from *SIAM News*, Volume 32, Number 3

Parallelization of an Unstructured-grid, Laser Fusion Design Code

By Aleksei Shestakov, Jose Milovich, and David Kershaw

APPLICATIONS ON ADVANCED ARCHITECTURE COMPUTERS

Greg Astfalk, Editor

The essential physics of laser fusion is described by the hyperbolic Euler equations for mass, momentum, and energy; diffusion equations that model heat conduction and radiation transport; and a model for laser energy deposition. In this article we describe the parallelization of ICF3D [4], a 3D, unstructured-grid, finite element C++ code initially written to simulate laser fusion experiments.

The ICF3D mesh consists of an arbitrary collection of hexahedra, prisms, pyramids, and/or tetrahedra. The only restriction is that cells can share faces only of like kind. We parallelize by decomposing the physical domain into a collection of disjoint subdomains, one per processing element (PE). The input files tag each cell with the number of the PE that owns it. The cells owned by a PE constitute the PE's subdomain. A cell that shares at least one vertex with a cell owned by a PE, but that is owned by another PE, is called a ghost cell of the first PE. Each PE receives a terse description of only its subdomain, along with a layer of ghost cells.

The decomposition is especially suited to distributed-memory architectures but can also be used on shared-memory systems. A single-program, multiple-data model is adopted. In the parallelization of ICF3D, difficulties arise at four levels:

- Embarrassingly parallel functions that do not require message-passing, e.g., the cell-by-cell calls to the equation-of-state.
- Straightforward parallelization of the temporally explicit hydrodynamic scheme.
- Functions requiring global communication, e.g., solution of the linear systems used to discretize the diffusion equations.
- Unpredictable point-to-point (PtP) communication arising in the laser ray-tracing scheme.

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. Some functions, such as the one that computes the hydrodynamic fluxes, are face-centered. Other functions are cell-centered. The diffusion equations, for example, are advanced by standard finite element techniques, which require integrating over cells; this integration is done by each PE over only the cells it owns. However, once the linear system has been assembled and properly distributed among the PEs, the calculation is vertex-centered. In this article we describe the assembly and solution of such systems, in which each PE sees only its subdomain and the surrounding ghost cells.

The PEs communicate using one of two types of message-passing function libraries, MPI or the native Cray SHMEM. Two types of communication arise, global and PtP. An example of the former is the calculation of the time step Δt. Each PE first computes its acceptable value, and a global reduction function then computes the minimum. In PtP communication, PE[i] exchanges messages only with PEs that own its ghost cells. For such exchanges, ICF3D relies on message-passing objects (MPOs) constructed during initialization. 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.

Hydrodynamics

Message exchanges of two types, face- and vertex-centered, are needed for the hydrodynamic scheme [3], a second-order extension of the Godunov method in which all variables are cell-based. Since it is compact and temporally explicit, the scheme is straightforward to parallelize. Second-order temporal accuracy is attained by a two-step Runge–Kutta integration. When the scheme is applied to $\partial_t f + \nabla \cdot \mathbf{F} = 0$, the equation is first multiplied by a basis function ϕ_i and then integrated over a cell, yielding a predicted value for

$$\int_{\text{cell}} \phi_i \, f \, dV$$

The integral of $\phi_i \nabla \cdot \mathbf{F}$ is integrated by parts, which gives two other integrals, one of which reduces to a sum of area integrals over the cell's faces. These face fluxes are solutions to Riemann problems whose initial conditions are the cell-based values of f on either side of the face. Hence, if an inter-PE boundary separates the jth and $(j + 1)$st cells, if PE[k] owns the jth cell, and if the latest value f_{j+1} has been passed and loaded into the proper ghost cell, then PE[k] easily computes the new cell average.

The hydrodynamic scheme's second-order spatial accuracy adds another complication. The dependent variables, which are allowed to vary over a cell, are uniquely determined by their vertex values (which may be different in a neighboring cell) and are therefore doubly indexed, first over the cell and then over the cell's vertices. Hence, the initial data for the Riemann problem are the values for the vertices adjoining the face; for data collection, the pointer is followed to the face, to the cell, and finally to the proper vertex.

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.

Vertex-centered message exchange, the second type of communication, is used by two procedures. One removes local extrema from the cell's vertex values. The other, which arises in Lagrangian calculations, determines the displacement of the vertex. Both are categorized as local reduction procedures, explicit with compact support. If the vertex lies on an inter-PE boundary, then, by definition, at least one ghost cell is attached. Special MPOs collect and distribute the required values to all PEs that own cells attached to the vertex.

Diffusion

In ICF3D, diffusion equations arise in the simulation of both heat conduction and radiation transport. In all cases the equation is of the form

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

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

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

where the ϕ_j are the basis functions. To advance (1), we use implicit temporal differencing and obtain

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

by absorbing Δt into D', L', and S'. Equation (2) is then multiplied by a basis function ϕ_i and integrated over the PE's subdomain. The index i of the basis functions ranges over the vertices of only the owned cells. Each multiplication by ϕ_i corresponds to one row of the linear sys-tem $A f = b$ for the nodal unknowns f_j^n. At this point, the system is incomplete, as equations corresponding to unknowns on the inter-PE boundary do not include integrals over ghost cells.

Parallelization is incorporated into the linear system solver, which begins by assembling the distributed linear system.

The distributed system. Since the linear systems are vertex-based, we extend the concept of PE ownership to vertices. If all cells adjoining a vertex are owned by PE[i], we let PE[i] own that vertex. This procedure leaves ambiguous the ownership of vertices on inter-PE boundaries and on the exterior of the ghost cells.We can determine ownership of the former without message-passing by having each PE survey the ownership of all cells attached to a vertex and assigning the vertex to the PE with the lowest number. Message-passing is required for the exterior vertices, however. Since PE[i] cannot access all cells attached to those vertices, during the initialization phase it receives messages containing the ownership information from all PEs that own its ghost cells. This solves the problem—if a PE owns a cell, it unequivocally knows the ownership of the cell's vertices.

Preconditioned conjugate gradients (PCGs) are used to solve the resulting large, sparse, and symmetric linear systems. One essential operation within the CG iterations is the multiplication of a matrix by a vector (MatVec). For the distributed system, 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 the storage of zeroes.

To facilitate the assembly of the distributed system, the vertices are sorted into six types, T_1 to T_6. We let V, W, and X define the sets of vertices that are attached only to owned cells, that lie on the inter-PE boundary, and that are found on the exterior of ghost cells, respectively. If O is the set of owned vertices and $S(x)$ is the set of vertices connected to x by the diffusion stencil then

$$\begin{aligned}
T_1 &= \left\{x \mid x \in V \,\&\, \forall y \in S(x),\, y \in O\right\} \\
T_2 &= \left\{x \mid x \in V \,\&\, \exists y \in S(x),\, y \notin O\right\} \\
T_3 &= \left\{x \mid x \in W \,\&\, x \in O\right\} \\
T_4 &= \left\{x \mid x \in W \,\&\, x \notin O\right\} \\
T_5 &= \left\{x \mid x \in X \,\&\, \exists y \in S(x),\, y \in O\right\} \\
T_6 &= \left\{x \mid x \in X \,\&\, \forall y \in S(x),\, y \notin O\right\}
\end{aligned}$$

The six types stem from the requirements for assembling the distributed system and for performing a MatVec. For T_1 and T_2 vertices, the PE can compute the entire row of matrix coefficients without input from other PEs. For T_2 vertices, however, the PE needs the latest values for some T_4 vertices before it can complete a MatVec. For T_3 vertices, the PE needs input from other PEs to complete the equation, and the latest values for some T_4 and T_5 vertices to perform a MatVec. Linear equations computed by the PE on its T_4 vertices are sent to the PE that owns the vertices. The T_6 vertices are not needed by the diffusion module. A PE's T_2 and T_3 vertices are of types T_5 and T_4, respectively, on neighboring PEs.

Before the solver is called, the diffusion equation leads, on each PE, to an (incomplete) linear system $Ax = b$, with A in block form:

$$A = \begin{pmatrix} A_{11} & A_{12} & A_{13} & A_{14} \\ A_{12} & A_{22} & A_{23} & A_{24} \\ A_{13} & A_{23} & A_{33} & A_{34} \\ 0 & A_{24} & A_{34} & A_{44} \end{pmatrix}$$

where the rectangular A_{ij} contain the interactions between vertices of types T_i and T_j. The vectors x and b are in similar block form. After computing the incomplete linear system, the solver calls certain MPOs, which assemble the distributed system. Once the system has been assembled, if the subscripts a and b divide the vertices into types $T_{1,2,3}$ and $T_{4,5}$, a MatVec is of the form

$$y_a = A_a\, x_a + A_b\, x_b \tag{3}$$

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

$$A_a = \begin{pmatrix} A_{11} & A_{12} & A_{13} \\ A_{12} & A_{22} & A_{23} \\ A_{13} & A_{23} & A_{33} + A'_{33} \end{pmatrix},$$

$$A_b = \begin{pmatrix} 0 & 0 \\ A_{24} & 0 \\ A_{34} + A'_{34} & A'_{35} \end{pmatrix} \tag{4}$$

In (4) the primes denote matrix coefficients sent to a PE after being computed by other PEs, i.e., on their T_4 vertices.

The parallel solver. Preconditioned conjugate gradients are used to solve the resulting linear equations. Each CG iterative step requires three SAXPYs, two dot products, one MatVec, and solution of the preconditioned system

$$Pz = r \tag{5}$$

The SAXPYs do not require message-passing. We use a global reduction function to calculate the dot products.

The MatVec is computed according to the splitting defined in (3), except that we interleave message-passing and computation as follows: After calling nonblocking receive functions, the PEs call the ready-to-send functions and, without waiting for the messages to arrive, calculate the first part of (3), i.e., $A_a x_a$. Afterward, the PEs halt at a barrier before adding $A_b x_b$ to the result.

The crux of an efficient PCG is a preconditioning matrix P that closely resembles A and at the same time renders (5) easy to solve. In ICF3D, preconditioners of two types are available: n-step Jacobi and a parallel version of incomplete Cholesky (IC). However, in our applications, n-step Jacobi with $n > 1$ is inefficient since A may not be an M matrix [4].

Our favored preconditioner is a parallel IC variant for which $P = LDL^{\mathrm{T}}$, 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 T_4 and T_5 vertices. In other words, we form an incomplete factorization of the matrix A_a in (4). Since A_a 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 precon-ditioner 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 nonowned $T_{4,5}$ vertices. In that light, a possible improvement would be to replace the homogeneous data with "stale" values of the z vector on the $T_{4,5}$ vertices.

P	4 PEs 512 cells/PE	16 PEs 128 cells/PE
IC	163	82
Jacobi	404	129

Table 1. *Execution time in seconds for a thermal wave problem with 32 × 8 × 8 cells and 2337 vertices, for different preconditioners P and PE configurations. IC indicates incomplete Cholesky.*

Table 1 shows the effects of increasing the number of PEs while keeping the mesh size fixed, comparing the two preconditioners used, IC and 1-step Jacobi. As expected, our parallel IC CG method degrades as the subdomains get smaller, while Jacobi scales reasonably well with the number of PEs. Although the problem size is small, the parallel IC preconditioner is superior to 1-step Jacobi. However, for problems in which each PE has many cells, we expect IC to outperform Jacobi and have therefore made it the default preconditioner.

Laser Ray Tracer

For the simulation of laser energy deposition, each laser beam is discretized into a number of rays, which are followed until they are absorbed or leave the physical domain [1]. The trajectories of the rays depend on the free electron number density n_e. Rays cannot enter regions in which $n_e > n_c$ (n_c, the critical density, is a function of the laser's frequency). Each ray is initialized with a certain energy, which it transfers to the mesh by inverse *bremsstrahlung* (radiation absorption by free electrons) and which is an explicit energy source for the heat conduction equation.

The ray-tracing algorithm first determines where the rays enter the problem boundary and then tracks them through the cells. Rays enter and exit cells through the faces. At each face crossing, the ray's new direction is determined by Snell's law, since n_e is, in general, discontinuous across the face. If in the next cell n_e is sufficiently high, the ray reflects. Rays are processed one at a time.

N	τ(sec)	e_N
1	845.61	—
2	364.28	1.161
4	199.16	1.061
8	99.61	1.061
16	58.14	.909
32	41.8	.632

Table 2. *Laser test problem for a total of 128 cells. N is the number of PEs, τ the execution time, and $e = \tau_1 / (N \tau_N)$ is the efficiency.*

Parallelizing the algorithm is a challenge because we maintain the existing decomposition of the domain. (The obvious scheme, in which the mesh is replicated on each PE and the rays are parcelled out among the PEs, does not scale.) We parallelize by combining the rays into batches. A PE processes a batch until each ray in the batch has been absorbed, has exited the problem, or has crossed an inter-PE boundary. Messages are then exchanged.

Since rays cross only on faces, we reuse the face-centered MPOs written for the hydrodynamic fluxes. Unfortunately, since the communication pattern is unpredictable—evolving conditions change the ray trajectories—we must employ a two-step scheme. After each batch has been processed, each PE exchanges messages with its face-sharing neighbors regarding the number of rays to pass. This establishes which PEs need to allocate buffers of the proper size to receive the ray information. As in the other MPOs, we use asynchronous nonblocking functions. Receives are posted first, followed by sends.

Although, for maximum efficiency, the algorithm requires that each PE have regions with $n_e < n_c$ accessible to the laser, it is remarkably efficient if this holds. Table 2 displays results from a test problem in which a row of 128 cells, with constant initial density $n_e < n_c$, is irradiated on one end by 10,000 rays. We use a batch size of 100, i.e., the right-most PE tracks the first 100 rays and passes them to its neighbor. Two PEs then process their own batches, and so forth. For these results, only N, the number of PEs, is varied. When the number of PEs is increased from one to two, we see that the execution time τ drops by more than half. For large N, the efficiency degrades because there are so few cells per PE.

Implosion Problem

To test the code, we devised a problem that simulates the implosion of an initially quiescent, spherical gas bubble caused by the sudden deposition of laser energy on its surface. We use CGS units in the following description, except for temperature, which is measured in keV, and time, which is occasionally measured in nsec (10^{-9} sec). A word of caution: We chose the parameters so as to create different regions of dominance in the three physics packages (laser, hydrodynamics, and nonlinear heat conduction), and they represent experimental values only. Briefly, the absorbed laser energy is a source of heat that is quickly diffused over the surface and drives a supersonic thermal wave inward. When the heat wave slows down, hydrodynamics takes over and an imploding shock wave arises.

We use an ideal gas equation-of-state in which $\gamma = 1.4$ and the specific heat $c_v = 10^{15}$ erg (gm keV)$^{-1}$. The heat flux is of the form $H = -\chi\nabla T$, where $\chi = \chi_0 T^{5/2}$ and $\chi_0 = 10^{11}$. Our chosen laser frequency of $f_\ell = 2.866 \times 10^{14}$ sec^{-1} yields a critical mass density of $\rho_c \approx 1.668 \times 10^{-3}$ gm $\cdot$ cm^{-3}. In order to have the laser deposit its energy only at the surface, we initialize the gas to be largely overdense, using the following point-centered distribution:

$$\rho\big|_{t=0} = \begin{cases} 10\times\rho_c & \text{if} \quad r \le r_0 - 2\Delta \\ 2\times\rho_c & \text{if} \quad r = r_0 - \Delta \\ 0.01\times\rho_c & \text{if} \quad r = r_0 \end{cases}$$

where $\Delta = r_0/16$ and $r_0 = 0.2$ is the initial radius of the bubble.

The laser package simulates 12 identical beams of circular cross-section with a radius equal to r_0. The beams are centered on the vertices of a regular icosahedron (centers of the pentagonal patches of a soccer ball). Each beam's intensity is 2.875×10^{13} W cm^2.

We initialize with $T|_{t=0} = 10^{-3}$ keV, which leads to an initial energy of $E_0 \approx 4.20 \times 10^8$ erg. The laser has a "flat top" temporal power deposition profile; it is turned on at $t = 0$ and turned off at $t = 2$ nsec. The maximum energy that can be delivered by the laser is thus 8.67×10^{11} erg. Because the capsule is largely overdense, however, most of the laser rays reflect initially and very little energy is deposited. After 2 nsec, only $E_\ell = 8.39 \times 10^{10}$erg—which is approximately 10% of the laser, but still nearly 200 times E_0—has been absorbed by the gas.

We use the following boundary conditions: For the heat conduction, no flux is allowed to escape. For the hydrodynamics, $p = 6.67 \times 10^7$ erg $\cdot$ cm^{-3}. This value, which is 10 times less than the average initial pressure in the outer shell, allows the bubble to expand, thereby letting more rays enter the problem and deposit energy.

The simulation is run on the domain depicted in Figure 1, in which the shading depicts the domain decomposition of a run with 16 PEs.

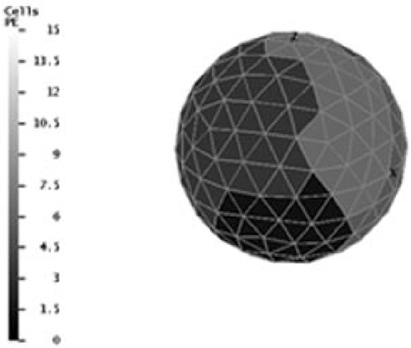

Figure 1. *Computational mesh for the laser implosion problem. The grid consists of 11,580 tetrahedra (16 radial cells), 2053 points, and 23,320 faces.*

The mesh is created with the LaGriT code [5], obtained from Los Alamos National Laboratory, and partitioned with METIS [2]. The grid consists of 16 "spherical shells," initially of uniform width $\Delta_r = 0.0125$. Unstructured grids allow us to customize the resolution; the innermost "sphere" is a regular icosahedron.

Figure 2 displays the density ρ across the $X = 0$ plane at $t = 12$ nsec, shortly after the shock has reflected off the origin; max(ρ) is nearly 50 times its initial value. The figure displays only the central part of the domain. (The exterior part has ablated out to a very large radius.)

After $t = 12$ nsec, the reflected shock continues to move outward. We end the simulation at $t = 16$ nsec; the final density is shown in Figure 3. As in Figure 2, only the central part of the domain is displayed.

Conclusion

In this article we have described the parallelization of a 3D, unstructured-grid, multiphysics code. Although we were concerned only with three physics packages, the techniques are immediately applicable to other modules, such as multigroup radiation transport and Monte Carlo transport.

We found that our paral-lelization scheme scales well. Given enough work (an adequate number of cells) for each PE, doubling the number of PEs approximately halves the execution time. Equivalently, if more PEs are available, larger problems can be run.

The coarse-grained parallelization is tailored for distributed-memory computers. The strategy can also be used on other architectures, such as clusters of compute nodes, each with several CPUs, or it can be extended to the context of fine-grained parallelism through the use of threads or compiler directives. Currently, however, portable directives like OpenMP are not available for C++ codes.

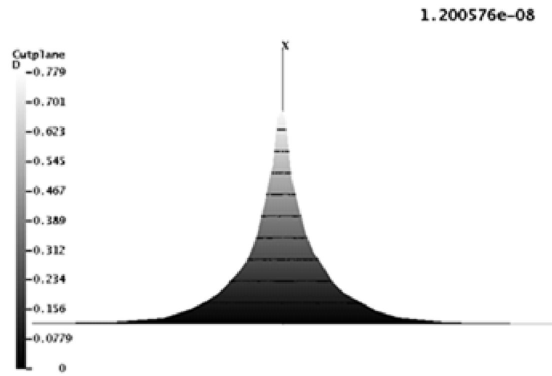

Figure 2. *Laser-driven implosion problem, with density ρ across the $X = 0$ plane, $t = 1.2 \times 10^{-8}$ sec.*

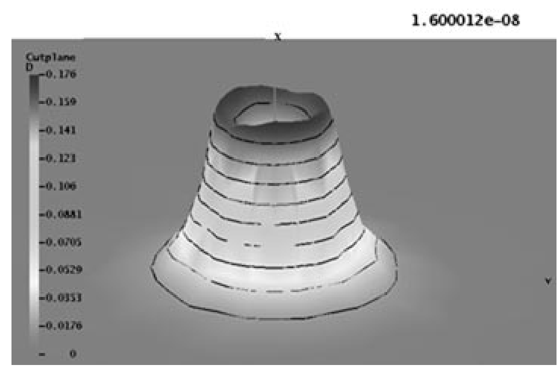

Figure 3. *Laser-driven implosion problem, with density ρ across the $X = 0$ plane, $t = 1.6 \times 10^{-8}$ sec.*

Acknowledgments

The authors gratefully acknowledge D. George and A. Kuprat of Los Alamos National Laboratory for supplying the LaGriT mesh-generation code. This work was performed under the auspices of the U.S. Department of Energy by the Lawrence Livermore National Laboratory under contract W–7405–ENG–48.

References

[1] 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., Program 40th Annual Meeting of the Division of Plasma Physics, 43:8 (1998), 1900.

[2] G. Karypis and V. Kumar, *A fast and high quality multilevel scheme for partitioning irregular graphs*, SIAM J. Sci. Comput., 20:1 (1998), 359–392. The METIS program is available on the Web at http://www-users.cs.umn.edu/~karypis/metis/metis/main.html.

[3] 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. Methods Appl. Mech. Eng., 158 (1998), 81–116.

[4] A.I. Shestakov, M.K. Prasad, J.L. Milovich, N.A. Gentile, J.F. Painter, and G. Furnish, *The radiation–hydrodynamic ICF3D code*, to appear in Comp. Methods Appl. Mech. Eng. Also available in ICFQuarterly Report, Lawrence Livermore National Laboratory, Livermore, CA, UCRL-LR-105821-96-6, 6:4 (1996), 165–180.

[5] See http://www.t12.lanl.gov/~lagrit.

Aleksei Shestakov (shestakov@llnl.gov) and Jose Milovich (milovich1@llnl.gov) are computational physicists at the Lawrence Livermore National Laboratory. David Kershaw (dkershaw@ms.com), a former Livermore physicist, is currently pursuing research in quantitative finance.