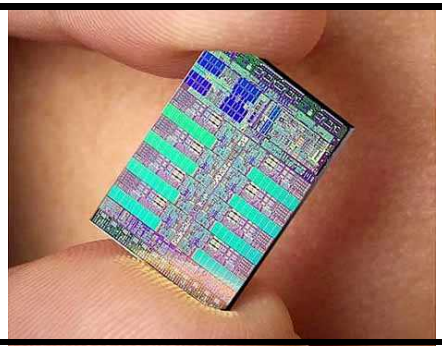
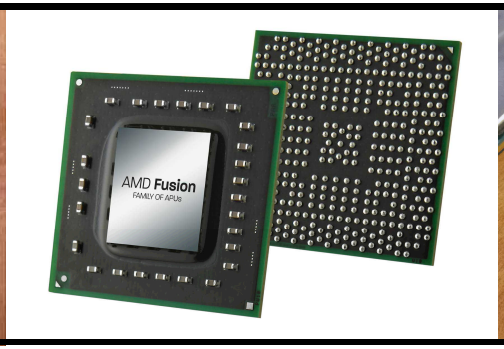
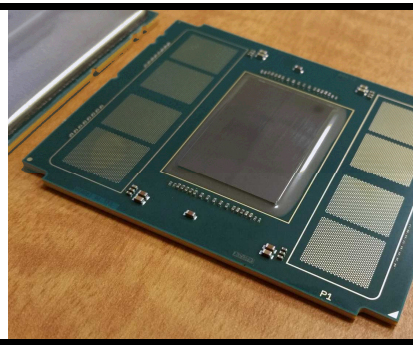
SAND2018-1188C



KokkosKernels Overview

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

crtrott@sandia.gov

Center for Computing Research

Sandia National Laboratories, NM

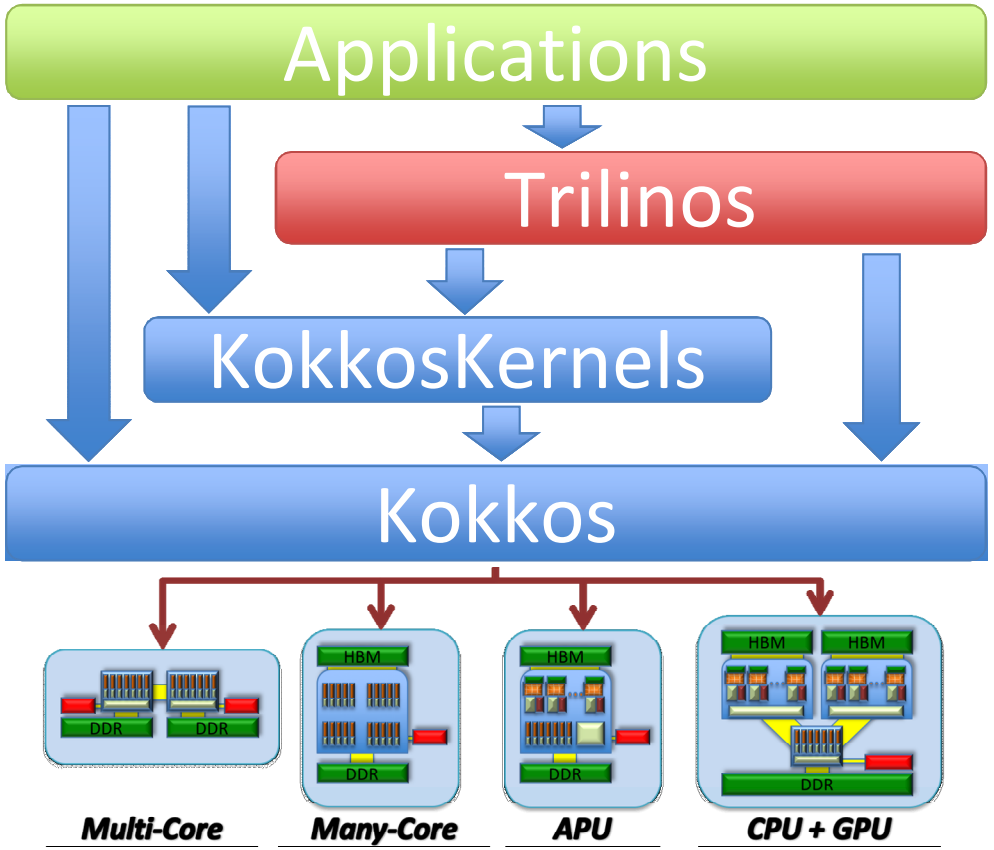
SAND2017-XXXX C

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

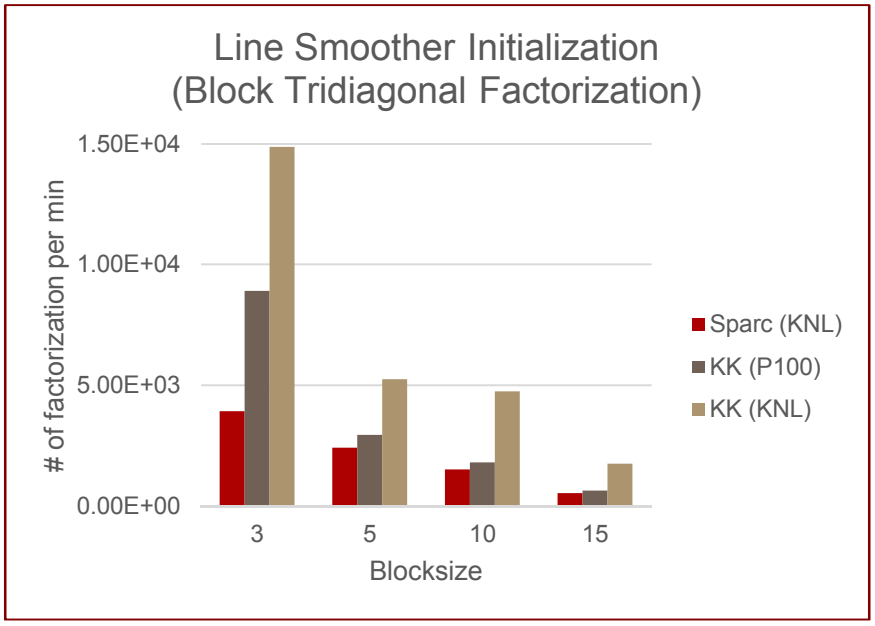
Graph

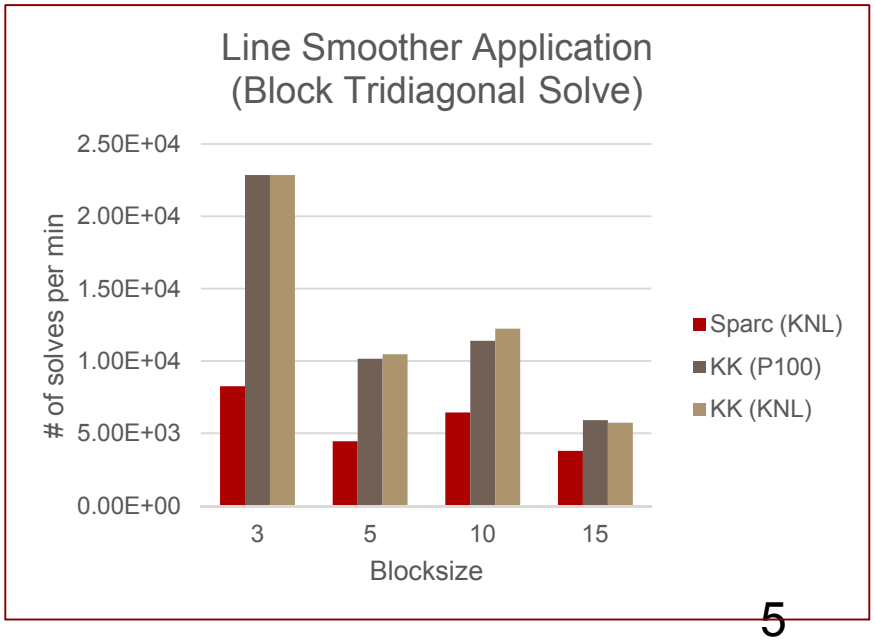
- Distance-1 and Distance-2 graph coloring
- Triangle enumeration for graph analytics
  - Using SpGEMM + Visitor Pattern: can be used to represent large problems

Example: 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 related to scientific applications
- Compact batched BLAS is proposed to efficiently vectorize small dense problems



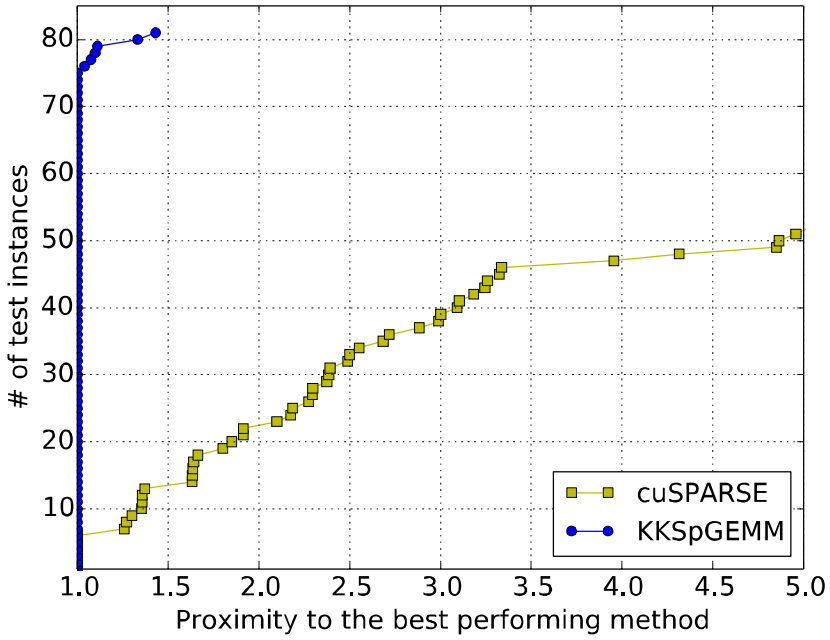


Sparse Matrix – Sparse Matrix Multiply

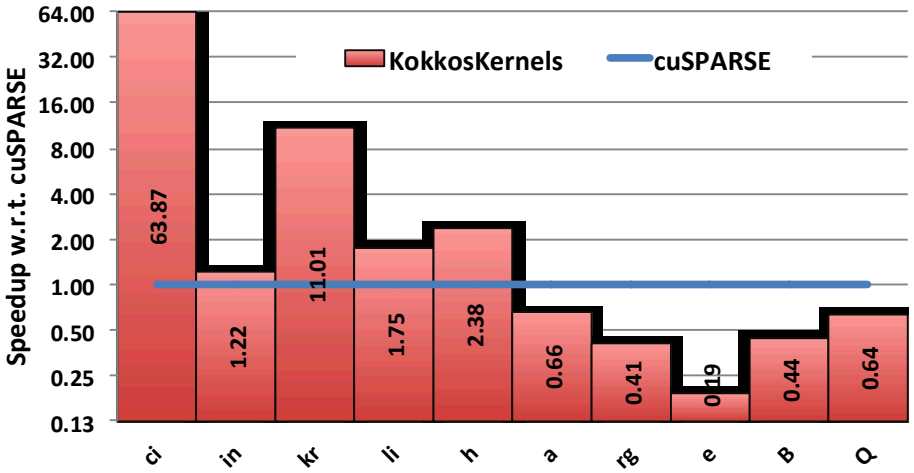
- The most expensive part of the multigrid setup
- It is also in GraphBLAS standard, as it can be used to represent various graph analytics problems: Triangle counting, Jaccard, Clustering
- A portable SpGEMM method: KKSpGEMM
  - Separates symbolic and numeric computations
  - Compression to reduce memory use and #ops

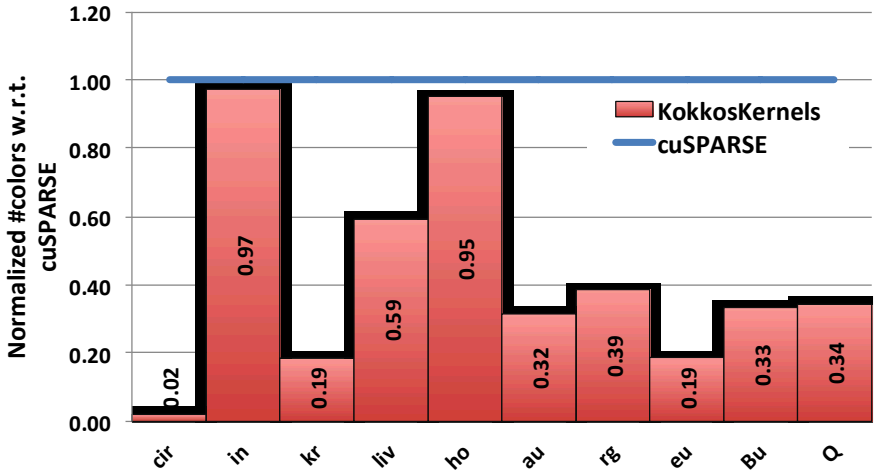
Geometric mean of GFLOPs for 81 instances:

P100	Power8	KNL
5.98	5.23	3.90



Graph Coloring and Multi-threaded Gauss-Seidel





- Goal: Identify independent data that can be processed in parallel.
- Performance: Better quality (4x on average) and run time (1.5x speedup) w.r.t cuSPARSE.
- Enables parallelization of preconditioners: Gauss Seidel: 136x on K20 GPUs w.r.t. serial SNB

Future Plans: Micro BLAS

Sandia National Laboratories

- Hierarchical Hardware requires hierarchy of function support
- Idea: Provide BLAS / SparseBLAS interface with hardware handles
- Example use-case: each CUDA block or KNL tile runs its own independent CG-Solve
- In contrast to batched BLAS no lockstep execution

Device Level

- Kernel uses whole GPU or all threads in the process
- Equivalent to current BLAS libraries

Team Level

- Kernel uses a CUDA block or all threads sharing a common L2 cache
- Utilization of local scratch important

Thread Level

- Kernel uses a single warp or thread
- Vectorization can be exploited
- Extremely fast synchronization possible

Serial Level

- Serial implementations
- Potentially as elemental functions allowing outer level vectorization

http://www.github.com/kokkos