UCRL-JC-130862
PREPRINT

Parallelization of an Unstructured Grid, Hydrodynamic-Diffusion Code

Alek I. Shestakov and Jose L. Milovich

This paper was prepared for submittal to the
Fifth International Symposium on Solving Irregularly Structured Problems in Parallel
Berkeley, California
August 9 - 11, 1998

5/20/98

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.

Lawrence Livermore National Laboratory

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.

Parallelization of an Unstructured Grid, Hydrodynamic-Diffusion Code *

Aleksei I. Shestakov and Jose L. Milovich

Lawrence Livermore National Laboratory
Livermore CA 94550

Abstract. We describe the parallelization of a three dimensional, unstructured grid, finite element code which solves hyperbolic conservation laws for mass, momentum, and energy, and diffusion equations modeling heat conduction and radiation transport. Explicit temporal differencing advances the cell-based gasdynamic equations. Diffusion equations use fully implicit differencing of nodal variables which leads to large, sparse, symmetric, and positive definite matrices. Because of the unstructured grid, the off-diagonal non-zero elements appear in unpredictable locations. The linear systems are solved using parallelized conjugate gradients. The code is parallelized by domain decomposition of physical space into disjoint subdomains (SDs). Each processor receives its own SD plus a border of ghost cells. Results are presented on a problem coupling hydrodynamics to non-linear heat conduction.

1 Introduction

We describe the parallelization of ICF3D [1], a 3D, unstructured grid, finite element code written in C++. The ICF3D mesh consists of an arbitrary collection of hexahedra, prisms, pyramids, and/or tetrahedra. The only restriction is that cells share like-kind faces. We parallelize by first decomposing the physical domain into a collection of disjoint subdomains (SDs), one per processing element (PE). The decomposition tags each cell with the PE number which will "own" it. A collection of cells owned by a PE comprises the PE's SD. A cell owned by another PE and which shares at least one vertex with an owned cell is called a *ghost* cell. Each PE receives a terse description of only its SD plus a layer of ghost cells. The decomposition is especially suited to distributed memory architectures (DMP) such as the CRAY T3E. However, it may also be used on shared memory processors (SMP). ICF3D is portable; it runs on uniprocessors and massively parallel platforms (MPP). A single program multiple data (SPMD) model is adopted. In ICF3D, three levels of parallelization difficulties arise:

1. Embarrassingly parallel routines such as equation-of-state function calls which do not require any message passing since each cell is owned by only one PE.

* 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.

2. Straightforward parallelization of temporally explicit algorithms such as the hydrodynamic package.
3. Difficult problems requiring global communication, e.g., the solution of the linear systems which are the discretization of the diffusion equations.

The mesh consists of cell, face, and vertex *objects.* Since the input files assign PE ownership only to the cells, some faces and vertices lie on inter-PE "boundaries." Physics modules such as the one advancing the hydrodynamic equations, which update cell-centered data, also compute face-centered quantities, e.g., fluxes. If a face lies on an inter-PE boundary, the flux across it is computed by the two PEs which own the cells on either side. Fluxes are computed after the PEs exchange appropriate information to ensure that both obtain the same flux. Modules such as the diffusion solver update vertex-centered data. These equations are advanced by standard finite element (FE) techniques which lead to large, sparse, symmetric positive definite (SPD) linear systems that are solved using preconditioned conjugate gradient (CG) methods. The assembly of the linear systems requires integrating over cells, i.e., a cell-centered computation, and is done by each PE over only its owned cells. However, once the linear system has been completely assembled and properly distributed among the PEs the calculation is vertex-centered. A principal result of this paper shows how to assemble and solve such systems with the restriction that each PE sees only its SD and the surrounding ghost cells.

The PEs communicate using message passing function libraries. Two types are available, MPI and the native CRAY SHMEM libraries. The former is portable; it is available on both SMPs and DMPs.

In ICF3D two types of communication arise, global and point-to-point (PtP). An example of the former is the calculation of a new time step Δt. First, each PE loops through its cells or vertices and finds an acceptable value, then a *global reduction* function forms a single scalar and distributes it to all PEs. In PtP communication, PE[i] exchanges messages only with those PEs that own its ghost cells. For such exchanges, ICF3D relies on special "message passing objects" (MPO) which are constructed during the initialization of the run. The MPO *constructor* relies on mesh connectivity information that ICF3D computes as it builds the mesh objects. The actual calls to MPI (or SHMEM) functions are made by the MPO *member functions.*

In the next section, we discuss what is required for initialization. In Sect. 3 we describe the parallelization techniques required by the hydrodynamic module. Section 4 deals with analogous issues for the diffusion packages. Section 5 displays results on a problem which couples the explicit hydrodynamic and implicit diffusion schemes. Concluding remarks appear in Sect. 6.

2 Initialization

Before describing the procedures specific to MPP simulations, we discuss those required to initialize any run, even those for uniprocessors. The input and output files describe the mesh in the AVS UCD format [3] which uses two lists. The first, of length N_v, is the indexed list of vertices. Each vertex is specified by a 3-tuple – the three coordinates of the vertex.[1] The second list, N_c long, is the indexed list of cells. Each entry in the cell list contains a string denoting the cell's type, e.g., hex and a properly ordered list of indices into the vertex list. In addition, each vertex and cell has a global sequence number (GSN) and each cell has an assigned PE number. The GSN is unchanged during the run. The vertices, cells, and to-be-constructed faces also have a local sequence number, but these are relevant only to the code itself during the run.

After reading the input file, the mesh is created by constructing the cell, vertex, and face objects. The objects are accessed by pointers. The objects also have their own interconnecting pointers.

For MPP runs, as the cells are read in, the cell objects are constructed so that the cell pointers first list the owned cells, then the ghost cells. There is also a considerable amount of sorting of cells and vertices to facilitate the construction of the MPOs. However, this effort is a small overhead in the eventual problem running time because the mesh connectivity, and hence the logical data of the MPOs, do not change during the course of the run.

3 Hydrodynamics

Two types of message exchanges, face and vertex centered, arise in the hydrodynamic scheme. The scheme [4] is conceptually straightforward to parallelize since it is compact and temporally explicit, although it is second order in both space and time. The method is an extension of the Godunov scheme in which all variables are cell-based. If applied to the equation, $\partial_t f + \nabla \cdot F = 0$, the scheme integrates over Δt and a cell to advance the average value f_j of the j-th cell:

$$V_j\,(f_j^{n+1} - f_j^n) + \Delta t \int dA \cdot F^n = 0\,,$$

where V_j is the cell volume, the superscript denotes the time level, and the area integral is a sum over the cell's faces. The face fluxes are solutions to Riemann problems whose initial conditions are the cell-based f^n on either side of the face. In 1D, this yields the explicit dependence,

$$f_j^{n+1} = \mathcal{F}(f_{j-1}^n, f_j^n, f_{j+1}^n)\,.$$

Hence, if an inter PE boundary separates the j-th and (j+1)-st cells, if PE[i] owns the j-th cell, and if the latest value f_{j+1}^n has been passed and loaded into the proper ghost cell, then PE[i] will compute the correct, new cell average.

The responsibility for the message passing lies with an MPO, which in this case is face-cell-centered. That is, both send and receive MPOs run through the same faces, but the sending MPO reads and packs data into a buffer from owned cells, while the receiving MPO unpacks a buffer and loads data into ghost cells.

[1] ICF3D may be run in either 3D Cartesian, cylindrical, or spherical coordinate systems.

The second order aspect of the scheme complicates the above procedure. Temporal accuracy is obtained by a two-step Runge-Kutta scheme which makes two passes through the coding. This implies two sets of message exchanges per time cycle, but does not cause any other complications. However, the spatial accuracy does complicate matters since in each cell, the dependent variables are non-constant and may have different values at each of the vertices. Across each face, the initial data of the Riemann problem are the values of the variables on the vertices of the cell adjoining the face. This data is obtained by following the pointer of the face, to the cell, and then to the correct vertex values.

The second type of message exchange is vertex-centered and is required by the limiting procedure which removes local extrema from the cell's vertex values. For example, after the DFE pressures are calculated, they are restricted (limited) to a range obtained from the average values of the adjacent cells. The procedure is also explicit with compact support. If the vertex lies on an inter-PE boundary, by definition, there is at least one ghost cell attached. Special MPOs collect and distribute the extremal values to all PEs that own cells attached to the vertex.

4 Diffusion

In contrast to the hydrodynamic module in which calls to the parallelization functions appear in several places, e.g., before computing fluxes, in the diffusion modules, the parallelization occurs after the equation is discretized, a "local" linear system assembled, and the system solver called.

In ICF3D diffusion equations arise in simulating heat conduction, or in the two versions of the diffusion approximation for radiation transport. In all cases the equation is of the form

$$G\,\partial_t f = \nabla\cdot(D\nabla f) + S - Lf\ , \tag{1}$$

where G, D, S, $L \geq 0$. The unknown function is approximated as,

$$f(x,t^n) = \sum_j \phi_j(x) f_j^n\ ,$$

where, if x_i is a vertex, $\phi_j(x_i) = \delta_{ij}$ is the usual basis function. To advance Eq. (1), we use implicit temporal differencing and obtain,

$$(G' + L' - \nabla\cdot D\nabla) f^n = G' f^{n-1} + S'\ . \tag{2}$$

Note that Δt is absorbed into D', L', and S'. Next, Eq. (2) is multiplied by a basis function ϕ_i and integrated over the "domain." For MPP applications, the relevant domain is the SD of the PE, i.e., only its owned cells. Thus, the index i of the ϕ_i function ranges over the vertices of only the owned cells. Each multiplication by ϕ_i corresponds to one row of the linear system, $Af = b$ for the nodal unknowns f_j^n. The matrix is SPD, but the system is incomplete since equations corresponding to unknowns on the inter-PE boundary do not include integrals over ghost cells.

The MPP methodology is incorporated into the solver which, for MPP runs, first calls another routine that assembles the *distributed* linear system.

4.1 The MPP Distributed System

Since the linear systems are vertex-based, we extend the concept of PE ownership to the vertices. If all cells adjoining a vertex are owned by PE[i], we let PE[i] own the vertex. This procedure leaves ambiguous the ownership of vertices on inter-PE boundaries and those on the "exterior" of the ghost cells. Ownership of the former may be determined without requiring message passing. The simplest algorithm is for each PE to survey the ownership of all cells attached to the vertex and assign the vertex to the PE of lowest number. Assigning ownership of the exterior nodes requires message-passing since a PE does not have access to all cells that are attached to them. Since a PE unequivocally knows who owns the vertices attached to all its owned cells, during the initialization phase, each PE receives a message from the owners of its ghost cells regarding the ownership of this PE's exterior vertices. Although the coding for this procedure is complicated, it is called only once during initialization.

One essential operation within the CG iterations is MatVec, the multiplication of a matrix by a vector. On MPPs, after the system has been distributed, the matrices are rectangular. The number of rows equals the number of owned vertices and the columns correspond to the number of vertices linked to the owned vertices. The matrices are stored in compressed row form to avoid storing zeroes.

To facilitate the assembly of the distributed system, the vertices are sorted into six types: T_1 to T_6. Before describing them, we define $\mathcal{V}$ to be the set of vertices owned by the PE, $\mathcal{W}$ to be the set of vertices not owned by the PE, $\mathcal{S}(x)$ as the set of vertices connected to the vertex x by the stencil, and as *exterior* the vertices of ghost cells which do not lie on inter-PE boundaries. Thus,

- $T_1 = \{x \mid x \in \mathcal{V}.\ \forall y \in \mathcal{S}(x),\ y \in \mathcal{V}\}$
- $T_2 = \{x \mid x \in \mathcal{V}.\ \exists y \in \mathcal{S}(x),\ y \in \mathcal{W}\}$
- $T_3 = \{x \mid x \in \mathcal{V}.\ x \text{ is on inter-PE bdry.}\}$
- $T_4 = \{x \mid x \in \mathcal{W}.\ x \text{ is on inter-PE bdry.}\}$
- $T_5 = \{x \mid x \text{ is exterior.}\ \exists y \in T_3,\ x \in \mathcal{S}(y)\}$
- $T_6 = \{x \mid x \text{ is exterior.}\ \forall y \in \mathcal{S}(x),\ y \in \mathcal{W}\}$

The six types stem from the requirements for assembling the distributed system and for performing a MatVec. For the T_1 and T_2 vertices, the PE can compute the entire row of matrix coefficients without input from other PEs. However, on T_2 vertices, before completing a MatVec, the PE needs the latest value on some T_4 vertices. On T_3 vertices, the PE needs input from other PEs for completing the calculation of the row and for a MatVec operation. Linear equations which the PE computes on its T_4 vertices are sent to the PE which owns them. The T_6 vertices are not needed in the diffusion module.

Before calling the solver, each PE computes a partial linear system, $Ax = b$, after performing the required finite element integrations over the owned cells. If we index the matrix and vector elements by type, then

$$A = \begin{pmatrix} A_{11} & A_{12} & A_{13} & 0 \\ A_{12} & A_{22} & A_{23} & A_{24} \\ A_{13} & A_{23} & A_{33} & A_{34} \\ 0 & A_{24} & A_{34} & A_{44} \end{pmatrix}, \ x = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix}, \tag{3}$$

and similarly for b. In Eq. (3) A_{ij} denotes the, not necessarily square, matrix of coefficients of interactions between types T_i and T_j vertices and similarly x_i are the type T_i vector elements.

After computing the incomplete linear system whose matrix is given by Eq. (3), for MPP runs, the solver first calls certain MPOs which assemble the distributed system. Each row of T_4 vertices is sent to the PE which owns the vertex. In the context of how to compute a MatVec $y = Ax$, if the subscripts a and b divide the vertices into types $T_{1,2,3}$ and $T_{4,5}$ respectively, we have

$$y_a = A_a x_a + A_b x_b \tag{4}$$

where,

$$x_a^{\mathrm{T}} = (x_1, x_2, x_3) \ , \ \ x_b^{\mathrm{T}} = (x_4, x_5) \ ,$$

$$A_a = \begin{pmatrix} A_{11} & A_{12} & A_{13} \\ A_{12} & A_{22} & A_{23} \\ A_{13} & A_{23} & A_{33} + A'_{33} \end{pmatrix}, \ \text{and} \ A_b = \begin{pmatrix} 0 & 0 \\ A_{24} & 0 \\ A_{34} + A'_{34} & A'_{35} \end{pmatrix}. \tag{5}$$

In Eq. (5) the primes denote matrix coefficients which have been computed by other PEs, i.e., on their T_4 vertices, and message-passed to this PE.

4.2 The Parallel Solver

The resulting linear equations are solved using preconditioned conjugate gradients (PCG). Two types of preconditioners are available: n-step Jacobi, and a parallel version of incomplete Cholesky (IC). Each iterative step of CG requires

- Three SAXPY, i.e., $\alpha x + y$ where α is scalar and x and y are vectors
- Two dot products
- One MatVec
- Solving the preconditioned system,

$$Pz = r \ . \tag{6}$$

The SAXPY operations do not require any message passing.

The dot products are calculated using a global reduction function.

The MatVec is computed according to the splitting defined in Eq. (5) except we interleave message passing and computation. The procedure is as follows. The PEs first call asynchronous receive functions and then halt at a barrier. Next, the PEs call the ready-to-send functions and, without waiting for the messages to arrive, calculate the first part of Eq. (4), i.e., $A_a x_a$. Afterwards, the PEs halt at another barrier before adding $A_b x_b$ to the result.

The key behind PCG is a preconditioning matrix P which closely resembles A and at the same time renders Eq. (6) easy to solve. Both the IC and n-step Jacobi preconditioners can be cast in the form of generalized polynomial preconditioners in which A is first factored into two parts,

$$A = M - N = M(I - G) \ , \ \ G \doteq M^{-1}N \ .$$

If $||G|| < 1$, then the inverse may be approximated by

$$A^{-1} = (I - G)^{-1}M^{-1} \approx (I + G + \cdots + G^{i-1})M^{-1} \ .$$

We now define P as the inverse of the approximation, i.e.,

$$P \doteq M \left(\sum_{i=0}^{j-1} G^i \right)^{-1} \tag{7}$$

where $G^0 = I$.

For n-step Jacobi, M = diag(A) and we set $j = n$ in Eq. (7); 1-step is the easily parallelizable diagonal scaling preconditioner. If $n > 1$, the preconditioner requires $n - 1$ operations, each of which consists of a multiplication by the off-diagonal elements of A, then by M^{-1}.

Unfortunately, in our applications n steps are usually no better than simple diagonal scaling and can be significantly worse than 1 step – see ref. [1].

Our favored preconditioner is a parallel IC variant which we now describe. In the context of Eq. (7) we set $j = 1$ and let $P = LDL^{\mathrm{T}}$, where L is lower triangular with unit diagonal, D is diagonal, and the sparsity pattern of L matches that of A except that we do not allow links to types T_4 and T_5 vertices. In other words, we form an incomplete factorization of matrix A_α in Eq. (5). Since A_α links only the owned nodes, the preconditioning step does not inhibit parallelization, nor does it require any message passing.

On each PE, our parallel IC preconditioner is equivalent to an incomplete decomposition of the underlying diffusion equation on the $T_{1,2,3}$ nodes with homogeneous Dirichlet data specified on the non-owned $T_{4,5}$ vertices. In that light, a possible improvement may be to replace the homogeneous data with "stale" values of the z vector on the $T_{4,5}$ vertices.

We end this section with a comparison of execution times of runs using two preconditioners, IC and 1-step Jacobi. Table 1 shows the effect of increasing the number of PEs while keeping the mesh size fixed. As expected, our parallel ICCG degrades as the SDs get smaller while Jacobi scales with the number of PEs. Although the problem size is small, the parallel IC preconditioner is superior to 1-step Jacobi.

5 Point Explosion

In this section, we present a calculation that uses the parallelization routines described above. Although the problem is spherically symmetric, we run in 3D and compare results to a 1D spherical calculation.

Table 1. Thermal Wave Problem. Execution time for different preconditioners and PE configurations, 32×8×8 cells, 2337 vertices.

	4 PEs	16 PEs
Preconditioner	512 cells/PE	128 cells/PE
ICCG	163.	82.
1-step Jacobi	404.	129.

The problem, suggested by Reinicke and Meyer-ter-Vehn [6] and later analyzed in ref. [7], consists of the sudden release of energy in the center of a cold, constant-density gas. The problem is a combination of the well-known hydrodynamic point explosion[8] and the spherically expanding thermal wave [9].

The equations of interest are the Euler equations with heat conduction:

$$\begin{aligned} \partial_t \rho + \nabla\cdot(\rho \mathbf{u}) &= 0 \,, \\ \partial_t(\rho \mathbf{u}) + \nabla\cdot(\rho \mathbf{u}\mathbf{u} + p) &= 0 \,, \\ \partial_t(\rho E) + \nabla\cdot(\mathbf{u}(\rho E + p)) &= -\nabla\cdot H \,, \end{aligned} \tag{8}$$

where the heat flux has the form,

$$H = -\chi \nabla T \,, \quad \text{where } \chi = \chi_0 T^b \,,$$

and where χ_0, and b are constants. In Eqs. (8), $\mathbf{u}$, ρ, p, T, and E denote the velocity, density, pressure, temperature, and total specific energy respectively. The thermodynamic variables satisfy the ideal gas equation of state,

$$p = (\gamma - 1)\rho e = (\gamma - 1)\rho c_V T \,.$$

At $t = 0$, the energy is all internal and is concentrated at the origin,

$$\rho E|_{t=0} = \rho e|_{t=0} = \mathcal{E}_0 \, \delta(r) \,,$$

while the quiescent gas is at constant density, $\rho|_{t=0} = 1$. We set,

$$\gamma = 7/5 \,, \quad b = 5/2 \,, \quad \chi_0 = c_V = 1 \,, \quad \text{and} \quad \mathcal{E}_0 = 0.76778 \,.$$

The numerical value for $\mathcal{E}_0$ was calculated a posteriori after we initialized the central T to a large value.

The solution may be estimated by separate analyses of the corresponding pure hydrodynamic and pure diffusion problems, both of which possess similarity solutions. However, in this case, the coupled problem is not self-similar.

The pure diffusion solution is characterized by a temperature front r_f which increases with time. The pure hydrodynamic problem is characterized by an infinite strength shock whose position r_s also grows with time. For the parameters chosen,

$$r_s \propto t^{2/5} \quad \text{and} \quad r_f \propto t^{2/19} \,.$$

Thus, in early times, diffusion dominates; in late times, hydrodynamics.

In ref. [7], the author computes an approximate time $t_\times$ and radius $r_\times$ for the two fronts to intersect. For the parameters chosen,

$$t_\times = 0.8945 \text{ and } r_\times = 0.9371 .$$

Figure 1 displays ρ and T at the late-time, hydrodynamically dominated regime, $t = 1.8$; at $r \approx 1.28$ we have a strong shock.

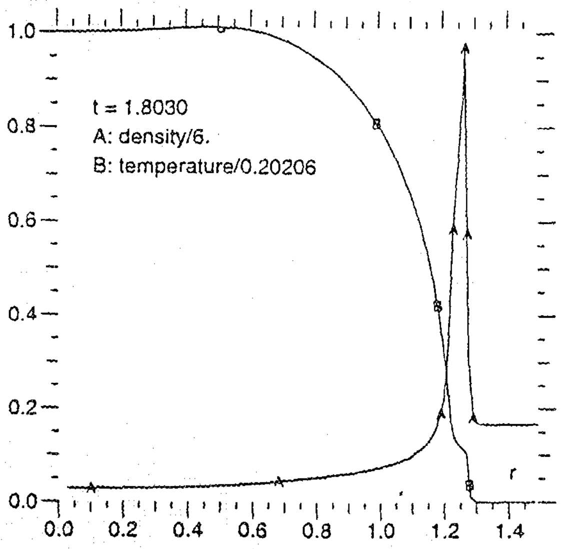

Fig. 1. Density and temperature vs. r at late time. T is normalized by $T(r = 0)$, ρ is normalized by the value across an infinite strength shock.

The results depicted in Fig. 1 were obtained by running ICF3D in "1D" mode, i.e., with only 1 cell in the transverse directions and 100 radial cells. We now display results of a 3D run in Cartesian coordinates on an unstructured grid on 64 PEs. The mesh consists of 11580 tetrahedra and 2053 nodes and is created by the LaGriT code [5] obtained from Los Alamos National Laboratory. The radial direction is discretized into initially 16 "spherical shells" of uniform width $\Delta_r = 0.125$. We partition the mesh using METIS [2]. Figure 2 displays the outside of the spherical domain. The innermost "sphere" consists of 20 tetrahedra all connected at the origin.

Figures 3 and 4 respectively display ρ and T across the $Z = 0$ plane at $t = 1.8$. The ρ result in Fig. 3 shows jaggedness in the transverse direction and an inability to reach the high compressions attained by the 1D result. Both errors are due to the coarseness of the grid and to the usage of tetrahedral cells. The result in Fig. 4 compares more favorably with Fig. 1. However, none of

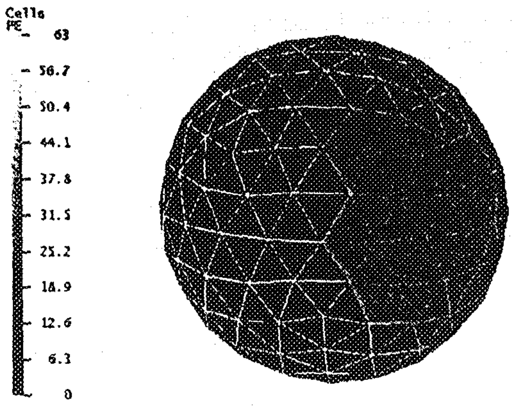

Fig. 2. Exterior of spherical domain. Colors represent PE numbers.

the discrepancies between the 3D and 1D results are due to the parallelization. There is no sign of the subdivision of the domain into 64 SDs.

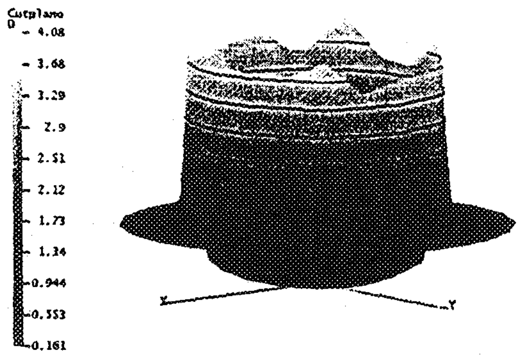

Fig. 3. Density at $t = 1.802$ across $Z = 0$ plane.

Lastly, we present a scalability study by varying the number of PEs while keeping the mesh size fixed. First, we define the parallelization efficiency E_N,

$$E_N = \frac{t_{N/2}}{2\,t_N}$$

where t_N is the execution time for N PEs. Table 2 lists E_N derived from the execution times of runs on an IBM SP2 for the above unstructured grid problem. Note that E_N remains close to 1 as N increases.

Table 2. Number of PEs N, execution time t_N, and parallelization efficiency E_N for a fixed size problem (11580 cells, 2053 nodes) run for 200 cycles.

N	t_N(sec)	E_N
8	1628.02	-
16	882.77	.922
32	489.78	.901
64	277.08	.884

6 Summary and Conclusion

We have presented a scalable method for parallelizing a 3D, unstructured grid, finite element code. The method is a SPMD model targeted for distributed memory architectures, but also works on shared memory computers. We decompose physical space into a collection of disjoint SDs, one per PE. The input files tag cells with a PE number. The code forms an analogous designation for the vertices. The SDs are surrounded by a layer of ghost cells which are used to store data that is owned, i.e., computed, by other PEs. Explicit calls to message passing functions communicate the data amongst the PEs. The calls are made by special MPOs which contain information that describes the PE network.

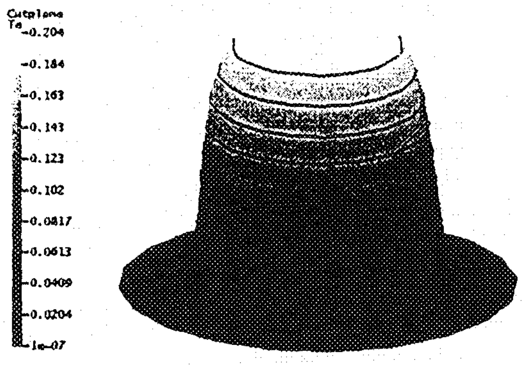

Fig. 4. Normalized T at $t = 1.802$ across $Z = 0$ plane.

We have demonstrated the method on a test problem requiring both temporally explicit and implicit schemes for partial differential equations. The concept of ghost cells leads to a straightforward parallelization of the explicit hydrodynamic scheme. The MPOs insure that the data computed by other PEs is placed in the expected locations.

Parallelization of the implicit discretization of the diffusion equations consists of a wrapper around the call to the linear system solver. For an MPP run, the linear system generated by the uniprocessor coding is incomplete since it is formed by integrating over only the owned cells. The parallelization completes the system by sending equations on non-owned vertices to the PE that owns them.

Since the linear systems are large, sparse, and SPD, they are solved using a parallelized PCG algorithm in which our noteworthy contribution is the parallel IC preconditioner. In a sense, it is even more incomplete than typical IC since our decomposition ignores links to non-owned vertices. Its utility depends on PEs having large SDs. In all cases studied, the number of CG iterations required using the parallel IC is less than that required by Jacobi. However, if the SDs are very small, and/or the matrix condition number is small, Jacobi is faster. In our applications, the parallel IC preconditioner has been very effective.

The important aspect of our parallelization strategy is its consistent reliance of domain decomposition of physical space. We are now parallelizing the laser deposition module along the same lines. Preliminary results are encouraging.

References

1. A. I. Shestakov, M. K. Prasad, J. L. Milovich, N. A. Gentile, J. F. Painter, G. Furnish, and P. F. Dubois, "The ICF3D Code," Lawrence Livermore National Laboratory, Livermore, CA, UCRL-JC-124448, (1997), submitted to *Comput. Methods Appl. Mech. Engin.*
2. G. Karypis and V. Kumar, "A fast and high quality multilevel scheme for partitioning irregular graphs," Technical Report TR 95-035, Department of Computer Science, Univ. Minn., 1995. To appear in *SIAM Journal on Scientific Computing 1997.* A short version appears in Intl. Conf. on Parallel Processing 1995. The METIS program is available on the web from: http://www-users.cs.umn.edu/ karypis/metis/metis/ main.html
3. *AVS Developer's Guide,* Advanced Visual Systems, Inc., Release 4, May 1992, p. E-1, 300 Fifth Ave., Waltham MA 02153.
4. D. S. Kershaw, M. K. Prasad, M. J. Shaw, and J. L. Milovich, *Comput. Methods Appl. Mech. Engin.*, **158** p. 81 (1998).
5. www.t12.lanl.gov/ lagrit.
6. P. Reinicke and J. Meyer-ter-Vehn, *Phys. Fluids* A **3** (7), p. 1807 (1991).
7. A. I. Shestakov, "Simulation of two Point Explosion-Heat Conduction Problems with a Hydrodynamic-Diffusion Code," Lawrence Livermore National Laboratory, Livermore, CA, UCRL-JC-124448, (1998), submitted to *Phys. Fluids* A.
8. L. D. Landau and E. M. Lifshitz, *Fluid Mechanics*, 2nd Ed., Pergamon Press, Oxford p. 404 (1987).
9. Ya. B. Zel'dovich and Yu. P. Raizer, *Physics of Shock Waves and High-Temperature Hydrodynamic Phenomena*, Vol. II, Academic Press, p. 668 (1966).

Acknowledgement

We gratefully acknowledge the help of D. George and A. Kuprat of Los Alamos National Laboratory for supplying the LaGriT mesh generation code.