

## Description of a Parallel, 3D, Finite Element, Hydrodynamics-Diffusion Code

A. I. Shestakov  
J. L. Milovich  
M. K. Prasad

This paper was prepared for submittal to  
NAFEMS Conference  
Newport, Rhode Island  
April 25-28, 1999

April 1, 1999



This is a preprint of a paper intended for publication in a journal or proceedings. Since changes may be made before publication, this preprint is made available with the understanding that it will not be cited or reproduced without the permission of the author.

#### DISCLAIMER

This document was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor the University of California nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or the University of California. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or the University of California, and shall not be used for advertising or product endorsement purposes.

# Description of a parallel, 3D, finite element, hydrodynamics-diffusion code\*

A. I. Shestakov<sup>†</sup>    J. L. Milovich<sup>‡</sup>    M. K. Prasad<sup>§</sup>

## Abstract

We describe a parallel, 3D, unstructured grid finite element, hydrodynamic diffusion code for inertial confinement fusion (ICF) applications and the ancillary software used to run it. The code system is divided into two entities, a controller and a stand-alone physics code. The code system may reside on different computers; the controller on the user's workstation and the physics code on a supercomputer. The physics code is composed of separate hydrodynamic, equation-of-state, laser energy deposition, heat conduction, and radiation transport packages and is parallelized for distributed memory architectures. For parallelization, a SPMD model is adopted; the domain is decomposed into a disjoint collection of subdomains, one per processing element (PE). The PEs communicate using MPI. The code is used to simulate the hydrodynamic implosion of a spherical bubble.

## 1 Introduction

This paper describes the ICF3D code system, initially written to simulate inertial confinement fusion (ICF) experiments generating high temperature plasmas. The system consists of two separate entities, a controller and the physics code itself. However, in the following, ICF3D refers to the physics code while *controller* refers to the code with which the user interacts. The controller is used to define problems, construct meshes, visualize results, and for runs on massively parallel platforms (MPP), to partition the domain into a collection of subdomains, one per processing element (PE). The controller runs on the user's workstation. ICF3D, on the other hand, may reside on another computer; possibly a remotely

---

\*Work performed under the auspices of the U.S. Department of Energy by the Lawrence Livermore National Laboratory under contract number W-7405-ENG-48.

<sup>†</sup>Lawrence Livermore National Laboratory, POB 808 L-38, Livermore, CA 94550

<sup>‡</sup>Lawrence Livermore National Laboratory, POB 808 L-561, Livermore, CA 94550

<sup>§</sup>Lawrence Livermore National Laboratory, POB 808 L-38, Livermore, CA 94550

located MPP. This design allows maximum utilization of a supercomputer as a number cruncher by not taxing it with tasks more suited to workstations.

Once the problem is specified (proper input files created, etc.), control is passed to ICF3D. This requires establishing a link over the network between the user's interactive session with the controller and the physics code. After passing control, ICF3D performs the calculation then returns the results to the controller. Control passing is made via UNIX socket programming. The controller and ICF3D are written in different languages in order to take advantage of features specific to them. Controller modules are themselves written in several languages, Python [1], FORTRAN, and C++. The physics code, on the other hand, is entirely written in C++.

The remainder of this paper is organized as follows: The following section describes the controller. Section 3 gives a brief overview of the physics code, its modules, and the parallelization methodology. Only ICF3D is parallelized since that is the computationally intensive part. Section 4 contains a simulation of a hydrodynamic implosion and § 5 is a summary.

## 2 ICF3D controller

ICF3D's execution is controlled via the object oriented (OO) scripting language Python [1] which also provides a very limited *steering* capability. Steering allows the user to directly interact with the code modules and data. Unfortunately, this requires that the controller and physics code be tightly coupled, something difficult to fulfill on MPP. In addition, since controllers use interactive languages, a tightly coupled system puts the additional burden of having those languages on the supercomputer. Consequently, for our limited steering, we have chosen a loosely coupled model that provides for flexibility in running the physics code on a variety of architectures without requiring coupling the physics code to the controller thereby obviating portability problems. Our approach – ICF3D controlled by Python running on a local workstation – evokes a distributed computing model, relegates supercomputers exclusively for computations and leaves the problem generation and post-processing chores to the desktop.

By taking advantage of Python's extendability we have written scripts that read the user's (Python) input file that describes the problem, generate and partition the mesh, direct code execution to the machine of choice (by dispatching appropriate input files and recovering the output files), and analyze the results. In addition, during execution we can query the code's progress, e.g., its time cycle and time step, create restart files, create output visualization files, and halt and restart the calculation

at will. While this is not as flexible as a truly steerable code, it has given us an invaluable means to develop and debug ICF3D and provided easy access to the number cruncher of choice. The interface is the same regardless of the computing engine, MPP or uniprocessor. Consequently, the parallelization is completely transparent to the user. Once control is returned to the user, results from an MPP are indistinguishable from those from a uniprocessor.

One of the controller's principal responsibilities is mesh generation. Since the generation of unstructured 3D finite element (FE) meshes is a difficult task, the controller provides the option of generating two types. For unstructured, Cartesian, tetrahedral grids, the controller uses the LaGriT code [2] of the Los Alamos National Laboratory and for MPP runs, LaGriT is linked to the METIS code [3] of Karypis and Kumar to partition the mesh into a collection of subdomains, one per PE. Python scripts and C++ functions interface with the FORTRAN LaGriT code and the METIS code, written in C.

The second type of mesh is logically structured; each vertex corresponds to a  $(k, l, m)$  triplet of positive integers. Such meshes may be constructed in any of the three coordinate systems allowed in ICF3D, Cartesian, cylindrical, or spherical. For MPP runs, these meshes are easily decomposed by slicing along any of the  $(k, l, m)$  logical planes. An additional flexibility is allowed whereby the user may describe the mesh in one coordinate system and instruct the controller to generate it and run the problem in another. For example, with one function call, the user discretizes a sphere into uniform radial concentric spherical shells with uniform polar, and azimuthal angles, and gets the grid in Cartesian coordinates. In this case, the controller automatically removes duplicate copies of the center point and those along the axis and constructs cells of all admissible types: tetrahedra, pyramids, prisms and hexahedra. After the conclusion of the run, the controller reformats the results into the user's original coordinates which eases visualization of the results.

We emphasize that the duality of meshes, structured and unstructured, is limited to the controller. The physics code is based on only unstructured grids. The ICF3D input file describing the mesh is in the Advanced Visual System (AVS) Unstructured Cell Data (UCD) format [4]. For MPP runs, the format is extended by tagging each cell with the PE number which "owns" it, and both cells and vertices are assigned unique "global" indices.

### 3 ICF3D physics code

ICF3D stems from an effort to apply the discontinuous finite element (DFE) method to solve the Euler equations for compressible hydrodynamics. For early work on DFE, we refer to works of Cockburn et al, [5],

[6], and [7]. Details of the ICF3D hydrodynamic scheme and early results on 3D problems are given by Kershaw et al [8].

After the initial successes on hydrodynamic test problems, the code was expanded to include equations-of-state of real materials, laser energy deposition, heat conduction, and radiation transport. The first description of ICF3D as a physics design code was given by Shestakov et al [9]. The code modules were later parallelized to take advantage of the promise of modern supercomputers with hundreds, eventually thousands of PEs. Shestakov and Milovich [10] describe the parallelization of the hydrodynamic and diffusion modules while Shestakov et al [11] describe the entire parallelization strategy, including the laser ray tracing module. The parallelization does not inhibit portability; the same code runs on uniprocessors, and both types of MPP: distributed as well as shared memory architectures.

ICF3D solves equations for the conservation of mass, momentum, and total matter energy densities,  $\rho$ ,  $\rho\mathbf{v}$ , and  $E$ :

$$\begin{aligned} (1) \quad & \partial_t \rho + \nabla \cdot \mathbf{F}_\rho = 0, \\ (2) \quad & \partial_t(\rho\mathbf{v}) + \nabla \cdot \mathbf{F}_{\rho\mathbf{v}} = \rho\mathbf{g}, \\ (3) \quad & \partial_t(E) + \nabla \cdot \mathbf{F}_E = \rho\mathbf{g} \cdot \mathbf{v} + H_\varepsilon + S_\varepsilon + K_{r\varepsilon}. \end{aligned}$$

Equation (3) is coupled to the transport (diffusion) equation of the radiation field energy density

$$(4) \quad dE_r/dt = \nabla \cdot D_r \nabla E_r - K_{r\varepsilon}.$$

In (1), (2), and (3),  $\mathbf{F}_i$  denotes the flux of  $i$ , i.e.,  $\mathbf{F}_{\rho v_x} = \rho v_x \mathbf{v} + p$ , where  $v_x$  is the  $x$  velocity component and  $p$  is the pressure. In (2) and (3),  $\rho\mathbf{g}$  represents an external force density. In (3),  $\varepsilon$  denotes the matter internal energy, and  $H_\varepsilon$  is the negative divergence of the heat flux,

$$H_\varepsilon = \nabla \cdot D_\varepsilon \nabla T,$$

where  $D_\varepsilon$  is the thermal coefficient, and  $T$  is the matter temperature. The term  $S_\varepsilon$  is an external source of energy (e.g., due to a laser),  $K_{r\varepsilon}$  denotes the radiation-to-matter coupling. In (4),  $d/dt$  is the Lagrangian derivative, and  $D_r$  is the diffusion coefficient of the radiation field.

The equations are solved using operator splitting. The time cycle begins by solving the hyperbolic conservation laws, i.e., (1) to (3) with the rhs, except for the  $\rho\mathbf{g}$  terms, set to zero. Next, material properties such as the specific heat,

$$\partial\varepsilon/\partial T|_\rho = c_v$$

are computed. If the problem involves laser energy deposition, that module then tracks the beams through the domain and computes the

energy source  $S_\varepsilon$ . After determining  $S_\varepsilon$ , the heat conduction module advances the equation,

$$(5) \quad c_v \partial_t T = H_\varepsilon + S_\varepsilon .$$

Heat conduction is followed by the final operation, radiation transport and coupling to the matter temperature. This effect is simulated by simultaneously advancing (4) and

$$c_v \partial_t T = K_{r\varepsilon} .$$

The radiation-to-matter coupling is given by,

$$K_{r\varepsilon} = -c\rho\kappa_P(S(T) - E_r) ,$$

where  $c$  is the speed of light,  $\kappa_P$  is the Planck averaged opacity, and  $S(T) = (4\sigma/c)T^4$  is the spectral average of the Planck function,  $\sigma$  is the Stefan-Boltzmann constant.

Operator splitting lets us compartmentalize the operations and thereby reuse modules written for simpler equations. For example, the original ICF3D hydrodynamic scheme is unchanged. A separate laser energy deposition module of Kaiser et al [12] computes  $S_\varepsilon$ , and a diffusion module discretizing equations of the type,

$$(6) \quad g\partial_t u = \nabla \cdot D\nabla u - au + s$$

advances (4) and (5).

ICF3D is written in the OO programming language C++ and the pdes are discretized using FE. These two features complement each other nicely. For example, in both the DFE and standard FE methods, one computes integrals of the type

$$(7) \quad \int_C f dV$$

where,  $C$  denotes a computational cell. Thus, once one cell integrator is written, it may be used (called) by other modules.

The modularity extends to a low level. Using the C++ lexicon, the ICF3D mesh is described in terms of cell, face, and vertex *objects*, made at the start of the run by the *constructor* for the appropriate *class*. The concept of *inheritance* is used by allowing different cell types: tetrahedra, pyramids, prisms, and hexahedra. Similarly, two types of faces arise: triangular and quadrilateral. Since the mesh is allowed to move (the hydrodynamic scheme is ALE), the cells are allowed to distort and the quadrilateral faces need not be planar. When integrating a function  $f$  over the entire domain, we cycle through the cells and use (7). Each cell knows its type and calls the appropriate Gaussian formula to approximate the integral. A similar procedure is used for integrals over cell faces.

### 3.1 Numerical details

Except for the novel DFE scheme for the hydrodynamics, ICF3D uses a standard Galerkin formulation for the spatial discretization of parabolic equation. Equations such as (6) are advanced using backward Euler except the coefficients  $g$ ,  $D$ , and  $a$  are fixed at the previous time level. The function  $u$  is given in terms of the usual piecewise “linear” basis functions,

$$u(x, t^n) = \sum_j \phi_j(x) u_j^n$$

The number of unknowns equals the number of vertices and we use an isoparametric mapping to compute integrals such as (7). Hence, on tetrahedra,  $u$  is linear, while on hexahedra,  $u$  has a trilinear representation. The discretization leads to large, sparse, SPD linear systems. Since the diffusion equations advance inherently positive quantities, we lump everything but the transport term in order to obtain an M matrix [9]. On uniprocessors, the systems are solved using ICCG. On MPP, we use an ICCG variant in which the preconditioner neglects coupling across inter-PE boundaries [10], [11].

The DFE hydrodynamic scheme differs from the above since it allows for discontinuous functions. The scheme is compact. Instead of integrating over the entire domain, the hydrodynamic system is multiplied by a test function, and integrated only over a cell. The flux divergence term is integrated by parts yielding two integrals,

$$(8) \quad \int_C \phi_i \nabla \cdot \mathbf{F} dV = \int_{\partial C} \phi_i \mathbf{F} \cdot \mathbf{n} dA - \int_C \nabla \phi_i \cdot \mathbf{F} dV .$$

The first integral on the rhs of (8) is a sum of the contributions from each cell’s face and represents the flux across the face from the cells on either side. Thus, the rhs is computed by first looping over the faces and solving the appropriate Riemann problems, then looping over cells [8]. The discontinuity arises since hydrodynamic variables such as  $\rho$  vary within each cell, but are discontinuous across the faces. For a first order scheme, this reduces to cell-centered variables, and is an extension of the method of Godunov [13]. The ICF3D second order scheme, has hefty storage requirements. Because of the allowed discontinuities, for each hydrodynamic variable, the number of unknowns equals the number of cells times the number of vertices of each cell. Thus, for a mesh consisting of  $N_c$  similar cells, with  $N_v$  vertices per cell, since both the conserved variables:  $\rho$ ,  $(\rho \mathbf{v})$ , and  $(\rho E)$  and the fundamental ones:  $\rho$ ,  $\mathbf{v}$ , and  $p$  and their cell averages are stored, the hydrodynamic scheme alone requires

$$(9) \quad (10 \times N_v + 10) \times N_c$$

locations to store one time level. Even though (9) could be reduced by not storing the nodal  $\rho$  variable twice and by using nodal values to

recompute the cell averages (needed in several places in the code), the storage requirement is still large.

### 3.2 Parallelization

Since this subject has already been discussed in [10] and [11], here we present only an overview. ICF3D parallelizes by decomposing the domain into a collection of disjoint subdomains, one per PE. Each PE receives a description of its subdomain (comprised by its owned cells) and a surrounding one cell-wide layer of *ghost* cells owned by other PEs. There is no restriction on the shapes of the subdomains (SDs) and indeed with METIS and unstructured grids, the SDs have ragged boundaries.

The PEs communicate using functions from the MPI message passing library. The actual calls are made by *member functions* of special message passing objects (MPO) which ICF3D constructs at start-up.

Since ICF3D has both a cell-centered scheme (hydrodynamics) and vertex centered (diffusion), there are two distributed mesh entities, cells and vertices. The input files tag each cell with the number of the PE which owns the cell. Vertex PE assignment is done by ICF3D during the initialization phase. This procedure itself requires some message passing since the algorithm that assigns a vertex to a PEs is based on a survey of all cells attached to the vertex.

The domain decomposition strategy and the ICF3D code modules lead to four different parallelization difficulties:

1. Embarrassingly parallel functions such as the cell-based equation-of-state calls which do not require message passing.
2. Straightforward parallelization of the temporally explicit hydrodynamic scheme in which the “difference” stencil extends only over immediate neighbor cells. This is resolved by the ghost cells which store the latest information message-passed to the PE.
3. Functions requiring global communication, e.g., solution of the large, sparse, unstructured linear systems arising from the discretizations of the diffusion equations. The systems are solved using preconditioned CG [10].
4. Unpredictable point-to-point communication arising in the parallelization of the laser energy deposition module in which each laser beam is discretized into a collection of rays or particles which traverse the mesh and deposit energy on the cells. The parallelization consists of collecting the rays as they cross the SD boundary and passing them to the PE which owns the neighboring cells.

## 4 Results

To illustrate one aspect of ICF3D’s capability, we present a problem using only the hydrodynamic module. Although the problem is simple, in fact, has spherical symmetry, we simulate in both 1D spherical mode<sup>1</sup> with a structured grid and in 3D, Cartesian mode on a unstructured tetrahedral grid to demonstrate ICF3D’s versatility and robustness

Since the problem has simple physics and uses only one module, readers interested in the performance of other code modules should consult the following references: Initial performance of the parallelization of the hydrodynamic and linear system solver appear in [9]. Results on a problem coupling hydrodynamics to non-linear heat conduction appear in [14] and [10]. In ref. [14], ICF3D was run in 1D spherical mode in both Lagrangian and ALE modes in which the grid points were restricted to move at half the fluid velocity. In ref. [10], the same problem was run on an unstructured tetrahedral grid in Cartesian geometry on a parallel machine using 64 PEs and results compared favorably to the finely meshed spherical runs. In ref. [11], the hydrodynamic, heat conduction, and laser deposition modules were combined to simulate the implosion of a spherical gas bubble driven by twelve laser beams centered on the vertices of an icosahedron. Lastly, a simulation of an ICF capsule which uses real material equations-of-state, hydrodynamics, realistic heat conduction, and radiation transport appears in [15]

Here, the problem consists of an imploding, spherical shock wave of infinite strength. Even though the problem is spherically symmetry, it is of interest since it is so difficult for 3D codes to maintain symmetry of the converging wave. In addition, codes limited to structured grids have the added burden of carrying the resolution (number of cells) needed on the outside of the sphere on the waist to the pole and center, e.g., if discretizing a sphere into uniform polar and azimuthal cells. This brings the additional complication of restrictive time steps due to the Courant condition as the wave reaches the center.

Unstructured grids relieve these complications since with such meshes one freely puts resolution where needed. In Fig. 1 we display the computational domain, a tetrahedral wedge of a regular icosahedron bounded by the sphere of radius  $r_0$ , the two azimuthal planes:  $\phi = \pm\pi/5$ , and the plane intersecting the origin and the points:  $(\theta, \phi) = (\theta_0, \pm\pi/5)$  where  $\cos\theta_0 = 1/\sqrt{5}$ . Such a domain allows us to easily view the progression of the implosion, maintain resolution where its desired, yet allow enough freedom for the implosion to lose symmetry with a poor numerical scheme. Nevertheless, even with the unstructured grid, the problem is relatively large. Equation (9) implies that there are nearly 300,000 unknowns.

---

<sup>1</sup>In 3D spherical coordinates with one cell in each of the angular direction

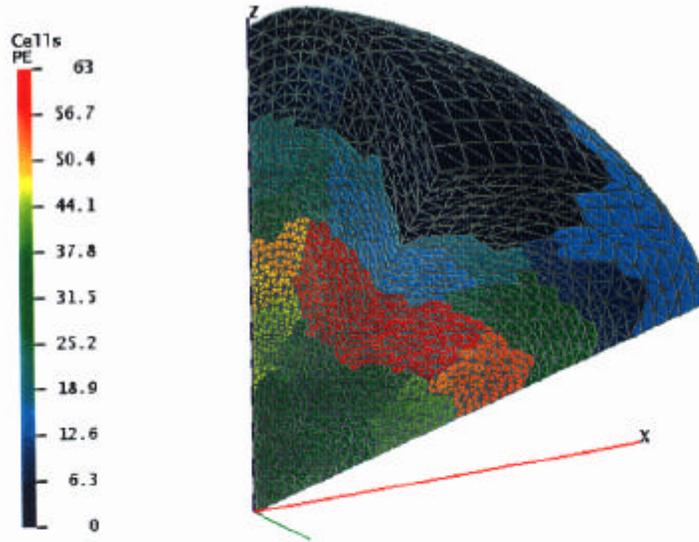


FIG. 1. 3D Domain of ideal gas implosion problem. Shading designates PE numbers. Grid consists of 28,208 tetrahedra (50 radial cells), 5791 points, and 58,455 faces.

The implosion results when a spherical gas bubble, initially of radius  $r_0 = 1.0$ , is subjected to a boundary pressure of magnitude  $p_b = 4/3$ . The gas equation-of-state is specified using,

$$p = (\gamma - 1) \rho \varepsilon \quad \text{where } \gamma = 5/3$$

The initial conditions mimic a cold, quiescent gas, initially of unit density,

$$\rho_0 = 1 \quad \text{and} \quad p_0 = \mathbf{v}_0 = 0$$

The gas motion results from the difference between  $p_b$  and  $p_0$ . Initially, the solution is similar to one with slab symmetry. The boundary pressure drives a shock into the gas which at first moves with speed  $\dot{r}_s \approx -4/3$ . Since the shock is of infinite strength, immediately behind the shock,

$$\rho = \frac{\gamma + 1}{\gamma - 1} \rho_0 \quad \text{and} \quad v_r = -1$$

If the problem stayed slab-symmetric, the shock would traverse a unit distance in a time of 0.75 secs. However, because of the converging geometry, the shock reflects off the origin at an earlier time which a 1D simulation places at  $t \approx 0.57$ . In Fig. 2 we display  $\rho$  at  $t = 0.56, 0.58$ , and  $0.6$  for a finely meshed, spherical 1D run and also display  $\rho$  at  $t = 0.6$  for two coarser discretizations. Results show that the coarsest mesh (50

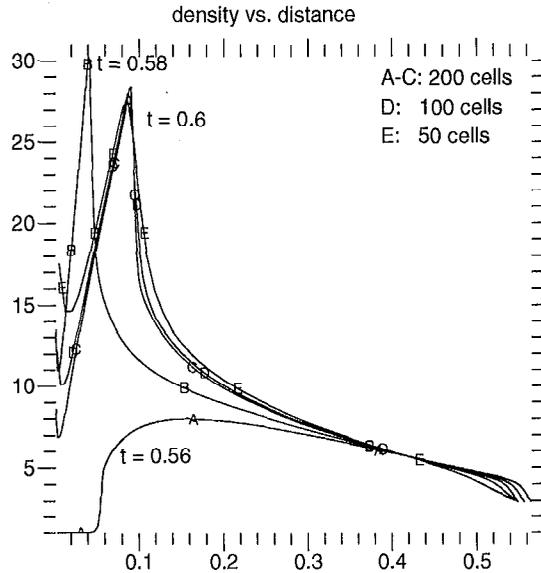


FIG. 2. *Ideal gas implosion;  $\rho$  vs.  $r$ . Lagrangian, 1D simulations with initially uniform cell widths. Curves A, B, and C are at  $t = 0.56$ ,  $0.58$ , and  $0.6$  resp. and use 200 cells. Curves D and E are at  $t = 0.6$  and use 100 and 50 cells resp.*

cells) suffices to obtain relatively good accuracy, especially if we compare the shock's position.

The 3D simulation is run on an unstructured grid discretizing the icosahedral wedge depicted in Fig. 1 in which the shading corresponds to the PE number. The simulation was done on 64 PEs of the LLNL IBM SP2 and ran until  $t = 0.9$  at which time the reflected shock had interacted with the incoming outer boundary. Figure 3 displays a side-on view of  $\rho$  at  $t = 0.6$

A comparison between Figs. 2 and 3 shows that  $\max(Z)$  and the shock position are very similar and the  $\min(\rho)$  also agree while the  $\max(\rho)$  are within 6% of each other since in the 1D result with 50 cells,  $\max(\rho) = 27.46$ . Lastly, Fig. 3 shows the code's ability to maintain spherical symmetry, despite running on the asymmetric tetrahedral grid.

## 5 Summary

We have presented a general overview of the ICF3D code system, a parallel, 3D, unstructured-grid FE code linked to an interactive controller. The code system gives the user great flexibility by allowing the physics computing engine to run on the workhorse computer of choice while having the comforts of an interactive initialization of problems and interpretation of results at a desktop workstation. In designing the system, we strived for portability. All that is required for ICF3D is

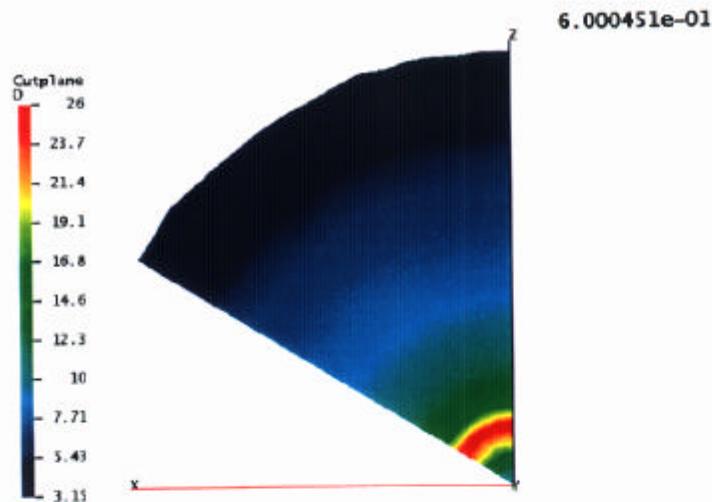


FIG. 3. *Ideal gas implosion; side-on view of density;  $t = 0.6$  sec;  $\max(Z) = 0.563$*

a C++ compiler and a UNIX operating system (OS) to enable linking across the network. The desktop, which runs the controller, needs more software support, specifically a Python interpreter, the ability to compile FORTRAN, C, and C++ codes, and again a UNIX OS to link with ICF3D.

ICF3D was designed to simulate ICF experiments and is written in modular form with separate hydrodynamic, equation-of-state, laser energy deposition, heat conduction, and radiation transport packages. The modules may be turned on and off more-or-less at will which allows us to run problems in which only some of the physics is relevant.

However, since we use robust numerical methods, ICF3D is extendable to other problems. For example, since both the heat conduction and radiation transport packages advance diffusion equations, non-linear elliptic equations may also be solved if the elliptic operator has the same form as the rhs of (6).

## References

- [1] See <http://www.python.org>
- [2] See <http://www.t12.lanl.gov/~lagrit>
- [3] G. KARYPIS and V. KUMAR, A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs, *SIAM J. Sci. Comp.*, **20**, 1 (1998) pp. 359-392. The METIS code is available on the web at: <http://www-users.cs.umn.edu/~karypis/metis/metis/main.html>
- [4] *AVS Developer's Guide*, Advanced Visual Systems, Inc., Release 4, May 1992, p. E-1, 300 Fifth Ave., Waltham MA 02153.

- [5] B. COCKBURN and C.-W. SHU – TVB Runge-Kutta Local Projection Discontinuous Galerkin Finite Element Method for Conservation Laws II: General Framework, Math. Comput. 52, 146 (1989) 411.
- [6] B. COCKBURN, S.-Y. LIN, and C.-W. SHU – TVB Runge-Kutta Local Projection Discontinuous Galerkin Finite Element Method for Conservation Laws III: One-Dimensional Systems, J. Comput. Phys. 84 (1989) 90.
- [7] B. COCKBURN, S. HOU, and C.-W. SHU – TVB Runge-Kutta Local Projection Discontinuous Galerkin Finite Element Method for Conservation Laws IV: The Multidimensional Case, Math. Comput. 54, 146 (1990) 545.
- [8] D.S. KERSHAW, M.K. PRASAD, M.J. SHAW, and J.L. MILOVICH, 3D Unstructured mesh ALE hydrodynamics with the upwind discontinuous finite element method, Comp. Meth. in App. Mech. Engin., 158 (1998) 81.
- [9] A.I. SHESTAKOV, M.K. PRASAD, J.L. MILOVICH, N.A. GENTILE, J.F. PAINTER, and G. FURNISH – The Radiation-Hydrodynamic ICF3D Code , Lawrence Livermore National Laboratory, Livermore, CA, UCRL-JC-124448, (1997), to appear in Comp. Meth. in App. Mech. Engin.
- [10] A.I. SHESTAKOV and J.L. MILOVICH – Parallelization of an Unstructured Grid, Hydrodynamic-Diffusion Code, in Solving Irregularly Structured Problems in Parallel, 5th International Symposium, IRREGULAR'98 Berk., Ca., USA, Aug. 9-11, 1998 Proceedings, A. Ferreira, J. Rolim, H. Simon, S.-H. Teng (Eds.) Lecture Notes in Computer Science 1457 Springer.
- [11] A.I. SHESTAKOV, J.L. MILOVICH, and D.S. KERSHAW – Parallelization of an Unstructured Grid, Hydrodynamic-Diffusion Code, submitted to SIAM News, contact: [shestakov@llnl.gov](mailto:shestakov@llnl.gov)
- [12] T.B. KAISER, J.L. MILOVICH, A.I. SHESTAKOV and M.K. PRASAD, A New Laser Driver for ICF Physics Modeling Codes on Unstructured 3D Grids, Bull. Am. Phys. Soc., 43, 1900 (1998).
- [13] S.K. GODUNOV, Mat. Sbornik 4 (1959) 271 [translated as JPRS 7225 (U.S. Dept. of Commerce, Washington DC, 1960)].
- [14] A.I. SHESTAKOV, Time Dependent Simulations of Point Explosions with Heat Conduction, Lawrence Livermore National Laboratory, Livermore, CA, UCRL-JC-132414, (1998), to appear in Phys. Fluids A.
- [15] A.I. SHESTAKOV, J.L. MILOVICH, M.K. PRASAD, and T.B. KAISER, Combining Cell and Node Centered Methods for Parallel, 3D, Unstructured-Grid Radiation-Hydrodynamic Codes, to be submitted to J. Comp. Phys., contact: [shestakov@llnl.gov](mailto:shestakov@llnl.gov)