Batched Linear Algebra in Kokkos Kernels

Presented by

Siva Rajamanickam, Luc Berger-Vergiat, Vinh Dang, Nathan Ellingwood, Kyungjoo Kim, Will McLendon, Christian Trott, Jeremiah Wilke

SAND2019-XXXXXX
Kokkos Ecosystem for Performance Portability

**Kokkos Ecosystem**

- **Kokkos Core**: parallel patterns and data structures; supports several execution and memory spaces
- **Kokkos Kernels**: performance portable BLAS; sparse, dense and graph algorithms
- **Kokkos Tools**: debugging and profiling support

Write-once using Kokkos for portable performance on different architectures

Kokkos Ecosystem addresses complexity of supporting numerous many/multi-core architectures that are central to DOE HPC enterprise
Focus of Kokkos Kernels

Deliver **portable** sparse/dense linear algebra and graph kernels
- These are the kernels that are in 80% of time for most applications
- Key problems: Kernels might need different algorithms/implementations to get the best performance
- Ninja programming needs in addition to Kokkos
- Users of the kernels do not need to be ninja programmers
- **Focus on performance of the kernels on all the platforms of interest to DOE**

Deliver **robust software ecosystem** for other software technology projects and applications
- Production software capabilities that give high performance, portable and turn-key
- Tested on number of configurations nightly (architectures, compilers, debug/optimized, programming model backend, complex/real, ordinal types…)
- Larger release/integration testing with Trilinos and applications
- Kokkos Support, github issues, tutorials, hackathons, user group meetings (planned)

**Kokkos Kernels delivers portable, high-performance kernels in a robust software ecosystem**
Hierarchical Parallelism in Kokkos

Three levels of Parallelism in Kokkos

- Device, Team and Vector Level Parallelism
- Device level maps to the accelerator or the entire device
- Team level Parallelism maps to the thread block or teams of threads (or the Xdim)
- Vector level parallelism maps to compiler directives (or the Y dim)
- Users write three level parallel kernels or call BLAS/LAPACK within some level of parallelism

Alternative way to represent hierarchical parallelism

- See MS 332: Thursday 4.10 PM session “A Performance Portable SIMD Scalar Type for Effective Vectorization Across Heterogeneous Architectures”, Damodar Sahasrabudhe
- For this talk assume the three levels of parallelism above

It is key to represent the parallelism in hierarchical, and portable manner. “Batched” BLAS exists within these hierarchical parallel framework.
“Batched” BLAS in Kokkos Kernels

Common hierarchical-parallel pattern in Kokkos based applications
- `parallel_for` over some entities (rows, elements, vertices, particles)
- BLAS/LAPACK call is team-level / serial call within the `parallel_for` with vectorization
- The device level loop is much larger than the BLAS call underneath. It is **not** one BLAS call, it is a set of BLAS calls and other operations together in a larger kernel.
- “Standard” Batched BLAS as proposed by the reference implementation introduces synchronization and data movement for these use cases
- Kokkos Kernels provides BLAS / LAPACK functionality at the team-level / serial level

Some exceptions
- Machine learning use cases where a layer could be fully expressed and utilize the device level concurrency
- “Standard” batched BLAS interface provides the required functionality here
- Kokkos Kernels can provide interfaces to vendor kernels and libraries like MAGMA
Kokkos Kernels interface to BLAS/LAPACK – Other considerations

Compact Layout

- Repack the matrices to be stored in compact layout for small matrix sizes in the batch
- Use traditional layouts for large matrix sizes or based on architectures
- Purely a memory access issue
  - Vectorized reads/writes
  - Coalesced memory access on a GPU
- See work by Kokkos Kernels and Intel MKL team (SC’17) for the usefulness of compact layout
  - Also implemented in Intel MKL

Usage of different memory layouts based on problem sizes and architecture needs is critical for performance
Primary Application Users

SPARC: state-of-the-art hypersonic unsteady hybrid structured/unstructured finite volume CFD code
- High performance line solvers; batched BLAS on CPUs and GPUs
- Actively researching scalable multigrid methods for hypersonic regime
- Time-stepping methods, Uncertainty quantification methods
- Performance-portable programming models

EMPIRE: next-gen unstructured-mesh FEM PIC/multifluid plasma simulation code
- Scalable solvers for electrostatic and electromagnetic systems for Trinity and Sierra architectures
- Thread-scalable, performance-portable, on-node linear algebra kernels to support multigrid methods
- Performance-portable programming models
- Non-linear solvers, discretization, and automatic differentiation approaches

Exawind: next-gen wind simulation code
- Scalable solvers for Trinity and Sierra architectures
- Thread-scalable, performance-portable, on-node linear algebra kernels to support multigrid methods
- Performance-portable programming models

Kokkos Kernels integrated into ECP applications and used with Kokkos based operators in the applications calling Kokkos Kernels.
Application Use Case: Tridiagonal solvers

- Application characteristics
  - One dimension of the mesh more important than the others when preconditioning
  - Multiple degrees of freedom per element gives rise to tiny blocks

- Block Jacobi preconditioner where each block is a Tridiagonal matrix
- Every scalar in the tridiagonal matrix is a small block matrix
  - Block sizes 5x5, 9x9, 15x15 etc
  - Typical number of diagonal blocks 512-1024
  - Key kernels needed DGEMM, LU, TRSM

Algorithm 1: Reference impl. TriLU

1. for $T$ in $\{T_0, T_1, \ldots, T_{m-1}\}$ do in parallel
2. for $r \leftarrow 0$ to $k - 2$ do
3.   $A^r := LU(A^r)$;
4.   $B^r := L^{-1}B^r$;
5.   $\hat{C}^r := \hat{C}U^{-1}$;
6.   $\hat{A}^{r+1} := \hat{C}^{r+1} - \hat{C}^r \hat{B}^r$;
7. end
8. $A^{k-1} := \{L \cdot U\};$
9. end

It is important to define BLAS and LAPACK kernels within the parallel regions
Developing line solvers for SPARC

- Develop Compact Batch Kernels needed by BLAS
- Different from community developed standards for Batched BLAS at the device level
- Proposed as part of the community standard and being developed as part of multiple implementations
- Optimized, vectorized implementation in Kokkos Kernels for Intel CPU, KNL and GPU platforms
- New implementations for Astra platform developed.
- Collaboration with Intel to develop assembly level kernels into Intel MKL (released as part of MKL 2018)
- Developed line solvers based on compact kernels and integrated compact BLAS based preconditioners in SPARC
  - Up to 2x speedup in total SPARC simulation time
  - Up to 8-20x speedup in solver time

- See even improved results from these plots MS-332 “Performance Portable SIMD Approach Implementing Block Line Solver For Coupled PDEs” – Kyungjoo Kim

Kokkos Kernels + Ifpack2 provides nearly perfect scalability for Intel KNL and GPU platforms

POC: Siva Rajamanickam (1465); Micah Howard (1515)
New features: Support Eigen Solvers within larger parallel kernels of an Chemistry application

Joint Work: Kokkos Kernels and Exascale Catalytic Chemistry (Kyungjoo Kim, Rob Fazle, Habib Najm)

The Computational Singular Perturbation method was developed (late 80s) for analysis and reduction of stiff chemical kinetic models for gas phase combustion

- Enables identification of cause-and-effect relations in chemical models
- Enables stiffness reduction and stable explicit large-time step time integration

CSP employs basis vectors to decouple fast and slow processes in the chemical system governing equations

- The eigenvectors of the Jacobian of the chemical source term are a good first order choice of CSP basis vectors

The dominant computational cost in CSP is the eigensolution for many typically-small (50x50) to mid-size (1000x1000) Jacobian matrices

- We are using Kokkos to enable fast eigensolutions for large batches of Jacobian matrices on advanced architectures

New Kokkos Kernels functionality for Householder transformations, Givens rotations, QR, Hessenberg, Schur decomposition, and Eigen Value decomposition

Slide Courtesy: Exascale Catalytic Chemistry project
Ongoing work

- Cholesky factorization,
- Forward and backward substitution,
- Computing the inverse of a factored matrix

Address Machine Learning use cases

- Convolution kernels
- Interface to “traditional” device level batched calls
- Lower level interface to Open libraries for machine learning