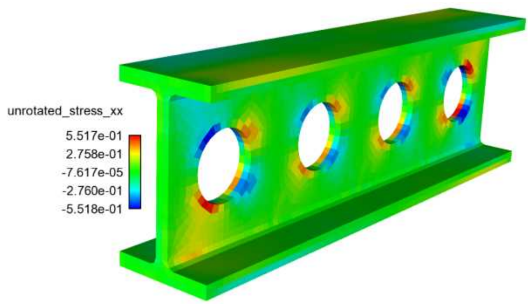
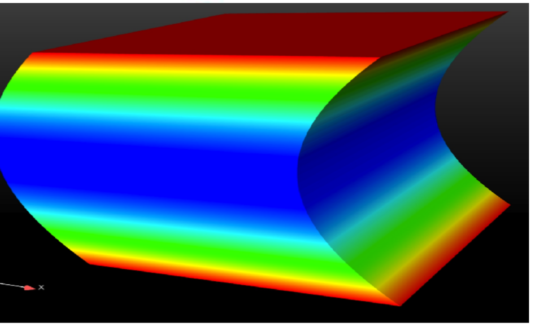
SAND2015-0099PE





SIERRA Solid Mechanics Trinity CoE Meeting December 9, 10 2014

SIERRA/SM Profiling

Mahesh Rajan, Michael Tupek, Kendall Pierson

Sandia National Laboratories

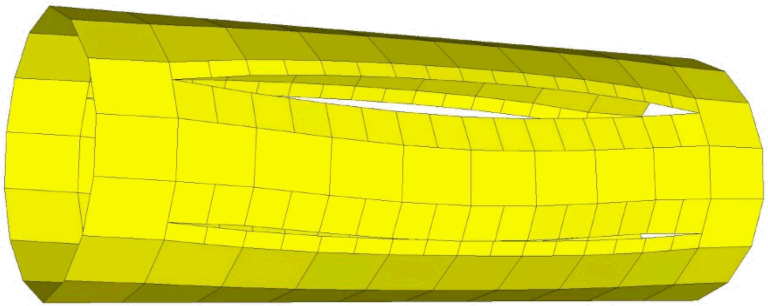
Exceptional service in the national interest

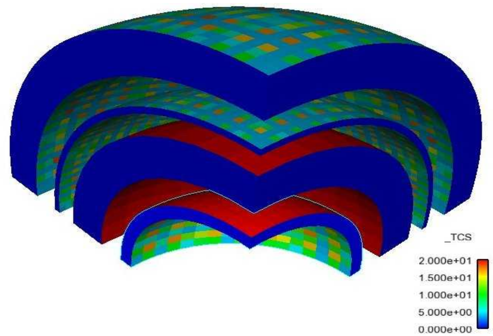
U.S. DEPARTMENT OF ENERGY

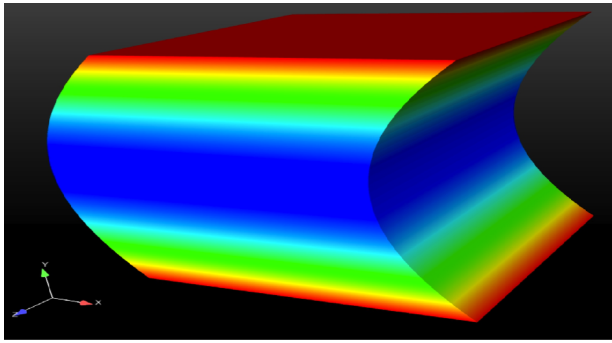
NNSA National Nuclear Security Administration

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. SAND NO. 2011-XXXXP

SIERRA/SM (Solid Mechanics)

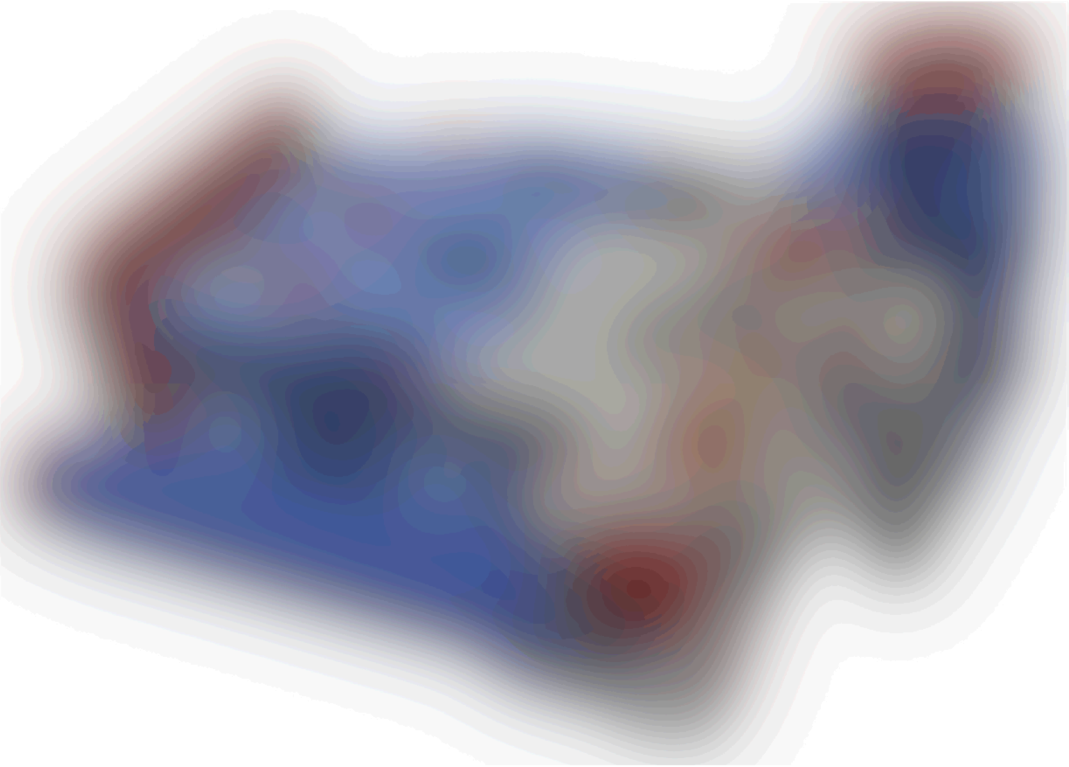






- **A general purpose massively parallel nonlinear solid mechanics finite element code for explicit transient dynamics, implicit transient dynamics and quasi-statics analysis.**
- **Built upon extensive material, element, contact and solver libraries for analyzing challenging nonlinear mechanics problems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

Explicit Dynamic Impact Problem



Runtime

Sandy bridge – 512 processors

Original: 48 hours

Current: 20h 11m 21s

Speed-Up: **2.38x**

Milli-second impact analysis

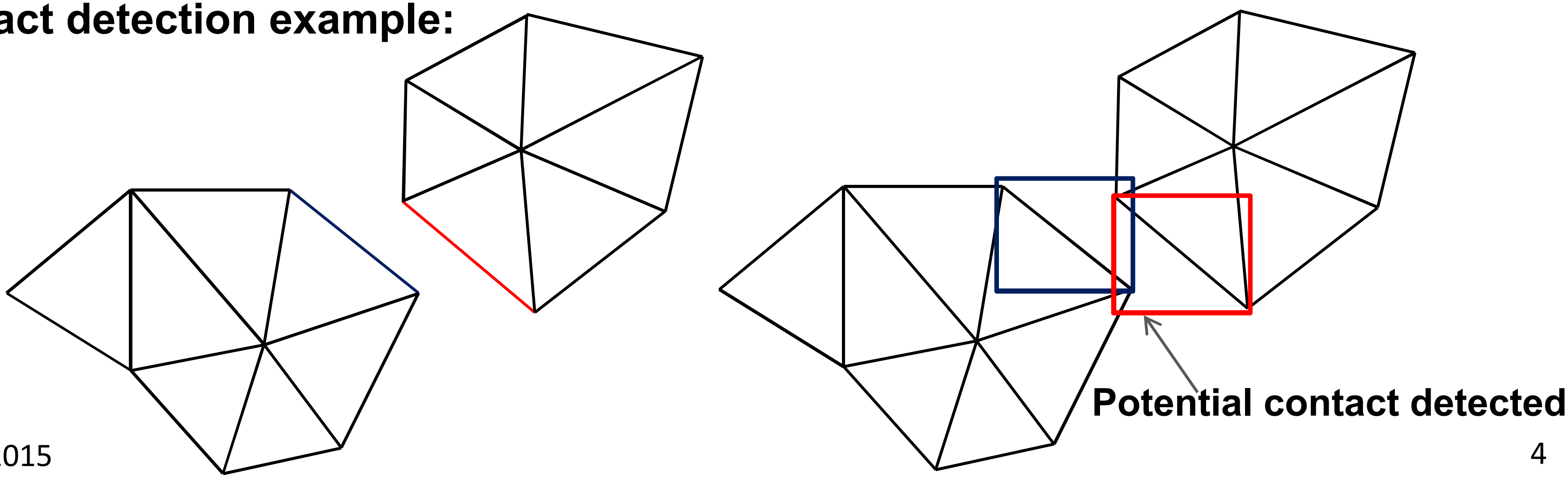
Runtime Breakdown by Capability

Capability	Percent of Run
Contact: Search and Enforcement	85.06%
Internal Force Calculation	3.75%
Energy Calculations	1.77%

SIERRA/SM Bottlenecks

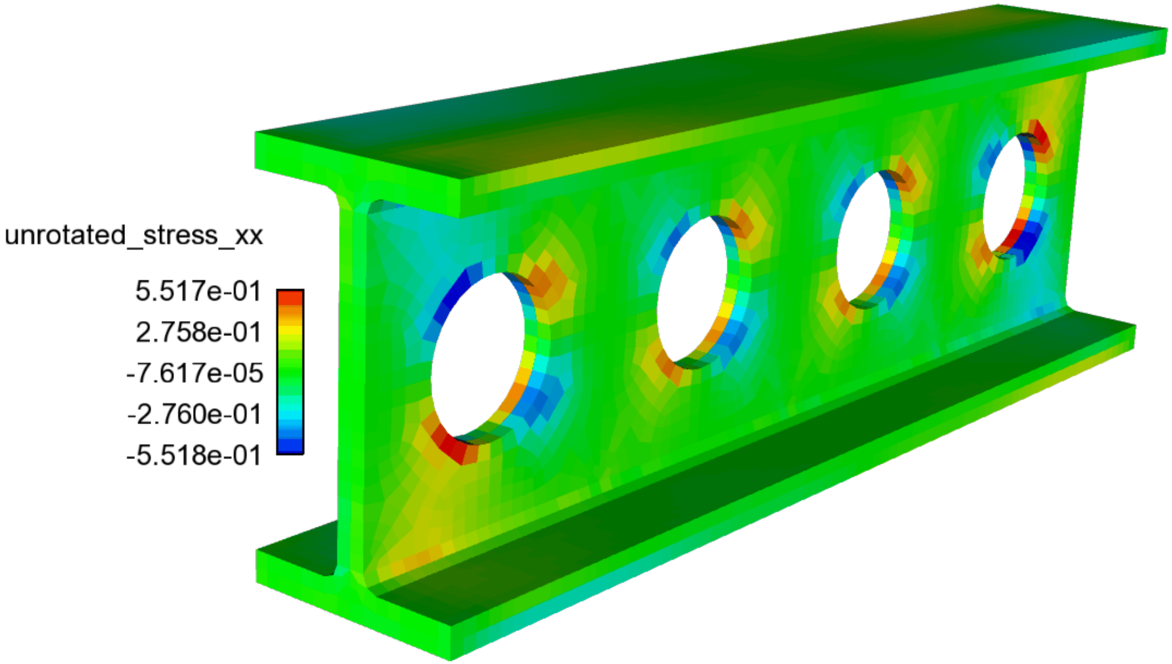
Application:	**Explicit dynamics with contact**	**Implicit with FETI pre-conditioner**	**Explicit dynamics w/o contact**
Hot spot:	Parallel **proximity search** and enforcing contact constraints	Serial **sparse direct solve**: matrix factorization and forward/backward solves	Assembling nonlinear **element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

Contact detection example:



I-Beam Problem (Quasi-Static)

-provided by Joe Bishop



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

Quasi-Static Solution Algorithm

1. Initialize Time Step, $t = 0$, $dt = dt0$
2. Compute Residual Force:

   $R(x,t) = F_{external}(x,t) - F_{internal}(x,t) + F_{contact}(x,t)$
3. **Iterate** until: $R(x,t) = 0$
4. If Converged, $t = t + dt$

Nonlinear Conjugate Gradient

1. k= 0
2. Loop, until converged

$R(x,t) = F_{external}(x,t) - F_{internal}(x,t) + F_{contact}(x,t)$

$G = M^{-1} R(x,t)$ // **preconditioning**

$S = G + beta * S^{k-1}$ // **axby**

alpha = LineSearch(S) // **extra residual call**

x = x + alpha * S // **axby**

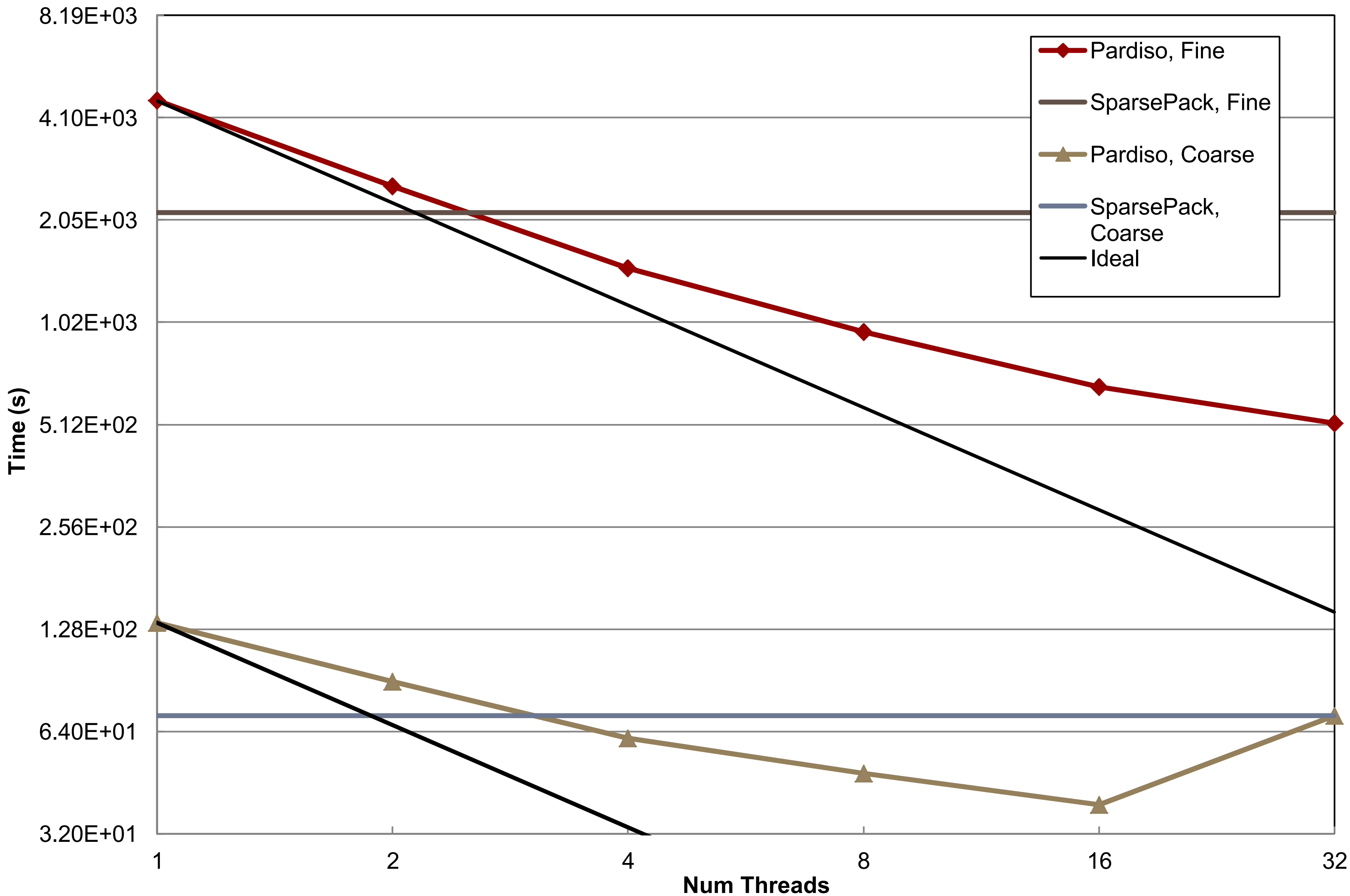
Compute ||R||, check convergence

Beta = compute_beta() // dot products, all reduce

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**

Pardiso vs. SparsePack direct solver



Sierra/SM Performance Profile

- Have performance profile on Chama (Sandy Bridge IB Cluster) with
  - vTune
  - HPCToolkit
  - Allinea Map
- Need to resolve build of Sierra/SM with CrayPat on Cray XE6 (Muzia)
- Used OpenSpeedShop(ossmpiotf) and Vampir to get MPI message characteristics
- Performance profile confirms expected hotspots:
  - Explicit dynamic: (MPI Sync time in Allreduce & Barrier)
    - Cont_DashEnforcement.c line 550 (Allreduce; 23.1%)
    - ContactCommunication.C line 2801 (Barrier; 5.9%)
  - Quasi-static:
    - 36% of run time out of which 52% of the time in MPI calls was at ~/mycode/FETI-DP/src/FETI_DP_FiniteElementData.C (line 919) feti::FetiDriver (FetiDriver.C line 228)
    - The other hot spot (5.3% of run time) is in ParallelCoarseGrid.C line 179; 88% of this time was in MPI (calls Allreduce BlockSparseSolver.C line 476)

Allinea Performance Report on Chama (Sandy Bridge, IB cluster) 512 core runs

Sandia National Laboratories

Explicit dynamic contact

Summary: Sierra/SM is MPI-bound in this configuration

CPU: 46.3%

MPI: 53.7%

CPU:

A breakdown of how the 46.3% total CPU time was spent:

Scalar numeric ops: 19.6%

Vector numeric ops: 2.4%

Memory accesses: 77.3%

MPI:

A breakdown of how the 53.7% total MPI time was spent:

Time in collective calls: 92.3%

Time in point-to-point calls: 7.7%

Effective collective rate: 1.71e+06 bytes/s

Effective point-to-point rate: 3.13e+08 bytes/s

Quasi-static (implicit)

Summary: Sierra/SM is CPU-bound in this configuration

CPU: 68.9%

MPI: 30.8%

CPU:

A breakdown of how the 68.9% total CPU time was spent:

Scalar numeric ops: 19.9%

Vector numeric ops: 10.2%

Memory accesses: 69.9%

MPI:

A breakdown of how the 30.8% total MPI time was spent:

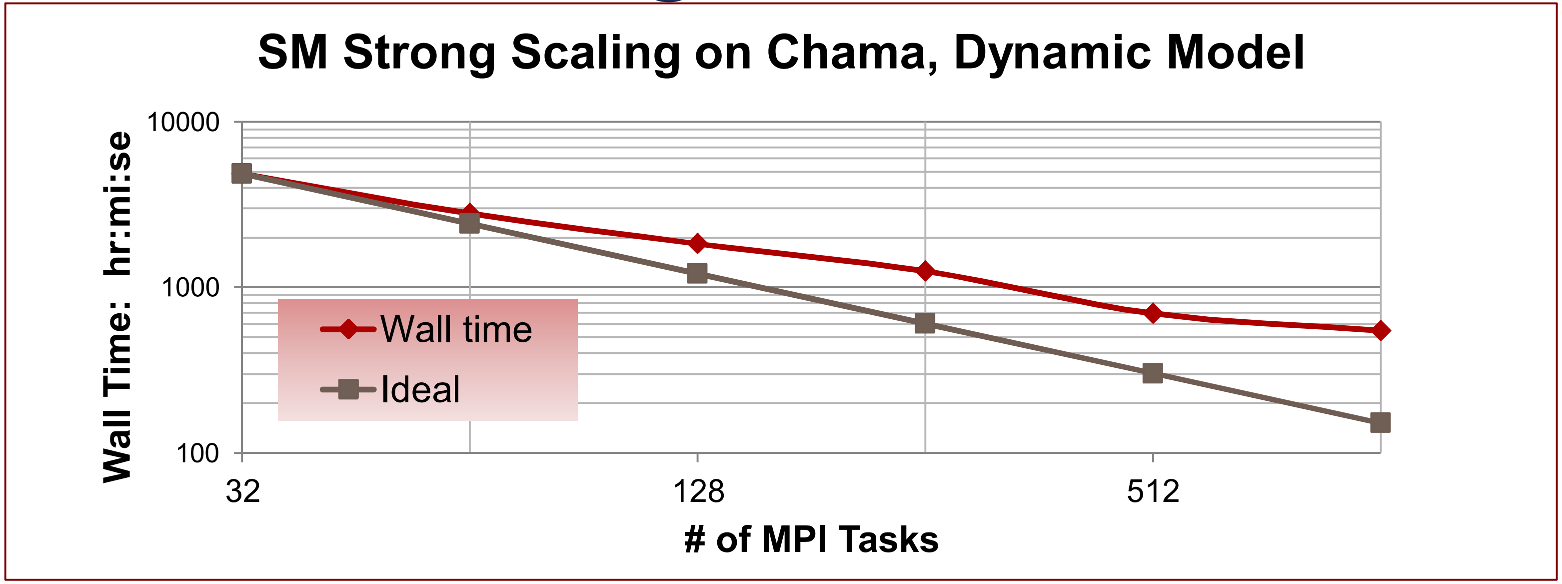
Time in collective calls: 94.7% |========|

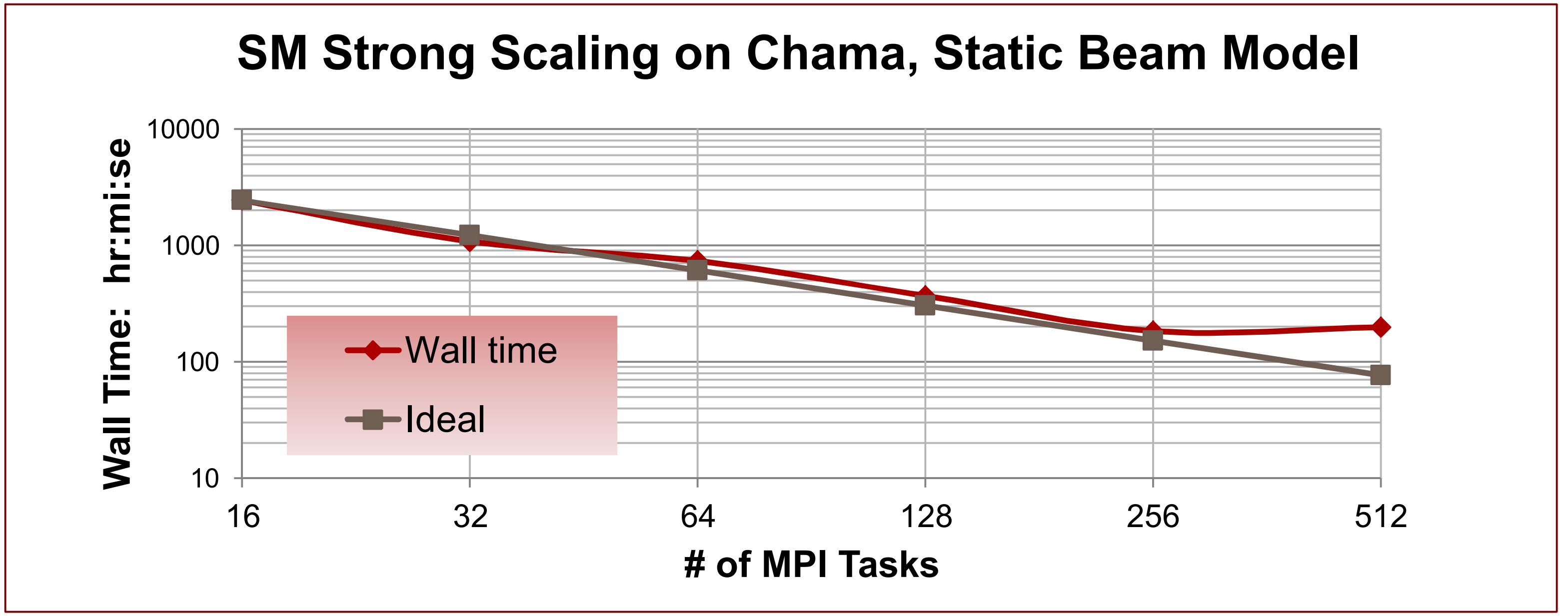
Time in point-to-point calls: 5.3% ||

Effective collective rate: 8.90e+07 bytes/s

Effective point-to-point rate: 2.80e+08 bytes/s

Sierra/SM scaling

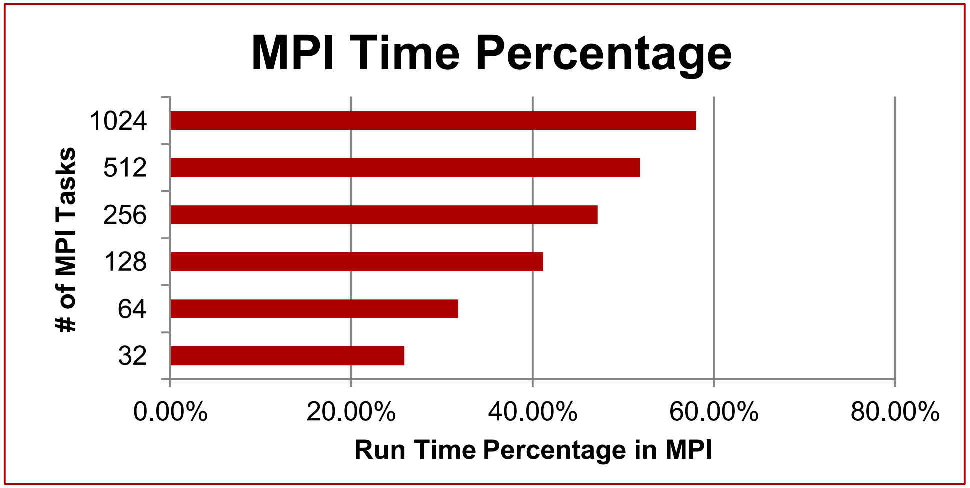


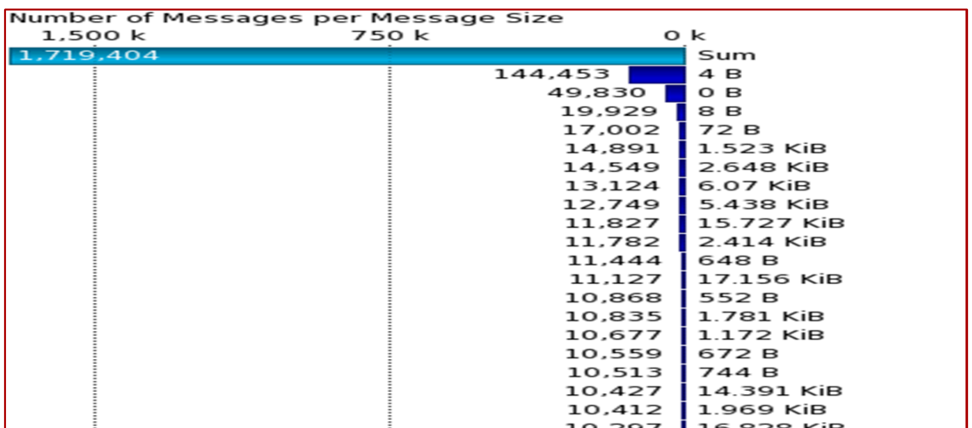


MPI Scaling

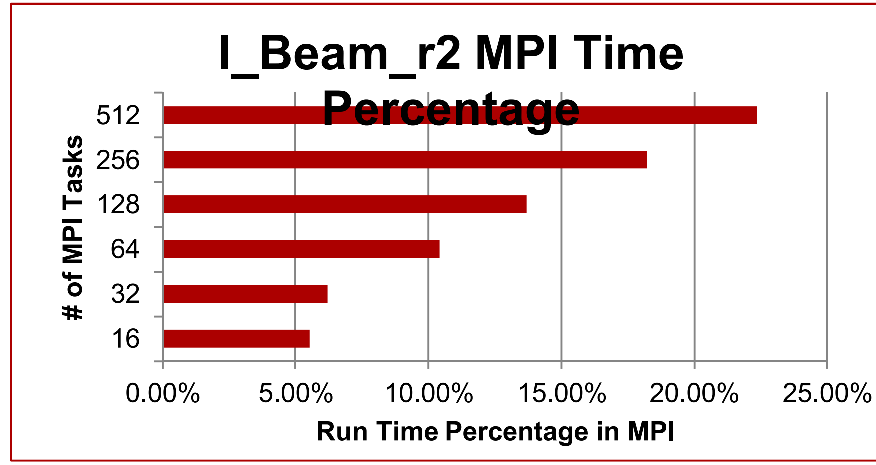
(Dynamics: large number of small (4 Byte) messages)
(Quasi-Statics: most messages in the 8kB to 25kB size)

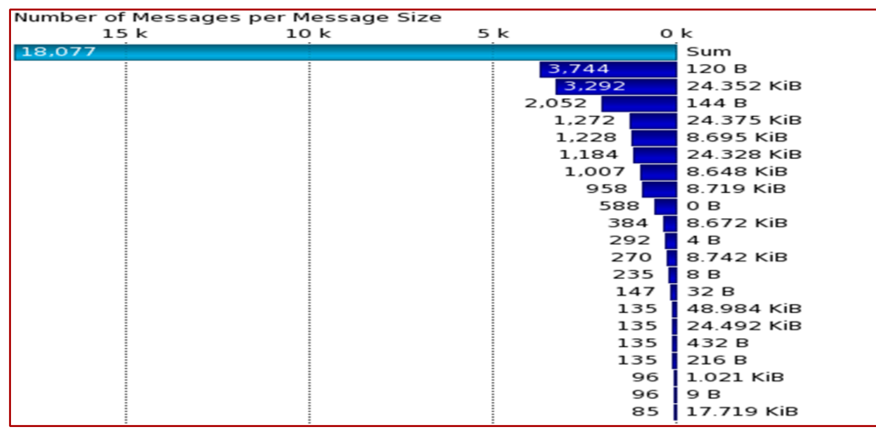
Explicit dynamic contact





Quasi-statics (implicit)



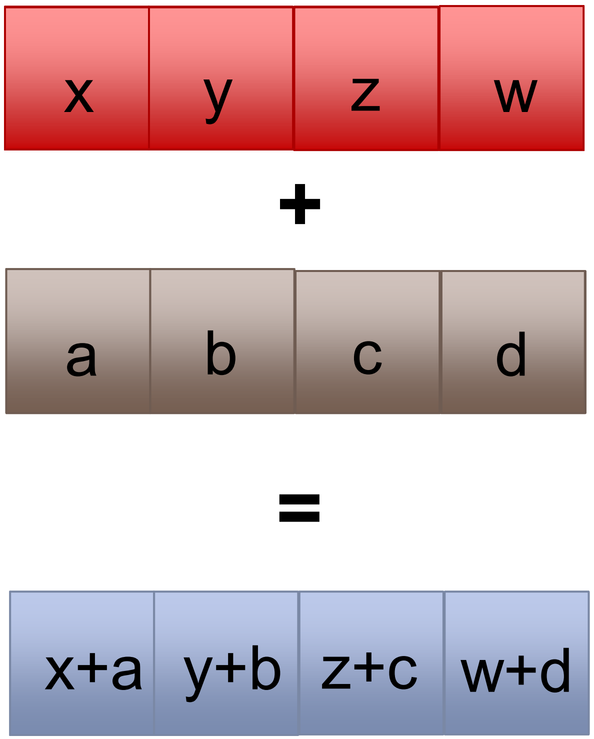


Sierra/SM Performance Summary

- 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

SSE2/AVX/AVX512 SIMD in Sierra-SM for nonlinear element assembly

Sandia National Laboratories



For simple loops, compilers can auto-vectorize:

```
for (int i=0; i < N; ++i) {
   a[i] = b[i] + c[i] * d[i];
}
```

Complicated loops don't auto-vectorize:

- Tensor33 multiply
- Eigenvectors
- Constitutive law evaluations

Sierra SSE2/AVX/AVX512 interface

Simd.h:

```
#if defined(AVX)
    const int ndoubles = 4;
    class Doubles { __m256d d };
#elif defined(SSE2)
    const int ndoubles = 2;
    class Doubles { __m128d d };
#else
    const int ndoubles = 1;
    typedef double Doubles;
#end
```

main.cc:

```
#include <Simd.h>

double x[ndoubles];

Doubles a = simd::load(x);
Doubles b = Doubles(2.1);

// operator overload:
Doubles c = a+b;

double output[ndoubles];
simd::store(output,c);
```

SIMD "EDSL"

Standard math functions:

sqrt, cbrt, log, exp, pow, fabs, copysign, min, max

Simd boolean types:

<, <=, >, >=, == returns booleans, e.g.,

Bools isTrue = x < 5;

Simd ternary:

Doubles z = if_then(isTrue, 1.0, y);

Simd reduction:

double a = reduceSum(z);

Operator overloads:

+, -, *, /, +=, -=, *=, /=

Also Simd Loads and Store

Bottlenecks:

_mm256_sqrt_pd() is only ~2X faster than std::sqrt()

Same with _mm512_sqrt_pd()?

Some compilers don't implement cbrt, log, exp, etc.

Performance improvements

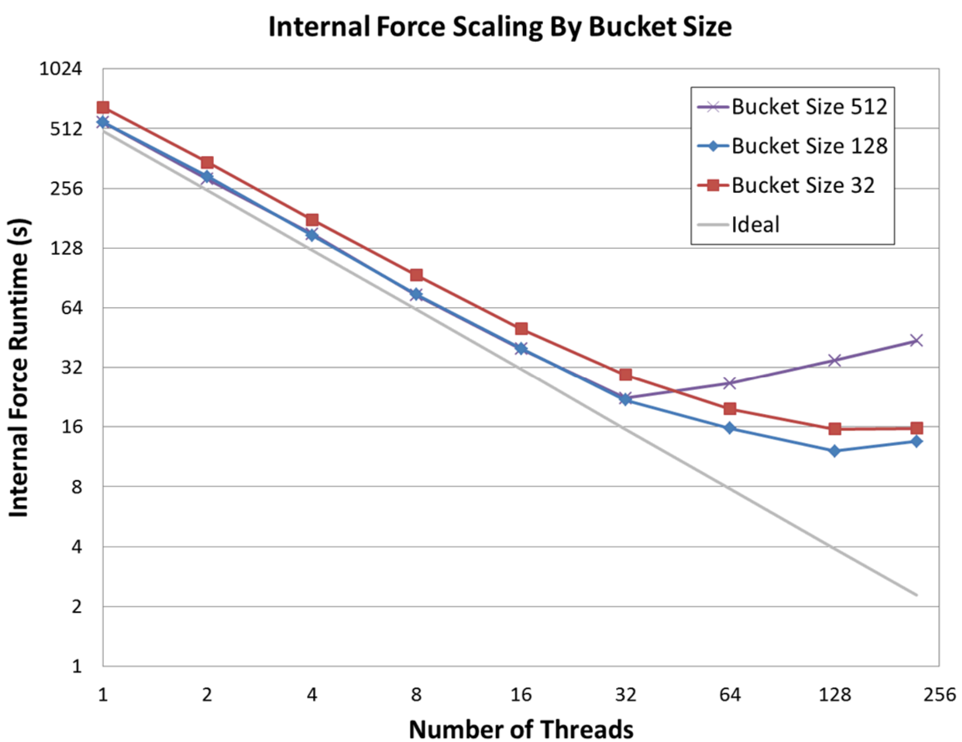
	SSE2	AVX	AVX512(KNC)
▪ Tensor multiply:	1.80 x	3.63 x	2.42 x
▪ Eigenvalue:	1.97 x	3.19 x	5.25 x
▪ Polar Decomp:	1.7 x	2.28 x	4.89 x

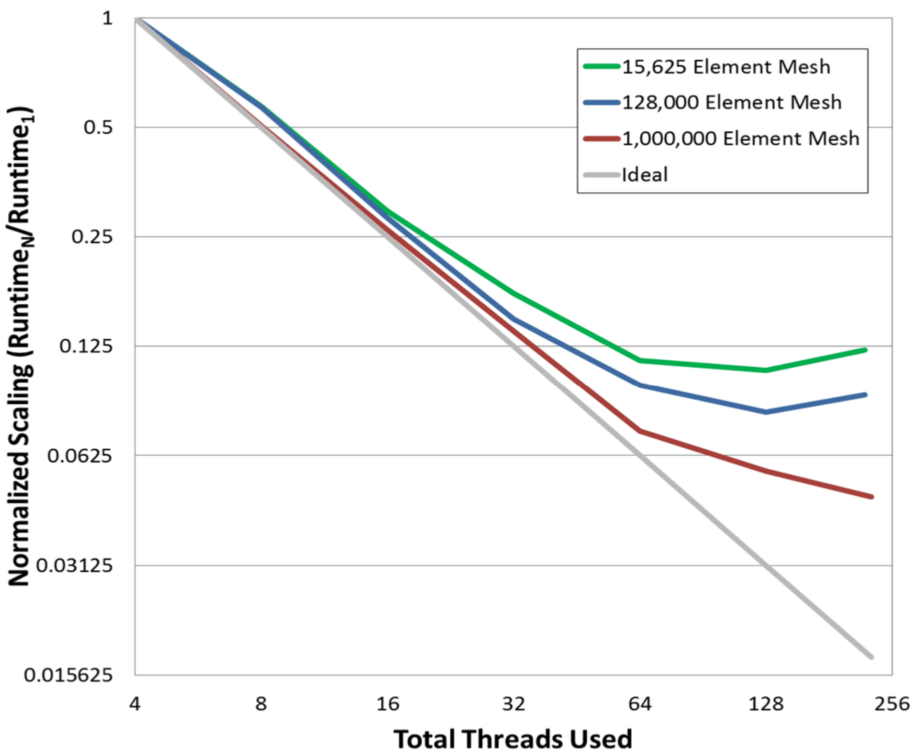
Real applications!

- Goodyear milestone: get run time of < 1.5 x Abaqus
- Previously at ~1.8 x
- Initial SIMD implementation
  - ***~ 40% overall improvement***
  - now at **~1.1-1.2x** !
- High velocity impact simulation
  - Originally, ~70% calculating 3x3 eigenvectors
  - Now ~10%

Early KNC results

- Sierra/SM compiles and runs on our test-bed KNC
- Use coloring algorithm to thread “force assembly”





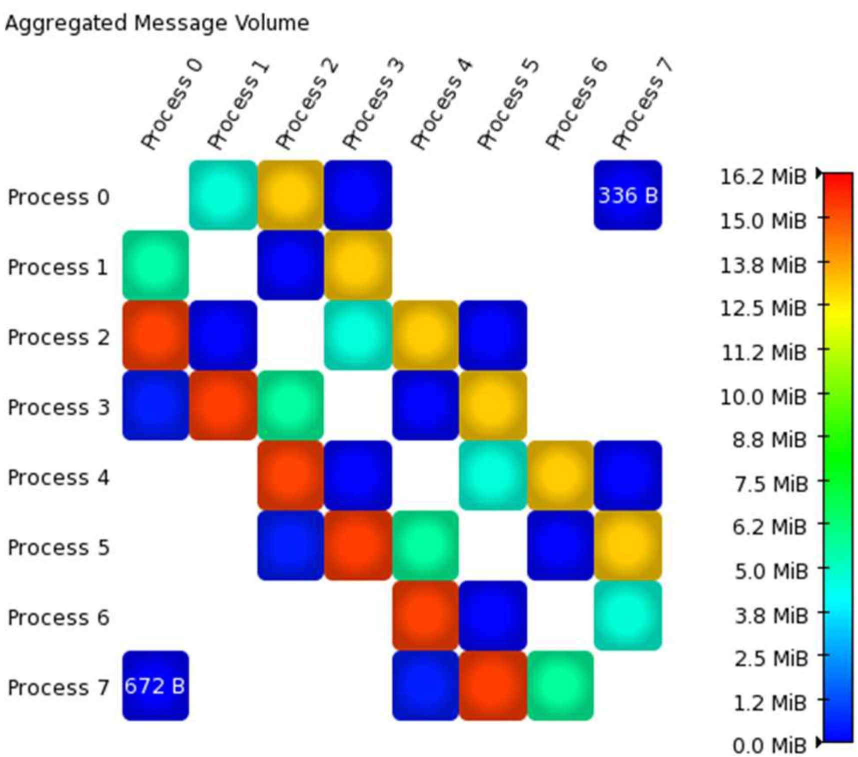
Data provided by Nate Crane

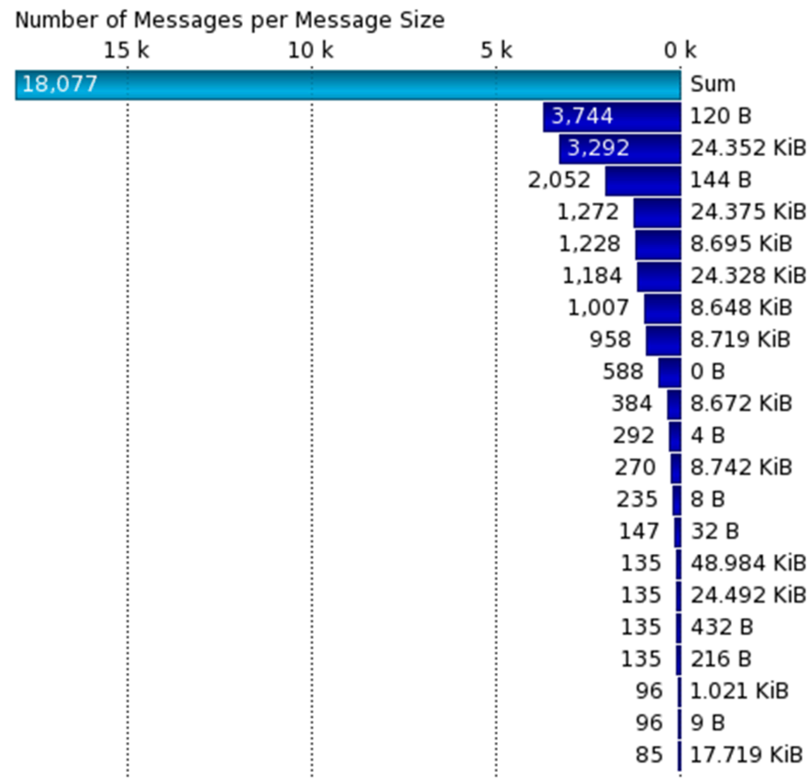
Take aways

- Solvers
  - Improve/thread sparse direct solvers
  - Bottleneck in both factorization and forward/backward solves
- Contact/Search
  - Serial cost dominated by random memory access
  - Parallel dynamic load balancing required
- Element assembly
  - Requires multi-threading (OpenMP?)
  - Better auto-vectorization would be nice
  - For now we are “hand” vectorizing

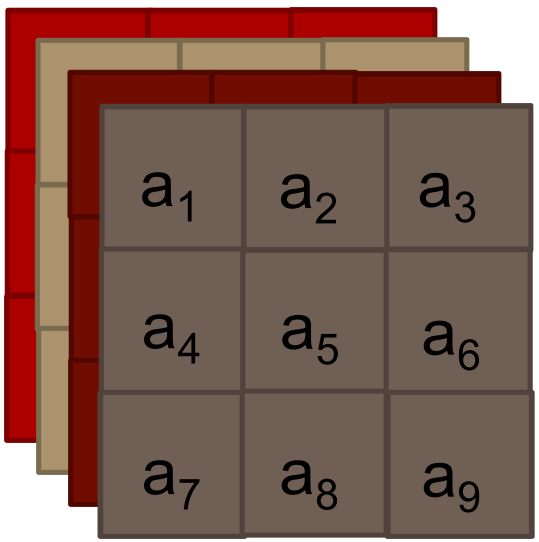
EXTRA SLIDES

QS model: Vampir Message Profile for 8 MPI tasks





SIMD Tensor class

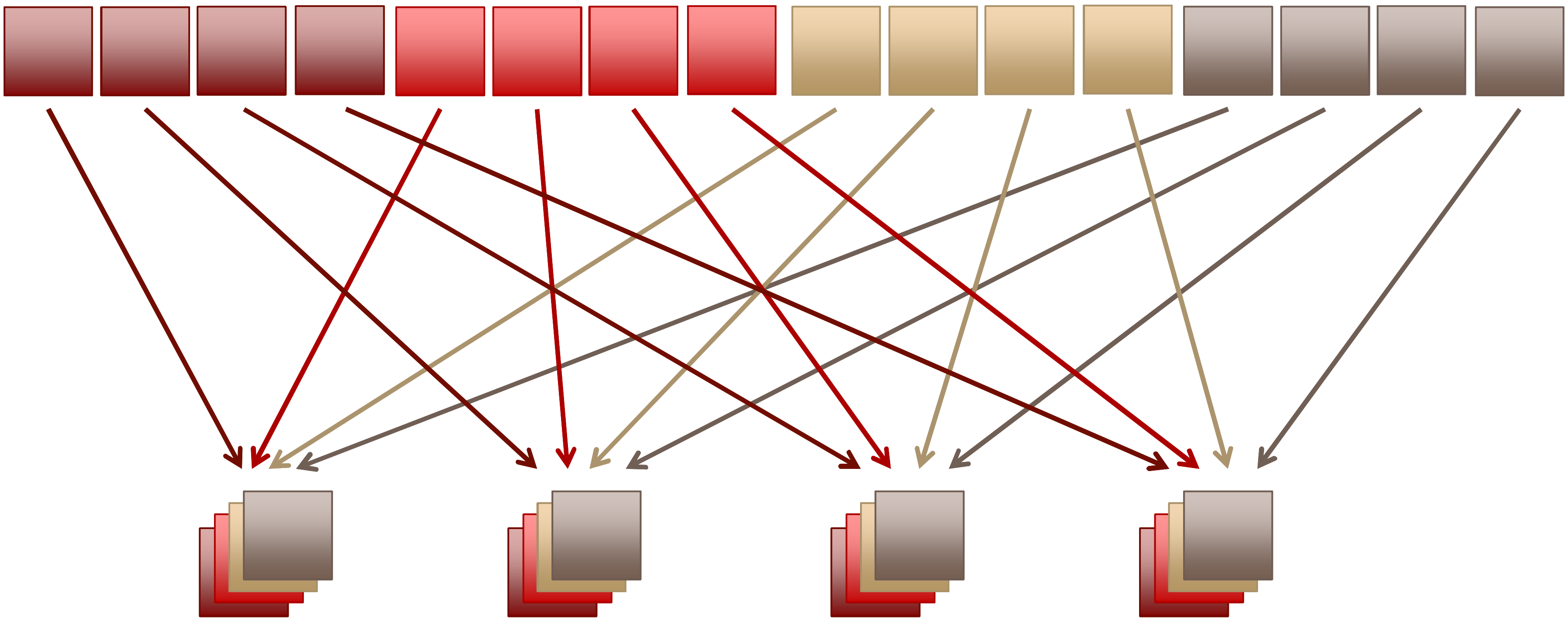


Process N tensors at a time
(N=4 with AVX, N=8 with AVX512):

```
double tensors[4*9];
// fill 4 tensor
Tensor33<Doubles> a(tensors)
c = mult(a,b);
Eigenvector(c,vects,vals);
c[0] = a[8] + b[4];
double output[4*9];
c.Store(output);
```

Loading 4 2x2 tensors

double a[4*tensor_size];

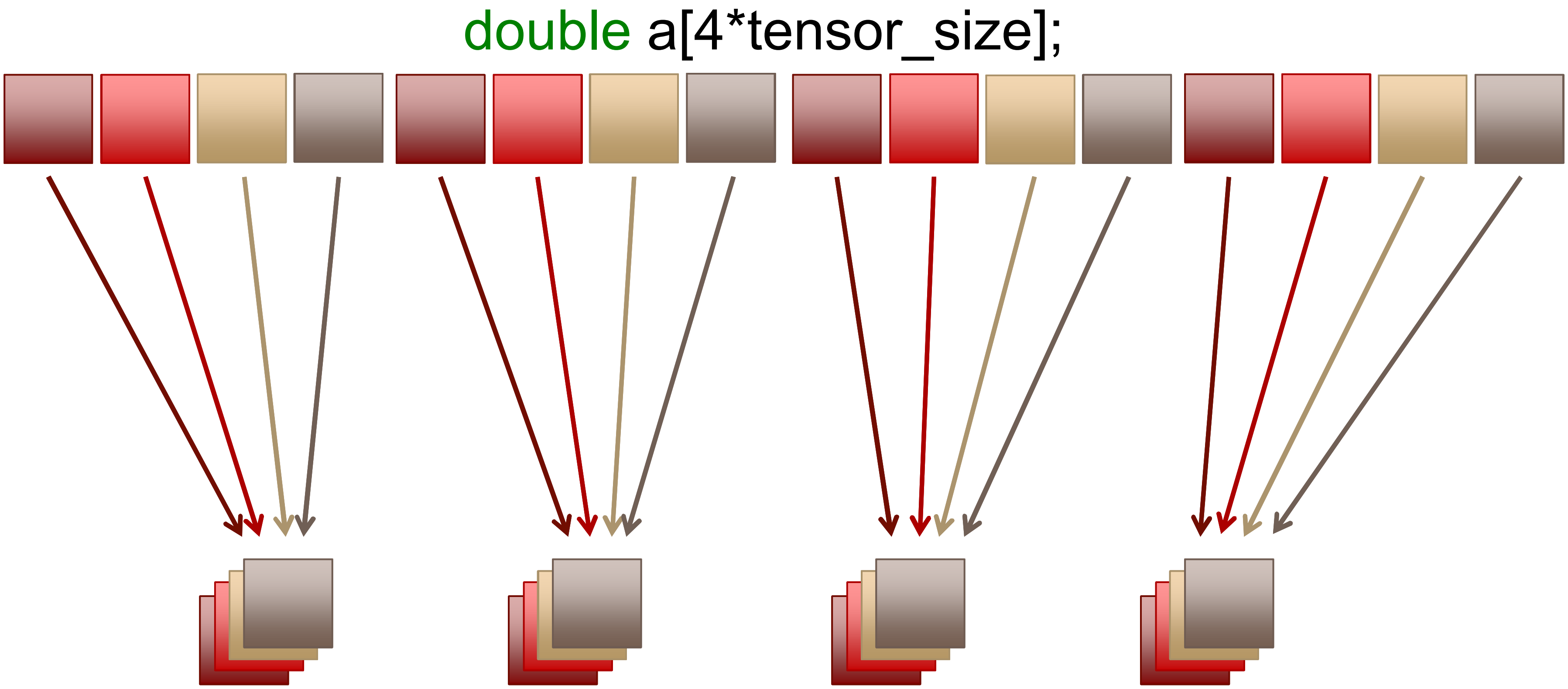


Doubles A[4];

for (int i=0; i < 4; ++i) A[i] = simd::load(a+i,tensor_size);

Slow memory access, but necessary unless we change memory layout of a.

Improved memory layout



Doubles A[4];
for (int i=0; i < 4; ++i) A[i] = simd::load_better(a+i);

Fast memory access, but requires some code refactor.

Fast approximation for 3x3 eigenvalues

- Analytic eigenvalue calculations require evaluating:

$$\cos(\arccos(x)/3)$$

- A Padé approximation can be derived (in Mathematica):

$$\cos(\arccos(x)/3) \approx \frac{0.866 + 2.13x + 1.89x^2 + 0.74x^3 + 0.12x^4 + 0.0066x^5}{1 + 2.26x + 1.8x^2 + 0.6x^3 + 0.078x^4 + 0.0027x^5}$$

- Error $< 5.6e^{-16}$ over entire range
- >7 X speed up over native C++ trig functions
- With AVX: speed up > 14 X