Special Issue Paper

International Journal of HIGH PERFORMANCE COMPUTING APPLICATIONS

The International Journal of High Performance Computing Applications
2018, Vol. 32(4) 523–539

DOI: 10.1177/1094342016677740
journals.sagepub.com/home/hpc
SAGE

Porting and optimization of solidification application for CPU–MIC hybrid platforms

Lukasz Szustak, Kamil Halbiniak, Lukasz Kuczynski, Joanna Wrobel and Adam Kulawik

Abstract

Modern heterogeneous computing platforms have become powerful HPC solutions, which could be applied to a wide range of real-life applications. In particular, the hybrid platforms equipped with Intel Xeon Phi coprocessors offer the advantages of massively parallel computing, while supporting practically the same parallel programming model as conventional homogeneous solutions. However, there is still an open issue as to how scientific applications can efficiently utilize hybrid platforms with Intel MIC coprocessors. In this article, we propose an approach for porting a real-life scientific application to such hybrid platforms, assuming no significant modifications of the application code. It allows us to take advantage of all the computing components, including two CPUs and two coprocessors, for the parallel execution of computational workloads. In this study, we focus on the parallel implementation of a numerical model of the dendritic solidification process in isothermal conditions. We develop a sequence of steps that are necessary for the porting and optimization of the solidification application to hybrid platforms with Intel coprocessors. The main challenges include not only overlapping data movements with computations, but also ensuring adequate utilization of cores/threads and vector units of processors, as well as coprocessors. To reach this aim, we propose an efficient and flexible method for the workload distribution between heterogeneous computing components. For implementing the potential benefits of the proposed approach, we choose a heterogeneous programming model based on a combination of the offload mode for Intel MIC and OpenMP programming standard. The developed approach allows us to execute the whole application up to 9.33 $\times$ faster than the original parallel version that uses two CPUs. Furthermore, the CPU–MIC hybrid platforms enable achieving the speedup of about 1.9 $\times$ that of the CPU platform with 24 cores based on the Ivy Bridge architecture, and about 1.5 $\times$ that of the Haswell-based CPU platform with 36 cores.

Keywords

Code optimization, heterogeneous programming model, hybrid architecture, Intel Xeon Phi, load balancing, numerical modeling of solidification, offload, OpenMP, partitioning, vectorization

1 Introduction

In recent years, it has become evident (Kurzak et al., 2011; Wyrzykowski et al., 2014a) that future designs of microprocessors and HPC systems will be hybrid and heterogeneous in nature. An example of this trend is hybrid platform equipped with the Intel Xeon Phi coprocessors (Parallel Programming, 2013; MICLAB, 2015) or GPU accelerators (Wyrzykowski et al., 2014a). These heterogeneous solutions rely on the integration of two major types of components in various proportions to speed up computation intensive applications: (a) multicore CPU technology and (b) special-purpose hardware and massively parallel accelerators.

The Intel Xeon Phi coprocessor (Parallel Programming, 2013; Intel Corporation, 2013) is the first product based on the Intel Many Integrated Core (Intel MIC) architecture. It includes a large number of cores with wide vector processing units, and supports practically the same parallel programming model as conventional homogeneous solutions based on CPUs. Although this architecture is designed for massively parallel applications, there is still an open issue how

Insitute of Computer and Information Science, Czestochowa University of Technology, Poland

Corresponding author:
Lukasz Szustak, Czestochowa University of Technology, Dabrowskiego 69, 42-201 Czestochowa, Poland.
Email: lszustak@icis.pcz.pl

scientific applications can efficiently utilize all components of CPU–MIC hybrid platforms.

In this study, we present an example of solving this problem by developing an approach for porting a real-life scientific application to such platforms. We focus on the parallel implementation of a numerical model of the dendritic solidification process in isothermal conditions (Adrian and Spiradek-Hahn, 2009; Warren and Boettinger, 1995). In this model, the growth of the 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 solution of PDEs is obtained using the meshless finite difference method (with 2D geometry), and an explicit scheme of calculations.

In our previous work (Szustak et al., 2016), we developed an approach for porting and optimizing an application for modeling alloy solidification on computing platforms with a single Intel Xeon Phi accelerator. That work presented a sequence of steps required for porting the main workloads of the application to the Intel Xeon Phi coprocessor. In this study, the coprocessor was used to perform the major parallel workloads, while the CPU was responsible for writing partial results to a file. Additionally, the CPU executed the part of the application that does not require massively parallel resources. The proposed approach allowed us to overlap (a) writing data to the file on the CPU side, (b) computations on the coprocessor side and (c) data transfers between the coprocessor and CPU. In consequence, the optimized version of the code was $3.45\times$ faster than the original parallel version.

In this article, we propose a new approach that allows us to accelerate the solidification application by efficiently using all the available computing resources of hybrid platforms equipped with Intel Xeon CPUs and Intel Xeon Phi coprocessors. In the proposed approach, both CPUs and coprocessors are responsible for performing the major parallel workloads to provide the best utilization of computing resources. To reach this aim, we develop a sequence of steps required for porting the application to hybrid platforms with accelerators, assuming no significant modifications of the code. The main challenges include not only overlapping data movements with computations, but also ensuring an adequate utilization of cores/threads and vector units of CPUs as well as coprocessors. In order to utilize all the components of hybrid platforms, we propose an efficient and flexible method for workload distribution between two CPUs and two coprocessors. For implementing the potential benefits of the proposed approach, we choose the heterogeneous programming model based on a combination of the offload model for the Intel MIC and the OpenMP programming standard.

To summarize, our article makes the following contributions.

1. An approach for porting the real-life scientific application to hybrid platforms with Intel MIC coprocessors, without significant modification of the program code, is proposed. It allows us to accelerate the original parallel code by taking advantage of all the computing devices, including CPUs and coprocessors, for the parallel execution of computational workloads.
2. The following optimizations are developed to accelerate the application: (a) optimization of data transfers between devices, and overlapping data movements with computations; (b) efficient workload distribution between available devices; (c) even distribution of workloads across cores/threads within devices, as well as optimization of partitioning CPU threads between work teams; (d) increased performance of vector processing by combining automatic vectorization and slight modifications of the code.
3. An experimental evaluation of the numerical accuracy of the developed approach is provided, taking into account differences in the architecture of Intel coprocessors compared to Intel Xeon CPUs. This evaluation shows that the simulation results for the developed and original codes differ negligibly. These differences are not cumulative, and they do not grow during the simulation.
4. Using the heterogeneous programming model, the developed implementation of the proposed approach allows us to execute the whole application up to $9.33\times$ faster than the original parallel CPU code. Moreover, the CPU–MIC hybrid implementation achieves the speedup of about $1.9\times$ against two Ivy Bridge-based CPUs with 24 cores, and about $1.5\times$ against the Haswell-based CPUs with 36 cores.

This article is organized as follows. Section 2 presents related work, while Section 3 outlines hybrid platforms equipped with Intel Xeon Phi coprocessors, including the platforms used in our experiments. Section 4 introduces the numerical model of solidification, which is based on the generalized finite difference method. Section 5 describes the idea of the adaptation of the solidification application to hybrid platforms, while Section 6 outlines a sequence of steps required for porting the application to CPU–MIC platforms, following the idea proposed in Section 5. Section 7 presents the evaluation of the proposed approach taking into account the numerical accuracy of computations performed in the heterogeneous environment. Section 8 shows performance results achieved by the proposed approach, while Section 9 concludes the article.

2 Related work

In order to reach the desired high computational power with low energy usage, many HPC systems are equipped not only with standard CPUs, but also accelerators. Three of the TOP500's top 10 sites for June 2016 exploit GPUs or Xeon Phi coprocessors, including the Tianhe-2 supercomputer which is ranked in second place (see http://top500.org). This machine contains a mixture of Intel Xeon E5-2692v2 and Intel Xeon Phi 31S1P devices.

The Intel MIC architecture is a relatively fresh computing platform; however, many researchers have been investigating this product in order to accelerate their applications. An exhaustive collection of such investigations is included in the Intel Xeon Phi Applications and Solutions catalogue (Liviero, 2015). Exploring a wide variety of HPC systems such as supercomputing systems based on multi- and many-core solutions is necessary to meet the challenges of modern science and technology.

Current investigations of porting existing parallel codes to Intel Xeon Phi coprocessors reveal many successes in optimizing specific computing kernels, and improving the performance in comparison with unoptimized versions (Szustak, Rojek, Wyrzykowski, et al., 2014; Szustak et al., 2016). A common conclusion is that the major performance improvements are usually noticeable for a single coprocessor when comparing a new optimized version versus an original parallel code for CPUs (e.g. Szustak et al., 2015, 2016). However, the improvements delivered for coprocessors are commonly suitable for CPUs as well (Szustak et al., 2015; Vladimirov, 2015). In consequence, the overall performance for a coprocessor is similar or even lower when compared to a single node with two Intel Xeon CPUs (Liu and Deng, 2015). Such a situation is not surprising taking into account recent improvements of processor architecture (Intel Corporation, 2015), and reported difficulties in achieving high performance on coprocessors (Liu et al., 2014; Liu and Deng, 2015; Vladimirov, 2015). At the same time, the usage of Intel Xeon Phi coprocessors has allowed the Tianhe-2 supercomputer to achieve petascale performance in earthquake modeling simulation (Heinecke et al., 2014).

The ability to fully exploit modern heterogeneous HPC systems becomes vital for achieving an optimal overall performance. An example of research in this direction is the methodology proposed in our previous works (Wyrzykowski et al., 2014a,b) for a stencil-based algorithm. It enabled us to efficiently utilize available resources by spreading computations across the entire CPU + GPU hybrid platforms. A new level of heterogeneous concurrent execution of a Monte Carlo photon transport simulation was presented by Wolfe et al. (2014). This simulation was extended to execute on any combination of CPUs, GPUs and MICs concurrently. The proposed approach allows each device to repeatedly grab portions of the domain, and compute concurrently until the entire domain has been simulated. A speedup of $13\times$ was observed when utilizing Intel Xeon X5650 CPU and Intel Xeon Phi 5110P coprocessors and an NVIDIA K40 GPU concurrently versus just the Intel Xeon CPU. Mapping parallel graph processing applications on a node with Intel Xeon Phi accelerators is a challenge considered by Chen et al. (2015). For this aim, the authors developed a specialized API for expressing SIMD parallelism, supported by efficient techniques, focusing on exploiting wide SIMD lanes, a massive number of cores, and partitioning of the work across CPUs and accelerators. The resulting heterogeneous CPU–MIC execution achieved a speedup of up to $1.41\times$ that of the CPU-only and MIC-only executions. At the same time, this solution required a deep interference in the basic code, while in our case we assume no significant modifications of the code.

The phase-field method is a powerful tool for solving interfacial problems in materials science (Steinbach, 2009). It has mainly been applied to solidification dynamics (Provatas and Elder, 2010), but it has also been used for other phenomena such as viscous fingering (Folch et al., 1999), fracture dynamics (Karma et al., 2001) and vesicle dynamics (Steinbach, 2009). The number of scientific papers related to the phase-field method exponentially increases from 1990 after Kobayashi's successful dendrite growth phase-field simulation, reaching about 400 positions in 2012 (according to the SCOPUS database) (Takaki, 2014).

In the initial period, calculations were carried out for single dendrites in 2D space. However, recent articles have described modeling a complex dendritic solidification with many grains, for the 3D space. One of the reasons for such progress is the fast growth of computing power. A convincing example of this trend is the peta-scale phase-field simulation of dendritic solidification performed on the TSUBAME2.0 supercomputer equipped with GPU accelerators (Shimokawabe et al., 2011). Therefore, the presented research is a part of the worldwide tendency to use modern computing machines for modeling the phase-field phenomena. There are many papers devoted to modeling the dendritic growth that use approaches such as cellular automata, the finite element method and the finite difference method (Adrian and SpiradekHahn, 2009; Choudhury et al., 2012; Zaeem et al., 2013). In this respect, the important contribution of this study is the utilization of the generalized finite difference method (GFDM). In particular, it allows us to model phenomena where the distribution of nodes in grids is diversified—concentrated in border areas of the interphase, and sparse in areas with a low diffusivity or already solidified. For such grids, it is not possible to

use methods which require regular grids, as for example does the classic finite difference method.

3 Hybrid platforms overview

This section presents an overview of the hybrid platforms equipped with conventional Intel Xeon CPUs and Intel Xeon Phi coprocessors. In particular, we outline the architecture of this platform, as well as introduce the programming model that allows for efficient utilization of available resources.

3.1 Intel CPU–MIC heterogeneous architecture

The heterogeneous Intel CPU–MIC architecture integrates in various proportions two major types of components (Colfax International, 2015; IT4Innovations, 2015): (a) general-purpose Intel processors and (b) massively parallel Intel Xeon Phi coprocessors. The HPC servers based on this architecture come in a variety of configurations to address diverse hardware, software, workload, performance and efficiency requirements. They also come in a variety of form factors, including even a very dense form factor that offers up to eight coprocessors per each server (Colfax International, 2015). However, a typical hybrid platform contains usually two Intel Xeon CPUs as well as one or two Intel Xeon Phi coprocessors (MICLAB, 2015; IT4Innovations, 2015).

An example of the heterogeneous Intel CPU–MIC platform is presented in Figure 1. The CPU processors are connected via the QuickPath Interconnect bus with ccNUMA capabilities (Intel Corporation, 2013; Parallel Programming, 2013). The Intel Xeon Phi coprocessors are delivered in form factor of a PCIe additional device, and do not provide a direct access to the main memory system of the server. This leads to the requirement of exchanging data through the PCIe bus between CPUs and coprocessors.

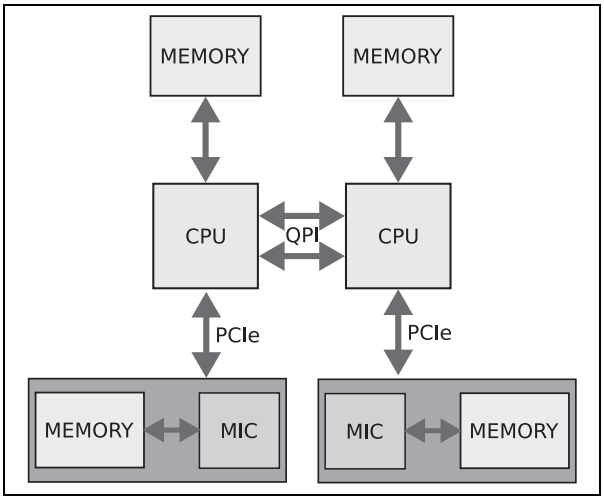

Figure 1. Heterogeneous CPU–MIC architecture.

3.2 Intel Xeon Phi coprocessor overview

The Intel Xeon Phi coprocessor is the first product based on the Intel MIC Architecture. It targets a variety of HPC segments (Parallel Programming, 2013; MICLAB, 2015; Xue et al., 2015) such as scientific research, physics, chemistry, biology, and climate simulation (Szustak, Rojek and Gepner, 2014; Szustak et al., 2015; Xue et al., 2015).

The coprocessor contains more than 50 cores, caches, memory controllers, and PCIe client logic (Intel Corporation, 2013; Rahman, 2013; Jeffers and Reinders, 2014). All these components are connected together by the bidirectional ring interconnect. Cores are clocked at about 1 GHz, and allow running up to four 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 six or eight memory controllers, that are evenly distributed on the bidirectional bus.

The Intel Xeon Phi coprocessor provides a general-purpose programming environment similar to that provided for Intel CPUs (Parallel Programming, 2013). It supports the source-code portability between CPU and coprocessor platforms, that gives the possibility to run the same code using different devices: Intel CPU or Intel MIC. Programmers can write their codes using the most popular programming languages like C, C++ and Fortran. This architecture supports also traditional parallel programming standards (Hager and Wellein, 2011) such as OpenMP, Intel Thread Building Blocks, Intel Cilk Plus, C++11 threads and MPI.

3.3 Programming models for hybrid platforms

The main challenge in achieving the desired computational performance is taking advantage of collaboration between CPUs and coprocessors. Generally speaking, to meet this challenge Intel offers two programming modes: MPI (or symmetric) mode and offload mode (Intel Corporation, 2013; Parallel Programming, 2013).

The first mode allows for running applications on CPUs and coprocessors using MPI, and then utilizing the computing resources of every kind of device by employing traditional multithreaded programming standards (e.g. OpenMP). The second mode also guarantees utilization of all the available resources, but

Table 1. Specification of target platforms (MICLAB, 2015).

Device type	First platform		Second platform	
	CPUs	Coprocessors	CPUs	Coprocessors
Number of devices	2	2	2	2
Name of each device	Intel Xeon E5-2699 V3	Intel Xeon Phi 7120P	Intel Xeon E5-2695 V2	Intel Xeon Phi 7120P
Release date	Q3 2014	Q2 2013	Q3 2013	Q2 2013
Number of cores	18	61	12	61
Number of threads	36	244	24	244
SIMD width (bits)	256	512	256	512
Frequency (GHz)	2.3	1.2	2.4	1.2
DP peak (Gflop/s)	662.4	1208.3	230.4	1208.3
LLC* size (MB)	45	30.5	30	30.5
Memory size (GB)	256	16	128	16
Memory bandwidth (GB/s)	68	352	59.7	352

*LLC (last level cache) refers to aggregated L2 caches for Intel Xeon Phi and L3 cache for CPU.

requires a different methodology for achieving this aim. In this mode, the programmer select code sections to run on the Intel Xeon Phi. This mode assumes using simple pragmas to specify code sections and data to be offloaded to a target device. The application starts on the CPU side, while selected regions are automatically transferred and run on the target device. A parallel programming standard such as OpenMP has to be additionally employed for utilizing the parallel computing resources of the processors and/or coprocessors. If for some reason, the coprocessor is unavailable, the code regions are executed on the CPU side.

An important advantage of the offload mode is an easy opportunity to avoid the overheads associated with running non-parallel regions of applications on the coprocessor side. In particular, massively parallel regions can be performed simultaneously by CPUs and processors, while sequential regions are executed by CPUs only. Since the studied application includes both sequential and massively parallel regions, we use a programming model which is a combination of the offload mode and the OpenMP programming standard. Such a combination gives a strong basis for the efficient utilization of CPUs and coprocessors by providing a flexible workload distribution.

3.4 Target platforms

In this study, we use two hybrid platforms (MICLAB, 2015) equipped with CPUs and Intel Xeon Phi coprocessors. The first platform includes two Intel Xeon E5-2699 v3 CPUs (Haswell EP architecture), two Intel Xeon Phi 7120P coprocessors, and 256 GB of DDR4-2133 main memory. Every CPUs consists of 18 cores clocked at 2.3 GHz, while each coprocessor contains 61 cores clocked at 1.238 GHz with 16 GB of on-board memory. The second platform includes two Intel Xeon E5-2695 v2 CPUs (Ivy Bridge EP architecture), two Intel Xeon Phi 7120P coprocessors, and 128 GB of DDR3-1866 main memory. These processors contain 2×12 cores clocked at 2.4 GHz.

For double precision floating-point operations, the theoretical peak performances of these two platforms are 3741.4 ($2 \times 662.4 + 2 \times 1208.3$) Gflop/s and 2877.4 ($2 \times 230.4 + 2 \times 1208.3$) Gflop/s, respectively. The presented values of the peak performance take into account the usage of SIMD vectorization with the vector size of 256 bits for CPU, and 512 bits for coprocessor, respectively. Table 1 summarizes parameters of these platforms.

4 Introduction to numerical model of solidification

In the numerical examples studied in this article, a binary alloy of Ni–Cu is considered as a system of the ideal metal mixture in the liquid and solid phases. The numerical model (Warren and Boettinger, 1995; Adrian and Spiradek-Hahn, 2009) refers to the dendritic solidification process in the isothermal conditions with constant diffusivity coefficients for the liquid and solid phases. It allows us to use the field-phase model defined by Warren and Boettinger (1995). In this model, the growth of microstructure during the solidification process is determined by solving a system of two PDEs (Warren and Boettinger, 1995; Longinova et al., 2001; Adrian and Spiradek-Hahn, 2009). The first equations define the phase content ϕ

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

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 denote the free energy of both components and cor is the stochastic factor which models thermodynamic fluctuations near the dendrite tip. The coefficient θ is calculated as follows

$$\theta = \frac{\partial\phi}{\partial y}\Big/\frac{\partial\phi}{\partial x} \tag{2}$$

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

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

where: D_c is the diffusion coefficient, V_m is the specific volume, R is the gas constant. In this model, the GFDM (Benito et al., 2008; Kulawik, 2013) is used to obtain the values of partial derivatives, with respect to dimensions x and y, that occur in equations (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 $\Delta t = 1e-7s$

$$\phi_i^{t+1} = \phi_i^t + \Delta t M_\phi\varepsilon^2\left[\eta^2\left(\frac{\partial^2\phi_i^t}{\partial x^2}+\frac{\partial^2\phi_i^t}{\partial y^2}\right)\right.$$
$$+\ \eta\eta'\left(\sin(2\theta)\left(\frac{\partial^2\phi_i^t}{\partial y^2}-\frac{\partial^2\phi_i^t}{\partial x^2}\right)+2\cos(2\theta)\frac{\partial^2\phi_i^t}{\partial x\partial y}\right.$$
$$-\frac{1}{2}\left(\eta'^2+\eta\eta''\right)\left(-\cos(2\theta)\left(\frac{\partial^2\phi_i^t}{\partial y^2}-\frac{\partial^2\phi_i^t}{\partial x^2}\right)\right.$$
$$+\ 2\sin(2\theta)\frac{\partial^2\phi_i^t}{\partial x\partial y}-\frac{\partial^2\phi_i^t}{\partial x^2}-\frac{\partial^2\phi_i^t}{\partial y^2}\Big)$$
$$-cH_B-(1-c)H_A-\mathrm{cor} \tag{4}$$

$$c_i^{t+1} = c_i^t + \Delta t\left\{\frac{\partial D_c}{\partial x}\left[\frac{\partial c_i^t}{\partial x}+\frac{V_m}{R}c_i^t(1-c_i^t)(H_B-H_A)\frac{\partial\phi}{\partial x}\right]\right.$$
$$+\ D_c\left[\frac{\partial^2 c}{\partial x^2}+\frac{V_m}{R}\left[(1-2c_i^t)\frac{\partial c_i^t}{\partial x}(H_B-H_A)\frac{\partial\phi}{\partial x}\right.\right.$$
$$+\ c_i^t(1-c_i^t)\frac{\partial(H_B-H_A)}{\partial x}\frac{\partial\phi}{\partial x}$$
$$+\ c_i^t(1-c_i^t)(H_B-H_A)\frac{\partial^2\phi}{\partial x^2}$$
$$+\ \frac{\partial D_c}{\partial y}\left[\frac{\partial c_i^t}{\partial y}+\frac{V_m}{R}c_i^t(1-c_i^t)(H_B-H_A)\frac{\partial\phi}{\partial y}\right]$$
$$+\ D_c\left[\frac{\partial^2 c_i^t}{\partial y^2}+\frac{V_m}{R}\left[(1-2c_i^t)\frac{\partial c_i^t}{\partial y}(H_B-H_A)\frac{\partial\phi}{\partial y}\right.\right.$$
$$+\ c(1-c_i^t)\frac{\partial(H_B-H_A)}{\partial y}\frac{\partial\phi}{\partial y}$$
$$\left.\left.\left.\left.+\ c_i^t(1-c_i^t)(H_B-H_A)\frac{\partial^2\phi}{\partial y^2}\right]\right]\right\} \tag{5}$$

These computations belong to the group of forward-in-time algorithms (Wyrzykowski et al., 2014a), as all the calculations performed in the current time step $t+1$ depend on results determined in the previous time step t. 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 calculated for nodes distributed across a considered domain. In equations (4) and (5), the values of derivatives in all the nodes are determined at every time step of the calculations. All these computations are the main workload for the resulting numerical algorithm.

The solutions of differential equations are obtained on the basis of the meshless finite difference method (for 2D geometry) and explicit scheme of calculations (Benito et al., 2008). For approximating values of derivatives in equations (4) and (5), we adopt the second-order Taylor expansion. In this numerical scheme, to provide the "best" approximation of derivatives in the central node of the n-pointed star (see example shown in Figure 2, where $n = 12$), the following condition is used

$$D = \sum_{j=1}^{n}\left(T_i - T_j + (T,_x)_i h_j + (T,_y)_i k_j + 0.5(T,_{xx})_i h_j^2 + 0.5(T,_{yy})_i k_j^2 + 0.5(T,_{xy})_i h_j k_j\right)^2 \frac{1}{l_j^{2m}} = 0 \tag{6}$$

where:

- variable T corresponds either to the phase content ϕ or the dopant concentration c;
- $l_j = \sqrt{h_j^2 + k_j^2}$ is the distance between the jth node ($j = 1, 2, \ldots, 12$) and the central node corresponding to the index i;

- m is a coefficient which determines the influence of the distance of each node on the values of derivatives.

Assuming the coefficient $m = 2$, the usage of equation (6) allows us to express the searched derivatives by the following formula

$$\begin{pmatrix} \frac{\partial T_i}{\partial x} \\ \frac{\partial T_i}{\partial y} \\ \frac{\partial^2 T_i}{\partial x^2} \\ \frac{\partial^2 T_i}{\partial y^2} \\ \frac{\partial^2 T_i}{\partial y \partial x} \end{pmatrix} = \begin{pmatrix} \sum_{j=1}^{n} \frac{h_j^2}{l_j^{2m}} & \sum_{j=1}^{n} \frac{h_j k_j}{l_j^{2m}} & \sum_{j=1}^{n} \frac{h_j^3}{2l_j^{2m}} & \sum_{j=1}^{n} \frac{h_j k_j^2}{2l_j^{2m}} & \sum_{j=1}^{n} \frac{h_j^2 k_j}{l_j^{2m}} \\ \sum_{j=1}^{n} \frac{h_j k_j}{l_j^{2m}} & \sum_{j=1}^{n} \frac{k_j^2}{l_j^{2m}} & \sum_{j=1}^{n} \frac{h_j^2 k_j}{2l_j^{2m}} & \sum_{j=1}^{n} \frac{k_j^3}{2l_j^{2m}} & \sum_{j=1}^{n} \frac{h_j k_j^2}{l_j^{2m}} \\ \sum_{j=1}^{n} \frac{h_j^3}{2l_j^{2m}} & \sum_{j=1}^{n} \frac{h_j^2 k_j}{2h_j^{2m}} & \sum_{j=1}^{n} \frac{h_j^4}{4l_j^{2m}} & \sum_{j=1}^{n} \frac{h_j^2 k_j^2}{4l_j^{2m}} & \sum_{j=1}^{n} \frac{h_j^3 k_j}{2l_j^{2m}} \\ \sum_{j=1}^{n} \frac{h_j k_j^2}{2l_j^{2m}} & \sum_{j=1}^{n} \frac{k_j^3}{2h_j^{2m}} & \sum_{j=1}^{n} \frac{h_j^2 k_j^2}{4l_j^{2m}} & \sum_{j=1}^{n} \frac{k_j^4}{4l_j^{2m}} & \sum_{j=1}^{n} \frac{h_j k_j^3}{2l_j^{2m}} \\ \sum_{j=1}^{n} \frac{h_j^2 k_j}{l_j^{2m}} & \sum_{j=1}^{n} \frac{h_j k_j^2}{l_j^{2m}} & \sum_{j=1}^{n} \frac{h_j^3 k_j}{2l_j^{2m}} & \sum_{j=1}^{n} \frac{h_j k_j^3}{2l_j^{2m}} & \sum_{j=1}^{n} \frac{h_j^2 k_j^2}{l_j^{2m}} \end{pmatrix}^{-1} \times \begin{pmatrix} \sum_{j=1}^{n} \frac{h_j}{l_j^{2m}}(T_j - T_i) \\ \sum_{j=1}^{n} \frac{k_j}{l_j^{2m}}(T_j - T_i) \\ \sum_{j=1}^{n} \frac{h_j^2}{2l_j^{2m}}(T_j - T_i) \\ \sum_{j=1}^{n} \frac{k_j^2}{2l_j^{2m}}(T_j - T_i) \\ \sum_{j=1}^{n} \frac{h_j k_j}{l_j^{2m}}(T_j - T_i) \end{pmatrix}, \tag{7}$$

where $h = h_x$, $k = h_y$.

The presented approach, which is based on the GFDM, allows for solving the partial differential equations both for regular and irregular grids. In the studied application, a 2D regular grid is used with nodes distributed uniformly across a square domain, so we have $n = 8$. To provide a required accuracy, 2000 nodes along each dimension are chosen as sufficient.

At the same time, two cases should be distinguished for this application. In the first case, computations are performed in all the nodes. It is assumed that the values of concentration of alloy component and phase content could change in every node of the grid. For this case, the workload of computational nodes (CPUs and coprocessors) is constant during the application execution, since a constant number of equations has to be solved. These assumptions correspond to modeling problems in which the variability of the solidification phenomenon in the whole domain has to be taken into account. In the presented model, this occurs during the growth of grains in the considered domain, or when modeling the phase transformations in the solid state.

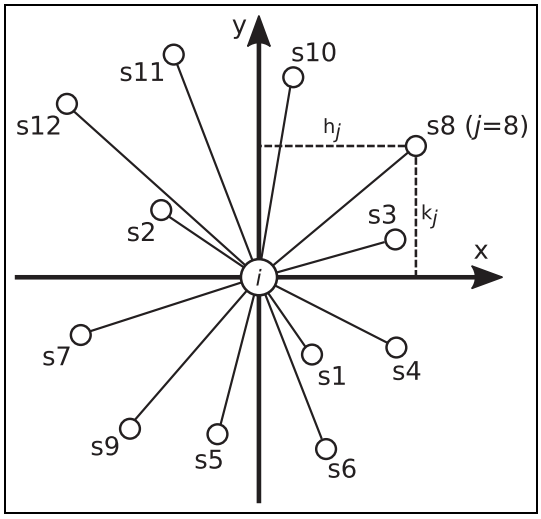

Figure 2. Example of grid nodes used for calculation.

In the second case, the model is able to solve the differential equations only in a part of the nodes. It is assumed that the required computations are performed only in the nodes in which the difference between values for the nodes of a single star are non-zero (Figure 2). The use of this selection criterion allows us to reduce significantly the number of computations. At the same time, the consequence is a significant workload imbalance between computing devices during the application execution, since the selection criterion is calculated after the partitioning of nodes between coprocessors and CPUs. In the initial stage of the grain growth, the differential equations are solved only in a small area around the condensation nuclei (this area is calculated on CPUs entirely). At that time, coprocessors check only if the selection criterion is satisfied.

5 Idea of adapting solidification application to hybrid platforms with coprocessors

In this section, we introduce the idea of the adaptation of the solidification application to a hybrid computing platform equipped with CPUs and coprocessors, where the emphasis is on platforms with two CPUs and two Intel accelerators. The main goal of this study is to accelerate computations by using all the available

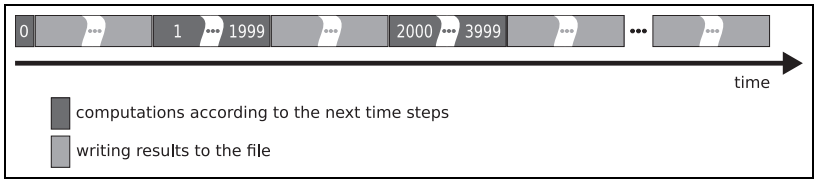

Figure 3. Timing diagram for the original version of solidification application.

computing resources of target platforms. We propose to execute the major parallel workloads on both CPUs and coprocessors, while the rest of the application is performed only by CPUs, as it does not require massively parallel resources. Such an approach allows us to utilize computational resources efficiently, but requires us to develop an efficient and flexible method for the workload distribution between CPUs and coprocessors. In consequence, the massively parallel regions that correspond to equations (4), (5) and (7) are executed both by CPUs and accelerators, while the other regions are processed by CPUs only.

In the studied application, computations are interleaved with writing partial results to a file. In the original version, parallel computations are executed for successive time steps, while writing results to the file is performed after the first time step, and then after executing every package of a selected number R of time steps. Such a scheme allows for observing and evaluating the grain growth during the simulation with a requested accuracy. The value of $R = 2000$ was selected in the original version of the application. Figure 3 illustrates the execution of the computational core of the studied application using CPUs only.

We propose using both CPUs and coprocessors to execute all the operations associated with the computational core, while leaving writing data to the file as the responsibility of CPUs only. A critical performance challenge of the proposed approach is the overlapping of all the computations with writing results to the file. Since CPUs are responsible for both computations and writing data to the file, we propose to partition CPU threads into two work teams, and assign writing data to one of them. More precisely, we assign a single CPU thread to the second team (this thread is responsible for writing data), while other CPU threads are assigned to the first one. In consequence, this approach allows us to simultaneously perform parallel computations using all the available devices (CPUs and accelerators), and writing data to the file.

The next challenge solved by our approach is related to an efficient and flexible distribution of the workload between all the available devices. For this aim, both the computation and communication costs are taken into account when ensuring load balancing between all the devices. Also, we propose to overlap computations with data movements as much as possible by scheduling simultaneously computations and data transfers to/from coprocessors, which is necessary to reduce the overhead caused by data transfers. The basic idea of adaptation of the solidification application to a hybrid platform with Intel Xeon Phi coprocessors is shown in Figure 4.

At the beginning of the computations, all the input data are transferred simultaneously to the coprocessors. Then the coprocessors and the first CPU work team starts computations for the first time step. After finishing the workloads assigned to both coprocessors, all the results are transferred back to the main memory. During these transfers, the first CPU work team performs computations for the first time step. The second CPU work team starts writing results to the file immediately after obtaining outcomes from the first CPU work team, as well as from the coprocessors. At the same time, the first CPU work team and the coprocessors start computations for the next package of time steps. This scheme is repeated many times for every package of R time steps.

Because of data dependencies in the studied application, it is necessary to exchange data between CPUs and coprocessors, between successive time steps. In particular, calculating a single output element by a device in every time step requires some input data which have been computed in other devices in the previous step. This leads to the necessity of providing adequate data transfers between CPUs and coprocessors, with the exchange point located in the main memory of CPUs.

6 Porting application to hybrid platforms with Intel Xeon Phi

This section presents a sequence of steps that are necessary for the adaptation of the studied application to hybrid platforms with Intel Xeon Phi coprocessors, following the idea proposed in the previous section. Among the three challenges solved in this way, the first one includes overlapping the following operations: (a) writing data to the file, (b) computations on the CPU and coprocessor sides and (c) data transfers between coprocessors and CPUs. The second challenge concerns an efficient and flexible distribution of workload between all the devices, while the last challenge refers to a maximum possible utilization of resources of CPUs and coprocessors. At the same time, we assume

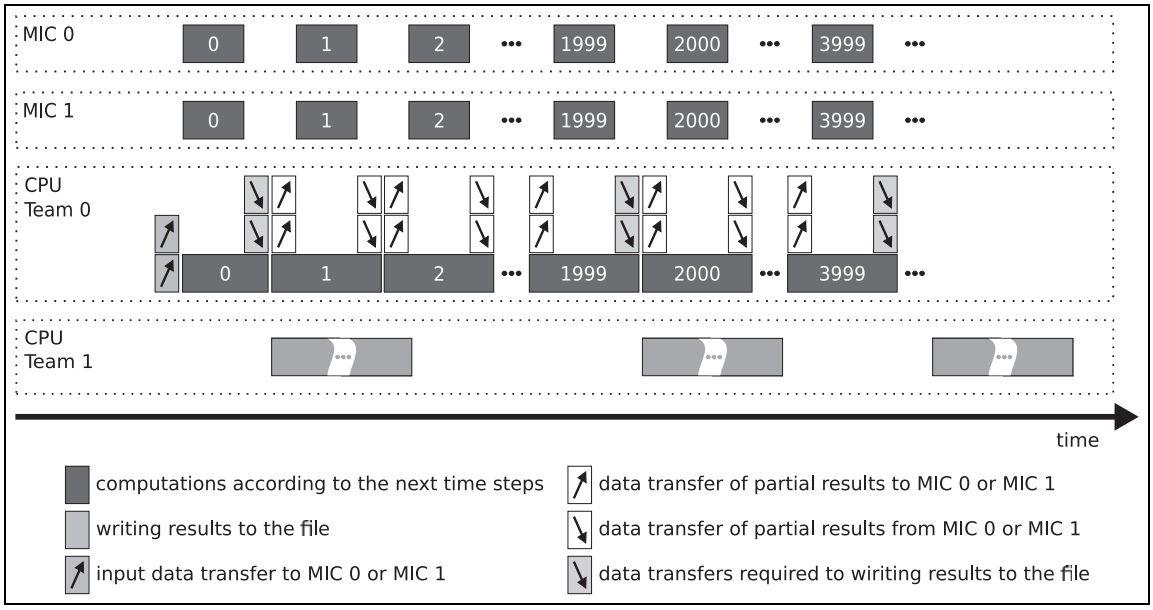

Figure 4. Timing diagrams illustrating the basic idea of adaptation of solidification application to hybrid platforms with Intel Xeon Phi coprocessors.

no significant modifications of the code, especially for the computing kernels. In this study, we do not provide any improvements for writing data to the file.

6.1 Partitioning CPU threads between work teams

When porting the studied application to hybrid platforms, one of the main steps is to overlap writing data to the file with computations performed by available devices. Since CPUs have to provide writing results to the file, we propose to partition the CPU threads into two work teams, where the second team, containing a single thread, is responsible for writing partial results to the file, while the main tasks of the first team are execution of calculations on CPUs and management of computations for coprocessors. The details of these tasks are presented in the next subsections.

To implement thread partitioning into two teams, we use the nested parallelism offered by the OpenMP programming standard (OpenMP, 2015). Using this feature, two CPU threads are created first, and then the first thread spawns new threads. When performing this step, providing the thread affinity is important to avoiding a situation where two or more threads run on the same core. Currently the affinity is adjusted manually. The concept of thread partitioning into two work teams is shown in Listing 1.

To overlap writing data with computations, it is important to use the multiple buffering technique (Wyrzykowski et al., 2012). In the proposed approach, it is enough to apply two buffers on the CPU side that are used alternatively for parallel computations and writing data. When one buffer is used to write results to the file, the second one is used to keep results of the computations.

```
int teamsNO = 2;
int teams[teamsNO];
teams[0] = threadsNO-1;
teams[1] = 1;
#pragma omp parallel num_threads(teamsNO)
{
  int idT = omp_get_thread_num();
  omp_set_nested(1);
  #pragma omp parallel num_threads(teams[idT])
  {
      /*...*/
  }
}
```

Listing 1. Partitioning threads into two work teams.

6.2 Partitioning and load balancing

The important step in porting our application is to provide for an optimal balancing of the workload between the platform components. This requires a suitable partitioning of computations. A target platform is viewed here as consisting of three components: (a) two CPUs connected as a ccNUMA architecture, (b) the first coprocessor (MIC0) and (c) the second coprocessor (MIC1). In consequence, all the computations, which are performed in the original algorithm to calculate several arrays, will be partitioned into three parts in a

proportion which takes into account performances of the platform components.

Furthermore, to reduce overheads for data transfers between the components we propose to assign the first part of computations to MIC0, the second part to CPUs and the last part to MIC1. Such an assignment has a clear advantage in data transfer overheads over the assignment of the adjacent parts of computations to two coprocessors. In the latter case, the required transfers take place for two communication paths: (a) MIC0 ↔ CPU ↔ MIC1 and (b) CPU ↔ MIC1. In the proposed case, the communication paths are shorter, and correspond to: (a) MIC0 ↔ CPU and (b) CPU ↔ MIC1. This is because the exchange point is located in the main memory of CPUs, and data transfers are necessary only between the adjacent parts, for every time step.

In order to ensure an optimal load balancing between all the components, we propose to determine the best workload distribution for a fixed problem size in an empirical way. Initially, computations are partitioned uniformly. Based on measurements of the execution time for each part, we perform a calibration of distribution. The calibration process is finished when the execution times for all the parts are the same, with a given accuracy. In consequence, the static partitioning is provided for successive executions of the application.

6.3 Parallelization of computations across threads

The previous optimization steps provide an efficient distribution of computations across all the devices, and overlapping computations with writing data to the file. Now the main challenge is to make the execution time of computations for a package of R time steps no longer than the time required for writing data to the file (see Figure 4). To reach this goal, all available cores/threads should be used successfully.

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 computations to all the available threads. Since Intel Xeon Phi coprocessors support the OpenMP standard, the application code can be successfully executed without any modifications on both CPUs and coprocessors. As a result, all the available threads of coprocessors and CPUs can be utilized to solve the modeling problem together. To ensure the best overall performance without significant modifications to the code, different setups for the scheduling clauses are evaluated for different devices, including `static`, `dynamic` and `guided`.

The joint work of coprocessors and CPUs requires running an extra task on the CPU side. Since Intel Xeon Phi is used in the offload mode, a task responsible for exchanging data and computations has to be submitted to and from accelerators between the subsequent time steps. Following the rules of the offload mode, the management of this task is assigned to the master thread of the first CPU team. The main disadvantage of this solution is related to management overheads that cause a workload unbalance on the CPU side, since the master thread is involved in computations as well. This situation is illustrated in Figure 5(a), which assumes the OpenMP static scheduling of loop iterations.

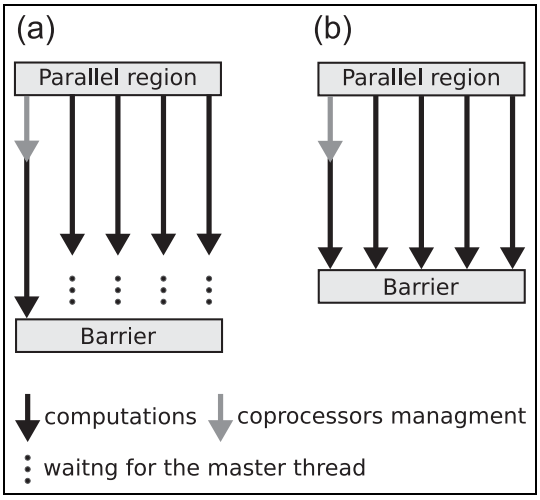

Figure 5. Parallelization of computations corresponding to either static scheduling (a) or dynamic scheduling (b).

In fact, the static scheduling splits a loop before computations, assigning the same or similar range of iterations to threads. Since the master thread is involved in the management of coprocessors and performs computations as well, the total time of its work is longer than that of the other threads. In consequence, the rest of the threads in the team have to wait until the master thread finishes tasks assigned to it.

In order to alleviate this overhead, we propose to use the OpenMP dynamic scheduling for parallel computations performed by CPUs, as is shown in Figure 5(b). It allows us to dynamically distribute the loop iterations among running threads during the application execution. The dynamic scheduling provides a run-time assignment of the next free block of loop iterations to the next idle thread. As a result, the workloads are distributed across the available CPU threads in a balanced way. In particular, the total amount of loop iterations assigned to the master thread will depend on the cost of the coprocessor management. After completing management activities, the master thread can start performing computations together with others threads.

6.4 Optimization of data movements

When porting the studied application to hybrid platforms, another major step is the optimization of data

transfers. As coprocessors are utilized in the offload mode, efficient data transfers through the PCIe bus are vital for the overall performance of computations. Generally speaking, the total size of transferred data as well as the total amount of data transfers have to be reduced to a minimum.

Following the previous section, all the necessary data are transferred to coprocessors only once at the beginning of computations. Then data are exchanged between coprocessors and CPUs in every time step. Therefore, it is necessary to determine a set of data that have to be transferred to and from coprocessors. Generally speaking, we transfer only data required for computations within the next time step. Additionally, asynchronous data transfers are utilized in order to overlap data movements with computations. This is achieved by applying a sequence of directives offered by the offload mode.

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 linear (or continuous) access for the required data. This is achieved by choosing an appropriate data structure. Typically, there are two major possibilities for laying out memory: array of structures (AoS) and structure of arrays (SoA). The original version of the studied application utilizes the AoS option. In this case, periodic access to the required data and/or copying some unnecessary data is required for transferring data to and from coprocessors. To avoid these overheads, we migrate to the SoA option in order to guarantee both linear access and transferral of only the necessary data.

Achieving the desired performance of data transfers requires a reduction in the number of memory allocations as well. In the proposed approach, data regions within the coprocessor memory are allocated only once at the beginning of computations, during the data exchange, and are then reused multiple times. This allows us to reduce significantly the performance overheads usually generated by memory allocations in the offload mode.

6.5 Vectorization

The next step required for ensuring the best possible performance of computations is their vectorization. The compiler-based automatic vectorization seems to be the most convenient method for achieving this goal. Automatic vectorization is provided by the Intel compiler that automatically uses SIMD instructions available in the Intel Streaming SIMD Extensions (Parallel Programming, 2013). The compiler detects operations in the program that can be executed in parallel, and then converts sequences of operations into parallel vector operations. In practice, the automatic vectorization usually occurs when the Intel compiler generates packed SIMD instructions through unrolling the innermost loop.

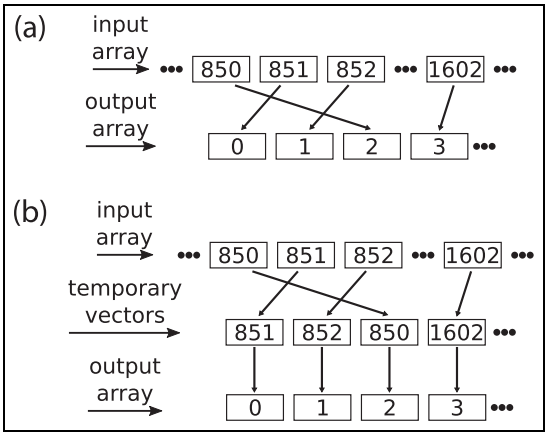

Figure 6. The concept of vectorization: (a) scalar computations based on irregular data access; (b) vectorization of computations using temporary vectors.

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 Figure 6(a). In this case, the automatic vectorization of computations fails because of irregular data access, which is unpredictable during compilation. To solve this problem, we propose to change the code slightly by adding temporary vectors responsible for loading the necessary data from the irregular memory region. It is enough to organize computations in the SIMD fashion, as is shown in Figure 6(b).

Additionally, appropriate keywords and directives should be provided as compiler hints, in order to improve auto-vectorization efficiency. Auto-vectorization is also assisted by 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 Experimental evaluation of numerical accuracy

This section provides an experimental evaluation of the proposed approach taking into account the numerical accuracy of computations performed by all the heterogeneous components. In general, some differences in the architecture of Intel coprocessors compared to Intel Xeon CPUs lead to a few small differences in implementation. The floating-point computations on the Intel Xeon Phi coprocessor may not give results that are bit-wise identical to the equivalent computations on

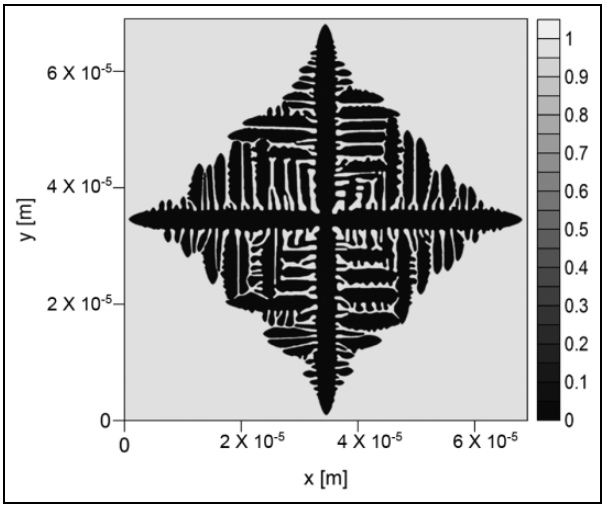

Figure 7. Phase content for the simulated time $t = 2.75\times10^{-3}$ s (original code).

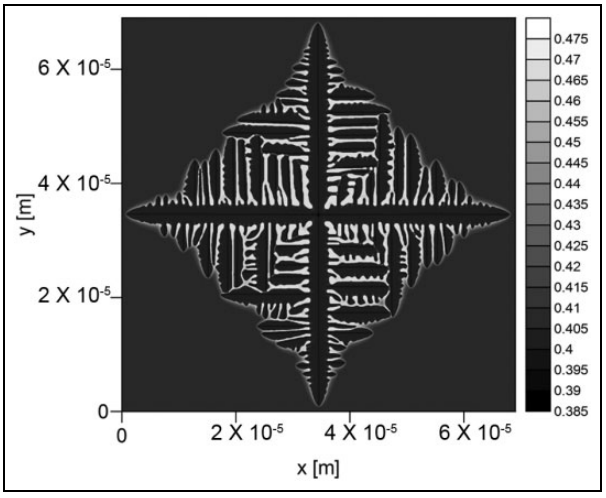

Figure 8. Concentration of alloy component for the simulated time $t = 2.75\times10^{-3}$ s (original code).

an Intel Xeon CPU (Corden, 2013), even though the underlying hardware instructions conform to the same standards. Additionally, the transformation of operations that are equivalent mathematically are not equivalent in the finite precision arithmetic (Corden and Kreitzer, 2015). These behaviors can lead to differences of outcomes in implementation, especially for the group of forward-in-time algorithms where subsequent time steps depend on previous ones.

To validate the numerical accuracy of the proposed approach, we compare experimentally the simulation results obtained from the original CPU parallel code with the new code running on a CPU–MIC hybrid platform. These experiments indicate a very high similarity of results obtained in both cases, so the differences are negligible. Figures 7 and 8 illustrate the final results of modeling alloy solidification using the original code.

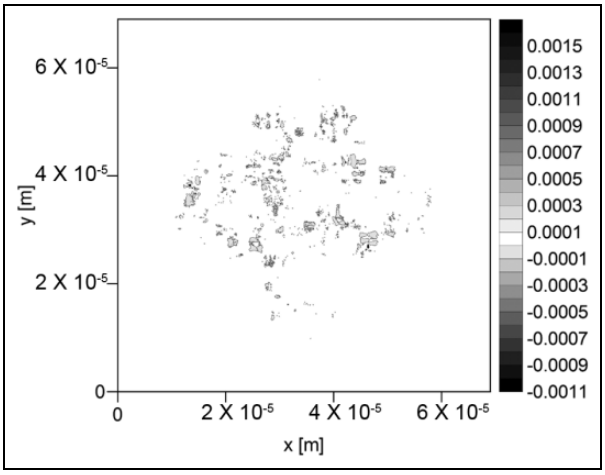

Figure 9. Difference in phase content between original and new codes for the simulated time $t = 2.75\times10^{-3}$ s.

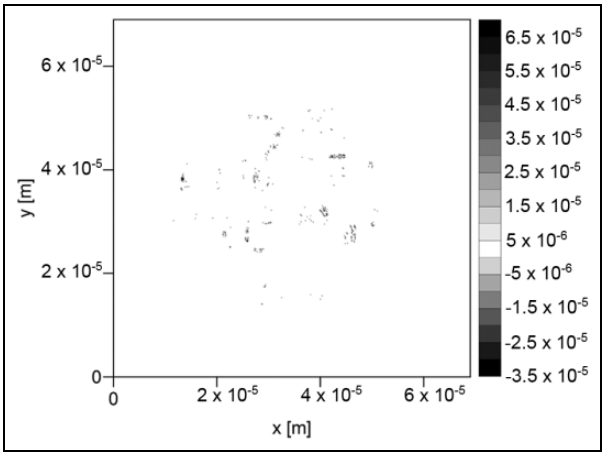

Figure 10. Difference in concentration of alloy component between the original and new codes for the simulated time $t = 2.75\times10^{-3}$ s.

For the phase content and concentration of the alloy component calculated after 110,000 time steps (simulated time $t = 2.75^{-3}$ s), the differences do not exceed 2 per mille, and refer only to the interphase areas (Figures 9 and 10). The solidified areas (corresponding to $\phi>0.05$) are identical in both cases, and these areas correspond to the same nodes. This conclusion also applies to the liquid areas ($\phi>0.95$). Therefore, it can be concluded that the simulation results for the two codes differ with each other negligibly. What should be emphasized in characterizing the correctness of the proposed solution is that these differences are not cumulative, and they do not grow during the simulation.

8 Performance results

In this section, we present performance results obtained for the new code which is developed using the approach

Table 2. Performance results for the solidification application with static intensity of computations, obtained for the original and optimized parallel versions.

Parallel version	Computing resources	First platform		Second platform	
		Time (min sec)	Speedup	Time (min sec)	Speedup
Basic	2×CPU	641′ 32″	-	666′ 06″	-
Optimized	1×MIC	235′ 34″	2.72×	235′ 34″	2.82×
Optimized	2×MIC	122′ 23″	5.24×	122′ 23″	5.44×
Optimized	2×CPU	129′ 01″	4.97×	141′ 14″	4.71×
Optimized	2×CPU + 2×MIC	68′ 42″	9.33×	74′ 01″	8.99×

Table 3. Performance results for the solidification application with dynamic intensity of computations, obtained for the original and optimized parallel versions.

Parallel version	Computing resources	First platform		Second platform	
		Time (min sec)	Speedup	Time (min sec)	Speedup
Basic	2×CPU	267′ 52″	-	324′ 16″	-
Optimized	1×MIC	97′ 03″	2.76×	97′ 03″	3.34×
Optimized	2×MIC	50′ 19″	5.32×	50′ 19″	6.44×
Optimized	2×CPU	48′ 58″	5.47×	59′ 27″	5.45×
Optimized	2×CPU + 2×MIC	32′ 09″	8.33×	38′ 57″	8.32×

described in the previous sections, for the double-precision floating-point format. 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 two hybrid platforms presented in Table 1. The original (basic) version of the solidification application uses the resources of two CPUs. All the tests are performed for 110,000 time steps, and the grid containing 4,000,000 nodes (2000 nodes along each dimensions x and y).

In our experiments, we evaluate different setups for the loop scheduling clauses in computations performed by Intel Xeon Phi coprocessors (static, dynamic, auto and guided) with different configurations for the size of chunks. The best performance corresponds to the static scheduling with chunk size equal to 128. In the case of the parallel workload processed by CPUs, we use the dynamic loop scheduling option, which allows us to reduce the overhead generated by the management of computations assigned to coprocessors. The CPU performance is evaluated exploring different sizes of chunks for the dynamic scheduling. In consequence, the best performance results are obtained for the chunk size equal to 128.

Furthermore, we perform tests of the new, optimized version of code using different configurations of computing resources. The proposed workload distribution allows a flexible usage of computing devices, including: (a) single coprocessor, (b) two coprocessors, (c) two CPUs and (d) all the devices (two CPUs and two coprocessors). In every case, the CPU is responsible for the management of coprocessors (CPU team 0), and writing results to the file (CPU team 1).

In our study, the proposed approach is evaluated using two cases of the solidification application. The first case is characterized by a static computational intensity, for a fixed problem size. In the second case, the intensity of computations changes dynamically for subsequent time steps during computations. The performance results obtained for the first case is presented in Table 2, while Table 3 corresponds to the dynamic intensity of computations.

For the first platform and static computational intensity, the execution of the basic version takes 641 min and 32 sec. The optimized version with the main workload processed on a single Intel Xeon Phi coprocessor is $2.72\times$ faster than the basic version. Its execution takes 235 min and 34 sec. The utilization of two coprocessors allows us to increase the performance $5.24\times$. The total time of the computations for this version is reduced to 122 min and 23 sec. The optimized version of code with the main workloads executed on two CPUs takes 129 min and 1 sec. This version is $4.97\times$ faster then the basic version. Finally, the utilization of all the available resources of the hybrid platform permits us to accelerate computations $9.33\times$. The total execution time for this version is 68 min and 42 sec. The presented performance results are achieved for the load balancing setup when each coprocessor calculates 24% of outcomes, while the two CPUs compute together 52% of outcomes.

For the static computational intensity, the performance results for the second platform are quite similar.

Table 4. Performance analysis for executing a package of 2000 time steps on 2×CPU + 2×MIC (first platform, static intensity).

	Aggregate time (s)
Total execution	74.90
Computations on CPUs	74.90
Computations on 1st MIC	72.27
Computations on 2nd MIC	73.07
Transfers of data	0.14
Writing data to file	11.53

Since this platform utilizes less powerful CPUs, with 12 cores each instead of the 18 cores of the first platform, the execution times for configurations using CPUs are now longer, but these differences are not high. In particular, when all the available resources are employed, the execution takes about 5 min 13 sec, which means an increase of about 7.6%.

Table 4 presents results of the performance analysis corresponding to a single package of 2000 time steps, for the static intensity of computations. Following the proposed idea of adaptation (Figure 4), data writing to the file is totally hidden behind parallel computations performed simultaneously by processors and coprocessors. All the computing devices are characterized by similar execution times. The cost of data transfers is practically negligible, and what is more, they are overlapped with computations on CPUs. As a result, the total execution time is determined by CPU computations, that take a slightly longer time than computations on any coprocessor.

Going to the case of the dynamic computational intensity, it should be noted that the first platform allows us to execute the basic version in 267 min and 52 sec. Employing a single coprocessor accelerates computations 2.76×, and all the computations take now 97 min and 3 sec. The version with two coprocessors takes now 50 min and 19 sec, which is 5.32× faster than the basic version. For two CPUs, the total execution time of the optimized version is reduced to 48 min and 58 sec, which is 5.47× faster than the basic code. Finally, employing all the available resources of the hybrid platform allows us to accelerate computations to 8.33×, which corresponds to the execution time of 32 min and 9 sec.

As in the case of the static intensity of computations, the performance results for the second platform are quite similar. But now the increase in the execution time is more significant. In particular, for the configuration with two CPUs, as well as in the case of employing all the available resources, this increase is about 21%.

It needs to be highlighted that also, in the case of the dynamic computational intensity, the domain is currently divided statically. It is evident that such an approach does not provide optimal load balancing, as the computational intensity changes during computations. For the first platform, we use a configuration when respectively 38% and 31% of outcomes are assigned to CPUs and each coprocessor, while for the second platform CPUs and each coprocessor are responsible for computing 34% and 33% of outcomes, respectively.

Analyzing results from Tables 2 and 3, we conclude that after optimization a single coprocessor is always slower than two CPUs. At the same time, in the case of static computational intensity, two coprocessors are faster than two CPUs for both platforms, while for dynamic computational intensity, two coprocessors are slower than two CPUs based on the Haswell architecture, which is used in the first platform. For the second platform, two accelerators perform calculations faster than two CPUs based on the Ivy Bridge architecture.

When the optimized code is executed by two Intel Xeon Phi coprocessors, each of them calculates half of all the outcomes. However, the time of the computations carried out by the two coprocessors is greater than 50% of the execution time achieved for a single coprocessor. The reason is the overheads caused by exchanging data between coprocessors. Since direct communication between two coprocessors is impossible, the point of data exchange is the CPU main memory, which usually generates some performance overhead.

Our final conclusion concerns the minimum value of the number R of time steps processed in every package. This value is achieved for the hybrid version with two CPUs and two MICs. Our experiments show that at least $R = 300$ and $R = 650$ time steps are enough to hide data writing to the file behind parallel computations, for static and dynamic computational intensity, respectively.

9 Conclusions and future work

Using hybrid platforms with accelerators is a promising direction for improving the efficiency of a wide range of real-life applications. Particularly, the Xeon Phi coprocessors open broad possibilities in this area, mainly due to the general-purpose programming environment provided by the Intel MIC architecture. This advantage can be successfully explored for quick and easy porting of program code to hybrid platforms with Intel Xeon Phi coprocessors, assuming no significant modifications of the code.

At the same time, achieving the highest overall performance requires taking advantage of all the available computing resources to work together. The heterogeneous programming model enables developers to meet this challenge. An effective example of this model is a combination of the offload mode for Intel Xeon Phi, and the OpenMP shared-memory programming standard. This successfully helped us to employ two Intel Xeon CPUs and two Intel Xeon Phi coprocessors for the parallelization of the solidification application. Additionally, it addresses key programming

productivity issues by allowing a separation of concerns between the expression of functional semantics and disclosure of portable parallelism, and the performance tuning and control over executing applications on hybrid architectures. In consequence, this combination is an efficient and flexible solution for porting extensive code that consists of both massively parallel and sequential regions to platforms with Intel Xeon Phi coprocessors.

The utilization of all the available resources of a hybrid platform stipulates the development of efficient and flexible methods for workload distribution across computing devices. Using coprocessors and CPUs to perform the major parallel workloads, and executing the rest of an application on CPUs, allows for successful acceleration of the whole application. Such a workload distribution permits an appropriate utilization of both coprocessors designed for massively parallel computations and CPUs that are designed for general usage.

The approach proposed in this article allows us not only to overlap data movements with computations, but it also provides an adequate utilization of cores/threads and vector units of CPUs, as well as coprocessors. Moreover, we propose the sequence of steps necessary for porting the original solidification application to hybrid platforms with accelerators that allows us to take advantage of all the available computing components, without significant modifications of the application code. The developed approach permits us to execute the whole application up to 9.33 × faster than the original parallel version. Furthermore, the CPU–MIC hybrid platforms allows achieving a speedup of about 1.9 × compared with the CPU platform with 24 cores based on the Ivy Bridge architecture, and about 1.5 × against the Haswell-based CPU platform with 36 cores. As a result, the total execution time is reduced from more than 10 to nearly 1 hour (see Table 2).

This work proves the efficiency of the proposed workload distribution for the application characterized by a static computational intensity. However, the static load balancing does not guarantee the best distribution of workload when the computational intensity is changed for successive time steps. Eliminating this disadvantage requires development of a dynamic (online) method for load balancing. This is a primary direction for our future work.

Additionally, the performance results achieved in this study provide the basis for further research on optimizing the solidification application taking into account features of memory, cache hierarchy and computing cores, as well as vector units. Another direction for future work is adaptation to heterogeneous clusters with Intel MICs, including further development and optimization of code.

Acknowledgements

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.

Declaration of Conflicting Interests

The author(s) declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

Funding

The author(s) disclosed receipt of the following financial support for the research, authorship, and/or publication of this article: This work was supported by the National Science Centre (Poland) (Grant number UMO-2015/17/D/ST6/04059), as well as partially supported by the Ministry of Education, Youth and Sports (Czech Republic) from the Large Infrastructures for Research Experimental Development and Innovations project "IT4Innovations National Supercomputing Center—LM2015070".

References

Adrian H and Spiradek-Hahn K (2009) The simulation of dendritic growth in Ni–Cu alloy using the phase field model. *Archives of Materials Science and Engineering* 40 (2): 89–93.

Benito J, Ureñ F and Gavete L (2008) The generalized finite difference method. In: Àlvarez MP (ed.) *Leading-Edge Applied Mathematical Modeling Research*. New York, NY: Nova Science Publishers, pp.251–293.

Chen L, Huo X, Ren B, et al. (2015) Efficient and simplified parallel graph processing over CPU and MIC. In: *Proceedings of the 2015 IEEE international parallel and distributed processing symposium (IPDPS)*, Hyderabad, India, 25–29 May 2015, pp.819–828. Piscataway, NJ: IEEE Xplore.

Choudhury A, Reuther K, Wesner E, et al. (2012) Comparison of phase-field and cellular automaton models for dendritic solidification in Al–Cu alloy. *Computational Materials Science* 55: 263–268.

Colfax International (2015) Colfax Servers based on Intel® Xeon Phi™ Coprocessors. Available at: http://www.colfax-intl.com/nd/xeonphi/servers.aspx (accessed 20 November 2016).

Corden M (2013) Differences in floating-point arithmetic between Intel Xeon Processors and the Intel Xeon Phi Coprocessor. Intel Corporation. Available at: https://software.intel.com/en-us/articles/differences-in-floating-point-arithmeticbetween-intel-xeon-processors-and-the-intel-xeon (accessed 28 March 2013).

Corden M and Kreitzer D (2015) Consistency of floating-point results using the Intel Compiler. Software Solutions Group, Intel Corporation. Available at: https://software.intel.com/en-us/articles/consistency-of-floating-point-resultsusing-the-intel-compiler (accessed 2 August 2012).

Folch R, Casademunt J, Hernandez-Machado A, et al. (1999) Phase-field model for Hele-Shaw flows with arbitrary

viscosity contrast. II. Numerical study. *Physical Review E* 60 (2): 1734–1740.

Hager G and Wellein G (2011) *Introduction to High Performance Computing for Science and Engineers*. Boca Raton, FL: CRC Press.

Heinecke A, Breuer A, Rettenberger S, et al. (2014) Petascale high order dynamic rupture earthquake simulations on heterogeneous supercomputers. In: *Proceedings of the 2014 ACM/IEEE international conference on high performance computing, networking, storage and analysis (SC'11)*, Seattle, WA, USA, 12–18 November 2011, pp.3–14. Washington, DC: IEEE Computer Society. IEEE Xplore

Intel Corporation (2013) *Intel Xeon Phi Coprocessor System Software Developers Guide*. Available at: https://software.intel.com/en-us/articles/intel-xeon-phi-coprocessor-system software-developers-guide (accessed 20 November 2016).

Intel Corporation (2015) *Intel Product Specifications*. Available at: http://ark.intel.com/ (accessed 20 November 2016).

IT4Innovations (2015) National Supercomputing Center IT4Innovations. Available at: http://www.it4i.cz.

Jeffers J and Reinders J (2014) *Intel Xeon Phi Coprocessor High-Performance Programming*. Waltham, MA: Elsevier.

Karma A, Kessler D and Levine H (2001) Phase-field model of mode III dynamic fracture. *Physical Review Letters* 87(4). 045501/1 -045501/4

Kulawik A (2013) *The Modeling of the Phenomena of the Heat Treatment of the Medium Carbon Steel*. Monografia, vol. 281. Czestochowa, Silesia: Wydawnictwo Politechnki Czestochowskiej.

Kurzak J, Bader D and Dongarra J (eds.) (2011) Scientific Computing with Multicore and Accelerators. Boca Raton, FL: *CRC Press*.

Liu Y and Deng L (2015) Acceleration of CFD Engineering Software on GPU and MIC. *Lecture Notes in Computer Science* 9532: 835–848.

Liu Y, Zhang X, Yang C, et al. (2014) Accelerating HPCG on TIANHE-2: A hybrid CPU–MIC algorithm. In: *Proceedings of the 2014 20th IEEE international conference on parallel and distributed systems (ICPADS)*, Hsinchu, Taiwan, 16–19 December 2014, pp.542–551. IEEE Xplore.

Liviero B (2015) Intel Xeon Phi: Application and solutions catalogue. Available at: https://software.intel.com/en-us/xeonphionlinecatalog (accessed 20 November 2016).

Longinova T, Amberg G and Ågren J (2001) Phase-field simulations of non-isothermal binary alloy solidification. *Acta Materialia* 49(4): 573–581.

MICLAB (2015) Pilot Laboratory of Massively Parallel Systems (MICLAB). Available at: http://miclab.pl (accessed 20 November 2016).

OpenMP (2015) OpenMP Application Programming Interface. Available at: http://www.openmp.org/

Colfax International (2013) *Parallel Programming and Optimization with Intel Xeon Phi Coprocessors*. Sunnyvale, CA: Colfax International.

Provatas N and Elder K (2010) *Phase-Field Methods in Materials Science and Engineering*. Hoboken, NJ: Wiley.

Rahman R (2013) *Intel Xeon Phi Coprocessor Architecture and Tools: The Guide for Application Developers*. New York, NY: APress.

Shimokawabe T, Aoki T, Takaki T, et al. (2011) 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'11)*, Seattle, WA, USA, 12–18 November 2011, 11 pages. IEEE Xplore: IEEE Computer Society.

Steinbach I (2009) Phase-field models in materials science. *Modelling and Simulation in Materials Science and Engineering* 17(7): 14 pages.

Szustak L, Halbiniak K, Kulawik A, et al. (2016) Toward parallel modeling of solidification based on the generalized finite difference method using Intel Xeon Phi. In: *Proceedings of 11th international conference on parallel processing and applied mathematics (PPAM 2015) Wyrzykowski R, Deelman E, Dongarra J, et al. (eds), Part I*. Volume 9573, pp. 411–412.

Szustak L, Rojek K and Gepner P (2014) Using Intel Xeon Phi coprocessor to accelerate computations in MPDATA algorithm. In: *Proceedings of 10th international conference on parallel processing and applied mathematics (PPAM 2013) Wyrzykowski R, Deelman E, Dongarra J, et al. (eds), Part I*. Volume 8384. pp. 582–592.

Szustak L, Rojek K, Olas T, et al. (2015) Adaptation of MPDATA heterogeneous stencil computation to Intel Xeon Phi coprocessor. *Scientific Programming*. Volume 2015, 14 pages.

Szustak L, Rojek K, Wyrzykowski R, et al. (2014) Toward efficient distribution of MPDATA stencil computation on Intel MIC architecture. In: *Proceedings of the 1st international workshop on high-performance stencil computations (HiStencils'14)*, pp.51–56. Vienna, Austria, 21 January 2014. HiStencils. Available at: http://www.exastencils.org/histencils/2014/

Takaki T (2014) Phase-field modeling and simulations of dendrite growth. *ISIJ International* 54(2): 437–444.

Vladimirov A (2015) Performance to power and performance to cost ratios with Intel Xeon Phi Coprocessors (And why 1× acceleration may be enough). 27 January 2015, 8 pages. Sunnyvale, CA: Colfax International. Available at: https://colfaxresearch.com/performance-to-power-and-performance-to-cost-ratioswith-intel-xeon-phi-coprocessors-and-why-1x-acceleration-may-be-enough/ (accessed 20 November 2016).

Warren J and Boettinger W (1995) Prediction of dendritic growth and microsegregation patterns in a binary alloy using the phase-field method. *Acta Metallurgica et Materialia* 43(2): 689–703.

Wolfe N, Liu T, Carothers C, et al. (2014) Heterogeneous concurrent execution of Monte Carlo photon transport on CPU, GPU and MIC. In: *Proceedings of the fourth workshop on irregular applications: Architectures and algorithms*, New Orleans, Louisiana, 16–21 November 2014, pp.49–52. Piscataway, NJ: IEEE Press.

Wyrzykowski R, Rojek K and Szustak L (2012) Model-driven adaptation of double-precision matrix multiplication to the cell processor architecture. *Parallel Computing* 38: 260–276.

Wyrzykowski R, Szustak L and Rojek K (2014a) Parallelization of 2D MPDATA EULAG algorithm on hybrid architectures with GPU accelerators. *Parallel Computing* 40(8): 425–447.

Wyrzykowski R, Szustak L, Rojek K, et al. (2014b) Towards efficient decomposition and parallelization of MPDATA on hybrid CPU–GPU cluster. *Lecture Notes in Computer Science* 8353: 434–444.

Xue W, Yang C, Fu H, et al. (2015) Ultra-scalable CPU–MIC acceleration of mesoscale atmospheric modeling on TIANHE-2. *IEEE Transactions on Computers* 64(8): 2382–2393.

Zaeem M, Yin H and Felicelli S (2013) Modeling dendritic solidification of Al–3%Cu using cellular automaton and phase-field methods. *Applied Mathematical Modelling* 37(5): 3495–3503.

Author Biographies

Lukasz Szustak received his MSc in Computer Science from the Czestochowa University of Technology in 2008 and his PhD in 2012. During this period, his doctoral research focused on adaptation of high performance computing to modern parallel architectures including hybrid platforms. Since 2012, Dr. Szustak is employed at Czestochowa University of Technology. His current work is associated with the development of efficient methods of scheduling, load balancing, and adaptations of stencil based computations to Intel MIC and CPUs architectures.

Kamil Halbiniak received his MSc degree in Computer Science in 2015 from the Czestochowa University of Technology, Poland. In the same year, he started his PhD studies in parallel computing, with Professor Roman Wyrzykowski as a scientific advisor. His research focus is on the adaptation of scientific applications to HPC computing platforms based on multiand manycore parallel architectures, including CPUs and Intel MIC.

Lukasz Kuczynski received his MSc degree in Computer Science in 2001 from the Czestochowa University of Technology, Poland. In 2010 he received his PhD degree in Computer Science for his dissertation on data management in grid systems. His research focuses on code optimization for multicore architectures, including Intel and ARM.

Joanna Wrobel received her MSc degree in Computer Science from the Czestochowa University of Technology, Poland, in 2011. Currently she is a PhD student at the Faculty of Mechanical Engineering and Computer Science. Her scientific activity is associated with the numerical modeling of thermo-mechanical phenomena and the use of artificial intelligence in this area.

Adam Kulawik received his MSc in Computer Science from the Czestochowa University of Technology, Poland, in 2000 and his PhD in Technical Science in 2005. His research focus on the numerical analysis of thermal phenomena, the phase transformations in the solid state and stresses occurring during the process of the heat treatment of steel. He develops software for engineering computations using the finite element method and the generalized finite difference method.