architecture provides a general-purpose programming environment similar to that provided for Intel CPUs [9]. It supports the source-code portability between CPU and coprocessor platforms, that gives possibility to run the same code using different devices: Intel CPU or Intel Xeon Phi. Programmers can write source code using most popular programming languages like C, C++ and Fortran. This architecture supports also traditional parallel programming standards such as OpenMP, Intel Thread Building Blocks, Intel Cilk Plus, C++11 threads and MPI.

One of the basic methods to utilize Intel Xeon Phi computing resources is programming in the native mode [9]. In this mode, a source code is compiled on the host using the cross compiler to generate binary for the MIC architecture. Then, the executable application file can be copied and run directly on the coprocessor. Coprocessors support also the heterogeneous programming model known as the offload mode [9]. In this mode, the programmer select code sections to run on the Intel Xeon Phi. This model uses simple pragmas/directives to specify code sections and data to be offloaded to the target device. The application starts on the CPU side, while selected regions are automatically transferred and run on the target device. If for some reason, the Intel Xeon Phi is unavailable, the code regions are executed on the CPU.

3 Target Platform

In this study, we use the platform [10] equipped with two Intel Xeon E5-2695 v2 processors (Ivy Bridge EP architecture), and the top-of-the-line Intel Xeon Phi 7120P coprocessor. Every processor consists of 12 cores clocked at 2.4 GHz, and 128 GB DDR3-1866 main memory. The coprocessor contains 61 cores clocked at 1.238 GHz, and 16 GB of on-board memory. Two CPUs offer 2×230.4 Gflop/s of theoretical peak performance totally, assuming double precision floating-point operations, while a single coprocessor gives 1.2 Tflop/s. The values of peak performance are given taking into account the usage of SIMD vectorization (words 256- or 512-bit wide for CPU or coprocessor, respectively). Table 1 presents a summary of this platform.

Table 1. Specification of test platform [10]

	2 × Intel Xeon E5-2695 V2	Intel Xeon Phi 7120P
Number of cores	2 × 12	61
Number of threads	2 × 24	244
SIMD length [bits]	256	512
Freq. [GHz]	2.4	1.2
Peak for DP [Gflop/s]	2 × 230.4	1208
L1/L2 cache [KB]	64/256	64/512
LLC[a] [MB]	2 × 30	28.5
Memory size [GB]	128	16
Memory bandwidth [GB/s]	2 × 59.7	352

[a]LLC (Last Level Cache) corresponds to either L3 cache for CPU, or aggregated L2 caches for Intel Xeon Phi.

4 Introduction to Numerical Model of Solidification

In the analyzed numerical examples, a binary alloy of Ni-Cu is considered as a system of the ideal metal mixture in the liquid and solid state. The numerical model [1,14] refers to the dendritic solidification process in the isothermal conditions with constant diffusivity coefficients for the liquid and solid phases. It allows to use the field phase model defined by Warren and Boettinger [14]. In this model, the growth of microstructure during the solidification process is determined by solving a system of two PDEs [1,8,14]. The first equations defines the phase content ϕ:

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

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

$$\frac{\partial c}{\partial t} = \nabla\cdot D_c\left[\nabla c + \frac{V_m}{R}c\,(1-c)\left(H_B\left(\phi,T\right) - H_A\left(\phi,T\right)\right)\nabla\phi\right], \tag{2}$$

where: D_c is the diffusion coefficient, V_m is the specific volume, R is the gas constant. In this model, the Meshless Finite Difference Method [2,6] is used to obtain the values of partial derivatives with respect to dimensions x and y, that occur in Eqs. 1 and 2.

In order to parallelize computations with a desired accuracy, the explicit scheme is applied with a small value of the time step $\Delta t = 1e-7s$:

$$\phi_i^{t+1} = \phi_i^t + \Delta t M_\phi \left[\varepsilon^2 \eta^2 \left(\frac{\partial^2 \phi_i^t}{\partial x^2} + \frac{\partial^2 \phi_i^t}{\partial y^2} \right) - cH_B - (1-c) H_A - cor + \right.$$
$$\left. + 2\varepsilon^2 \eta\eta' \left(\frac{\partial \phi_i^t}{\partial x} \frac{\partial \theta}{\partial x} + \frac{\partial \phi_i^t}{\partial y} \frac{\partial \theta}{\partial y} \right) + \varepsilon^2 \left(\eta'^2 + \eta\eta'' \right) \left(\frac{\partial \phi_i^t}{\partial x} \frac{\partial \theta}{\partial y} - \frac{\partial \phi_i^t}{\partial y} \frac{\partial \theta}{\partial x} \right) \right], \tag{3}$$

$$c_i^{t+1} = c_i^t + \Delta t \left\{ \frac{\partial D_c}{\partial x} \left[\frac{\partial c}{\partial x} + \frac{V_m}{R} c(1-c)(H_B - H_A) \frac{\partial \phi}{\partial x} \right] + \right.$$
$$+ D_c \left[\frac{\partial^2 c}{\partial x^2} + \frac{V_m}{R} \left[(1-2c) \frac{\partial c}{\partial x} (H_B - H_A) \frac{\partial \phi}{\partial x} + \right. \right.$$
$$\left. \left. + c(1-c) \frac{\partial (H_B - H_A)}{\partial x} \frac{\partial \phi}{\partial x} + c(1-c)(H_B - H_A) \frac{\partial^2 \phi}{\partial x^2} \right] \right] +$$
$$+ \frac{\partial D_c}{\partial y} \left[\frac{\partial c}{\partial y} + \frac{V_m}{R} c(1-c)(H_B - H_A) \frac{\partial \phi}{\partial y} \right] + \tag{4}$$
$$+ D_c \left[\frac{\partial^2 c}{\partial y^2} + \frac{V_m}{R} \left[(1-2c) \frac{\partial c}{\partial y} (H_B - H_A) \frac{\partial \phi}{\partial y} + \right. \right.$$
$$\left. \left. + c(1-c) \frac{\partial (H_B - H_A)}{\partial y} \frac{\partial \phi}{\partial y} + c(1-c)(H_B - H_A) \frac{\partial^2 \phi}{\partial y^2} \right] \right] \Bigg\},$$

where θ is a function of derivatives of ϕ-variable.

These computations correspond to the forward-in-time algorithms [16]. The application code consists of two main blocks of computations, which are responsible for determining either the phase content ϕ and the dopant concentration c. In the model, the values of ϕ and c are calculated for nodes uniformly distributed across the square region, where 751 nodes along every dimension are chosen as sufficient for providing a required accuracy in the example examined in this paper. The values of derivatives at all nodes (Eqs. 3 and 4) are determined at every time step of calculations. All these computations are the main workload for the resulting numerical algorithm.

5 Idea of Adapting the Solidification Application to Computing Platforms with Intel Xeon Phi

In this section, we consider the idea of adaptation of the solidification application to computing platforms with Intel Xeon Phi accelerators. The main goal of our study is to accelerate calculations using Intel Xeon Phi. We propose to employ the coprocessor to perform the major parallel workloads, and use CPU only to execute the rest of application that do not require massively parallel resources.

As a result, the computations that correspond to Eqs. 3 and 4 are performed using the Intel Xeon Phi coprocessor.

In the studied application, computations are interleaved with writing partial results to a file. In the original version, parallel computations are executed for subsequent time steps, while writing results to the file is performed after the first time step, and then after every 100 time steps. The CPU does not perform any computations during writing results to the file. Figure 1 illustrates execution of the computational core of the studied application using CPU only.

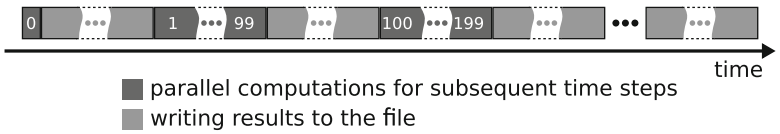

Fig. 1. Implementation of the original version of solidification algorithm

We propose to use Intel Xeon Phi for executing all operations associated with the computational core, and make the CPU responsible for writing partial results to the file. This requires to perform adequate data transfers between processor and coprocessor. Since the studied application belongs to the forward-in-time algorithms, where subsequent time steps depend on the previous ones, all the input data required for the first time step are transferred to the coprocessor, while others input data necessary for subsequent time steps are computed by the coprocessor. So after finishing computations for the first time step, and then for every package of 100 time steps, the coprocessor transfers its outcomes back to the host processor that is responsible for writing these results to the file.

A critical performance challenge here is to overlap workload performed by Intel Xeon Phi with data movements. To reach this goal, we propose to perform simultaneously data transfers between the CPU and Intel Xeon Phi, writing data to the file, as well as computations. This idea is illustrated in Fig. 2.

After finishing computations for the 1st time step, both transfers of the partial results from the coprocessor to CPU and computations for the next package

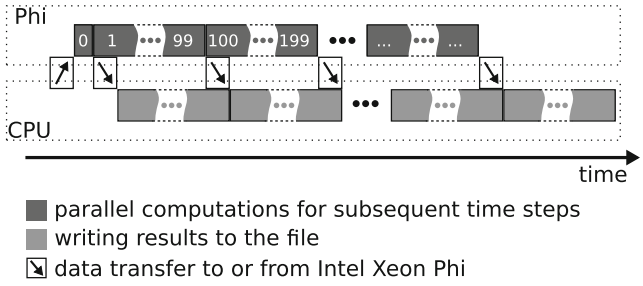

Fig. 2. Idea of parallelization of numerical modeling of solidification using both Intel CPU and Intel Xeon Phi

of 100 time steps are started in the same time. Then during computations for the package of 100 time steps, and after transferring data, the CPU writes required data to the file. Such a scheme repeats for every package of 100 time steps, and gives possibility for overlapping of computations and data movements only when the time of computations will be not shorter than the time of data movements.

6 Porting Application to Intel Xeon Phi

This section presents a sequence of steps that are necessary for porting the main workloads of the application to the Intel Xeon Phi coprocessor, following the idea proposed in the previous section. The first challenge includes overlapping the following operations: (i) data writing to the file on the CPU side, (ii) computations on the coprocessor side, (iii) data transfers between the coprocessor and CPU. The second challenge concerns an adequate utilization of Intel Xeon Phi computing resources. At the same time, we assume no significant modifications of the code, especially for the computing kernels. In this study, we do not provide any improvements for data writing to the file. Furthermore, only the massively parallel regions of the application are transferred to the coprocessor, while the remaining parts are executed on the CPU side. Such a distribution allows for an appropriate utilization of both the coprocessor designed for parallel computations, and the CPU that features a more general usage.

6.1 Optimization of Data Movements

Determining a set of data that have to be transferred between the coprocessor and CPU is the first step for porting the studied application to Intel Xeon Phi. Since the coprocessor is used in the offload mode, the data transfers through the PCIe bus are crucial for the overall performance [4,5]. Generally speaking, the total amount of data transfers between the coprocessor and processor has to be maximally reduced.

This aim is addressed by the proposed idea of adaptation presented in Sect. 5. Following this idea, the required input data are transferred to the coprocessor only once before computations. Then, the appropriate portion of data has to be exchanged after every package of 100 time steps (see Fig. 2).

Selecting an appropriate method for providing efficient data transfers is important for the overall performance. A basic solution to provide the desired efficiency is to ensure a linear (or continuous) access for the required data. It is achieved by choosing an appropriate data structure. Typically, there are two major possibilities for laying out memory [3]: array of structures (AoS) and structure of arrays (SoA). The original version of the studied application utilizes the AoS option. In this case, a periodic access to the required data and/or copying some unnecessary data are necessary for transferring data to and from the coprocessor. To avoid these overheads in the proposed approach, we migrate to the SoA option in order to guarantee both the linear access, and transferring only the necessary data.

```
for(/*...*/)
{
   // Parallel computation for subsequent time steps on Intel MIC side
   #pragma offload target(mic) signal(&offload)
   /*...*/

   // Asynchronous receiving data from coprocessor
   #pragma offload_transfer target(mic)
   /*...*/

   // Asynchronous writing data to the file on CPU side
   /*...*/

   // Waiting for computation finishing
   #pragma offload_wait target(mic) wait(&offload)
   /*...*/
}
```

Fig. 3. Asynchronous data transfers between CPU and coprocessor

To overlap computations and data movements, the asynchronous transfers between the CPU and coprocessor is utilized. The offload mode supports such a solution be applying an adequate sequence of pragmas, as shown in Fig. 3.

The next step is associated with exploration of multiple buffering techniques [15]. To provide simultaneous computations and communications, it is enough to apply two buffers on the coprocessor side, that are responsible for keeping results of subsequent packages of 100 time steps. When one buffer is used for saving results during computation of a given package of 100 time steps, another one is employed for transferring results of the previous package from the coprocessor to CPU. This is achieved at the cost of some extra memory space. Moreover, the double buffering technique is also deployed on the CPU side, which is necessary to overlap data writing to the file with data transfers from the coprocessor to CPU. When data writing to the file utilizes one buffer, the results are transferred to the next one, and so on. In order to improve the overall performance, the memory regions for the buffers are allocated only once for the coprocessor side, and then reused multiple times. It allows us to reduce the number of memory allocations in the offload mode that usually generates a significant performance overhead [4,5].

6.2 Multithreading and Vectorization Optimizations

The previous optimization steps give possibility to overlap computation, communication, as well as data writing to the file. Now the main constraint is to make the time required by computation for a package of 100 time steps no longer then the time of data writing (see Fig. 2). To reach this goal, both cores/threads and vector units should be successfully utilized.

The original version of the studied application employs the OpenMP parallel programming standard to utilize cores/threads. This version uses the basic work-sharing construction `#pragma omp parallel for` to assign work to all the available threads. Since the Intel Xeon Phi coprocessors supports the OpenMP standard, the application code can be successfully executed without any modifications. As a result, all the available threads of the Intel Xeon Phi coprocessor can be utilized for the joint problem solving. To ensure the best overall performance assuming no significant modifications of the code, different setups for the scheduling clauses are evaluated, including `static`, `dynamic`, and `guided`.

The next step required for porting the application to Intel Xeon Phi is associated with applying the vectorization of computations. The compiler-based automatic vectorization [9] seems to be the most convenient method for achieving this goal. The automatic vectorization is provided by the Intel Compiler that automatically uses SIMD instructions available in the Intel Streaming SIMD Extensions. The compiler detects operations in the program that can be done in parallel, and then converts sequences of operations to parallel vector operations. In practice, the automatic vectorization usually occurs when the Intel compiler generates packed SIMD instructions through unrolling the innermost loop.

However, in the studied case the innermost loop can not be vectorized safely because of complexity of computations as well as data dependencies. In fact, calculating a single output element in the innermost loop requires a set of input elements with dynamically determined indexes. An example of such a situation is presented in Fig. 4a. In this case, the automatic vectorization of computations fails because of an irregular data access, unpredictable during compilation. To solve this problem, we propose to change slightly the code by adding temporary vectors responsible for loading the necessary data from the irregular memory region. It is enough to provide SIMD computations. This idea is illustrated in Fig. 4b.

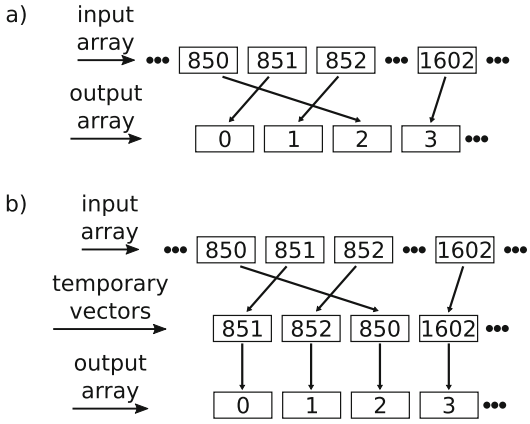

Fig. 4. Idea of vectorization: (a) scalar computation based on irregular data access, (b) vectorization of computation using temporary vectors

Additionally, appropriate keywords and directives should be provided as compiler hints, in order to improve the efficiency of auto-vectorization. The auto-vectorization is also assisted with applying an appropriate data alignment for the vectorized data. This forces the compiler to create data objects in memory aligned to specific byte boundaries.

7 Performance Results

In this section, we present performance results obtained for the approach described in Sects. 5 and 6, assuming double precision floating point numbers. All the benchmarks are compiled using the Intel icpc compiler (v.15.0.2) with the optimization flag -O3. The resulting code is executed in the offload mode on the platform equipped with the Intel Xeon E5-2695 V2 CPU and Intel Xeon Phi 7120P coprocessor, while the original version of the solidification application uses parallel resources of two CPUs. All the tests are performed for 2626 time steps, and the 564 001 nodes (751 nodes along dimensions x and y).

In our benchmarks, we evaluate different loop scheduling options (`static`, `dynamic`, `auto` and `guided`) with different configuration for the size of chunks. For all the performance tests, we achieve similar performance results with differences below 5 %. The best performance corresponds to the static scheduling with equal-sized chunks of loop assigned to threads.

We check also the impact of the auto-vectorization for the overall performance. For the vectorized regions of the code, the computation are accelerated of about 2 times. This relatively small performance gain is mainly caused by overheads required for providing the irregular data access (see Fig. 4). The total performance gain of vectorization decreases when not vectorizable regions are taken into account. In consequence, the final speedup of the code with vectorization is 1.6 against the scalar code.

Table 2 presents the performance results for modeling solidification obtained for the original and proposed (optimized) codes. The total execution of the basic version takes 335 s. It includes the aggregated time of computations (244 s) and the aggregated time of data writing (94 s). In this case, all computations are not overlapped with writing output data to the file (see Fig. 1).

As compared with the original version, the optimized code performs all the workloads 3.45 times faster. Its execution takes 97 s, and includes computations, data writing, and transfers from and to the coprocessor. The aggregate time required to write data to the file is the same as in the case of the original version (94 s), while the aggregate time of computations (80 s) is shorter about 3 times. The transfer of all the input data takes 2.41 s, and occurs only at the beginning of the application execution, while transfers of partial results from the coprocessor takes totally 0.07 s only.

For about 81.8 % of the application execution time, data movements are overlapped with computations. Moreover, almost all the computations performed by the Intel Xeon Phi device are overlapped with data movements, and this allows us to hide 99.26 % of computations behind data writing.

Table 2. Performance results for modeling solidification using original and optimized codes

	Original version	Optimized version
Total Time [s]	335	97
Aggregate time of computations [s]	242	80
Aggregate time of data writing [s]	94	94
Time of input data transfers from CPU to Intel Xeon Phi [s]	-	2.41
Aggregate time of output data transfers from Intel Xeon Phi to CPU [s]	-	0.07

8 Conclusions

Using computing platforms with Intel Xeon Phi is a promising direction for improving the efficiency of parallel computations in the numerical modeling of solidification. Porting applications to the Intel MIC architecture requires to use the native or offload mode, where the latter proved to be a better solution for the studied application. In this paper, we propose the method for porting and optimizing the application for modeling alloy solidification on computing platforms with Intel Xeon Phi. In comparison with the original parallel version of the code, the optimized version performs all the workloads 3.45 times faster.

The offload mode is an efficient solution for porting large codes that consist of both massively parallel and sequential regions to platforms with Intel MIC. Using the coprocessor to perform major parallel workloads, and employing the CPU only to execute the rest of an application give a strong possibility to accelerate the whole application. Such a workload distribution allows for an appropriate utilization of both the coprocessor designed for massively parallel computations, and the CPU that is designed for the general usage.

The Intel MIC architecture provides a general-purpose programming environment that allows for quick and easy code porting. However, the utilization of the offload mode requires to implement data transfers between the processor and coprocessor efficiently. The solution of this issue is provided by the proposed method of the adaptation of the application for modeling alloy solidification.

This method allows us to overlap (i) data writing to the file on the CPU side, (ii) computations on the coprocessor side, and (iii) data transfers between the coprocessor and CPU. As a result, for about 81.8 % of the application execution time, data movements are overlapped with computations. When considering only the time of computations, the proposed adaptation allows us to hide 99.26 % of computations behind data movements. Such a high degree of overlapping is achieved by applying multithreading and vectorization optimizations. This allows us to reduce the time of executing the parallel workload by the coprocessor, and makes it no longer then the time of data movements.

Acknowledgments. The authors are grateful to the Czestochowa University of Technology for granting access to Intel CPU and Xeon Phi platforms providing by the MICLAB project No. POIG.02.03.00.24-093/13.

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 Inc. (2008)
3. Hager, G., Wellein, G.: Introduction to High Performance Computing for Scientists and Engineers. CRC Press, Boca Raton (2011)
4. Intel Xeon Phi Coprocessor System Software Developers Guide. Intel Corporation (2013)
5. Jeffers, J., Reinders, J.: Intel Xeon Phi Coprocessor High-Performance Programming. Elsevier Inc., Waltham (2013)
6. Kulawik, A.: The modeling of the phenomena of the heat treatment of the medium carbon steel. Wydawnictwo Politechnki Czestochowskiej, Monografia, no. 281 (2013) (in Polish)
7. Kurzak, J., Bader, D., Dongarra, J. (eds.): Scientific Computing with Multicore and Accelerators. CRC Press, Boca Raton (2011)
8. Longinova, T., Amberg, G., Ågren, J.: Phase-field simulations of non-isothermal binary alloy solidification. Acta Mater. **49**(4), 573–581 (2001)
9. Parallel Programming and Optimization with Intel Xeon Phi Coprocessors, Handbook on the Development and Optimization of Parallel Applications for Intel Xeon Processors and Intel Xeon Phi Coprocessors. Colfax International, Sunnyvale, CA (2013)
10. Pilot Laboratory of Massively Parallel Systems (MICLAB). http://miclab.pl
11. Rahman, R.: Intel Xeon Phi Coprocessor Architecture and Tools: The Guide for Application Developers. APress, Berkeley (2013)
12. Szustak, L., Rojek, K., Gepner, P.: Using Intel Xeon Phi coprocessor to accelerate computations in MPDATA algorithm. In: Wyrzykowski, R., Dongarra, J., Karczewski, K., Waśniewski, J. (eds.) PPAM 2013, Part I. LNCS, vol. 8384, pp. 582–592. Springer, Heidelberg (2014)
13. Szustak, L., Rojek, K., Olas, T., Kuczynski, L., Halbiniak, K., Gepner, P.: Adaptation of MPDATA heterogeneous stencil computation to Intel Xeon Phi coprocessor. Sci. Program. **2015**, 14 (2015)
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)
15. Wyrzykowski, R., Rojek, K., Szustak, L.: Model-driven adaptation of double-precision matrix multiplication to the Cell processor architecture. Parallel Comput. **38**(4–5), 260–276 (2012)
16. Wyrzykowski, R., Szustak, L., Rojek, K.: Parallelization of 2D MPDATA EULAG algorithm on hybrid architectures with GPU accelerators. Parallel Comput. **40**(8), 425–447 (2014)
17. Xue, W., Yang, C., Fu, H., Wang, X., Xu, Y., Liao, J., Gan, L., Lu, Y., Ranjan, R., Wang, L.: Ultra-scalable CPU-MIC acceleration of mesoscale atmospheric modeling on Tianhe-2. IEEE Trans. Comput. **64**(8), 2382–2393 (2015)