Preparation of Codes for Trinity

Courtenay T. Vaughan, Mahesh Rajan, Dennis C. Dinge, Clark R. Dohrmann, Micheal W. Glass, Kenneth J. Franko, Kendall H. Pierson, and Michael R. Tupek

Sandia National Laboratories

CUG Conference
April 26-30, 2015

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation,
a wholly owned subsidiary of Lockheed Martin Corporation, for the
U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

Trinity

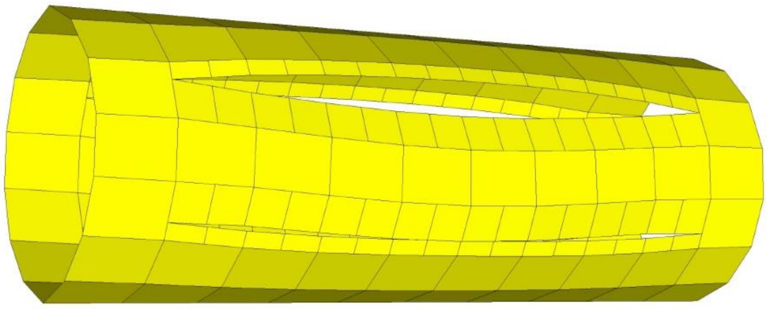
- Cray XC40
- Total of about 19000 nodes
  - About half are Intel Haswell with 2 processors per node and 16 cores per processor running at 2.3 GHz and 128 GB memory per node
  - About half are 60+ core Intel Knights Landing processors
- About 42 PetaFlops peak

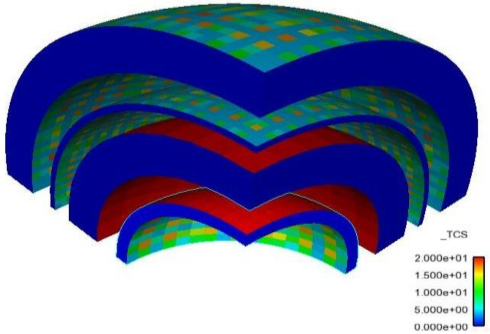
Intel and Cray Center of Excellence

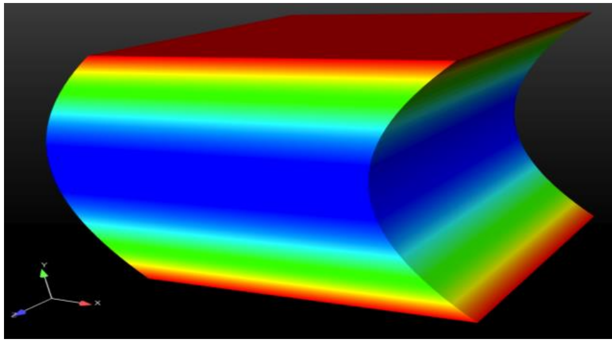
- Focus on SIERRA applications
  - SIERRA/Solid Mechanics (SM)
  - SIERRA/Aerodynamics
  - SIERRA/Structural Dynamics (SD)
- SIERRA is a large C++ framework
  - provides framework for several codes
  - Includes several Third Party Libraries
  - Contains common C++ classes and methods
  - Common infrastructure for parallel codes

Sandia National Laboratories

SIERRA/SM (Solid Mechanics)





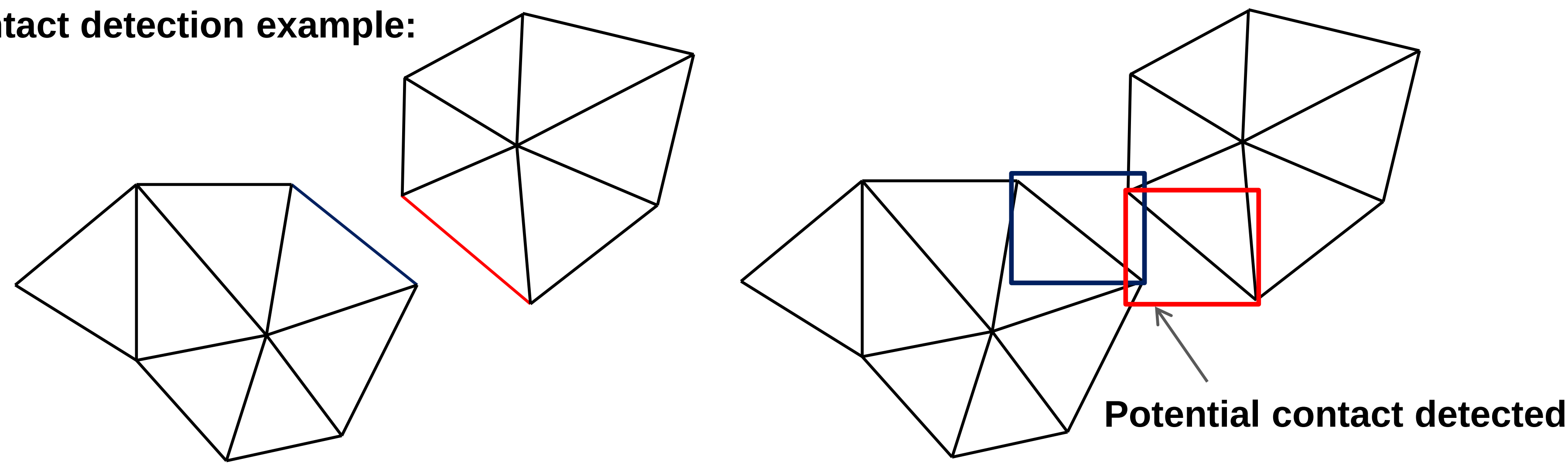


- **A general purpose massively parallel nonlinear solid mechanics finite element code for explicit transient dynamics, implicit transient dynamics and quasi-statics analysis.**
- **Built upon extensive material, element, contact and solver libraries for analyzing challenging nonlinear mechanics problems for normal, abnormal, and hostile environments.**
- **Similar to LSDyna or Abaqus commercial software systems.**

SIERRA/SM Bottlenecks

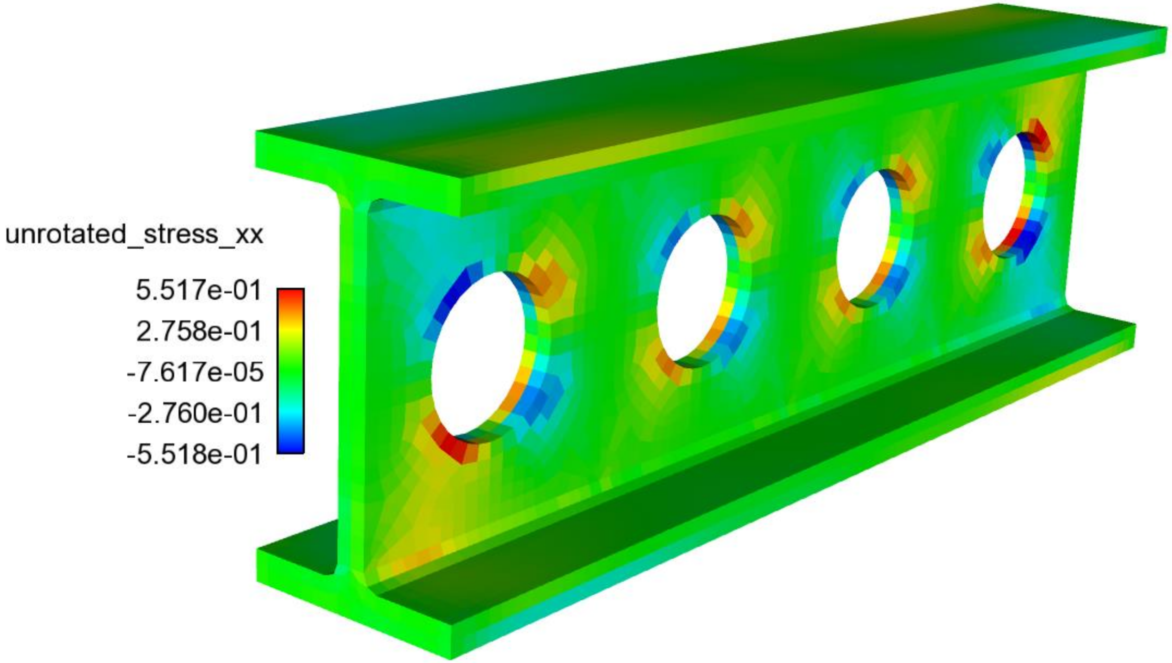
Application:	**Explicit dynamics with contact**	**Implicit with FETI pre-conditioner**	**Explicit dynamics w/o contact**
Hot spot:	Parallel **proximity search** and enforcing contact constraints	Serial **sparse direct solve**: matrix factorization and forward/backward solves	Assembling nonlinear **element residuals** and computing material response

Contact detection example:



I-Beam Problem (Quasi-Static)

-provided by Joe Bishop



<u>Mesh</u>:

- 3 Different mesh refinements: 8,576, 68,608, and 548,864 elements
- Mean Quadrature and SD hex elements

<u>Unique Features:</u>

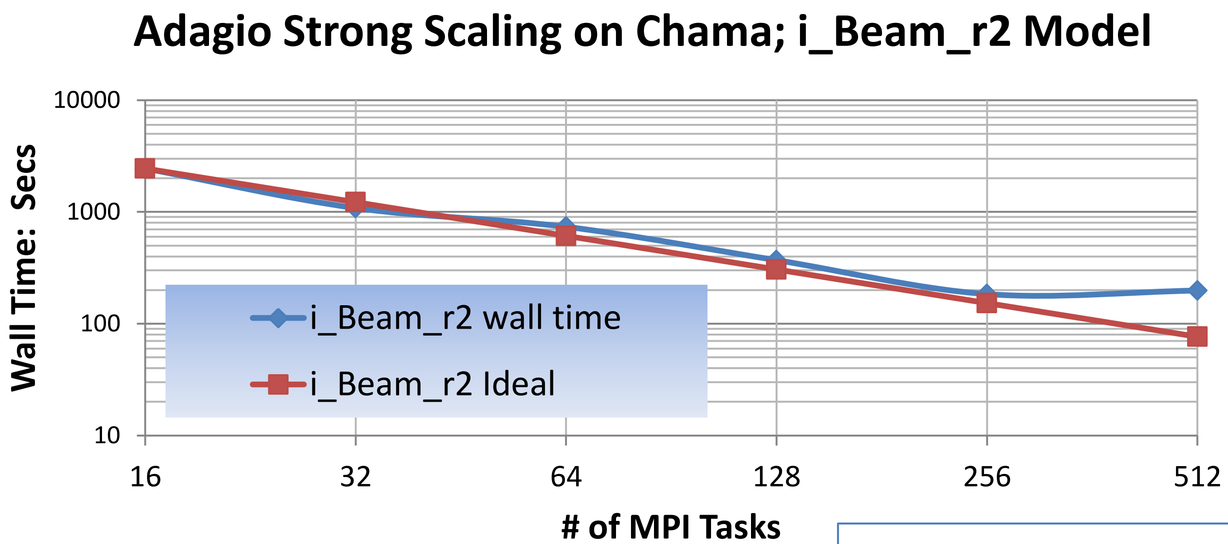
- Crystal Plasticity material model
- Problem does not converge when mesh is refined

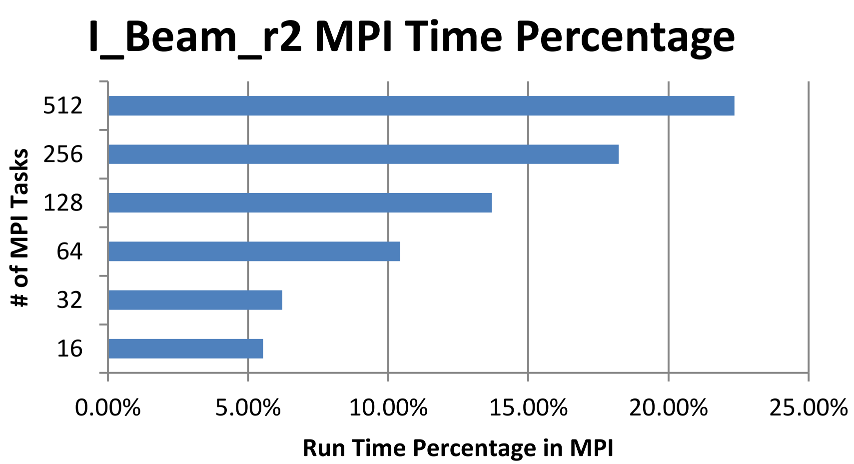
Preconditioning with linear solver

- The preconditioning step dominates the cost (>90%).
- Occurs one per time step
- Accomplished with a Jacobian matrix which requires an iterative linear solver algorithm to provide M^{-1}
- Iterative linear solve done with the FETI (Finite Element Tearing & Interconnecting) domain decomposition algorithm
- FETI requires a local solve, coarse solve, and a preconditioner solve (similar to most domain decomposition algorithms)
- Extensively uses sparse direct solvers

QS Model Strong Scaling on Chama and MPI overhead with scale

(nodes= 619,581, elements=548,864)





@--- Aggregate Time (top twenty, descending, milliseconds) ----------------

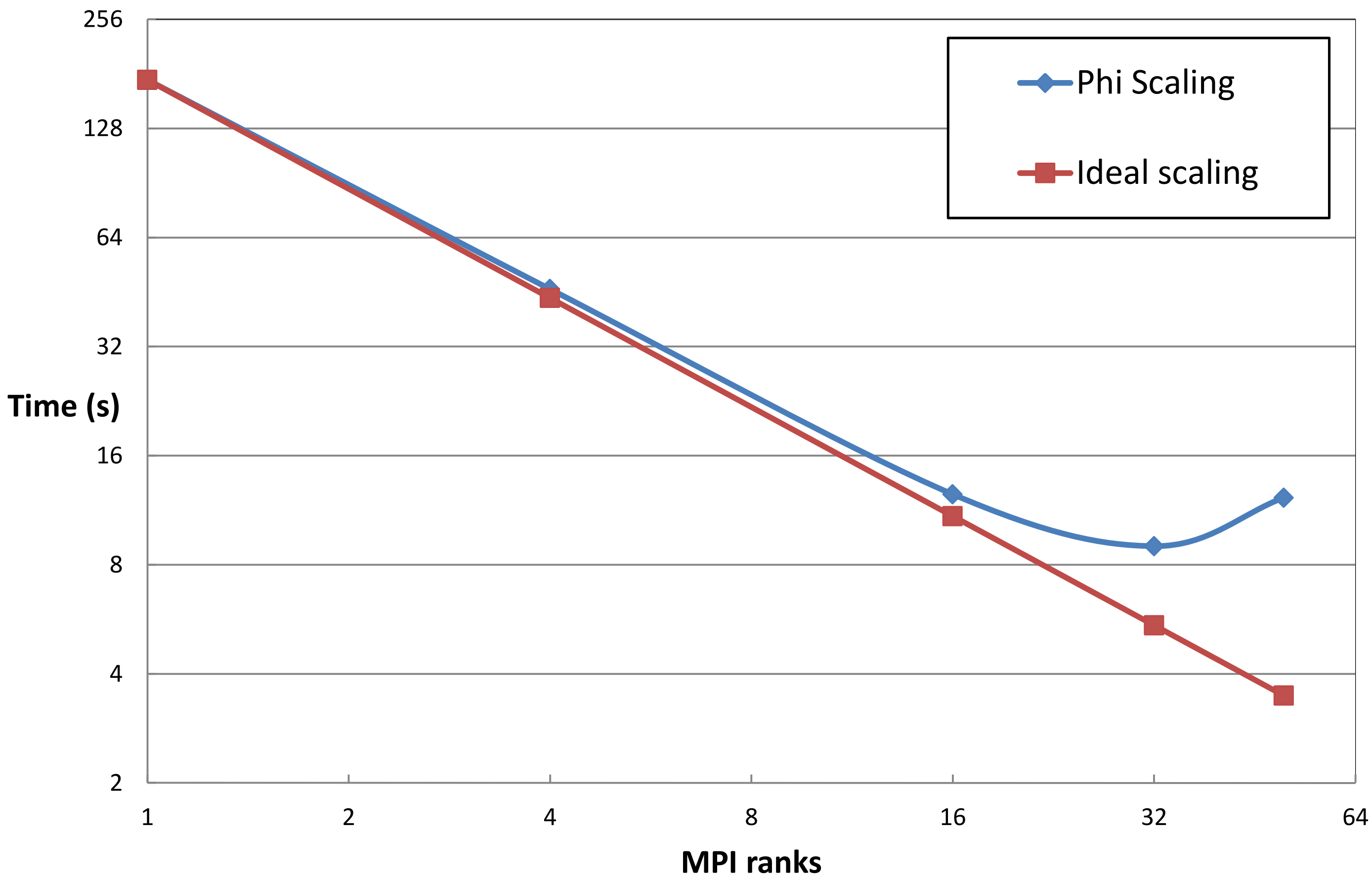
----

Call	Site	Time	App%	MPI%	COV
Allreduce	133	4.05e+06	3.90	17.46	0.09
Allreduce	168	4.01e+06	3.87	17.31	0.08
Barrier	53	3.4e+06	3.28	14.65	0.22
Allreduce	189	1.8e+06	1.74	7.78	0.00
Bcast	10	1.27e+06	1.22	5.48	0.01
Allreduce	167	1.19e+06	1.15	5.13	0.05
Bcast	98	5.68e+05	0.55	2.45	0.04

mpiP Top 5 MPI functions and call sites; 512 MPI tasks

Early KNC results

- Adagio compiles and runs on our test-bed KNC
- Scaling has proven difficult (with MPI and OpenMP)

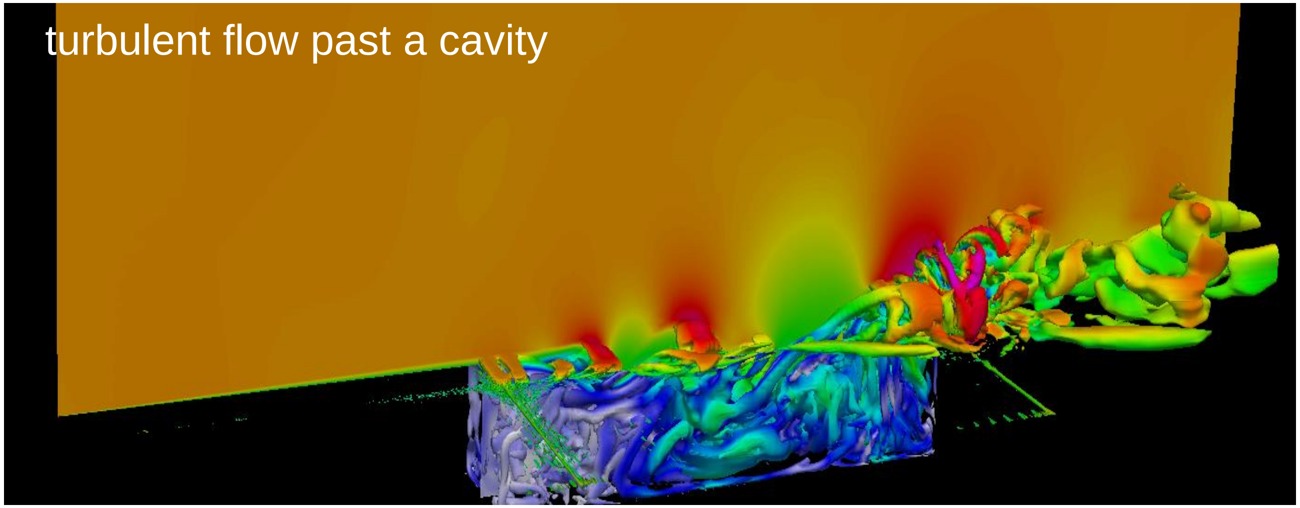


Adagio Performance Summary

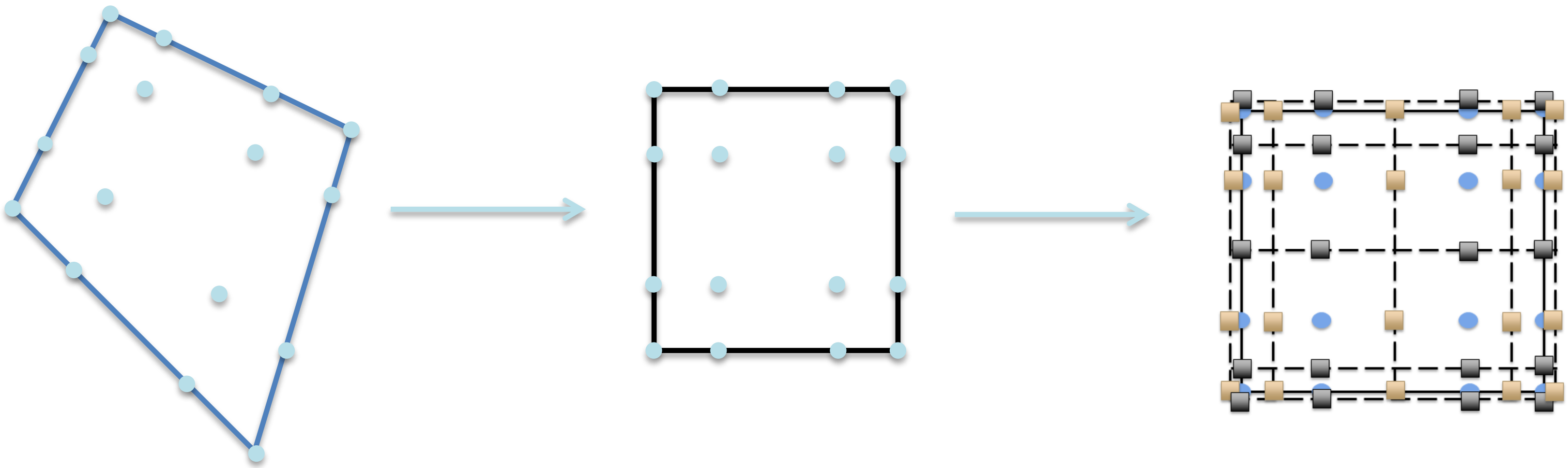
- Explicit dynamics dominated by MPI globals at scale
  – Try asynchronous collectives?
  – May benefit from optimization for small messages
- Quasi-statics
  – Need to investigate improvements after use of threading and vectorization with Pardiso / MKL
  – Leverage math library threading/vectorization

Summary of Sierra/Aero

- Unstructured meshes
- One and two equation turbulence models
- LES and Hybrid RANS
- Uses either FETI or Trilinos for sparse matrix operations and solvers.
- Assembly is substantial portion of the computational cost.



High-Order Unstructured Collocation



Unstructured Element | Standard Element | Flux-based Formulation

- Still under development
- Provably Entropy(Nonlinear) Stable
- Discontinouos formulation
- High computational intensity
- Accurate on unstructured topologies
- Trilinos Solvers for implicit solves

Trilinos Solver

- Uses Tpetra, Ifpack2 and Belos libraries
- For matrix assembly, preconditioning and solvers respectively.
- Symmetric Gauss-Seidel for preconditioner
- GMRES for solver

Aero Profile w/comments

|| 28.4% | 35715.2 | 545.8 | 1.5% |tftk::linsys::TpetraBaseBlockLinearSystem::sumInto

This function fills the actual linear system with values from the application code.

|| 19.9% | 25054.0 | 391.0 | 1.5% |Tpetra::Experimental::BlockCrsMatrix<double, int, long, KokkosClassic::SerialNode>::localGaussSeidel

This is the main work routine of the preconditioner (local on each process) that computes a smoothed solution for symmetric gauss-seidel. It is called twice for each linear iteration.

|| 14.5% | 18261.9 | 4939.1 | 21.5% |sierra::conchas::ElementFlux::operator()

This is the main computation of the residual and sensitivities for the linear system.

|| 13.7% | 17243.7 | 232.3 | 1.3% |Tpetra::Experimental::BlockCrsMatrix<double, int, long, KokkosClassic::SerialNode>::localApplyBlockNoTrans

This is a sparse matrix-vector multiply.

|| 2.9% | 3631.9 | 32.1 | 0.9% |tftk::linsys::TpetraBaseBlockLinearSystem::zeroSystem

This zeros the linear system.

|| 2.7% | 3427.8 | 39.2 | 1.1% |sierra::conchas::TpetraLinearSystem::scaleBlockMatrix

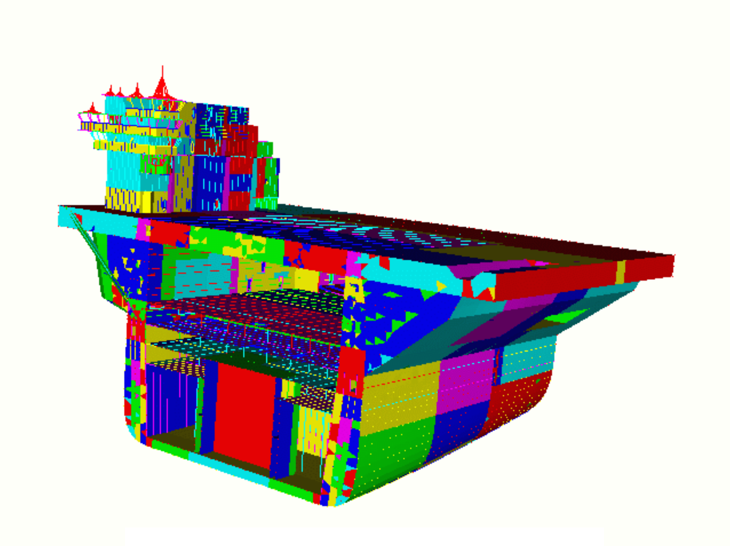
This modifies the linear system.

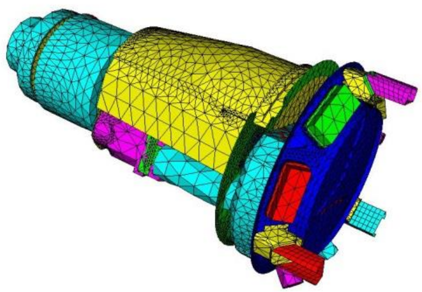
|| 1.6% | 2050.4 | 624.6 | 23.5% |sierra::conchas::FluxPenalty::operator()

This is the coupling terms for computing the residual and sensitivities for the linear system.

Domain Areas

- General Structural Dynamics, Finite Elements
  - Vibrations, normal modes, implicitly integrated transient dynamics, frequency response analysis
  - Shells, Solids, Beams, Point Masses
  - Complicated Large Structures
  - Typically many constraint equations
- Acoustics and Structural Acoustics
  - Even larger systems
  - More constraints
  - Infinite Elements (nonsymmetric)
- Optimization, UQ and Inverse Methods
  - Adjoint methods
  - Material and Parameter inversion
  - Verification and Validation





Sierra/SD Algorithms

- Domain Decomposition Linear Solvers
  - Sparse linear solver dependence
    - Threaded sparse solvers could play important future role
  - Alternative algorithms for new architectures
    - Flexibility in choice of subdomains, over-decomposition, …
- Eigen Solvers
  - Arpack current workhorse
    - Sparsekit sparse matrix utility package dependence
  - Trilinos/Anasazi
    - Could move in this direction going forward
  - Linear solver dependence
- Orthogonalization
  - Important to both linear and eigen solvers

Linear Solver Role

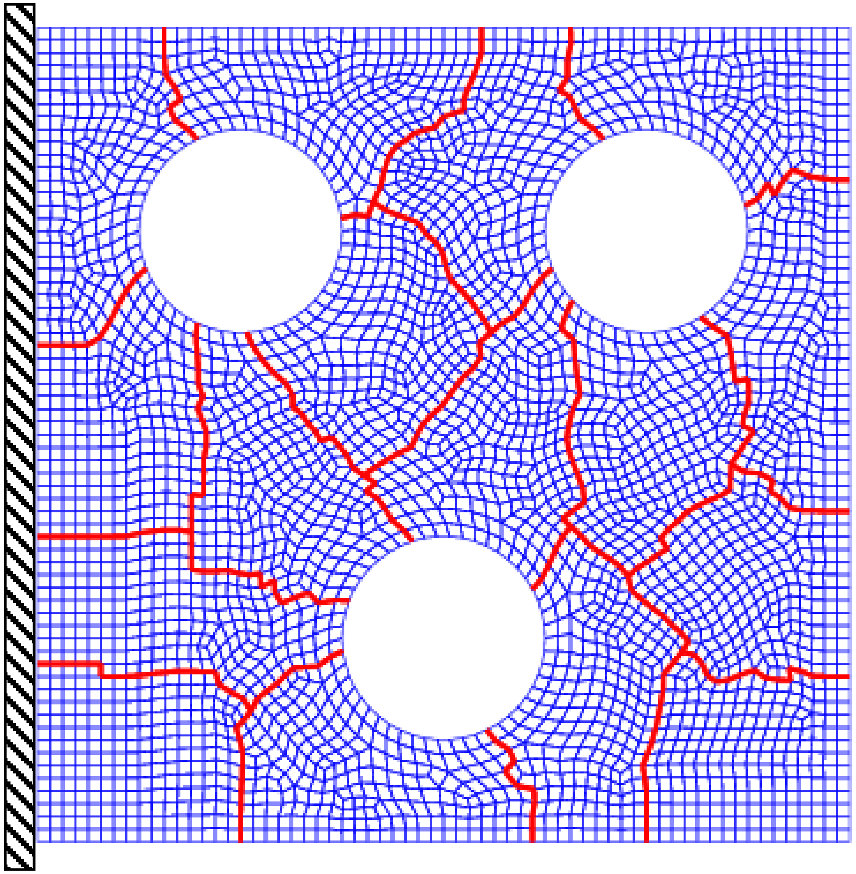
Selected Sierra-SD performance test results (chama)

Name	Analysis Type	Solve time/ Total time	Solve phase/ Solve time
mc2912	modal	0.96	0.90
nfn9	modal	0.98	0.97
endevco	transient	0.85	0.98
largerv	static	0.71	0.52

- Transient analysis (one solve for each time step)
- Modal analysis (multiple solves for each eigenmode)
- Each “solve” may take 10s to 100s of iterations

A lot of time in solve phase (initialization time often much smaller), final two columns can be even closer to 1 in practice

Domain Decomposition 101



- Partition into smaller subdomains
- Solve local (subdomain) problems
- Solve global (coarse) problem
- Combine local & global solutions
- Multilevel extensions
- Inexact solves
- Rich theory

B. Smith, P. Bjorstad, and W. Gropp, *Domain Decomposition: Parallel Multilevel Methods for Elliptic Partial Differential Equations*, Cambridge University Press, 1996.

A. Toselli and O. Widlund, *Domain Decomposition Methods: Algorithms and Theory*, Springer, 2005.

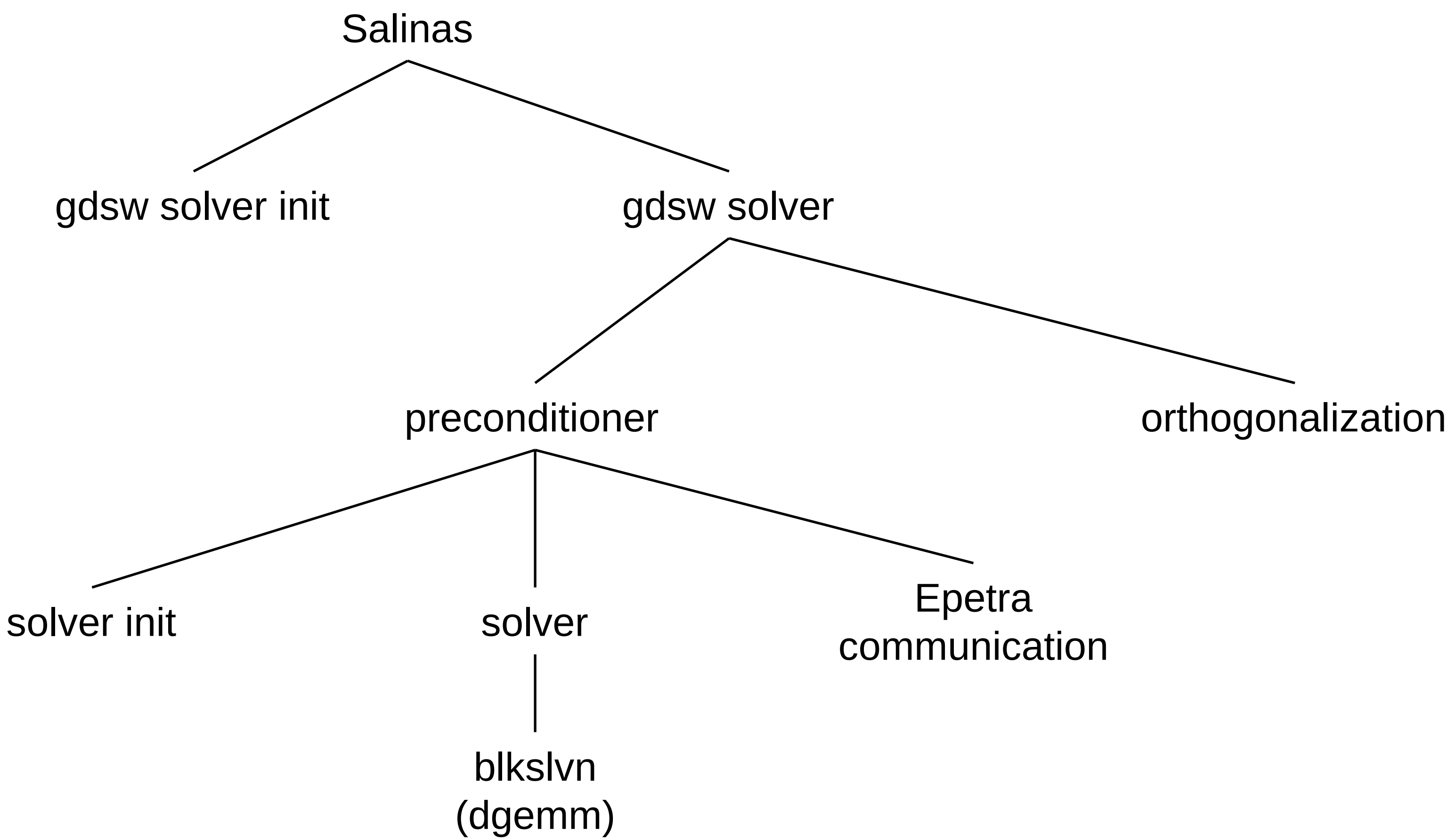
Sierra/SD TPLs

- Sparse Direct Solvers
  - SPRSBLKLLT (supernodal, left-looking, Ng & Peyton)
  - SuperLU (for complex frequency domain analysis)
  - Pardiso (option for Intel platforms, future importance?)
  - NoPivot (in-house code, left-looking, threads)
    - Movement to Trilinos/Amesos2
- Parallel Linear Algebra
  - Trilinos/Epetra movement to Trilinos/Tpetra for solver
- Dense Linear Algebra
  - BLAS, LAPACK, MKL, ScaLapack
- Graph Partitioning
  - (Par)Metis, Chaco, Zoltan/phg

Target Problems for CoE Focus

- NFN9 subsystem model
  - Currently runs on 120 processors
  - Refine mesh for scaling studies
  - OUO model
- Sparse Linear Solvers
  - Focus mainly on solve phase
  - Will provide representative linear systems
  - Evaluate performance of threaded and/or GPU accelerated solvers
- Goals:
  - Profile performance for improved speed, especially in solve phase
  - Identify problem areas
  - Suggestions for improvement
  - Reduce per-core memory footprint

Simplified Code Structure



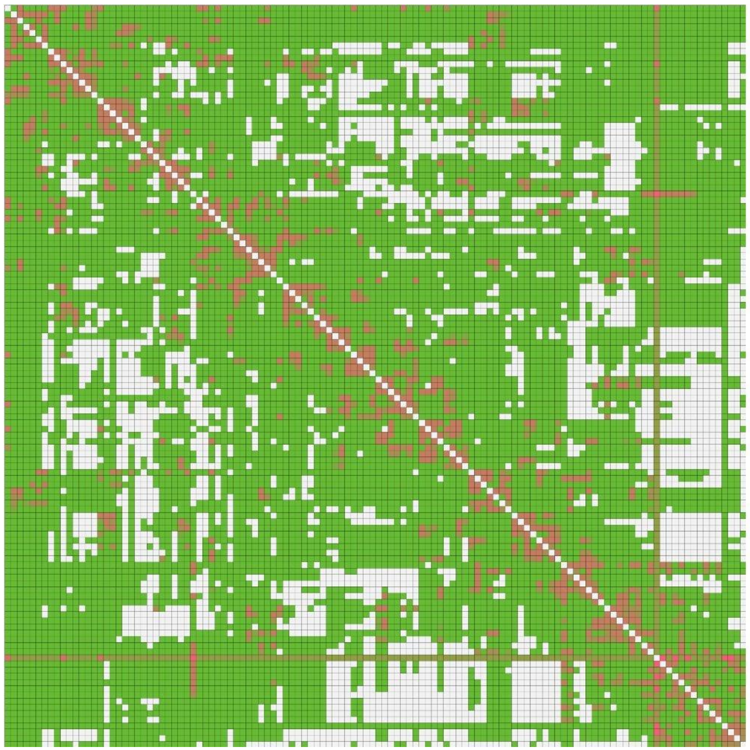
Overview

- Total time 1029.5 sec
  - User 538.5 sec (52.3%)
    - blkslvn 450.8 sec (43.7%)
  - MPI 9.6 sec (0.9%)
  - MPI_SYNC 481.4 sec (46.8%)
    - MPI_Barrier 352.3 sec (34.2%)
    - MPI_Allreduce 123.0 sec (11.9%)
- Total FLOPS 343.0e9 - double precision
  - 331.5 MFLOPs/rank (3.5% peak)

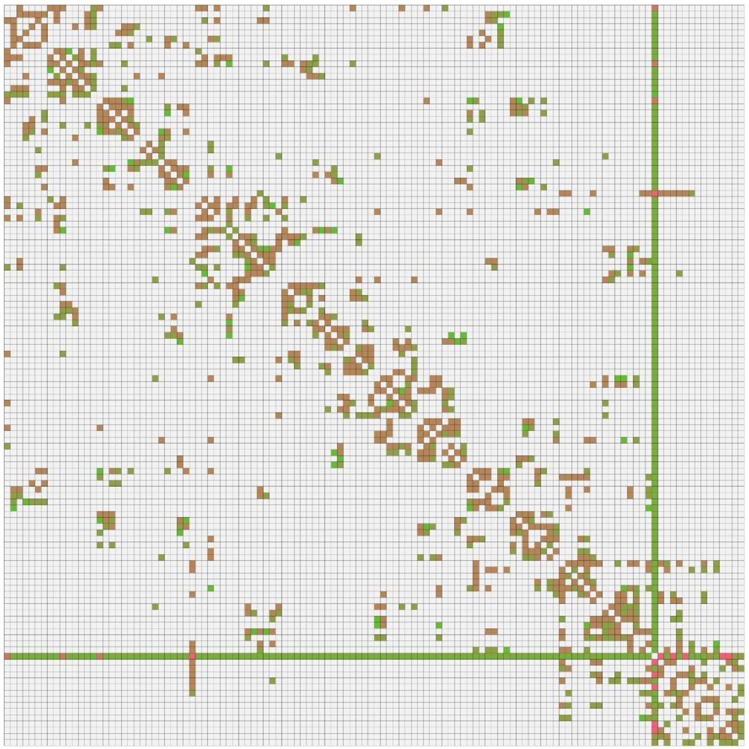
Preconditioner Solve

- On node backsolve
  - Shows 0 time when instrumented
  - called in .h file
- Calls blkslvn (FORTRAN)
- blkslvn called average of 6182 times
- calls dgemm
  - CrayPat loses connection to dgemm(shows up in call tree attached to root)
- Time for direct solve not in calling routines
- blkslvn takes 450.8 sec (83.7% of user time)

Communication Matrices



Whole Code



GDSW Solver

Summary

- Shown three applications from SIERRA Framework with performance profiling
- Significant time spent in two areas:
  - Solvers
  - Matrix Assembly
- Haswell performance should follow current processors
  - How to utilize the extra features of Haswell?
- Some experience with Knights Corner
  - How to translate to Knights Landing