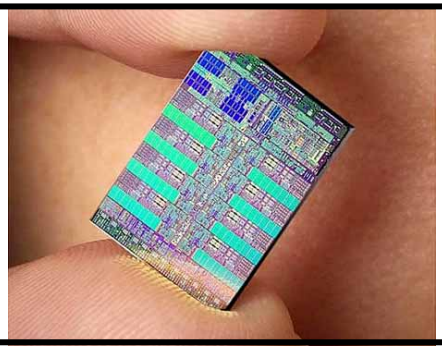
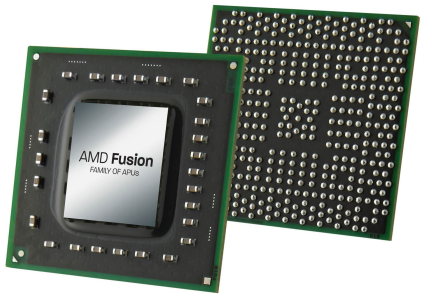
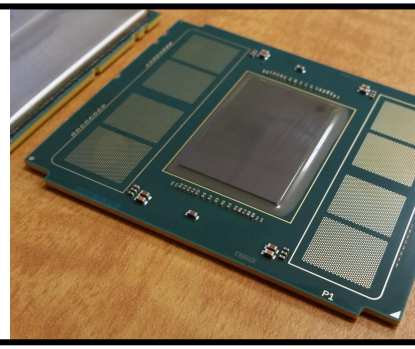
Exceptional service in the national interest

Sandia National Laboratories



KokkosKernels Overview

Nathan Ellingwood,

S. Rajamanickam, V. Dang, K. Kim, M. Deveci, C.R. Trott

srajama@sandia.gov

Center for Computing Research

Sandia National Laboratories, NM

Unclassified Unlimited Release

SAND2018-X

U.S. DEPARTMENT OF ENERGY

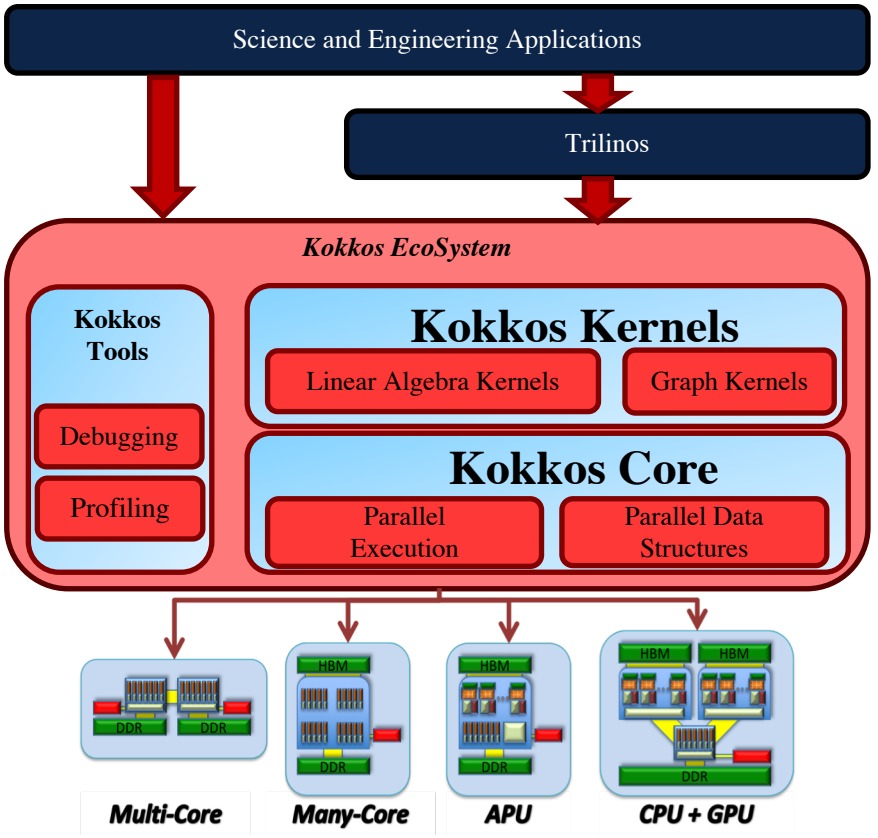
NNSA National Nuclear Security Administration

Sandia National Laboratories is a multimission 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-NA-0003525.

Goal of the Project

KokkosKernels provides math kernels for dense and sparse linear algebra as well as graph computations. It has multiple aims:

- Portable BLAS, Sparse and Graph kernels
- Generic implementations for various scalar types and data layouts
- Access to major vendor optimized math libraries.
- Expand the scope of BLAS to hierarchical implementations.



Capabilities: BLAS

BLAS-1 functions are available as multi-vector variants.

– abs(y,x)	y[i] = |x[i]|
– axpy(alpha,x,y)	y[i] += alpha * x[i]
– axpby(alpha,x,beta,y)	y[i] = beta * y + alpha * x[i]
– dot(x,y)	dot = SUM_i (x[i] * y[i])
– fill(x,alpha)	x[i] = alpha
– mult(gamma,y,alpha,A,x)	y[i] = gamma * y[i] + alpha * A[i] * x[i]
– nrm1(x)	nrm1 = SUM_i(|x[i]|)
– nrm2(x)	nrm2 = sqrt (SUM_i(|x[i]| * |x[i]|))
– nrm2w(x,w)	nrm2w = sqrt (SUM_i((|x[i]|/|w[i]|)^2))
– nrminf(x)	nrminf = MAX_i(|x[i]|)
– scal(y,alpha,x)	y[i] = alpha * x[i]
– sum(x)	sum = SUM_i(x[i])
– update(a,x,b,y,g,z)	y[i] = g * y[i] + b * y[i] + a * x[i]
– gemv(t,alph,A,x,bet,y)	y[i] = bet*y[i] + alph*SUM_j(A[i,j]*x[j])
– gemm(tA,tB,alph,A,B,bet,C)	C[i,j]=bet*C[i,j]+alph*SUM_k(A[i,k]*B[k,j])

Capabilities II

Sandia National Laboratories

Sparse

- CSR-Sparse Matrix Class providing fundamental capabilities
- SPMV: Sparse Matrix Vector Multiply
- SpGEMM: Sparse Matrix Matrix Multiply; separate symbolic and numeric phase
- GS: Gauss-Seidel Method using graph coloring: symbolic, numeric, solve phases

Batched BLAS

- DGEMM
- DTRSM
- DGETRF

Graph

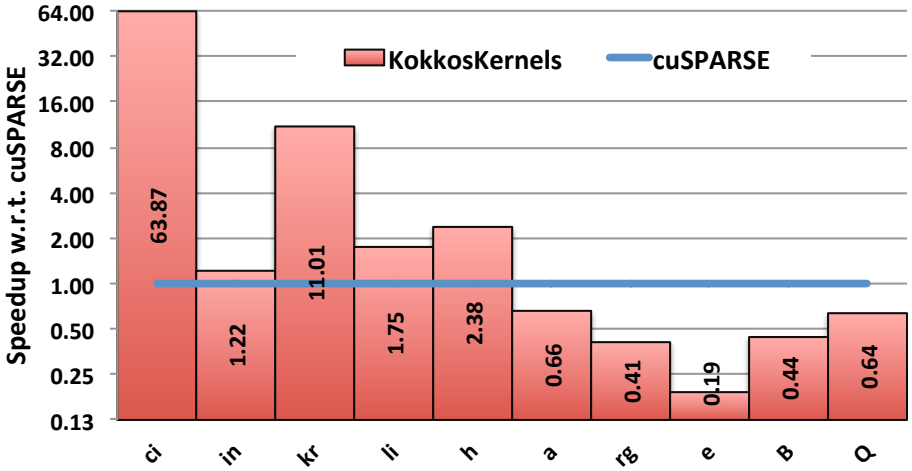
- Distance-1 and Distance-2 graph coloring
- Triangle enumeration for graph analytics
  - Using SpGEMM + Visitor Pattern: can be used to represent large problems

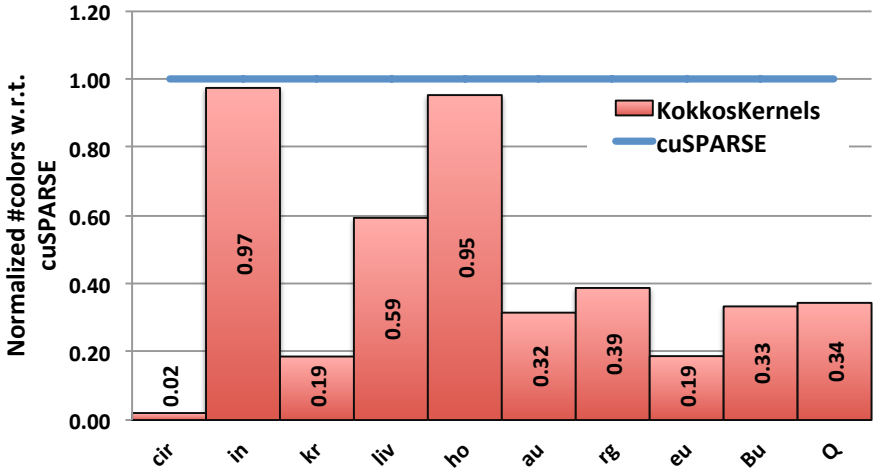
Kernel 1: Graph Coloring and Multi-threaded Gauss-Seidel

Sandia National Laboratories

- Goal: Identify independent rows that can be processed in parallel for a parallel preconditioner.
  - Distance-1 graph coloring problem.
- Other Approaches possible for Gauss-Seidel preconditioning:
  - Level-set based Gauss-Seidel (similar to a triangular solve)
    - Dynamic parallelism is difficult on GPUs
  - Block Gauss-Seidel similar to MPI
    - Can become (block) Jacobi preconditioner as #threads increase as in GPUs
- Distance-1 Graph Coloring
  - We use the Gebremedhin and Manne greedy graph coloring approach with a twist (Edge-based coloring, bit arrays for forbidden color, and edge filtering techniques for better GPU performance).
  - Details of the algorithm are in the paper *Parallel Graph Coloring for Manycore Architectures, M. Deveci, E. Boman, K. Devine and S. Rajamanickam*, IPDPS 2016.

Graph Coloring and Multi-threaded Gauss-Seidel





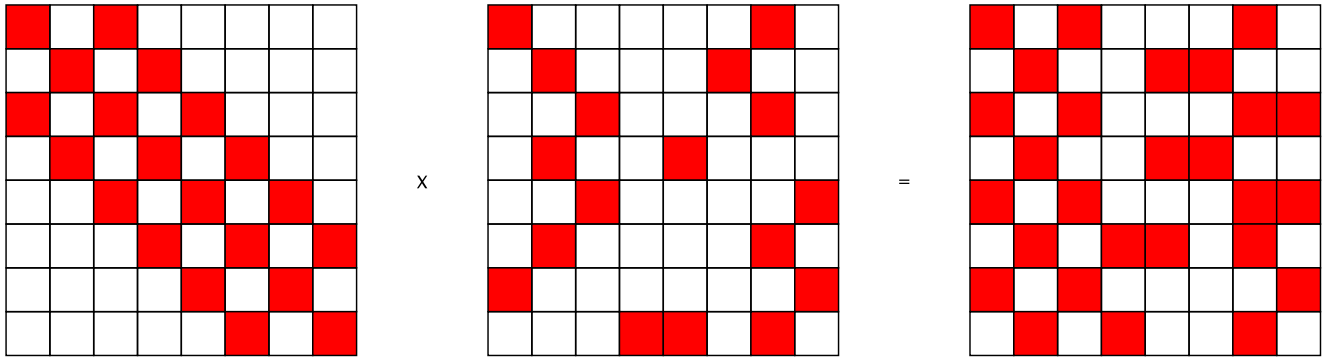
- **Performance**: Better quality (4x on average) and run time (1.5x speedup) w.r.t cuSPARSE.
- Performance portable implementation allows better results on the KNL as well.
- Enables parallelization of preconditioners: Gauss Seidel: 136x on K20 GPUs w.r.t. serial Sandy Bridge (significant for a triangular solve like kernel)
- Application Integration
  - Integrated in Trilinos preconditioners (IFPACK2 package)
  - Evaluated in the Exascale Computing Project Wind Energy application Nalu

Current work on Graph Coloring and Preconditioners

Sandia National Laboratories

- **Problem 1**: Distance-2 graph coloring for aggregation in multigrid preconditioners
  - Need a performance portable distance-2 graph coloring with ability to run well on the GPUs
    - Critical for the setup phase of MueLU multigrid library to be able to run on the GPUs
  - Forbidden arrays are larger (GPUs), edge-filtering is complicated, and edge-based algorithms more complex than distance-1 coloring
  - *First cut: Will McKlendon's poster from the minisymposterium.*
- **Problem 2:** Deterministic distance-1and distance-2 graph coloring for multigrid preconditioners and Gauss-Seidel preconditioners
  - Non-determinism due to multithreaded coloring is unacceptable in certain high consequence scenarios. Example: gold standard output maintained from past real (physical) experiments need to be met for qualification, verification and validation.
  - Aggregation quality of the preconditioner heavily depends on the coloring result
  - Better debugging
  - *First cut: Luc Berger-Vergiat's poste from the minisymposterium.*

Kernel 2: Sparse Matrix – Sparse Matrix Multiply



- SpGEMM: fundamental block for
  - Setup phase of algebraic multigrid methods that use RxAxP
  - Various graph analytics problems: clustering, betweenness centrality…
- More complex than most of the other sparse blas and graph problems:
  - Extra irregularity: nnz of C is unknown beforehand.
    - Introduced a 2-phase approach for reuse of symbolic structure within a non-linear iteration with the same pattern
  - Requirement of thread private data structures
    - More complicated especially on GPUs especially when the kernel needs to be used for regular (multigrid) problems and irregular (graph analytics) problems

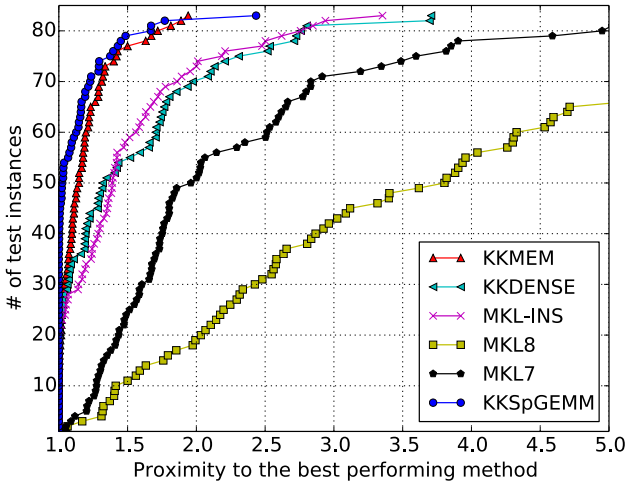
Sparse Matrix – Sparse Matrix Multiply

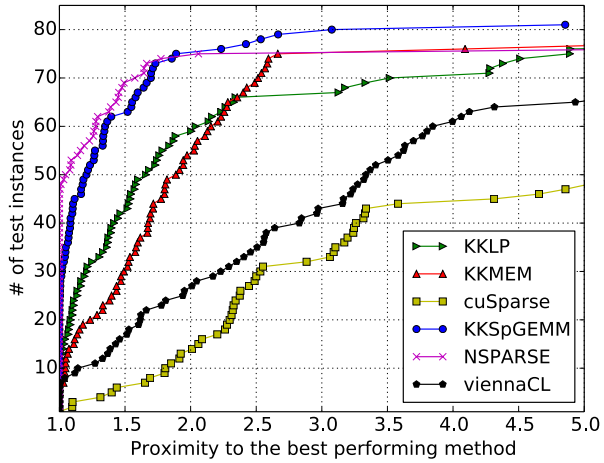
Sandia National Laboratories

- Kokkos based implementation allows performance-portability
  - In reality slightly different algorithmic choices do well on KNLs and GPUs
  - Kokkos allows us to to run the different algorithms on all the architectures and explore the performance trade-offs
- Several other implementations exist
  - Intel MKL, Thrust, AMGx, NSPARSE, ViennaCL, bhSparse
- Key Performance Differentiators
  - Hashmap accumulators – Two choices for accumulators (sparse, dense)
  - Compression technique to reduce the number of insertions
  - Hierarchical parallelism for better load balancing using Kokkos
- Algorithmic details available in the following papers
  - *Performance-Portable Sparse Matrix-Matrix Multiplication for Many-Core Architectures, M. Deveci, C. Trott, S. Rajamanickam, IPDPSW AsHES 2017*. (**KKMEM**)
  - Multi-threaded Sparse Matrix-Matrix Multiplication for Many-Core and GPU Architectures, *M. Deveci, C. Trott, S. Rajamanickam, ArXiV 2018. (**KKSpGEMM** – Meta algorithm with better hierarchical parallelism)*

Sparse Matrix – Sparse Matrix Multiply

- Integration:
  - Integrated into the Tpetra package of Trilinos and used by the multigrid solver MueLU within a distributed memory sparse matrix-matrix multiply
  - Integration with several Exascale Computing Project applications (ExaWind, EMPIRE)
- Performance Profile – 82 different instances of SpGEMM
  - Quicker and higher better. KNL DDR mode (left): KK-SpGEMM is the best for ~50 test instances, within 1.5x of the best instance for all but 3 instances. GPU (right): KKSpGEMM and NSPARSE are the two best methods.
  - See the paper for matrices and detailed analysis of the results.

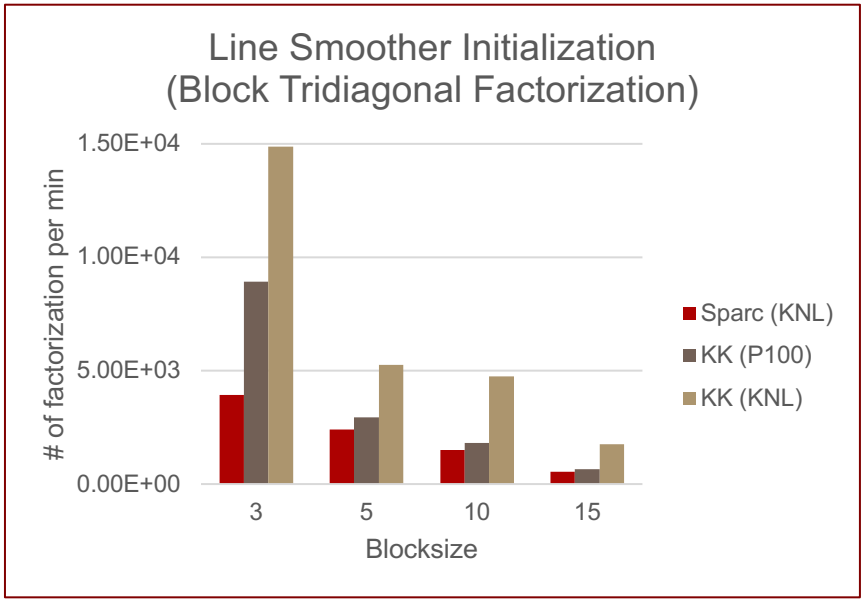


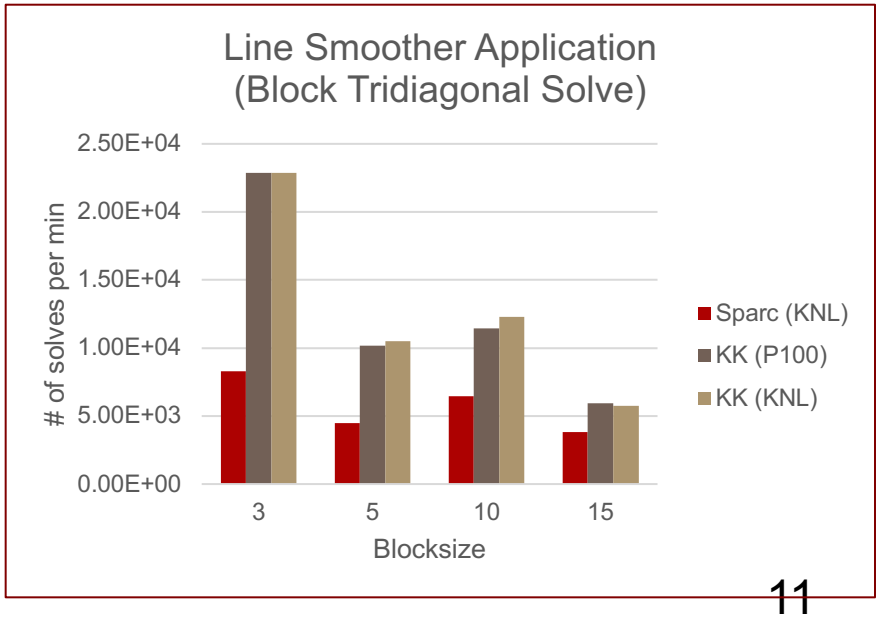


Kernel 3: Batched BLAS

SPARC: Sandia in-house finite volume solver for reactive fluid problems

- A block sparse systems arises from coupled multi-physics problems
- To solve the block systems, line preconditioner is often constructed, which requires block tridiagonal factorization and solves
- Typical block sizes are 3, 5, 10 and 15, which is common in scientific applications
- Compact batched BLAS is proposed to efficiently vectorize small dense problems
- Integration: ECP ATDM Application SPARC
- See the paper *Designing Vector-Friendly Compact BLAS and LAPACK Kernels, K. Kim, T. Costa, M. Deveci, A. Bradley, S. Hammond, M. Guney, S. Knepper, S. Story, SC 2017* for the KNL performance analysis.





Conclusion

- Kokkos Kernels is available publicly both as part of Trilinos and as part of the Kokkos ecosystem
- Interfaces to Intel, NVIDIA and other vendor provided kernels available in order to leverage their high-performance libraries
- Several new kernels are being added as needed by the applications
  - E.g.: Distance-2 Coloring, Deterministic coloring, RCM ordering, tensor contractions for scientific computing applications
- Download at https://github.com/kokkos/kokkos-kernels

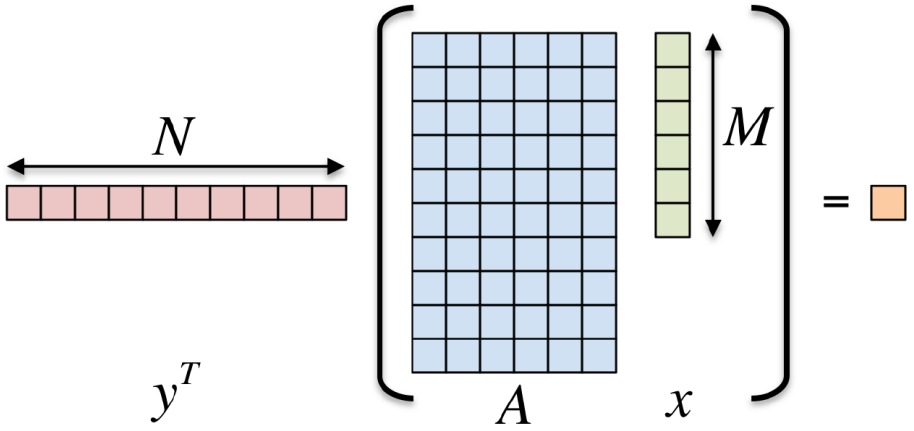
Tutorial Exercises

Calling KokkosKernels from Kokkos

Vinh Q Dang

KokkosKernels Interface: Inner Product

- Goal: re-implement the inner product exercise using
  1. KokkosKernels BLAS functions
  2. KokkosKernels team-based BLAS function
- Details: <**y**,**A*****x**>
  - **y** is *N*x1, **A** is *N*x*M,* **x** is *M*x1
  - Look for comments labeled with "EXERCISE"
  - Compile and run on OpenMP, CUDA backends
  - Compare performance of these two implementations
  - Try LayoutLeft and LayoutRight



KokkosKernels Interface: Inner Product

Sandia National Laboratories

- Details:
  1. KokkosKernels BLAS functions
     - Convert from hierarchical parallel execution to using BLAS functions
     - **tmp** = **A*x** (extra view, holds **gemv** results)
     - result = <**y**,**tmp**>
  2. KokkosKernels team-based BLAS function
     - Same as hierarchical parallel execution
     - Call team-based **dot** within each team to perform `<A[teamId,:],x>`

- Hint: KokkosKernels Functions

```
result = KokkosBlas::dot(x,y)
```

performs `result = SUM_i(y[i]*x[i])`

```
KokkosBlas::gemv("N",alpha,A,x,beta,y)
```

performs matrix-vector multiplication

```
y[i] = beta*y[i] +
alpha*SUM_j(A[i,j]*x[j])
```

```
KokkosBlas::Experimental::dot(teamId,x,y)
```

performs `result = SUM_i(y[i]*x[i])` within each thread team

KokkosKernels Interface: Conjugate Gradient Solver

Sandia National Laboratories

- Goal: implement conjugate gradient solver for square, symmetric, positive-definite sparse matrix
- Details: **A*x=b**
  - **b** is *Nx*1
  - **A** is *NxN* symmetric, positive-definite sparse matrix
  - **x** is *Nx*1
  - Look for comments labeled with "EXERCISE"
  - Use KokkosKernels BLAS and KokkosKernels Sparse BLAS

- Algorithm:

$$\mathbf{r_0} = \mathbf{b} - \mathbf{A} * \mathbf{x_0}$$

$$\mathbf{p_0} = \mathbf{r_0}$$

$$k = 0$$

while $\|\mathbf{r}_k\| > \varepsilon$ and $k<N$

$$\alpha = \frac{\mathbf{r}_k^T * \mathbf{r}_k}{\mathbf{p}_k^T * \mathbf{A} * \mathbf{p}_k}$$

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha * \mathbf{p}_k$$

$$\mathbf{r}_{k+1} = \mathbf{r}_k - \alpha * \mathbf{A} * \mathbf{p}_k$$

$$\beta = \frac{\mathbf{r}_{k+1}^T * \mathbf{r}_{k+1}}{\mathbf{r}_k^T * \mathbf{r}_k}$$

$$\mathbf{p}_{k+1} = \mathbf{r}_{k+1} + \beta * \mathbf{p}_k$$

$$k = k + 1$$

KokkosKernels Interface: Conjugate Gradient Solver

Sandia National Laboratories

- Details:
  - Sparse matrix generation is provided
  - Compile and run on OpenMP, CUDA backends
  - Vary problem size: -N #
  - Compare performance of CPU vs GPU

- Hint: KokkosKernels Functions

```
result = KokkosBlas::dot(x,y)
```

performs `result = SUM_i(y[i]*x[i])`

```
KokkosBlas::axpy(alpha,x,y)
```

performs `y[i] = y[i] + alpha*x[i]`

```
KokkosBlas::axpby(alpha,x,beta,y)
```

performs `y[i] = beta*y[i] + alpha*x[i]`

```
KokkosSparse::spmv("N",alpha,A,x,beta,y)
```

performs sparse matrix-vector multiplication

```
y[i] = beta*y[i] +
alpha*SUM_j(A[i,j]*x[j])
```

Sandia
National
Laboratories

Exceptional service in the national interest

http://www.github.com/kokkos