SAND2018-7918C

Exceptional service in the national interest

Sandia National Laboratories

Multiscale Solid Mechanics on Next-Generation Computing Hardware

David Littlewood
J. Antonio Perez
Michael Tupek
Brian Lester

13th World Congress on Computational Mechanics

July 25th, 2018

U.S. DEPARTMENT OF ENERGY

NNSA National Nuclear Security Administration

Sandia National Laboratories is a multi-mission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

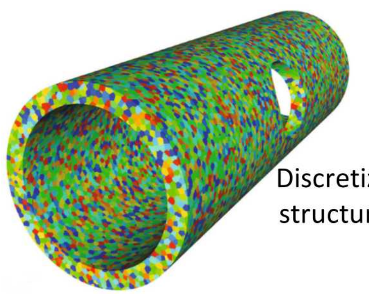
High-Fidelity Simulation across Multiple Length Scales

Mesoscale mechanisms can strongly affect system-level response

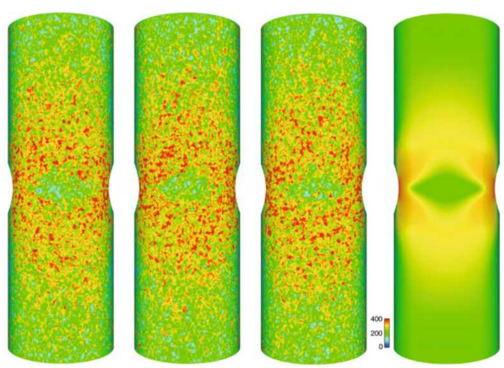
- Mesoscale influences stress concentrations, localization, material damage, failure, multiphysics phenomena, others …
- Component reliability depends on material variability, mean response is inadequate

Resolving the microstructure in engineering-scale simulations is intractable

- Motivates a multiscale strategy (domain coupling, FE^2, MsFEM, HMC, …)
- **Emerging hardware and software are critical for viability of multiscale methods**

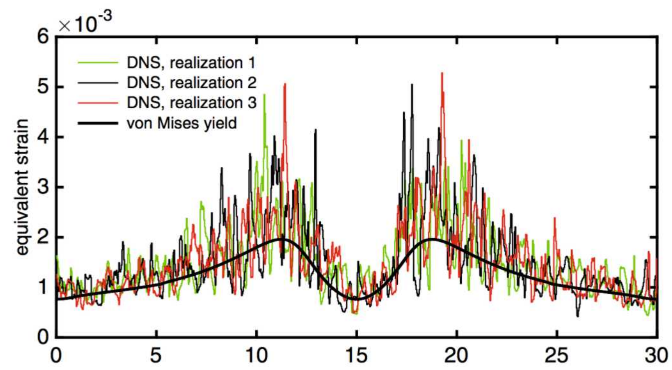

Discretized grain structure model

Von Mises Stress



Discretized grain structure

Homogenized



Bishop, J.E, Emery, J.M, Battaile, C.C., Littlewood, D.J., and Baines, A.J., 2016. Direct numerical simulations in solid mechanics for quantifying the macroscale effects of microstructure and material model-form error. *JOM: The Journal of the Minerals, Metals, and Materials Society 68(5), 1427-1445.*

Bishop, J.E., Emery, J.M., Field, R., Weinberger, C., and Littlewood, D.J. 2015. Direct numerical simulations in solid mechanics for understanding the macroscale effects of microscale material variability. *Computer Methods in Applied Mechanics and Engineering* 287, 262-289.

The Supercomputing Landscape is Changing

- What is the current definition of a "next generation" supercomputer?
  - On-node accelerators
    - Intel Knights Landing (KNL), NVidia GPU, etc.
  - Enables increased parallelism, e.g., MPI + X
  - Flops are cheap, memory management is critical
- DOE/NNSA Advanced Technology Systems
  - Trinity ATS-1, LANL
    - Intel Xeon (Haswell) & Intel Xeon Phi (KNL)
  - Sierra ATS-2, LLNL
    - IBM POWER9 CPUs & NVidia GPUs
  - Future ATS platforms …

ATS-1 Trinity
http://www.lanl.gov/projects/trinity/

ATS-2 Sierra
https://asc.llnl.gov

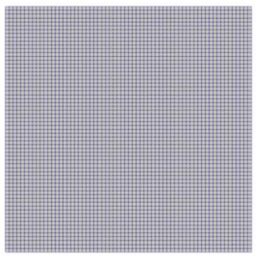
National Nuclear Security Administration

Advanced Simulation and Computing

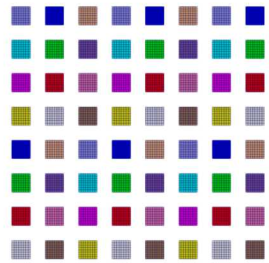
Predicting, with confidence, the behavior of nuclear weapons through comprehensive, science-based simulations.

Alternative Strategies for Software Parallelization

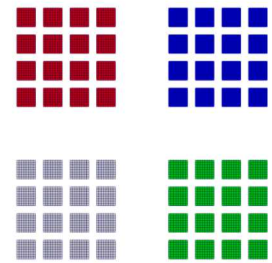
Choice of parallelization strategy is tied to hardware architecture



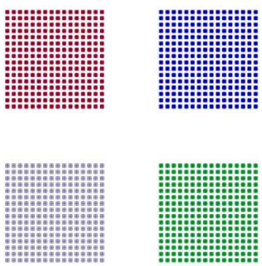
Serial execution
No decomposition



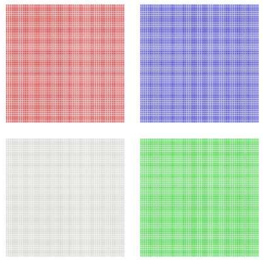
Traditional MPI
One MPI partition per core



MPI + X
One MPI partition per node
One thread per core $\mathcal{O}(10)$



MPI + X with large number of available threads (e.g., KNL)
One MPI partition per core
One thread per hardware thread $\mathcal{O}(100)$



MPI + X on a GPU machine
One MPI partition per core
One “thread“ per GPU execution path $\mathcal{O}(1000)$

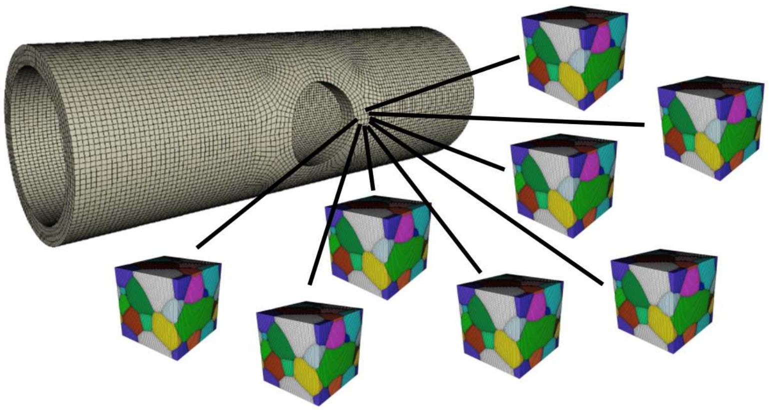
Advanced Computational Solid Mechanics Code Design

Goal: Enable high-fidelity computational simulation on next-generation computing platforms

- *Opportunity*: Run existing codes faster and on larger meshes
- *Opportunity*: New modeling approaches that were previously intractable
- *Risk*: Existing codes may not run well (or at all) on next-gen platforms

Exemplar:

FE^2 multiscale method is currently intractable for engineering-scale simulations, but may become viable with next-generation hardware



Current Efforts for Next-Gen Computational Solid Mechanics

Sandia National Laboratories

Adapting material models for performance portability

- Apply *Kokkos* package to Sandia material model library for improved performance across a variety of computing platforms
- Emphasis on material model API, data structures

High-performance engineering-scale simulations

- Focus on explicit transient dynamics
- Apply & evaluate software tools for next-gen HPC
  - *Kokkos, Qthreads, DARMA*

Hierarchical FE^2 multiscale approach

- Sub-models activated as needed based on macroscale behavior
- Creates load balancing challenge, amenable to asynchronous many-task (AMT) scheduling

NNSA National Nuclear Security Administration

ASC™ Advanced Simulation and Computing

Predicting, with confidence, the behavior of nuclear weapons through comprehensive, science-based simulations.

Applying *Kokkos* to Material Models

What is Kokkos?

- Abstraction layer / API for performance portability on NGP architectures

Design strategy

- `Kokkos::View` data structures enable optimal layout patterns and efficient transfer between host and device
- `Kokkos::parallel_for` mechanism for simultaneous evaluation of the constitutive model on a large number of material points

Key considerations

- Kernels executed on accelerator(s) must adhere to device restrictions
  - Thread safety
  - GPU utilization requires CUDA compatibility (severe restriction!)
- Strive for future-proof design that is compatible with existing material model API, restrict exposure to *Kokkos* complexity

Littlewood, D.J. and Tupek, M.R.. Adapting material models for improved performance on next-generation hardware. Memorandum SAND2007-5873, Sandia National Laboratories, Albuquerque, NM and Livermore, CA, 2017.

Kokkos Software Engineering Strategy

Store data in Kokkos::View containers

```
using Layout = Kokkos::CudaSpace::execution_space::array_layout;
using ExecutionSpace = Kokkos::CudaSpace::execution_space;
using View = Kokkos::View<double *, Layout, ExecutionSpace>;
View my_data;
```

Execute computational kernels using Kokkos::parallel_for

```
Kokkos::parallel_for("Stress",
                     mdpolicy_2d,
                     KOKKOS_LAMBDA (const int i_elem, const int i_ipt) {

def_grad_n = Kokkos::subview(def_grad_data_step_n, i_elem, i_ipt, Kokkos::ALL);
...

material_d->GetStress(time_previous,
                     time_current,
                     def_grad_n,
                     def_grad_np1,
                     stress_n,
                     stress_np1);
});
```

Performance Results for the Neo-Hookean Model

- Full utilization of a single compute node on conventional and next-gen hardware platforms
- `get_stress()` called on ~1M material points divided into 2K worksets
- Speed-up is given relative to serial execution on Intel Haswell architecture

Table 1: Performance comparison for a single node at full utilization: Speed-up of the neo-Hookean model relative to serial Haswell.

Configuration	Platform	Speed-up
Haswell: 32 cores	ascic	25.1
Haswell: 32 cores + 4-wide SIMD	ascic	88.5
Broadwell: 32 cores	ascic	40.0
Broadwell: 32 cores + 4-wide SIMD	ascic	124
KNL: 64 cores + 4x hyperthreads	mutrino	42.7
KNL: 64 cores + 2x hyperthreads[1] + 8-wide SIMD	mutrino	177
Kepler: Nvidia GPU	ascicgpu	162

Performance Results for the J2-Plasticity Model

- Full utilization of a single compute node on conventional and next-gen hardware platforms
- `get_stress()` called on ~1M material points divided into 2K worksets
- Speed-up is given relative to serial execution on Intel Haswell architecture

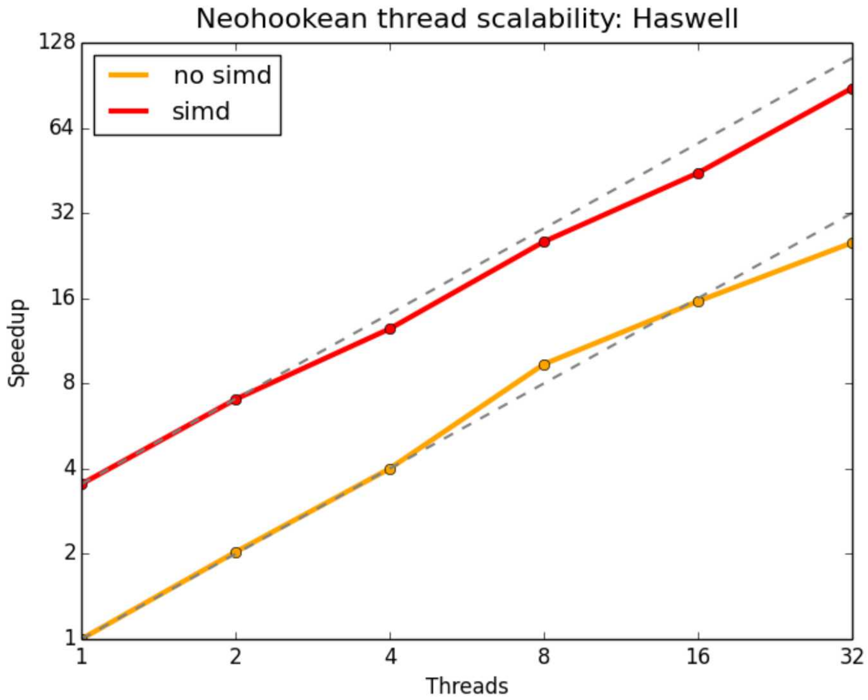
Table 2: Performance comparison for a single node at full utilization: Speed-up of the J2-plasticity model relative to serial Haswell.

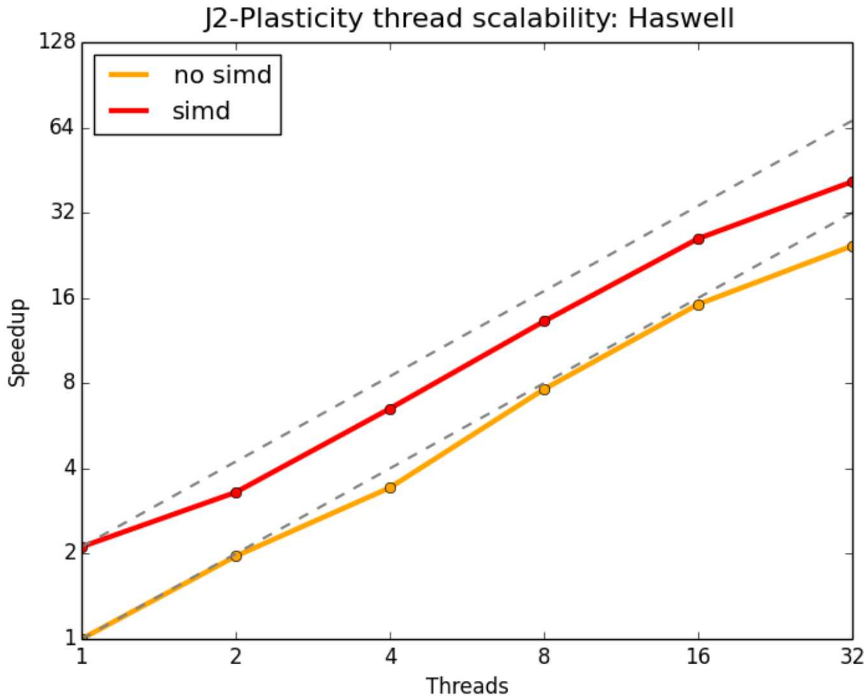
Configuration	Platform	Speed-up
Haswell: 32 cores	ascic	24.4
Haswell: 32 cores + 4-wide SIMD	ascic	42.2
Broadwell: 32 cores	ascic	34.9
Broadwell: 32 cores + 4-wide SIMD	ascic	53.8
KNL: 64 cores + 4x hyperthreads	mutrino	32.2
KNL: 64 cores + 4x hyperthreads + 8-wide SIMD	mutrino	76.4
Kepler: Nvidia GPU	ascicgpu	71.3

Performance Results: Thread scalability on Intel Haswell

Unit test results for Neo-Hookean and J2-plasticity models

- `get_stress()` called on ~1M material points divided into 2K worksets
- Speed-up is given relative to serial execution on Intel Haswell architecture
- Results show that NGP material models scale well on traditional hardware
- Results demonstrate effectiveness of SIMD vectorization

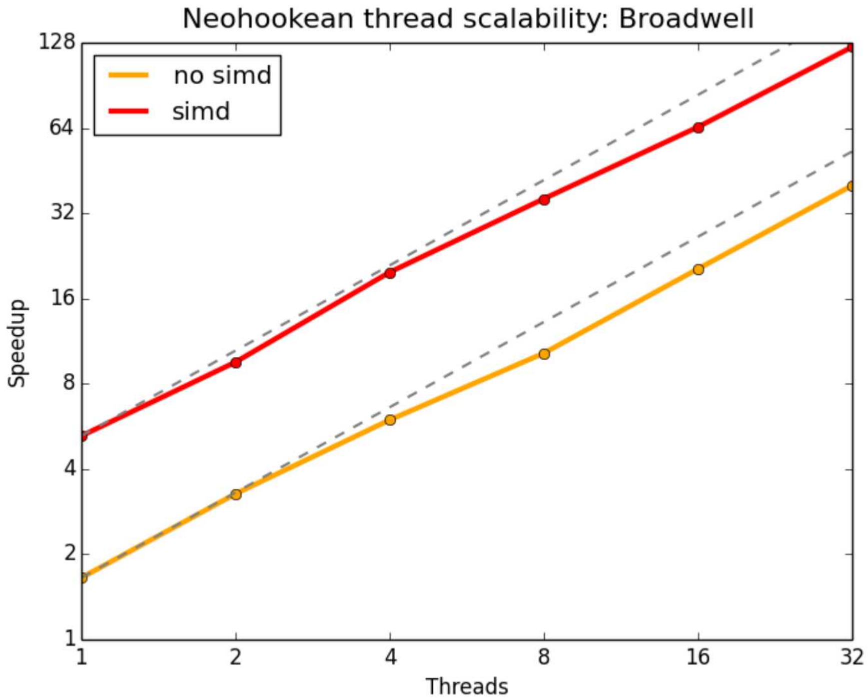


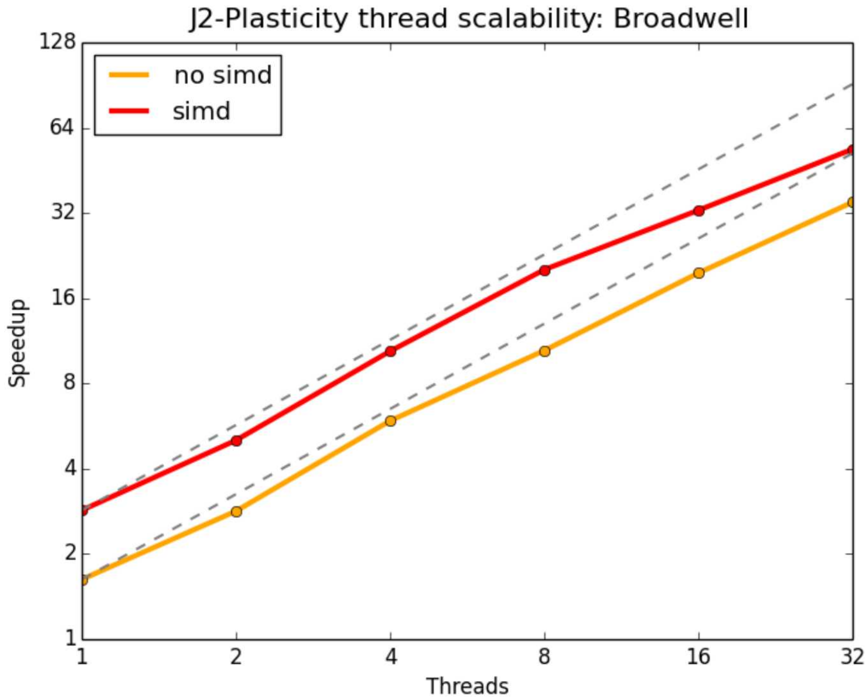


Performance Results: Thread scalability on Intel Broadwell

Unit test results for Neo-Hookean and J2-plasticity models

- `get_stress()` called on ~1M material points divided into 2K worksets
- Speed-up is given relative to serial execution on Intel Haswell architecture
- Results show that NGP material models scale well on traditional hardware
- Results demonstrate effectiveness of SIMD vectorization

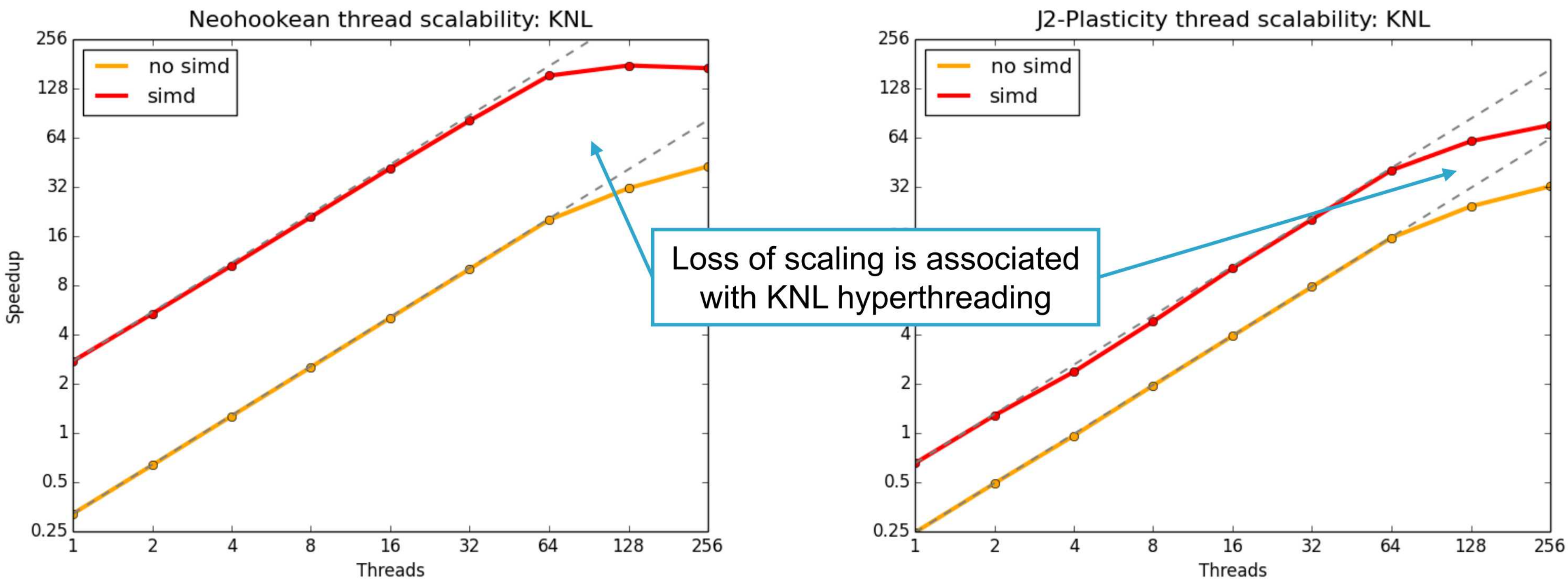




Performance Results: Thread scalability on Intel Knights Landing

Unit test results for Neo-Hookean and J2-plasticity models

- `get_stress()` called on ~1M material points divided into 2K worksets
- Speed-up is given relative to serial execution on Intel Haswell architecture
- Test executed on 1 KNL CPU with 64 cores, 8-wide SIMD, and up to 4 hyperthreads per core

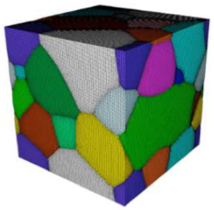


Enabling Full-Scale Explicit Dynamics Simulations

Goal: Solid mechanics proxy app that fully integrates recently-developed HPC software tools

- MPI + X via standard MPI, *Kokkos*, *Qthreads*
- *Kokkos* for performance portability
- *DARMA* for asynchronous many-task scheduling
- *Qthreads* for high-performance multi-threading

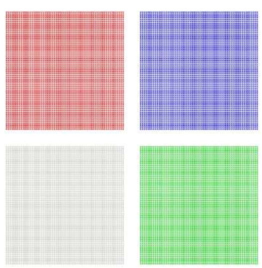
NimbleSM



Full Integration of *Kokkos* within *NimbleSM*

Enabling execution on GPUs requires pervasive code modifications

- Design strategy:
  - Apply Kokkos::parallel_for mechanism to execute computationally intensive kernels on multiple data sets simultaneously
  - Store data in Kokkos::View structures for performance portability
- Principal challenge:
  - Computational kernels must be compatible with CUDA
  - Limited functionality available (i.e., no access to std:: namespace)



MPI + X on a GPU machine
One MPI partition per core
One “thread“ per GPU execution path $\mathcal{O}(1000)$

Software Design

for each time step n **do**
    $t^{n+\frac{1}{2}} \leftarrow \frac{1}{2}\left(t^{n} + t^{n+1}\right)$
    $t^{n+1} \leftarrow t^{n} + \Delta t$
    $\mathbf{v}^{n+\frac{1}{2}} \leftarrow \mathbf{v}^{n} + \left(t^{n+\frac{1}{2}} - t^{n}\right)\mathbf{a}^{n}$
    for each d.o.f. i with a kinematic boundary condition **do**
        $v_i^{n+\frac{1}{2}} \leftarrow$ prescribed value
    $\mathbf{u}^{n+1} \leftarrow \mathbf{u}^{n} + \mathbf{v}^{n+\frac{1}{2}}\Delta t$
    ▷ Compute internal forces
    `element.ComputeDeformationGradients()` GPU
    `material_model.ComputeStress()` GPU
    `element.ComputeNodalForces()` GPU
    ▷ Sum internal forces at MPI partition boundaries
    `mpi.VectorReduction(internal_force)` CPU
    $\mathbf{a}^{n+1} \leftarrow \mathbf{M}^{-1}\boldsymbol{f}^{\,n+1}$
    $\mathbf{v}^{n+1} \leftarrow \mathbf{v}^{n+\frac{1}{2}} + \left(t^{n+1} - t^{n+\frac{1}{2}}\right)\mathbf{a}^{n+1}$
    if designated output step
        `io_system.WriteToFile()` CPU
end for

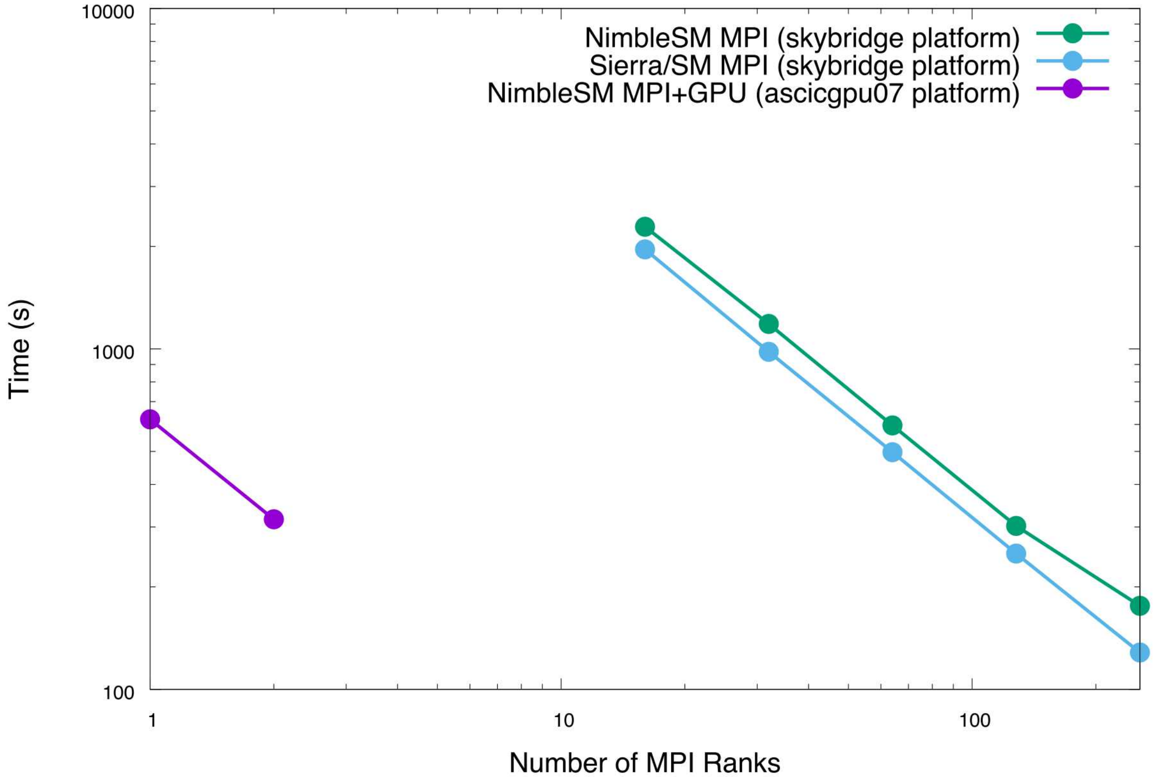
Element and material model objects instantiated on the GPU

MPI operations require data transfer between GPU and network

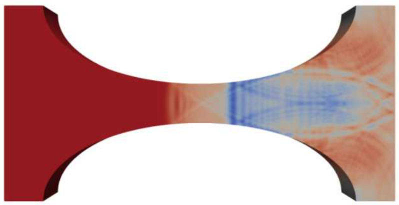
I/O operations require data transfer between GPU and network

Initial GPU Performance Results

- Explicit transient dynamics simulation
  - Neo-Hookean material model
  - Fully-integrated element formulation
- **Preliminary results suggest ~50x performance gain**



Wave propagation simulation
~5 million elements



Questions?

David Littlewood
djlittl@sandia.gov