LA-UR- 98-2999
CONF-980791--

Tecolote: An Object-Oriented Framework for Physics Development

The Tecolote Team
Jean Marshall[1], Lee Ankeny[1], Sean Clancy[1], John Hall[1], Jody Heiken[1], Kathleen Holian[1], Stephen Lee[1], Guy McNamara[1], James Painter[1], Mark Zander[1], Richard Graham[2]

The POOMA Team
Julian Cummings[3], Scott Haney[3], Steven Karmesin[3], William Humphrey[3], John Reynders[3], Timothy Williams[3]

[1]Los Alamos National Laboratory, X-CI, MS F663, Los Alamos, NM 87545, USA,
{jcm, laa, spc, jxyh, jheiken, ksh, srlee, mcnamara, jwp, zander}@lanl.gov
[2]Silicon Graphics Inc., Mountain View, CA 94043, USA,
rlgraham@lanl.gov
[3]Los Alamos National Laboratory, ACL, Los Alamos, NM 87545, USA,
{julianc, swh, srk, wfh, reynders, zippy}@lanl.gov

Abstract. We describe a C++ physics development environment, called the Tecolote Framework, which allows model developers to work more efficiently and accurately. This Framework contains a variety of meshes, operators, and parallel fields, as well as an input/output (I/O) subsytem and graphics capabilities. Model developers can inherit Tecolote's generic model interface and use the Framework's high-level field and operator components to write parallel physics equations. New Tecolote models are easily registered with the Framework, and they can be built and called directly from the input file, which greatly expedites model installation. In the process of developing an extensible and robust framework, we have found appealing solutions to some of the serious problems we encounter when parallelizing and extending our older codes. We also discuss memory and performance issues for a large hydrodynamics application built in this Framework.

1 Introduction: a Parallel Prototyping Environment

We are developing a large C++ physics code to run on the teraflop parallel platforms that are being built by the Accelerated Strategic Computing Initiative program. (ASCI spans three national laboratories and includes both hardware and software development for sophisticated scientific applications.) Instead of writing a single-purpose application, our strategy has been to generalize our code infrastructure to create an object-oriented framework that can be used to jumpstart a variety of new physics-modeling efforts.

DISCLAIMER

Our physics-prototyping environment, called the Tecolote Framework [1], is built on top of the Parallel Object-Oriented Methods and Applications (POOMA) Library [2], which provides a portable and robust parallel understructure. Model developers using the Tecolote Framework can be confident that as soon as their model runs correctly on a workstation, it will also give correct results in parallel on the large ASCI platforms.

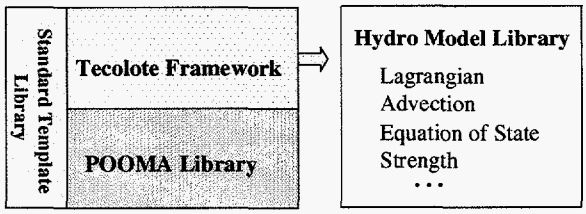


Fig. 1. Tecolote is built on top of the POOMA Library, which enables parallelism in the Framework. Both Tecolote and POOMA rely heavily on the C++ Standard Template Library. Physics models are built using Framework components, and they are separated from the computer-science Framework into a series of physics libraries

Several different physics applications now take advantage of the Tecolote Framework. Our first application is a 1D/2D/3D multimaterial, Eulerian hydrodynamics code, which has recently run a 60-million-cell calculation across 1000 Silicon Graphics (SGI) processors. In addition, a Monte Carlo neutron-transport application [3] and a new hydrodynamics code, which has multiple velocities to allow large shears between materials, are also using this Framework.

2 Tecolote is a Component Architecture with Connectors

One of the Tecolote project's goals is to put in place a generic infrastructure that can build and support a variety of physics applications. To that end, we provide a set of high-level building components, such as meshes, operators, and parallel fields, as well as an I/O subsystem and graphics capabilities. In addition, the Framework supplies a generic model interface and a database that allows data to be safely shared among models.

The mechanism for gluing these components together into a complete application is available through the Framework's compact, functional language with its meta-data registration process. Each new Framework application must register the components it needs. Once a component is registered, the Framework can build it, initialize its parameters, and run its methods directly from instructions in the input file, which greatly expedites model installation.

A skeleton application that compiles and runs, but has no registered physics models, is available to new applications. This skeleton application is a complete support system, including parallelism, data management, I/O, and graphics. A new

application can be built by dropping new physics models into this existing support code.

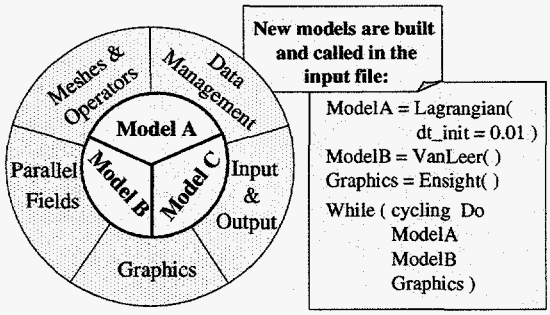


Fig. 2. A complete application can be built by installing new physics models into the existing Tecolote infrastructure

3 Object-Oriented Solutions to Code-Development Challenges

The Tecolote Project is dedicated to developing more robust and extensible coding methods for large applications that evolve over decades. By exploring generic programming and object-oriented techniques, in addition to many of the newer language features of C++, we have found appealing solutions to some of the serious problems we encounter when parallelizing and extending our older procedural codes. We describe some of these solutions to illustrate how model developers can work more efficiently and accurately using this Framework.

3.1 Touch Only One Existing File To Add A New Model

Previously, when adding a new model to our large procedural codes, developers had to modify as many as 50 existing routines. These routines did specific tasks for all the models, such as initializing memory, parsing input, performing general I/O, as well as managing the branching logic that traversed the models selected by the user at run time. A model developer had to hunt through thousands of lines of code to determine where to put the new model's components and behaviors. In addition, touching that many existing routines greatly increased the possibility of unintended side effects.

The Tecolote Framework uses encapsulation to ensure that each model is as self-contained as possible; therefore, we can install a new model into the Framework by simply adding it to a list of registered meta classes, which involves adding just four lines of meta information. This one registration procedure connects a new model into the Framework's full support system, including the run-time functional language, the database, and the I/O subsystem. It is important to realize that this one registration procedure allows a model installer to call a new model directly from the input file

without adding *any* branching logic to the code. Because only one existing file is touched, unintended side effects are rare.

3.2 Handle Global Data in a Safe and Extensible Manner: DataDirectory

In our large physics applications, some fields must be shared among models. Neither global common blocks nor passing data via argument lists has successfully solved the global data problem for our applications. Global common blocks, in practice, expose more data than is necessary. Passing by argument often requires several levels of calls before the data reaches the underlying work routines, and this process obscures which routines are actually changing the data. Moreover, both of these methods depend on implicit ordering, which is fragile.

The Tecolote Framework has a simple hierarchical database, called the DataDirectory, to help handle global data in a safe and extensible manner. Each application can build a DataDirectory hierarchy that reflects its own internal structure, and this hierarchy can be tailored to package data appropriately for use in the application's physics models.

By default, the Tecolote Framework instantiates a root-level DataDirectory for each application. Our hydrodynamics codes also create a hierarchy for materials, including a parent DataDirectory, called the MaterialSet, and a child directory for each material in the current calculation. The MaterialSet contains a list of all the materials as well as references to the material-independent fields (e.g., velocity). Each material directory contains a unique collection of references to fields used by the material-specific models. Some chemistry models require further specialization of a material into its chemical species, and the DataDirectory hierarchy easily extends to this specialization. A separate DataDirectory holds the various I/O modules.

The DataDirectory is a Standard Template Library multimap that can store *any type* of data by string name and reference. To get a reference to global data, a physics model must first have access to a particular DataDirectory in the hierarchy, and then it must retrieve the data reference explicitly by name. For safety, the data type is checked at run time to catch any type mismatch. Models in the Tecolote Framework now access only the global data they need, and this data is retrieved explicitly by name (instead of implicitly by order) categorically marking which routines can change a global data item:

DataDirectory access-function prototype:

```
Data&      GET(dataName, DataType, DataDirectory);
```

Example call:

```
Density = GET("Density", ScalarField, Copper);
```

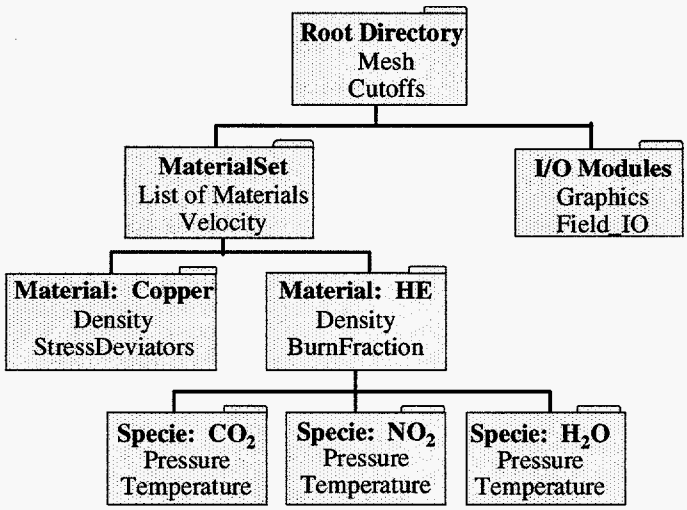


Fig. 3. A sample DataDirectory hierarchy for a hydrodynamics code. DataDirectory hierarchies can be used to reflect the internal structure of an application and to package data for use in physics models

3.3 Facilitate Model Installation with a Generic Interface

To easily add new models to the Tecolote Framework and be able to call them interchangeably at run time, we needed a generic model interface. However, because different models use different dependent and independent state variables, there was no obvious interface. Generalization was further complicated by the fact that multimaterial models must access all materials in the problem description, while single-material models work on only one material.

The DataDirectory is the key element of the Tecolote model interface. All models inherit from the same model base class, whose constructor receives a DataDirectory containing the appropriate data to be extracted by the model. Multimaterial models, for example, receive a MaterialSet DataDirectory, and single-material models receive a Material directory. Each model then extracts from its particular DataDirectory the various data items it needs, usually at construction time.

A DataDirectory search starts at an input directory location and continues up the hierarchy until the first match is located. This hierarchical search allows, for example, a single-material model to request both material-dependent and material-independent fields in the same manner, starting from the material directory level.

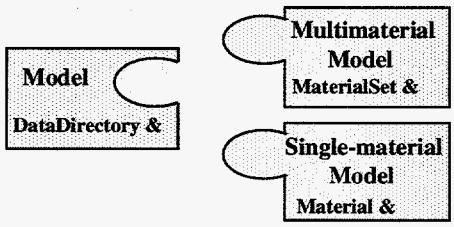


Fig. 4. A generic model interface, which receives a DataDirectory reference at construction time. Multimaterial models receive a MaterialSet DataDirectory, and single-material models receive a Material directory containing fields for a single material. The models then get references to the particular data items they need through the DataDirectory's access functions

3.4 Hide the Computer Science, so Physicists Write Equation-Like Code

The POOMA Field, an F90-like mechanism for data storage and computation, is the underlying parallel object in the Tecolote Framework. This Field is templated on type, dimension, mesh, and centering (e.g., cell or vertex), allowing ultimate flexibility for algorithm development. The Field handles domain decomposition, iterations over the data, external boundary conditions, and interprocessor communication, masking these mechanical details from the physics code.

The Framework also includes a variety of mesh options, with default operators defined on each mesh type. Customized operators can easily be added to the Framework and swapped into a calculation via the input file.

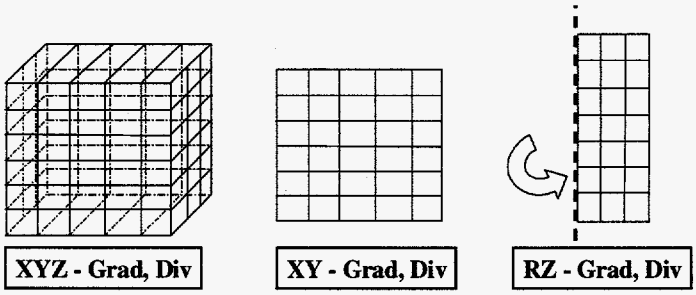


Fig.5. Operators are defined for each mesh type. Processor-to-processor communication is built into the operators, when required, so they operate across the entire mesh, even when it is distributed across multiple processors

These field and operator abstractions allow model developers to write high-level code that looks much like the physics equations themselves. Also, because details of the dimension and mesh type are hidden, it is possible to write code that works on a

variety of meshes. Our Eulerian hydrodynamics code, for example, traverses identical lines of code, regardless of whether the code has been compiled for X, XY, XYZ, or RZ geometry (dimension and mesh are template parameters, which are set at compile time).

This equation from our Lagrangian predictor-corrector model updates the velocity field across the entire mesh, even when it is distributed across many processors:

```
Velocity = Old.Velocity + Div( Stress, DivStress )
                        * dt / ( Density + eps );
```

Because there is no reference to dimension or mesh type in this line of code, it works identically in 1D, 2D, and 3D, as well as for Cartesian and cylindrical geometry. One, two, or three components of the velocity field are updated in the one expression, depending upon the vector dimension. The Div (divergence) operator is a stencil operation that involves communication occurring "under the hood." Calculational efficiency is maintained through extensive use of expression templates.

3.5 Conceal Messy Details of Compressed Data Storage

Our Eulerian hydrodynamics code uses interface reconstruction to track multiple materials within a computational cell. Because any given material exists on only part of the mesh, a full-mesh, material-dependent field contains many irrelevant cells, which we often dummy out with zero values.

If we choose not to store these irrelevant zeros, we must use an indirect-addressing scheme, which can reduce code readability and increase coding errors. However, by hiding both the data layout and the iterations over the data, it is possible to compress the data without sacrificing code readability. For example, the expression below evaluates a complete material pressure Field without any reference to the compressed data storage in our POOMA sub-layer:

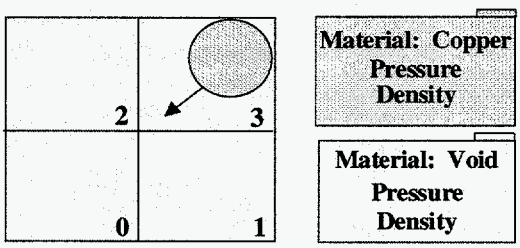
```
Pressure = Density * Energy * ( gamma-1.0 );
```

POOMA automatically compresses a Field on any subdomain where a Field value is constant (e.g., 0.0); consequently, POOMA stores only a single element for that entire subdomain. The compression ratio usually increases with the number of subdomains, or virtual processes (vnodes), so it is often advantageous to define more vnodes than the actual number of physical processors.

The above material equation expands, inside the POOMA layer, to an outer loop over the vnodes on the processor and an inner loop over each vnode element. If a material is absent in a particular vnode, its Fields are compressed to a single element per Field. Thus, the equation reduces to one trivial computation for the entire vnode, saving computation time as well as storage.

Vnode compression saves more memory as the number of problem materials increases. This fact is significant for users of our Eulerian code, because they are often forced to define duplicate materials to assure the correct ordering of materials

during the advection phase of the calculation. Therefore, thirty-five to fifty materials in a calculation is common, and data compression is critical.



3 compressed subdomains for Copper fields:

0	0	0	Field elements for subdomain 3

4 compressed subdomains for Void fields:

0	0	0	0

Fig. 6. POOMA Fields are automatically compressed on subdomains that contain a constant value. Because the copper sphere above exists entirely in subdomain three, copper fields on the other domains contain only zeros. Thus, these zero fields are stored as one compressed value per domain. Void fields are zero in all domains, so they are always stored compressed

3.6 Load Balance Dynamically with Virtual Nodes

In addition to providing a mechanism for data compression, POOMA vnodes present a conceptually easy mechanism for dynamic load balancing during a parallel calculation. If the problem domain is divided into many more vnodes than physical processors, these encapsulated packets can be dynamically distributed across the processors according to some weighting function (e.g., amount of work or compression ratio). Load balancing with vnodes is particularly successful for the particle-transport code that is using the Framework.

The hydrodynamics codes, however, currently pay a penalty for using a large number of vnodes. These codes require frequent vnode-to-vnode communication, which is handled by passing messages to update ghost-cells. Therefore, there is a trade-off between increasing the number of vnodes to facilitate load balancing and accommodating the additional cpu and memory requirements of the larger surface-to-volume ratio. However, POOMA is moving toward thread-based vnodes, which will use the shared memory facility inside a multiprocessor machine to access neighbor data. This new paradigm should help make vnode load balancing more effective for our hydrodynamics applications.

3.7 Develop and Test New Models on a Workstation and Then Run in Parallel

Physics modelers who use the Framework are able to develop and test their models on a single-processor workstation where they have debuggers and a relatively friendly development environment. When they are satisfied their model works correctly on one processor, they can be confident that it will also work in parallel on the ASCI platforms because the POOMA parallel infrastructure has already been thoroughly tested and is now quite robust.

Framework code does not have to make explicit synchronization and communication calls because the POOMA layer automatically handles these issues. Therefore, there is little chance that new code will introduce a message mismatch or race condition that could halt parallel development for weeks.

4 Size and Performance Issues

As our end-users continue to require larger and faster calculations for ASCI applications, we continue to develop more sophisticated platform and physics-specific optimizations. We have run a 60-million-cell hydrodynamics calculation across 1000 processors, which is a larger calculation than we can run on the earlier production machines. Our ASCI machine will increase to 6000 SGI processors by the end of 1998, providing an opportunity for even larger calculations.

The challenges to code developers of these massive applications are enormous. Parallel robustness and encapsulation to prevent side effects are of the utmost importance, because a race condition or a subtle bug in a 6000-processor calculation will be nearly impossible to track down.

Scalar efficiency as well as good parallel scaling become more critical as we run more finely-resolved calculations. As meshes refine, time steps decrease, forcing us to run more cycles in addition to computing more cells. (For example, halving the mesh in 3D requires sixteen times the compute power: a factor of eight for the 3D geometry refinement and a factor of two for the time-step refinement.) Dynamic load balancing is an issue, particularly for codes with mixed-cell treatments because the mixed cells, though usually a small percentage of the total cells, will dominate the grind time.

The Tecolote Project is trying to address these challenges. The Framework relies on the robust communications in the POOMA Field to give a worry-free parallelism for our present applications as well as future development. In addition, the POOMA Field abstraction with its overloaded operators and Portable Expression Template Engine (PETE), provides an opportunity to perform optimizations in the number-crunching support layers, without having to rewrite the physics coding itself. The mechanism for dynamic load balancing also exists in the POOMA layer through its virtual nodes.

Optimizations over the last year have improved the performance of our Eulerian hydrodynamics code, and it is scaling well up to 512 processors. Our C++ grind

times are comparable to those of the predecessor F90 code. We suspect that our best efficiency option now lies in reducing the number of ghost cells so we can increase the number of virtual nodes for better compression and load balancing. Work on this front is proceeding.

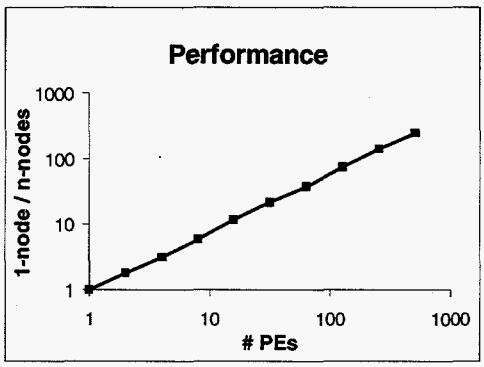


Fig. 7. Performance scaling for an Eulerian hydrodynamics code built with the Tecolote Framework

5 Summary

The Tecolote Framework provides physics application developers with a parallel computer-science support layer that frees them to concentrate almost entirely on physics algorithms. This Framework is already successfully supporting a variety of physics options, and we continue to add capability and optimizations.

References

1. Holian, K., Ankeny, A., Clancy, S., Hall, J., Marshall, J., McNamara, G., Painter, J., Zander, M.: TECOLOTE: A Framework for Hydrodynamics Physics. Los Alamos National Laboratory document LA-UR 97-4263 (1997)
2. Humphrey, W., Karemsin, S., Bassetti, F., Reynders, J.: Optimization of Data Parallel Field Expressions in the POOMA Framework. In: Ishikawa, Y., Oldehoeft, R., Reynders, J., Tholburn, M. (eds): Scientific Computing in Object-Oriented Parallel Environments. Lecture Notes in Computer Science, Vol. 1343. Springer-Verlag, Berlin Heidelberg New York (1997)
3. Lee, S., Cummings, J., Nolen, S., Keen, M.: MC++ and a Transport Physics Framework. In: Ishikawa, Y., Oldehoeft, R., Reynders, J., Tholburn, M. (eds): Scientific Computing in Object-Oriented Parallel Environments. Lecture Notes in Computer Science, Vol. 1343. Springer-Verlag, Berlin Heidelberg New York (1997)