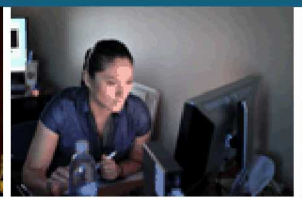
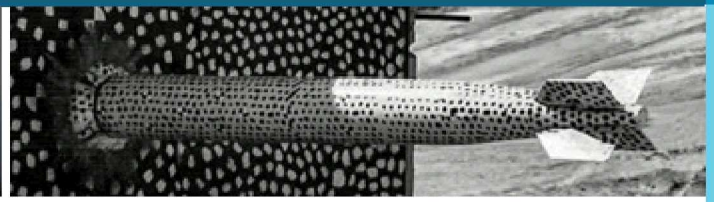
SAND2020-1452C

Kokkos Kernels





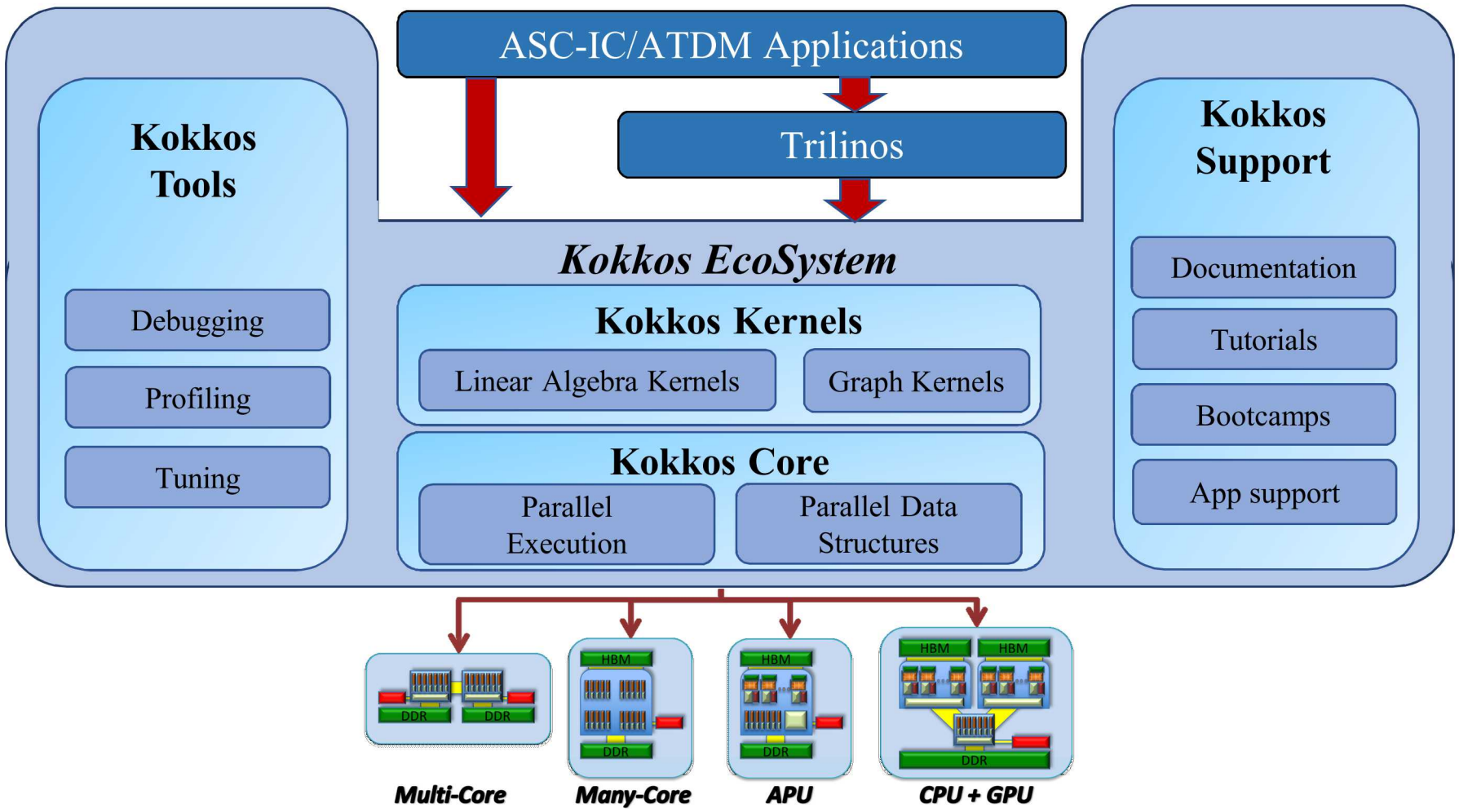
PRESENTED BY

Siva Rajamanickam, Seher Acer, Luc Berger-Vergiat, Vinh Dang, Nathan Ellingwood, Brian Kelley, Kyungjoo Kim, Christian Trott, Jeremiah Wilke

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-NA0003525.

SAND2020-XXXXXX

Kokkos Ecosystem for Performance Portability



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

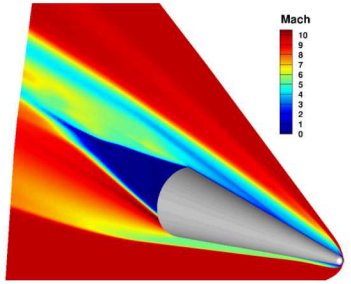
- These are the kernels that are in 80% of time for most applications
- Key problems: Kernels might need different algorithms/implementations to get the best performance
- Ninja programming needs in addition to Kokkos
- Users of the kernels do not need to be ninja programmers
- ***Focus on performance 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

ECP Applications



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 in an agile manner at all stages from understanding requirements, designing kernels and evaluating them.

New Features in Kokkos Kernels 3.0

Sparse Linear Algebra
- ✓Cluster Gauss-Seidel
- ✓ Sparse ILU factorization
- ✓ Sparse triangular solves for sparse L and U
- ✓ Sparse triangular solves for supernodal L and U
- ✓ Structured sparse matrix vector multiply

Dense Linear Algebra
- ✓ Faster kernels for orthogonalization
- ✓ Complex support for dense LU factorization
- ✓Interfaces to vendor libraries
- ✓More BLAS and LAPACK support with Kokkos views

Graph Algorithms
- ✓Distance-2 graph coloring
- ✓Faster distance-1 graph coloring
- ✓Balanced distance-1 coloring
- ✓Balanced “well shaped” graph clustering
- ✓RCM ordering for preconditioners

Portable Vectorization
- ✓ Support ARM platforms
- ✓ Improved application performance on CPU, KNL, GPU and ARM
- ✓ Portable SIMD primitive

Team Level Kernels
- ✓ Team level sorting utilities
- ✓ Team level DFS
- ✓ More team level BLAS and LAPACK support

Software
- ✓CMake support
- ✓ETI changes to allow ETI file generation at compile time
- ✓ Improved testing
- ✓ Increased robustness

Kokkos Kernels is rapidly growing to support the needs of computational science applications.

Distance-2 Coloring, Cluster Gauss-Seidel

Brian Kelley

- Distance-2 graph coloring: vertices ≤ 2 hops apart can't share a color.
- Equivalent to coloring GG^T, or G^2 if symmetric (the 3 test graphs below are)
- New asymptotically faster algorithm is $O(V\delta\gamma)$, rather than $O(V\delta^2\gamma)$
  - δ = degree, γ = #colors
  - γ and δ are constant given a PDE matrix structure (e.g. 27-point stencil)
- Speeds up MueLu multigrid aggregation on V100 GPU by up to 8.5x
  - Aggregation no longer a bottleneck in setup on GPU

Sequential	V	δ	New Kokkos	Old Kokkos	ColPack	Zoltan (no MPI)
Graph:			Time (sec)	Slowdown factor vs. new Kokkos algorithm		
af_shell7	505K	34.8	0.0223	24x	12.7x	20.4x
cfd1	70.7K	25.8	0.0025	76x	22.3x	17.8x
msc04515	4.5K	21.6	0.000129	18.2x	25.2x	19.3x

CUDA	New Kokkos	Old Kokkos
af_shell7	1.28	2.0x
cfd1	0.115	2.2x
msc04515	0.00534	10.4x

Measured with Intel Xeon W-2155 (Skylake) and Nvidia Quadro P2000.

Work on parallel version is ongoing.

Cluster Gauss-Seidel

- One Gauss-Seidel iteration: for i = 1…m, $\mathbf{x_i = D_i^{-1}(b_i - \langle A_{i:}, x \rangle)}$
  - Order matters: update to x_i affects x_{i+k} later
- KokkosKernels has had parallel multicolor Gauss-Seidel (MTSGS) for several years
  - Colors vertices, and applies G-S over each color set in parallel

Rationale: if *i* is not adjacent to *j*, then the old value of x_i can be used in x_j's update. This works because x_i does not appear directly in x_j's update formula.

- But if there is a path from *i* to *j*, then updating x_i would still have a (small) effect on x_j in classical GS, inducing error in the parallel version.

New cluster GS: coarsens graph into "clusters", and colors coarsened graph

- Each color still processed in parallel
- Average path between vertices updated in parallel is much longer, reducing error
- Same parallelism (in principle), but significantly higher preconditioner quality than MTSGS
- Work in progress: have demonstrated superior preconditioning, but slow
  - Reduced preconditioned CG iterations by 19% (to within 1% of sequential GS) on af_shell7, compared to MTSGS
  - But, apply time is still 3-4x slower than MTSGS and Chebyshev iterations.
  - It should be possible to close the gap by improving load balance and clustering quality.

Structured Sparse Matrix-Vector (SpMV) kernel

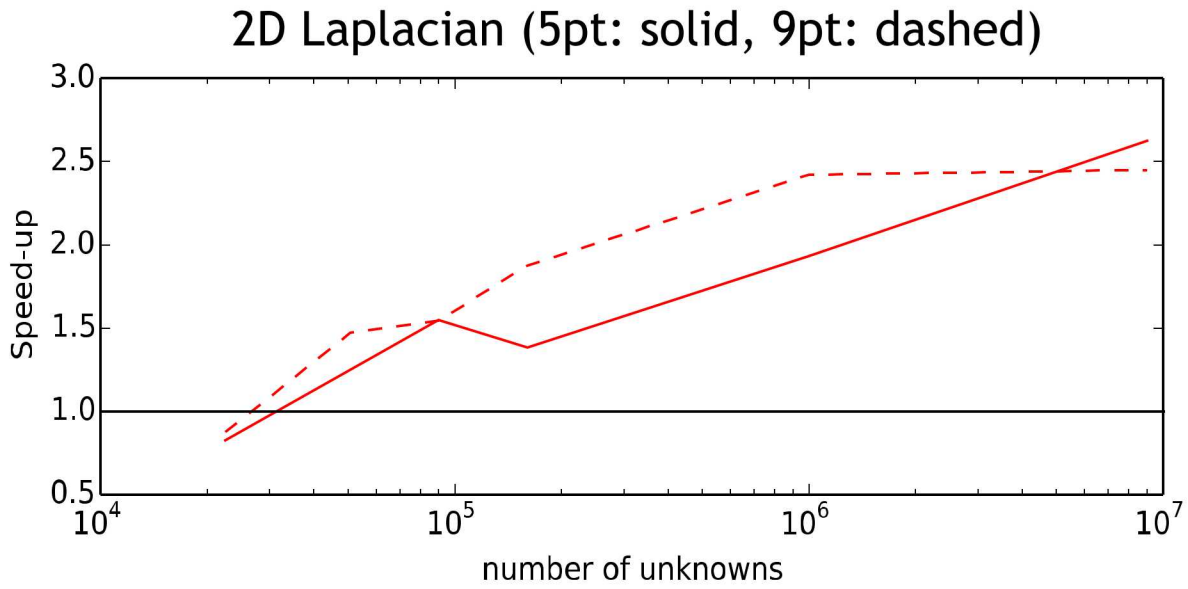
Implements: $y=\alpha*y+\beta*A*x$

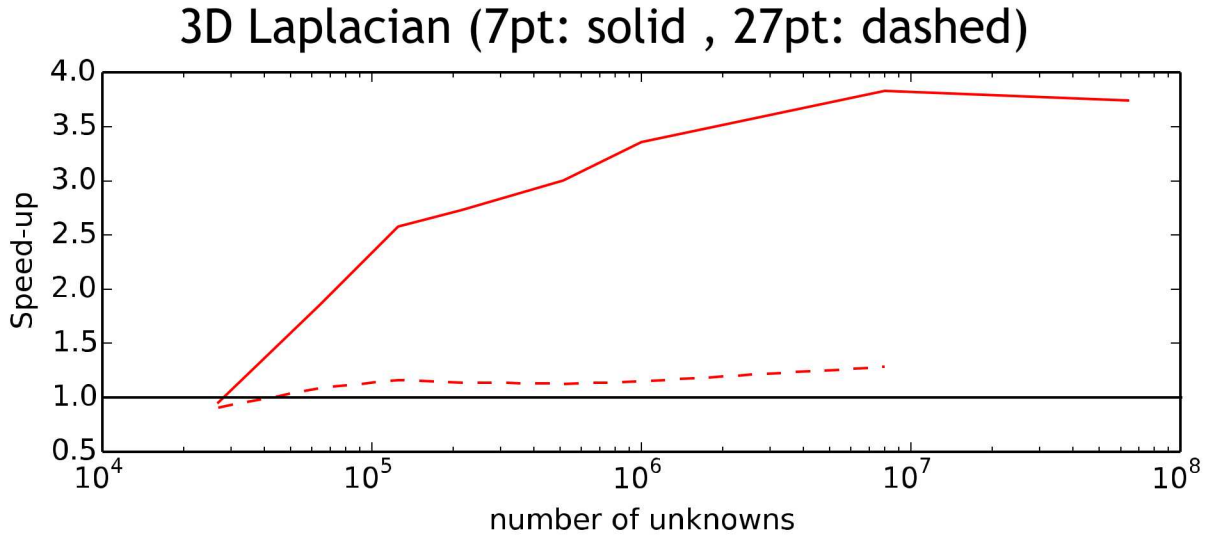
with A a CrsMatrix representing an operator discretized on a structured grid.

Stencils implemented:
- 5/9pt stencil (2D FD/FE Laplace)
- 7/27pt stencil (3D FD/FE Laplace)

Top (resp. bottom) figures shows the speedup obtained compared with generic SpMV implementation for 2D (resp. 3D) stencils.

Further improvements can be obtained for Multiphysics problems.
Additional optimizations are investigated for 27pt stencil.





Improving linear solvers strong scaling on Summit for ExaWind

Test problem:

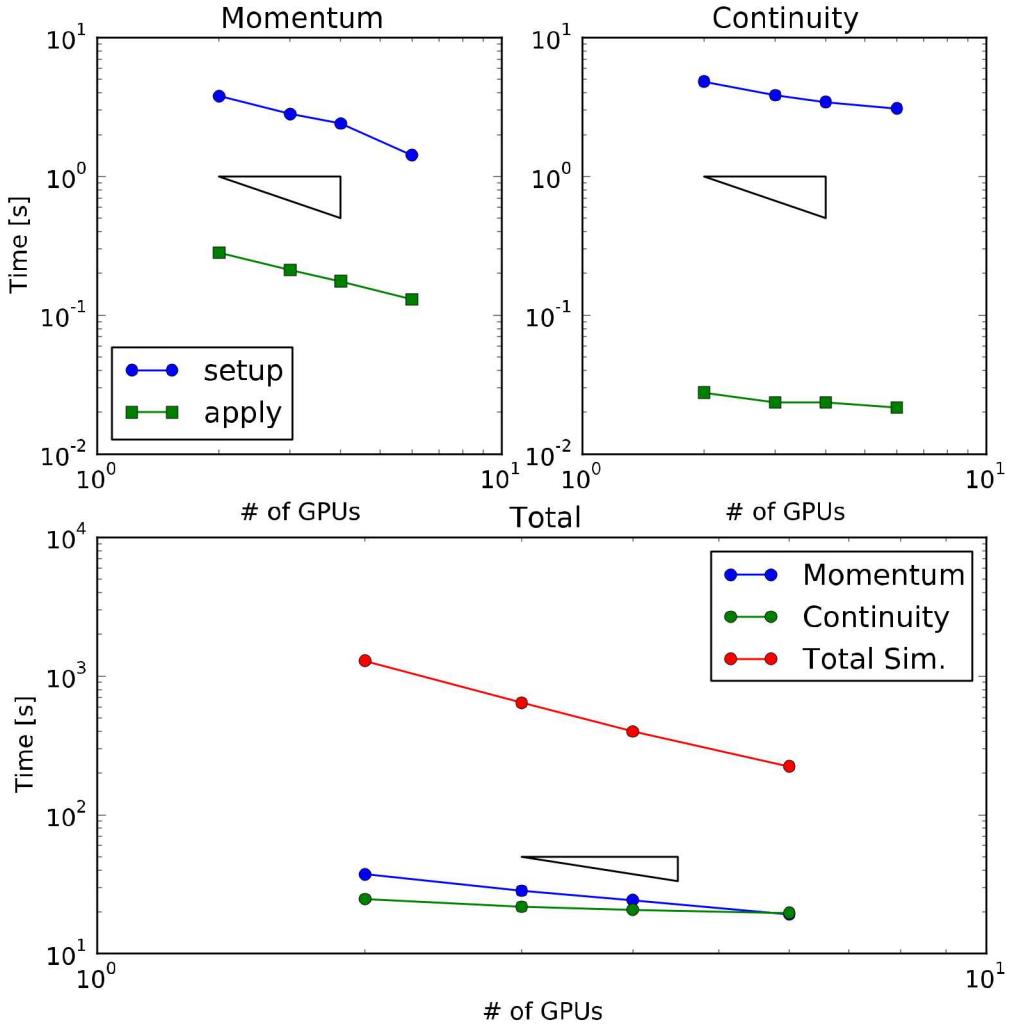
- 5kmx5kmx1km atmospheric boundary layer
- 20m resolution (i.e. ~3.2M nodes)
- Momentum solved with GMRES+Gauss-Seidel
- Continuity solved with GMRES+SA-AMG

Figure shows:

- Top: time for a single linear solver setup and apply for the momentum equation (left) and the continuity equation (right)
- Bottom: the total time associated with solving the momentum equation, the continuity equation and the total simulation time

Observations:

- All assembly and linear solvers are performed on GPU
- Momentum solver strong scales almost linearly
- Continuity solver scaling still requires improvement but cost per iteration is low
- Overall simulation time is scaling very well on Summit



Sparse-triangular solve in Kokkos-kernels

Motivations

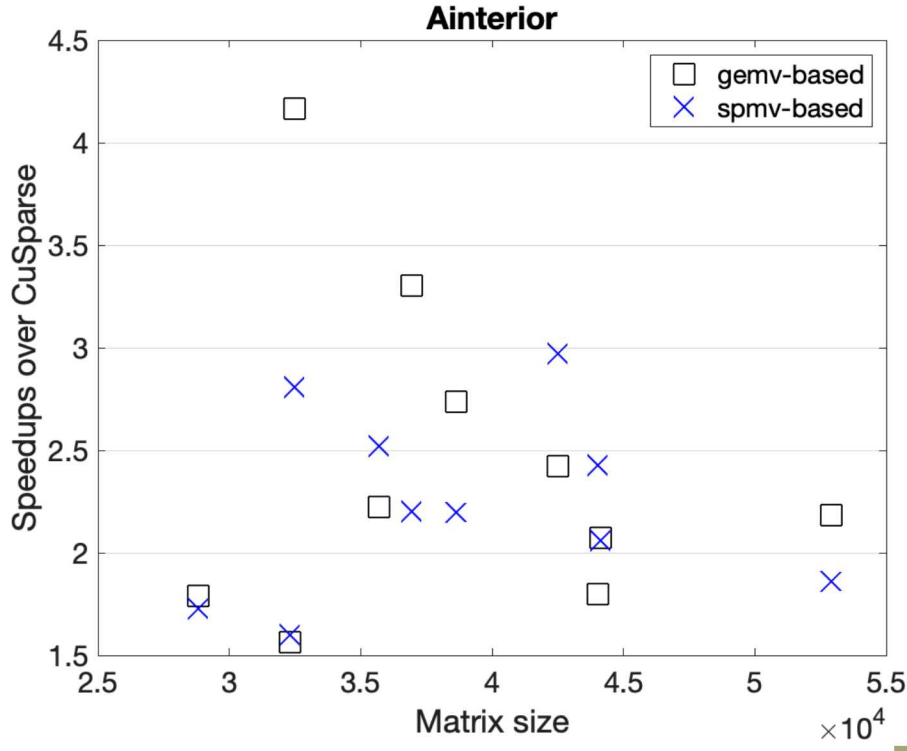
- Some distributed-memory solvers (e.g., DD) relies on the sparse-triangular solves as local solver
- Some of them use local direct solver (e.g., SuperLU, Cholmod, Techos)
- We could have $O(10^5)$ of triangular solves per factorization

Supernode-based sparse-triangular solve with level-set scheduling

- Interfaced with SuperLU & Cholmod
- Batched-team or device-level Kokkos-kernels at each level of scheduling
- Option to call SpMV at each level
  (i.e., triangular inverse as the product of partitioned inverses)

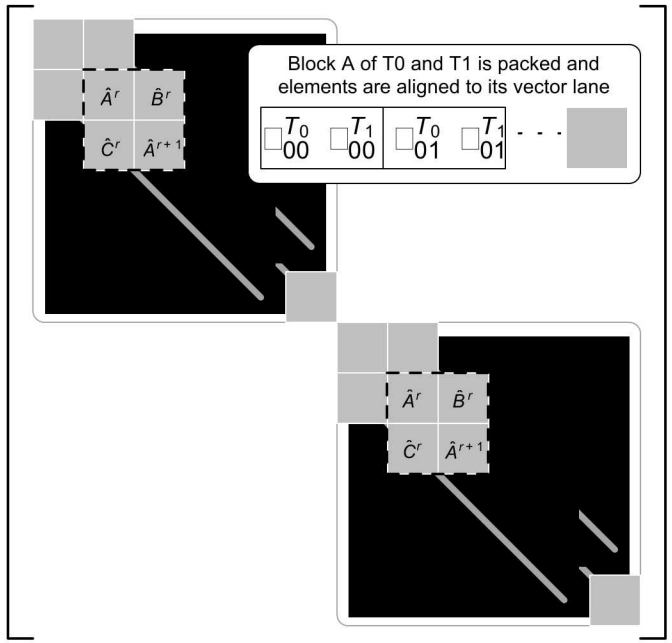
	DAG	Merge	Invert-offdiag	SpMV-DAG	Merge	Invert-offdiag
L-solve (CSC)	3.54x	4.24x	5.31x	3.03x	6.59x	14.11x
U-solve (CSC)	4.00x	4.98x	4.87x	4.39x	8.30x	13.17x

-- Speedups over CuSparse for A_20x20x20_electricity (n=27,783) on P100 --

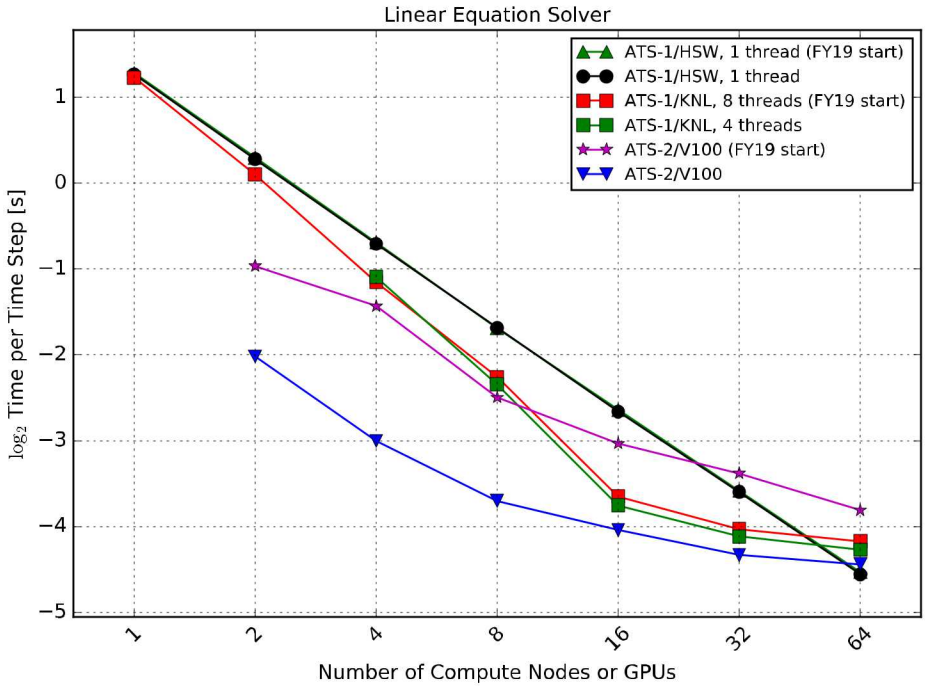


Block Tridiagonal Solver Performance on ARM

- A portable vectorization approach for next generation computing platforms was developed for block tridiagonal solver for SPARC (Sandia reactive flow solver).
  - Pack multiple blocks into a compact (interleaved) data format for efficient vectorization.
- Demonstrated scalable performance for Intel Xeon and NVIDIA GPU architectures where exploiting wide vector units are essential for performance.



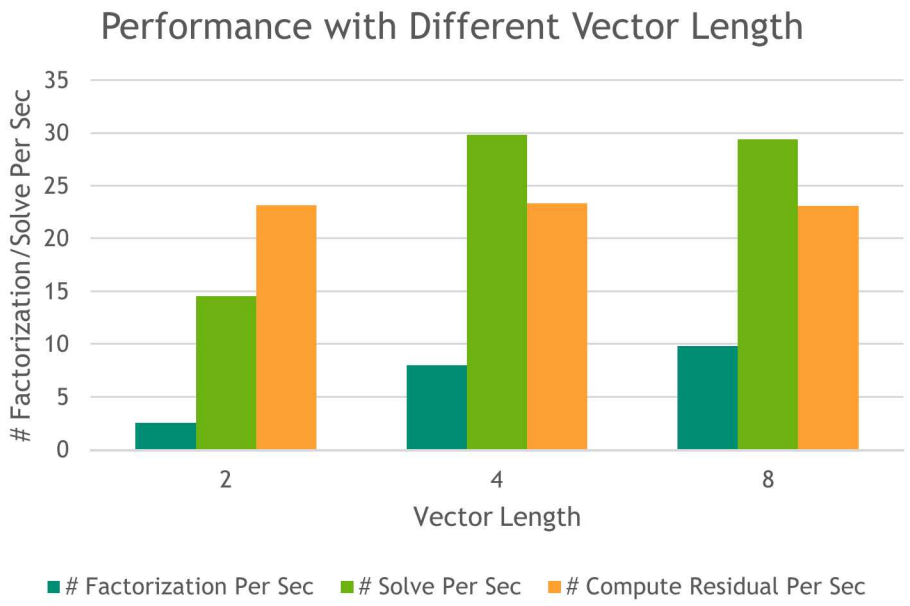
Compact Data Layout for Block Tridiagonals

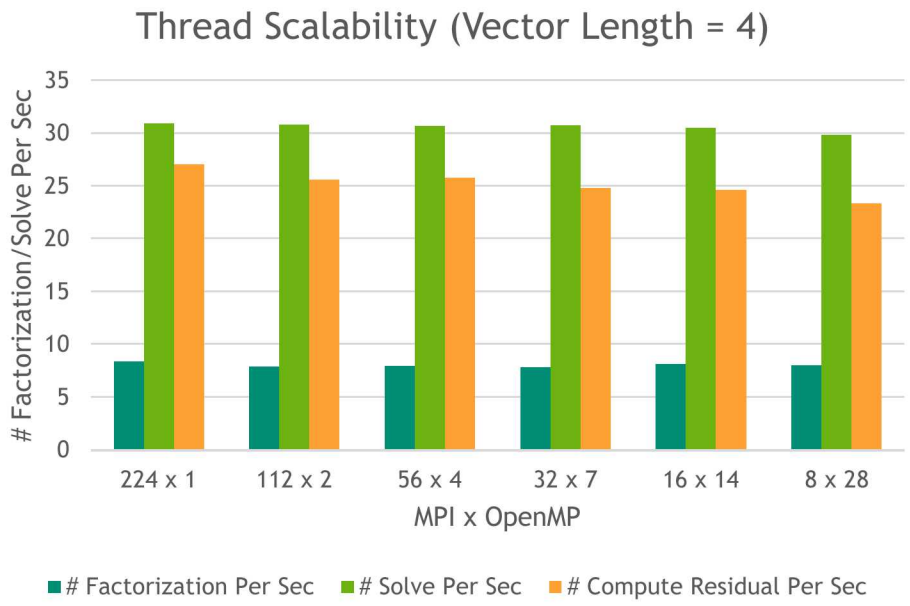


Strong Scale of block tridiagonal solver on Intel HSW, KNL, and NVIDIA V100

Block Tridiagonal Solver Performance on ARM

- Performance Test
  - 2x28 Core ARM Thunder X2, 2GHz , **two 128 bit vector units**.
  - A cube domain 256x224x100 with block size 7 (550k block tridiagonal matrices and total 40 million unknowns).
- Some observations:
  - Factorization and solve phases are performed on compact data layout; compute residual phase uses block CRS matrix format.
  - A wider vector length than hardware vector units (128bits) is necessary to hide latency and improve throughput.
  - Using 256 bit vector length, **factorization and solve performs 3x and 2x speedup** compared to a case using hardware vector length (128bit).
  - Demonstrate a portable (**no code change but the vector length**), vectorized and thread-scalable block line solver for ARM architectures.





A faster GEMM implementation

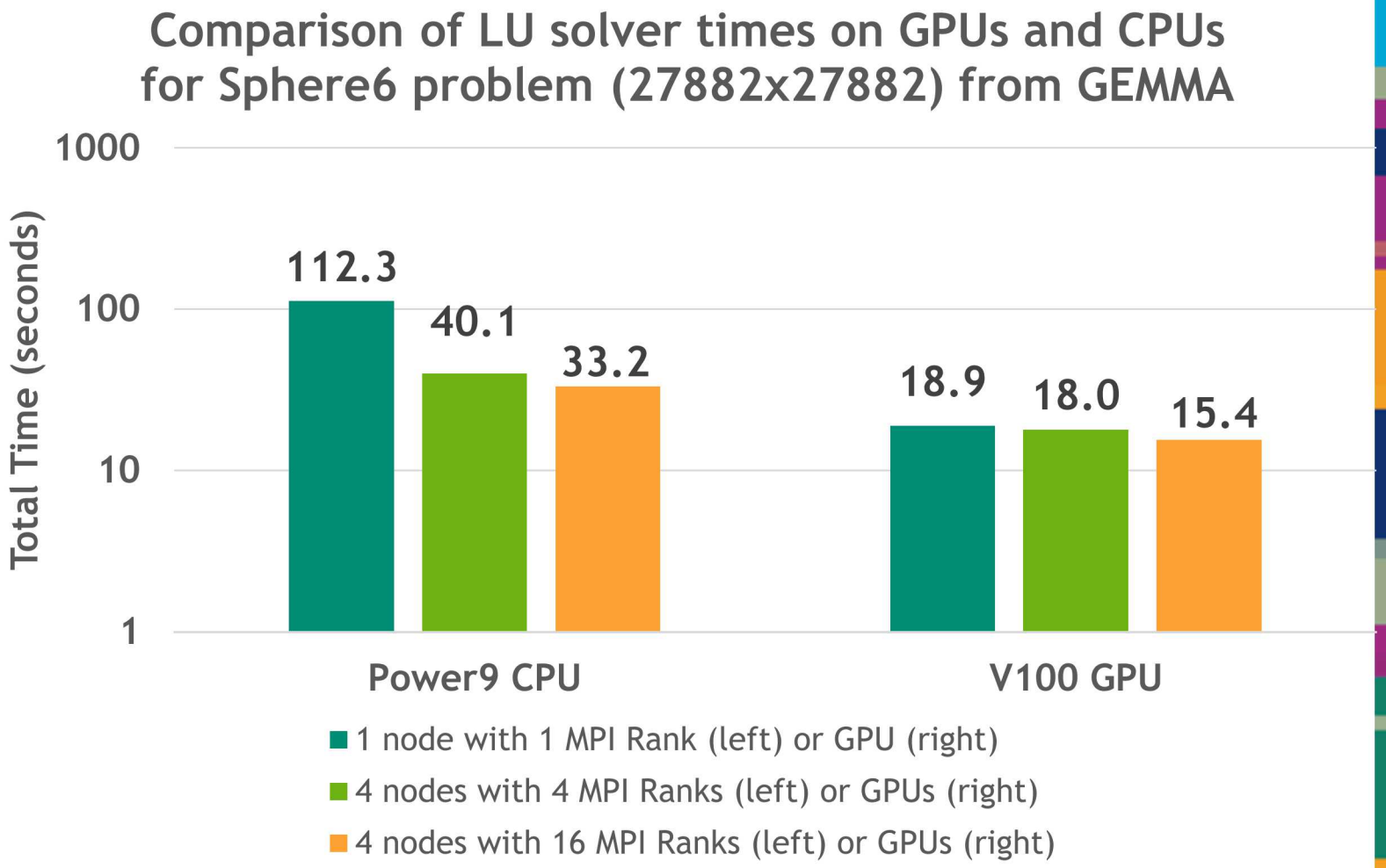
- Implements C = alpha C + beta A^TB for dense, **tall and skinny matrices** A and B
  - Performs a dot product a for each entry of C
- A common use case in iterative solvers, e.g., orthogonalization with multiple vectors
- Uses **two-level Kokkos parallelism**
  - Each thread team works on a portion of a dot-product
  - Team size and number of teams are determined according to matrix sizes
- Kokkos-based GEMM is **faster** than cuBLAS GEMM:

Execution times of GEMM implementations in seconds
(on an NVIDIA Volta V100 GPU, A and B are of size n x s)

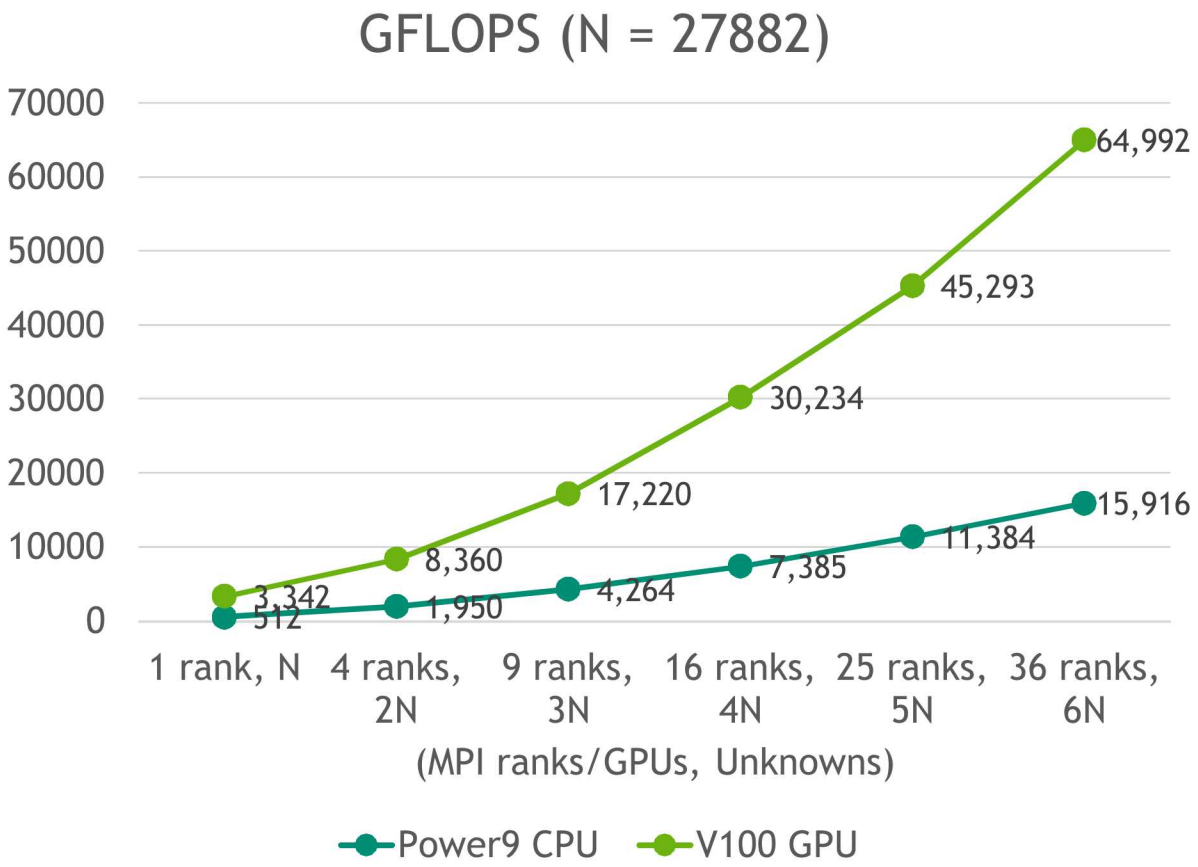
n	s = 3		s = 5		s = 7	
	cuBLAS	Kokkos	cuBLAS	Kokkos	cuBLAS	Kokkos
1,000	0.02	0.03	0.02	0.03	0.03	0.04
10,000	0.07	0.04	0.07	0.03	0.11	0.05
100,000	0.17	0.04	0.18	0.07	0.21	0.11
1,000,000	2.58	0.19	2.59	0.45	3.00	0.91
10,000,000	45.68	3.08	45.62	6.81	46.40	12.03

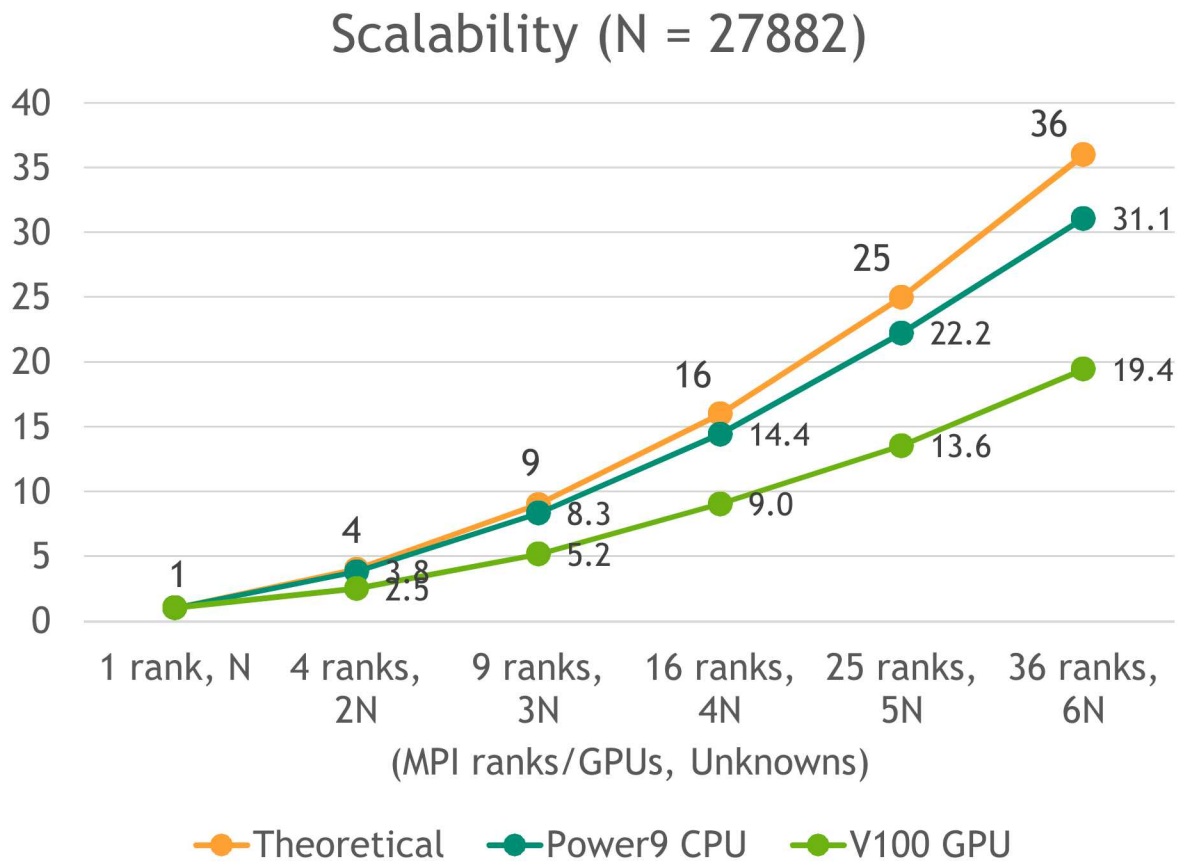
Adelus: **A** Dense **LU** Solver Package

- Performs partial pivoting LU factorization and solves dense linear equation systems using MPI
- Targets performance portability with Kokkos and Kokkos Kernels
- Matrices are torus-wrap mapped and evenly distributed onto the processors to load balance the computation and communication
- CUDA-aware MPI is employed on GPU architectures when possible



Adelus: Weak Scalability





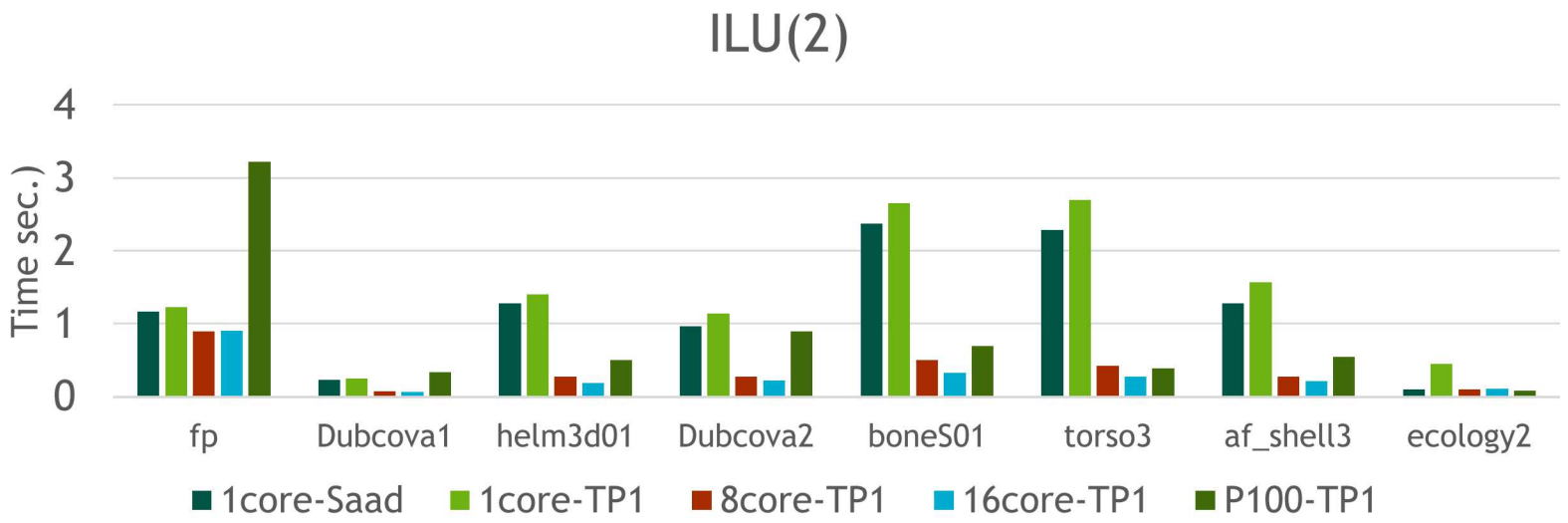
- Problem size increases with the number of MPI ranks/GPUs, such that each MPI rank/GPU holds the same amount of matrix portion
- GPUs are ~4x faster than CPUs. Scalability needs further improvement

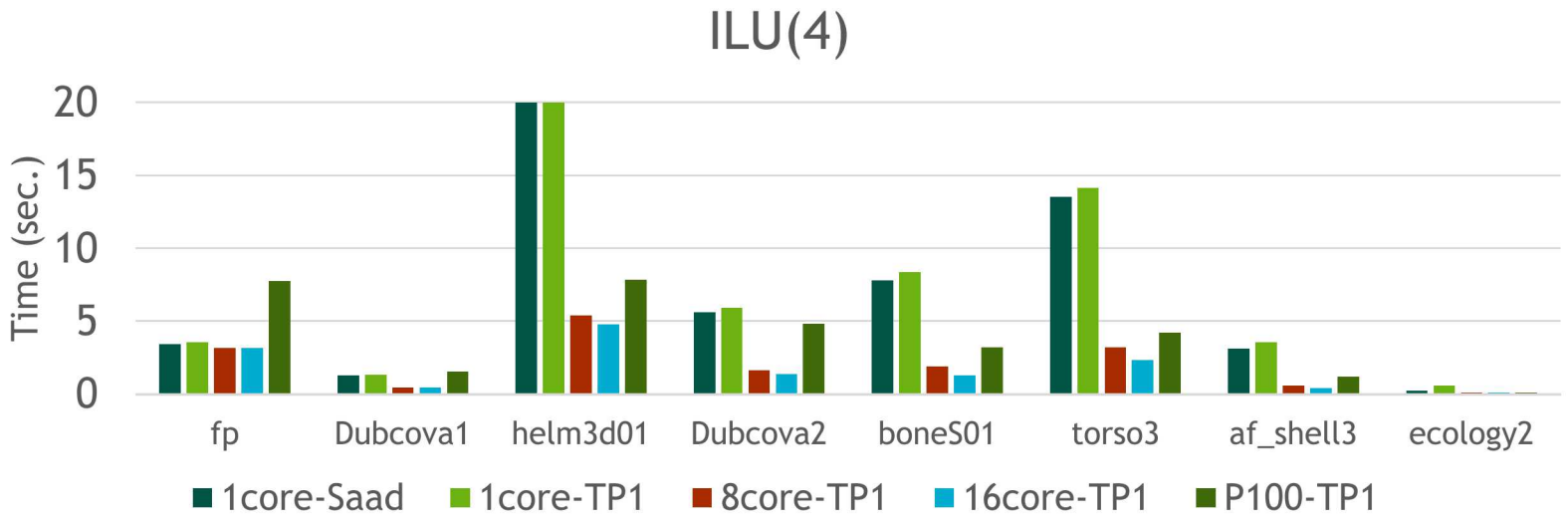
$$S = \frac{GFLOPS(ranks/GPUs, unknowns)}{GFLOPS(1,1N)}$$

where ranks/GPUs = 1, 4, 9, 16, 25, 36
unknowns = 1N, 2N, 3N, 4N, 5N, 6N

ILU(k)

- Symbolic phase (on host) constructs L and U patterns and groups the independent rows into levels
- Numeric phase (on host or GPU) obtains the resulting L and U factors by iterating sequentially across levels
- Kokkos team policies are used to implement hierarchical parallelism
  - Each thread team is assigned to a row in a single level
  - Threads in a team collectively update elements within a single row
- Optimization for GPU is in progress





FUTURE DIRECTIONS/ GOALS

Future Directions (New Features)

Support machine learning needs of ECP applications
- Batched linear algebra
- Portable convolution kernels
- Needs of the ExaLearn project

Support new preconditioners and linear algebra kernels
- Smoothers for linear solvers
- Support for new linear algebra kernels (fused kernels)

Support a SIMD data type

Support new graph kernels for analytics applications
- Linear algebra based graph kernels
- Multithreaded, portable GraphBLAS