Phase field model for three-dimensional dendritic growth with fluid flow

Jun-Ho Jeong,[1] Nigel Goldenfeld,[2] and Jonathan A. Dantzig[1]
[1]*Department of Mechanical and Industrial Engineering, University of Illinois at Urbana-Champaign, Urbana, Illinois 61801*
[2]*Department of Physics, University of Illinois at Urbana-Champaign, Urbana, Illinois 61801*
(Received 3 May 2001; published 20 September 2001)

We study the effect of fluid flow on three-dimensional (3D) dendrite growth using a phase-field model on an adaptive finite-element grid. In order to simulate 3D fluid flow, we use an averaging method for the flow problem coupled to the phase-field method and the semi-implicit approximated projection method (SIAPM). We describe a parallel implementation for the algorithm, using the CHARM++ FEM framework, and demonstrate its efficiency. We introduce an improved method for extracting dendrite tip position and tip radius, facilitating accurate comparison to theory. We benchmark our results for 2D dendrite growth with solvability theory and previous results, finding them to be in good agreement. The physics of dendritic growth with fluid flow in three dimensions is very different from that in two dimensions, and we discuss the origin of this behavior.

DOI: 10.1103/PhysRevE.64.041602 PACS number(s): 81.10.Aj, 05.70.Ln, 81.30.Fb, 64.70.Dv

I. INTRODUCTION

Dendrites are the basic microstructural form for most crystalline materials. They express the underlying crystalline symmetry, as well as the growth conditions which existed when the dendrite was formed. Dendrites may form from the vapor phase (e.g., snowflakes), from solution (e.g., polymer crystals), or by solidification from the melt (e.g., most metals). In this work, we focus our attention on growth from the melt, which is important in many materials processing applications. Dendritic growth produces local compositional variations which determine the macroscopic properties of the material. These features persist through subsequent processing, and it is therefore important to understand the mechanisms by which the microstructural pattern is selected.

Beginning with the morphological stability theory of Mullins and Sekerka [1], the dynamics of pattern selection is now reasonably well understood. For a review of the theory, see Langer [2] and Kessler *et al.* [3]. There is now a consensus that the so-called ''microscopic solvability'' theory [4,5] agrees very well with numerical calculations [6,7]. These comparisons were performed in two dimensions (2D), where accurate time-dependent simulations of dendrite growth are tractable using phase-field methods and also level set techniques [8]. In this paper, we focus on the phase-field method; for a description of the level set method as applied to solidification problems, see Chen *et al.* [9].

It is known that fluid flow during solidification dramatically alters the solidification structure [10]. Using typical values for the local flow velocity, material properties and process parameters, one can anticipate that the interdendritic flow is dominated by viscous forces [$\mathrm{Re}\sim\mathcal{O}(0.1)$], but that the diffusion fields for temperature and solute are dominated by advective effects [$\mathrm{Pe}\sim\mathcal{O}(10-100)$]. The presence of the flow admits the possibility of instabilities due to the flow itself, in addition to the morphological instabilities normally found in crystal growth.

The effect of fluid flow on dendritic growth is the object of this research. This is an inherently three-dimensional phenomenon, as can be seen in the schematic drawing in Fig. 1. A pair of dendrites is shown growing into a flow which is nominally perpendicular to their primary growth direction. The mechanism by which the flow alters the growth pattern is the transport of solute from the leading edge to the trailing edge of the dendrite [11]. In 2D, this occurs by the flow going up and over the dendrite. However, in 3D, it is much easier for this transport to take place by having the flow go *around* the dendrite. Thus, in order to correctly model this phenomenon, we must do 3D simulations.

An example of a dendrite computed with fluid flow present is given in Fig. 2. This calculation was done using the methods we describe later in this paper, but we introduce it here to provide a context for describing the physical problem. The shape of the dendrite is complex, and it evolves during the computation. The far-field flow on the left-hand side of the dendrite is a uniform velocity, directed parallel to a preferred crystalline growth direction. The figure shows stream traces for the flow over the dendrite. Notice that growth is enhanced in the directions counter to the flow. Side branches also appear preferentially on the leading edge of the transverse arms, and the trailing arm is completely suppressed. More will be said about this in Sec. VI.

The surface of the dendrite represents the interface between the solid and liquid, and there are two boundary conditions which must be satisfied on this interface. First there is the condition of local thermodynamic equilibrium,

$$T = T_m - \Gamma(\boldsymbol{n})\kappa - \beta(\boldsymbol{n})\boldsymbol{V}\cdot\boldsymbol{n}, \tag{1}$$

where T is the interface temperature, T_m is the equilibrium melting point of the pure material (with a flat interface), Γ is

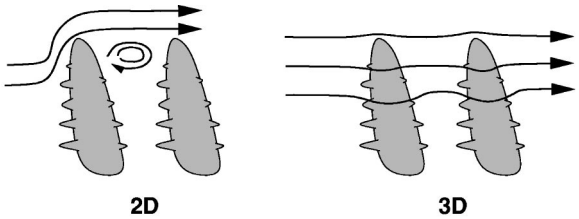

FIG. 1. Schematic drawing of the flow over dendrites growing perpendicular to a superimposed flow, comparing 2D and 3D phenomena.

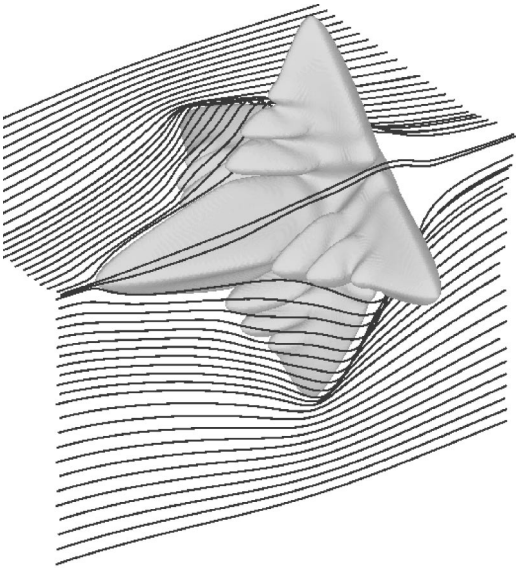

FIG. 2. Computed streamtraces for flow over a growing isolated dendrite.

the ratio of the surface energy to the entropy of fusion, κ is the Gaussian curvature of the surface, β is a kinetic coefficient, $\boldsymbol{V}$ is the interface velocity, and $\boldsymbol{n}$ is the normal vector to the interface. The dependence of Γ and β on $\boldsymbol{n}$ introduces crystalline anisotropy into the problem. There is also an energy balance for the motion of the interface,

$$k_s \boldsymbol{\nabla} T_s \cdot \boldsymbol{n} - k_\ell \boldsymbol{\nabla} T_\ell \cdot \boldsymbol{n} = \rho_s L_f \boldsymbol{V} \cdot \boldsymbol{n}, \tag{2}$$

where the subscripts s and ℓ refer to the solid and liquid phases, respectively, k is the thermal conductivity, ρ_s is the solid density, and L_f is the latent heat of fusion. We refer to the solidification problem where these boundary conditions are explicitly satisfied as the *sharp interface problem.*

One of the computational issues in this problem is that the position of the interface is *a priori* unknown, and therefore enforcement of the boundary conditions is difficult. Rather than track the phase boundary explicitly, we introduce a continuous order parameter $\phi \in [-1,1]$, where $\phi = -1$ corresponds to the liquid, $\phi = 1$ corresponds to the solid, and the level set $\phi = 0$ is identified as the interface. Details of the method and the selection of parameters to ensure convergence of the phase-field model to the sharp interface problem described above can be found in Refs. [6,12].

The phase-field method introduces a finite width W_0 for the interface, which must be kept small if the calculations are to be meaningful. In particular, we require that W_0 must be of the order of the capillary length d_0, which is a material property, typically ranging from 1×10^{-9} to 1×10^{-8} m. At the same time, the computations must resolve the diffusion field surrounding the dendrite, and this can be of order 1×10^{-4} m for physically relevant growth conditions. Finally, the grid spacing at the interface must be on the order of W_0 to preserve contact with the sharp interface model. Thus, the spatial grid must resolve at least five orders of magnitude. Uniform grid approaches are clearly limited to two dimensions, and even then they are very computationally intensive.

In this paper, we resolve this difficulty by solving the phase-field equations on an adaptive finite-element mesh. The methods are discussed in greater detail in the following sections; just a sketch is provided here. The 3D domain is meshed with hexahedral elements, stored in an octree data structure. Local error estimators are used to selectively refine or coarsen the mesh, and this permits tracking of the interface as well as resolution of gradients in the other fields. There are six degrees of freedom at each node (three velocities, pressure, temperature, and ϕ), and a typical computation, such as the one shown in Fig. 2 has up to 300 000 nodes and thus well over 1×10^6 unknowns. While adaptive grid methods make the computations feasible, the full 3D problem remains a formidable challenge.

The outline of this paper is as follows: In Sec. II we describe numerical methods. In Sec. III we present a detailed description of the 3D adaptive grid refinement algorithm. Sec. IV describes a parallel implementation using CHARM ++ and presents the results of our effort to accelerate our code. In Sec. V we explain the difficulties generically encountered in measuring interface velocities, even with phase-field methods, and present accurate schemes to calculate the interface velocity. In Sec. VI, we present results for 3D dendrite growth without and with fluid flow. In Sec. VII, we conclude and discuss our results.

II. NUMERICAL METHODS

The numerical implementation of the adaptive grid technique applied to the phase-field model without convection has been described in detail elsewhere [12], so we focus here on the fluid flow problem. Beckermann *et al.* [13] introduced an averaging method for the flow problem coupled to the phase field, and we follow that approach here. Beckermann *et al.* performed only 2D calculations, but their methods extend naturally to 3D. The phase average Ψ_k of a variable Ψ for phase k over volume ΔV, is defined as

$$\Psi_k = \frac{2 \int_{\Delta V} X_k \Psi \, dV}{1 - \phi}, \tag{3}$$

where $X_k \in [0,1]$ is an existence function. The phase averages of the velocity and pressure are used for deriving the mixture continuity equation, averaged liquid momentum equation, and averaged energy conservation equation. The formulation ensures that the fluid velocity is extinguished in the solid, and further that the shear stress at the liquid-solid interface is handled correctly.

The governing equations for simulating dendritic growth with fluid flow are the mixture continuity equation, averaged liquid momentum equation, averaged energy conservation equation, and phase field equation as follows:

(i) The mixture continuity equation

$$\boldsymbol{\nabla} \cdot \left[\frac{1 - \phi}{2} \boldsymbol{u} \right] = 0, \tag{4}$$

where $\boldsymbol{u}$ is the velocity vector.

(ii) The averaged momentum equation

$$\frac{\partial}{\partial t}\left[\left(\frac{1-\phi}{2}\right)\boldsymbol{u}\right]+\boldsymbol{u}\cdot\boldsymbol{\nabla}\left[\left(\frac{1-\phi}{2}\right)\boldsymbol{u}\right]+\left(\frac{1-\phi}{2}\right)\frac{\boldsymbol{\nabla}p}{\rho_0}$$
$$=\nu\nabla^2\left[\left(\frac{1-\phi}{2}\right)\boldsymbol{u}\right]-\nu\frac{h(1-\phi^2)(1+\phi)}{8\delta^2}\boldsymbol{u}, \tag{5}$$

where t is time, p is pressure, ρ_0 is the (constant) density, ν is the kinematic viscosity, $\delta=W_0/\sqrt{2}$ is the characteristic interface width, and h is a constant ($=2.757$) which ensures that the interface shear stress is correct for a simple shear flow (see Beckermann *et al.* [13]).

(iii) We write the averaged energy conservation equation in terms of a dimensionless temperature $\theta=c_p(T-T_m)/L_f$, scaled by the specific heat c_p and latent heat of fusion L_f:

$$\frac{\partial\theta}{\partial t}+\left(\frac{1-\phi}{2}\right)\boldsymbol{u}\cdot\boldsymbol{\nabla}\theta=D\nabla^2\theta+\frac{1}{2}\frac{\partial\phi}{\partial t}, \tag{6}$$

where $D=\alpha\tau_0/W_0^2$ in which α is the thermal diffusivity and τ_0 is a time characterizing atomic motion in the interface.

(iv) The 3D phase-field evolution equation is given by

$$\tau(\boldsymbol{n})\frac{\partial\phi}{\partial t}=[\phi-\lambda\theta(1-\phi^2)](1-\phi^2)+\boldsymbol{\nabla}\cdot[W(\mathbf{n})^2\boldsymbol{\nabla}\phi]$$
$$+\partial_x\left(|\nabla\phi|^2W(\mathbf{n})\frac{\partial W(\mathbf{n})}{\partial(\partial_x\phi)}\right)$$
$$+\partial_y\left(|\nabla\phi|^2W(\mathbf{n})\frac{\partial W(\mathbf{n})}{\partial(\partial_y\phi)}\right)$$
$$+\partial_z\left(|\nabla\phi|^2W(\mathbf{n})\frac{\partial W(\mathbf{n})}{\partial(\partial_z\phi)}\right), \tag{7}$$

where λ is a dimensionless constant that controls the tilt of the double-well potential which forces ϕ to the attractors at ±1. Anisotropy is included in this equation by writing the interface mobility τ and width W as functions of the local normal vector $\boldsymbol{n}$. Following Karma and Rappel [6], we choose

$$W(\boldsymbol{n})=W_0a_s(\boldsymbol{n}),\quad \tau(\boldsymbol{n})=\tau_0a_s^2(\boldsymbol{n}), \tag{8}$$

with

$$a_s(\boldsymbol{n})=(1-3\epsilon_4)\left[1+\frac{4\epsilon_4}{1-3\epsilon_4}\frac{(\partial_x\phi)^4+(\partial_y\phi)^4+(\partial_z\phi)^4}{|\nabla\phi|^4}\right]. \tag{9}$$

The constant parameter ϵ_4 fixes the strength of the anisotropy in the interface energy.

We solve the 3D flow equations using the semi-implicit approximate projection method (SIAPM) as developed by Gresho [14]. SIAPM is a predictor-corrector method which can solve Eq. (4) and Eq. (5) effectively, especially for large 3D problems, because it uses relatively small amounts of memory. The velocity degrees of freedom are solved in a segregated form, and the pressure is updated using a projection method. For a detailed discussion of the algorithm, the reader is referred to the original paper [14], and we present only an operational description of the algorithm here. The algorithm consists of four steps.

(i) Compute an intermediate velocity $\tilde{\boldsymbol{u}}^{n+1}$ from

$$\left(\frac{1}{\Delta t}\boldsymbol{M}-\frac{1}{2}\boldsymbol{K}+\boldsymbol{F}\right)\tilde{\boldsymbol{u}}_i^{n+1}=\left(\frac{1}{\Delta t}\boldsymbol{M}+\frac{1}{2}\boldsymbol{K}\right)\boldsymbol{u}_i^n-\boldsymbol{A}(\boldsymbol{u}^n)\boldsymbol{u}_i^n-\boldsymbol{G}_i\boldsymbol{p}^n, \tag{10}$$

where $\tilde{\boldsymbol{u}}_i^{n+1}$ is the vector of nodal values of the intermediate velocity component i at time step $n+1$, $\boldsymbol{u}_i^n$ is the corresponding vector at time step n, and $\boldsymbol{p}^n$ is the vector of nodal pressures at timestep n. The coefficient matrices are defined in terms of the velocity shape functions $\boldsymbol{N}$ as follows:

$$\boldsymbol{M}=\int_\Omega\frac{(1-\phi)}{2}\boldsymbol{N}^T\boldsymbol{N}d\Omega, \tag{11}$$

$$\boldsymbol{K}=\int_\Omega\nu\frac{(\phi-1)}{2}\left(\frac{\partial\boldsymbol{N}^T}{\partial\boldsymbol{x}}\frac{\partial\boldsymbol{N}}{\partial\boldsymbol{x}}+\boldsymbol{I}\frac{\partial\boldsymbol{N}^T}{\partial x_k}\frac{\partial\boldsymbol{N}}{\partial x_k}\right)d\Omega, \tag{12}$$

$$\boldsymbol{F}=\int_\Omega\frac{\nu h(1-\phi^2)(1+\phi)}{8\delta^2}\boldsymbol{N}^T\boldsymbol{N}d\Omega, \tag{13}$$

$$\boldsymbol{A}(\boldsymbol{u}^n)=\int_\Omega\frac{(1-\phi)}{2}\boldsymbol{N}^Tu_k^n\frac{\partial\boldsymbol{N}}{\partial x_k}d\Omega, \tag{14}$$

$$\boldsymbol{G}_i=\int_\Omega\frac{(1-\phi)}{2\rho_0}\boldsymbol{N}^T\frac{\partial\boldsymbol{N}}{\partial x_i}d\Omega. \tag{15}$$

(ii) The velocity field found in the first step is generally not divergence free. The next step corrects the pressure to obtain an approximately divergence-free velocity field by solving a Poisson equation for $\Delta\boldsymbol{p}^{n+1}=(\boldsymbol{p}^{n+1}-\boldsymbol{p}^n)$:

$$\boldsymbol{L}\Delta\boldsymbol{p}^{n+1}=-\frac{1}{\Delta t}\boldsymbol{D}(\tilde{\boldsymbol{u}}^{n+1}-\boldsymbol{u}^n), \tag{16}$$

where

$$\boldsymbol{L}=\int_\Omega\frac{(1-\phi)}{2\rho_0}\frac{\partial\boldsymbol{N}^T}{\partial x_k}\frac{\partial\boldsymbol{N}}{\partial x_k}d\Omega, \tag{17}$$

$$\boldsymbol{D}=\int_\Omega\frac{(1-\phi)}{2}\boldsymbol{N}^T\frac{\partial\boldsymbol{N}}{\partial\boldsymbol{x}}d\Omega. \tag{18}$$

$\boldsymbol{p}^{n+1}$ is then updated from

$$\boldsymbol{p}^{n+1}=\boldsymbol{p}^n+\Delta\boldsymbol{p}^{n+1}. \tag{19}$$

(iii) Finally, the projected velocity $\boldsymbol{u}^{n+1}$ is computed in a corrector step by solving

$$\boldsymbol{u}^{n+1}=\tilde{\boldsymbol{u}}^{n+1}-\Delta t\boldsymbol{M}_L^{-1}\boldsymbol{G}\Delta\boldsymbol{p}^{n+1}, \tag{20}$$

where

```
type element
  integer                          :: num_element            !! Element number
  integer                          :: level                  !! Refinement level
  integer                          :: lneigh                 !! Number of neighbor elements
  type(connectivity), pointer      :: connect                !! Pointer of connectivity
  type(connectivity), pointer      :: connect_mid            !! Pointer of connectivity for disconnected nodes
  integer                          :: midindex(6)            !! Index to check for discontinuous nodes
  type(neighbor_elements), pointer:: neighbor                !! Pointer to neighbor elements
  integer                          :: num_parent(LimitLevel) !! Parent element numbers
  integer                          :: num_history(LimitLevel)!! Time step number
  integer                          :: merge                  !! Index to check if the element should be merged
  real*8                           :: error                  !! Error estimator value at time step n
  integer                          :: nver                   !! Vertex node number for disconnected edge node
  integer                          :: ntype                  !! Element type (liquid/solid/interface)
                                                             !! If phi = -1         , ntype = 1(liquid)
                                                             !! If phi = 1          , ntype = 2(solid)
                                                             !! If -1 < phi < 1     , ntype = 3(interface)
  real*8                           :: pe                     !! Element pressure
  type(element), pointer           :: previous               !! Previous element in linked list
  type(element), pointer           :: next                   !! Next element in linked list
end type element
```

FIG. 3. Element linked data structure for adaptive grid.

$$\boldsymbol{M}_L = \int_\Omega \frac{1-\phi}{2} N d\Omega. \tag{21}$$

The computations are started using an initial velocity field $\boldsymbol{u}_0$ determined from the boundary conditions. In order to obtain a field $\boldsymbol{u}_0$ whose discrete divergence is close to zero, we perform J (typically 10) iterations on the following system (shown in pseudocode):

$$\begin{aligned} &\text{For } I=1 \text{ to } J[\\ &\qquad Lq^I = -Du^{I-1} \\ &\qquad u^I = u^{I-1} - M_L^{-1} G q^I]. \end{aligned} \tag{22}$$

The variable q plays the role of a temporary pressure update, but the actual initial pressure is zero everywhere.

Equations (10) and (16) are solved by the conjugate gradient (CG) method with diagonal preconditioning [15]. SIAPM can calculate the velocity field for large 3D problems much faster than fully implicit time-stepping methods, because convergence is reached for Eq. (10) in a few iterations and the number of degrees of freedom of Eq. (16) is one. The CG iteration for Eq. (16) converges more slowly, typically 50–200 iterations.

The averaged energy equation (6) is also solved using the CG method with diagonal preconditioning. Streamline upwind schemes [16] are employed for the convection terms in Eq. (10) and Eq. (6). The 3D phase-field equation is a nonlinear system. In order to solve the system implicitly, an iterative method such as the Newton-Raphson method is required. We use instead an explicit time-stepping scheme where a linear system is solved. Stable solutions are obtained from the explicit scheme, because the variation of the ϕ field exists only in the interface region and a sufficiently small time increment Δt is used.

III. THREE-DIMENSIONAL ADAPTIVE GRID REFINEMENT ALGORITHM

To resolve the interface, the grid spacing Δx must be smaller than the characteristic interface width W_0, which must in turn be on the order of the capillary length, for the solution of the phase-field model to converge to the sharp interface limit. On the other hand, the system size L required for simulation is determined by the size of the diffusion field ahead of the dendrite. A ratio of $L/\Delta x \sim 10^3 - 10^4$ is typical. For simulation of a single 3D dendrite, this implies that at least 10×10^6 elements must be used in a uniform grid. Such a simulation of 3D dendrite growth with fluid flow would be intractable.

In this problem, there is an important characteristic that the various fields vary most rapidly in the interface region, whose width is much smaller than L. For this reason, adaptive grid refinement techniques can be applied very effectively. The 3D adaptive grid refinement is described in the next few sections.

A. Error estimating procedure

The basis of the code is the element data structure, illustrated in Fig. 3. The structure consists of arrays for element connectivity, neighbors, and also the element pressure.

The grid is locally adapted based on an element-by-element error estimate, with a hybrid scheme using the magnitude of ϕ and the interelement variation of the derivatives of θ. We use ϕ as an indicator to define the specific region in which the finest elements should be distributed. Specially, if an element includes a node where

$$\phi_{min} \leqslant \phi \leqslant \phi_{max}, \tag{23}$$

then the element is divided until its refinement level becomes the maximum level. We control the width of the region with the finest elements through the values of ϕ_{min} and ϕ_{max}. We proceed by defining a grid and then solving a predetermined number of time steps on that grid N_{ref} (typically $N_{ref}=20$

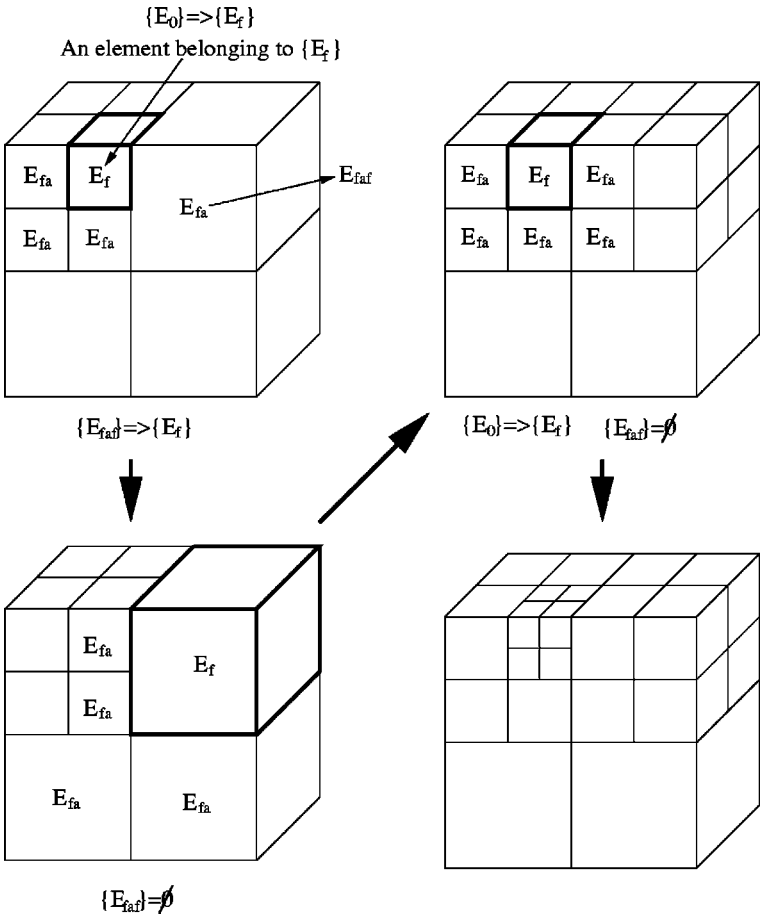

FIG. 4. Dividing sequence for refinement. The arrows indicate the sequence of refinement, as discussed in the text.

-100), and then adapt the grid. We require that the interface remain within the fine grid during the time steps. We found that $\phi_{min}=-0.99$ and $\phi_{max}=0.9$ gave consistent results. The reason for the asymmetry is that the interface moves from the solid region $\phi>0$ to the liquid region $\phi<0$. Outside of the interface region ($\phi<\phi_{min}$ or $\phi>\phi_{max}$), we used an error estimator based on the magnitudes of the derivatives of θ as follows:

$$E_e=\int_{\Omega_e}\nabla\theta\cdot\nabla\theta d\Omega, \tag{24}$$

where Ω_e is the element. The estimated error E_{sub} of a subelement divided from a parent element with error E_p can be calculated from the asymptotic rate of convergence of the finite element approximation as

$$E_{sub}=\left(\frac{h_{sub}}{h_{parent}}\right)^k E_p=0.5^k E_p, \tag{25}$$

where k is the exponent for the asymptotic rate of convergence ($k=2$ for linear elements), and h_{sub} and h_{parent} are the element sizes of the subelement and parent element, respectively. A limit value E_{limit} is calculated from

$$E_{limit}=\gamma\max[(0.5^k)^{(N_{max}-N_e)}E_p]|_{e=1}^{N_T} \tag{26}$$

where γ is a scale coefficient (typically 10), N_T is the total number of elements, N_{max} is maximum refinement level, and N_e is the refinement level of element e. The element is divided until its estimated error becomes smaller than a specified limit value. Once the elements to be refined have been selected, the grid is refined using the procedure described in the next section.

B. Grid refinement procedure

The grid refinement procedure begins by storing the pointers of neighboring elements and the refinement level of each element. Refinement is required in an element whenever the estimated error exceeds the limit value or when the absolute difference between the level of an element and that of its neighbors exceeds 1. The latter is called the single-level rule. Refinement is carried out successively at each refinement level, beginning with the minimum, according to the procedure illustrated in Fig. 4, and described below.

(i) After checking the computed elemental errors, a set $\{E_0\}$ is created which consists of elements in the current refinement level, whose error exceeds the present limit value. These elements will be refined after the neighboring elements which must be refined to satisfy the single-level rule are found and refined.

(ii) A recursive search is then performed to find the outermost elements which need to be refined to satisfy the single-level rule. We do this by first defining a new set $\{E_f\}$

whose initial value is $\{E_0\}$. The neighbor elements to each element in $\{E_f\}$ are then examined, and any neighboring element that does not satisfy the single-level rule is added to a new set $\{E_{faf}\}$. If $\{E_{faf}\}$ is not a null set, then $\{E_f\}$ is replaced by $\{E_{faf}\}$. This procedure is repeated until $\{E_{faf}\}$ becomes a null set. At this point, the set $\{E_f\}$ contains the outermost elements which need to be refined to satisfy the single-level rule, and these elements are then divided into eight subelements by bisection of each face. (See Fig. 4.) At this point, the set $\{E_f\}$ is again set equal to $\{E_0\}$, and the search begins anew. This process is repeated until the search for elements to fill set $\{E_{faf}\}$ yields a null set. Then, the elements in set $\{E_0\}$ are refined, and the refinement process is completed in the next step.

(iii) The nodal coordinates, element connectivities, neighbor arrays, parent arrays, nodal refinement levels, etc., are updated. The neighbor array is the array to store the element numbers of neighboring elements. The parent array is the array to store the element number of the parent elements for the coarsening procedure, described in the next section.

The recursive search for elements violating the single-level rule in the second step above may seem inefficient, because it performs multiple searches, finding the same elements. However, this procedure makes it possible to perform the searches with each element knowing only about its nearest neighbors. This gives greater efficiency in the computational phase, because it limits the memory allocation required within the element data structure.

C. Grid unrefinement procedure

The unrefinement procedure is accomplished through a loop in which the refinement level is decreased incrementally from the maximum level to the minimum level.

(i) A set $\{\{E_{sf}\}\}$ is created, containing all elements which are eligible for unrefinement based on the value of the error estimator. Each element of $\{\{E_{sf}\}\}$ is a subset $\{E_{sf}\}$, consisting of eight elements to be merged into one. If an element whose refinement level equals the current level of the unrefinement loop has smaller estimated error than a prescribed limit value, then the element is placed in a temporary set $\{E_{temp}\}$. The elements belongs to $\{E_{temp}\}$ are sorted into the subsets $\{E_{tsf}\}$ according to their parent element.

(ii) If the number of elements belonging to any subset $\{E_{tsf}\}$ is eight and an element created from the eight elements satisfies the single level rule, then $\{E_{tsf}\}$ becomes the subset $\{E_{sf}\}$.

(iii) The eight elements belonging to each $\{E_{sf}\}$ are then merged into the parent element. As we did for grid refinement, the nodal coordinates, elemental connectivities, neighbor array, parent array, and nodal refinement levels are updated.

D. Treatment of disconnected nodes

The adapted grid will have so-called disconnected nodes which appear whenever an element has a neighbor whose refinement level differs by one, as illustrated in Fig. 5. An element with connectivity [6, 3, 12, 4, 1, 2, 13, 5] contacts two neighbor elements of lower refinement level.

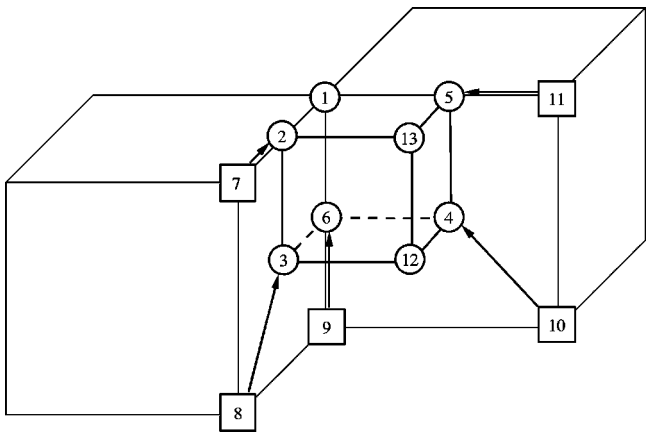

FIG. 5. Configurations of disconnected sides and nodes.

To ensure continuity of the solution between elements, the following constraints are enforced for each degree of freedom, here represented by the symbol v_i:

(i) disconnected edge midnodes

$$v_2=\frac{v_1+v_7}{2},\quad v_5=\frac{v_1+v_{11}}{2},\quad v_6=\frac{v_1+v_9}{2}, \tag{27}$$

(ii) disconnected face midnodes

$$v_3=\frac{v_1+v_7+v_8+v_9}{4},\quad v_4=\frac{v_1+v_{11}+v_{10}+v_9}{4}. \tag{28}$$

After applying the constraints, the original element connectivity is modified to be [8, 12, 10, 9, 7, 13, 11, 1], and the element shape functions are changed appropriately. This change is made to facilitate domain decomposition in the parallel implementation, described next.

IV. PARALLELIZATION BY CHARM++

CHARM++ is a message-passing parallel runtime system for machines from clusters of workstations to tightly coupled symmetric multiprocessing machines [17]. A parallel FEM code can be written in FORTRAN90 using interface routines from the CHARM++ FEM framework [18]. The FEM framework program consists of three kinds of subroutines. Service routines such as ''INIT'' and ''FINALIZE'' do I/O, startup, and shutdown tasks, and are called only on the first processor. The main work is done using ''DRIVER'' routines, replicated on every processor. In our parallel implementation, the adapted grid is newly regenerated every 20–100 time steps. After each regeneration, the adapted grid is partitioned into chunks assigned to each processor using METIS [19]. This function is also handled through the interface to CHARM++.

In the DRIVER routine on each processor, the temperature and velocity fields are calculated using the preconditioned CG method in an the element-by-element scheme, and the ϕ field is solved using an explicit time-stepping scheme, as described earlier. All calculations for the CG method are accomplished through the products of the elemental stiffness matrixes and local solution vectors. This creates additive contributions for the residual vector for each degree of free-

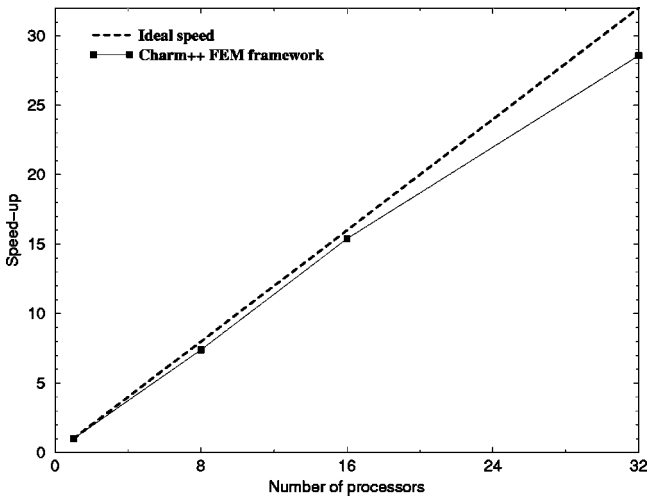

FIG. 6. Speed-up for CHARM++ FEM framework.

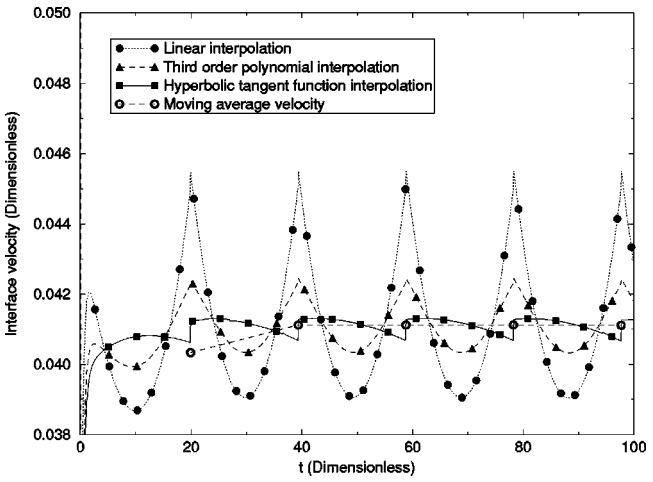

FIG. 7. Interface velocity vs time for the 1D test problem.

dom. Some ''shared nodes'' appear on more than one processor. The array is calculated for each chunk, and the values of the field variables for all shared nodes are combined with the other chunks by using the CHARM++ function ''FEM_UPDATE_FIELD.'' For calculating the inner product of arrays and finding the maximum error values, each nodal value is combined across all chunks and the shared nodes ares not double-counted by using the CHARM++ function ''FEM_REDUCE_FIELD.'' After the computations in the DRIVER routine on all processors are completed, the calculated data are transfered from DRIVER to INIT. The grid is adapted, then repartitioned. These procedures are repeated until the simulation is finished.

In order to rate the parallel performance of the code, we compute the ratio SP, defined as

$$\mathrm{SP}=\frac{(\text{run time on one processor})}{(\text{run time on } n \text{ processors})}. \tag{29}$$

A sample result is shown in Fig. 6, where we used a grid with 296 636 elements and 349 704 nodes computing over 20 time steps. The value SP=28.8 for 32 processors is typical for our code and shows that the code has been effectively parallelized.

V. TIP VELOCITY MEASUREMENT

The steady-state tip velocity is a convenient measure to compare the computational results to dendritic growth theories. In the phase-field method, tip velocity is inferred from the temporal evolution of $\phi=0$ along the primary growth axis. Karma and Rappel [6] were able to derive interface velocities from the numerical solution by using a third-order polynomial to interpolate ϕ, using neighbor and next-neighbor grid point values. Using a 1D analog problem as a basis, they showed that this scheme gives an interface velocity with a small amplitude oscillation about the mean, due to the interpolation of the moving front on the fixed grid.

This multipoint scheme is somewhat problematic when adaptive gridding is used, because the selection of points for interpolation is incompatible with our data structure. We developed a new scheme using a hyperbolic tangent function to perform the interpolation. This approach provides equivalent control of the tip position oscillation, yet it requires only two grid points, corresponding to the nodes in the element which contains the interface ($\phi=0$). Let us denote the x coordinates and ϕ values of these two points as (x_1,ϕ_1) and (x_2,ϕ_2). We interpolate ϕ along the axis as

$$\phi(x;x_c,W)=-\tanh\left(\frac{x-x_c}{W}\right), \tag{30}$$

where x_c and W are fitted parameters corresponding to the zero crossing of ϕ and the interface width, respectively. The parameters x_c and W are determined as follows:

$$W=\frac{2(x_1-x_2)}{\ln[(1-\phi_1)(1+\phi_2)]-\ln[(1+\phi_1)(1-\phi_2)]}, \tag{31}$$

$$x_c=\frac{W}{2}\ln\frac{1+\phi_1}{1-\phi_1}+x_1. \tag{32}$$

The interface velocity is then computed by a finite difference in time between successive interface locations.

To demonstrate the scheme, we examined the same 1D test problem considered by Karma and Rappel [6], viz.,

$$\tau_0\frac{\partial\psi}{\partial t}=W_0^2\frac{\partial^2\psi}{\partial x^2}+\psi-\psi^3+\Delta, \tag{33}$$

using $\tau_0=1$, $W_0=1$, $\Delta x=0.8$, and $\Delta=0.02$. This problem has a traveling-wave solution which propagates at velocity $V=0.041$. The computed interface velocity is shown in Fig. 7 for a variety of interpolation schemes. It can be seen that the hyperbolic tangent interpolation scheme has a small oscillation amplitude, slightly smaller than that of the third-order polynomial. The average interface velocity can be eas-

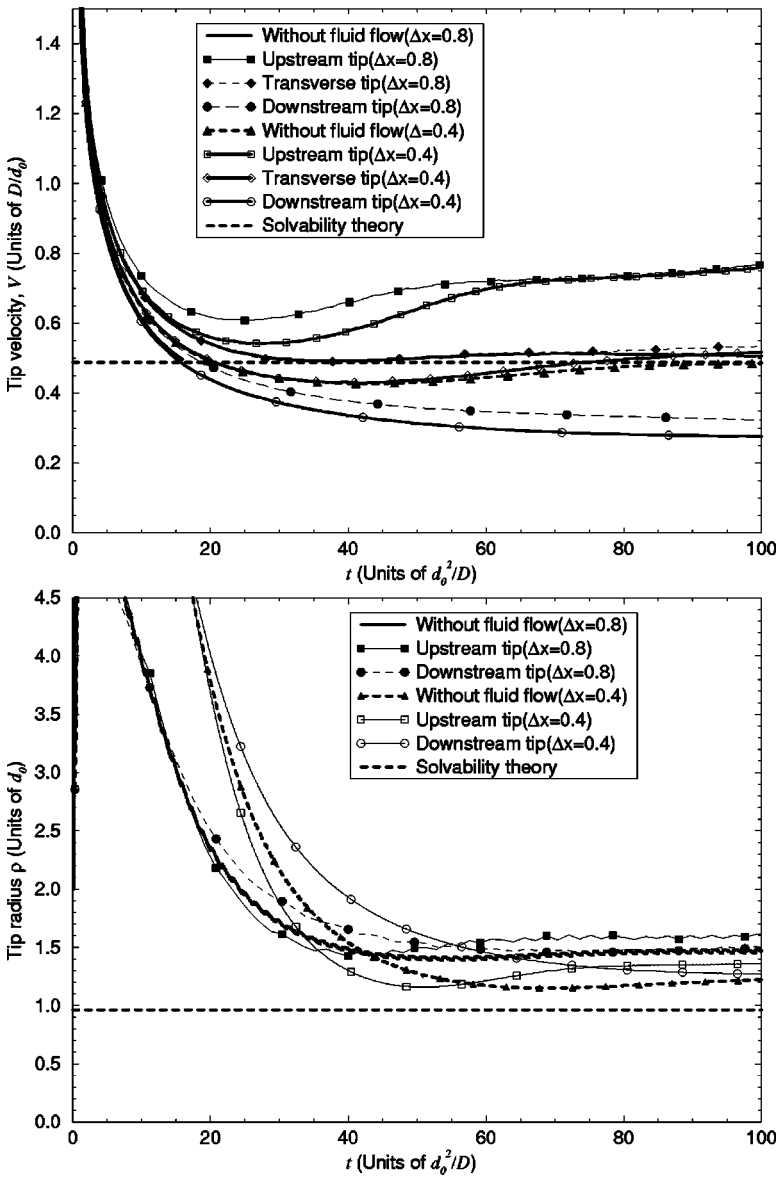

FIG. 8. Tip velocities and tip radii in 2D.

ily extracted by selecting pairs of interpolated points separated by the amount of time it takes to cross one grid spacing. We refer to this as the moving average tip velocity. This method is used to compute the interface velocities reported for the calculations in the remainder of this paper.

VI. RESULTS

A. Two-dimensional verification problem

In order to validate our 3D code, we simulated the 2D example analyzed by Beckermann *et al.* [13]. The computa-

TABLE I. Results for the simulations of dendrite growth in 2D and 3D.

	Solvability solution (2D)	2D ($\Delta x = 0.8$)	2D ($\Delta x = 0.4$)	3D ($\Delta x = 0.8$)
V_{tip}(no flow)	0.489	0.502	0.486	0.915
$V_{tip}^{upstream}$(flow)		0.766	0.761	1.038
$V_{tip}^{downstream}$(flow)		0.324	0.276	0.800
$V_{tip}^{transverse}$(flow)		0.533	0.516	0.915
ρ_{tip}(no flow)	0.959	1.47	1.22	2.90
$\rho_{tip}^{upstream}$(flow)		1.60	1.36	3.50
$\rho_{tip}^{downstream}$(flow)		1.50	1.28	2.41

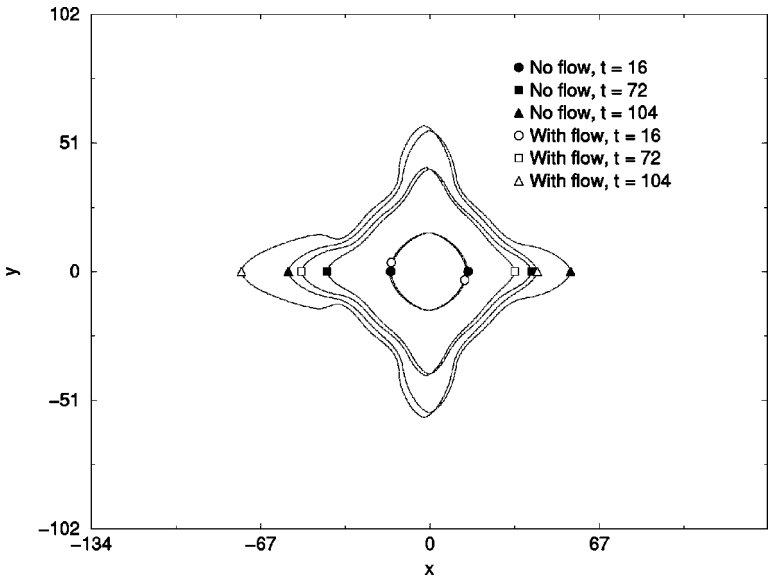

FIG. 9. Computed interfaces in 2D without and with fluid flow at $t=16$, 72, and 104.

tional domain is a square with an initial circular seed. A 3D domain is created by extending the square domain and the circular seed into the y direction.

We used a box of edge length $L=204.8$ so that subelements of $\Delta x=0.8$ or 0.4 are created through repeated bisection of the domain. The flow enters at the left edge, with $u_x=1$ and $u_y=0$. The top and bottom surfaces are symmetry boundaries. Beckermann *et al.* [13] used $L=230.4$ and similar values for Δx. The problem parameters are undercooling $\Delta=0.55$, thermal diffusivity $D=4$, and anisotropy $\epsilon_4=0.05$. We also used $\Delta t=0.016$, $W=1$, $\tau_0=1$, and $\lambda=6.383$. The capillary length $d_0(=a_1W/\lambda)$ is 0.139, the kinematic viscosity ν is 92.4, and the Prandtl number Pr calculated from these parameters is 23.1. These physical parameters correspond to succinonitrile. The adapted grid is newly regenerated every 20 time steps. We examined two minimum grid spacings, $\Delta x_{\min}=0.8$ and $\Delta x_{\min}=0.4$. The largest element size was $\Delta x_{\max}=3.2$ for both cases.

For 2D dendrite growth without fluid flow, when using $\Delta x=0.8$ and $\Delta x=0.4$, the steady-state dendrite tip velocities V_{tip} scaled with D/d_0 are 0.017 44 and 0.016 89, respectively. Those values are in good agreement with solvability theory (0.017 00) and previous phase-field results [13]. We calculated the tip radius ρ_{tip} using the method in Ref. [6], where the tip curvature is computed using estimates of the second derivatives at the tip interpolated along the two coordinate directions at the tip. We obtained $\rho_{tip}/d_0=10.58$ and 8.78 for $\Delta x=0.8$ and $\Delta x=0.4$, respectively. The ratio ρ_{tip}/d_0 for $\Delta x\rightarrow 0$ can be computed by Richardson extrapolation, plotting ρ_{tip} versus Δx^2. The extrapolated value of this ratio for $\Delta x\rightarrow 0$ is 8.20, which is somewhat larger than the solvability solution (6.90).

An alternative method to fit the tip radius, given by Wheeler *et al.* [20], was also used. In this method, we fit to an effective Ivantsov solution by calculating the parabolic tip radius ρ^p_{tip} from the slope of z versus x^2 in the region behind the tip where the curve becomes straight. The tip radius obtained this way is $\rho^p_{tip}=3.84$ and the tip Peclet number

$$\mathrm{Pe}^p=\frac{V_{tip}\rho^p_{tip}}{2D}=0.237. \tag{34}$$

The relative difference between the Pe^p and Pe^{Ivan} (0.257) [13] obtained from the Ivantsov relation is less than 8%. Thus, using ρ^p_{tip} does indeed remove uncertainties from using the value at just one point.

Figures 8 and 9 and Table I summarize the results for 2D simulations. For the upstream tips, the steady tip velocities for $\Delta x=0.8$ and $\Delta x=0.4$ agree with the results of Beckermann *et al.* [13] within 9% percent. For $\Delta x=0.4$, the upstream tip grows approximately 55% faster than in the growth without flow at $t=100$, while the downstream tip

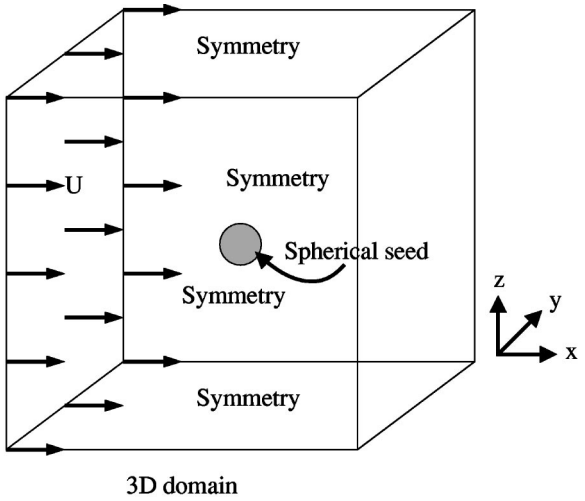

FIG. 10. Schematic diagram of 3D computation domain and boundary conditions.

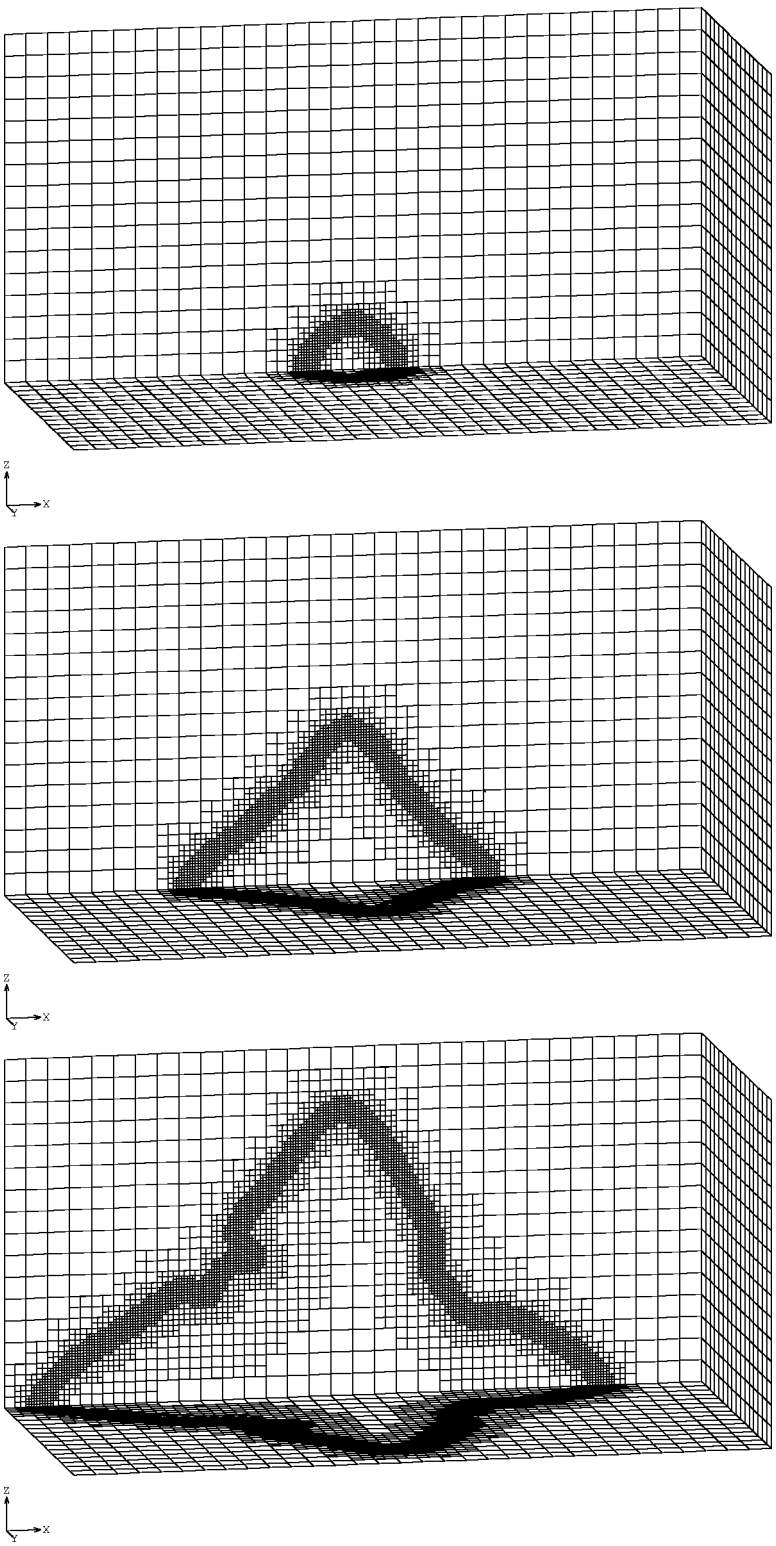

FIG. 11. Grid configurations of adapted grids $t=9.6$, 43.2, and 86.4.

grows approximately 45% slower. Beckermann *et al.* reported that the upstream tip grows 40% faster and the downstream tip grows more than 30% slower. The trend for changing the tip velocities according to the growing directions agrees well with the results by Beckermann *et al.* [13] For the upstream tips, the ratios $\rho_{tip}/\rho_{tip}^{0}$ with and without fluid flow are 1.09 and 1.12 for $\Delta x=0.8$ and 0.4, respectively. Those values also agree well with the result (1.11) obtained by Beckermann *et al.* [13] The ratio σ_0^*/σ^*, where the subscript 0 refers to the the solution in the absence of fluid flow and $\sigma^*=2d_0D/\rho_{tip}^2V_{tip}$, is 1.95 and 0.63 for the upstream tip and downstream tip, respectively. These results agree reasonably well with those of Beckermann *et al.*

A detailed comparison is not meaningful because our

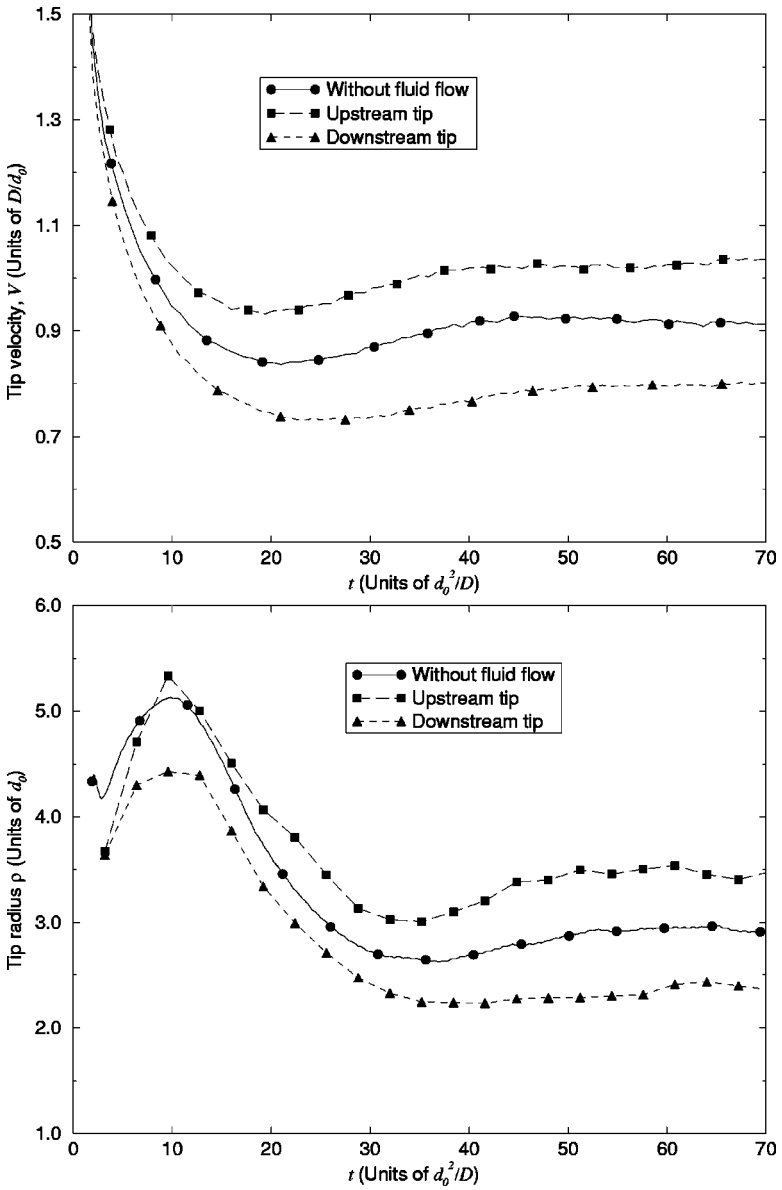

FIG. 12. Tip velocities and tip radii in 3D.

simulations have not reached steady state, and in some cases are not fully grid converged, as shown in Table I. Figure 9 shows interfaces of the dendrites with and without fluid flow in 2D. The interfaces are in good agreement with previous results [13].

B. Three-dimensional computations

We simulated 3D dendrite growth with fluid flow using a cube computation domain with a spherical seed, as shown in Fig. 10. The inlet velocity boundary conditions are imposed on the left side boundary and their values are $u_x = U$, $u_y = 0$, and $u_z = 0$. For this particular problem, we again have $U = 1$. The largest Δx is 6.4, while the smallest Δx is 0.8. Time increments of $\Delta t = 0.016$ and $\Delta t = 0.008$ are used to model growth without and with fluid flow, respectively. The other parameters are identical to the parameters for the 2D analysis in previous section. Figure 11 shows several of the grid configurations in the analysis. The scale factor γ used in the adaptive grid procedure was 10 for this simulation.

The simulations using $\Delta x = 0.8$ and $\Delta x = 0.4$ consumed about 40 h and 190 h, respectively, on a single processor of the IBM RS/6000 machine with clock rate of 200 MHz. In the simulation where $\Delta x = 0.8$, the initial mesh consisted of 10 607 elements, and the final mesh had 208 357 elements. These numbers should be compared with a fully dense mesh, which would have had over 4×10^6 elements. With fluid flow, computing the velocity and pressure fields consumes approximately 80%–90% of the total CPU time. We performed some the simulations using a different time increment for velocity field Δt_V, larger than the time increment for temperature and ϕ fields $\Delta t_{\theta\phi}$. For $\Delta x = 0.8$ and $\Delta t_V = 5\Delta t_{\theta\phi}$, the CPU time decreased by 76%, and the results for ρ_{tip} and V_{tip} agree within 5% with the results obtained by using $\Delta t_V = \Delta t_{\theta\phi}$. The detailed results presented next were computed using $\Delta t_V = \Delta t_{\theta\phi} = 0.008$ in order to compare our results with the previous 2D results. This run took approximately 250 h on 16 processors of the ORIGIN2000 at the National Center for Supercomputing Applications

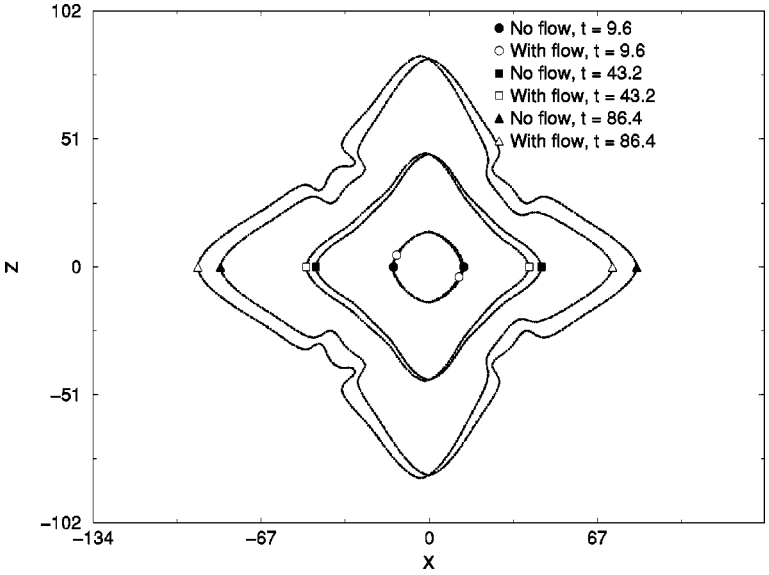

FIG. 13. Computed interfaces in 3D without and with fluid flow at $t=9.6$, 43.2, and 86.4.

(NCSA). The same run using $\Delta t_{\theta\phi}=0.016$ and $\Delta t_V=5\Delta t_{\theta\phi}$ consumed approximately one-eighth of the run time for $\Delta t_V=\Delta t_{\theta\phi}=0.008$, and the deviation in the results was less than 3%. Thus, a run time of about 32 h on 16 processors yielded essentially the same result.

Without fluid flow, the steady scaled tip velocity $V_{tip}d_0/D$ is 0.0318 and the ratio $V_{tip}^{2D}/V_{tip}^{3D}$ for $\Delta x=0.8$ is 0.55. Karma and Rappel [6] reported that this ratio is 0.39 for $\Delta=0.65$ and effective anisotropy $\epsilon_e=0.0269$. The ρ_{tip}/d_0 is 20.86 and the ratio $\rho_{tip}^{2D}/\rho_{tip}^{3D}$ for $\Delta x=0.8$ is 0.51. The ratio $\sigma_{2D}^*/\sigma_{3D}^*$ is 7.09, where σ^* is the selection constant $\sigma^*=2Dd_0/\rho_{tip}^2V_{tip}$.

The time evolution of the dendrite tip velocities and tip radii are shown in Fig. 12. With fluid flow, the upstream tip grows 13% faster than it does without flow, while the downstream tip grows 13% slower. We stopped the calculations at $t=86.5$, before the diffusion field encounters the end of the computational domain. These results seem to show that the effect of 3D flow on the upstream dendrite tip velocity is much weaker than the effect of 2D flow. However, the reason for the trend is that the effect of forced flow on tip velocity is highly dependent on the ratio U/V_{tip} where U is the inlet velocity. The ratio U/V_{tip}^{3D} is 45% smaller than the ratio U/V_{tip}^{2D}, and this accounts for the reduced effect. The ratio ρ_{tip}/ρ_{tip}^0 is 1.21 for 3D for the upstream tip, which is slightly larger than the ratio $\rho_{tip}/\rho_{tip}^0=1.09$ for 2D. The tip radius for the downstream in 3D decreases by 17% due to the flow effect, while the tip radius in 2D increases by 2%. Thus, the effect of the forced flow on the downstream tip is much stronger in 3D than in 2D. The cause of the trend is that in 2D the fluid flows vertically over the dendrite, while in 3D the fluid flows both vertically and laterally over the dendrite.

Figure 13 shows several interfaces in the x-z plane. The tilt angle of the transverse arms into the flow in 3D is 2.5°, while the tilt angle in 2D is 2.3°. From Fig. 13 we can see

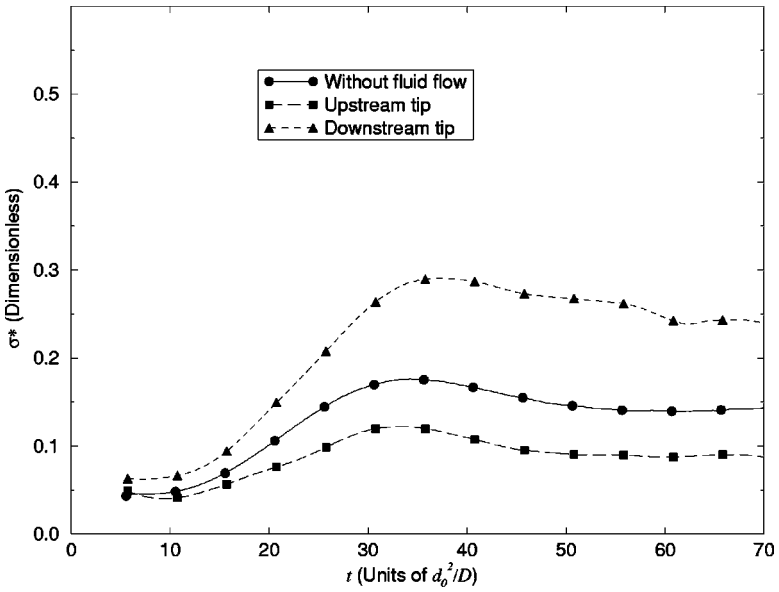

FIG. 14. σ^*s in 3D.

that sidebranches begin to appear on the upstream side of the transverse tip more quickly than on the other sides.

Bouissou *et al.* [21] showed that when surface tension effects are neglected, the Peclet number Pe_V computed using V_{tip} is related to the Peclet number computed using the inlet velocity, Pe_U, for a given undercooling. In order to assess the effect of the forced flow on the tip Peclet number, we computed the the difference between the tip Peclet number with fluid flow, $\mathrm{Pe}_V^f = V^f \rho^f / 2D$, and the tip Peclet number without fluid flow, $\mathrm{Pe}_V^0 = V^0 \rho^0 / 2D$, divided by the Peclet number related to the inlet velocity, $\mathrm{Pe}_U^f = U \rho^f / 2D$:

$$\Delta \mathrm{Pe} = \frac{\mathrm{Pe}_V^f - \mathrm{Pe}_V^0}{\mathrm{Pe}_U^f}, \tag{35}$$

We found that for the upstream tip in 2D, Pe_U^{2D} is 0.305 and it is similar to $\mathrm{Pe}_U^{3D} (= 0.275)$, even though the effect of the forced flow on the tip velocities and tip radii in 2D and 3D is much different.

We found that the ratio σ_0^* / σ^* for 3D is 1.675 for upstream tip and 0.603 for the downstream tip. With flow present, it takes somewhat longer for σ^* to settle onto a steady value, as seen in Fig. 14. The trend in σ^* for the upstream tip with fluid flow agrees with experiments by Bouissou *et al.* [21] on alloys of PVA and ethanol, while the trend is opposite to the experiments by Lee *et al.* [22] on SCN forced past a needle. Further work is needed to examine this phenomenon.

VII. CONCLUSIONS

In this paper, we presented an efficient algorithm using 3D adaptive grid refinement in order to study the effect of fluid flow on 3D dendrite growth using a phase-field model. We simulated 3D dendrite growth with fluid flow using adapted grids with at most 300 000 elements, while over 4 000 000 elements would be required for an uniform grid. We also present a parallel implementation by using the CHARM++ FEM framework. A speed-up factor SP=28.8 for 32 processors is typical for the larger meshes.

We also introduce a new scheme using a hyperbolic tangent function to interpolate tip position. We found that the scheme, which requires just two points for interpolation, has a smaller oscillation amplitude than third-order polynomial interpolation using four points.

Test cases showed that the dendrite tip velocities and radii for 2D dendrite growth were in satisfactory agreement with solvability theory and previous computational results [13]. We found that the effect of fluid flow on dendrite tip velocity in 2D is much larger than in 3D, because the ratio U/V_{tip}^{2D} is approximately 2 times larger than the ratio U/V_{tip}^{3D}. For the downstream tip, the tip radius in 3D decreases by 17%, and the tip radius in 2D increases by 2%. The cause of the trend is that in 3D the fluid flows both vertically and laterally over the dendrite and the effect of the forced flow in 3D is much stronger than in 2D where the fluid flows vertically over the dendrite.

We examined the effect of the forced flow on the tip Peclet number by computing ΔPe: the difference between the tip Peclet number in the growth with fluid flow and the tip Peclet number in the growth without the fluid flow, divided by the Peclet number related to the forced flow. We found that ΔPe in 2D and in 3D are within 10%, even though the tip radii and velocities vary by much more than that. In 3D growth with fluid flow, σ^* for the upstream tip decreases by 39%. The trend for σ^* for the upstream tip agrees with experiments (reduction of 37%) by Bouissou *et al.* [21] on alloys of PVA and ethanol. The ratio $\sigma_0^* / \sigma^{*3D}$ for the upstream tip is 14% smaller than the ratio $\sigma_0^* / \sigma^{*2D}$.

ACKNOWLEDGMENTS

This work was supported by the NASA Microgravity Research Program, under Grant No. NAG8-1249, and the National Science Foundation under Grant No. NSF DMS 98-73945. The authors wish to thank S. Kale, M. Bhandarkar, O. Lawlor, and T. Hinrichs for their help with the parallelization of the code. We also thank C. Beckermann and A. Karma for helpful discussions and sharing of their results.

[1] W. W. Mullins and R. F. Sekerka, J. Appl. Phys. **35**, 444 (1964).
[2] J. Langer, in *Chance and Matter*, Les Houches Session XLVI, edited by J. Souletie, J. Vannenimus, and R. Stora (North-Holland, Amsterdam, 1987), pp. 629–711.
[3] D. A. Kessler, J. Koplik, and H. Levine, Adv. Phys. **37**, 255 (1988).
[4] E. Ben-Jacob, N. Goldenfeld, B. Kotliar, and J. Langer, Phys. Rev. Lett. **53**, 2110 (1984).
[5] D. Kessler, J. Koplik, and H. Levine, Phys. Rev. A **30**, 3161 (1984).
[6] A. Karma and W.-J. Rappel, Phys. Rev. E **53**, 3017 (1995).
[7] N. Provatas, N. Goldenfeld, and J. Dantzig, Phys. Rev. Lett. **80**, 3308 (1998).
[8] Y.-T. Kim, N. Goldenfeld, and J. Dantzig, Phys. Rev. E **62**, 2471 (2000).
[9] S. Chen, B. Merriman, S. Osher, and P. Smereka, J. Comput. Phys. **135**, 8 (1997).
[10] S. H. Davis, *Theory of Solidification* (Cambridge University Press, Cambridge, England, 2001).
[11] J. A. Dantzig and L. S. Chao, In *Proceedings of the 10th U.S. National Congress of Applied Mechanics*, edited by J. Lamb (ASME, New York, 1986), p. 249.
[12] N. Provatas, N. Goldenfeld, and J. Dantzig, J. Comput. Phys. **148**, 265 (1999).
[13] C. Beckermann, H.-J. Diepers, I. Steinbach, A. Karma, and X. Tong, J. Comput. Phys. **154**, 468 (1999).
[14] P. M. Gresho, S. T. Chan, M. A. Christon, and A. C. Hindmarsh, Int. J. Numer. Methods Fluids **21**, 837 (1995).
[15] X. Mikic and E. C. Morse, J. Comput. Phys. **61**, 154 (1985).

[16] A. N. Brooks and T. J. R. Hughes, Comput. Methods Appl. Mech. Eng. **32**, 199 (1982).
[17] L. V. Kale and S. Krishnan, in *Parallel Programming using C++*, edited by G. V. Wilson and P. Lu (MIT Press, Cambridge, MA, 1996), p. 175.
[18] M. Bhandarkar and L. V. Kalé, in Proceedings of the International Conference on High Performance Computing (HiPC), Vol. 1970, *2000*, edited by M. Valero, V. K. Prasanna, and S. Vajpeyam (unpublished).
[19] G. Karypis, http://www-users.cs.umn.edu/~karypis/metis/
[20] A. A. Wheeler, B. T. Murray, and R. J. Schaefer, Physica D **66**, 243 (1993).
[21] P. Bouissou, B. Perrin, and P. Tabeling, Phys. Rev. A **40**, 509 (1989).
[22] Y.-W. Lee, R. Ananth, and W. N. Gill, J. Cryst. Growth **132**, 226 (1993).