SAND2011-6826C

Sierra Structural Dynamics Multi-threaded evaluations

The SierraSD application is a large FE element application running on an MPI communications layer. Domain decomposition solvers and matrix assembly form the primary computational kernels. We evaluate means of incorporating threads into the algorithms both for matrix assembly and for linear solver performance.

Strategies:

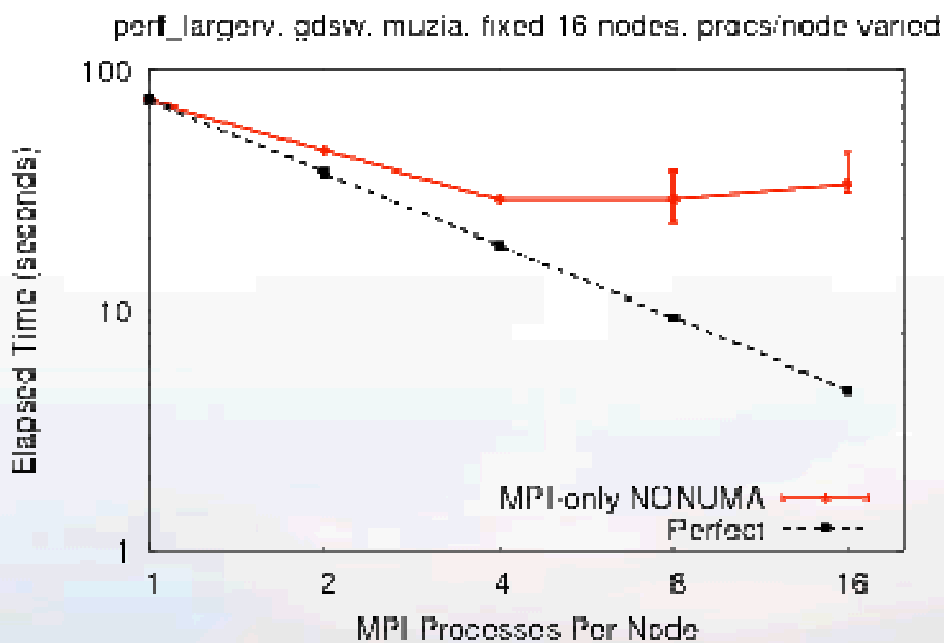
Expectations

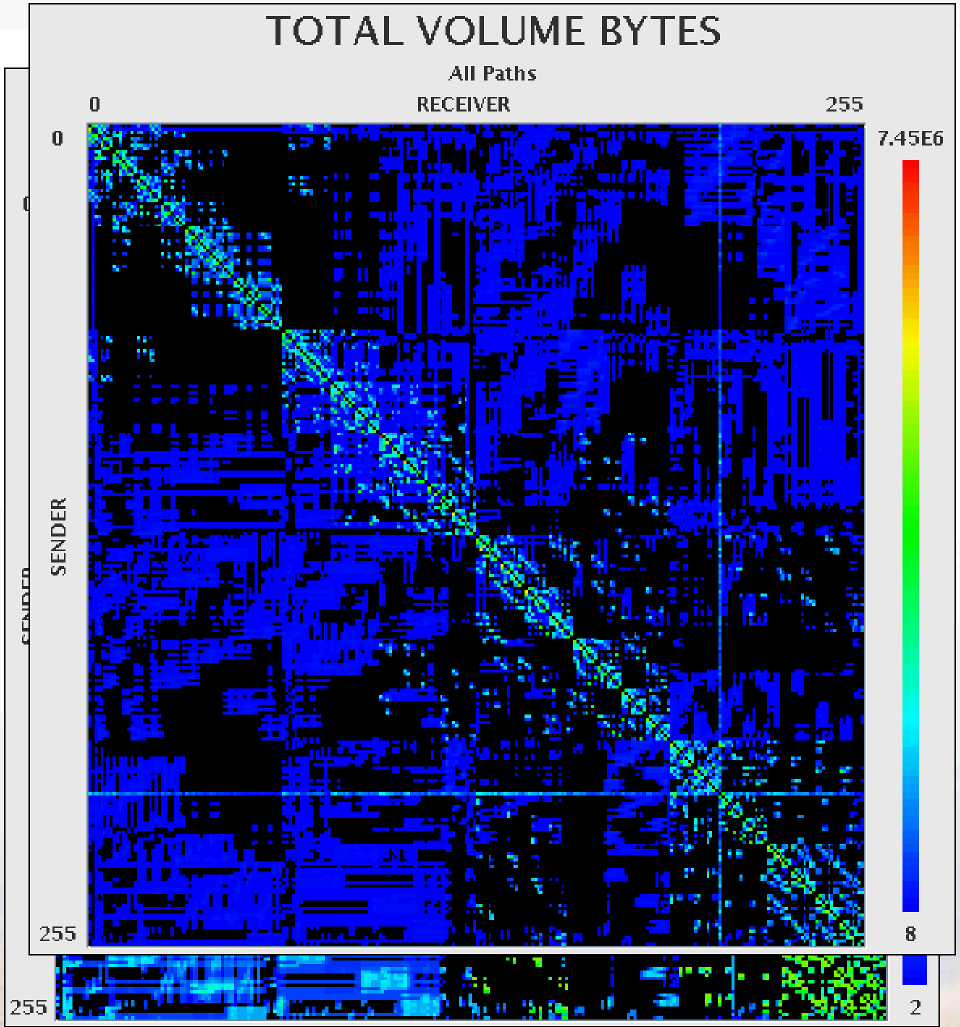
- Future platforms will have more cores available.
- The memory per core will shrink, but memory per node will likely not change much.

Strategy

- Constraints on solvers and memory will dictate the number of subdomains. We continue to tie a subdomain to an MPI process.
- Additional cores are available "for free". These cores may be used in threaded applications. We examine
  1. Threaded assembly of element matrices to subdomain matrices.
  2. Threaded direct solvers (used in the domain decomposition linear solvers).

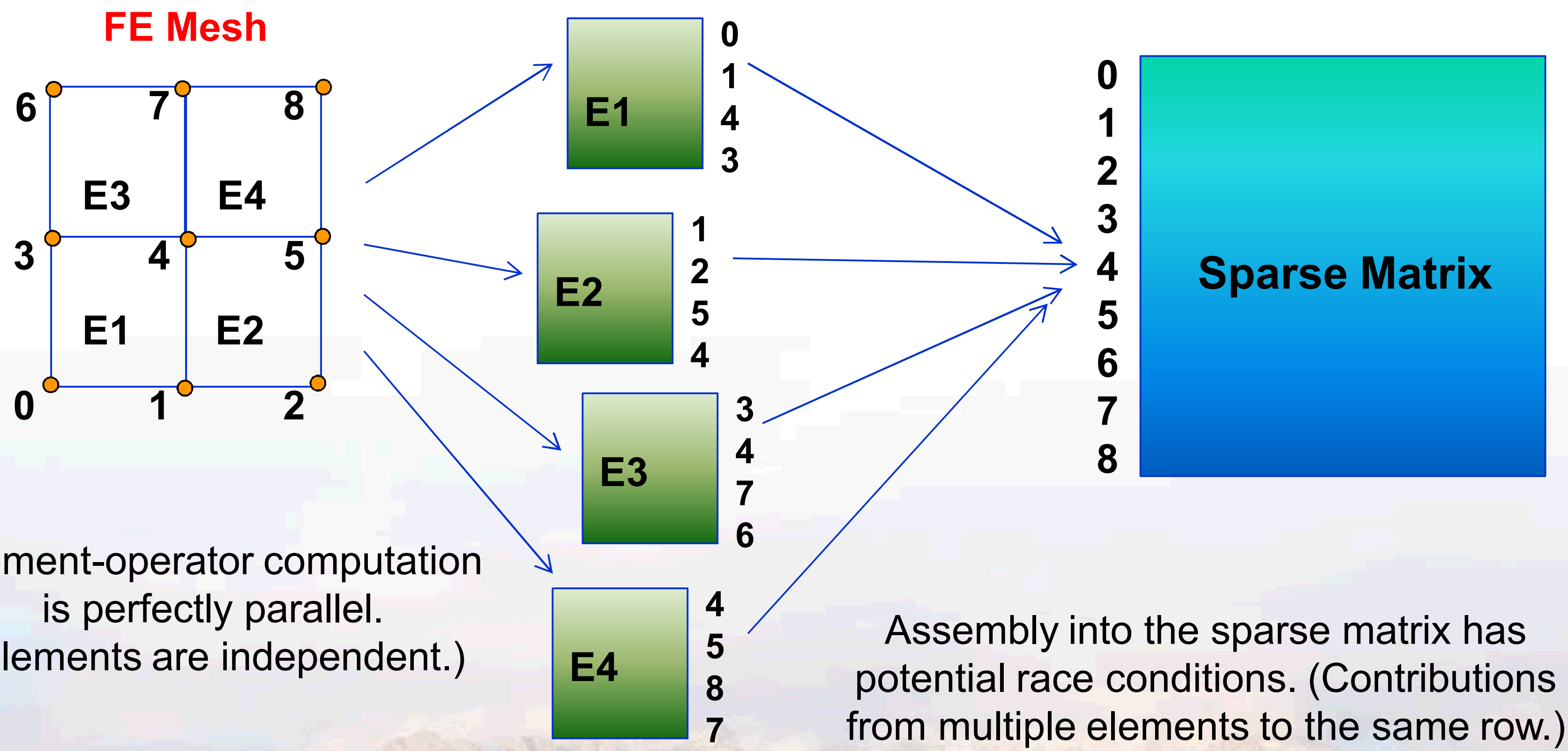
Initial Characterization of Application





SierraSD threading project: parallelize stiffness matrix assembly

Problem: Compute dense element-stiffness operator for each element, assemble into sparse matrix.



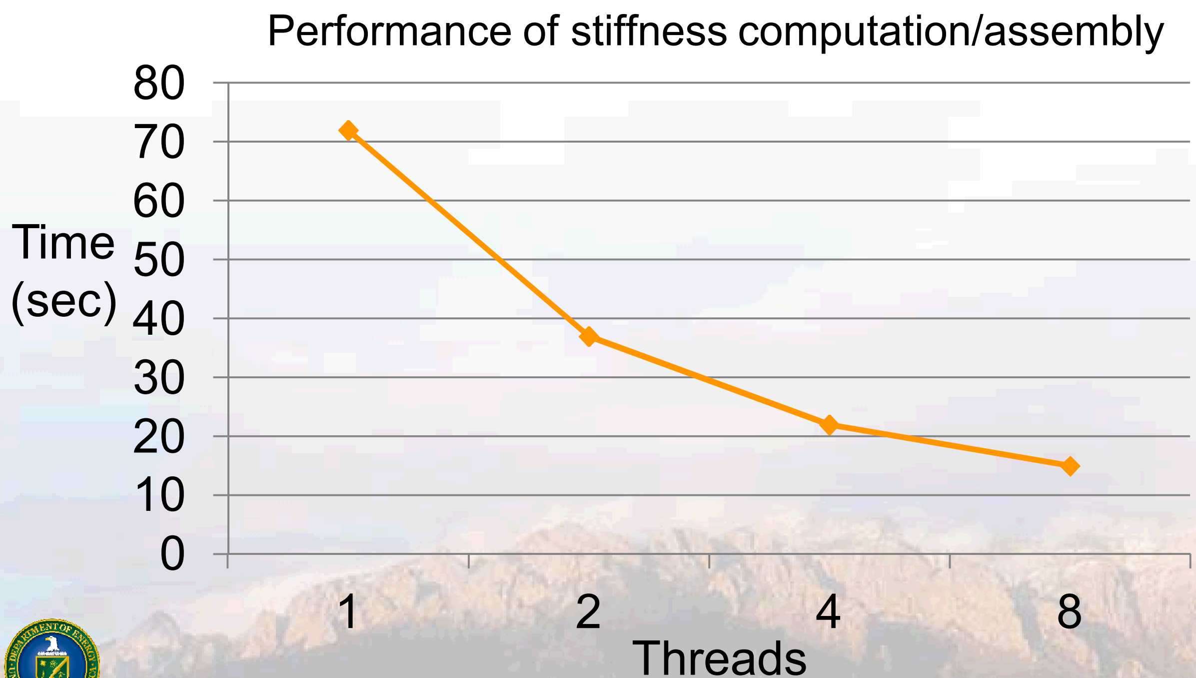
Element-operator computation is perfectly parallel. (Elements are independent.)

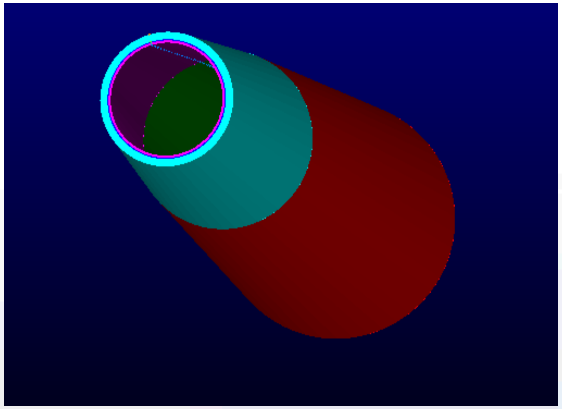
Assembly into the sparse matrix has potential race conditions. (Contributions from multiple elements to the same row.)

SierraSD threading project: stiffness matrix assembly

Approach:

1. Make SierraSD’s element-loop thread-parallel using Intel Thread Building Blocks (TBB) **tbb::parallel_for** mechanism
2. Add locking mechanism for sparse matrix rows (using **tbb::atomic** mechanism) to protect against race conditions in assembly.





Model: ‘perf_layered_frusta’
6 element-blocks: 20,160 elems

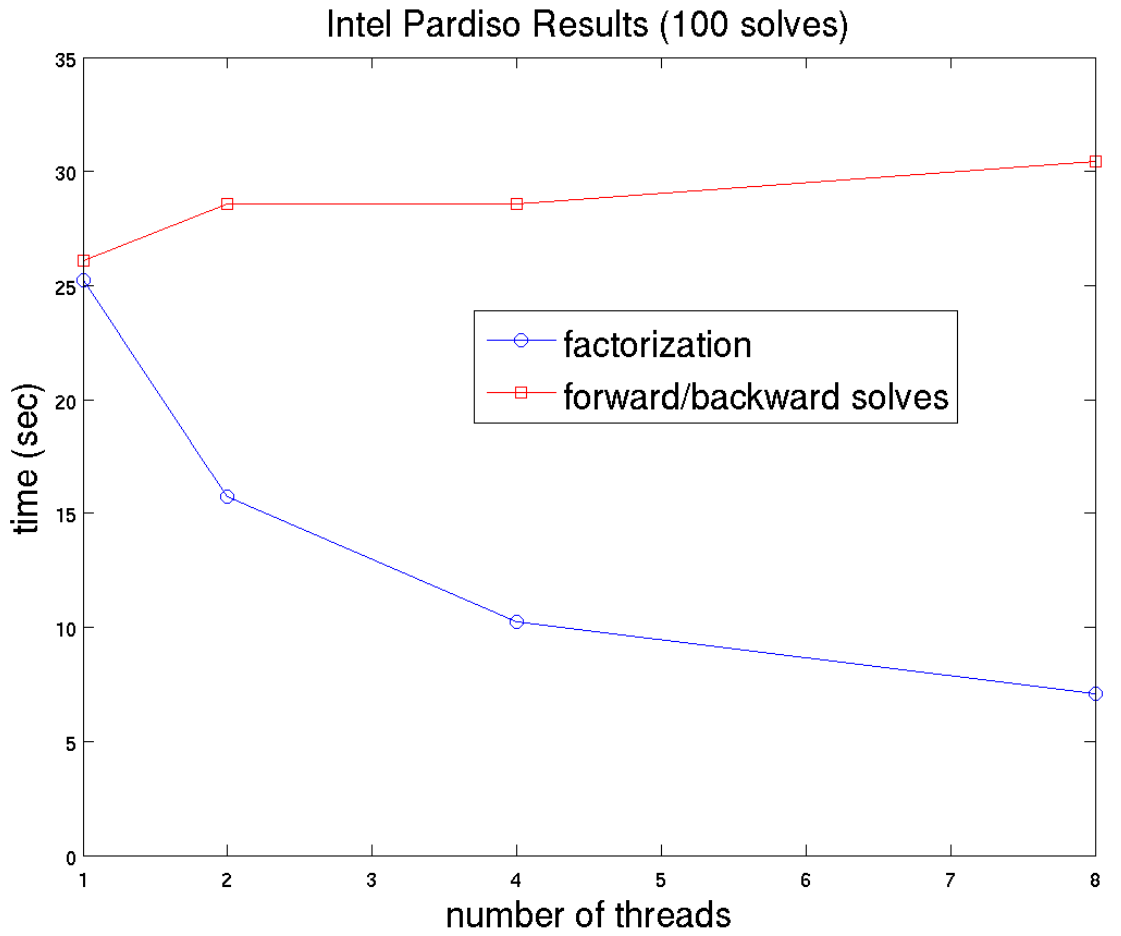
SierraSD threading project: direct solvers

Background:

1. Direct solvers are at the heart of domain decomposition solvers such as FETI-DP and GDSW (both local and global systems of linear equations need to be solved)
2. Threading of the direct solvers has the potential to significantly reduce analysis times
3. For most SierraSD applications, effective threading of the forward/backward solution phase is more important than the factorization phase
    - Transient analysis (one solve for each time step)
    - Modal analysis (multiple solves for each eigenmode)

SierraSD threading project: direct solvers

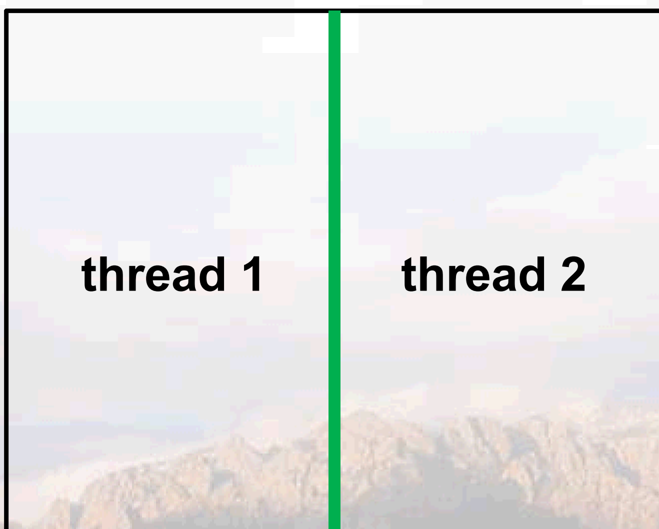
32x32x32 HEX8 cube model: illustration of basic problem

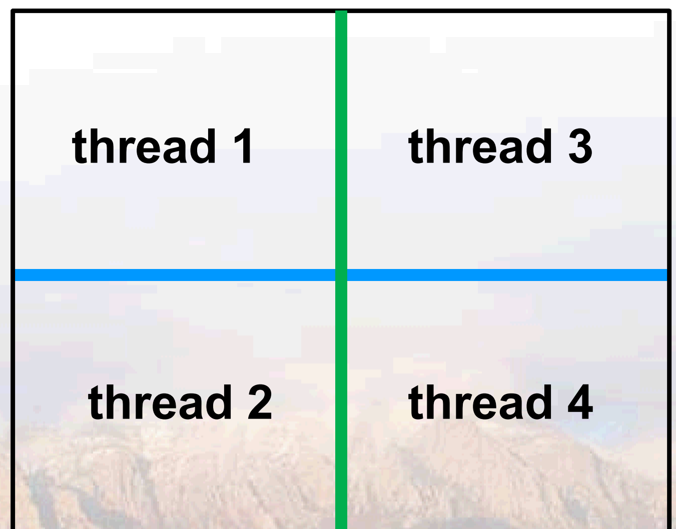


SierraSD threading project: direct solvers

Approach:

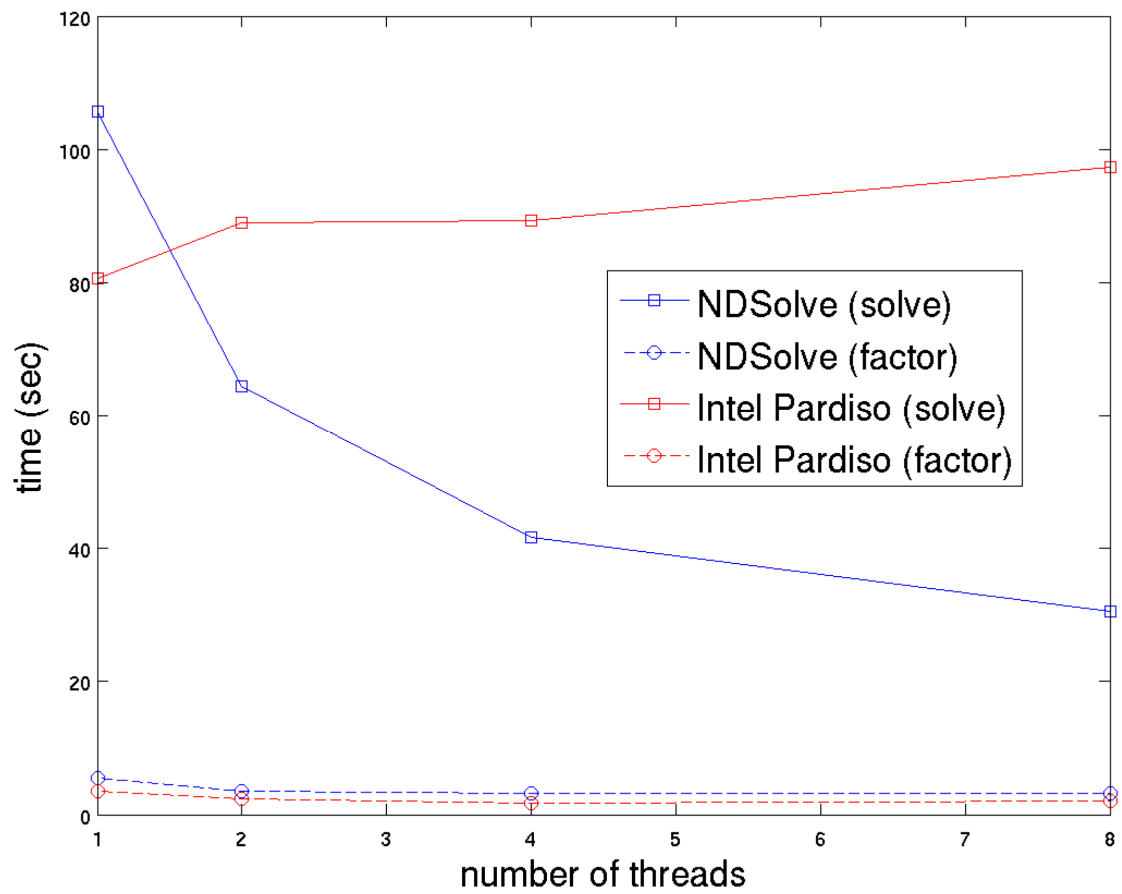
1. Develop thread-parallel version of existing SierraSD direct solver using Intel Threading Building Blocks (TBB) tbb::parallel_for functionality (not part of original plan)
2. Recursively partition problem into smaller parts using Nested Dissection ordering and apply thread to each part





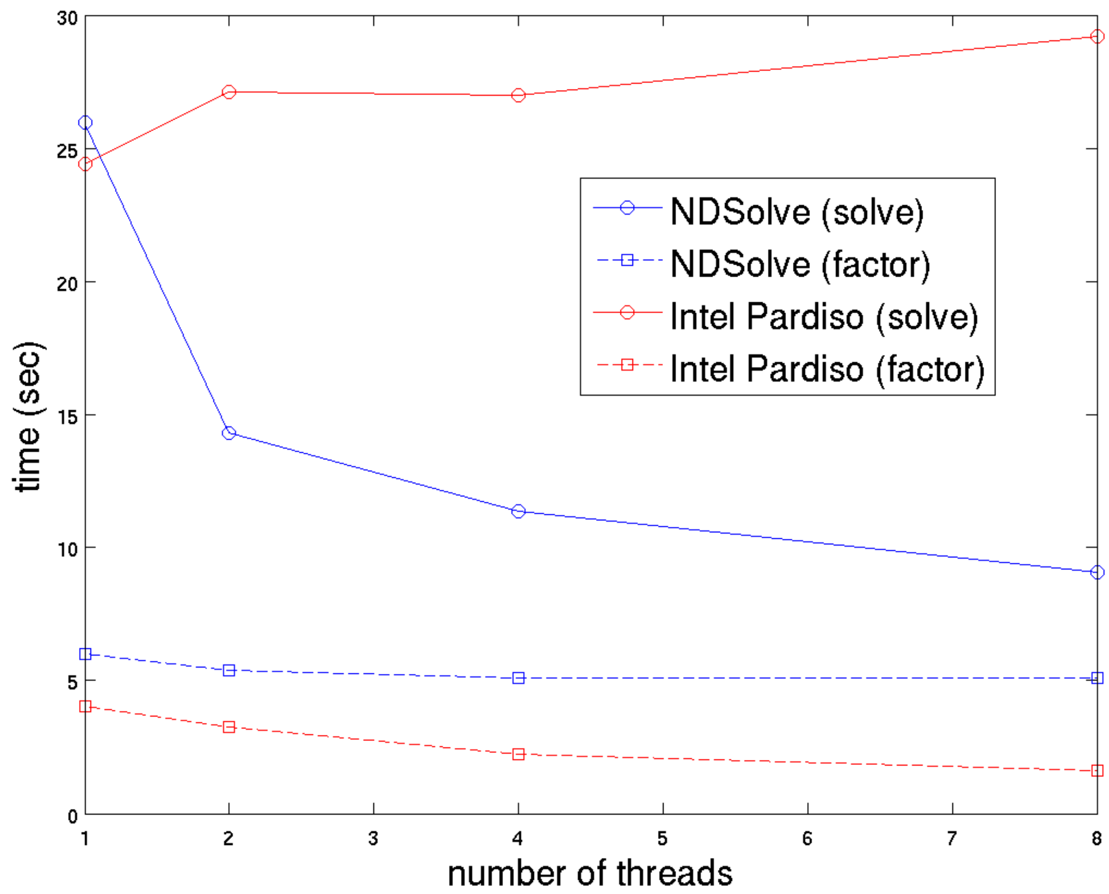
SierraSD threading project: direct solvers

Ship model (2x), 1281 solves (400 modes), 21753 elements



SierraSD threading project: direct solvers

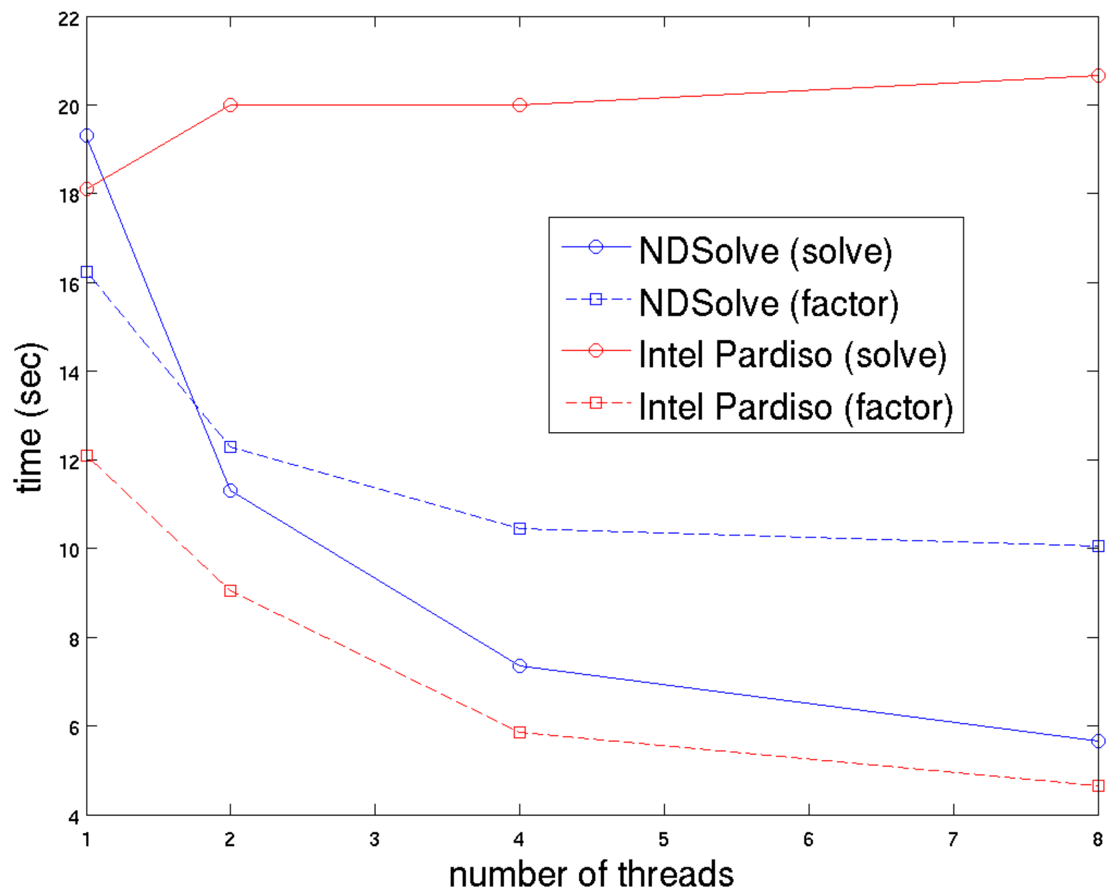
Tire model, 301 solves (100 modes), 23940 elements



Sandia National Laboratories

SierraSD threading project: direct solvers

16x16x128 HEX8 beam model, 100 solves, 32768 elements



Summary

- Element assembly routines can be successfully threaded over the elements. We may need to rework subdomain matrices.
- Commercial multithreaded solvers, such as Pardiso/intel, may offer some benefit, but may not address our most important issues. For example, factor is improved but solve is not.
- Commercial BLAS routines multithreading may conflict with in house threaded applications.
- The effectiveness of threaded BLAS routines may vary significantly from routine to routine.
- Threaded parallel sparse direct solvers can show improvement in both factor and solve phase. Such threaded solvers could show significant impact across a broad range of applications.
- Evaluations were with Intel compilers, libraries and packages.
- Most success in using the TBB for threads.