MiniFENL: Fully Hybrid Parallel and Performance Portable Nonlinear Finite Element Mini-application using MPI+Kokkos

- Finite element solution of a simple nonlinear equation: $R(T) = -k \Delta T + T^2 = 0$
  - Restrict geometry, coefficient, and boundary conditions to obtain analytic solution for verification
  - 3D Cartesian domain with specified number of linear or quadratic hexahedral finite elements along each axis
  - Use non-uniform element sizes to force non-uniform element computations

- Solve via Newton’s method: $T_{i+1} = T_i - J^{-1}(T_i) \times R(T_i)$; $J(T) = \frac{\partial}{\partial T} R(T)$
  - Form linearized Residual and Jacobian for the nonlinear equation
  - Solve sparse linear system with simple conjugate gradient solver

- Research thread parallel algorithms and their performance
  - In-situ generation of domain-decomposed finite element mesh
  - Construction of sparse linear system graph from mesh connectivity
  - Computation of per-element contributions to Residual and Jacobian
  - Assembly of element contributions into sparse linear system
  - Solve sparse linear system with sufficient accuracy for Newton step

- Kokkos C++ library enables performance portable implementation across manycore architectures
  - Currently supported devices: multicore CPU with NUMA, Intel Xeon Phi, NVIDIA GPU (Kepler for best performance)
  - Thread-parallel execution of application kernels via parallel-for, parallel-reduce, and parallel-scan patterns
  - Multidimensional arrays with polymorphic data layout for device-appropriate memory access patterns

Kokkos is publicly available through the Trilinos project at trilinos.sandia.gov
Development team: H. Carter Edwards (PI), Daniel Sunderland, and Christian Trott
Thanks to our alpha users: Jonathan Berry, Erik Bowman, Matthew Bettencourt, Irina Demeshko, Kenneth Franko, Glen Hansen, Mark Hoemmen, Greg Mackey, Roger Pawlowski, Eric Phipps, Siva Rajamanickam

- Both strategies are thread-scalable
- Scatter-Atomic-Add strategy
  + Single, simple kernel applied via parallel-for over finite elements
  + Atomic-add is slower than regular add operation ($+=\$)
  + Non-deterministic order of summation with floating point round-off
- Gather-Sum strategy
  + Deterministic order of summation for bit-wise reproducibility
  + Large per-element Residual and Jacobian scratch arrays
  + Kernel #1: parallel-for over finite elements and fill scratch arrays
  + Kernel #2: parallel-for over matrix rows and “mine” scratch arrays

- Devices
  - NVIDIA Kepler K40 (Atlas) 12 GB
  - Intel Xeon Phi COES2, 61 cores, 1.2 GHz, 16 GB using only 60 cores with 1 or 4 hyperthreads per core

- Scale problem size: number of nodes (and elements)
  - Small problem size: parallel startup time dominates
  - Large problem size: computation time dominates

- Scatter-atomic-add is the winning strategy
  - If you can tolerate loss of bit-wise reproducibility
  - Avoids per-element Residual and Jacobian scratch arrays
  - Atomic-add is performant compared gather-sum operation

- Potential performance improvements
  - Hardware support for double precision atomic-add instead of 64bit CAS (compare-and-swap) implementation
  - Threaded element coloring algorithm allow scatter-add strategy to use $+=\$ instead of atomic-add