Available online at www.sciencedirect.com

SCIENCE DIRECT®

Journal of Volcanology and Geothermal Research 139 (2005) 1–21

Journal of volcanology and geothermal research

www.elsevier.com/locate/jvolgeores

Parallel adaptive numerical simulation of dry avalanches over natural terrain

A.K. Patra[a,*], A.C. Bauer[a], C.C. Nichita[b], E.B. Pitman[b], M.F. Sheridan[c], M. Bursik[c], B. Rupp[c], A. Webber[c], A.J. Stinton[c], L.M. Namikawa[d], C.S. Renschler[d]

[a]Department of Mechanical and Aerospace Engineering, State University of New York-Buffalo, 605 Furnas Hall, SUNY, Buffalo, NY 14260, USA

[b]Department of Mathematics, University at Buffalo, SUNY, Buffalo, NY 14260, USA

[c]Department of Geology, University at Buffalo, SUNY, Buffalo, NY 14260, USA

[d]Department of Geography, University at Buffalo, SUNY, Buffalo, NY 14260, USA

Accepted 29 June 2004

Abstract

High-fidelity computational simulation can be an invaluable tool in planning strategies for hazard risk mitigation. The accuracy and reliability of the predictions are crucial elements of these tools being successful. We present here a new simulation tool for dry granular avalanches using several new techniques for enhancing numerical solution accuracy.

Highlights of our new methodology are the use of a depth-averaged model of the conservation laws and an adaptive grid Godunov solver to solve the resulting equations. The software is designed to run on distributed memory supercomputers and makes use of digital elevation data dynamically, i.e., refine the grid and input data to finer resolutions to better capture flow features as the flow evolves. Our simulations are validated using quantitative and qualitative comparisons to tabletop experiments and data from field observations. Our software is freely available and uses only publicly available libraries and hence can be used on a wide range of hardware and software platforms.

Keywords: granular flow; Grain flow; flow simulation; GIS; adaptive methods; parallel computing

1. Introduction

Volcanic activity results in a variety of mass flows ranging from passive gas emission and slow effusion of lava to explosions accompanied by development of a stratospheric plume with associated dense, pyroclastic flows of red-hot ash, rock, and gas that race along the surface away from the volcano. Often, these flows mix with melted snow creating a muddy mix of ash, water, and rock. Accompanying seismic activity at a volcano can trigger slope failures generating a giant debris avalanche that can ruin vast areas of

* Corresponding author. Tel.: +1 716 645 2593; fax: +1 716 645 3875.
E-mail address: abani@eng.buffalo.edu (A.K. Patra).

productive land, destroy structures, and injure or kill the population of entire cities. In these avalanches and debris flows, particles are typically centimeter- to meter-sized, and the flows, sometimes as fast as hundreds of meters per second, range over tens of kilometers. As these flows slow, the particle mass sediments out, yielding deposits that can be a hundred meters deep and many kilometers in length.

The task of modeling these events is complex and at present only beginning to be understood. Nevertheless, public safety planning needs and scientific investigations will benefit greatly from the development of tools and designed to answer the simple question:

If a mass flow were to be initiated at a particular location, what areas are going to be affected and to what degree by that flow?

In this paper, we describe our efforts at developing a tool (the TITAN2D simulation code) to satisfy this need.

A popular class of models for these events treats them as depth-averaged granular flows governed by Couloumb-type interactions (Hutter et al., 1993; Iverson and Denlinger, 2001; Gray, 1997) with or without a pore fluid—this is the model of the physics we choose to use in this development. We will use as a starting point the equations of Iverson and Denlinger (2001) in the dry limit. The scale of the flows and complexity resulting from the modeling of flows over natural terrain will require large-scale computation and special techniques to reliably obtain good numerical simulations of the complex flows—the focus of this paper. Our numerical algorithm for solving the governing model equations is an adaptive grid second-order Godunov solver (Toro, 1997). Local mesh adaptivity is crucial to reliably resolving flow features and the necessary shock capture. We have also developed suitable computational techniques (multiprocessor computing, dynamic load balancing, etc.) to enable the necessary large-scale simulations on popularly available cluster computers and more efficient distributed memory multicomputers.

An important feature of this work is the incorporation of a direct connection to geographic information system (GIS) databases. Thus, we obtain required topographic data dynamically as needed by the progress of the simulation at resolutions appropriate for the accuracy of the computation. These ingredients allow us to simulate large flows over realistic terrain; here, we provide an example of such a simulation at little Tahoma Peak (see accompanying paper in this issue by Sheridan et al., 2004).

We begin our discussion with a review of the physics modeling used in our simulation tool and derive the basic governing equations. We follow by providing details of the solution methodology. Numerical tests are used to verify the code and study its performance under different choices of model and numerical parameters. Validation using tabletop experiments and simulation of observed flows completes the presentation. It is important to note that as a modeling tool, the TITAN2D code has several new features that reduce computation and modeling errors, but as with all simulation tools for such complex physical phenomena, many assumptions are inherent in the results presented. All results and outputs are hence qualified and subject to the validity of the assumptions made. Regardless, we believe that careful use of such tools can provide much valuable insight and assist in the hazard analysis process.

2. Governing equations

2.1. Models

We model the geophysical mass flows on realistic terrain, such as depth-averaged granular continuua. This approach for describing debris flows was first suggested by Savage and Hutter (1989). The original one-dimensional theory was later generalized to two dimensions by the same authors, by introducing a simple curvilinear coordinate system with orthogonal directions being set by the maximum slope (x-axis), the normal to the local surface (z-axis), and a cross-slope axis normal to the other two. However, these equations are not frame-invariant and hence unsuitable for modeling of flows over general terrain. In recent work, Iverson and Denlinger (2001) derive depth-averaged, frame-invariant equations for fluidized granular masses on three-dimensional terrain. They also include the effect of interstitial fluid using a simple mixture theory approach. These equations form a system of hyperbolic conservation laws, referred to as the debris flow equations (DFE). In a follow-up paper, Denlinger and Iverson (2001) also report on basic numerical solutions to the DFE using a first-order Godunov method and an approximate Riemann solver.

These debris flow equations in the zero pore pressure limit constitute the starting point of our work. We solve the equations using a finite volume scheme with a second-order Godunov solver. The program runs in parallel, using the message passing interface standard (MPI) to allow communication between multiple processors. The algorithm uses a local adaptive mesh refinement for shock capturing, and dynamic load balancing for the efficient use of the computational resources.

We begin by briefly reviewing the derivations of the model equations. For a detailed description of the depth-averaged theory for debris flows, we refer the reader to the reference papers cited above.

2.2. Basic equations and boundary conditions

In a fixed Cartesian coordinate system OXYZ, with origin O defined so that the plane OXY is approximately parallel to the basal surface, we write the conservative form of the equations for an incompressible continuum:

$$\nabla \cdot \boldsymbol{u} = 0 \tag{1}$$

$$\partial(\rho_0 \boldsymbol{u}) + \nabla \cdot (\rho_0 \boldsymbol{u} \otimes \boldsymbol{u}) = -\nabla \cdot T + \rho_0 g \tag{2}$$

where ρ_0 is the density of the medium, $\boldsymbol{u}$ is the velocity field, T is the Cauchy stress tensor, and g is the gravitational acceleration.

The granular material is assumed to be an incompressible continuum satisfying a Mohr Coulomb law, which states that slip planes appear inside the bulk as soon as the internal state of stress overpasses the Coulomb criteria of failure, $\sigma_t/\sigma_n=\tan\varphi_{\mathrm{int}}$, where σ_n and σ_t are the normal and shear stresses acting on a plane element inside the granular material, and φ_{int} is the internal friction angle of the medium.

Kinematic boundary conditions are imposed at the free surface interface, of equation $F_s(x,t)=s(x,t)-z=0$, and at the basal surface interface, with equation $F_b(x,t)=b(x,t)-z=0$:

$$\partial_t F_s + (\boldsymbol{u} \cdot \nabla) F_s = 0 \qquad \text{at } F_s(x,t) = 0 \tag{3}$$

$$\partial_t F_b + (\boldsymbol{u} \cdot \nabla) F_b = 0 \qquad \text{at } F_b(x,t) = 0 \tag{4}$$

Written explicitly on components, the above equations are:

$$\partial_t s + v_x \partial_x s + v_y \partial_y s - v_z = 0 \qquad \text{at } z = s(x,t) \tag{5}$$

$$\partial_t b + v_x \partial_x b + v_y \partial_y b - v_z = 0 \qquad \text{at } z = b(x,t) \tag{6}$$

where $\boldsymbol{u}=(v_x,\ v_y,\ v_z)$ denotes the velocity vector and its components.

Boundary conditions for the stresses are stress-free condition at the free surface and a Coulomb-like friction law imposed at the interface between the granular flow and the basal surface:

$$T^s n^s = 0 \qquad \text{at } z = s(x,t) \tag{7}$$

$$T^b n^b - n^b(n^b \cdot T^b n^b) = \frac{\boldsymbol{u}^r}{|\boldsymbol{u}^r|} \tan\varphi_{\mathrm{bed}}$$

$$(n^b \cdot T^b n^b) \qquad \text{at } z = b(x,t) \tag{8}$$

The superscripts s and b applied to the stress and to the components of the normal vector refer to values of the variables which are assumed at the free surface (s) and at the base of the flow (b). n denotes normals to surfaces with the superscripts s and b referring to the free surface and the base of the flow. $\boldsymbol{u}^r=\boldsymbol{u}^{b+}-\boldsymbol{u}^{b-}$ is the velocity vector whose components are equal to the difference between the upper- and lower-side velocity values at the boundary layer of infinitesimally small thickness that forms at the basal contact surface. The factor $(\boldsymbol{u}^r)/|\boldsymbol{u}^r|$ in the tangential component of the shear at the surface element of normal n^b, indicates that the Coulomb friction opposes the avalanche motion.

The three-dimensional system gives a detailed description of the flow; however, the model requires supplementary relations, such as an equation for the free surface and also a three-dimensional treatment of Coulomb stresses, which add to the complexity of the problem.

Under the additional assumption that the flowing layer is thin compared to its lateral extension, the detailed motion of the mass through depth becomes relatively unimportant except in a thin layer near the bed. The processes in this boundary layer can be approximated by the Coulomb-type basal sliding law, and an average of the system of equations over the depth of the flow provides a simpler model that can be handled numerically. The depth-averaged model is appropriate for geophysical mass flows when the assumption of shallowness holds. The case for this assumption has been made by many (see, for example, Hutter et al., 1993; Iverson and Denlinger, 2001).

2.3. Depth-averaged theory

In this subsection, we give an outline of the depth-averaging procedure used to derive the system of equations for the dry avalanches model.

We start by integrating the continuity equation in the z direction. Using Leibniz' formula to interchange the differentiation and integration operators, we obtain:

$$\partial_t h + \partial_x(h\bar{v}_x) + \partial_y(h\bar{v}_y) - [\partial_t z + v_x\partial_x z + v_y\partial_y z - v_z]_b^s = 0 \quad (9)$$

where the subscript x, y, z, and t refer to the coordinate axes and time. The notation ∂_x indicates partial derivative with respect to x.

In Eq. (9), $\bar{v}_x$ and $\bar{v}_y$ are the averaged lateral velocities defined as follows:

$$h\bar{v}_x = \int_b^s v_x d_z, \quad h\bar{v}_y = \int_b^s v_y d_z,$$
$$h(x,t) = s(x,t) - b(x,t) \quad (10)$$

where $s(x, t)$ and $b(x, t)$ are the free surface and base as defined earlier. Substituting the kinematic boundary conditions from Eqs. (5) and (6) in Eq. (9), we obtain the equation for the depth-averaged mass balance:

$$\partial_t h + \partial_x\left(h\bar{v}_x\right) + \partial_y\left(h\bar{v}_y\right) = 0 \quad (11)$$

Integrating the x momentum equation in the normal direction and using the Leibnitz formula to interchange the order of differentiation and integration, we obtain the depth-averaged x momentum balance equation:

$$\rho[\partial_t(h\bar{v}_x) + \partial_x(h\bar{v}_x^2) + \partial_y(h\bar{v}_x\bar{v}_y)] - \rho[v_x(\partial_t z + v_x\partial_x z + v_y\partial_y z - v_z)]_b^s = -\partial_x(h\bar{T}_{xx}) - \partial_y(h\bar{T}_{yx}) - [T_{zx}]_b^s + \rho g_x h \quad (12)$$

These equations need to be supplemented with constitutive models. The shallowness assumption gives a "hydrostatic" equation for the normal stresses in the z direction:

$$T_{zz} = (h - z)\rho g_z \quad (13)$$

which after depth averaging becomes a relation for the depth-averaged normal stress in the z direction, $\bar{T}_{zz}=\rho g_z h/2$. Using the Mohr Coulomb theory, the depth-averaged normal stresses $\bar{T}_{xx}$ and $\bar{T}_{yy}$ can be related to the normal stress $\bar{T}_{zz}$ by using a lateral stress coefficient k_{ap}, so that:

$$\bar{T}_{xx} = \bar{T}_{yy} = k_{\text{ap}}\bar{T}_{zz} \quad (14)$$

The active or passive state of stress is developed if an element of material is elongated or compressed, and the formula for the corresponding states can be derived from the Mohr diagram. It may be easily shown that:

$$k_{\text{ap}} = 2\frac{1 \pm [1 - \cos^2\varphi_{\text{int}}(1 + \tan^2\varphi_{\text{bed}})]^{1/2}}{\cos^2\varphi_{\text{int}}} - 1 \quad (15)$$

in which "−" corresponds to an active state ($\partial\bar{v}_x/\partial x+\partial\bar{v}_y/\partial y>0$), respectively, "+" to be the passive state ($\partial\bar{v}_x/\partial x+\partial\bar{v}_y/\partial y<0$).

The shear stresses $\bar{T}_{yx}$ and $\bar{T}_{xy}$ can also be related to the normal stresses $\bar{T}_{xx}$ and $\bar{T}_{yy}$, using a simplification of the Couloumb (nonlinear) model to assume a constant proportionally simplification based on a long history of such a practice in soil mechanics (Rankine, 1857) and an alignment of the stress axis. The equation for the lateral shear stresses can be written as:

$$\bar{T}_{yx} = \bar{T}_{xy} = -\operatorname{sgn}(\partial\bar{v}_x/\partial y) \times k_{\text{ap}}\frac{1}{2}\rho g_z h \sin\varphi_{\text{int}} \quad (16)$$

Finally, the formula for the shear stress at the basal surface T_{zx} can be derived from the basal sliding law. For curving beds, this relation is:

$$T_{zx} = -\frac{\bar{v}_x}{\sqrt{\bar{v}_x^2 + \bar{v}_y^2}}\left[\rho g_z h\left(1 + \frac{\bar{v}_x}{r_x g_z}\right)\right] \times \tan\varphi_{\text{bed}} \quad (17)$$

where r_x is the radius of local bed curvature, and the "−" indicates that basal Coulomb stresses oppose basal sliding. Note that the above relationship is slightly modified from the original in Iverson and Denlinger (2001), where $sgn(\bar{v}_x)$ was used instead of $\bar{v}_x\big/\sqrt{\bar{v}_x^2+\bar{v}_y^2}$. Our observation indicates that in cases where the momentums in the x and y directions differ significantly (e.g., flow down a channel), this relationship provides the necessary scaling in each coordinate direction. With this modification, the friction mobilized is in proportion to the velocity in that direction.

Substituting the kinetic boundary conditions (Eqs. (5) and (6)) and the conditions for the stresses at the free surface (Eq. (7)), and also using the relations between stresses (Eqs.), (16), and (17) that are derived from the Coulomb theory in Eq. (12), the depth-averaged x momentum equation can be written:

$$\partial_t(h\bar{v}_x) + \partial_x\left(h\bar{v}_x^2 + \frac{1}{2}k_{\mathrm{ap}}g_z h^2\right) + \partial_y(h\bar{v}_x\bar{v}_y) = g_x h - hk_{\mathrm{ap}} sgn(\partial\bar{v}_x/\partial_y)\partial_y(g_z h)\sin\varphi_{\mathrm{int}} - \frac{\bar{v}_x}{\sqrt{\bar{v}_x^2+\bar{v}_y^2}}\left[g_z h(1+\frac{\bar{v}_x}{r_x g_z})\right]\tan\varphi_{\mathrm{bed}} \tag{18}$$

The relation for the y momentum equation is similar, and it can be obtained by interchanging x and y in Eq. (18).

3. Solution techniques

3.1. First-order scheme

The system of equations governing the flow of dry avalanches on arbitrary topography was derived in terms of conservative variables and can be written in vectorial form (overbars have been omitted to simplify the notations):

$$\boldsymbol{U}_t + \mathbf{F}(\boldsymbol{U})_x + \mathbf{G}(\boldsymbol{U})_y = S(\boldsymbol{U}) \tag{19}$$

where $\boldsymbol{U}=(h, hv_x, hv_y)^t$, $\mathbf{F}=(hv_x,\ hv_x^2+0.5k_{\mathrm{ap}}g_z h^2,\ hv_x v_y)^t$, $\mathbf{G}=(hv_y,\ hv_x v_y,\ hv_y^2+0.5k_{\mathrm{ap}}g_z h^2)^t$, and $S=(0, S_x, S_y)^t$ and where:

$$S_x = g_x h - hk_{\mathrm{ap}} sgn\left(\frac{\partial v_x}{\partial y}\right)\partial_y(g_z h)\sin\varphi_{\mathrm{int}} - \frac{v_x}{\sqrt{v_x^2+v_y^2}}\left[g_z h\left(1+\frac{v_x}{r_x g_z}\right)\right]\tan\varphi_{\mathrm{bed}}$$

$$S_y = g_y h - hk_{\mathrm{ap}} sgn\left(\frac{\partial v_y}{\partial x}\right)\partial_x(g_z h)\sin\varphi_{\mathrm{int}} - \frac{v_y}{\sqrt{v_x^2+v_y^2}}\left[g_z h\left(1+\frac{v_y}{r_y g_z}\right)\right]\tan\varphi_{\mathrm{bed}} \tag{20}$$

The components of the unknown vector $\boldsymbol{U}$ represent pile height and two components for the depth-averaged momentum. The above system of equations is strictly hyperbolic away from vacuum state h=0 and can be solved numerically by using standard techniques.

We use an explicit Euler scheme for the differential equation with right hand side given by the source terms $S(\boldsymbol{U})$ and a Godunov finite volume solver for the remaining system of hyperbolic conservation laws. Background information on these methods are found for example in Toro (1997) and Hirsch (1990). We are currently investigating higher-order methods including adaptive discontinuous Galerkin Methods, and the results will be presented in a future paper.

We use a Cartesian mesh to discretize our domain. The conservative variables (h, hv_x, hv_y) are discretized as piecewise constants on each rectangular computational cell, and the equations are approximated by infinite differences. The evolution of the flow to the next time step depends on the advective flux at the cell interface, which results form the wave interaction at the boundaries between cells. The fluxes are computed by solving the Riemann problem for the two constant states at each side of the boundary edge.

For the one-dimensional system $\boldsymbol{U}_t+\mathbf{F}(\boldsymbol{U})_x=S$, a Godunov scheme gives the explicit formula $\boldsymbol{U}_i^{n+1}=\boldsymbol{U}_i^n-\Delta t/\Delta x[\mathbf{F}_{i+1/2}^n-\mathbf{F}_{i-1/2}^n]+S_i$, where $\mathbf{F}_{i+1/2}^n$ is the intercell numerical flux corresponding to the boundary between cells i and i+1. The discussion below describes the treatment of fluxes in the x direction, and in practice a similar expression has to be derived for the physical flux $\mathbf{G}$.

We use the HLL (Harten, Lax, van Leer; Toro, 1997) approximation Riemann problem at the cell interface. Other exact or approximate Riemann solvers may be used [e.g., HLLC (Toro, 1997) and Roe (LeVeque, 1992)]; however, we have found there to be little difference, provided the computational grid is sufficiently fine.

The first-order HLL solver we implemented uses cell-centered values. Characteristic speeds are the eigenvalues of the Jacobian matrix of $\mathbf{F}$ and are given by $(\boldsymbol{u}_i+c_i, \boldsymbol{u}_i, \boldsymbol{u}_i-c_i)$, where $c_i=\sqrt{2k_{\mathrm{ap}}h_i}$. We estimate the signal velocities in the solution of the Riemann problem by the following choice proposed by Davis (1998):

$$C_{i+1/2}^l = \min(0, \min(\boldsymbol{u}_{i+1}-c_{i+1}, \boldsymbol{u}_i - c_i)) \tag{21}$$

$$C_{i+1/2}^r = \max(0, \max(\boldsymbol{u}_{i+1}+c_{i+1}, \boldsymbol{u}_i + c_i)) \tag{22}$$

and the fluxes at the frontier between cells by:

$$\mathbf{F}^n_{i+1/2} = \frac{C^r_{i+1/2}\mathbf{F}(\boldsymbol{U}^n_i) - C^l_{i+1/2}\mathbf{F}(\boldsymbol{U}^n_{i+1}) + C^l_{i+1/2}C^r_{1+1/2}(\boldsymbol{U}^n_{i+1} - \boldsymbol{U}^n_i)}{C^r_{i+1/2} - C^l_{i+1/2}} \tag{24}$$

where **F** is the physical flux described in Eq. (19).

Flow fronts occur when zero flow depth exists adjacent to a cell with nonzero flow depth. The errors in front propagation speeds can be very large, therefore separate estimates for speeds are needed in this case. For a front moving in the positive x direction $c_{i+1}=h_{i+1}=0$, and the correct solution consists of a single rarefaction wave associated with the left eigenvalue. The wet/dry front corresponds to the tail of the rarefaction and has exact propagating speed $\boldsymbol{u}_{i+1}=\boldsymbol{u}_i+2c_i$. This problem is similar to the problem involving vacuum states in shock tubes, and the rationale for this approach is discussed in Toro (1997).

Front tracking is another way of dealing with wet/dry fronts; however, it is quite complicated and computationally expensive for multidimensional flows, hence we chose not to implement it in the current version of our code.

3.2. Second-order description

To implement the second-order Godunov method we follow the Van Leer approach described in Davis (1998). To increase spatial accuracy, the solution is represented by piecewise linear approximations, and slope limiting is used to prevent unphysical oscillations. To increase time accuracy, a second-order explicit predictor corrector scheme is implemented.

Eq. (19) can be rewritten:

$$\boldsymbol{U}_t + \mathbf{A}\cdot\partial_x\boldsymbol{U} + \mathbf{B}\cdot\partial_y\boldsymbol{U} = S(\boldsymbol{U}) \tag{25}$$

where **A** and **B** are the Jacobian matrices of **F** and **G**, respectively.

Given $\mathbf{U}_{i,j}^{\,n}$, the $(i,\ j)$ cell average at time $n\Delta t$, the midtime predictor step is:

$$\boldsymbol{U}^{n+\frac{1}{2}}_{i,j} = \boldsymbol{U}^n_{i,j} - \frac{\Delta t}{2}\mathbf{A}^n_{i,j}\Delta_x\boldsymbol{U}^n_{i,j} - \frac{\Delta t}{2}\mathbf{B}^n_{i,j}\Delta_y\boldsymbol{U}^n_{i,j} + \frac{\Delta t}{2}S^n_{i,j} \tag{26}$$

where, in the formula above, $\Delta_x\boldsymbol{U}$, and $\Delta_y\boldsymbol{U}$ are the limited slopes for $\boldsymbol{U}$ in the x and y directions, respectively.

In the corrector step, a conservation update of $\boldsymbol{U}$ is computed as follows:

$$\boldsymbol{U}^{n+1}_{i,j} = \boldsymbol{U}^{n+\frac{1}{2}}_{i,j} - \frac{\Delta t}{\Delta x}\left[\mathbf{F}^n_{i+1/2} - \mathbf{F}^n_{i-1/2}\right] - \frac{\Delta t}{\Delta y}\left[\mathbf{G}^n_{i+1/2} - \mathbf{G}^n_{i-1/2}\right] + \Delta t S_{i,j} \tag{27}$$

This time, the numerical fluxes are computed using as left and right states the values obtained by interpolating the center values to the edge position; that is, for $\mathbf{F}^n_{i+1/2}$ use Eq. (24) with $\boldsymbol{U}^l_{i+1/2,j}=\boldsymbol{U}_{i,j}^{\,n+1/2}+(\Delta_x/2)\Delta_x\boldsymbol{U}^n_{i,j}$ and $\boldsymbol{U}^r_{i+1/2,j}=\boldsymbol{U}^{n+1/2}_{i+1,j}-(\Delta_x/2)\Delta_x\boldsymbol{U}^n_{i+1,j}$ instead of $\boldsymbol{U}^n_i$ and $\boldsymbol{U}_{i+1}^{\,n}$. In the above formulas, Δ_x and Δ_y are used again to denote the limiting slopes in the x and y directions.

4. Computational methodology

In this section, we will describe the core computational methodologies used in the TITAN2D code.

4.1. Adaptive methods

Adaptive methods, wherein the resolution of the numerical approximation is tailored to the solution, have demonstrated their ability to improve the accuracy of numerical simulation without significantly increasing the computational cost. Berger and Collela examined solution-adaptive schemes for hyperbolic problems in Berger and Collela (1989). Although major gains can be obtained by using adaptive methods, these gains do come with a price. In order to utilize adaptive methods successfully, we need to determine (a) where and how to adapt, and (b) how to make the process efficient.

4.1.1. Error indicators

The prerequisite to the successful use of adaptive methods is knowing where to adapt the computational grid. Ideally, an error estimate is calculated which bounds the error in the numerical approximation, and a convergence rate with regard to the mesh parameters is used to minimize the solution error. For many nonlinear problems, such as the equations presented earlier, determining an error estimate and/or convergence rates can be difficult and computationally very expensive. In these cases, error indicators are often used to adapt the mesh. Error indicator shows regions where there are errors in the numerical approximation but do not bound the error. Often, the simplest adaptive strategies involve only examining areas of large derivatives and/or large fluxes or other regions of interest.

In the current version of the TITAN2D tool, we use a relatively simple measure of error for adaptivity. A simple scaled norm of the fluxes around the boundary serves as the primary error indicator. We define the quantity:

$$E_K = \frac{1}{d_k} \oint_{\partial\Omega_K} |\mathbf{F}|^2 ds \tag{28}$$

as a measure of the error associated with cell K. A fixed fraction refinement strategy, i.e., all cells whose E_K ranks in the top p percent are selected for refinement. While this strategy does a good job of selecting areas with high flow rates for refinement, it does not correctly track the flow front where refinement is critical. Thus, we supplement this criteria by also tracking the change in the local flow depth variable with time, i.e., $h_K(t_n) - h_K(t_{n-1})$. A combination of these criteria appears to provide satisfactory refinement of grids. More sophisticated strategies are the subject of current work.

4.1.2. Local refinement of mesh

Local refinement (often termed h refinement in the literature on adaptive methods terminology we avoid here) can be achieved by either splitting an element up into smaller elements or remeshing the domain. Automatic mesh generation is not as efficient as splitting the elements, thus we employ the latter. For the quadrilateral elements used in our work here, an irregular mesh as shown in Fig. 1 will result from splitting elements locally. Such irregular meshes require complex connectivity information to be maintained and updated as the grid changes. To keep this information manageable and the coding complexity manageable, we impose the one-irregularity rule (Demkowicz et al., 1989). This essentially requires that an element can have at most two neighbors on an edge. The result of the one-irregularity rule is that the local refinement of an element can trigger refinements of neighboring elements. This is shown in Fig. 6.

4.1.3. Unrefinement of a mesh

While refinement of a mesh is done to maintain solution quality, unrefinement of a mesh is done to minimize the amount of computation needed to calculate an approximate solution. As the simulation proceeds in a dynamic problem, certain portions of the mesh that were previously adapted to maintain numerical solution quality no longer need such a high mesh resolution and can be unrefined. Because unrefinement of a mesh is only done to save computation, it is not useful to have computationally intensive procedures to calculate which elements to unrefine. Because of this, the unrefinement scheme is not as aggressive as the refinement scheme. To reduce the amount for computation for the unrefinement scheme, three main ideas are used:

(1) *Unrefinement of refined elements*: Only elements that were previously refined are allowed to be unrefined, and the elements are only unrefined to their parent element. This reduces the complexity of the unrefinement procedure and ensures that no new elements need to be created. A drawback of this constraint is that no unrefinement can be performed beyond the original mesh, and if the original mesh is too fine, it cannot be adopted to a proper size.

(2) *Triggered unrefinement*: Because of the one-irregularity rule, unrefinement of a group of elements may trigger unrefinement of other elements. If unrefinement of a group of elements would trigger unrefinement of other elements, the unrefinement is not performed. This ensures that no elements are unrefined which should not be unrefined. Calculating the triggered unrefinement can be a significant computational task.

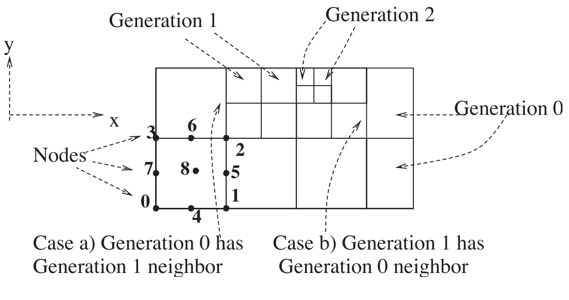

Generation 1
Generation 2
y
x
Generation 0
Nodes
3
6
2
7
8
5
0
4
1
Case a) Generation 0 has Generation 1 neighbor
Case b) Generation 1 has Generation 0 neighbor

Fig. 1. An adapted irregular mesh.

(3) *Unrefinement of an element on a subdomain boundary*: Unrefinement of an element which is on the subdomain boundary is not allowed. Because of this, unrefinement on one processor will not require updating neighbor information on another processor. This reduces the amount of interprocessor communication. While this may seem too restrictive, it is alleviated through the use of dynamic load balancing. It is assumed that after performing dynamic load balancing, the elements that should have been unrefined but were not because they were on the subdomain boundary will become interior elements and can be unrefined later on.

4.1.4. Ghost cells

To calculate the finite difference approximations of the derivatives in the governing equations and the fluxes between elements, the elements in the simulation code need solution information from neighboring elements. For serial codes, all elements will be on the same processor and thus accessible to the processor. This is not the case for parallel codes, because neighboring elements may be located on a different processor and not be directly accessible. To avoid excess interprocessor communication, a "copy" of the off-processor neighboring elements is stored on the processor. These "ghost cells" only store information, and no simulation calculations are performed on these cells. Fig. 2 shows the ghost cells for a two subdomain partition.

4.2. Adaptive Godunov algorithm description

The basic Godunov scheme presented in Section 3 can be adapted with relative ease for use on irregular meshes (see also Berger and Collela, 1989). The calculation of numerical fluxes at the interface between different grids and at the interface between domains belonging to different processors have to be treated separately. Fluxes crossing boundaries have to be balanced at each cell interface so that global and local conservation of mass and momentum are preserved.

The calculation of fluxes is done concurrently on the positive_x_side (see Fig. 1) of the current element being updated and on the negative_x_side of the neighbor element that shares the interface. With the "one-irregularity rule", the two new possible configurations are (a) generation 0 element has a generation 1 neighboring element to the "positive_x_side", and (b) generation 1 element has a generation 0 element neighbor on the "positive_x_side". For the case when the two elements have the same generation, the basic Godunov scheme is used.

Each cell is defined by its nine nodes. The variables of interest (height and momentum) are stored at the center (node 8). Fluxes are stored at the edge nodes (nodes 4–7). To preserve the conservation of mass and momentum, flux balance has to be imposed at each cell interface. The mathematical relations to be satisfied are $\mathbf{F}_p=0.5\times(\mathbf{F}_{m1}+\mathbf{F}_{m2})$ for case (a), and $\mathbf{F}_m=0.5\times(\mathbf{F}_{p1}+\mathbf{F}_{p2})$ for a case (b). The indicators m and p refer to the "minus" or "plus" side of the interface being investigated, the indices "1" and "2" denote the sons of the element with higher generation number, and $\mathbf{F}$ is the numerical fluxes. The above relations may be regarded as integral balance of fluxes over appropriate regions of the linear interface.

The second situation when separate treatment of fluxes has to be done is at the interface between two processors. For the current element, if the interface is on the positive_x_side, only the flux for the current element are calculated, because the neighbors sharing the interface are ghost cells. Similarity, if the interface is on the negative_x_side, only the fluxes on that side of the current element are being updated.

4.3. Code integration in a geographic information system (GIS)

Because we deal in our applications with real-world coordinates, we link our new simulation code dynamically to a geographic information system (GIS) that provides geospatial database. Integration of the simulation code with GIS functionality and GIS data layers requires acquiring appropriate data sources at a resolution compatible with the resolution of our computational grid. Most digital elevation models (DEMs) available use a pixel or raster-based grid data format instead of the computational mesh (or lattice in GIS terminology) used in our modeling approach. This requires that the grid-based elevation data must then be extracted in a way to accurately represent the computational grid potentially at a different resolution (see Fig. 3). Note that in the case of comparable resolutions, GIS data must be appropriately interpolated to avoid

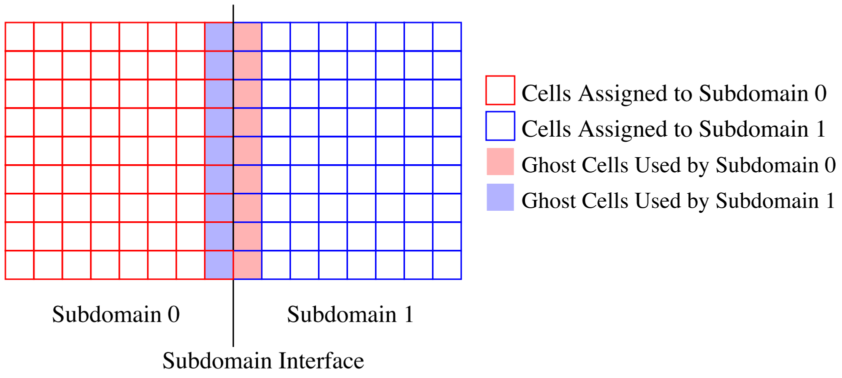

Cells Assigned to Subdomain 0
Cells Assigned to Subdomain 1
Ghost Cells Used by Subdomain 0
Ghost Cells Used by Subdomain 1
Subdomain 0
Subdomain 1
Subdomain Interface

Fig. 2. Example of ghost cells for a two subdomain partition.

creating mathematical artifacts that do not represent the existing topography in the real world.

We decide to use the open-source GIS GRASS (http://www3.baylor.edu/grass/, 2003) that allows a tight coupling between model and GIS code to prepare the initial DEM model input at a particular resolution, and while the model is running, additional elevation mesh points are requested by the simulation at various resolutions derived from the DEM. Elevation and curvature data are interpolated by the model code based on the demands of the dynamically created computational grid at various resolutions.

Because the grid is adapted during the simulation, the computational resolution changes. As new grid points are added to the simulation, new elevation data must be obtained from the GIS system. The other option of deriving elevation and curvature data by interpolating data the from the new computational grid's parent cell degrades the solution quality. Integration of geographic information systems (GIS), such as GRASS (http://www3.baylor.edu/grass/, 2003), with our codes must be accomplished such that our simulations can query the GIS database for accurate topographic information for the newly introduced grid cell. Towards this goal, we have developed a set of interface routines to interactively obtain elevation, slope, and curvature information from GRASS databases.

If the GIS resolution is much finer than the desired computational resolution, then the GIS data can be

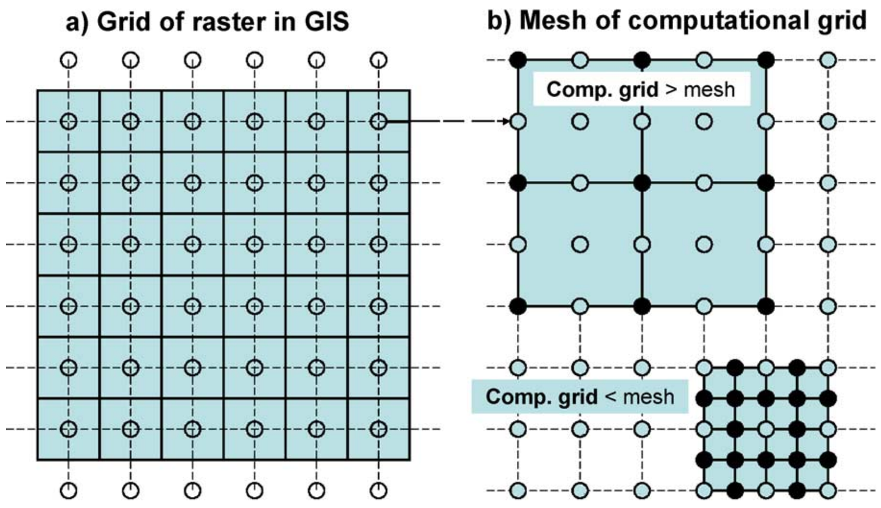


Fig. 3. GIS and computational grids at different resolutions. In the case of comparable resolutions, GIS data must be appropriately interpolated to avoid creating false artifacts.

obtained by a simple look-up process of the GIS grid cell that contains the point for which new data are being requested. However, if the GIS resolution is comparable or smaller, then this process can lead to artificial features. Because of this, the GIS data must be interpolated carefully using interpolation techniques over the computational cell area. In an early implementation, we used the piecewise constant approach to obtaining elevation data, even for cases in which the GIS resolution was comparable to the computational grid. This resulted in the generation of "false" artifacts i.e. small zones in which the GIS created artificial steep features. Such artifacts can greatly corrupt the solution. A more careful study of the effect of postprocessing GIS data is currently underway.

4.4. Parallel adaptive simulations

While adaptive methods can increase the accuracy of a simulation without major increases in computing power, many physical problems still cannot be accurately simulated on single-processor machines. To increase the available computing power, multiple-processor machines need to be used. In order to get the best simulation accuracy, the simulation code must run efficiently on all of these machines, especially those with distributed memory. Good data management and problem decomposition are critical for an adaptive code to run efficiently on such distributed memory-parallel machines.

4.4.1. AFEAPI

An extended version of the finite element data management system, Adaptive Finite Elements Application Programmers Interface (AFEAPI; Patra et al., 2002; Laszloffy et al., 2000; Long, 2001), was used in this code. Originally, AFEAPI was developed for statistic, *hp* adaptive finite element simulations of linear elastostatistics. For this work, AFEAPI was modified to handle a dynamic, *h* adaptive finite difference simulation to the DFE equations presented earlier. AFEAPI greatly simplified the task of managing adaptivity in a parallel environment.

Adding, deleting, and modifying mesh objects while maintaining mesh consistency in a parallel environment, dynamically adjusting mesh partitioning, and migrating appropriate cell objects to maintain load balance during the simulation are among the many cumbersome tasks handled by AFEAPI. Some of the principal ideas of AFEAPI are using Space Filling Curves (SFC; see below) for the cell ordering, a hash table data structure for accessing cell data, and dynamic load balancing. The advantages of using the SFC is that there is support for hierarchical adaptive mesh refinement, fast key generation, key uniqueness, a global address space, and memory locality of data based on physical locality of mesh objects (i.e., the cells). The locality properties serve to improve cache performance in hierarchical memory systems.

4.4.2. Partitioning with space filling curves

The main goals of load balancing are to assign work evenly to a set of processors and to minimize the communication between processors. For grid-based computations, an easy way to obtain such a work distribution is to partition the mesh and assign work associated with different pieces to different processors. If the computational load changes as the computation proceeds, as is the case for h-adaptive methods, the load balancing must be performed in conjunction with the computation. This is called dynamic load balancing. The major constraints of dynamic load balancing are minimizing the time to calculate a division of the work and minimizing the amount of objects that need to be moved between processors. There are currently many different load-balancing algorithms and libraries available (see Hendrickson and Devine, 2000 for a review). In our work here, a variant of the Space Filling Curves partitioning algorithm originally introduced by Salomon and Warren (1993).

4.4.3. Space-filling curves

Space-filling curves are continuous functions that map a bounded one-dimensional interval R onto a bounded n-dimensional space $\boldsymbol{U}_n$, h_n: $R \rightarrow \boldsymbol{U}_n$. Sagan's (1994) recent text provides a very readable review of their salient features. The mapping h_n is continuous and surjective but not injective (onto but not one-to-one). In finite precision computations though, this mapping becomes bijective and a point $x_i \in \boldsymbol{U}_n$ actually represents a 'small' hypercube in the n-dimensional space. The size of the hypercube is determined by the recursion level of the SFC and the size of the original n-dimensional domain. If the original n-dimensional domain is mapped into a unit n-dimensional hypercube

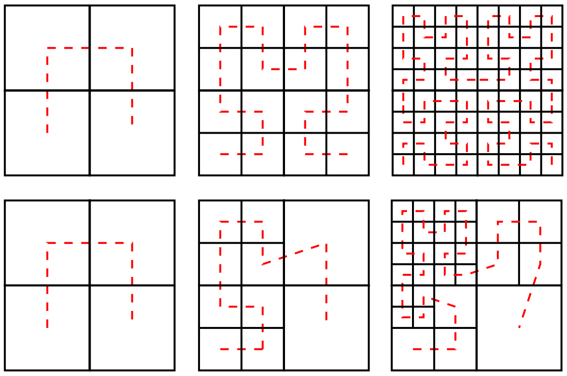


Fig. 4. Finite precision SFC passing through the points of a two-dimensional domain.

as done in Edwards and Browne (1996), then $\boldsymbol{U}_n$=[0,1]^n and the lengths of the sides of the hypercube will be 2^{-r}, where r is the recursion level of the finite precision SFC. The top half of Fig. 4 shows how the SFC traverses through a square for the first three levels of recursion. This is done by calculating the inverse mappings of points in $\boldsymbol{U}_n$ of the SFC, $\xi_i = h_n^{-1}(x_i)$, $x_i \in \boldsymbol{U}_n$ and then sorting the points $\xi_i \in R$. If a set of points are given in $\boldsymbol{U}_n$, an SFC can be calculated which traverses through all of them. The result of this property is the preservation of the spatial locality across the mapping. The bottom half of Fig. 4 shows how the SFC traverses through points of a nonuniform grid.

4.4.4. SFC partitioning

The key idea of the SFC partitioning algorithm is that the task of partitioning objects in an n-dimensional space is easily accomplished by ordering the objects using an appropriate index space and then partitioning the index space. SFCs provide an easy technique for the ordering of objects which are located in an n-dimensional space. The typical scheme for calculating SFC partitions is shown in Table 1 and is illustrated in Fig. 5.

Table 1
Space filling curve partitioning algorithm

1. Find a representative coordinate $x_i \in U^n$ for each object i.
2. Calculate a bounding box B^n in U^n such that $U^n \subset B^n$ and a mapping g: $B^n \rightarrow [0,1]^n$.
3. Calculate the location in R for each object by h_n^{-1} o $g(x_i)$.
4. Sort all of the objects according to their location in R.
5. Calculate the location of the cuts in R that will produce the desired partitioning.

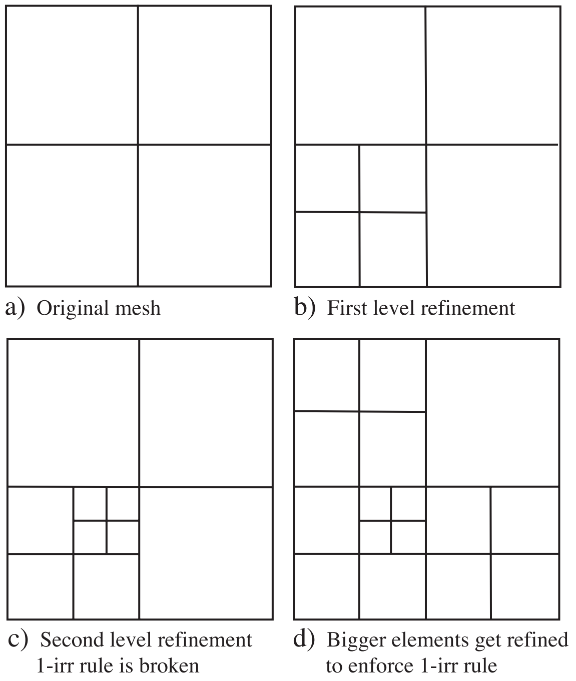

a) Original mesh
b) First level refinement
c) Second level refinement 1-irr rule is broken
d) Bigger elements get refined to enforce 1-irr rule

Fig. 5. SFC partitions of two-dimensional FEM grid that has been h adapted.

R can be viewed as a weighted line itself, because the objects have already been associated with points on the line. To calculate cuts along the weighted line, most SFC partitioning algorithms perform some type of k-way cut calculation to find the cuts between processors in Step 5 of Table 1 of the algorithm.

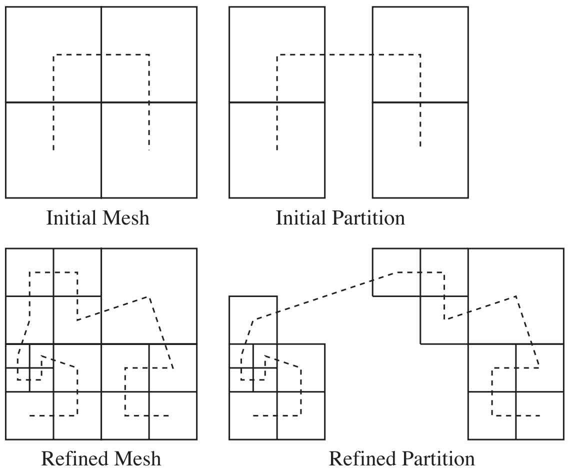


Fig. 6. Triggered refinement as a result of the one-irregularity rule.

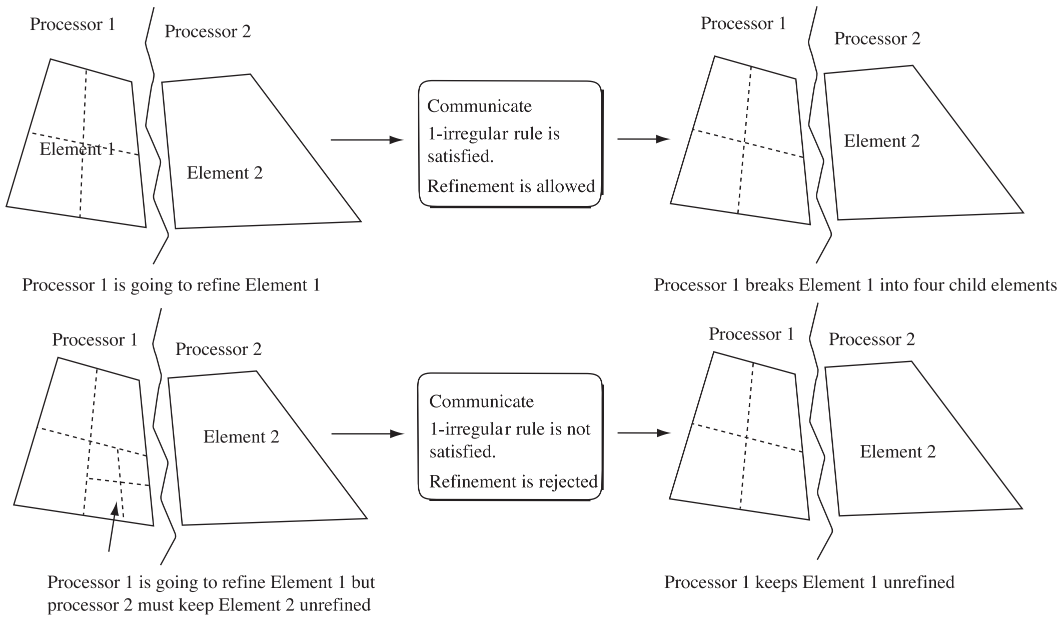

Processor 1
Processor 2
Element 1
Element 2
Communicate
1-irregular rule is satisfied.
Refinement is allowed
Processor 1 is going to refine Element 1
Processor 1 breaks Element 1 into four child elements
Communicate
1-irregular rule is not satisfied.
Refinement is rejected
Processor 1 is going to refine Element 1 but processor 2 must keep Element 2 unrefined
Processor 1 keeps Element 1 unrefined

Fig. 7. Allowed and disallowed refinement at a subdomain interface.

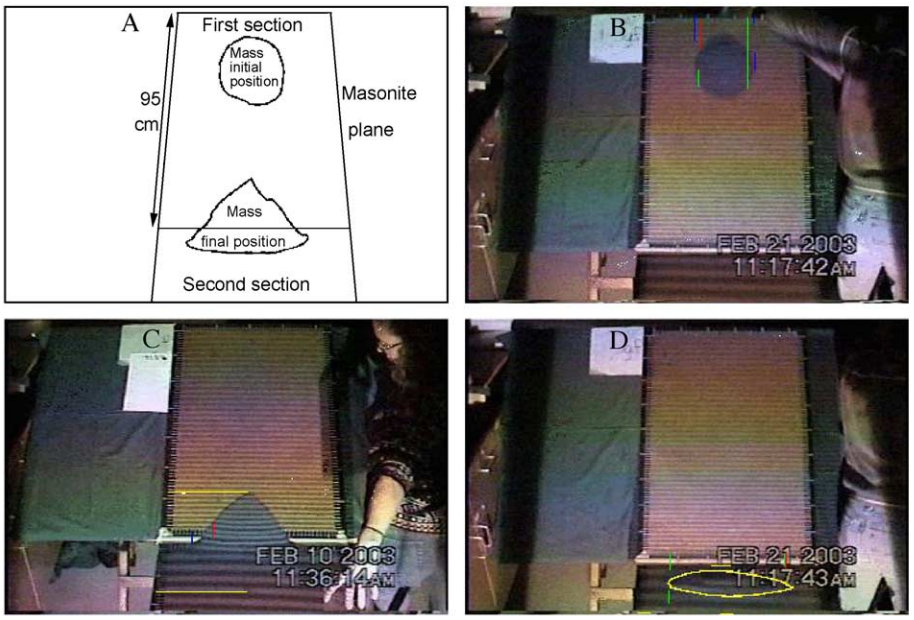


Fig. 8. Experimental setup and example runs. (A) Schematic diagram traced from image showing positions of masonite planes and sandpile masses. Angle of plane was measured with a digital construction level. (B) Starting of experiment at 44.3° with mass of 425.3 g. Image taken 0.23 s after start of experiment. (C) Final position of sandpile from experiment at 31.8° with mass of 425.11 g. Image taken 3.13 s after start of experiment. (D) Final position of sandpile from experiment at 44.3° (as B). Image taken 1.26 s after start of experiment. Note shorter time to final position and more distal final position than in (C). Final sandpile is outlined in yellow for visualization.

4.4.5. Refinement and partitioning

Recall our description of the "one-irregularity" rule for mesh refinement. This can be quite difficult to implement in a parallel code, because refinement in one subdomain can trigger refinement on other subdomain (see Fig. 6). Because of this, the current refinement scheme does not allow triggered refinement in other subdomains. This is shown in Fig. 7. Thus, in the implementation of the refinement process, we reject any refinements that will induce such a pattern of refinement. While this does have an impact on the scheme, in practice the deletrious effects are negligible, because upon the next repartitioning, such cells move away from the "no-refine" zone and are then successfully refined.

5. Verification and validation

We describe here an extensive program of verification and validation of the TITAN2D code. Verification of complex codes like TITAN2D essentially implies the testing of the code for consistency. We have conducted a series of numerical tests to check for consistent behavior. Principal among these have been: (a) checks for convergence of the solution as the grid parameters dx, $dy \rightarrow 0$; (b) comparison of solutions from adapted grids to solutions from very fine nonadaptive grids; and (c) reasonable checks that expected symmetries, and other physically expected behavior are observed.

For validation, we will employ a series of comparisons of the simulation output to tabletop experiments and field observations. Our primary laboratory scale tests were a series of sand flows down a flat inclined plane. The sand is allowed to flow down and spread. We describe now the setup and different runs.

5.1. Validation: inclined plane experiments

5.1.1. Experiments

Laboratory experiments were conducted using sand flows released on a masonite plane (Fig. 8A). In many respects, the experimental setup was similar to that used by Poliquen and Forterre (2002). The masonite plane measured 190×60 cm and consisted of two parts. The first section was tilted at angles of 23.9°–44.3° with an adjustable mount. Particles were released instantaneously on the upper part of this section either from a smaller hemispherical container or a larger cylindrical container. The tilted section was joined to a second section, which dipped from 1° to 2° downstream. The mass of particles released from the hemispherical container was ~43 g, while that released from the cylindrical container was ~425 g. Particles were playground sand grains sieved so that only the 2–2.5φ (177–250 μm) fraction was used. The particles were dyed blue with clothing dye to aid in visualization. The basal friction angle for this material was tested by a number of methods to lie at 18°–29°.

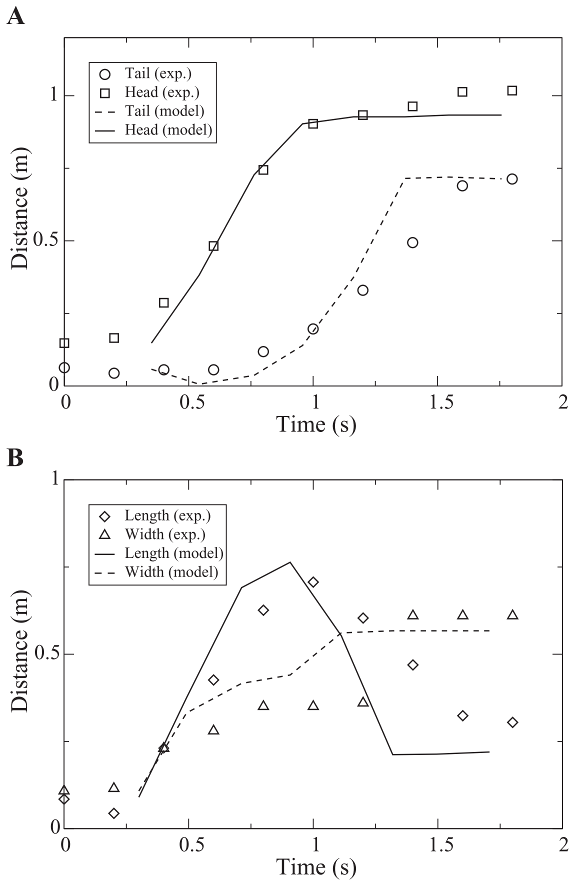


Fig. 9. Simulated and experimental observations of the front and tail of a pile of granular material sliding down a flat inclined plane at 38.5°. The propagation of the experimental and numerical flows matches well with a time offset of 0.3 s added to the numerical flow. (A) Propagation in the downslope direction as indicated by the position of flow head and tail. (B) Propagation in the cross-slope direction as indicated by the width and the extension and elongation of the pile as indicated by the difference in head and tail positions.

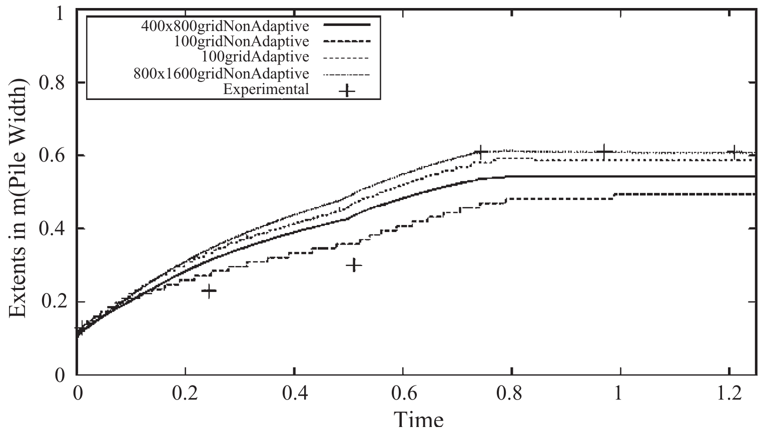


Fig. 10. Simulated and experimental observations of the width of a pile of granular material sliding down a flat plane at 38.5°. Effect of grid adaptivity is illustrated in the plot. Here, an adaptive grid with three levels of local refinement on a base grid 100×200 uniform cells yields solutions comparable to a grid with 800×1600 cells, while the nonadaptive grid solutions are further away. Experimental observations are close to the computed values from either the fine resolution grids or the adaptive grid in the later half of the flow. Early on there is significant difference between the experimental data and simulations.

The large variance resulted from differences in the test methodology. The basal maximum angle of stability was 36°. Both the internal friction angle and the internal maximum angle of stability were 37.3°. The propagation of the sand was measured by videotaping, while a horizontal grid was projected onto the plane to aid in visualization. Video frames were then grabbed with a digital frame grabber, and the sand propagation was measured directly from the frames by measuring the lateral spreading, as well as the advance of the head and tail of the flowing mass. Because of the difficulty in ascertaining the edge of the flow during

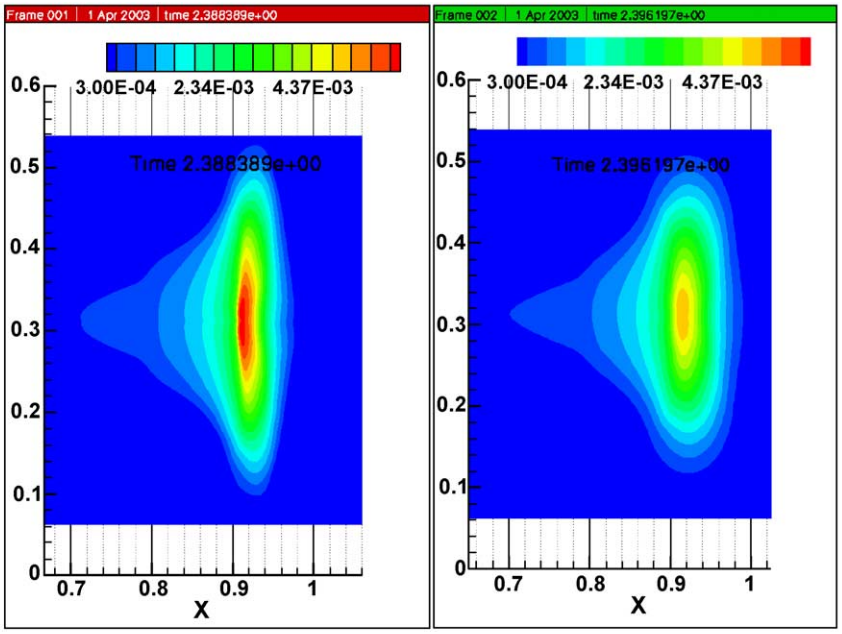


Fig. 11. Shape of the pile after 2.4 s of simulations of flow down an inclined plane starting with a 200×100 grid and three levels of local refinement on the left and a nonadapted grid on the right. The y-spread is much smaller for the coarse grid.

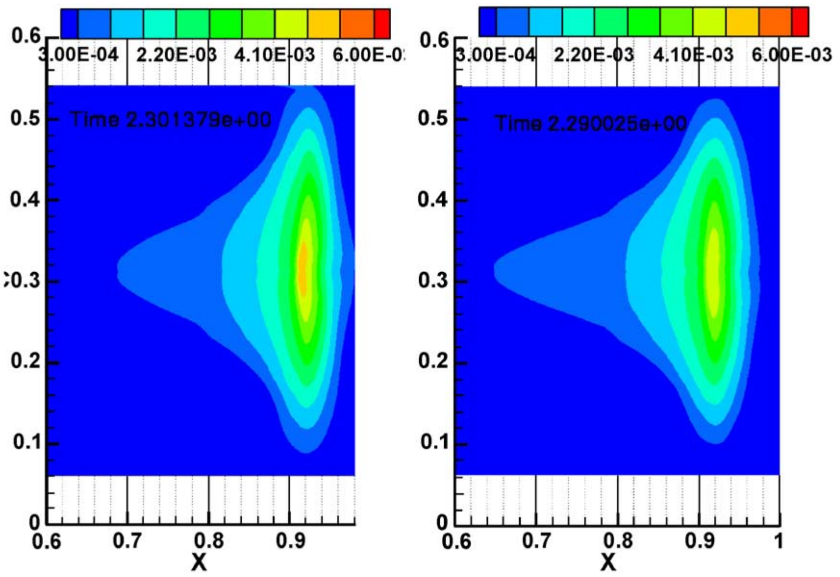


Fig. 12. Shape of the pile after 2.3 s of the simulation of flow down an inclined plane starting with a 200×100 grid and three levels of local refinement on the left and a nonadapted grid with a 800×400 grid on the right.

time steps when the material was thinly spread and because of geometrical distortions, the error in the measurements of positions of the flow is estimated to range from 1 to ~2.5 cm.

A typical experiment proceeded as follows (Fig. 8B–D). The video camera was started and the operator then filled the starting container with sand as the base was placed flush to the test plane. The container was removed with a smooth motion to avoid undue disturbance of the particles. The test mass then began propagation downslope, with the head initially moving at a noticeably greater speed than the tail, which appeared to be stationary for a short time. The sand grains spread laterally as well as downstream so that the mass rapidly attained teardrop shape. With time, this teardrop shape elongated and spread laterally, although the tail did ultimately propagate downstream.

Once on the lower test section, the particles in the head began to deposit in a teardrop shape that was noticeably less elongated in the downstream direction than the actively propagating mass. The final disposition of the mass resembled a conic section with its base on the lower test section and its apex at some height on the upper test section that depended on the slope angle of the upper test section. The only

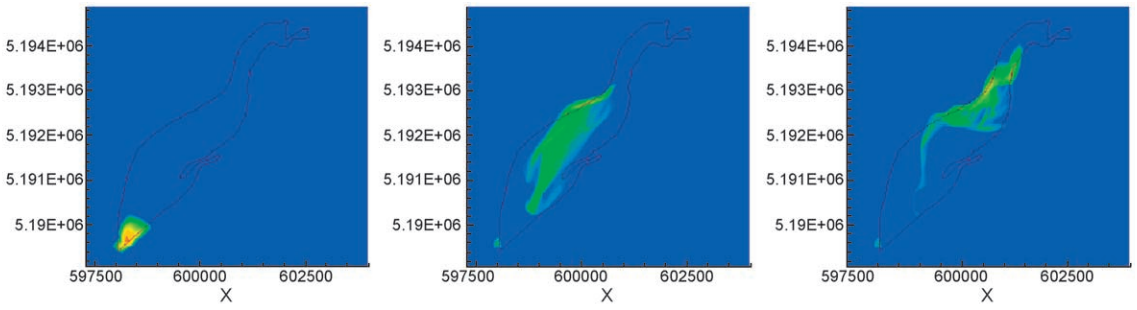


Fig. 13. Simulated flow on Little Tahoma Peak-simulated runout vs. field observations (red outline; see Sheridan et al., 2004).

exception to this geometry was in experiments at angles within the range of the basal friction angle, in which case the mass was arrested on the upper test section.

5.1.2. Simulations

The runout distance is one of the most fundamental parameters measured for granular natural flows. It is closely related to the so-called Heim coefficient, the ratio of the fall height to the runout distance for the center of mass of the pile. The final position of the flow front varies little with slope angle because it is controlled by the momentum of the flow as it reaches the base of the first slope section. The final position of the flow tail then controls the runout distance of the center of mass. Fig. 9A shows that the simulations are able to predict the position of the flow tail well over a range of slope inclinations. Fig. 9A and B show sample comparisons of the flow simulations with the experiments for the case of a ~425 g mass and slopes of 38.5°. The plots show good quantitative comparison of the evolving pile shape, speed, and runout

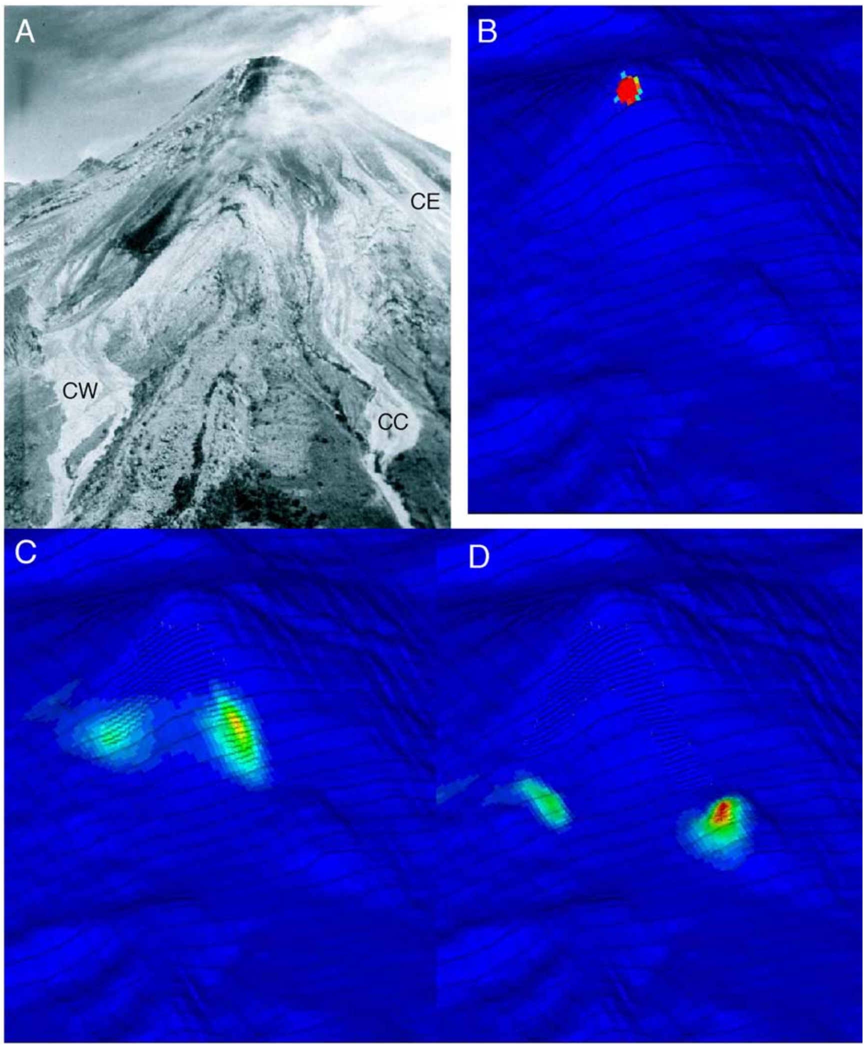


Fig. 14. (A) Aerial view of the southwest flank of Colima volcano from Saucedo et al. (in review). The lighter areas are deposits of the 1991 pyroclastic flows. (B) Initial position of the mass used in the simulation. (C) The simulated mass propagates downslope. (D) After 1000 time steps, the simulated mass in approximately at the positions shown by (CW) and (CC) in (A).

distance if an offset of +0.3 s is applied to the numerical results. We hypothesize that this offset is necessary because of unmodeled internal rearrangement of the packing of the particles that comprise the pile following withdrawal of the container, friction angle measurement errors, and an error in timing of the beginning of the experiment that must be ≤ -0.1 s. We investigate next the effect of grid adaptivity on the accuracy of the simulations. Fig. 10 shows that the computed width of the pile in the later half is well resolved and correlates well with experiments using either an adaptive grid (three levels of local refinement on a base grid of 100×200 cells) or a fine grid (800×1600 cells). The convergence of the adapted grid solution to that from a fine grid indicates that the code is consistent. The poor correlation early on indicates that the modeling is inaccurate in that flow regime. Figs. 11 and 12 show typical results at approximately the same time (2.4 s after the start of the flow) using a coarse mesh of 200×100 grid, the same initial coarse mesh and three levels of adaptivity and a fine mesh of 800×400 grid points. We noticed that all expected symmetries about a midplane are indeed observed. Secondly, the solution (especially the difficult to capture spread in the y direction) from the adaptive mesh and the fine mesh are quite similar, while the solution from the coarse mesh is quite different from the fine mesh. This indicates that our adaptivity formulation and implementation are consistent.

The testing of the model against these experiment proved to be more difficult than that against the deposits of natural flows (next subsection). On the open plane, because of the low gradient in pile height near the flow edge, it is critical to carefully define the flow edge. Flows on the open plane also possess obvious geometric characteristics that required careful definition of the coordinate system before model and data matched.

5.2. Validation: tests on real terrain

We include here two sample simulations on Little Tahoma Peak, Mt. Rainier, Washington and Volcan de Colima, Mexico. Fig. 13 shows the simulation for the rockfall avalanche on Little Tahoma Peak of 1963 with an overlay showing the extent of the deposit from field observations. Additional details of these calculations can be found in Sheridan et al. (2004).

Fig. 14 shows calculations on Volcan de Colima, Mexico, for comparison with deposits. There were thousands of rockfalls and numerous block and ash flows during the 1991–1999 eruptions of Colima Volcano, with volumes ranging from a few cubic meters to 10^6 m^3. All flows followed channels or relative topographic lows, propagating for distances

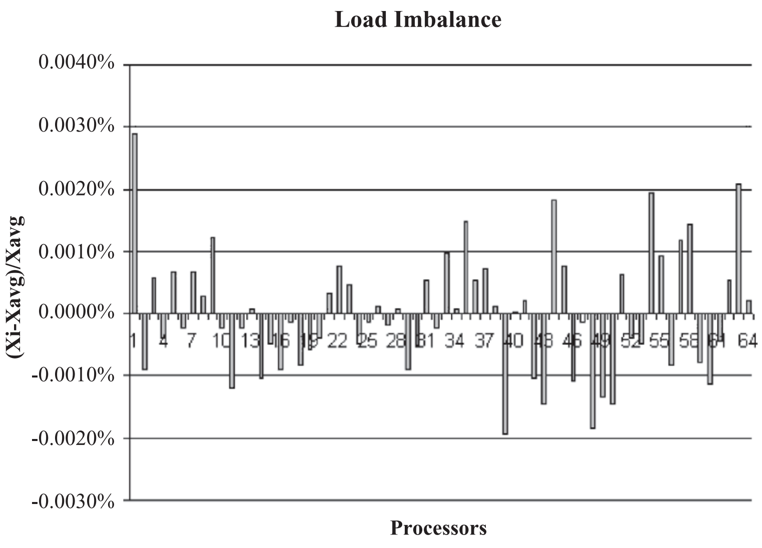


Fig. 15. Plot shows the load imbalance for each processor when using 64 processors.

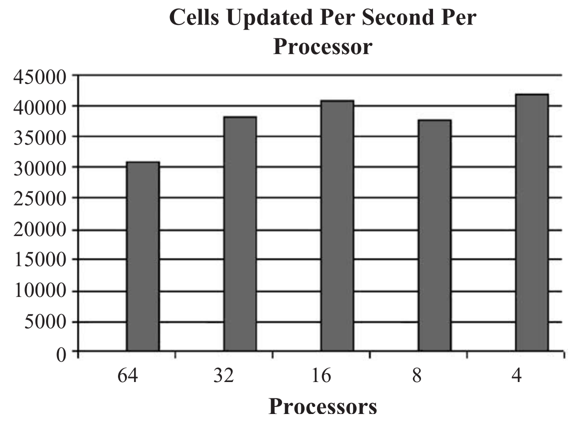


Fig. 16. Plot shows the number of cells updated per second per processor as a function of the number of processors.

up to 4 km. For a given flow volume, the TITAN2D model approximated flow path and runout distances well, but had difficulty resolving cross-slope extent in areas where the natural flow was confined within channel walls that were poorly resolved on the DEM (Rupp et al., 2003).

In both cases, the simulations provide reasonable comparisons with field observations consisting of flow paths and area covered by the deposit. Detailed matching of deposit thickness and position, and observations of flow speed are subjects for future work.

6. Code characterization

In addition to the verification and validation we described in the previous section, we have carried out a number of tests to demonstrate the computational efficiency of the code. The primary goal of this testing is to demonstrate that our codes are able to use the parallel computing hardware efficiently, and the complex procedures for implementing parallel adaptivity do not impose an undue performance penalty.

6.1. Scalability and load balance

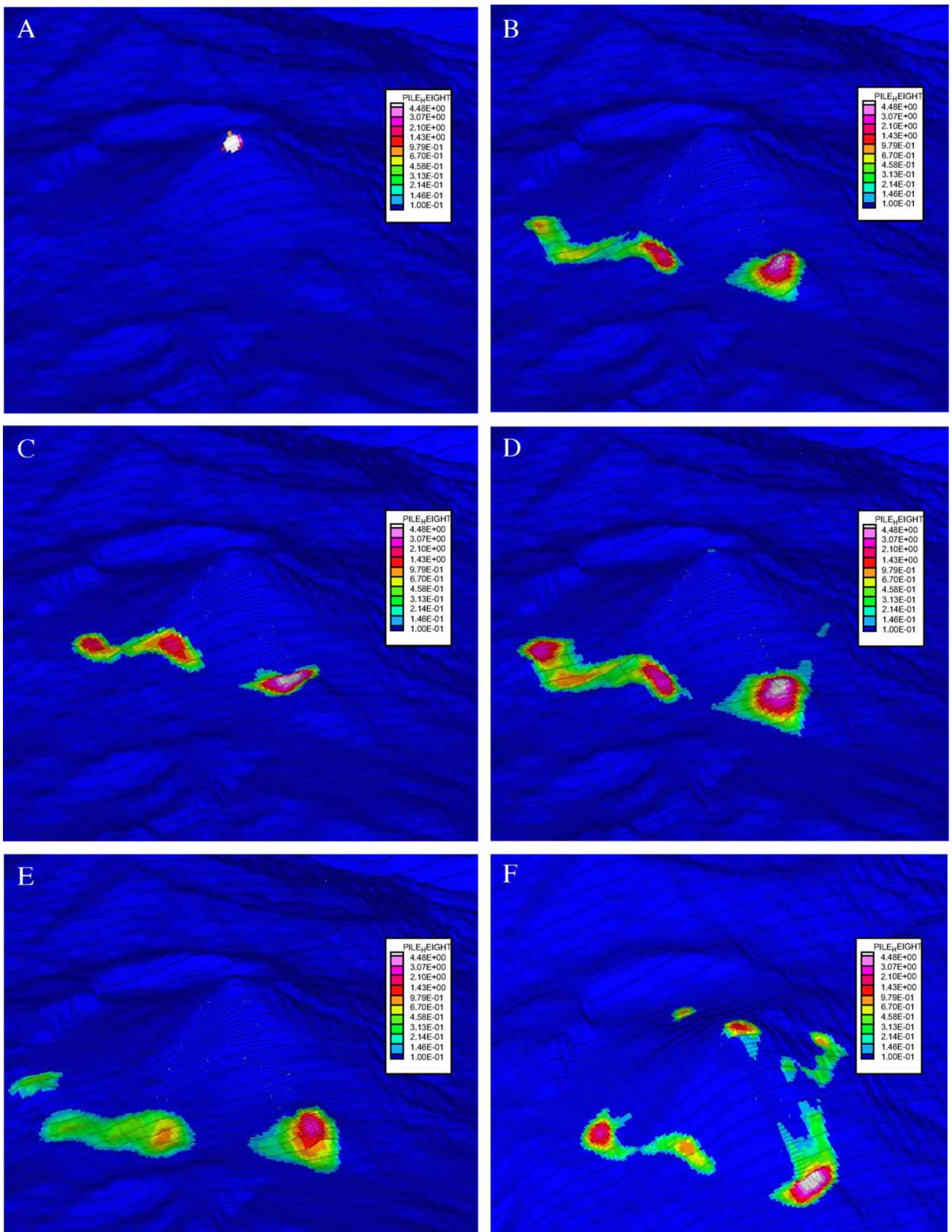
A simple measure of parallel code performance is the load balance, i.e., the problem decomposition. Ideally, each process is assigned an identical amount of work. However, in practice, we deviate from this, and one or more processors are often very lightly loaded. For the TITAN2D code, we measured this load distribution by counting the actual number of cell updates assigned to each processor i, $L(i)$ as:

$$L(i) = \frac{X_i - X_{\text{avg}}}{X_{\text{avg}}} \tag{29}$$

where X_i is the number of cell updates assigned to i, and X_{avg} is the average. Fig. 15 plots the load imbalances associated with different processors. We observe that the imbalances are extremely small.

In the next series of tests to characterize the performance of the code, we examine its scalability. Scalability of a parallel code attempts to measure the efficiency with which the algorithm/code is able to use additional processors as the problem size is increased. On distributed memory-parallel computers, the problem must be first decomposed into a series of subproblems, which can be solved concurrently, and the results must then be synchronised. The tasks of problem decomposition and synchronisation of results are thus additional overheads incurred by the parallelization. These costs also usually increase with the number of processors. Scalable parallel codes are able to control this increase—thus the amount of useful work done per processor per second remains relatively constant. Fig. 16 shows the number of cells updated per second per processor of a PC cluster on 4, 8, 16, 32, and 64 processors for a test run simulating flows on little Tahoma peak. As more processors are added, finer grids are employed maintaining the ratio of (50×50 grid cells per processors in the initial grid). We note that the count of cells updated per second per processor appears to be stable and does not degrade significantly

Fig. 17. Output of the TITAN2D model after 1400 time steps for granular flows on the southwest flank of Colima volcano, Mexico. Flows simulate block and ash flows that have occurred at the volcano during the most recent eruptive episode, which began in 1991. The simulated flows were initiated at the top of the volcano where the growth of unstable domes in the natural case results in initiation of block and ash flows. The largest of the flows in nature have propagated to the hill on which the simulated flows stop. Small changes in parameter values or initial conditions can result in significant errors in estimates of propagation distance. (A) Position of the initial pile for (B)–(E); (B) internal friction angle 30°, bed friction angle 20°; (C) internal friction angle 30°, bed friction angle 25°; (D) internal friction angle 30°, bed friction angle 20°, mass twice that of (A); (E) internal friction angle 16°, bed friction angle 15°; and (F) same as (C) but the center of the initial pile has been moved 150 m.


A
B
C
D
E
F
PILE_HEIGHT
4.48E+00
3.07E+00
2.10E+00
1.43E+00
9.79E-01
6.70E-01
4.58E-01
3.13E-01
2.14E-01
1.46E-01
1.00E-01

with increasing numbers of processors. Tests on other simulations of other sites have yielded similar results. Our analysis indicates that the slight degradation in performance is largely due to the shared access to a single GIS data set and the lower scalability of the parallel input–output system available on the commodity PC cluster-type computer used here.

6.2. Parameter studies

The simulations require us to provide parametric inputs for (a) the bed and internal friction angles, (b) location, extent, and height of initial mass, and (c) terrain topography.

In a recent study, Rupp et al. (2003) studied the effect of parameter variations on simulations using the TITAN2D code. Fig. 17 shows the results of these calculations. These results indicate that the outputs of these calculations are critically dependent on the choice of initial conditions and to a lesser extent dependent on the surface and the material characterization (the friction angles). Such insights resulting from even these preliminary calculations can be invaluable in guiding future field studies and hazard risk planning.

7. Conclusions and future work

In this paper, we have described the successful development of a new tool for simulation of geophysical mass flows. Such a tool can be used by geoscientists and public safety planners seeking to estimate the hazard risk from such flows. In associated efforts, we are also developing visualization and collaboration tools to make our systems widely accessible.

We have attempted to use the state-of-the-art computational methodologies, including parallel computing and adaptive schemes, to obtain high-quality numerical solutions. We have also integrated the simulation tool with geographic information systems to obtain accurate topological data and convenient access to other geospatial features (e.g., location of roads) that motivates specific simulations. Our software is publicly available, and we have created versions that run efficiently on inexpensive desktop personal computers and also on very high-end supercomputers. The use of grid adaptivity (and the resulting computation efficiency) enables us to run fairly accurate calculations on inexpensive desktop machines. For the best calculations though, we need to use high-end distributed memory-parallel computers.

Future efforts on the development of the TITAN2D tool will take two directions. The first will be enhance the quality of the model to enable it to deal with fluidized mixtures as opposed to the dry avalanches modeled now. As described in the parameter studies, the outputs of the simulations are dependent on choices of several parameters. Because parameters for field studies from the digital elevation maps to the initial conditions and estimates of friction angles have a large variability, we will explore systematic approaches to quantify this uncertainty.

Acknowledgements

This work was supported by NSF Grants ITR0121254. Computational resources were provided by the Center for Computational Research, University at Buffalo.

References

Berger, M., Collela, P., 1989. Local adaptive mesh refinement for shock hydrodynamics. J. Comput. Phys. 82, 64–84.

Davis, S.F., 1998. Simplified second order Godunov type methods. SIAM J. Sci. Statist. Comput. 9, 445–473.

Demkowicz, L., Oden, J.T., Rachowicz, W., Hardy, O., 1989. Toward a universal *h–p* adaptive finite element strategy: Part I. Constrained approximation and data structure. Comput. Methods Appl. Mech. Eng. 77, 9–112.

Denlinger, R.P., Iverson, R.M., 2001. Flow of variably fluidized granular material across three-dimensional terrain: 2. Numerical predictions and experimental tests. J. Geophys. Res. 106, 533–566.

Edwards, H.C., Browne, J.C., 1996. Scalable dynamic distributed array and its application to a parallel hp adaptive finite element code. Proc. POOMA '96 Santa Fe, New Mexico, http://www.acl.lanl.gov/Pooma96.

Gray, J.N.M.T., 1997. Granular avalanches on complex topography. In: Fleck, N.A., Cocks, A.C.F. (Eds.), Proceedings of IUTAM Symposium on Mechanics of Granular and Porous Materials. Kluwer Academic Publishers, pp. 275–286.

Hendrickson, B., Devine, K., 2000. Dynamic load balancing in computational mechanics. Comput. Methods Appl. Mech. Eng. 184, 485–500.

Hirsch, C., 1990. Numerical Computation of Internal and External Flows. John Wiley and Sons.

http:// www3.baylor.edu/grass/ at Baylor University accessed April, 2003.

Hutter, K., Siegel, M., Savage, S.B., Nohguchi, Y., 1993. Two dimensional spreading of a granular avalanche down an inclined plane: Part 1. Theory. Acta Mech. 100, 37–68.

Iverson, R.M., Denlinger, R.P., 2001. Flow of variably fluidized granular material across three-dimensional terrain: 1. Coulomb mixture theory. J. Geophys. Res. 106, 537–552.

Laszloffy, A., Long, J., Patra, A.K., 2000. Simple data management, scheduling and solution strategies for managing the irregularities in parallel adaptive hp finite element simulations. Parallel Comput. 26, 1765–1788.

LeVeque, R.J., 1992. Numerical Methods for Conservation Laws. Birkhauser Verlag.

Long, J., 2001. Integrated Data Management and Dynamic Load Balancing for hp and Generalized FEM, PhD dissertation, Mechanical and Aerospace Engineering Dept., University at Buffalo.

Patra, A., Laszloffy, A., Long, J., 2002. Data structures and load balancing for parallel adaptive hp finite element methods to appear in. Computers and Mathematics with Applications.

Poliquen, O., Forterre, Y., 2002. Friction laws for defense granular flows: application to the motion of a mass down a rough inclined plane. J. Fluid Mech. 453, 133–151.

Rankine, W.J.M., 1857. On a stability of loose earth. Philos. Trans. R. Soc. Lond. 147, 9–27.

Rupp, B., Bursik, M., Patra, A., Pitman, B., Bauer, A., Nichita, C., Saucedo, R., Macias, J., 2003. Simulation of pyroclastic flows of Colima Volacano, Mexico, using the TITAN2D program. European Geophysical Society 2003, Geophysical Research Abstract vol. 5. , pp. 12857.

Sagan, H., 1994. Space Filling Curves. Springer Verlag, Heidelberg.

Salomon, J., Warren, M., 1993. Parallel Hashed Oct-Trees. Proceedings of Supercomputing '93, Portland, Oregon, Nov.

Saucedo, R., Macias, J., Bursik, M., in review. Pyroclastic flow deposits of the 1991 eruption of Colima Volcano. Mexico, Bull. Volcanology.

Savage, S.B., Hutter, K., 1989. The motion of a finite mass of granular material down a rough incline. J.F.M. 199, 177–215.

Sheridan, M.F., Stinton, A.J., Patra, A., Pitman, E.B., Bauer, A., Nichita, C.C., 2004. Evaluating TITAN2D mass-flow model using 1963 Little Tahoma Peak avalanches. Mount Rainier Washington. J. Volcanol. Geotherm. Res. 139, 89–102 (this issue).

Toro, E.F., 1997. Riemann Solvers and Numerical Methods for Fluid Dynamics. Springer-Verlag.