SAND2014-1229C

Mini-Driver Application for Testing FEM Assembly on Modern Architectures

Matthew T. Bettencourt[1]

Eric C Cyr[1]

[1]Scalable Algorithms,
Sandia National Laboratories,
Albuquerque, New Mexico, USA

SIAM Parallel Processing
Portland Oregon
Feb 19th, 2014

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

Outline

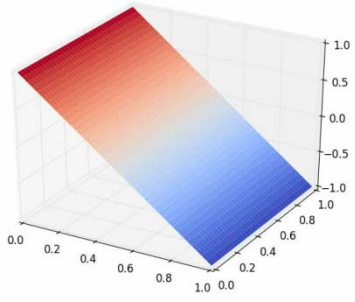
1. Overview – problem description, tools, …
2. Different assembly approaches
3. Results
4. Summary

Problem Description

- You've heard it 1000 times
  - Many more less powerful CPU cores on a chip
  - Memory to FLOPs is going down – codes require restructuring
- However, we have large FEM applications running on current hardware
  - How do we know how to migrate all this code to a new architecture?
- At Sandia we developed the Mantevo Mini-App suite http://mantevo.org
  - Mini-apps should show the complexity of a full application in a small package which is easy to understand, and simple to change
  - Minimal dependency on non-standard APIs, simple to build
- Multi-core solvers are currently active area of research
  - We feel less so about the rest of a typical FEM algorithm

Mini Driver – FEM Assembly Layer

- Develop a FEM assembly test code
  - Solve added to check correctness
- Model problem
  - Viscous Burgers equation $\frac{\partial u}{\partial t} + u \cdot \nabla u = \upsilon \Delta u$
- FEM
  - Standard Galerkin formulation with theta method time integration in residual correction form
  - Stiffness matrix computed with the help of Automatic Differentiation (AD) and the Sacado library
    - Hand turned Jacobian not implemented or compared
    - Full Jacobian calculation compared to just residual calculation



Kokkos: C++ Library / Programming Model for Manycore Performance Portability

- Portable to Advanced Manycore Architectures
  - Multicore CPU, NVidia GPU, Intel Xeon Phi (potential: AMD Fusion)
  - Maximize amount of user (application/library) code that can be compiled without modification and run on these architectures
  - Minimize amount of architecture-specific knowledge that a user is required to have
  - Allow architecture-specific tuning to easily co-exist
  - Only require C++1998 standard compliant
- Performant
  - Portable user code performs as well as architecture-specific code
  - ➢ Thread scalable – not just thread safety (no locking!)
- Usable
  - Small, straight-forward application programmer interface (API)
  - ➢ Constraint: don't compromise portability and performance

Kokkos: Collection of Libraries

- Core – lowest level portability layer
  - Portable data-parallel dispatch: parallel_for, parallel_reduce, parallel_scan
  - Multidimensional arrays with device-polymorphic layout for transparent and device-optimal memory access patterns
- Containers – built on core arrays
  - ➢ UnorderedMap – fast find and thread scalable insertion
  - Vector – subset of std::vector functionality to ease porting
  - Compress Row Storage (CRS) graph
- Linear Algebra
  - Sparse matrices and linear algebra operations
  - Wrappers to vendors' libraries
  - Portability layer for Trilinos manycore solvers
- Examples – where the code for this presentation resides
  - MiniFENL: finite element solution of non-linear system of equations

What is Automatic Differentiation (AD)?

- Technique to compute analytic derivatives without hand-coding the derivative computation
- How does it work -- freshman calculus
  - Computations are composition of simple operations (+, *, sin(), etc…) with known derivatives
  - Derivatives computed line-by-line, combined via chain rule
- Derivatives accurate as original computation
  - No finite-difference truncation errors
- Provides analytic derivatives without the time and effort of hand-coding them
- Provided by Trilinos/Sacado library

$$y = \sin(e^x + x \log x), \quad x = 2$$

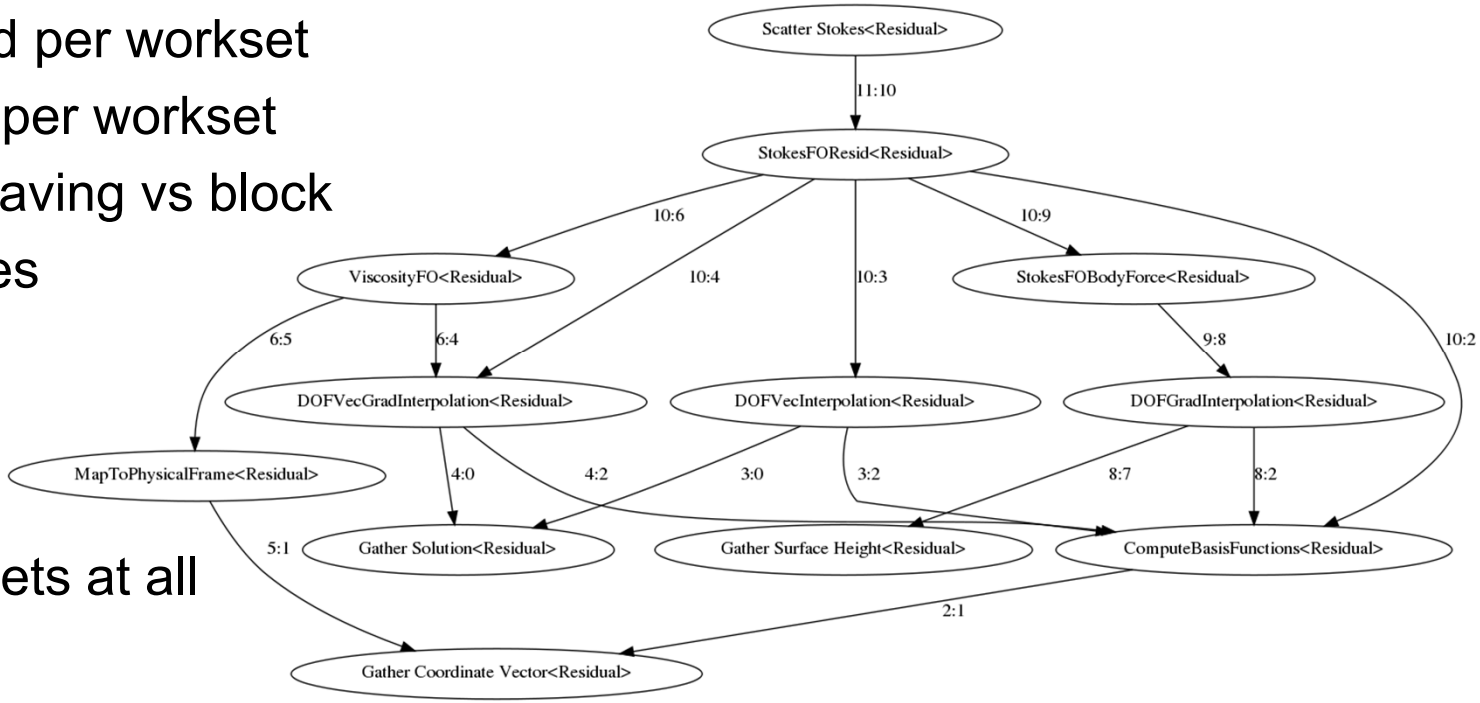
	x	$\frac{d}{dx}$
$x \leftarrow 2$	2.000	1.000
$t \leftarrow e^x$	7.389	7.389
$u \leftarrow \log x$	0.301	0.500
$v \leftarrow xu$	0.602	1.301
$w \leftarrow t + v$	7.991	8.690
$y \leftarrow \sin w$	0.991	-1.188

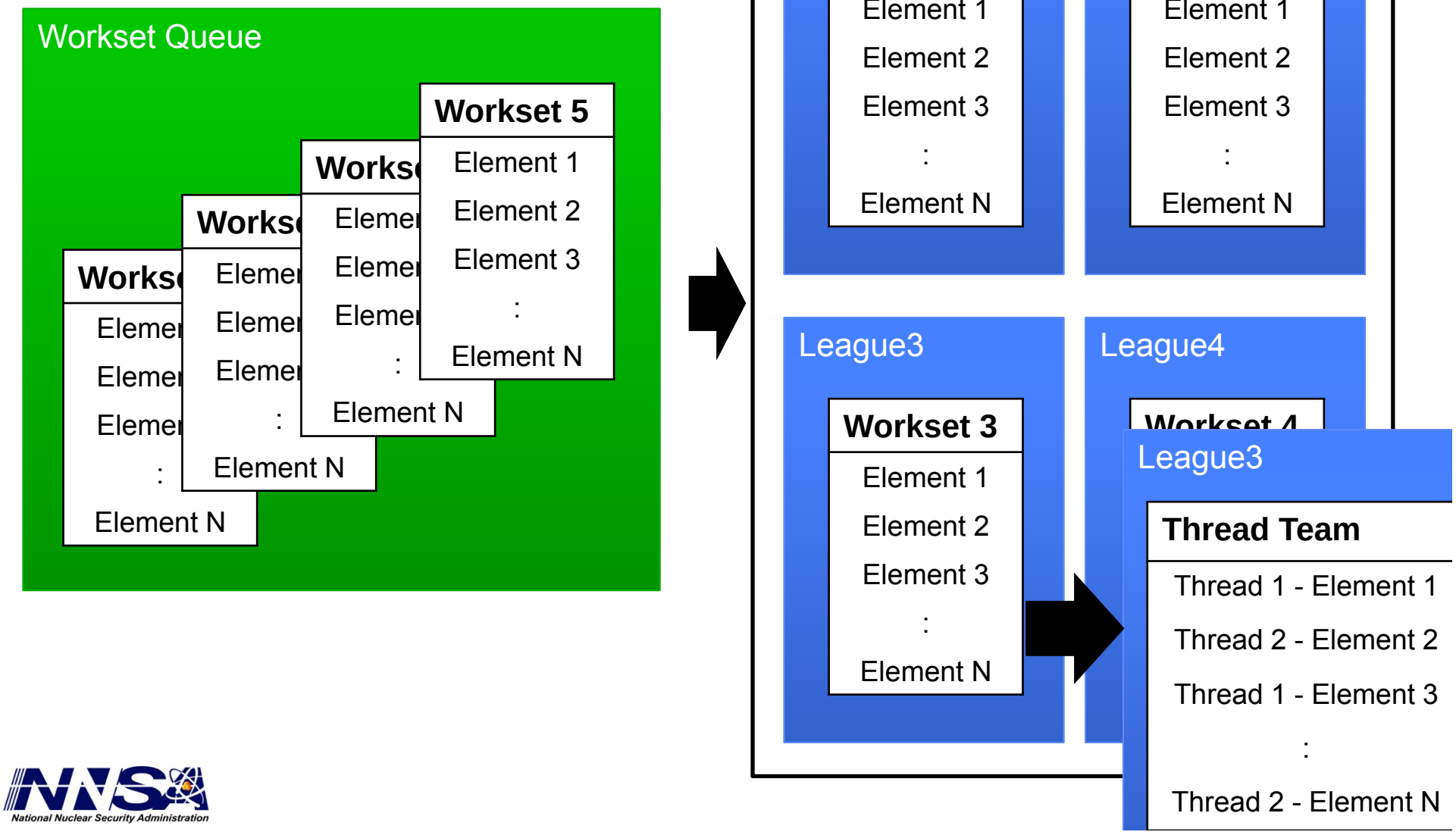
Hardware and Terminology

- Hardware platforms
  - Intel Xeon Nehalem 2x6 cores
  - Intel Knights Corner (Phi) with 57 cores x4 hyperthreads
  - Nvidia Tesla k20x cards
- Terminology
  - Thread – lowest unit of computing resource
  - Thread team – group of threads trained on a common task
    - Common shared local memory
    - Might be 4 threads in a team on a Phi or 256 on Tesla
  - League – group of teams tasked to handle the full workload

Assembly Approaches

- Worksets
  - Typical assembly follows a gather-work-scatter approach
  - Worksets have been developed to improve cache performance on scalar hardware
  - How to parallelize multicore worksets?
  - Thread per workset
  - Team per workset
  - Interleaving vs block updates
- No worksets at all

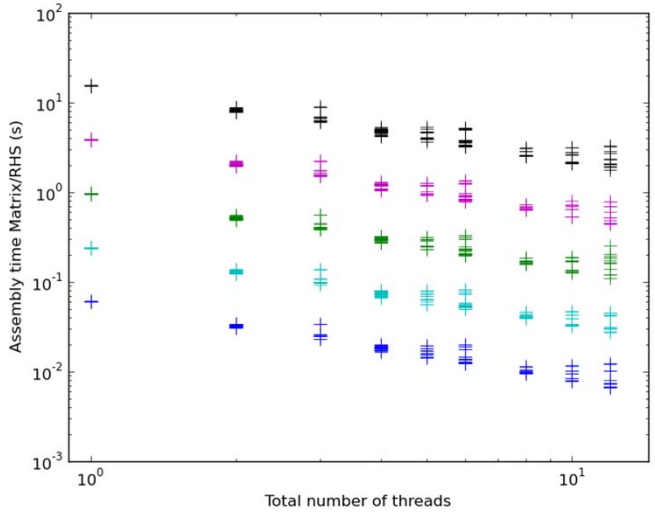


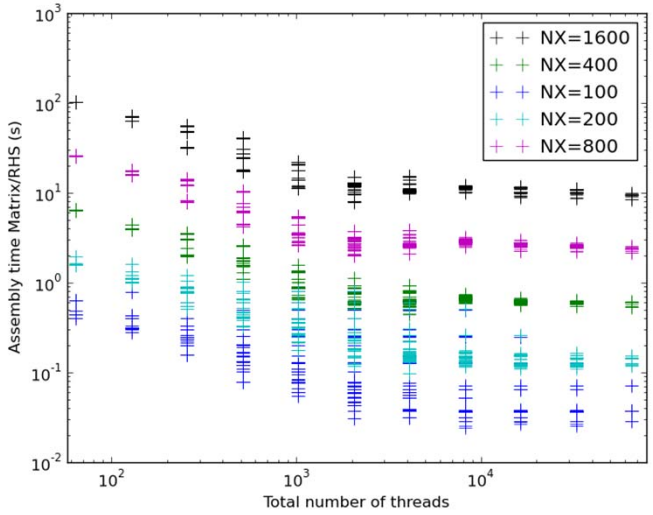


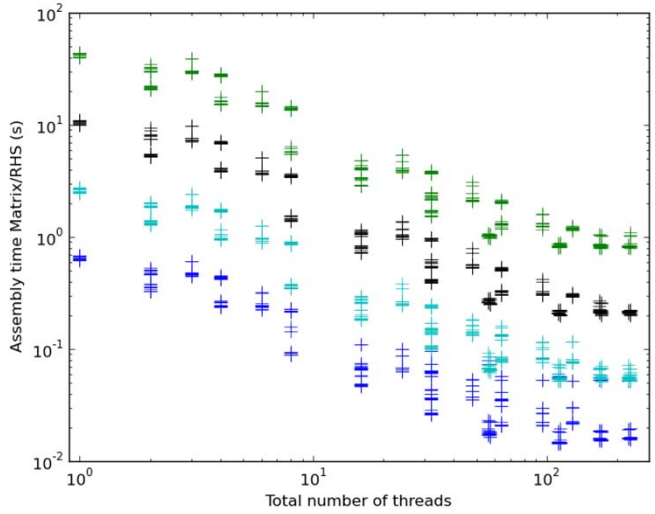
National Nuclear Security Administration

General Trends

- Scaling over all options for Matrix and RHS calculation
  - Upper Right – Desktop
  - Lower Left – Nvidia
  - Lower Right – Xeon Phi

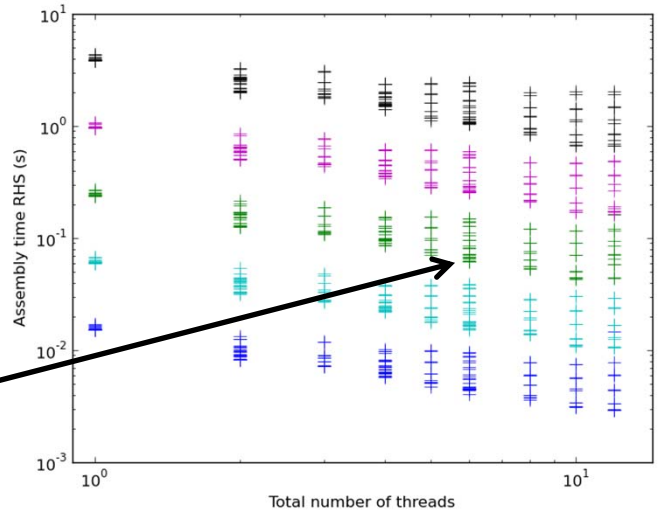




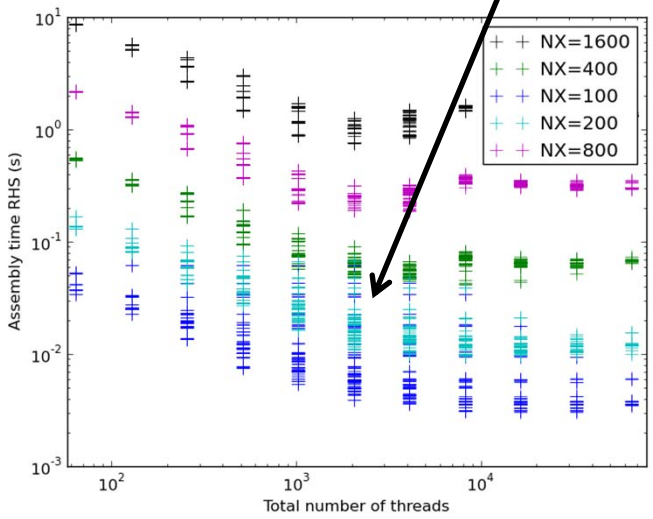


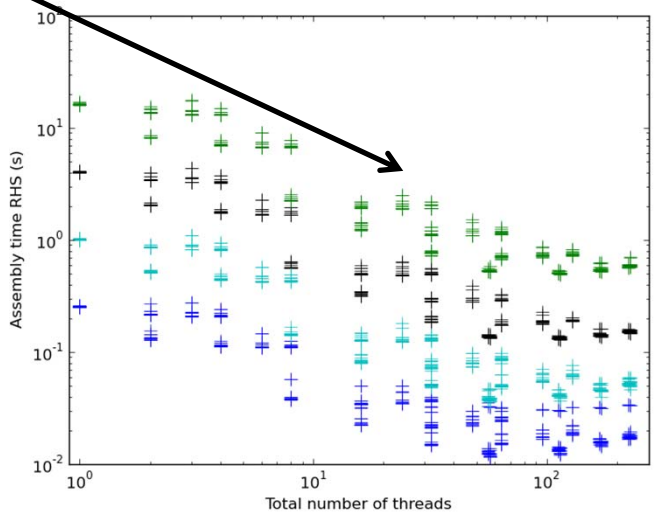
General Trends

- Scaling over all options for RHS calculation only
  - Upper Right – Desktop
  - Lower Left – Nvidia
  - Lower Right – Xeon Phi



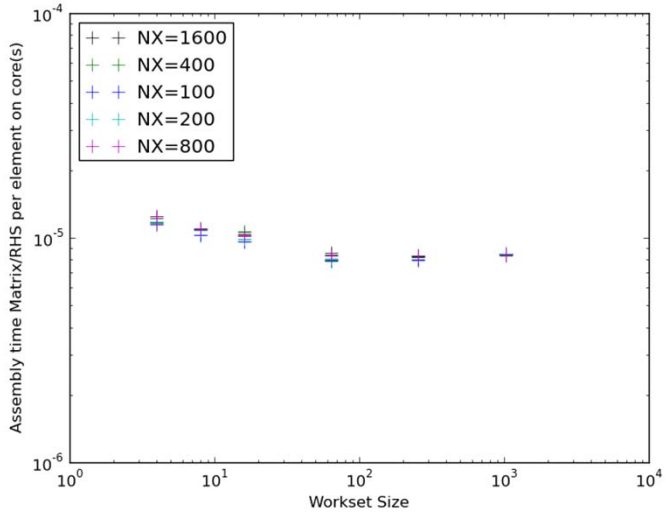
Variability examined next slides

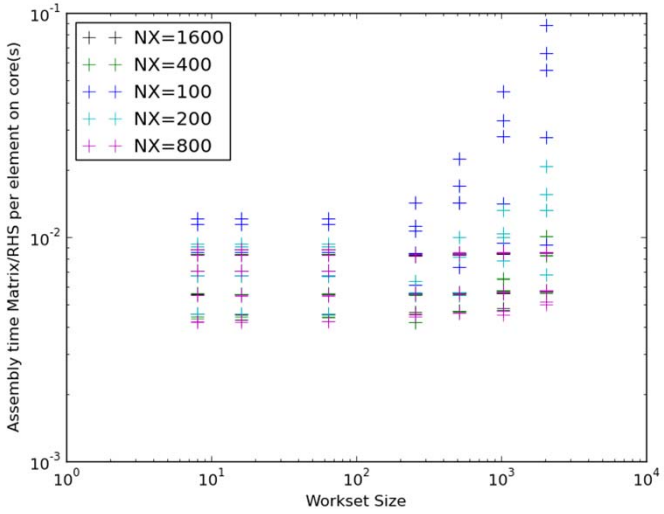


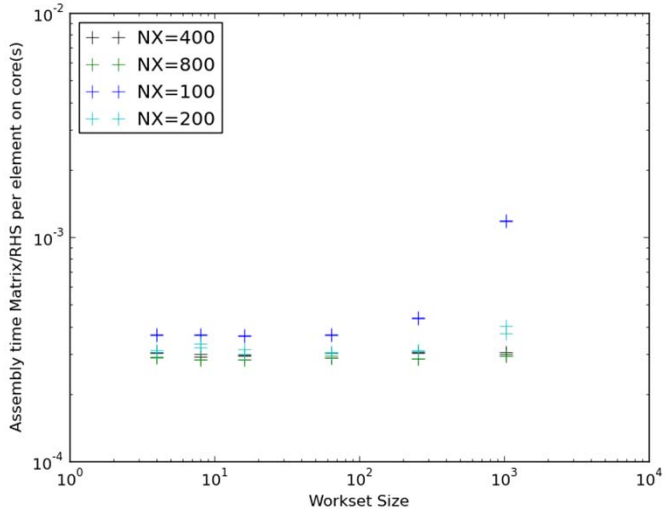


Weak Scaling – Worksets Sizes

- Effect of workset size
  - UR – Desktop – 10 threads
  - LL – Nvidia – 1024 threads
  - LR – Phi – 224 threads
- Effect is <u>minor!</u>
  - Except for desktop

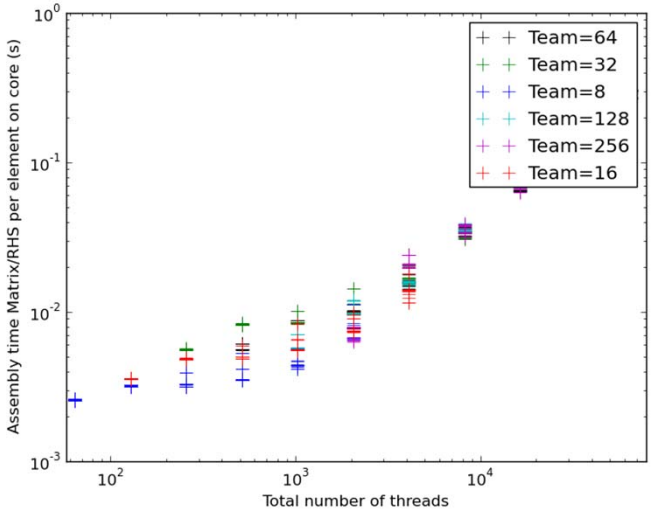


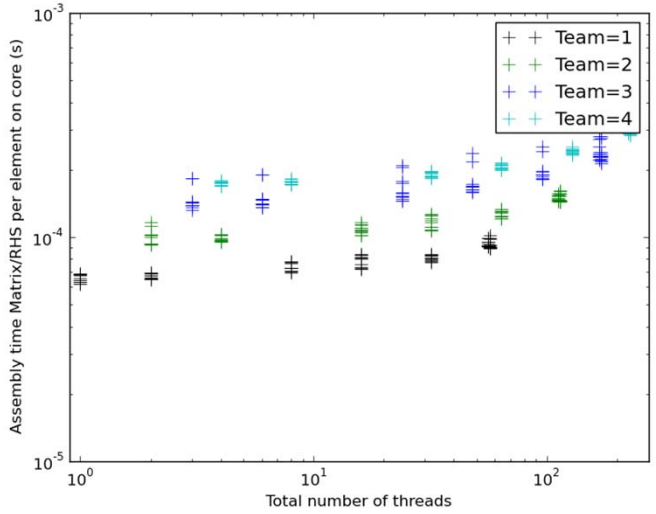




Weak Scaling – Thread Team Size

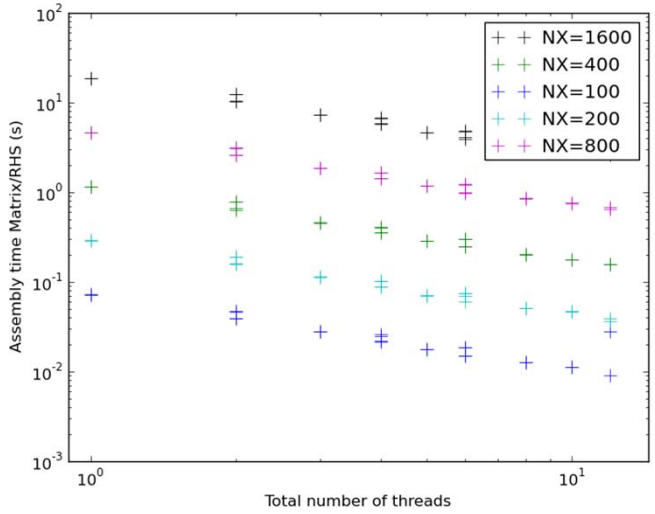
- Nvidia below run with a total of 1024 threads
  - Optimal small thread team size
  - Smaller variance with large problem size
- Xeon Phi below run with on a 400x400 problem
  - Diminished return using hyperthreads
  - Using all the threads/cores slightly faster most times, optimal may be 3 of 4

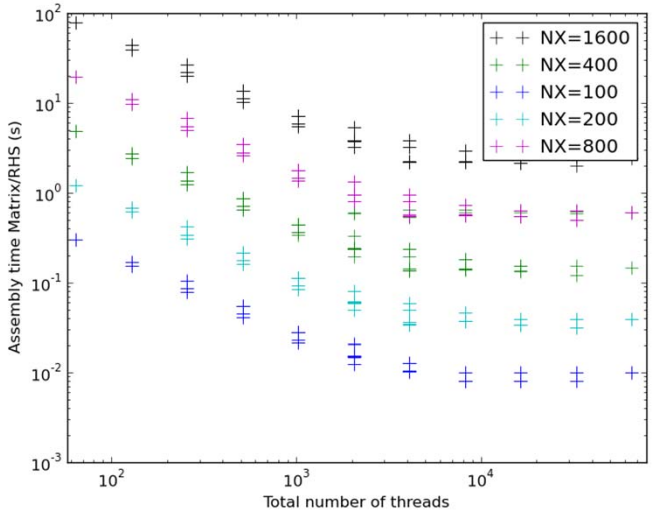


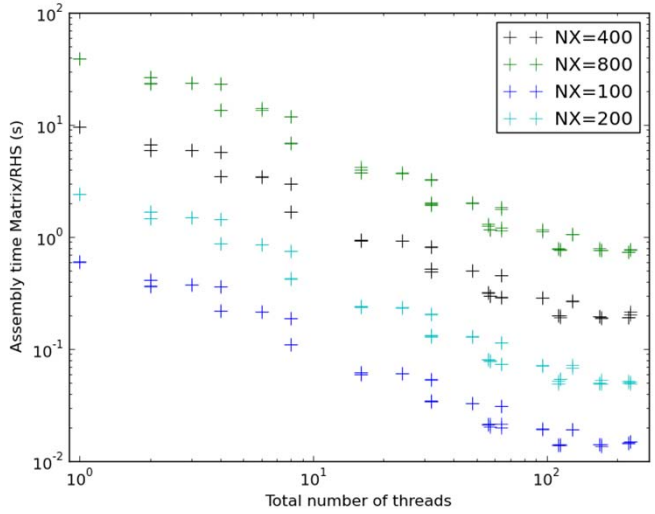


Simpler Approach – No Worksets

- Scaling over all options for Matrix and RHS calculation
  - Upper Right – Desktop
  - Lower Left – Nvidia
  - Lower Right – Xeon Phi

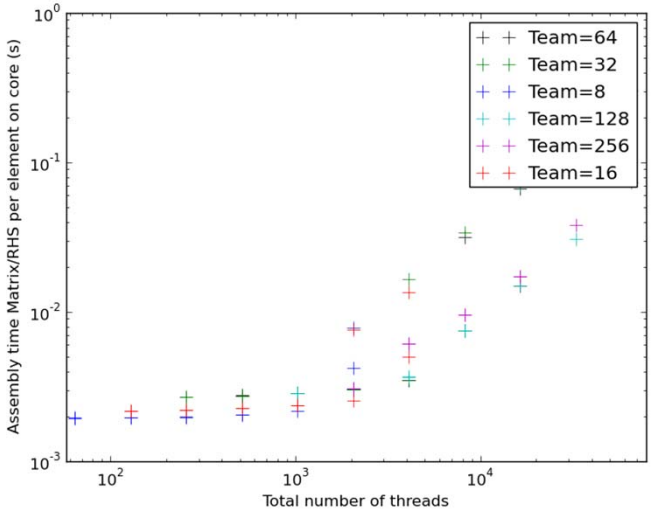


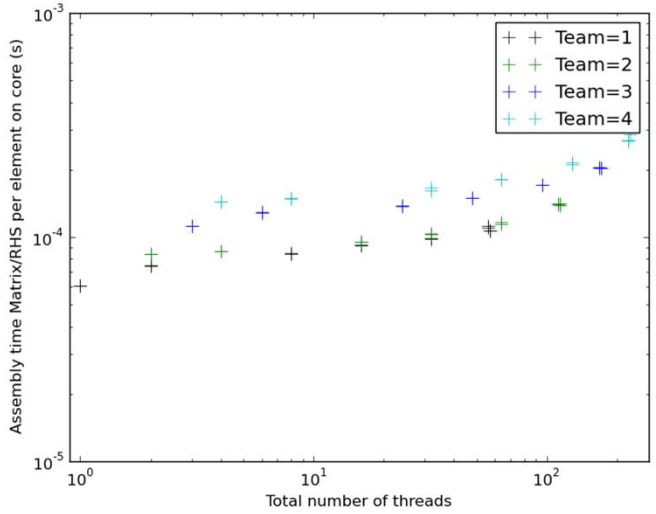




Weak Scaling – Thread Team Size – No Worksets

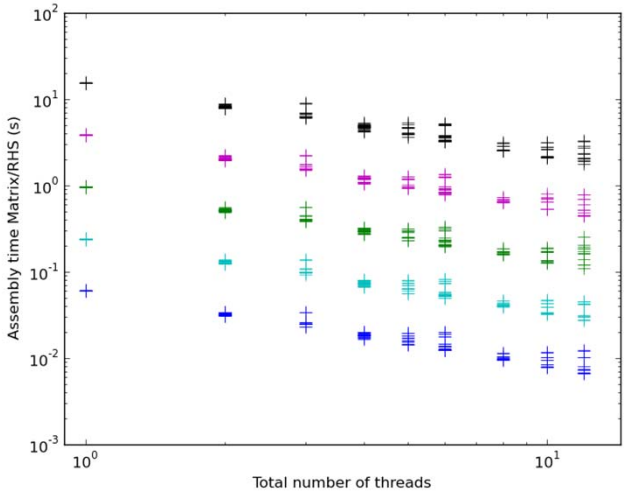
- Nvidia below run with a total of 1024 threads
  - Optimal small thread team size
  - Smaller variance with large problem size
- Xeon Phi below run with on a 400x400 problem
  - Diminished return using hyperthreads
  - Using all the threads/cores slightly faster most times, optimial may be 3 of 4

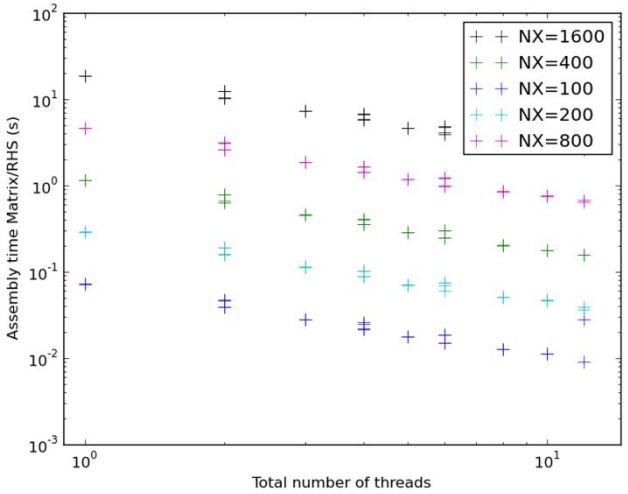




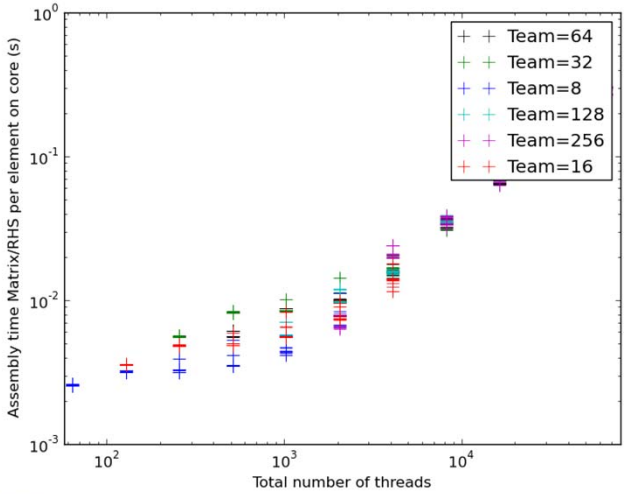
Worksets or No Worksets?

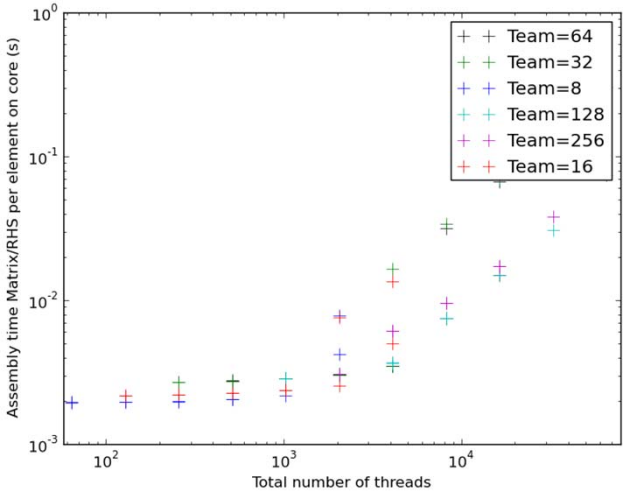
- Desktop – Left: Worksets Right: No Worksets ~30%





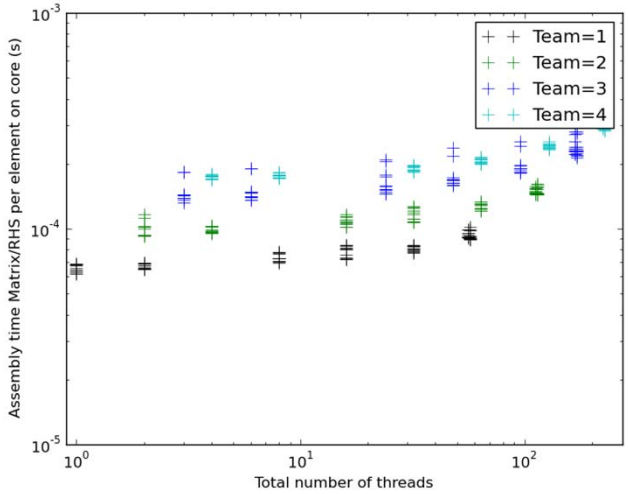
- Nvidia – Left: Worksets Right: No Worksets ~-50%

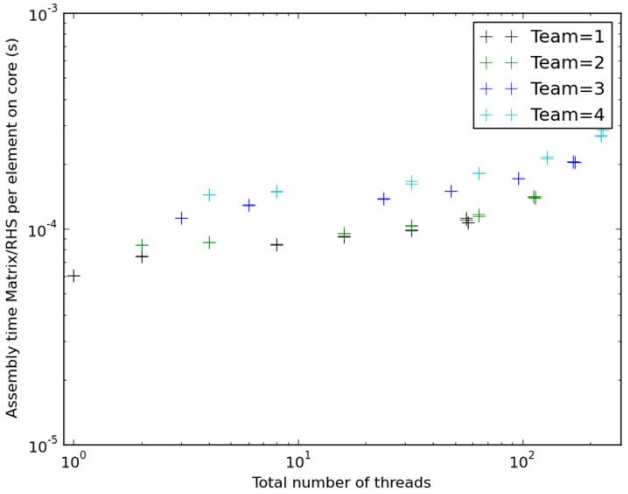




Worksets or No Worksets?

- Xeon Phi– Left: Worksets Right: No Worksets No Diff





- Worksets
  - Depends on the platform, they were developed for today's hardware, work well on it.
  - More complicated to code

Summary

- A simple code to test parallel multi threaded assembly of a finite element code
  - Threading tested on Xeon Phi, Nvidia and a standard desktop
  - Kokkos was the abstraction layer, it was simple to use and robust!
  - Code hopefully to be released as part of Mantivo Mini-apps
- Worksets were compared on the different platforms
  - Showed improved performance on today's hardware, but not next gen systems
  - Thread team can provide a large benefit on some platforms – Nvidia for example