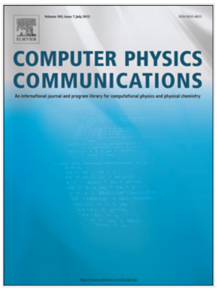


Large-scale parallel lattice Boltzmann—Cellular automaton model of two-dimensional dendritic growth

Bohumir Jelinek, Mohsen Eshraghi, Sergio Felicelli, John F. Peters

PII: S0010-4655(13)00314-7
DOI: http://dx.doi.org/10.1016/j.cpc.2013.09.013
Reference: COMPHY 5138

To appear in: *Computer Physics Communications*

Received date: 10 January 2013
Revised date: 14 August 2013
Accepted date: 17 September 2013

Large-scale parallel lattice Boltzmann - cellular automaton model of two-dimensional dendritic growth

Bohumir Jelinek[a], Mohsen Eshraghi[a,b], Sergio Felicelli[a,b], John F. Peters[c]

[a]Center for Advanced Vehicular Systems, Mississippi State University, MS 39762, USA
[b]Mechanical Engineering Dept., Mississippi State University, MS 39762, USA
[c]U.S. Army ERDC, Vicksburg, MS 39180, USA

Abstract

An extremely scalable lattice Boltzmann (LB) - cellular automaton (CA) model for simulations of two-dimensional (2D) dendritic solidification under forced convection is presented. The model incorporates effects of phase change, solute diffusion, melt convection, and heat transport. The LB model represents the diffusion, convection, and heat transfer phenomena. The dendrite growth is driven by a difference between actual and equilibrium liquid composition at the solid-liquid interface. The CA technique is deployed to track the new interface cells. The computer program was parallelized using Message Passing Interface (MPI) technique. Parallel scaling of the algorithm was studied and major scalability bottlenecks were identified. Efficiency loss attributable to the high memory bandwidth requirement of the algorithm was observed when using multiple cores per processor. Parallel writing of the output variables of interest was implemented in the binary Hierarchical Data Format 5 (HDF5) to improve the output performance, and to simplify visualization. Calculations were carried out in single precision arithmetic without significant loss in accuracy, resulting in 50% reduction of memory and computational time requirements. The presented solidification model shows a very good scalability up to centimeter size domains, including more than ten million of dendrites.

Keywords:
solidification, dendrite growth, lattice Boltzmann, cellular automaton, parallel computing

PROGRAM SUMMARY

Manuscript Title: Large-scale parallel lattice Boltzmann - cellular automaton model of two-dimensional dendritic growth
Authors: Bohumir Jelinek, Mohsen Eshraghi, Hebi Yin, Sergio Felicelli
Program Title: 2Ddend
Journal Reference:
Catalogue identifier:
Licensing provisions: CPC non-profit use license
Programming language: Fortran 90
Computer: Linux PC and clusters
Operating system: Linux
RAM: memory requirements depend on the grid size
Number of processors used: 1–50,000
Keywords: MPI, HDF5, lattice Boltzmann
Classification: 6.5 Software including Parallel Algorithms, 7.7 Other Condensed Matter inc. Simulation of Liquids and Solids
External routines/libraries: MPI, HDF5
Nature of problem: Dendritic growth in undercooled Al-3wt%Cu alloy melt under forced convection.
Solution method: The lattice Boltzmann model solves the diffusion, convection, and heat transfer phenomena. The cellular automaton technique is deployed to track the solid/liquid interface.
Restrictions: Heat transfer is calculated uncoupled from the fluid flow. Thermal diffusivity is constant.
Unusual features: Novel technique, utilizing periodic duplication of a pre-grown “incubation” domain, is applied for the scale up test.
Running time: Running time varies from minutes to days depending on the domain size and a number of computational cores.

1. Introduction

The microstructure and accompanying mechanical properties of the engineering components based on metallic alloys are established primarily in the process of solidification. In this process, the crystalline matrix of the solid material is formed according to morphology and composition of the crystalline dendrites growing from their nucleation sites in an undercooled melt. Among other processes, solidification occurs during casting and welding, which are commonly deployed in modern manufacturing. Therefore, understanding of the phenomena of solidification is vital to improve the strength and durability of the products that encounter casting or welding in the manufacturing process.

The phenomenon of dendrite growth during solidification has been the subject of numerous studies. When a new simulation technique is developed, the first and necessary step is to validate it by examining the growth of a single dendrite, or a small set of dendrites in a microscale specimen. If the model compares well with physical experiments, it can be deployed for simulating larger domains. Since the new techniques are initially implemented in serial computer codes, the size of the simulation domain is often limited by the memory and computational speed of a single computer. Despite the current advances in the large scale parallel supercomputing, only a handful of studies of larger, close-to-physical size solidification domains have been performed. Parallel simulations of the 3D dendrite growth [1] have been performed utilizing the phase field model [2]. Improved, multigrid phase field schemes presented by Guo et al. [3] allow parallel simulations of tens of complex shape 2D dendrites in a simulation domain of up to 25 μm × 25 μm size. Shimokawabe et al. [4] deployed a modern heterogeneous GPU/CPU architecture to perform the first peta-scale 3D solidification simulations in a domain size of up to 3.1 mm × 4.8 mm × 7.7 mm. However, none of these models included convection.

For the solidification models incorporating effects of convection, lattice Boltzmann method (LBM) is an attractive alternative to the conventional, finite difference and finite element based fluid dynamics solvers. Among LBM advantages are simple formulation and locality, with locality facilitating parallel implementation.

In this work, the cellular automaton (CA) technique instead of a phase field model is deployed to track the solid-liquid interface, as suggested by Sun et al. [5]. Interface kinetic is driven by the local difference between actual and equilibrium liquid composition [6]. A serial version of the present LBM-CA model with a smaller number of dendrites was validated against theoretical and experimental results [7–11]. In the following, we demonstrate nearly ideal parallel scaling of the model up to millions of dendrites in centimeter size domains, including thermal, convection, and solute redistribution effects.

2. Continuum formulation for fluid flow, solute transport, and heat transfer

Melt flow is assumed to be incompressible, without external force and pressure gradient, governed by simplified Navier-Stokes equations (NSE)

$$\rho\left[\frac{\partial \boldsymbol{u}}{\partial t} + \boldsymbol{u}\cdot\nabla\boldsymbol{u}\right] = \nabla\cdot(\mu\nabla\boldsymbol{u}), \tag{1}$$

where the $\mu = \nu\rho$ is the dynamic viscosity of melt.

Time evolution of the solute concentration in the presence of fluid flow is given by the convection-diffusion equation

$$\frac{\partial C_l}{\partial t} + \boldsymbol{u}\cdot\nabla C_l = \nabla\cdot(D_l\nabla C_l), \tag{2}$$

where C_l is the solute concentration in the liquid phase and D_l is the diffusion coefficient of solute in the liquid phase. Solute diffusion in the solid phase is neglected.

Heat transfer in the presence of fluid flow is also governed by a convection-diffusion type (2) equation

$$\frac{\partial T}{\partial t} + \boldsymbol{u}\cdot\nabla T = \nabla\cdot(\alpha\nabla T), \tag{3}$$

where T is the temperature, t is time, and α is the thermal diffusivity.

3. Lattice Boltzmann method

The lattice Boltzmann method (LBM) [12–15] is a simulation technique for solving fluid flow and transport equations.

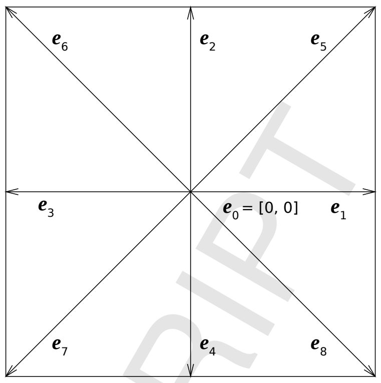


Figure 1: D2Q9 lattice vectors.

LBM treats the fluid as a set of fictitious particles located on a d-dimensional lattice. Primary variables of LBM are particle distribution functions f_i. Particle distribution functions represent portions of a local particle density moving in the directions of discrete velocities. For a lattice representation DdQz, each point in the d-dimensional lattice link to neighboring points with z links that correspond to velocity directions. We chose D2Q9 lattice, utilizing nine velocity vectors $\boldsymbol{e}_0$–$\boldsymbol{e}_8$ in two dimensions, as shown in Fig. 1. Distribution functions f_0–f_8 correspond to velocity vectors $\boldsymbol{e}_0$–$\boldsymbol{e}_8$. Using the collision model of Bhatnagar-Gross-Krook (BGK) [16] with a single relaxation time, the evolution of distribution functions is given by

$$f_i(\boldsymbol{r} + \boldsymbol{e}_i\Delta t, t + \Delta t) = f_i(\boldsymbol{r}, t) + \frac{1}{\tau_u}\left(f_i^{eq}(\boldsymbol{r}, t) - f_i(\boldsymbol{r}, t)\right) \tag{4}$$

where $\boldsymbol{r}$ and t are space and time position of the lattice site, Δt is the time step, and τ_u is the relaxation parameter for the fluid flow. Relaxation parameter τ_u specifies how fast each particle distribution function f_i approaches its equilibrium f_i^{eq}. Kinematic viscosity ν is related to the relaxation parameter τ_u, lattice spacing Δx, and simulation time step Δt by

$$\nu = \frac{\tau_u - 0.5}{3}\frac{\Delta x^2}{\Delta t}. \tag{5}$$

The macroscopic fluid density ρ and velocity $\boldsymbol{u}$ are obtained as the moments of the distribution function

$$\rho = \sum_{i=0}^{8} f_i, \quad \rho\boldsymbol{u} = \sum_{i=0}^{8} f_i\boldsymbol{e}_i. \tag{6}$$

Depending on the dimensionality d of the modeling space and a chosen set of the discrete velocities $\boldsymbol{e}_i$, the corresponding equilibrium particle distribution function can be found [17]. For the D2Q9 lattice, the equilibrium distribution function f_i^{eq}, including the effects of convection $\boldsymbol{u}(\boldsymbol{r})$, is

$$f_i^{eq}(\boldsymbol{r}) = w_i\rho(\boldsymbol{r})\left(1 + 3\frac{\boldsymbol{e}_i\cdot\boldsymbol{u}(\boldsymbol{r})}{c^2} + \frac{9}{2}\frac{(\boldsymbol{e}_i\cdot\boldsymbol{u}(\boldsymbol{r}))^2}{c^4} - \frac{3}{2}\frac{\boldsymbol{u}(\boldsymbol{r})\cdot\boldsymbol{u}(\boldsymbol{r})}{c^2}\right), \tag{7}$$

with the lattice velocity $c = \Delta x/\Delta t$ and the weights

$$w_i = \begin{cases} 4/9 & i = 0 \\ 1/9 & i = 1,2,3,4 \\ 1/36 & i = 5,6,7,8. \end{cases} \tag{8}$$

Weakly compressible approximation of NSE (1) can be recovered from the LBM equations (4–8) by Chapman-Enskog expansion [18]. Approximation is valid in the limit of low Mach number M, with a compressibility error in the order of $\sim M^2$ [13].

Deng et al. [19] formulated a BGK collision based LBM model for the convection-diffusion equation (2). In analogy with equation (5), the diffusivity D_l of solute in the liquid phase is related to the relaxation parameter τ_C for the solute transport in the liquid phase, lattice spacing Δx, and simulation time step Δt by

$$D = \frac{\tau_\mathrm{C} - 0.5}{3}\frac{\Delta x^2}{\Delta t}. \tag{9}$$

Similarly, thermal diffusivity α is related to the relaxation parameter for the heat transfer τ_T as follows

$$\alpha = \frac{\tau_\mathrm{T} - 0.5}{3}\frac{\Delta x^2}{\Delta t}. \tag{10}$$

Corresponding LBM equation for the solute transport is

$$g_i(\boldsymbol{r} + \boldsymbol{e}_i\Delta t, t + \Delta t) = g_i(\boldsymbol{r}, t) + \frac{1}{\tau_\mathrm{C}}\left(g_i^\mathrm{eq}(\boldsymbol{r}, t) - g_i(\boldsymbol{r}, t)\right), \tag{11}$$

and for the heat transfer

$$h_i(\boldsymbol{r} + \boldsymbol{e}_i\Delta t, t + \Delta t) = h_i(\boldsymbol{r}, t) + \frac{1}{\tau_\mathrm{T}}\left(h_i^\mathrm{eq}(\boldsymbol{r}, t) - h_i(\boldsymbol{r}, t)\right) \tag{12}$$

Macroscopic properties can be obtained from

$$C_\mathrm{l} = \sum_{i=0}^{8} g_i, \quad T = \sum_{i=0}^{8} h_i. \tag{13}$$

while the equilibrium distribution functions are

$$g_i^\mathrm{eq}(\boldsymbol{r}) = w_i C_\mathrm{l}(\boldsymbol{r})\left(1 + 3\frac{\boldsymbol{e}_i\cdot\boldsymbol{u}(\boldsymbol{r})}{c^2} + \frac{9}{2}\frac{(\boldsymbol{e}_i\cdot\boldsymbol{u}(\boldsymbol{r}))^2}{c^4} - \frac{3}{2}\frac{\boldsymbol{u}(\boldsymbol{r})\cdot\boldsymbol{u}(\boldsymbol{r})}{c^2}\right) \tag{14}$$

$$h_i^\mathrm{eq}(\boldsymbol{r}) = w_i T(\boldsymbol{r}). \tag{15}$$

In equation (15), the effect of fluid flow on the temperature is not considered.

4. Solidification

In the present model, the dendrite growth is controlled by the difference between local equilibrium solute concentration and local actual solute concentration in the liquid phase. If the actual solute concentration C_l in an interface cell is lower than the local equilibrium concentration C_l^eq, then the fraction of solid in the interface cell is increased by

$$\Delta f_\mathrm{s} = (C_\mathrm{l}^\mathrm{eq} - C_\mathrm{l})/(C_\mathrm{l}^\mathrm{eq}(1 - k)), \tag{16}$$

to satisfy the equilibrium condition at the interface [6]. The partition coefficient k of the solute is obtained from the phase diagram, and the actual local concentration of the solute in the liquid phase C_l is computed by LBM. The interface equilibrium concentration C_l^eq is calculated as [6]

$$C_\mathrm{l}^\mathrm{eq} = C_0 + \frac{T^* - T_{\mathrm{C}_0}^\mathrm{eq}}{m_l} + \Gamma K\frac{1 - \delta\cos(4(\phi - \theta_0))}{m_l} \tag{17}$$

where T^* is the local interface temperature computed by LBM, $T_{\mathrm{C}_0}^\mathrm{eq}$ is the equilibrium liquidus temperature at the initial solute concentration C_0, m_l is the slope of the liquidus line in the phase diagram, Γ is the Gibbs-Thomson coefficient, δ is the anisotropy coefficient, ϕ is the growth angle, and θ_0 is the preferred growth angle, both measured from the x-axis. K is the interface curvature and can be calculated as [20]

$$K = \left[\left(\frac{\partial f_\mathrm{s}}{\partial x}\right)^2\left(\frac{\partial f_\mathrm{s}}{\partial y}\right)^2\right]^{-3/2} \times\left[2\frac{\partial f_\mathrm{s}}{\partial x}\frac{\partial f_\mathrm{s}}{\partial y}\frac{\partial^2 f_\mathrm{s}}{\partial x\partial y} - \left(\frac{\partial f_\mathrm{s}}{\partial x}\right)^2\frac{\partial^2 f_\mathrm{s}}{\partial y} - \left(\frac{\partial f_\mathrm{s}}{\partial y}\right)^2\frac{\partial^2 f_\mathrm{s}}{\partial x}\right]. \tag{18}$$

The cellular automaton (CA) algorithm is used to identify new interface cells. The CA mesh is identical to the LB mesh. Three types of the cells are considered in the CA model: solid, liquid, and interface. Every cell is characterized by the temperature, solute concentration, crystallographic orientation, and fraction of solid. The state of each cell at each time step is determined from the state of itself and its neighbors at previous time step. The interface cells are identified according to Moore neighborhood rule—when a cell is completely solidified, the nearest surrounding liquid cells located in both axial and diagonal directions are marked as interface cells. The fraction of solid in the interface cells is increased by Δf_s at every time step until the cell solidifies. Then the cell is marked as solid.

During solidification, the solute partition occurs between solid and liquid and the solute is rejected to the liquid at the interface. The amount of solute rejected to the interface is determined using the following equation

$$\Delta C_\mathrm{l} = C_\mathrm{l}(1 - k)\Delta f_\mathrm{s} \tag{19}$$

Rejected solute is distributed evenly into the surrounding interface and liquid cells.

5. Test case

As a demonstration, Fig. 2 shows flow of the Al-3wt%Cu melt between solidifying dendrites in a changing temperature field. The simulation domain of 144 μm × 144 μm size is meshed on a regular 480 × 480 lattice with 0.3 μm spacing. At the beginning of the simulation, one hundred random dendrite nucleation sites with arbitrary crystallographic orientations are placed in a domain of undercooled molten alloy. Initially constant temperature of the melt was T = 921.27 K, equivalent to 4.53 K undercooling. Solidifying region is cooled at the front

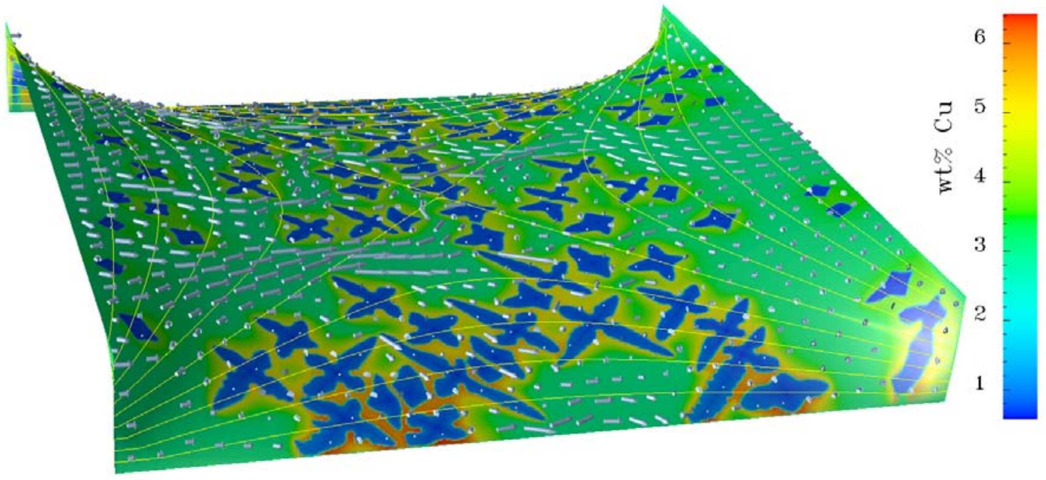


Figure 2: Flow of Al-3wt%Cu alloy melt between solidifying dendrites in variable temperature field. Solidifying region is cooled at the front and back boundaries, while heat is supplied from the left (inlet) and right (outlet) boundaries. Arrows represent velocity vectors of melt. Colors represent solute concentration. Both contour lines and height of the sheet represent temperature.

Table 1: Physical properties of Al-3wt%Cu alloy considered in the simulations.

Density ρ (kg m−3)	Diffusion coeff. D (m^2 s^{-1})	Dynamic viscosity μ (N s m^{-2})	Liquidus slope m_l (K wt%$^{-1}$)	Partition coeff. k	Gibbs-Thomson coeff. Γ (mK)	Degree of anisotropy δ
2475	3×10^{-9}	0.0024	-2.6	0.17	0.24×10^{-6}	0.6

and back boundaries, while heat is supplied from the left (inlet) and right (outlet) boundaries at the rate of 100 K/m. Fig. 2 demonstrates that the dendrites grow faster in the regions with lower temperature and slower in the regions with higher temperature, as expected. The drag and buoyancy effects are ignored—the dendrites are stationary and do not move with the flow. The nonslip boundary condition at the solid/liquid interface is applied using the bounce-back rule for both fluid flow and solute diffusion calculations. Simplicity of the bounce back boundary conditions is an attractive feature of the LBM. At the bounce back boundary, the distribution functions incoming to the solid are simply reflected back to the fluid. This method allows efficient modeling of the interaction of fluid with complex-shaped dendrite boundaries. Total simulated time was 2.32 ms, involving 150,000 collision-update steps. The physical properties for the Al-Cu alloy considered in the simulations are listed in Table 1. The relaxation parameter $\tau_u = 1$ was chosen for the fluid flow. With a lattice spacing of $\Delta x = 0.3\ \mu$m, this leads to the time step of $\Delta t = 15.47$ ns. For the solute transport and temperature, the relaxation parameters were set according to their respective diffusivities to follow the same time step.

6. Serial optimization

Reducing accuracy of data representation together with corresponding reduction of computational cost present a simple option of saving computational time and storage resources [21]. Along with the default, double precision variables, we implemented an optional single precision data representation. This resulted in 50% reduction in memory and processing time requirements. Undesirable consequence of reducing accuracy was that the results in single precision representation differed significantly from double precision results. We found that the number of valid digits in single precision was not large enough to represent small changes in the temperature. To achieve better accuracy, we changed the temperature T representation to a sum of T_0 and $\Delta T = T - T_0$, where T_0 is the initial temperature and ΔT is local undercooling. With this modification, a visual difference between results in single and double precision calculations was negligible.

Ordering of the loops over two dimensions of arrays has a profound impact on the computational time. In Fortran, matrices are stored in memory in a column-wise order, so the first index of an array is changing fastest. To optimize the cache use, the data locality needs to be exploited, thus the inner-most computational loop must be over the fastest changing array index. This can lead to reduction in computational time in the order-of-magnitudes. We chose to implement "propagation optimized" [22] storage of the distribution functions, where the first two array indices represent x and y lattice coordinates and the last index of the array represents the nine components of the distribution function. Further serial optimization, not considered in this work, could be achieved by combining collision and streaming steps, loop blocking [22, 23], or by elaborate improvements of propagation step [24].

The performance analysis of the code was done using HPC-T [25]. It revealed that some of the comparably computationally intensive loops introduced more computational cost than others because not all the constants utilized in them were reduced to single precision. Consolidation of the constants into a single precision improved serial performance. Also, it was found that array section assignments in the form `a[1:len-1]=a[2:len]` required temporary storage with an additional performance penalty. Significant reduction in computational time was achieved by replacing these assignments with explicit `do` loops.

7. Parallelization

A spatial domain decomposition was applied for parallelization. In this well-known, straightforward and efficient approach, the entire simulation domain is split into equally sized subdomains, with the number of subdomains equal to the number of execution cores. Each execution core allocates the data and performs computation in its own subdomain. Given the convenient locality of the LBM-CA model, only the values on the subdomain boundaries need to be exchanged between subdomains. `MPI_Sendrecv` calls were utilized almost exclusively for MPI communication. `MPI_Sendrecv` calls are straightforward to apply for the required streaming operation. They are optimized by MPI implementation and are more efficient than individual `MPI_Send` and `MPI_Receive` pairs, as they generate non-overlapping synchronous communication. Error-prone non-blocking communication routines are not expected to provide significant advantage, as most of the computation in the present algorithm can start only after communication is completed. As it was uncertain whether the gain from OpenMP parallelization within shared-memory nodes would bring a significant advantage over MPI parallelization, the hybrid MPI/OpenMP approach was not implemented.

7.1. Collision and streaming

LBM equations (4, 11, 12) can be split into two steps: collision and streaming. Collision, the operation on the right-hand-side, calculates new value of the distribution function. The collision step is completely local—it does not require values from the surrounding cells. Each execution core has the data it needs available, and no data exchange with neighboring subdomains is required.

The second step, assignment operation in equations (4, 11, 12), is referred to as streaming. Streaming step involves propagation of each distribution function to the neighboring cell. Except for the stationary f_0, each distribution function f_i is propagated in the direction of the corresponding lattice velocity $\boldsymbol{e}_i$ ($i = 1 \ldots 8$). For the neighboring cells belonging to the computational subdomain of another execution core, the distribution functions are transferred to the neighboring subdomains using MPI communication routines. During the streaming step, permanent storage is allocated only for values from the local subdomain. When the streaming step is due, temporary buffers are allocated to store the data to be sent to (or received from) other execution cores. As an example, the streaming operations involving the distribution function f_8 on the execution core 5 are shown in Fig. A.6 in the Appendix.

7.2. Ghost sites

When calculation in a particular lattice cell needs values from the neighboring cells, the neighboring cells may belong to the computational subdomains of other execution cores. Therefore, the values needed may not be readily available to the current execution. To provide access to the data from other executions, an extra layer of lattice sites is introduced at the boundary with each neighboring subdomain. Values from these extra boundary layers, referred to as ghost layers, are populated from the neighboring subdomains. Population of the ghost sites is a common operation in parallel stencil codes. Fig. A.7 shows MPI communication involved when the ghost sites are populated for the execution core 5.

In the process of solidification, solute is redistributed from the solidifying cells to the neighboring cells. In this case, the ghost layers are used to store the amount of solute to be distributed to the neighboring subdomains. Only difference between this case and the simple population of ghost layers shown in Fig. A.7 is that the direction of data propagation is opposite.

7.3. Parallel input/output

As the size of the simulation domain increased, the storage, processing, and visualization of results required more resources. We implemented parallel writing of simulation variables in a binary HDF5 [26] format. Publicly available HDF5 library eliminated the need to implement low level MPI i/o routines. Data stored in the standard HDF5 format are straightforward to visualize using common visualization tools.

In the binary format, the data is stored without a loss in accuracy. Taking advantage of that, we also implemented capability of writing and reading restart files in HDF5 format. The restart files allow exact restart of the simulation from any time step, and also simplify debugging and data inspection.

8. Parallel performance

8.1. Computing systems

Kraken, located at Oak Ridge National Laboratory, is a Cray XT5 system managed by National Institute for Computational Sciences (NICS) at the University of Tennessee. It has 9,408 computer nodes, each with 16 GB of memory and two six-core AMD Opteron "Istanbul" processors (2.6 GHz), connected by Cray SeaStar2+ router. We found that the Cray compiler generated the fastest code on Kraken. Compilation was done under Cray programming environment version 3.1.72 with a Fortran compiler version 5.11, utilizing parallel HDF5 [26] library version 1.8.6, and XT-MPICH2 version 5.3.5. The compiler optimization flags "-O vector3 -O scalar3" were applied, with a MPICH_FAST_MEMCPY option set to enable optimized `memcpy` routine. Vectorization reports are provided with the code.

Talon, located at Mississippi State University, is a 3072 core cluster with 256 IBM iDataPlex nodes. Each node has two six-core Intel X5660 "Westmere" processors (2.8 GHz), 24 GB of RAM, and Voltaire quad data-rate InfiniBand (40 Gbit/s) interconnect. On Talon, Intel Fortran compiler version 11.1 was

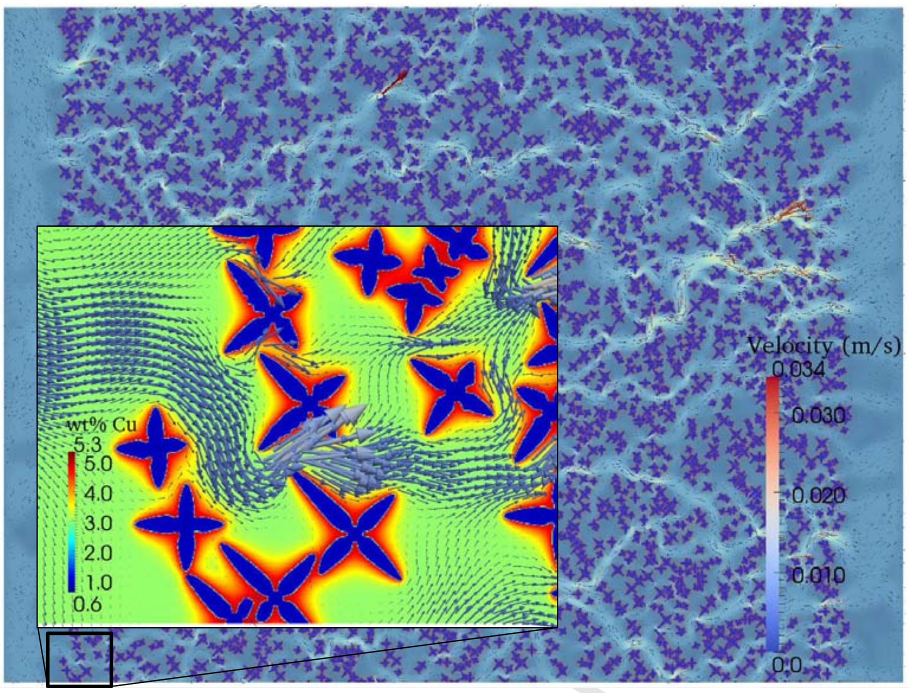


Figure 3: Final snapshot of the dendrite incubation domain, with a magnified portion. Image shows a flow of alloy melt between solidifying dendrites. Arrows represent velocity vectors of melt. Colors of background and dendrites represent solute concentration, while color and size of arrows represent magnitude of velocity.

utilized with "-O3 -shared-intel -mcmodel=large -heap-arrays" options.

8.2. *Scaling test configuration*

Scalability of the code needs to be measured for the typical computational task. As one dendrite growth step is performed every 587 basic time steps of simulation, the minimal, 587 time step run was deployed as a performance test. At the beginning of simulation, dendrites are small and their growth represents only a small portion of the computational load. To measure the performance with characteristic computational load, we first grow the dendrites into a reasonable size in a so called "incubation region" (Fig. 3), and then store intermediate results needed for exact restart of simulation in the HDF5 files. The performance is then evaluated in the 587 time step execution starting from this configuration.

The incubation domain presents a flow of the Al-3wt%Cu melt between solidifying dendrites in spatially constant temperature field with periodic boundary conditions. The computational domain of 2.4 mm × 1.8 mm size is meshed on a regular 8000 × 6000 lattice with 3264 dendrite nucleation sites. Total incubation time was 6.19 ms, with 400,000 collision-update steps. Initial, spatially constant temperature of the melt was $T = 921.27$ K, equivalent to 4.53 K undercooling. The temperature is decreasing at the constant cooling rate of 100 K/s.

8.3. *Strong scaling*

To characterize the gain from parallelization, one can compare the calculation time of the task of size (2D area) A on one execution core with the calculation time on multiple cores, referred to as the strong scaling. Ideally, the task taking $T(1)$ seconds on one core should take $T(1)/p$ seconds on p cores—that would mean speed up of p, or 100% parallel efficiency. Intuitively, the speed up is defined as

$$S(p, A) = \frac{T(1, A)}{T(p, A)} \tag{20}$$

For ideal parallel performance, $S(p, A) = p$. An efficiency,

$$\eta(p, A) = \frac{S(p, A)}{p} 100\% = \frac{T(1, A)}{p\,T(p, A)} 100\%, \tag{21}$$

is the ratio between the actual speed up $S(p)$ and the ideal speed up p. Efficiency value is 100% for the ideal parallel performance.

Ideal performance is expected e.g. when the tasks solved by individual cores are independent. When the tasks to be solved by individual cores depend on each other, the efficiency usually decreases with the number of cores—when communication costs become comparable with computational costs, the efficiency goes down.

Fig. 4 shows the parallel speed up obtained for the domain of 8000 × 6000 lattice cells. Due to the high memory bandwidth requirement of the algorithm, an increase in the utilized number

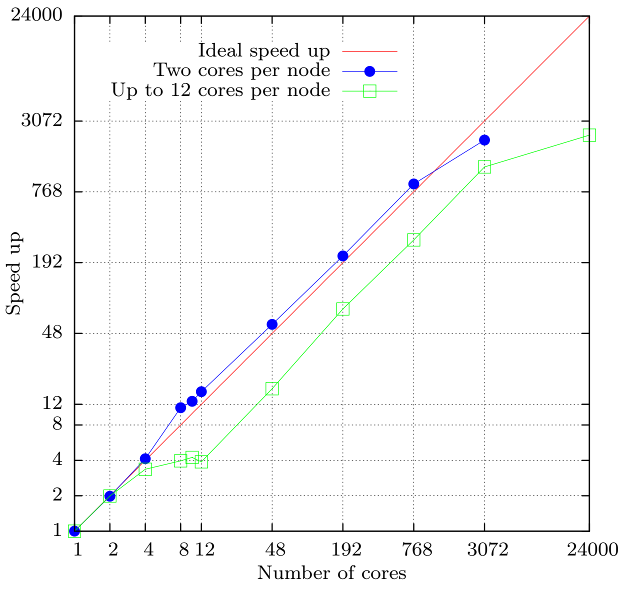


Figure 4: Strong scaling (speed up) on Kraken. A grid of 8000 × 6000 lattice points is split equally between increasing number of cores. This domain was simulated for 587 time steps, taking 9035 seconds on a single core. The number of cores is equal to the number of MPI processes.

of cores in one node causes severe parallel performance loss - efficiency decreases to ∼30%. On the contrary, when two cores per node are used, the parallel efficiency remains close to 100% up to 3072 cores.

8.4. *Weak scaling*

Increased number of the execution cores and associated memory allows to solve problems in larger domains. If the number of cores is multiplied by p, and the simulation domain also increases by the factor of p, the simulation time should not change. This, so called weak scaling of the algorithm, is characterized by the scale up efficiency, defined as

$$\eta'(p, pA) = \frac{T(p, pA)}{T(1, A)} \tag{22}$$

To test weak scaling (scale up) of the present algorithm, the incubation domain (Fig. 3) was read in from the restart files utilizing one supercomputer node, and then duplicated proportionally to increasing number of nodes. Starting from the restart point, 587 time step execution was performed. Fig. 5 shows fairly constant calculation time, demonstrating nearly perfect scalability of the LBM/CA model.

The largest simulation was performed on 41,472 cores of Kraken at ORNL. Kraken has a total of 112,000 cores. The initial configuration was obtained from the base “incubation” domain (Fig. 3) replicated 72 times in the x-direction and 48 times in the y-direction, representing a total domain size of 17.28 cm × 8.64 cm. The corresponding grid had (8000×72) × (6000×48) cells, i.e. over 165 billion nodes. Uniform flow

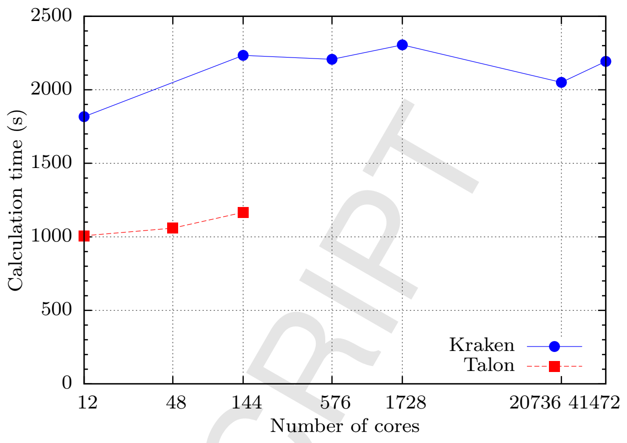


Figure 5: Weak scaling (scale up). A grid of 8000 × 6000 lattice points is the constant-size calculation domain per node. The global simulation domain increases proportionally to the number of cores. The total number of cores is equal to the total number of MPI processes.

of melt with velocity u_0 = 2.3 mm/s, was forced into the left (inlet) and out of the right (outlet) boundary. Solidifying region was cooled at the top and bottom boundaries, while heat was supplied from the left and right boundaries. The simulated domain contained 11.28 million dendrites. Duration of the largest simulation was 2,250 seconds, including 57 seconds spent on reading the incubation domain and replicating it to all execution cores. Importantly, it was found that consideration of convection effects on solidification in the LBM-CA model does not significantly affect the calculation time as observed in other models [8].

The exclusive calculation time, involving 587 LBM steps and one dendrite growth step, was 2,193 seconds. 133.22 giga lattice site updates per second were performed. The full 400,000 steps calculation of the 17.28 cm × 8.64 cm domain size would take about 17 days using 41,472 cores of Kraken, or about 6 days using all 112,000 cores.

9. Conclusions

The presented model of dendritic growth during alloy solidification, incorporating effects of melt convection, solute diffusion, and heat transfer, shows a very good parallel performance and scalability. It allows simulations of unprecedented, centimeter size domains, including ten millions of dendrites. The presented large scale solidification simulations were feasible due to 1) CA technique being local, highly parallelizable, and two orders of magnitude faster than alternative, phase-field methods [27], 2) local and highly parallelizable LBM method, convenient for simulations of flow within complex boundaries changing with time, and 3) availability of the extensive computational resources. The domain size and number of dendrites presented in the solidification simulation of this work are the largest known to the authors to date, particularly including convection effects. Although such large 2D domains may not be necessary to capture a representative portion of a continuum structure, we expect that the outstanding scalability and parallel performance shown by the model will allow simulations of 3D microstructures with several thousands of dendrites, effectively enabling continuum-size simulations. A similar scalability study with a 3D version of the model [11] is currently under way and will be published shortly.

Acknowledgment

This work was funded by the US Army Corps of Engineers through contract number W912HZ-09-C-0024 and by the National Science Foundation under Grant No. CBET-0931801. Computational resources at the MSU HPC^2 center (Talon) and XSEDE (Kraken at NICS, Gordon at SDSC, Lonestar at TACC) were used. Computational packages HPCT [25], P - E [28], and C PAT [29] were used to assess the code performance and scalability bottlenecks. Images were made using OpenDX with 5 module [30] and Paraview tool `http://www.paraview.org/`. This study was performed with the XSEDE extended collaborative support guide of Reuben Budiardja at NICS. An excellent guide and consultation on HPCT was provided by John Mellor-Crummey, Department of Computer Science, Rice University. The authors also acknowledge the Texas Advanced Computing Center (TACC) at The University of Texas at Austin for providing HPC resources, training, and consultation (James Brown, Ashay Rane) that have contributed to the research results reported within this paper. URL: `http://www.tacc.utexas.edu`

Access to Kraken supercomputer was provided through NSF XSEDE allocations TG-ECS120004 and TG-ECS120006.

References

[1] W. L. George, J. A. Warren, A Parallel 3D Dendritic Growth Simulator Using the Phase-Field Method, Journal of Computational Physics 177 (2002) 264–283.

[2] W. Boettinger, J. Warren, C. Beckermann, A. Karma, Phase-field simulation of solidification 1, Annual review of materials research 32 (2002) 163–194.

[3] Z. Guo, J. Mi, P. S. Grant, An implicit parallel multigrid computing scheme to solve coupled thermal-solute phase-field equations for dendrite evolution, Journal of Computational Physics 231 (2012) 1781–1796.

[4] T. Shimokawabe, T. Aoki, T. Takaki, T. Endo, A. Yamanaka, N. Maruyama, A. Nukada, S. Matsuoka, Peta-scale phase-field simulation for dendritic solidification on the TSUBAME 2.0 supercomputer, in: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis, SC '11, ACM, New York, NY, USA, 2011, pp. 3:1–3:11.

[5] D. Sun, M. Zhu, S. Pan, D. Raabe, Lattice Boltzmann modeling of dendritic growth in a forced melt convection, Acta Materialia 57 (2009) 1755–1767.

[6] M. F. Zhu, D. M. Stefanescu, Virtual front tracking model for the quantitative modeling of dendritic growth in solidification of alloys, Acta Materialia 55 (2007) 1741–1755.

[7] H. Yin, S. D. Felicelli, Dendrite growth simulation during solidification in the LENS process, Acta Materialia 58 (2010) 1455–1465.

[8] H. Yin, S. D. Felicelli, L. Wang, Simulation of a dendritic microstructure with the lattice Boltzmann and cellular automaton methods, Acta Materialia 59 (2011) 3124–3136.

[9] D. K. Sun, M. F. Zhu, S. Y. Pan, C. R. Yang, D. Raabe, Lattice Boltzmann modeling of dendritic growth in forced and natural convection, Computers & Mathematics with Applications 61 (2011) 3585–3592.

[10] M. Eshraghi, S. D. Felicelli, An implicit lattice Boltzmann model for heat conduction with phase change, International Journal of Heat and Mass Transfer 55 (2012) 2420–2428.

[11] M. Eshraghi, S. D. Felicelli, B. Jelinek, Three dimensional simulation of solutal dendrite growth using lattice Boltzmann and cellular automaton methods, Journal of Crystal Growth 354 (2012) 129–134.

[12] D. H. Rothman, S. Zaleski, Lattice-Gas Cellular Automata: Simple Models of Complex Hydrodynamics, Aléa-Saclay, Cambridge University Press, 2004.

[13] S. Succi, The lattice Boltzmann equation for fluid dynamics and beyond, Oxford University Press, New York, 2001.

[14] M. C. Sukop, D. T. Thorne, Lattice Boltzmann Modeling - An Introduction for Geoscientists and Engineers, Springer, Berlin, 2006.

[15] D. Wolf-Gladrow, Lattice-Gas Cellular Automata and Lattice Boltzmann Models: An Introduction, Lecture Notes in Mathematics, Springer, 2000.

[16] P. L. Bhatnagar, E. P. Gross, M. Krook, A Model for Collision Processes in Gases. I. Small Amplitude Processes in Charged and Neutral One-Component Systems, Physical Review 94 (1954) 511–525.

[17] Y. H. Qian, D. D'Humiéres, P. Lallemand, Lattice BGK Models for Navier-Stokes Equation, EPL (Europhysics Letters) 17 (1992) 479.

[18] S. Chapman, T. G. Cowling, The Mathematical Theory of Non-uniform Gases: An Account of the Kinetic Theory of Viscosity, Thermal Conduction and Diffusion in Gases, Cambridge University Press, 1970.

[19] B. Deng, B.-C. Shi, G.-C. Wang, A New Lattice Bhatnagar-Gross-Krook Model for the Convection-Diffusion Equation with a Source Term, Chinese Physics Letters 22 (2005) 267.

[20] L. Beltran-Sanchez, D. M. Stefanescu, A quantitative dendrite growth model and analysis of stability concepts, Metallurgical and Materials Transactions A 35 (2004) 2471–2485.

[21] M. Baboulin, A. Buttari, J. Dongarra, J. Kurzak, J. Langou, J. Langou, P. Luszczek, S. Tomov, Accelerating scientific computations with mixed precision algorithms, Computer Physics Communications 180 (2009) 2526–2533.

[22] G. Wellein, T. Zeiser, G. Hager, S. Donath, On the single processor performance of simple lattice Boltzmann kernels, Computers & Fluids 35 (2006) 910–919.

[23] C. Körner, T. Pohl, U. Rüde, N. Thürey, T. Zeiser, Parallel Lattice Boltzmann Methods for CFD Applications, in: A. M. Bruaset, A. Tveito, T. J. Barth, M. Griebel, D. E. Keyes, R. M. Nieminen, D. Roose, T. Schlick (Eds.), Numerical Solution of Partial Differential Equations on Parallel Computers, volume 51 of *Lecture Notes in Computational Science and Engineering*, Springer Berlin Heidelberg, 2006, pp. 439–466.

[24] M. Wittmann, T. Zeiser, G. Hager, G. Wellein, Comparison of different propagation steps for lattice Boltzmann methods, Comput. Math. Appl. 65 (2013) 924–935.

[25] L. Adhianto, S. Banerjee, M. Fagan, M. Krentel, G. Marin, J. Mellor-Crummey, N. R. Tallent, HPCTOOLKIT: tools for performance analysis of optimized parallel programs, Concurrency and Computation: Practice and Experience 22 (2010) 685–701.

[26] The HDF Group., Hierarchical data format version 5, 2000-2010, `http://www.hdfgroup.org/HDF5`, 2012.

[27] A. Choudhury, K. Reuther, E. Wesner, A. August, B. Nestler, M. Rettenmayr, Comparison of phase-field and cellular automaton models for dendritic solidification in Al–Cu alloy, Computational Materials Science 55 (2012) 263–268.

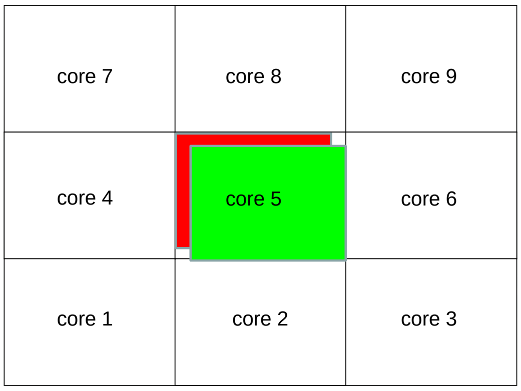
[28] M. Burtscher, B.-D. Kim, J. Diamond, J. McCalpin, L. Koesterke, J. Browne, PerfExpert: An Easy-to-Use Performance Diagnosis Tool for HPC Applications, in: 2010 International Conference for High Performance Computing, Networking, Storage and Analysis (SC), pp. 1–11.

[29] CRAY, Using Cray Performance Analysis Tools, `http://docs.cray.com/books/S-2376-52/S-2376-52.pdf`, 2011.

[30] I. Szcześniak, J. R. Cary, dxhdf5: A software package for importing HDF5 physics data into OpenDX, Computer Physics Communications 164 (2004) 365–369.

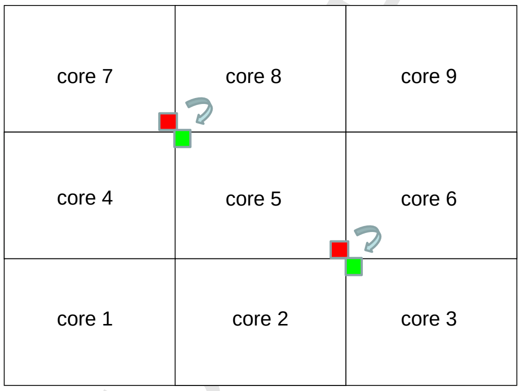
Appendix A. MPI operations

(a) Local streaming (propagation) of the matrix data.



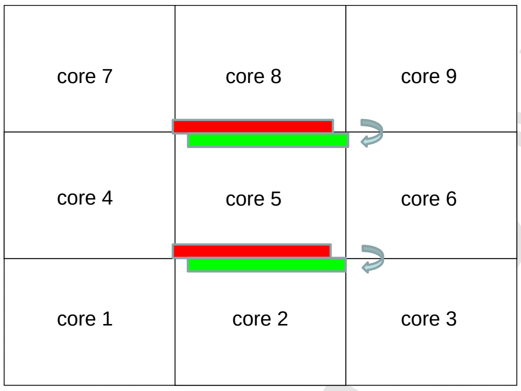
```
do i, j
  recv[i, j] = send[i-i, j-1]
end do
```

(b) Streaming of the corner values diagonally.



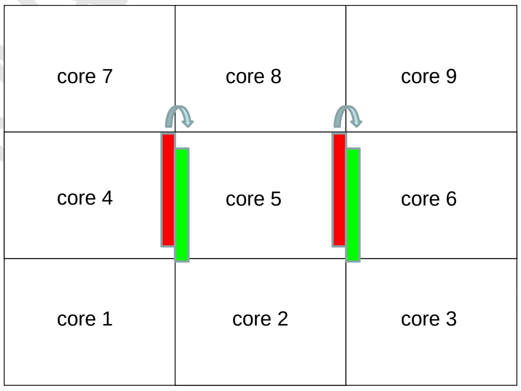
```
buffer = send
MPI_Sendrcv_replace(buffer, dst = 3, src = 7)
recv = buffer
```

(c) Streaming of the vertical boundary lines.



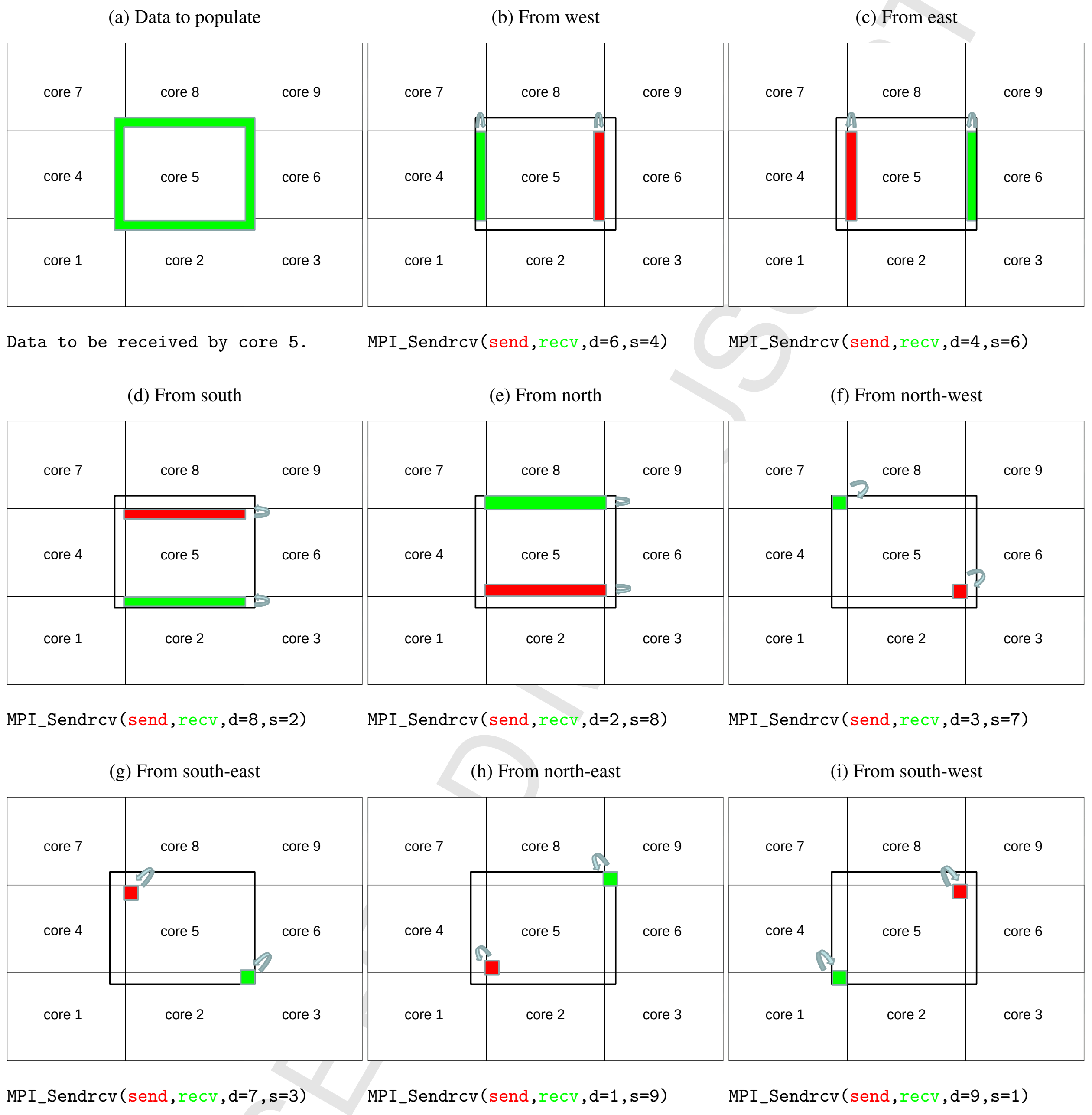
```
buffer = send
MPI_Sendrcv_replace(buffer, dst = 2, src = 8)
recv = buffer
```

(d) Streaming of the horizontal boundary lines.



```
buffer = send
MPI_Sendrcv_replace(buffer, dst = 6, src = 4)
recv = buffer
```

Figure A.6: Streaming of the distribution function f_8 (diagonal, south-west direction). All operations performed by the execution core 5 are shown. Red colored regions are to be propagated, green colored are the destination regions. The buffer is allocated temporarily to store the data to be sent and received. Streaming in the horizontal (vertical) direction involves one local propagation, and, unlike in the diagonal direction, only one MPI send-receive operation that propagates horizontal (vertical) boundary lines. Each core determines the rank of source and destination execution individually using MPI Cartesian communicator.



```
Data to be received by core 5.
MPI_Sendrcv(send,recv,d=6,s=4)
MPI_Sendrcv(send,recv,d=4,s=6)
MPI_Sendrcv(send,recv,d=8,s=2)
MPI_Sendrcv(send,recv,d=2,s=8)
MPI_Sendrcv(send,recv,d=3,s=7)
MPI_Sendrcv(send,recv,d=7,s=3)
MPI_Sendrcv(send,recv,d=1,s=9)
MPI_Sendrcv(send,recv,d=9,s=1)
```

Figure A.7: Populating the ghost values (green area of subfigure (a)) on the execution core 5. Each subdomain permanently stores an extra "ghost" layer of values to be received from (or to be sent to) the neighboring subdomains. Synchronously with receiving the data (recv), the execution core 5 sends the data (send) in the direction opposite to where the data is received from.

Program title: 2Ddend

Catalogue identifier: AEQZ_v1_0

Program summary URL: http://cpc.cs.qub.ac.uk/summaries/AEQZ_v1_0.html

Program obtainable from: CPC Program Library, Queen's University, Belfast, N. Ireland

Licensing provisions: Standard CPC licence, http://cpc.cs.qub.ac.uk/licence/licence.html

No. of lines in distributed program, including test data, etc.: 29767

No. of bytes in distributed program, including test data, etc.: 3131367

Distribution format: tar.gz

Programming language: Fortran 90.

Computer: Linux PC and clusters.

Operating system: Linux.

Has the code been vectorised or parallelized?: Yes. Program is parallelized using MPI. Number of processors used: 1-50000

RAM: Memory requirements depend on the grid size

Classification: 6.5, 7.7.

External routines: MPI (http://www.mcs.anl.gov/research/projects/mpi/), HDF5 (http://www.hdfgroup.org/HDF5/)

Nature of problem:
Dendritic growth in undercooled Al-3wt%Cu alloy melt under forced convection.

Solution method:
The lattice Boltzmann model solves the diffusion, convection, and heat transfer phenomena. The cellular automaton technique is deployed to track the solid/liquid interface.

Restrictions:
Heat transfer is calculated uncoupled from the fluid flow. Thermal diffusivity is constant.

Unusual features:
Novel technique, utilizing periodic duplication of a pre-grown "incubation" domain, is applied for the scale up test.

Running time:
Running time varies from minutes to days depending on the domain size and a number of computational cores.