LA-UR- 97-4263
CONF-9709108--

TECOLOTE: AN OBJECT-ORIENTED FRAMEWORK FOR HYDRODYNAMICS PHYSICS

K.S. Holian, L.A. Ankeny, S.P. Clancy, J.H. Hall, J.C. Marshall, G.R. McNamara, J.W. Painter, and M.E. Zander

Los Alamos National Laboratory, Los Alamos, NM 87545, USA

Tecolote is an object-oriented framework for both developing and accessing a variety of hydrodynamics models. It is written in C++, and is in turn built on another framework -- Parallel Object-Oriented Methods and Applications (POOMA). The Tecolote framework is meant to provide modules (or building blocks) to put together hydrodynamics applications that can encompass a wide variety of physics models, numerical solution options, and underlying data storage schemes, although with only those modules activated at runtime that are necessary. Tecolote has been designed to separate physics from computer science, as much as humanly possible. The POOMA framework provides fields in C++ to Tecolote that are analagous to Fortran-90-like arrays in the way that they are used, but that, in addition, have underlying load balancing, message passing, and a special scheme for compact data storage. The POOMA fields can also have unique meshes associated with them that can allow more options than just the normal regularly-spaced Cartesian mesh. They also permit one-, two-, and three-dimensions to be immediately accessible to the code developer and code user.

RECEIVED
FEB 02 1998
OSTI

I. Introduction

In today's world, the advances in physics models and numerical models that are used in hydrodynamics are proceeding at an increasingly rapid pace. In addition, it seems that new computer architectures are being put out practically daily. In this environment of such mind-boggling change, it is imperative to have a framework, that is relatively portable, in which to do rapid prototype development. It is important to be able to build on what others have done, rather than using valuable time to implement, on another architecture, or in another language, that which has already been done.

Tecolote is an object-oriented framework, written in C++, that was designed for the development and implementation of a wide variety of hydrodynamics applications. It is also meant for the rapid development and testing of any kinds of models, numerical or physical, that are related to hydrodynamics. It already has a number of physics models (such as equation of state, material strength, and high explosive burn) implemented in the framework, that can be used as the building blocks for hydrodynamics applications. In addition, the basic modules required for an Eulerian hydrodynamics application have been written. We are now testing the Eulerian application on the new ASCI Blue Mountain computer at Los Alamos, in order to get a feel for the efficiency of an application that is written in the framework.

II. Philosophy

Before even designing the framework, we laid out a groundwork for the features that we were determined to have in any framework that we designed. We have kept these in mind throughout the design and implementation of Tecolote, and we feel that we have accomplished these principles.

19980327 068

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED

MASTER

DTIC QUALITY INSPECTED 3

First, we wanted to have a framework that was object-oriented in nature. This means that modules are as independent from each other as possible. With (well-designed) object-oriented coding, in principle the addition of new modules should be easier and more trouble-free than it is in a procedural code.

We also wanted to be able to code in such a way that the computer code would look as much like the original physics equations as possible. And, we did not want the high-level physics coding to change if new underlying mesh geometries were added to the possibilities, as would almost surely be the case if an alternative Lagrangian-Eulerian (ALE) option were added to the framework. In addition, we wanted the applications to be able to work in one, two, or three dimensions. Related to the flexibility of mesh geometry requirement, was the requirement that we be able to represent mesh-wide variables on a variety of centerings. An example of this would be a staggered-mesh application that uses cell-centered state variables, but vertex-centered velocities. Various physics operators, such as the divergence, need to know the centering of a variable, in order to do the calculation correctly.

Another important requirement was portability. With the new computer architectures that have multiple processors (or multiple boxes that each contain multiple processors), the question of portability is more difficult, yet also more crucial. We wanted to be able to code in such a way that, at the highest level, we would not have to clutter the coding with provisions for message passing and load balancing. We felt that this should be done at a lower level, in such a way that these could be tuned for different architectures.

III. POOMA Library

An important aspect of the Tecolote framework is that it is built on the POOMA library [1] (also written in C++). This library provides fields for Tecolote, which are similar to Fortran-90 arrays, but with extra features. The POOMA fields, like Fortran-90 arrays, automatically take care of message passing on platforms with multiple physical processors. But they also have a scheme for more efficient memory storage, and will have provisions for automatic load balancing in the future. Both of the former features are based on the concept of virtual nodes. In addition, the POOMA fields can be laid on top of different types of meshes, not just the standard Cartesian meshes of Eulerian codes. This will allow for future development of hydrocode applications that are purely or partially Lagrangian in nature. The POOMA library takes care of performing mathematical operations, such as divergence and gradient, correctly for a given mesh geometry.

A. Compressed Storage

As mentioned above, the compressed storage of POOMA fields is based on the virtual node idea. The physical geometry of the mesh over which the application is operating is divided up into subunits, called virtual nodes. When a field variable is constant over the entire virtual node, the storage for that field variable is collapsed down to just one value for the whole subunit, rather than one value for each point of the grid in that subunit. This is somewhat analagous to the compression that is regularly practiced on graphics files that contain a great number of repeated values.

In Eulerian hydrocodes, it is a common occurrence to initially include a great deal of mesh with void in it, so that the materials in a problem may expand into the void. Often many zeroes are stored for a field variable, wasting a great deal of storage space. The POOMA fields solve this problem. Another situation is that in which one has many materials in a problem. Each material may have a number of state variable fields associated with it, such as density, energy, and pressure. These fields need to be represented on the entire mesh of the problem. However, each material may be found only in a small part of the mesh. Again, the fields can be collapsed down to storing only one value across an entire virtual node.

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, 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 any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

In addition, POOMA automatically keeps track of the fields. As material moves across a mesh, virtual nodes expand and compress automatically, depending on the state of the field.

B. Load Balancing

Virtual nodes will also be used for automatic load balancing on multiple-physical-processor machines, that is planned for a future implementation. The idea is that, ideally, the problem in question would be divided into far more virtual nodes than there are physical processors. Initially, each physical processor would get one virtual node to work on in an expression. When a physical processor finished its work, it would get another virtual node to work on. Hopefully, no physical processor would be sitting idle for long. Implementation of the load balancing scheme is especially important, given the compressed storage of the fields.

IV. Tecolote Physics Library

The most important thing about Tecolote is that it provides a framework in which to develop new hydrodynamics models and methods. And we are trying to encourage code reuse not only in the computer science arena, but also in the physics area as well. With that in mine, we plan to have available a wide variety of physical and numerical solution modules so that the developer of a new method or model does not have to waste time recoding, and so that his or her valuable time need only be spent on the new problem of interest.

So far, the modules that we have implemented are those that are needed by an Eulerian hydrodynamics code (which is the first application that we have that uses the Tecolote framework). These include such things as a predictor-corrector Lagrangian module, and a split-advection rezoner module, based on the vanLeer scheme. In addition, we have a module that uses the Youngs' reconstruction method to calculate the interfaces between materials in a mixed-material cell.

In addition, we have a number of analytic equations of state that can be used, for example, by the Lagrangian module. We will soon also have the SESAME tabular equation-of-state option [2].

We plan to soon implement an elasto-plastic module that calculates material strength, and that can use a variety of models for the yield condition of a material. Also, there will be a variety of options for such other physical behavior as high explosive burn and fracture.

As time goes on, and as physicist-programmers add options to the physics library part of Tecolote, there will be even more building blocks that will be available for hydrodynamics applications.

V. Tecolote Computer Science Features

Perhaps the most imporant part of the Tecolote complex will be its physics capabilities. However, we have implemented a number of computer science features that have been designed with the idea of easing the work required for new model developers. Following are the explanations of several of these features.

A. Operators for Different Mesh Geometries

POOMA fields have the capability of being laid out on different types of meshes. In addition, each layout can be in one, two, or three dimensions. Of course, different geometrical configurations require different solutions for operators, such as the gradient operator. POOMA has implemented default operators for each type of mesh and dimension

that is currently available. However, there are often different numerical methods for calculating the results of a physical operation.

Tecolote uses the concept of operator objects, which can either call the default POOMA operator, or can override the default with a different customized calculation. This is a fairly straightforward thing to do, since we are using the object-oriented features of C++ for the framework.

This was an important feature to have when we implemented the axisymmetric geometry for a two-dimensional mesh. The mesh itself is laid out in regular Cartesian coordinates. But, instead of the Cartesian x-y geometry, the mesh is meant to represent an axisymmetric r-z geometry, with the axis of rotation around the z-axis, and the r-coordinate defining the radius. The default POOMA operators in this case are for the Cartesian x-y geometry. We were able to easily override the default operators, and provide the correct ones for an axisymmetric geometry (but only in the case where that is desired).

Therefore, our Eulerian application can currently be compiled and run in one-, two, or three-dimensional Cartesian geometry, as well as two-dimensional axisymmetric geometry. We will add other options, such as one-dimensional spherical geometry, as interest warrants.

Another important characteristic of the operator objects is that high-level coding that uses an operator is written once, and will never have to be changed (with the addition of new geometry options). This relieves the programmer of tedious code maintenance, and confusing arrays of if tests for different mesh options. For example, following is a line of coding from the Lagrangian predictor-corrector module:

```
Velocity = Old.Velocity + Div(Stress, DivStress) * dt / ( VertDensity + EPSILON )
```

This line of coding updates the velocity according to the divergence of the stress. Velocity and Old.Velocity are vector fields describing the velocity at two different time levels (required for a predictor-corrector algorithm). The divergence operator Div operates on a tensor field, in this case called Stress. DivStress is a reference to a field that stores the output of the divergence operation. (It is provided solely for efficiency reasons.) The parameter dt is a constant representing the time step. VertDensity is a scalar field that is the average density at the vertices of the mesh (calculated by averaging the densities around each vertex). EPSILON is a macro that just represents a very small number, to avoid a divide by zero.

The impressive thing is that this line of coding works for all mesh geometries and all dimensions. It will never have to be changed.

The actual operator used for a particular application and particular problem is chosen at run-time in the input file. This will be discussed in more detail below.

B. DataDirectory

One traditional problem that needs to be solved in any hydrocode is how to get the different modules to communicate with one another. Different physics packages need to be able to access different sets of variables, to either modify or to use those variables. For example, a material strength yield surface model for a particular material might need to access the pressure, density, stress deviators, and strain rate for that material, but would not need to access other state variables. Whereas another model of the same type might need the density, energy, and stress deviators. The usual way of solving this problem is to pass the required variables, or pointers to arrays of variables, in the subroutine call for the model. This can lead to hard-to-maintain, or confusing coding.

The way that we have dealt with this situation in Tecolote is with what we call a DataDirectory. The DataDirectory is basically a map that associates the address of a field of variables with a unique name (represented by a string). Now, we need only pass the address of (or a reference to) a DataDirectory into a physics module (rather than a whole list

of field addresses). That module, then, can look up the fields that it requires in the DataDirectory. A physics module is allowed also to put a new field into the directory that can then be shared with other modules. Actually, the DataDirectory can not only store locations to POOMA fields, but also locations for any types of variables (e.g., integer, Boolean, float, string, and arrays of the former) that a physics module might want to share with other modules.

The DataDirectory has a tree structure, that is analagous to the Unix directory system. This is to accomodate the fact that, in a given application, one might require two sets of variables at different snapshots in time, or sets of state variables for the different materials.

Perhaps it is best to illustrate the DataDirectory by an example of how it would be arranged for the Eulerian application. (Each application and each unique problem running that application work together to create a unique DataDirectory.) For the Eulerian code, there are a few fields of data that reside at the highest level of the DataDirectory. That is, they are time- and material-independent. An example of this would be the field that holds the cell invariant Eulerian volumes. Then, the next level down in the DataDirectory would include fields of variables that are required to exist simultaneously at two different time levels for the Lagrangian predictor-corrector step of the calculation. These would be such fields as the average pressure or density in a cell, for which there is only one value in a cell, no matter how many materials in the cell. Finally, at the next level down are the fields that are material dependent, such as density, pressure, and energy. A unique copy of each of these fields is required for each material. Additionally, for each field for each material, two copies are required for the two times of the predictor-corrector algorithm. It is also possible to imagine, in the future, that there might be a further level down in the hierarchy. A given material might have different species, that each require their own fields. The DataDirectory is infinitely adaptable.

C. Simplified I/O

Tecolote has many features with regard to I/O that make it more or less automatic. The goal is to free the physicist-programmer from spending valuable time in coding either the input to his/her new model, or adding new output to the various types of dumps written out.

In the case of model parameters, the programmer can merely register these parameters as being possible input. This is accomplished by the fact that models are objects, the parameters are object members, and the framework knows that certain "persistent" parameters can be initialized in an input file that creates specific instances of a model object. Of course, all these object members have default values, in case the parameters are not initialized in the input file.

Tecolote outputs a number of different types of dumps. For example, there is a restart dump that contains that information required to restart a problem from a particular snapshot in time. Additionally, there are special dumps of fields that one might want to look at graphically, and various kinds of ASCII dumps to keep track of the progress of a problem. As new models are added to the physics repertoire, there will be new constants and variable fields that should be included in the output files. Tecolote has a method for registering variables for diferent kinds of output. Once the registration is specified, the model programmer need do nothing further. All appropriate output will be taken care of automatically.

D. Run-Time Instantiation

The algorithm and model objects required for a given simulation, using a given application, are created at run-time from an ASCII input file. This has several advantages.

One is that space on the computer is not set aside until run-time. This is particularly important when using a large framework containing many algorithms and models, perhaps most of which will not be required for a given run.

Another is that the code user can create a customized application just from the input file, using the existing algorithms and models provided by the framework. No recoding of the main program is required.

Thirdly, one can easily compare a new model, algorithm, or operator with another of a like kind, and be assured that all other parts of the calculation are identical. This removes the comparing-apples-and-oranges problem in comparing the effectiveness and accuracy of different ways of doing the same operation. Also, it makes it quite easy to determine differences in timings for the different methods of accomplishing the same goal.

VI. Conejo -- An Eulerian Application

The first application that we have implemented using the Tecolote is an Eulerian application, which we call Conejo. The physics in the application is the same that is represented in the Mesa/Pagosa [3, 4] family of codes. However, there is one difference in that the application can be compiled for one-, two-, or three-dimensional Cartesian geometry, or two-dimension axisymmetric geometry.

Conejo has a Lagrangian predictor-corrector step in which the vertices of the cells change according to the PdV work done on a cell. Then there is an advection step in which the material is remapped back onto the original Eulerian mesh. The remap step uses the vanLeer method of limiting to reduce noise in the calculation. The amount of material advected across a cell interface in the remap step for a mixed-material cell is calculated using the Youngs' method to place interfaces between the materials in the cell. Conejo uses operator splitting for the advection remap, and therefore, only considers one direction at a time.

We have been running some simple test problems using Conejo, so that we can get a feel for the maximum size of problem that we will be able to run (for an Eulerian application), as well as to get a feeling for the timings, as compared to our other Eulerian codes.

The platform that we have been using for the test problems is the new Cray/SGI supercomputer (known as ASCI Mountain Blue) that was purchased by Los Alamos National Laboratory. The initial configuration of the machine (more capacity will be added later) is 8 boxes with 32 processors each. Each box has 16 Gbytes. The theoretical peak speed is 400 MFlops per processor, although the likely speed for a real optimized application is about 50 Mflops per processor.

At this point, it is a bit premature to report timings for our problems. More work is required to completely implement the compressed field structure. However, we are already running within a factor of 4 (on the Cray/SGI machine) of the Eulerian hydrocode written in C, that also runs on the same machine.

We have been able to run a 256X256X256 7-material problem on one box of ASCI Blue. Again, our capacity will increase as the field compression capability is optimized. We are just beginning to look at various optimization issues now.

VII. References

1. [to be provided]

2. K.S. Holian, "T-4 handbook of material properties data bases, Vol. 1c: equations of state", Los Alamos National Laboratory Report LA-10160-MS (1984, unpublished).

3. K.S. Holian, D.A. Mandell, T.F. Adams, F.L. Addessio, J.R. Baumgardner, andS.M. Mosso, "MESA: A 3-d computer code for armor/anti-armor applications", in Proceedings of the Supercomputing World Conference (San Diego, CA, June 16-19, 1990).

4. D.B. Kothe, J.R. Baumgardner, S.T. Bennion, J.H. Cerutti, B.J. Daly, K.S. Holian, E.M. Kober, S.J. Mosso, J.W. Painter, R.D. Smith, W.H. Spangenberg, M.D. Torrey, "A parallel finite-difference Eulerian method for transient three-dimensional multi-material deformation and fluid flow", Los Alamos National Laboratory Report LA-UR-93-2400 (1993, unpublished).

M98002662

Report Number (14) LA-UR--97-4263
CONF-9709108--

Publ. Date (11) 199712
Sponsor Code (18) DOE/DP, XF
UC Category (19) UC-700, DOE/ER

DOE