Preparation of Codes for Trinity

Sandia National Laboratories

CUG Conference
April 26-30, 2015
• 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
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:

<table>
<thead>
<tr>
<th>Explicit dynamics with contact</th>
<th>Implicit with FETI pre-conditioner</th>
<th>Explicit dynamics w/o contact</th>
</tr>
</thead>
</table>

## 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:

Potential contact detected
I-Beam Problem (Quasi-Static)

-provided by Joe Bishop

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

Unique Features:
- 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)

Adagio Strong Scaling on Chama; i_Beam_r2 Model

I_Beam_r2 MPI Time Percentage

<table>
<thead>
<tr>
<th># of MPI Tasks</th>
<th>Run Time Percentage in MPI</th>
</tr>
</thead>
<tbody>
<tr>
<td>512</td>
<td>0.00%</td>
</tr>
<tr>
<td>256</td>
<td>5.00%</td>
</tr>
<tr>
<td>128</td>
<td>10.00%</td>
</tr>
<tr>
<td>64</td>
<td>15.00%</td>
</tr>
<tr>
<td>32</td>
<td>20.00%</td>
</tr>
<tr>
<td>16</td>
<td>25.00%</td>
</tr>
</tbody>
</table>

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

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

- Still under development
- Provably Entropy (Nonlinear) Stable
- Discontinuous 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
<table>
<thead>
<tr>
<th>28.4%</th>
<th>35715.2</th>
<th>545.8</th>
<th>1.5%</th>
<th>tfstk::linsys::TpetraBaseBlockLinearSystem::sumInto</th>
</tr>
</thead>
</table>

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

<table>
<thead>
<tr>
<th>19.9%</th>
<th>25054.0</th>
<th>391.0</th>
<th>1.5%</th>
<th>Tpetra::Experimental::BlockCrsMatrix&lt;double, int, long, KokkosClassic::SerialNode&gt;::localGaussSeidel</th>
</tr>
</thead>
</table>

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.

<table>
<thead>
<tr>
<th>14.5%</th>
<th>18261.9</th>
<th>4939.1</th>
<th>21.5%</th>
<th>sierra::conchas::ElementFlux::operator()</th>
</tr>
</thead>
</table>

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

<table>
<thead>
<tr>
<th>13.7%</th>
<th>17243.7</th>
<th>232.3</th>
<th>1.3%</th>
<th>Tpetra::Experimental::BlockCrsMatrix&lt;double, int, long, KokkosClassic::SerialNode&gt;::localApplyBlockNoTrans</th>
</tr>
</thead>
</table>

This is a sparse matrix-vector multiply.

<table>
<thead>
<tr>
<th>2.9%</th>
<th>3631.9</th>
<th>32.1</th>
<th>0.9%</th>
<th>tfstk::linsys::TpetraBaseBlockLinearSystem::zeroSystem</th>
</tr>
</thead>
</table>

This zeros the linear system.

<table>
<thead>
<tr>
<th>2.7%</th>
<th>3427.8</th>
<th>39.2</th>
<th>1.1%</th>
<th>sierra::conchas::TpetraLinearSystem::scaleBlockMatrix</th>
</tr>
</thead>
</table>

This modifies the linear system.

<table>
<thead>
<tr>
<th>1.6%</th>
<th>2050.4</th>
<th>624.6</th>
<th>23.5%</th>
<th>sierra::conchas::FluxPenalty::operator()</th>
</tr>
</thead>
</table>

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)

<table>
<thead>
<tr>
<th>Name</th>
<th>Analysis Type</th>
<th>Solve time/Total time</th>
<th>Solve phase/Solve time</th>
</tr>
</thead>
<tbody>
<tr>
<td>mc2912</td>
<td>modal</td>
<td>0.96</td>
<td>0.90</td>
</tr>
<tr>
<td>nfn9</td>
<td>modal</td>
<td>0.98</td>
<td>0.97</td>
</tr>
<tr>
<td>endevco</td>
<td>transient</td>
<td>0.85</td>
<td>0.98</td>
</tr>
<tr>
<td>largerv</td>
<td>static</td>
<td>0.71</td>
<td>0.52</td>
</tr>
</tbody>
</table>

- 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


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

- Salinas
  - gdsw solver init
  - gdsw solver
    - preconditioner
      - solver init
      - solver
        - blkslvn (dgemm)
    - orthogonalization
      - Epetra communication
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