Journal of Computational Physics **177,** 264–283 (2002)
doi:10.1006/jcph.2002.7005, available online at http://www.idealibrary.com on IDEAL®

A Parallel 3D Dendritic Growth Simulator Using the Phase-Field Method[1]

William L. George* and James A. Warren†

*Mathematical and Computational Sciences Division, National Institute of Standards and Technology, Gaithersburg, Maryland 20899; and †Metallurgy Division and Center for Theoretical and Computational Materials Science, National Institute of Standards and Technology, Gaithersburg, Maryland 20899
E-mail: william.george@nist.gov; james.warren@nist.gov

Received March 31, 2001; revised January 8, 2002

We describe an implementation of a parallel finite-difference algorithm for the simulation of alloy solidification in three dimensions using the phase-field model. We also describe the visualization of the output from this simulator. Although this type of simulation has been accomplished before in two dimensions, extending this to three dimensions presents scaling problems for both the computations and the subsequent rendering of the results for visualization. This is due to the $\mathcal{O}(n^4)$ execution time of the simulation algorithm as well as the $\mathcal{O}(n^3)$ space requirements for holding the required three-dimensional arrays of field parameters. Additionally, rendering the output of the three-dimensional simulation stresses the available software and hardware when the simulations extend over computational grids of size $500 \times 500 \times 500$. Parallel computing libraries and hardware-supported rendering combine to help make this simulator simple to implement, portable, and efficient even when run in heterogeneous environments. This has all been accomplished using simple static uniform grids and finite differencing.

Key Words: phase-field method; dendritic growth; binary alloys; finite difference; parallel processing; MPI.

1. INTRODUCTION

The microstructures which form during the solidification (freezing) of a material play an enormous role in the properties of the solid material. In particular, during the solidification of an alloy, the microsegregation patterns (i.e., the distribution of one alloy component in the

[1] Certain commercial products are identified in this paper in order to adequately describe this simulator and related hardware and software. Such identification is not intended to imply recommendation or endorsement by the National Institute of Standards and Technology, nor is it intended to imply that the identified products are necessarily the best available for the purpose.

0021-9991/02 $35.00

other at a microscopic level) which result during dendritic and/or cellular solidification of an alloy are of substantial interest to the materials engineer. Modeling the motion of solute in the dendritic branches has be an active area of research for many years. For example, the transport of solute in the interdendritic mushy regions by diffusion has been modeled by Bower *et al*. [1] and Hunt [2], while the coarsening (reduction of surface area) of the dendritic structure has been done by (Kattamis *et al*. [3], Feurer and Wunderlin [4], Kirkwood [5], Mortensen [6], and Voller and Sundarraj [7]). Other work has modeled the diffusion of solute in the solid (Brody and Flemings [8], Battle and Pelke [9], Kobayashi [10]), and Beckermann *et al*. [11] were interested in fluid flow around the dendrite. A summary of much of this work is available in Kurz and Fisher [12]. Also, Osher and Tryggvason have recently edited a special issue of this journal on the general topic of computational methods for multiphase flows [13], including an article by Tryggvason *et al*. [14] that explores a front-tracking method for multiphase flow and gives a review of the various techniques that have been developed for solving this and related problems.

Recently, Warren and Boettinger (WB) [15] developed a phase-field model of a binary alloy, based on a great deal of earlier work on the phase field method in pure materials [16–18] and alloys [19, 20]. The WB work was done in two dimensions, although there was nothing in the equations limiting the dimensionality. The limitation to two dimensions was mostly due to the lack, at the time, of the necessary computing power required to perform the calculations in three dimensions, as well as the algorithms to implement the calculations in an efficient manner. However, with the advent of parallel programming environments and parallel computers, as well as the algorithms to do such calculations, it is now possible to calculate 3D alloy microstructures.

This work was motivated by the limitations of WB's method, in an effort to develop a framework for the simulation of 3D alloy microstructures. There have been several other efforts at 3D phase-field simulation of dendrites, some using parallel algorithms, each with certain goals in mind: Kobayashi showed the first 3D phase field dendrites [21]; Karma and coworkers [22–25] studied the shape of the dendritic tip and developed the *thin interface* limit of the phase-field equations, which reduced the overall computational load by allowing the use of a larger grid spacing (Δx), and therefore a smaller computational grid for a given size simulation; and Jeong *et al*. [26] have studied the effects of fluid flow on dendritic growth in 3D using a parallel finite-element algorithm with adaptive grid refinement. Schmidt [27] developed a 3D dendritic growth simulator using two coupled finite-element algorithms with totally independent dynamic grids in which one algorithm solves the heat equation and the other tracks the boundary of the evolving liquid–solid interface. Other simulations involving moving boundaries in three dimensions have been developed with parallel algorithms: Aliabadi and Tezduyar [28] deal with moving boundaries in 3D for aerospace applications using a finite-element method with deformable spatial grids, and Zhou and Derby [29] studied the sintering of ceramic particles in 3D using finite-element methods with moving spatial grids.

The simulator we have developed is distinguished from other related parallel 3D simulators in several aspects. Because of our use of a static regular grid and a data-parallel style finite-differencing algorithm, the source code is relatively short: approximately 5200 lines of C code in total, including all comments, input/output routines, checkpoint and restart codes, timing and other instrumentation codes, and debugging codes. This small source code has made this simulator easy to maintain and easy to modify as we have experimented with the phase-field model. Compared to algorithms based on the use of dynamic grids, this design

can be viewed as a trade-off between high memory requirements (to hold the large dense grids) and the additional code complexity needed to manage computations on dynamic and irregular grids. The relatively small size of the source code is also due to the use of the MPI-based library DPARLIB, which offloads most of the effort in managing distributed arrays. Also, this simulator uses dynamic load balancing to maintain good performance when run on clusters of heterogeneous machines and to compensate for nonuniform computational loads inherent in the algorithm. This has resulted in a simulator that runs efficiently in a wide variety of environments, from parallel machines such as IBM SPs and SGI Origins to clusters of heterogeneous PCs and workstations.

In summary, our effort in this project has been fourfold: (i) the translation of the 2D WB code to a 3D parallel code; (ii) its implementation as a generic, modular, and portable code, allowing for application to the variety of partial differential equations which are found in the physical sciences; (iii) the visualization of the simulated phenomena in the "best" manner possible; and (iv) the exploration of phenomena inherently three dimensional in nature. This manuscript addresses our progress in addressing these four tasks and details those areas where challenges remain.

2. PHASE-FIELD MODEL

The WB phase-field model of isothermal binary alloy solidification can be expressed as two differential equations. To determine these equations we start with an entropy of the form

$$S = \int dV \left(s(\phi, c, e) - \frac{\epsilon^2}{2} \Gamma^2(\nabla\phi) \right), \tag{1}$$

which is motivated by the Cahn–Hoffman ξ-vector work of Wheeler and McFadden [30]. The entropy density s is a function of the phase-field ϕ, the energy density e, and the solute concentration c. The $\phi = 1/2$ contour is chosen as the location of the interface between liquid $\phi = 0$ and solid $\phi = 1$. The explicit form of $s(\phi, c, e)$ can be found in WB. The gradient penalty function $\Gamma(\nabla\phi)$ is a homogeneous degree-one function of its argument, i.e.,

$$\Gamma(a\nabla\phi) = a\Gamma(\nabla\phi), \tag{2}$$

where a is a constant. For this work we use a different model of Γ, appropriate for 3D [21],

$$\Gamma = c_1|\nabla\phi| + c_2 \frac{\phi_x^4 + \phi_y^4 + \phi_z^4}{|\nabla\phi|^3}, \tag{3}$$

where subscripts indicate a derivative and c_1 and c_2 are constants. For all of these calculations we take $c_1 = 1 - 3\gamma_s$ and $c_2 = 4\gamma_s$, where γ_s represents the deviation of the system from isotropy. This choice of c_1 and c_2 reduces to the anisotropy choice of WB, in two dimensions.

Using the formalisms of irreversible thermodynamics, we are able to derive three gradient flow equations from the above entropy: $\partial e/\partial t$, $\partial c/\partial t$, and $\partial\phi/\partial t$. We will ignore the energy equation, as was done in WB, and perform calculations at constant temperature. With this set of assumptions we are left with two equations of evolution, the first being the phase-field

equation

$$\frac{1}{M_\phi}\frac{\partial \phi}{\partial t} = \epsilon^2 \sum_{i=1}^{3} \frac{\partial}{\partial x_k}\left(\Gamma \frac{\partial \Gamma}{\partial \phi_{x_k}}\right) - (1-c)H^A - cH^B. \tag{4}$$

M_ϕ is a constant set by kinetic considerations discussed below, it is usually different in the two pure phases, and it can be written as

$$M_\phi = (1-c)M^A + cM^B, \tag{5}$$

where $M^{A,B}$ are the mobilities of an interface in pure A, B. If $\gamma_s = 0$ then $\partial\phi/\partial t$ satisfies an isotropic diffusion equation with a source term. The first right hand term in Eq. (4) is multiplied by the gradient entropy coefficient ϵ^2; the remaining terms are the H^A and H^B, which are defined by

$$H^A(T,\phi) = W^A \frac{\partial g}{\partial \phi} + 30 g(\phi) L^A \left(\frac{1}{T} - \frac{1}{T_m^A}\right), \tag{6}$$

$$H^B(T,\phi) = W^B \frac{\partial g}{\partial \phi} + 30 g(\phi) L^B \left(\frac{1}{T} - \frac{1}{T_m^B}\right). \tag{7}$$

In these expressions, $W^{A,B}$ are the free energy barrier heights multiplying the derivative of the double well $g(\phi) = \phi^2(1-\phi)^2$, while $L^{A,B}$ are the latent heats, T is the temperature, and $T_m^{A,B}$ are the melting temperatures of pure components A, B.

The second differential equation in the WB model governs the evolution of the concentration c,

$$\frac{\partial c}{\partial t} = \nabla \cdot D_c \left[\nabla c + \frac{v_m}{R} c(1-c)(H^B - H^A)\nabla \phi\right], \tag{8}$$

with

$$D_c = D_S + p(\phi)(D_L - D_S). \tag{9}$$

In Eq. (9), D_S is the bulk solid diffusion coefficient and D_L is the bulk liquid diffusion coefficient. The function $p(\phi)$ is a smoothed step function such that $p(0) = 0$ and $p(1) = 1$. In WB $p(\phi)$ was chosen as $p(\phi) = \phi^3(10 - 15\phi + 6\phi^2)$. Thus, for a single-phase liquid or solid, $\phi = 0$ or 1 everywhere, and Eq. (4) reduces to the ordinary solute diffusion equation.

The above equations form the basis of the mathematical problem we wish to solve in three dimensions. These equations are often compared with the so-called "sharp interface" equations of solidification. The term sharp interface has really only been in use since the advent of the phase-field method, which introduced the notion of a diffuse interface to the solidification problem. The sharp interface equations can be obtained from the phase-field equations in the limit that the interface thickness $\delta \to 0$. When taking this limit we need to determine what quantities to hold fixed, and, following WB, we keep the surface energy constant. With this choice we may relate sharp interface materials parameters with phase-field parameters. In particular we have that

$$W^A = 18\left(\frac{\sigma^A}{T_M^A \epsilon}\right)^2, \tag{10}$$

where σ^A is the surface energy of material A. The thickness of the interface δ^A is defined by solving the one-dimensional phase-field equation for isothermal coexistence of liquid and solid at a planar interface of pure A to find

$$\phi(x) = \frac{1}{2}\left[1 + \tanh\left[\frac{\sqrt{W^A}x}{\sqrt{2}\epsilon}\right]\right], \tag{11}$$

where x is the spatial coordinate, and then identifying

$$\delta^A = \frac{\epsilon}{\sqrt{4W^A}}. \tag{12}$$

Note that with this definition ϕ changes from 0.1 to 0.9 over a distance of approximately $6\delta^A$. M^A is given by

$$M^A\epsilon^2 = \frac{\sigma^A T_M^A \beta^A}{L^A}, \tag{13}$$

where β^A is the linear kinetic coefficient for interface attachment that relates interface undercooling to interface speed by $V_n = \beta^A(T_M^A - T)$. Since all of the above equations can also be written for $A \to B$, and ϵ is assumed to be independent of concentration, there is a restriction on the independence of these material parameters, specifically

$$\frac{\sigma^A\delta^A}{T_M^A} = \frac{\sigma^B\delta^B}{T_M^B}. \tag{14}$$

Although M^A, M^B, ϵ, W^A, and W^B appear naturally in these expressions for the phase-field dynamics, they are not parameters which are typically employed in solidification. With this in mind we rewrite the above relations so that, given σ^A, δ^A, δ^B, β^A, and β^B, we can determine these less-intuitive parameters (σ^B is fixed from Eq. (14)). We find

$$\begin{aligned} M^A &= \frac{\left(T_M^A\right)^2\beta^A}{6\sqrt{2}L^A\delta^A}; \quad M^B = \frac{\left(T_M^B\right)^2\beta^B}{6\sqrt{2}L^B\delta^B}; \quad W^A = \frac{3\sigma^A}{\sqrt{2}T_M^A\delta^A}; \\ W^B &= \frac{3\sigma^B}{\sqrt{2}T_M^B\delta^B}; \quad \epsilon^2 = \frac{6\sqrt{2}\sigma^A\delta^A}{T_M^A} = \frac{6\sqrt{2}\sigma^B\delta^B}{T_M^B}. \end{aligned} \tag{15}$$

With these relations we now examine how to best solve the phase-field equations.

3. SIMULATION PARAMETERS

The simulation solves Eqs. (4) and (8), using finite-difference approximations to the derivatives, over a 3-D grid of uniformly spaced points. Each grid point is centered in a volume of size $\Delta x \times \Delta y \times \Delta z$. The simulations use a cubic volume letting $\Delta x = \Delta y = \Delta z$. The time step between iterations is Δt. The values for Δx and Δt are determined so that the computation remains stable as the simulation progresses and so that we obtain sufficiently resolved pictures of the dendrite. The values used for Δt and Δx are determined by the

material parameters, such as melting point, and by the initial conditions in the simulation, such as temperature and the relative concentrations of the materials.

The tests that have been run during the development of this code simulate the solidification of a binary alloy with cubic anisotropy. Nickel material parameters were chosen for one of the materials (material A) and an approximation to copper for the other (material B). The system is initialized in a supersaturated state with a small seed crystal introduced to initiate the solidification process. The temperature of the material is held constant during the entire simulation.

Values for the material parameters used for the test runs of the simulator shown in this paper are melting temperatures of materials A and B, $T_M^A = 1728$ K and $T_M^B = 1000$ K; solid and liquid diffusion coefficients, $D_L = 1.0 \times 10^{-5}$ cm^2/s and $D_S = 1.0 \times 10^{-9}$ cm^2/s (same for both materials A and B); latent heats $L^A = 2350$ J/cm^3 and $L^B = 1728$ J/cm^3; the molar volume of the material; $v_m = 7.42$ (cm^3); the deviation from isotropy, $\gamma_s = 0.03$; and the linear kinetic coefficients for interface attachment; $\beta^A = 0.33$ and $\beta^B = 0.39$ cm/(K $\cdot$ s).

To fully specify a run of this simulator, values for several other parameters must be given. The material temperature is held at $T = 1356$ K in the entire volume and for the duration of the simulation. The initial relative concentration of materials A and B throughout the volume is set to $c_\infty = 0.40831$. This value, along with the material temperature results in a supersaturated state at the beginning of the simulation. The liquid-to-solid interface thicknesses used are $\delta^A = 6.11 \times 10^{-6}$ cm and $\delta^B = 4.50 \times 10^{-6}$ cm. The grid spacing, Δx, is chosen based on several criteria. In order to resolve the dendrite to a sufficient level of detail, Δx should be as small as possible. However, for a given edge length L, as the resolution Δx gets smaller, the number of grid points $n = L/\Delta x$ increases. Since the execution time for this phase-field algorithm, as discussed in the following section, is $\mathcal{O}(n^4)$, and the space requirement is $\mathcal{O}(n^3)$, keeping Δx as large as possible is desirable. Similarly, in order to allow the dendrite to fully develop, the size of the volume must not be too small, i.e., L must be large. From experience gained from both the two-dimensional and three-dimensional simulators it has been determined that a Δx of approximately 3.5×10^{-6} cm reveals sufficient detail. The volume of material we wish to simulate is a cube approximately 3.5×10^{-3} cm on a side. This corresponds to a finite-difference grid of size $1000 \times 1000 \times 1000$. So, our immediate goal is to be able to perform simulations on grids of this size.

Like the grid spacing, Δx, there are also restrictions on the time-step Δt. As Δt increases, the execution time of the simulation decreases. However to maintain stability in this finite-difference algorithm over time-step iteration the Δt must not be too large relative to Δx. Stability analysis for a diffusion equation, with diffusion constant D, shows that the time step must be chosen such that $\Delta t < \Delta x^2/6\text{D}$ [31]. Our equations are more complex, but through experimentation the setting $\Delta t = \Delta x^2/8D_L$ has been found to work well.

The initial seed crystal for the dendrite in the phase-field volume is placed in one corner of the volume and reaches $10\Delta x$ along each axis from that corner. This diamond-shaped area is initialized to $\phi = 1.0$ (solid) while the remainder of the volume is initially set to $\phi = 0.0$ (liquid). To initiate the generation of side branches along the dendrite, noise is inserted into the simulation.

For a 200^3 grid, a simulated time of 0.2 ms was found to be sufficient for the dendrite to reach from the seeded corner of the volume to the adjacent corners. Since the speed of the tip of the dendrite is nearly constant, for simulations on other-size volumes, this time is scaled linearly with the length of an edge of the volume.

4. IMPLEMENTATION

This simulation code, called bin3d (binary alloy in three-dimensions), is implemented as a portable data-parallel style program using the C programming language and the message-passing library MPI [32].

As described in the previous section, this simulation solves Eqs. (4) and (8), using finite-difference approximations to the derivatives, over a 3D grid of uniformly spaced points.

The main data structures, therefore, are the 3D arrays that hold the current state of the phase field and the current concentration of the solute. These arrays are distributed among the available processors and, for each time step, each point in the phase and concentration volumes is updated by the processor that owns that point.

This simple scheme was the result of a nearly direct translation of a serial version of this simulation to a parallel version. The translation from serial to parallel was greatly simplified by the use of the portable library DPARLIB [33, 34] that has been designed to support portable data-parallel programming in C. This library, which uses MPI for all interprocessor communications, manages most of the details of data distribution which are common to data-parallel-style MPI programs. DPARLIB also supports many common data-parallel operations, such as array shifting, data reductions, and elemental computations on n-dimensional arrays (for $1 \leq n \leq 7$). These operations, along with the direct use of the MPI library, simplifies the coding of data-parallel algorithms and results in portable applications. We have been able to run bin3d on all of our available parallel machines without changes to the source code.

In addition to the array-shifting operations, DPARLIB provides support for the computation of *stencils*. A stencil is a computation that updates each element of an array with a weighted sum of neighboring elements. This weighted sum forms a pattern (stencil) around each array element that is applied at each element. For example, many 2D finite-difference algorithms compute what is known as a 5-point stencil, such as

$$A[i, j] = w_1 B[i, j] + w_2 B[i + 1, j] + w_3 B[i - 1, j] + w_4 B[i, j - 1] + w_5 B[i, j + 1],$$

where the w_n values are the weights. Figure 1 shows this stencil computation at one element of a 10×10 array graphically.

The *depth* of a stencil is the maximum distance from the center point of the computation to any of the elements included in the stencil. For the 5-point stencil shown above, the stencil depth is 1, which is common. When such a stencil computation is performed in a

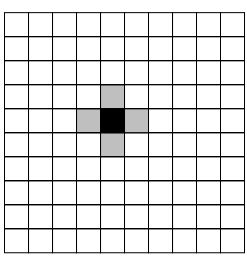

FIG. 1. A 5-point stencil. The center element (black) is the element that is being computed. The four surrounding shaded elements are used in the stencil computation. This pattern is repeated at each element of this 10×10 array.

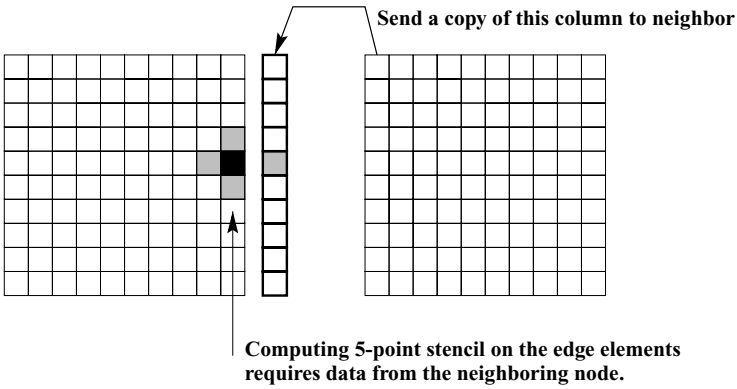

FIG. 2. Five-point stencil at the edge of the local subarray. Each 10×10 array is held in a separate node and so to compute the stencil along the edge of the local array the neighboring node must send one column of elements.

data-parallel program, each process must have access to some array elements not local to that process. Support for this type of computation is provided by DPARLIB with routines that can gather the required array elements that are owned by neighboring processors (see Fig. 2). This gathering can be viewed as a special type of array shift, with a shift distance equal to the stencil depth, in which the local subarray is not moved but the neighboring elements are received and placed into a separate *border* array. In the phase-field simulation bin3d, almost all of the communication required is handled by these DPARLIB support routines.

The updates for both the phase and concentration fields in this simulation are based on 3D 7-point stencils. The stencil points consist of the center point plus all elements directly adjacent along all three array axes. To simplify the parallel code, all of the main 3D arrays are distributed among the processors along only one axis. Since the stencils computed in this algorithm are all of depth 1, these computations require only that the neighboring 2D slice of each 3D array be gathered for each update. For performance reasons, the x axis was chosen to be distributed so that the communication of these subarray slices would involve only contiguous blocks of data. So, for a simulation over a grid of size $n_x \times n_y \times n_z$ using P processors, and assuming an even distribution of these arrays, each processor will own a subarray of size $n_x/P \times n_y \times n_z$ and the stencil computations will require the communication of subarray slices of size $n_y \times n_z$.

There are two problems that have been encountered using the simple data distribution scheme of assigning an equal-size subarray to each processing node. First, due to details of the implementation of the update algorithm, the amount of computation needed to update each grid point depends partially on its current phase. Grid points that are in the liquid phase, with $\phi = 0.0$, require less computation to update than grid points that have $\phi > 0.0$. This creates a small load imbalance that changes over time as the dendrite grows. Second, and more important, our machines are not comprised of sets of identical processors. In particular, our IBM SP contains three different types of CPUs: 66-MHz POWER2, 120-MHz POWER2, and 200-MHz POWER3 CPUs. Dividing the grid points evenly between processors in this environment can therefore cause large load imbalances. To solve this problem, DPARLIB supports dynamic load balancing in which the arrays can be redistributed as needed to, for example, match the compute power of each processor. Briefly, this load balancing is accomplished by periodically (e.g., once per hour) measuring the compute time needed

per iteration for each node and redistributing the arrays such that this metric is the same for each node. This technique helps avoid load imbalances caused by both of the problems described here since it simply detects the existence and amount of a load imbalance and does not attempt to determine the cause of the load imbalance. Using this generic form of dynamic load balancing as provided by DPARLIB required only minor additions to the source code of bin3d and greatly improved its overall performance.

Another optimization made in implementing this parallel algorithm was to allow for the possibility of concurrent computation and communication. This can minimize the communications overhead that is inherent in parallel message-passing programs. On each iteration of the algorithm (each time step), each element of the phase-field and concentration arrays are updated. For each iteration, each processor first initiates the sending of its local boundary subarray slices to its neighbors through a call to a nonblocking DPARLIB routine. While this communication proceeds, each processor updates only the *interior* elements of its subarray. The interior subarray elements are those elements that do not require any data from the neighboring nodes in order to be updated. For the array distribution used here, this includes all subarray elements except those that are on the extreme ends of the x axis in the local subarray. Once these interior elements are updated, each node then waits to receive the required boundary slices, which should have arrived, assuming the number of interior elements is large enough to to require sufficient computation time and assuming the machine supports concurrent communication and computation. After receiving the boundary slices, each processor finishes the updating of their subarray elements to complete the iteration.

At regular intervals in these iterations, a snapshot of the system is taken by writing out the phase and the concentration arrays to files. This snapshot is occasionally accompanied by a dump of checkpointing information so that the simulation can be restarted if needed. The frequency of checkpointing is set independently of the snapshot frequency.

The space requirement of this implementation is $\mathcal{O}(n^3)$ since it is dominated by the 3D arrays that are used. There are two phase-field arrays and two concentration arrays, one each for the current and previous iterations. Each update also uses three other arrays: two to allow for efficient and mass-conserving finite differences of the solute field and one of uniformly distributed pseudorandom numbers. One additional array is used for some intermediate results during an array-reduction operation. So a total of eight arrays of size $n_x \times n_y \times n_z$ are used. All of these arrays hold C `double` elements, which are 8 bytes each. The stencil computations also require space for each processor to hold the boundary slices. Each processor uses six boundary slices of size $n_y n_z$. If we run on P processors, this adds $6Pn_y n_z$ elements to the space requirement. For the larger simulations the space required for these boundary slices is insignificant when compared to the full 3D arrays. The largest run of bin3d to date has been on a grid of size 500^3, which therefore required approximately 8 GB of memory. This simulation was run on 32 processors so the per-processor memory requirement was approximately 0.25 GB.

For each time step (iteration) the algorithm requires $\mathcal{O}(n^3)$ operations to update the phase-field and concentration arrays. The intent of each run is to allow the main tip of the dendrite, which is seeded at one corner of the volume, to reach the opposite corners of the volume along the three axes. The speed of propagation of the dendrite tip is nearly constant, so as the size of the volume is increased, the distance the dendrite tip must travel increases linearly, which means that the number of time steps required to fully develop the dendrite is $\mathcal{O}(n)$. Ignoring the communications requirements, the execution time of this simulation is therefore $\mathcal{O}(n^4)$.

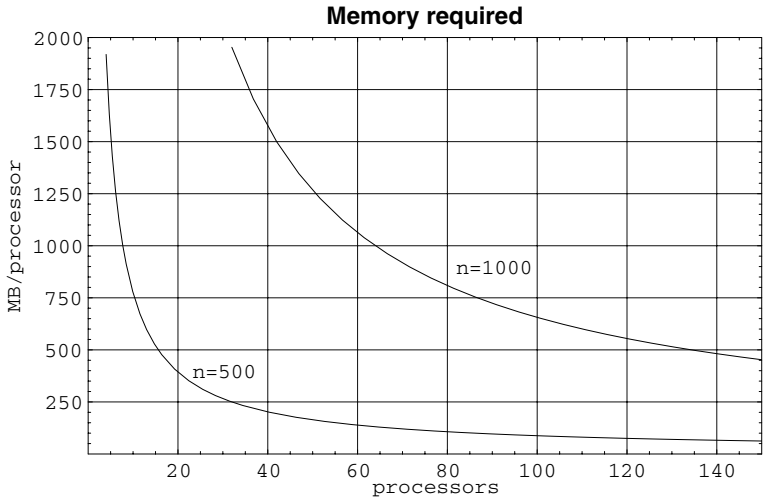

FIG. 3. Memory requirements of the bin3d simulator. Assuming that processing nodes contain at least 1 GB of memory per processor, this plot shows that 70 or more nodes will be needed to complete a simulation on a 1000^3 volume.

Between iterations, each processor must exchange 2D slices of the phase and concentration arrays with the adjacent processors. The cost of each exchange is therefore $\mathcal{O}(n^2)$ and the number of such exchanges is the same as the number of iterations, which is $\mathcal{O}(n)$. Communication costs should therefore be $\mathcal{O}(n^3)$. For a fixed number of processors, as the problem size increases, the communications cost will quickly become insignificant compared to the computational cost. Also, the ability to overlap this communication with the computation can hide much of this overhead.

A series of timing tests have been run while developing this application in order to determine the resources that will be needed to regularly run simulations using our target size of $n = 1000$. A simple model was developed to predict the execution time based on problem size n and the number of processors p.

Figure 3 shows the estimated per-processor memory requirement of bin3d for grids of size 500^3 and 1000^3 and for 4–150 processors. A simulation on a 1000^3 grid is not too large for most computing sites with moderately large clusters. For a typical IBM SP or an SGI Origin 2000 with least 1 GB of memory per node, 70 or more nodes would be adequate for this problem size. Also, using a typical Beowulf cluster (PC cluster running Linux), with 512–1000 MB of memory per node, from 70 to 150 nodes would be needed to run a simulation over a grid of size 1000^3.

Figure 4 shows the estimated time required to complete a simulation for various-size grids using 24 200-MHz POWER3 processors of an IBM SP. Each processing node on the IBM contained 2 GB of memory. This graph is based on several test runs of the simulator for a grids of size n^3 for $n = 50, 100, 150, 200, 250,$ and 300, and for even numbers of processors from 2 to 24. Some combinations were not run due to memory limitations. Since bin3d executes an $\mathcal{O}(n^4)$ algorithm, the results from those test runs were used to do a least-squares fit to a fourth-degree polynomial in n while assuming perfect speedup with the number of processors. The execution time model used is therefore $T(n, p) = c_1(n/p) + c_2(n^2/p) + c_3(n^3/p) + c_4(n^4/p)$. This is the curve plotted in the Fig. 4 for $p = 24$ along with the actual timing results. The same information is shown in Fig. 5 for

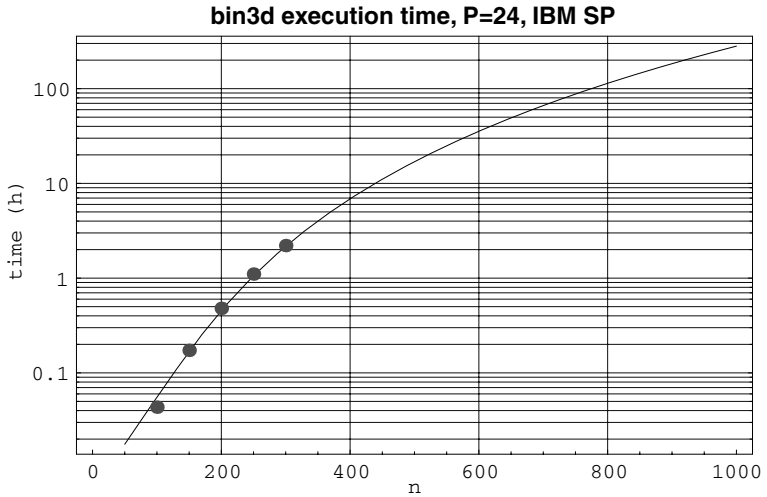

FIG. 4. Execution time, in hours, versus problem size n for the bin3d simulator. This graph shows bin3d performance on 24 processors of an IBM SP with 200-MHz POWER3 processors. The problem size n defines the number of grid points along each edge of the cubic volume in which the simulation occurs. The discrete points plotted are the results from actual test runs. The plotted curve is the execution time function $T(n, p = 24)$, fitted to timing data from tests on our IBM SP. This curve is extended out to the target problem size of 1000^3.

a Beowulf cluster of $p = 16$ PCs, each with a 333-MHz Intel Pentium II processor. These graphs show that the simple $T(n, p)$ model fits the timing date closely.

As an additional check on the $T(n, p)$ execution time model, Fig. 6 shows a graph of $T(n, p)$ versus the number of processors for a problem of size $n = 250$. This estimate is

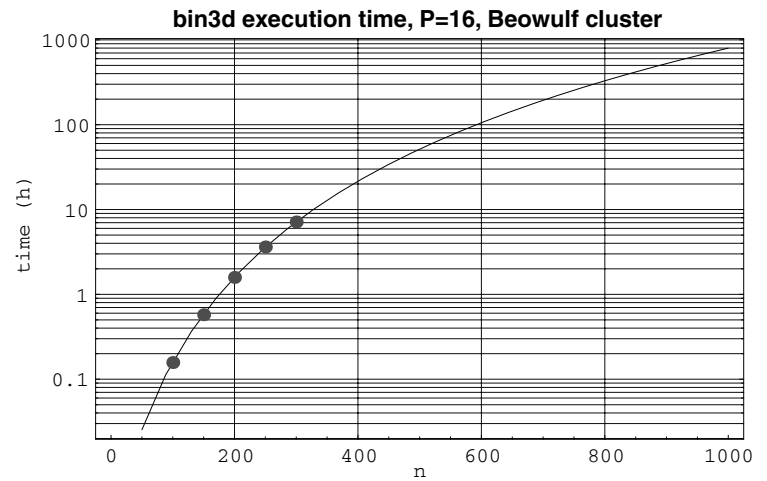

FIG. 5. Execution time, in hours, versus problem size n for the bin3d simulator. This graph shows bin3d performance on 16 processors of a Beowulf cluster of 333-MHz Intel Pentium II processors. The problem size n defines the number of grid points along each edge of the cubic volume in which the simulation occurs. The discrete points plotted are the results from actual test runs. The plotted curve is the execution time function $T(n, p = 16)$, fitted to the timing data from tests on our Beowulf cluster. This curve is extended out to the target problem size of 1000^3.

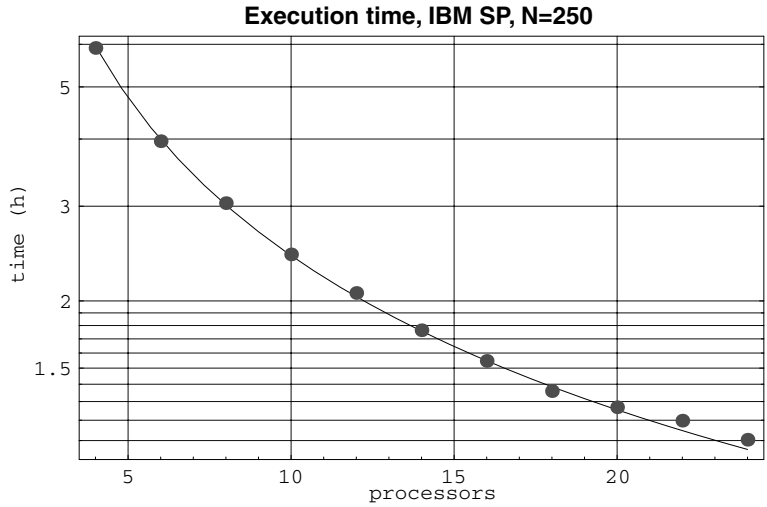

FIG. 6. Execution time, in hours, versus number of processors for a grid of size $250 \times 250 \times 250$. This graph shows actual timing results along with a plot of the $T(n, p)$ execution time model for our IBM SP with a problem size $n = 250$.

plotted along with actual simulation times for $n = 250$ and an even numbers of processors, from 4 to 24, for the IBM SP. A similar graph is shown in Fig. 7 for the 16-processor Beowulf cluster. These graphs demonstrate that our predictions, especially for larger values for n/p, are accurate. At small n/p ratios communications overhead becomes significant with respect to the number of computations performed per iteration and our predictions become inaccurate. However, since typical problem sizes of interest ($n \geq 300$) along with the size of the parallel machines available results in high n/p ratios, these predictions should be reliable.

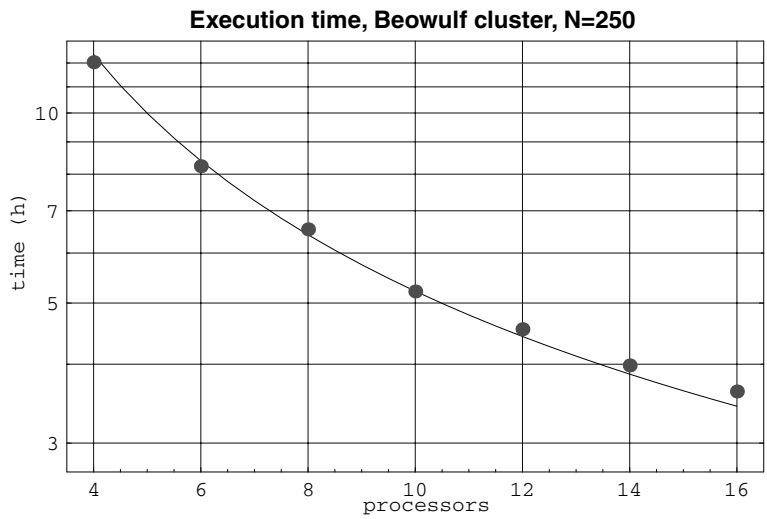

FIG. 7. Execution time, in hours, versus number of processors for a grid of size $250 \times 250 \times 250$. This graph shows actual timing results along with a plot of the $T(n, p)$ execution time model for our Beowulf cluster with a problem size $n = 250$.

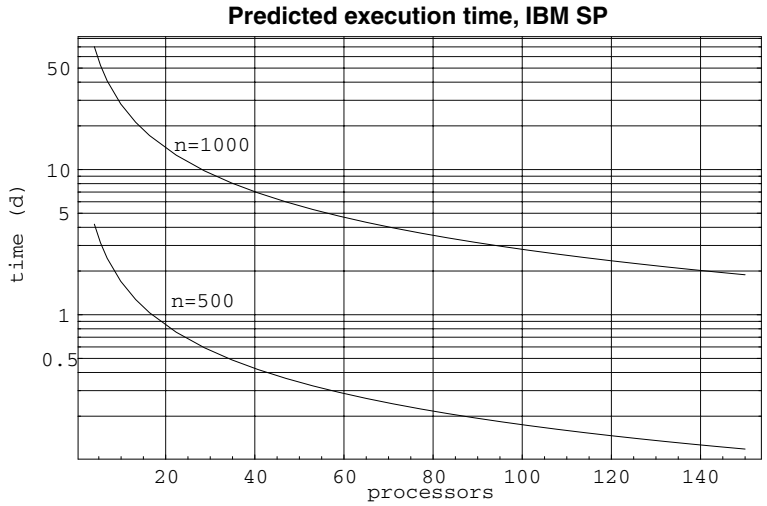

FIG. 8. Estimated execution time, in days, versus number of processors for the bin3d simulator on an IBM SP with 200-MHz POWER3 CPUs.

Finally, in Figs. 8 and 9 the estimated execution time versus the number of processors is shown for bin3d on larger versions (more processors) of both our IBM SP and Beowulf cluster. Due to the memory requirements, it was previously determined that at least 70 IBM SP nodes would be needed, or a PC cluster with at least 140 nodes, to run a simulation with $n = 1000$. According to the data shown in Figs. 8 and 9 the predicted times for these simulations will be about 4 days for the IBM SP and about 5 days for the Beowulf cluster. On the same-size machines, a simulation with $n = 500$ would take about 6 h (0.25 days) on the IBM SP and about 7 h (0.3 days) on the Beowulf cluster.

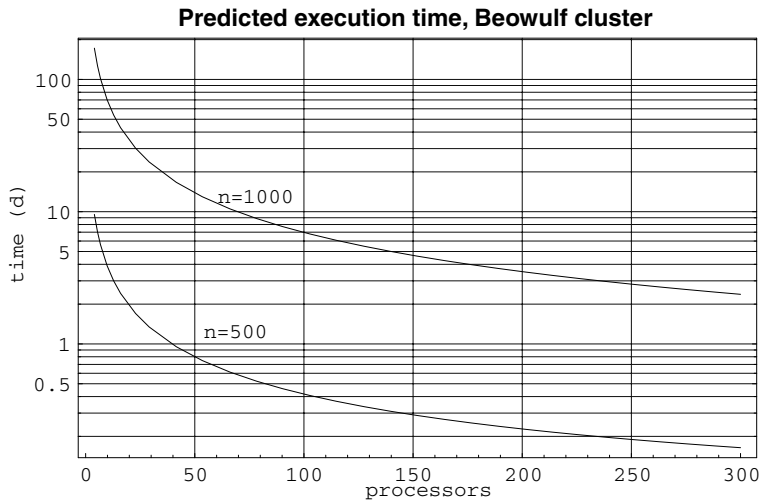

FIG. 9. Estimated execution time, in days, versus number of processors for the bin3d simulator for a Beowulf cluster of PCs with 333-MHz Intel Pentium II CPUs.

5. VISUALIZATION

The output from the bin3d program consists of snapshots of the phase and concentration volumes taken at equally spaced simulation time intervals. For each point in each volume, an 8-bit value is stored. For the phase data, this value is a direct mapping from the floating point range 0.0–1.0 to the integer range 0–255. The concentration ratios are also limited to the range 0.0–1.0, but in the simulations the actual concentrations never exceed 0.6 or drop below 0.2. This range is an estimate based on previous results and on the chemistry of the simulation. So, to visualize with the highest resolution, the concentration values are mapped from the floating point range 0.2–0.6 to the integer range 0–255. If the concentration at any point is outside the 0.2–0.6 range during a snapshot, an error message is printed and the snapshot value at that point is forced to be the nearest extreme (0 or 255). This has not yet occurred.

To conserve disk space, these snapshot files are compressed. The snapshots early in the simulations contain many duplicate values and compress greatly, by over a factor of 200 when running a simulation on a 300^3 volume. As the simulation progresses these compressed snapshot files grow in size. For example, for one particular simulation on a 300^3 grid, the 27-MB phase volume compresses to about 123 KB for the initial snapshot and slowly grows to nearly 1 MB for the final snapshot of the simulation, which is still over a factor of 10 reduction in file size.

A number of snapshots, usually from 40 to 100, are taken during a simulation in order to produce a smooth animation showing the growth of the dendrite. Since the phase of the alloy is represented as a continuum from liquid to solid and not as either solid or liquid, the surface of the dendrite is arbitrarily assigned to be at the midpoint-phase value. In the snapshot files for the phase this corresponds to the byte value 128. So, the first task is to compute an isosurface of the phase snapshot at the value of 128. Next, the concentration data is used to choose a color for the surface of the dendrite at each point on this isosurface. Each of the frames rendered in this way is then stored in a separate file.

This processing from the raw compressed snapshot files to rendered TIFF images is automated using batch scripts written in IDL (interactive data language, from Research Systems, Inc.) and the unix Bourne shell. Since each snapshot is completely independent, the scripts start multiple concurrent IDL sessions to process the snapshots in parallel. We are currently using a 14-processor SGI Onyx2 to render these frames.

The snapshot files from bin3d contain data for only one-eighth of the final 3D dendrite picture that is generated. The first operation on this raw data is to mirror it along all three axes, one at a time. For example, in a 500^3 simulation this will produce a 1000^3 volume of data from each snapshot file. The initial seed for the dendrite is therefore located at the center of the final dendrite picture. Due to limitations in the visualization software, the 500^3 snapshots can only be mirrored along two axes, so the image displayed in Fig. 10 is $1000 \times 1000 \times 500$ grid points. As exemplified in Fig. 10, the exterior of this simulated dendrite shows a large amount of detail faithful to the dendrite observed in nature.

Additional insight into the process of dendritic growth can be obtained by viewing slices through these dendrites. This can highlight some of the smaller features of the dendrites as well as display some of the inner structure. Examples of these slices are shown in Figs. 11 and 12. This is another visualization that is produced from the snapshot files using either IDL and Bourne shell scripts or, occasionally, AVS (Application Visualization System,

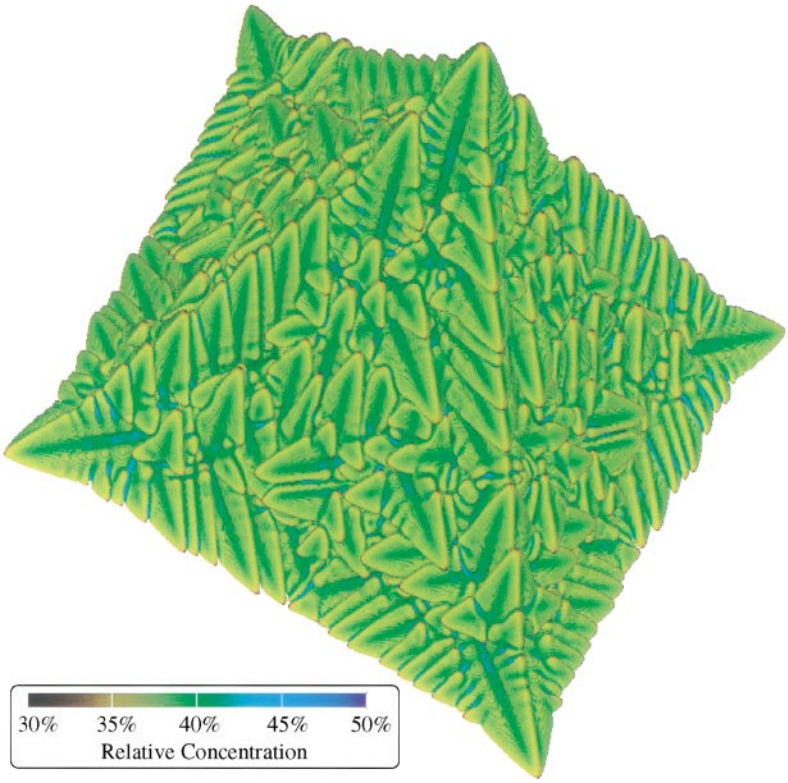

FIG. 10. A simulated dendrite computed over a finite-difference grid of size 500 × 500 × 500 and mirrored along the x and y axes. The surface of the dendrite is colored according to the relative concentration of the materials at the surface of the dendrite.

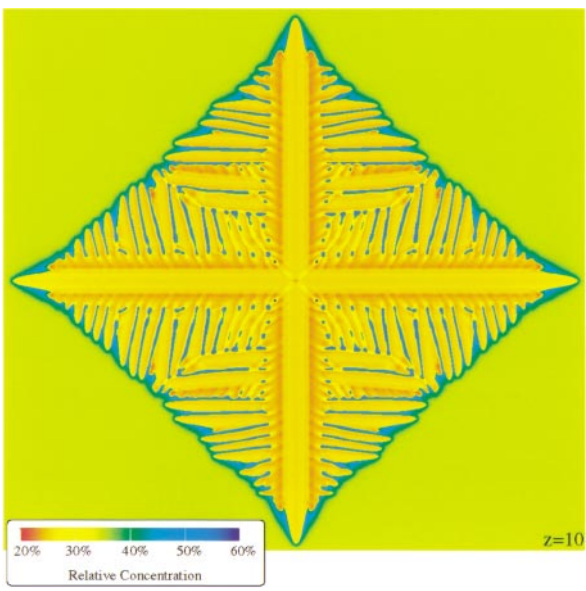

FIG. 11. A 2D slice near the base of the dendrite shown in Fig. 10. The z value given on this image indicates the exact grid position of the slice along the z axis ($z = 0$ at the base of the image in Fig. 10).

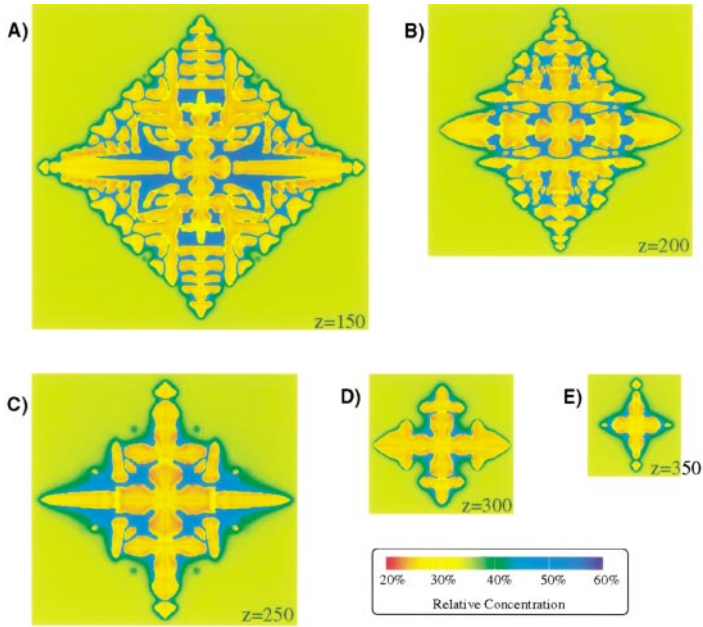

FIG. 12. Images labeled A through E are slices through the dendride shown in Fig. 10 and show details of the internal structure of the dendrite. Starting at A, each image is a slice taken closer to the advancing tip of the dendrite. The z values give the exact location of each slice along the z axis. The scale of these images is the same as the image shown in Fig. 11; however the surrounding areas have been cropped to conserve space.

FIG. 13. A 3D dendrite, rendered for use in an immersive visualization environment, using glyphs and semitransparent colors. In this image the snapshot data from the simulator has been mirrored along all three axes, giving a symmetric six-pointed star structure.

from Advanced Visualizations Systems, Inc.). Just like the full dendrites, these slices are produced as either still shots or in animations made from series of images.

We are currently experimenting with 3D viewing of these images using a high performance visualization machine, a 14-processor SGI Onyx2, and more fully immersive environments such as the VT-CAVE [35]. Some of this experimental work uses a new technique, which replaces the direct calculation of the isosurface, for rendering the dendrite from the raw snapshot data. These visualization experiments, described briefly below, display the entire volume of data but highlight the surface of interest by manipulating the color-mapping tables and setting the alpha (transparency) value for the data points. This gives a result similar to computing an isosurface without the computational- and memory-intensive isosurface calculations.

The SGI Onyx2 systems have high-performance hardware that can provide interactive viewing for large amounts of polygonal data. We have developed a visualization procedure that converts the 3D snapshot data into a polygonal data set that can take advantage of this hardware acceleration. Each data point within the dendrite, i.e., with a phase of 128 or less, is represented by a glyph of three planar quadrilaterals oriented in each of the three orthogonal planes (xy, xz, yz). The size of these glyphs correspond to the 3D grid voxel size, where a voxel is the term used for a 3D pixel or volume element. A semitransparent color value, computed as a function of concentration, is assigned to the glyph. A full-color scale ranging from black to white represents low to high areas of concentration. The speed of the interactive display is determined by the number of glyphs (polygons) used to form the dendrite. As previously stated, phase values in the range of 0–128 are inside the dendrite. Interactivity can be increased by restricting the range of the values selected for glyphs. For example, Fig. 13 uses glyphs for phase values from 28 to 128. However, the trade-off for increasing interactivity is a more sparse representation of the dendrite. Using standard SGI software, OpenGL Performer, this polygonal representation is easily displayed. The semitransparent colors allow a certain amount of internal structure to be revealed and the additive effects of the semitransparent colors produce an isosurface approximation; however, the quality of the image does not translate well to the printed page. A series of polygonal representations from the simulator snapshots are cycled, producing a 3D animation of dendrite growth that can be interactively viewed. Most of the currently available immersive virtual reality (IVR) systems are based on OpenGL Performer. Thus, utilizing this format immediately allows the dendrite growth animation to be placed in an IVR environment for enhanced insight.

6. FUTURE DIRECTIONS

This work has developed the infrastructure for the simulation of dendritic growth in three dimensions. This infrastructure will allow investigations into various properties of dendrites as well as a variety of additional solidification phenomena. Currently, only the most basic of test simulations have been completed with bin3d. The simulation code as it currently exists is capable of supporting many more experiments. Indeed, areas of interest to us include coarsening of spherical particles (surface tension phenomena), coalescence of dendrite arms, and thin-film dendritic growth. However, there are many more possible investigations (even for such a simple model system as a binary alloy) than any single research group could hope to perform.

In addition, as has been have emphasized, a simulation over a 1000^3 volume is desired so that sufficient detail of the dendritic structure can be observed. Previous simulations in two

dimensions were completed over 1000^2 grids so this would allow more direct comparison with results from those 2D simulations.

We also intend on extending the capabilities of the algorithms used in this application. For example, this application performs the same set of computations at all grid points, even though the phase field equation is trivial except in the interfacial region. Reducing the computational load by identifying and omitting those computations could at least halve the simulation time. This is a form of dynamic grid management in which areas of the volume that are not of interest are computed over a coarser grid (see Provatas *et al*. [36] for a discussion of using adaptive finite elements to solve the phase-field equations in 2D, and see Jeong *et al*. [26] for a discussion of this technique in used in a 3D simulation).

As machines with more processors and larger main memories become available, it will become possible to directly compute the entire dendrite instead of computing only the corner (octant) and then mirroring it to produce the final image. Since this project began, our computing facility has been upgraded from a 32-CPU IBM SP to an 80-CPU IBM SP and is in the process of upgrading from a 16-node Beowulf cluster of 333-MHz Intel Pentium II CPUs to a 128-node Beowulf cluster of 750-MHz Intel Pentium III CPUs, each with 1 GB of main memory. With this level of computational hardware available we are now able to complete simulations over grids of size 1000^3.

Similarly, the integration of the visualization process with the simulation, enabling a more interactive execution of the simulator, is being considered. For small volumes the wall-clock time between snapshots can be synchronized with the time required to display the rendered snapshots. An alternative method would be to stream the snapshot data to the visualizer as fast as it is generated with the visualizer processing snapshots at its own speed. Snapshots that arrive while the visualizer is busy processing can simply be discarded.

The available commercial data visualization packages are not capable of handling the amount of data this simulation produces. In each visualization application that has been tried, the 32-bit memory addressing used in the program has limited the volume size for which isosurfaces can be successfully computed. We are currently investigating the modification of OpenDX (Data Explorer), a data visualization package that was recently released in open source by IBM [37]. We are working to produce a version of the OpenDX libraries that uses 64-bit addressing so that all of the memory available can be utilized.

7. CONCLUSION

We have demonstrated the feasibility of simulating the growth of dendrites in three dimensions using a simple finite-difference implementation of the phase-field method. We have also begun to succeed in visualizing the resulting raw snapshots using various techniques and available visualization systems. Although the algorithms used to implement the phase-field method in parallel are relatively unsophisticated, given the rapidly increasing computational power of Beowulf clusters and other parallel machines, we will soon have the ability to regularly produce highly detailed simulations using the available bin3d simulator. Additionally, the dynamic load balancing provided by DPARLIB enhances portability by ensuring efficient execution in a wide variety of execution environments, including heterogeneous clusters of machines.

The code developed herein is generic, modular, and portable, allowing for application to the variety of partial differential equations which are found in the physical sciences.

These methods could be applied to phenomena from astrophysics to the quantum realm. By extending the phase-field algorithms implemented in 2D to 3D and using MPI and DPARLIB, and through the use of high-performance visualization hardware, we now have the ability to explore the phenomena of dendritic growth in three dimensions through high-resolution simulations.

Further work on this project through algorithm development should yield a more interactive simulator (at lower resolution) as well as improved performance, with respect to both space and time requirements, on higher resolution simulations.

ACKNOWLEDGMENTS

We thank Judith Devaney for supporting this project and for providing an environment in which this work could progress, William Boettinger for his advice and many helpful discussions, and Steven Satterfield for his assistance in developing specialized visualization techniques.

REFERENCES

1. T. F. Bower, H. D. Brody, and M. C. Flemings, *Trans. Met. Soc. AIME* **43**, 624 (1966).
2. J. D. Hunt, in *Solidification and Casting of Metals* (Metals Society, Amsterdam, 1979), p. 3.
3. T. Z. Kattamis, J. C. Couglin, and M. C. Flemings, *Trans. Met. Soc. AIME* **239**, 1504 (1967).
4. U. Feurer and R. Wunderlin, *DGM Fachber* **38** (1977).
5. D. H. Kirkwood, *Mater. Sci. Eng.* **73**, L1 (1985).
6. A. Mortensen, *Metall. Trans.* **22A**, 569 (1991).
7. V. R. Voller and S. Sundarraj, in *Moldeling of Casting, Welding and Advanced Solidification Processes—V*, edited by T. S. Piwonka, V. Voller, and L. Katgerman (TMS, Warrendale, PA, 1993), p. 251.
8. H. D. Brody and M. C. Flemings, *Trans. Met. Soc. AIME* **236**, 615 (1966).
9. T. P. Battle and R. D. Pelke, *Met. Trans.* **21A**, 357 (1990).
10. S. Kobayashi, *J. Cryst. Growth* **88**, 87 (1988).
11. C. Beckermann, H. J. Diepers, I. Steinbach, A. Karma, and X. Tong, *J. Comp. Phys.* **154**, 468 (1999).
12. W. Kurz and D. J. Fisher, *Fundamentals of Solidification* (Trans Tech Publishing, Zürich, Switzerland, 1989).
13. S. J. Osher and G. Tryggvason, *J. Comp. Phys.* **169**, 249 (2001).
14. G. Tryggvason, B. Brunner, A. Esmaeeli, D. Juric, N. Al-Rawahi, W. Tauber, J. Han, S. Nas, and Y. J. Jan, *J. Comp. Phys.* **169**, 708 (2001).
15. J. A. Warren and W. J. Boettinger, *Acta Met. Mater.* **43**, 689 (1995).
16. J. S. Langer, in *Directions in Condensed Matter Physics*, edited by G. Grinstein and G. Mazenko (World Scientific, Philadelphia, 1986), p. 164.
17. G. Caginalp, in *Applications of Field Theory to Statistical Mechanics*, edited by L. Garrido (Springer-Verlag, Berlin, 1985), p. 216.
18. J. B. Collins and H. Levine, *Phys. Rev. B* **31**, 6119 (1985).
19. A. A. Wheeler, W. J. Boettinger, and G. B. McFadden, *Phys. Rev. A* **45**, 7424 (1992).
20. A. A. Wheeler, W. J. Boettinger, and G. B. McFadden, *Phys. Rev. E* **47**, 1893 (1993).
21. R. Kobayashi, *Physica D* **63**, 410 (1993).
22. A. Karma and W.-J. Rappel, *Phys. Rev. E* **53**, 3017 (1996).
23. A. Karma and W.-J. Rappel, *J. Cryst. Growth* **174**, 54 (1997).
24. A. Karma and W.-J. Rappel, *Phys. Rev. E* **57** (4), 4323 (1998).
25. A. Karma, Y. H. Lee, and M. Plapp, *Phys. Rev. E* **61**, 3996 (2000).
26. J.-H. Jeong, N. Goldenfield, and J. A. Dantzig, *Phys. Rev. E* **64**, 041602 (2001).
27. A. Schmidt, *J. Comp. Phys.* **125**, 293 (1996).

28. S. K. Aliabadi and T. E. Tezduyar, *Int. J. Numer. Methods Fluids* **21** (10), 783 (1995).
29. H. Zhou and J. J. Derby, *Int. J. Numer. Methods Fluids* **36** (7), 841 (2001).
30. A. A. Wheeler and G. B. McFadden, *Eur. J. Appl. Math.* **7**, 367 (1996).
31. W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes in C* (Cambridge University Press, Cambridge, UK, 1997), ISBN 0-521-43108-5.
32. Message Passing Interface Forum, *Int. J. Supercomput. Appl. High Perform. Comput.* **8** (3/4), 159 (1994), special issue on MPI.
33. W. George, *C-DParLib Reference Manual*, Natl. Inst. Stand. Technol. Interagency Report NISTIR (NIST, 2000).
34. W. George, *C-DParLib User's Guide*, Natl. Inst. Stand. Technol. Interagency Report NISTIR (NIST, 2000) .
35. *The VT-CAVE*, available at www.cave.vt.edu.
36. N. Provatas, N. Goldenfeld, and J. Dantzig, *J. Comp. Phys.* **148**, 265 (1999).
37. *OpenDX: The Open Source Software Project Based on IBM's Visualization Data Explorer*, available at www.opendx.org.