



MPI Applications: How We Use MPI in ALEGRA

Allen C. Robinson
Richard R. Drake

Sandia CSRI Workshop on
Next-generation scalable applications: When MPI-only is not enough

June 3-5, 2008

Bishop's Lodge Resort
Santa Fe, New Mexico

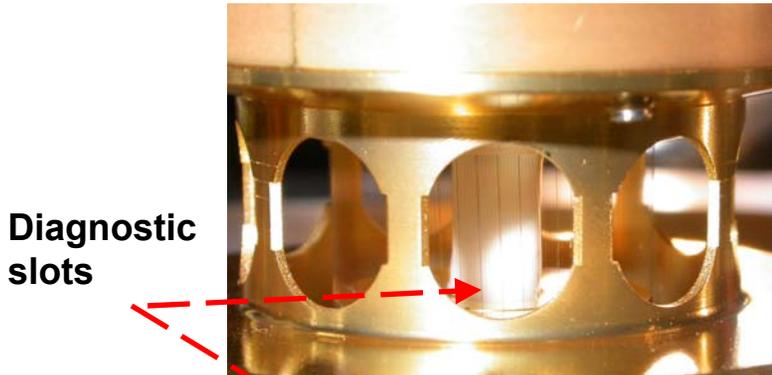


Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy's National Nuclear Security Administration under Contract DE-AC04-94AL84000.

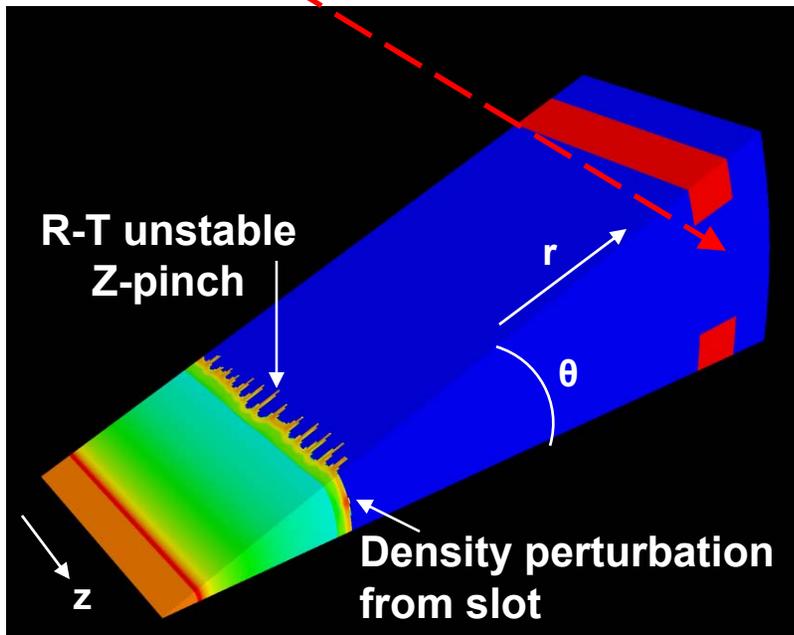




ALEGRA is an arbitrary Lagrangian Eulerian (ALE) multi-material and multi-physics code



Diagnostic slots



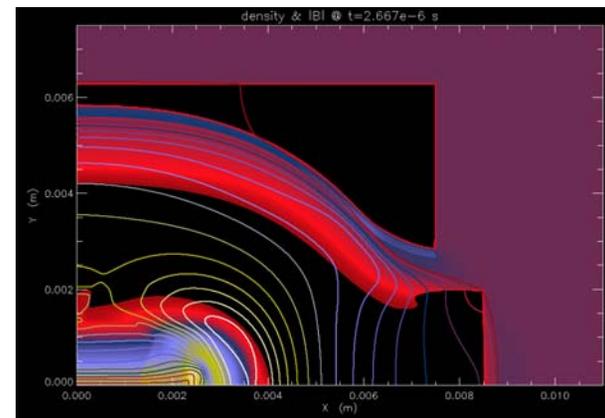
Complicated geometries

Multiple materials

Material interfaces

Implicit multilevel solvers

Complicated material models





Multi-material, Multi-physics Data Layout

Code memory is laid out with respect to topological entities.

Multi-material dynamic memory management and per-element adaptivity infrastructure issues drove this.

Most of the cost of multi-physics is in solvers so memory layout is less of an issue. Solver performance is most critical in this case.

However, the current layout has performance implications for simple Lagrangian hydrodynamics.



How we use MPI

We have wrapper functions for all MPI calls in the NEVADA mesh infrastructure. Most developers don't really program directly in MPI. The Mesh API handles it. Linear solver MPI coding is also not visible to most developers.

An easy description for which variables to pack and communicate is available.

We don't use MPI-2

Single layer of "ghost" elements on unstructured mesh are updated using a IRECV and blocking SEND methodology. Easy to control processor entity or ghost entity updates.

We are migrating toward using TRILINOS/EPETRA for matrix assembly to remove dependencies on ghost elements and relieve the mesh infrastructure of ghost element support requirements for solvers.

Trilinos/ML performance can be a big cost. Load balancing in multilevel solves is crucial to maintain performance.

Global operations with an entity length equal to the number of processors are frequently used in parallel algorithms.

Some particle information is communicated between processors.



What about the future?

Currently, developers are relatively happy with our programming paradigms.

I understand that our future platforms will likely have many more cores than memory bandwidth to feed them.

Our current algorithms and software depends heavily on decent bandwidth to memory. Algorithms and code base will not easily be modified but could be modified if a clear long term win is visible.

We will need to find ways to reliably and predictably get more out of the available memory bandwidth.

Minor software/algorithmic/process flaws today may be near fatal weaknesses tomorrow from scalability, performance and robustness points of view.



Final thoughts

MPI works because developers are given full control over an explicit communications mechanism associated with distinct memory spaces.

There can be unexplained and difficult to analyze contention/performance issues on multi-core architectures. This general situation has been true for some time with cache based systems.

The “typical” application developer may understand performance issues in a general sense but does not have a concrete, affordable process to ensure excellence. How can performance issues be made transparent?

Compiler writers by themselves will not be able to add much automatic portable parallel processing value. Explicit software/hardware mechanisms with immediate feedback to application developers will be much better.

To be successful the application developer needs a portable, obvious and testable way to manage their parallel streams and associated communication and memory accesses.