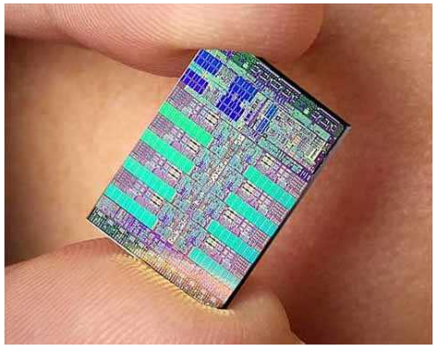
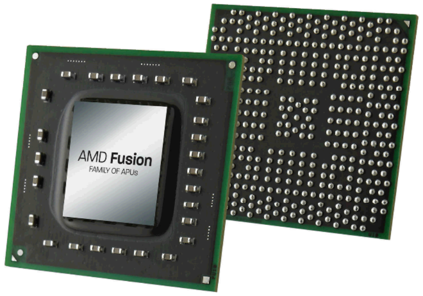
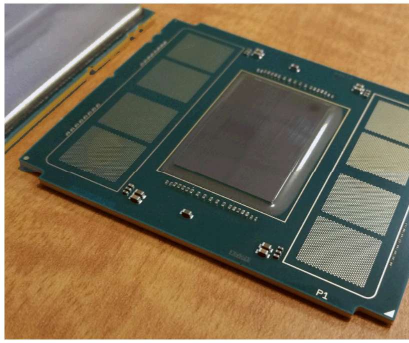
SAND2016-2672C

Exceptional service in the national interest



# Performance Portability for Linear Algebra with Kokkos

***Christian Trott***, Carter Edwards, Nathan Ellingwood, Si Hammond

Mehmet Deveci, Erik Boman, Andrew Bradley, Mark Hoemmen, Siva Rajamanickam

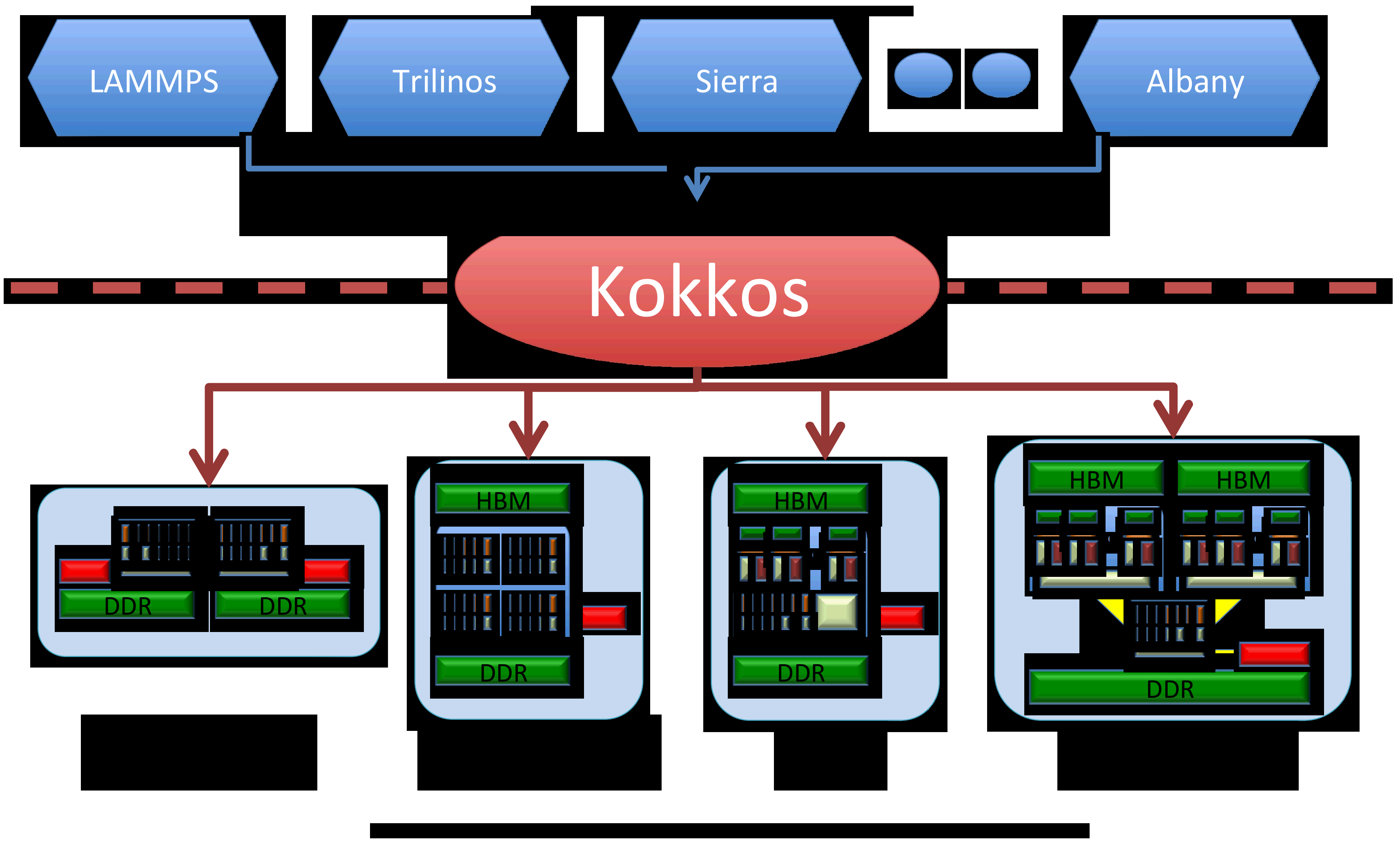
crtrott@sandia.gov

Center for Computing Research

Sandia National Laboratories, NM

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. 2016-0933PE

# Kokkos: *Performance, Portability and Productivity*



## Kokkos: *Performance, Portability and Productivity*

- A programming model implemented as a C++ library
- Abstractions for Parallel Execution and Data Management
  - Execution Pattern: What kind of operation (for-each, reduction, scan, task)
  - Execution Policy: How to execute (Range Policy, Team Policy, DAG)
  - Execution Space: Where to execute (GPU, Host Threads, PIM)
  - Memory Layout: How to map indicies to storage (Column/Row Major)
  - Memory Traits: How to access the data (Random, Stream, Atomic)
  - Memory Space: Where does the data live (High Bandwidth, DDR, NV)
- Supports multiple backends: OpenMP, Pthreads, Cuda, Qthreads, Kalmar (experimental)
- Sandia application teams committed to Kokkos as its path for transitioning legacy codes, and as part of its new codes
  - Trilinos, LAMMPS, Albany, Sierra Mechanics, …

## Going Production

- Kokkos released on github in March 2015
  - Develop / Master branch system => merge requires application passing
  - Testing Nightly: 11 Compilers, total of 90 backend configurations, warnings as errors
  - Extensive Tutorials and Documentation > 300 slides/pages
- Trilinos NGP stack uses Kokkos as only backend
  - Tpetra, Belos, MueLu etc.
  - Working on threading all kernels, and support GPUs
- Sandia Sierra Mechanics going to transition to Kokkos
  - Decided to go with Kokkos instead of OpenMP (only other realistic choice)
  - FY 2016: prototyping threaded algorithms, explore code patterns
  - Data management postponed to FY 2017 and follow on
- Sandia ATDM has Kokkos as big component
  - All ATDM Apps are using Kokkos
  - Add System level Tasking with Dharma later

# KokkosP Profiling Interface

- Dynamic Runtime Linkable profiling tools
  - Not LD_PRELOAD based (horray!)
  - Profiling hooks are always enabled (i.e. also in release builds)
    - Compile once, run anytime, profile anytime, no confusion or recompile!
  - Tool Chaining allowed (many results from one run)
  - Very low overhead if not enabled
- Simple C Interface for Tool Connectors
  - Users/Vendors can write their own profiling tools
  - VTune, NSight and LLNL-Caliper
- Parallel Dispatch can be named to improve context mapping

- Initial tools: simple kernel timing, memory profiling, thread affinity checker, vectorization connector (APEX-ECLDRD)
- Contact: Simon Hammond (sdhammo@sandia.gov)

## Enhancing Productivity: Using C++ Lambdas

- C++11 Feature which simplify using abstraction layers

***Pragma Based OpenMP:***

```
#pragma omp parallel for
for(int i=0; i<N; i++) {
  a[i] += b[i];
}
```

***Functor Based Kokkos:***

```
struct vector_add {
    View<double*> a;
    View<double*> b;
    vector_add(View<double*> a_, View<double*> b_):
      a(a_),b(b_){}
    KOKKOS_INLINE_FUNCTION
    void operator() (const int&i) const {
        a(i) += b(i);
    }
};

parallel_for( N, vector_add(a,b));
```

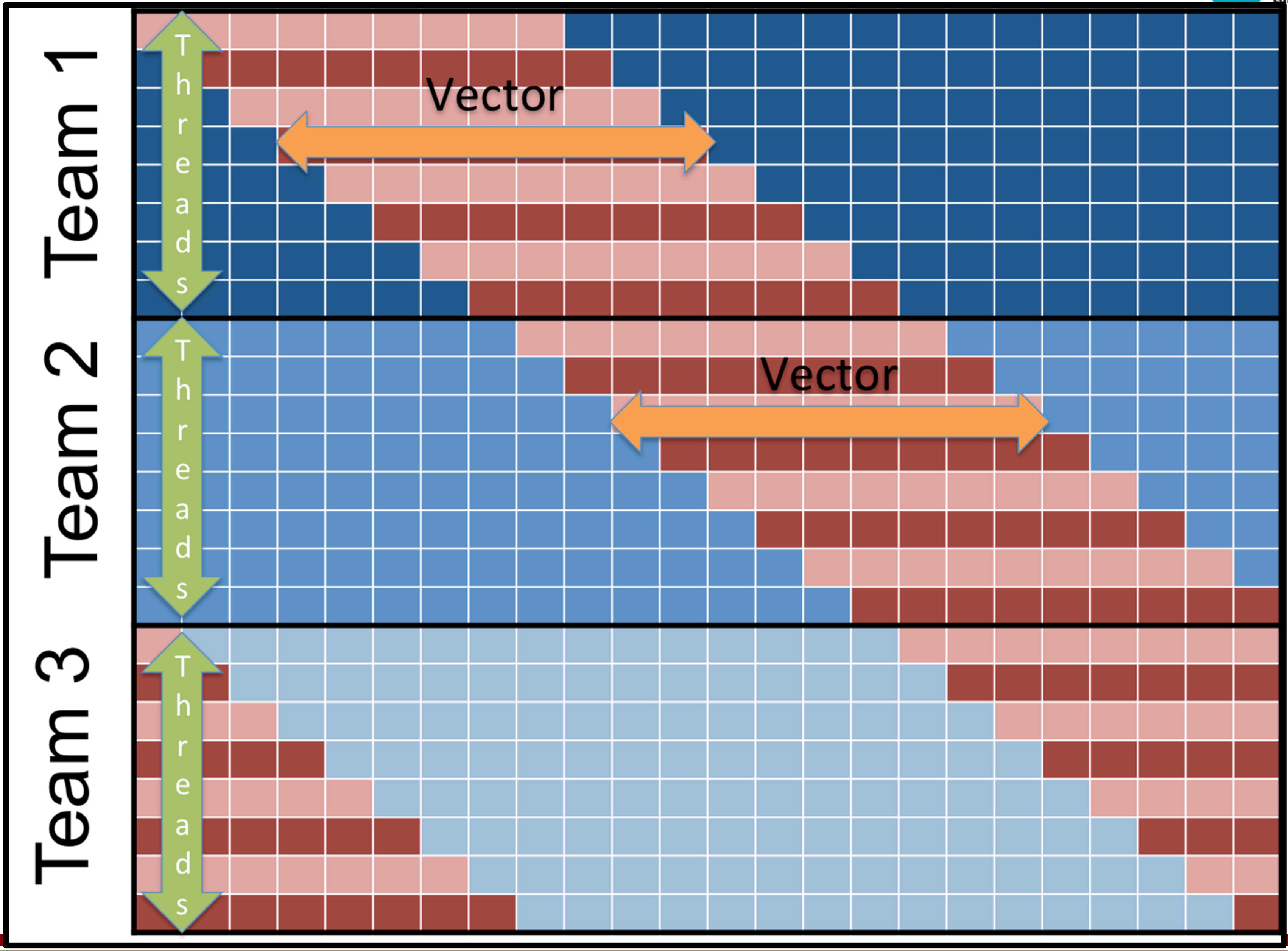
***LAMBDA Based Kokkos:***

```
parallel_for( N, KOKKOS_LAMBDA (const int& i) {
  a[i] += b[i];
});
```

## Under development: KokkosKernels

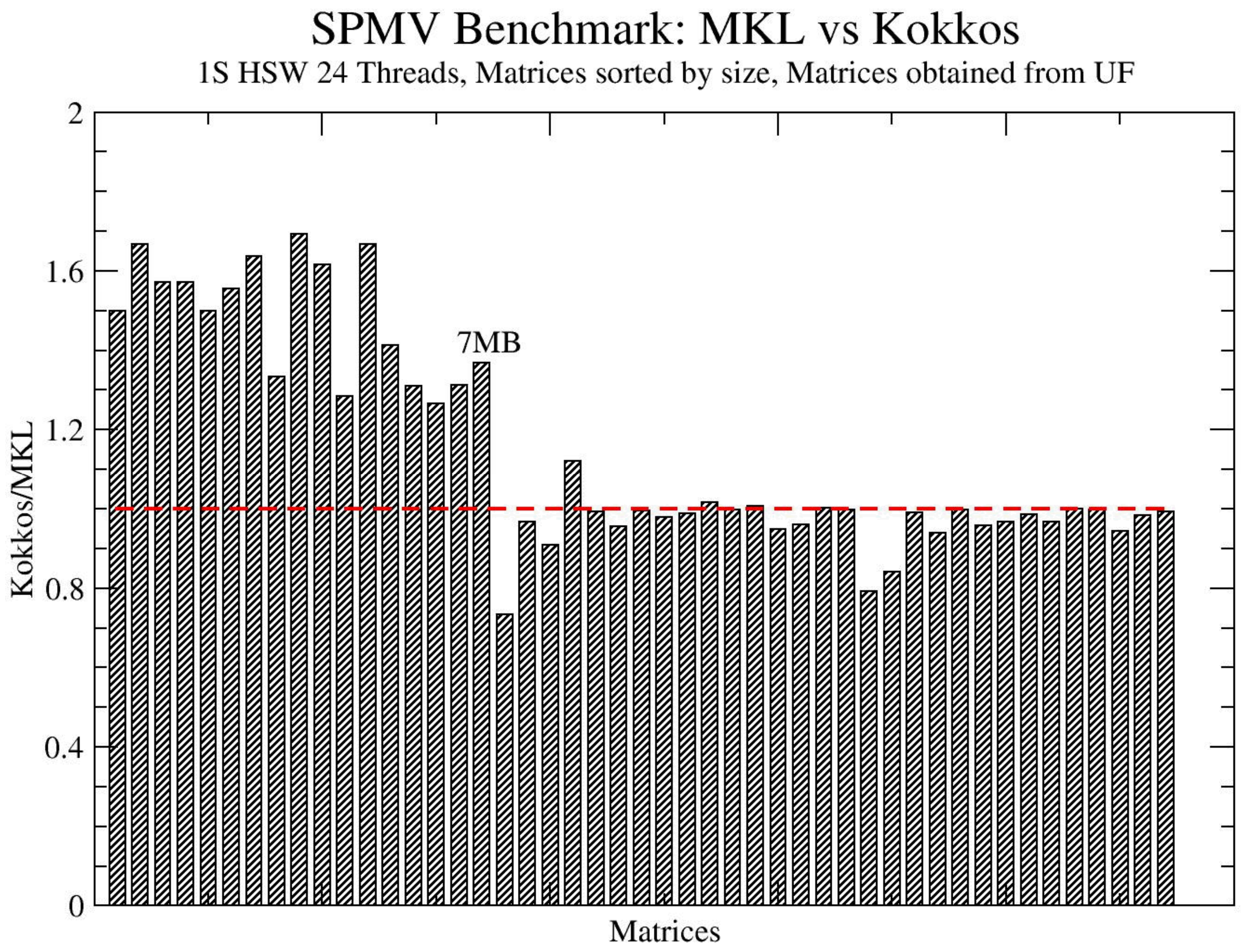
- Provide BLAS (1,2,3); Sparse; Graph and Tensor Kernels
- No required dependencies other than Kokkos
- Local kernels (no MPI)
- Hooks in TPLs such as MKL or CuBlas/CuSparse if applicable
- Provide kernels for all levels of hierarchical parallelism:
  - Global Kernels: use all execution resources available
  - Team Level Kernels: use a subset of threads for execution
  - Thread Level Kernels: utilize vectorization inside the kernel
  - Serial Kernels: provide elemental functions (OpenMP declare SIMD)
- Work started based on customer priorities; expect multi-year effort for broad coverage
- People: Many developers from Trilinos contribute
  - Consolidate node level reusable kernels previously distributed over multiple packages

## SPMV – Using Hierarchical Parallelism



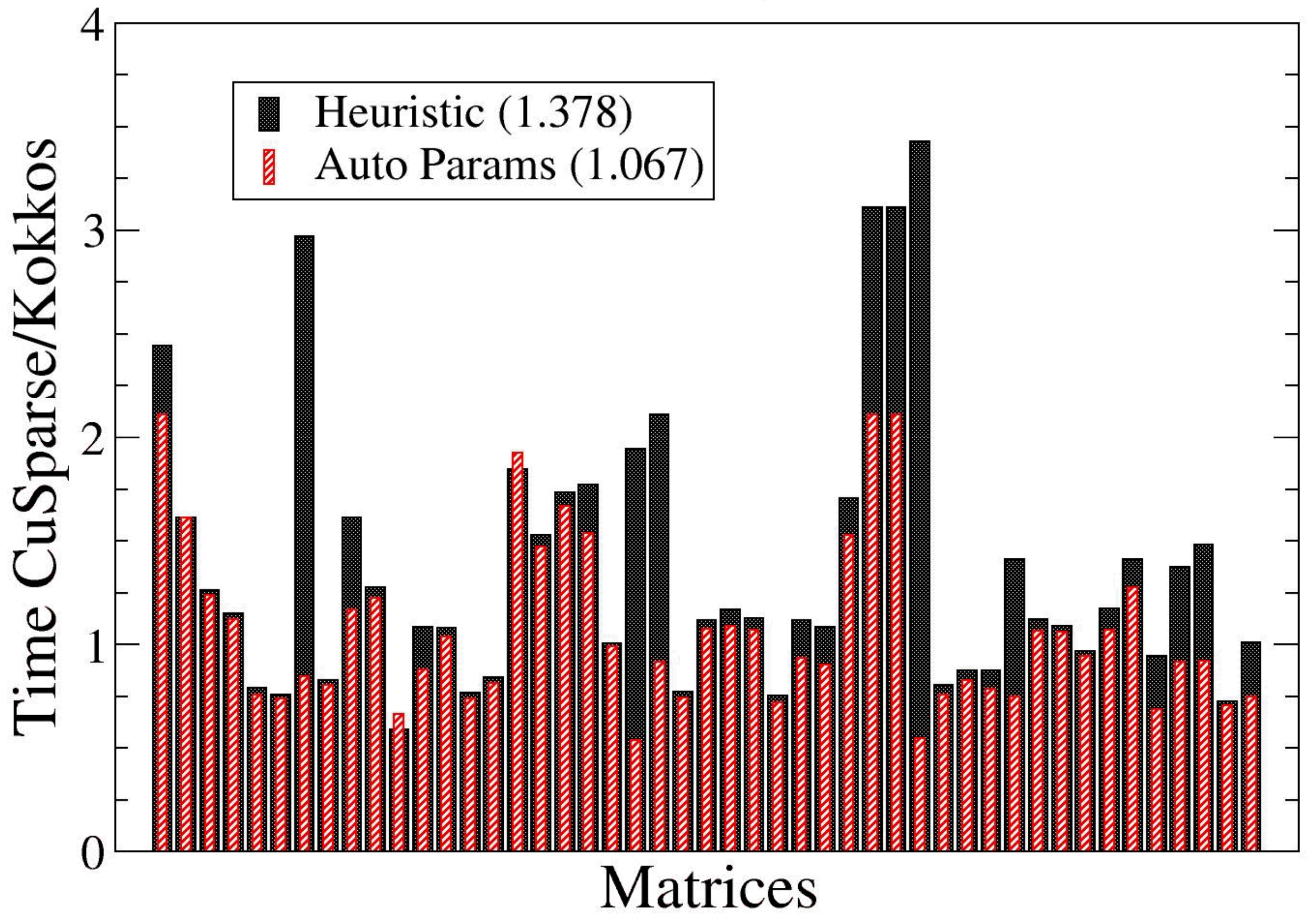
# SPMV – Using Hierarchical Parallelism

```
void spmv(Matrix A, Scalar alpha, XType x, Scalar beta, YType y) {
 int nnz_per_team = 2048;
 int conc = execution_space::concurrency();
 while((conc * nnz_per_team * 4> A.nnz())&&(nnz_per_team>256)) nnz_per_team/=2;

 int nnz_per_row = A.nnz()/A.numRows();
 int rows_per_team = (nnz_per_team+nnz_per_row-1)/nnz_per_row;
 int vector_length = GetVectorLength(A);
 const int nworkset = (y.dimension_0()+rows_per_team-1)/rows_per_team;

 parallel_for(TeamPolicy<Schedule<Dynamic> >(nworkset, AUTO(),vector_length),
  KOKKOS_LAMBDA (const TeamPolicy<>::member_type& team)  {
  const int startRow = team.league_rank() * rows_per_team;
  const int endRow = startRow + rows_per_team < A.numRows() ?
                   startRow + rows_per_team : A.numRows()

  parallel_for(TeamThreadRange(team,startRow,endRow), [&] (const int& loop) {
   const SparseRowViewConst<MatrixType,SizeType> row = A.template rowConst<SizeType>(iRow);
   const int row_length = row.length;
   Scalar sum = 0;

   parallel_reduce(ThreadVectorRange(team,row_length), [&] (const int& iEntry, Scalar& lsum) {
    const Scalar val = conjugate ?
            ATV::conj (row.value(iEntry)) :
            row.value(iEntry);
    lsum += val * x(row.colidx(iEntry));
   },sum);

   single(PerThread(team), [&] () {
    sum *= alpha;
    y(iRow) = beta * y(iRow) + sum;
   });
  });
}
```


SPMV Benchmark: MKL vs Kokkos
1S HSW 24 Threads, Matrices sorted by size, Matrices obtained from UF
Kokkos/MKL
Matrices
7MB
0
0.4
0.8
1.2
1.6
2

# SPMV Benchmark: CuSparse vs Kokkos

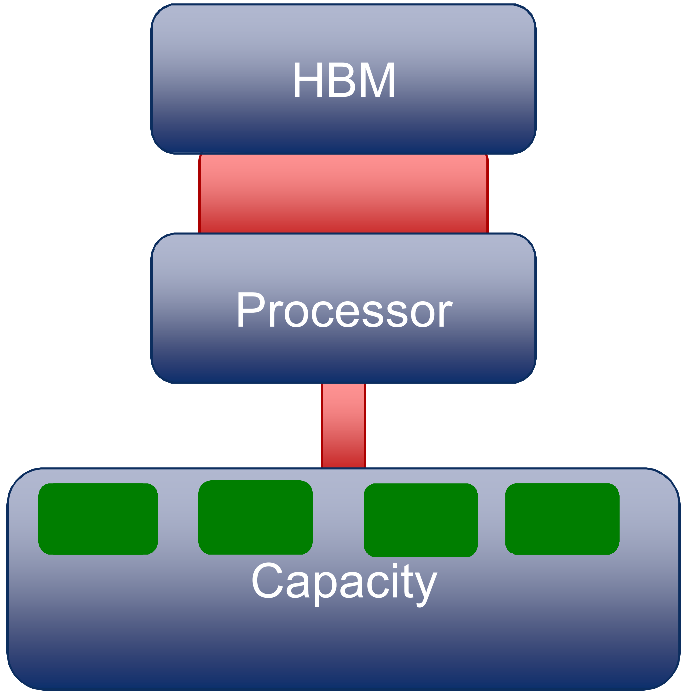
K40c Cuda 7.5; Matrices sorted by size; Matrices from UF.



# High Bandwidth Memory

- Main Problem: How to decide who can put things in scarce HBM
- Strategy One: Stage in individual linear systems temporarily
  - Most of our Apps solve multiple linear systems at the same time
  - Aggregate Memory footprint > HBM, but individual linear system < HBM
  - Can be supported by TPetra today: Keep copies of all systems in capacity memory, create temporary copies in HBM for individual solves
- Strategy Two: Domain Decomposition Solvers
  - Divide full problem into subdomains
  - Develop solvers which can work on individual subdomains with enough data reuse to amortize data transfer
  - Copy in one subdomain at a time
- Advantage: Relatively straight forward, No persistent HBM usage
  - No need for inter-package arbitration on HBM usage quotas

# High Bandwidth Memory



**Cost Estimate (Bandwidth Bound):**

*Run From Capacity Memory*

Time = $N_{iter}$ * Size / $BW_{Capacity}$

*Run From HBM*

Time = $N_{iter}$ * Size / $BW_{HBM}$ + Size / $BW_{Capacity}$

*Expect*

$BW_{HBM}/BW_{Capacity}$ ~ 5-20

***Question:*** Generally need higher parallelism to achieve $BW_{HBM}$ vs $BW_{Capacity}$
=> What about Direct Solvers?

# The Way Forward

- Stabilize Kokkos Capabilities
  - Support tasking on all platforms
  - Make sure compilers optimize through layers
  - Harden KNL support for High Bandwidth Memory
- Broaden Implementation Coverage for Kokkos Kernels
- Support Production Teams in Adoption
- Develop more Documentation
- Extend profiling tools to help with transition

| | |
|---|---|
| www.github.com/kokkos/kokkos: | Kokkos Core Repository |
| www.github.com/kokkos/kokkos-tutorials: | Kokkos Tutorial Material |
| www.github.com/kokkos/kokkos-tools: | Kokkos Profiling Tools |
| www.github.com/trilinos/Trilinos: | Trilinos Repository |

Sandia National Laboratories

*Exceptional service in the national interest*