

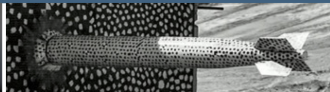
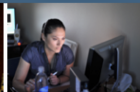


Sandia
National
Laboratories

Performance portable batched sparse linear solvers in Kokkos Kernels



 **CCR**
Center for Computing Research



Presented by:

K. Liegeois, S. Rajamanickam & L. Berger-Vergiat

Linear Solver BoF, May 10, 2022



EXASCALE
COMPUTING
PROJECT



Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & 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.

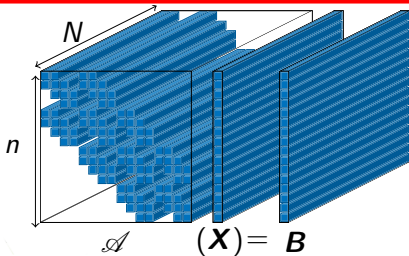


- ▶ Introduction:
 - ▶ Batched Sparse Linear systems;
 - ▶ Kokkos and Kokkos Kernels;
- ▶ Strategies for batched Krylov methods;
- ▶ Team batched SPMV;
 - ▶ Implementation;
 - ▶ Performances;
- ▶ Team batched GMRES;
 - ▶ Implementation;
 - ▶ Performances;
- ▶ Conclusions and future work.

Numerical strategies for solving PDE problems can lead to a **large number** of **small similar linear systems** to solve **independently**.

Example: a FE2 multiscale method requires a finite element computation for each Gauss point of the macroscopic scale mesh. Those systems share the same sparsity pattern and can be solved independently.

Need for a **performance portable strategy** to **solve large numbers** of relatively **small sparse linear systems**.



Batched size: $N \gg 1$,
Number of rows: $10 \leq n \leq 2000$.



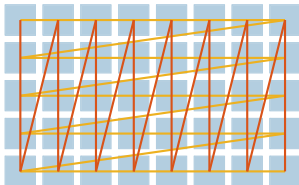
Kokkos:

- ▶ C++ performance portability library;
- ▶ Enables single source performance portable codes;
- ▶ Provides programming models for shared-memory parallelism;
- ▶ Provides 3 levels of hierarchical parallelism: team level, thread level, vector level;
- ▶ Provides data abstractions for performance portability.

Kokkos Kernels:

- ▶ Targets the performance portable implementation of linear algebra kernels;
- ▶ Provides computational kernels which rely both on the Kokkos data abstractions and programming models;
- ▶ Provides interface to vendor kernel implementations.

- ▶ An array of zero or more dimensions;
- ▶ Users can specify **left** (as in Fortran), **right** (as in C++), or stride layout;
- ▶ Views can be defined on the host or the device;
- ▶ Best layout for performance depends on the used shared-memory parallelism.



Introduction: Kokkos hierarchical parallelism



- ▶ A thread team is a collection of threads which can synchronize and which share a *scratch pad* memory;
- ▶ Instead of mapping a 1-D range of indices to hardware resources, Kokkos' thread teams map a 2-D index range (equivalent to 1-D grid of 1-D blocks in CUDA);
- ▶ The maximal number of teams is not architecture dependent, it is only limited by the integer size type;
- ▶ The maximal team size (# threads per team) is architecture dependent;
- ▶ The vector level needs to be vectorizable.

Team-1			
Thread-1		Thread-2	
v1	v2	v3	v4

Team-2			
Thread-3		Thread-4	
v5	v6	v7	v8

Kokkos	GPUs	CPUs
Team	Thread block	Work assigned to group of hyper threads
Kokkos thread	(full, half, quarter...) Warp	Work assigned to a single thread
Vector lane	Threads within a warp	Vectorization units

Parallelize over individual problems:

- ▶ A particular **team** is associated with **a unique system** at a given time;
- ▶ Every system **converges independently**;
- ▶ **Vectorization** and **coalesced memory read** in the Sparse Matrix-Vector multiplication (SPMV) kernel are **graph dependent**.

Approach used by the Ginkgo team:

H. Anzt, A. Kashi, P. Nayak, et al. <https://ginkgo-project.github.io>.

Parallelize over subsets of problems (two existing approaches):

- ▶ A particular **team** is associated with **a subset of systems** at a given time;
- ▶ Reuse of **common variables** such as the sparsity pattern, more **data parallelism**, improved **memory access pattern**;
- ▶ First subset approach: Solving the coupled problems:
 - ▶ The matrices are **gathered** into one matrix, the Krylov method is then applied to the system;
 - ▶ The **convergence** depends on the **union** of the **spectra** of all the matrices; this can be worse than the worst convergence taken one by one.



Parallelize over subsets of problems (two existing approaches):

- ▶ A particular **team** is associated with **a subset of systems** at a given time;
- ▶ Reuse of **common variables** such as the sparsity pattern, more **data parallelism**, improved **memory access pattern**;
- ▶ Second subset approach: Solving the problems independently:
 - ▶ The **systems** are kept independent, they are **not coupled**, the spectra are not gathered;
 - ▶ The main drawback is the **code divergence**: inside a same subset, the Krylov methods might require different numbers of iterations for different systems to converge; this can lead to issues such as **overflow** if not treated carefully;
 - ▶ Needs an **implementation** of the used **kernels** which supports **subsets of values** instead of one value.



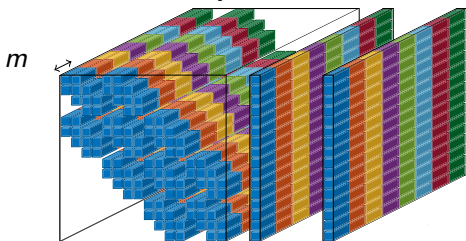
Parallelize over subsets of problems (two existing approaches):

- ▶ A particular **team** is associated with **a subset of systems** at a given time;
- ▶ Reuse of **common variables** such as the sparsity pattern, more **data parallelism**, improved **memory access pattern**;
- ▶ Second subset approach: Solving the problems independently: **Rest of this talk**
 - ▶ The **systems** are kept independent, they are **not coupled**, the spectra are not gathered;
 - ▶ The main drawback is the **code divergence**: inside a same subset, the Krylov methods might require different numbers of iterations for different systems to converge; this can lead to issues such as **overflow** if not treated carefully;
 - ▶ Needs an **implementation** of the used **kernels** which supports **subsets of values** instead of one value.

Chosen batched strategy in Kokkos Kernels



First, a **team parallel loop** is used to loop **over subsets** of size m of the N **matrices**. Then, a team has to solve m systems simultaneously.



$$1 \leq m \leq 50.$$

One team per color.

Software requirements:

- ▶ Krylov **solvers** at the **team level** which deal with possible occurrences of **code divergence** (as discussed in the case of the ensemble propagation in Liegeois (2020));
- ▶ **Performance portable batched** Level 1 and 2 **BLAS** functions (AXPY, DOT, COPY, SPMV, and GEMV) at the **team level**.

Rest of this talk



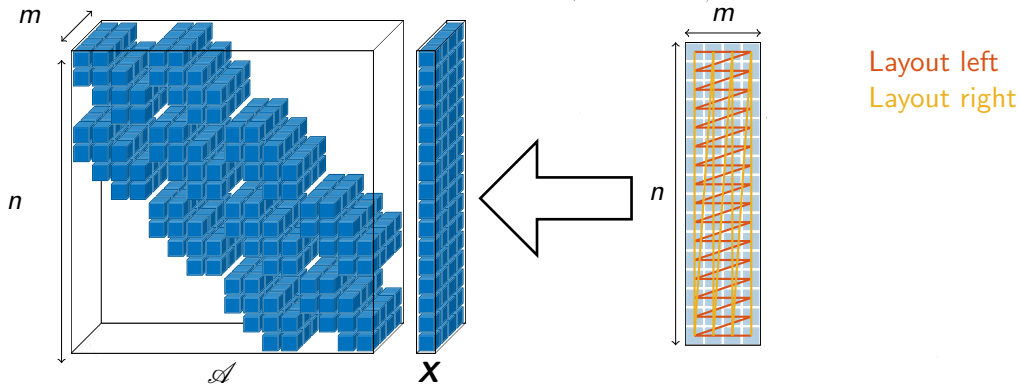
To illustrate the last software requirement, we discuss the case of the batched Sparse Matrix-Vector multiplication (SPMV):

$$\mathbf{y}_{\ell} = \alpha_{\ell} \mathbf{A}_{\ell} :: \mathbf{x}_{\ell} + \beta_{\ell} \mathbf{y}_{\ell} \quad \text{for all } \ell = 1, \dots, m.$$

Targeted properties:

- ▶ To achieve maximum hardware occupancy,
- ▶ To have good memory access patterns such as a high percentage of coalesced memory read on GPU,
- ▶ To have good performance independently of views layout,
- ▶ To have a balanced workload amongst teams and threads,
- ▶ To avoid unnecessary reduction and memory synchronization.

- ▶ nm independent products between $\mathbf{a}_{\ell j}$ and \mathbf{x}_{ℓ} ,
- ▶ TeamVector loop over the nm indices to distribute evenly the work,
- ▶ The mapping of the index of the loop to the row fiber depends on the layout to enforce as much coalesced memory loads as possible

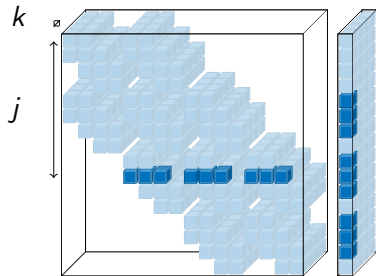


```
Kokkos::parallel_for(
  Kokkos::TeamVectorRange(member, 0, m * n),
  [&](const int& i) {
    int j, k;
    getIndices<layout>(i, n, m, j, k);
    const int rowLength = row_ptr(j + 1) - row_ptr(j);
    ValueType sum = 0;
    for (int l = 0; l < rowLength; ++l)
      sum += values(k, row_ptr(j) + l) *
            X(k, colIndices(row_ptr(j) + l));

    sum *= alpha(k);
    Y(k, j) = beta(k) * Y(k, j) + sum;
  });
```

where:

```
template <typename layout> KOKKOS_INLINE_FUNCTION
typename std::enable_if<std::is_same<layout,
  Kokkos::LayoutLeft>::value, void>::type
getIndices(const int i, const int /*n*/,
           const int m, int &j, int &k) {
  j = i / m; k = i % m;
}
```



- ▶ At the vector level, every i (and therefore the pair (j, k)) is associated with only one vector lane.
- ▶ No reduction nor memory synchronization are needed.

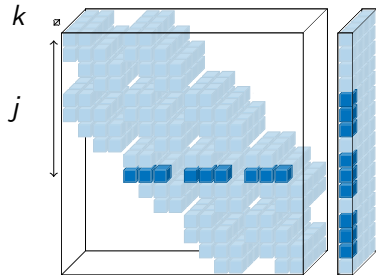
Base implementation: for comparison purpose



```
for (int k = 0; k < m; ++k) {
    Kokkos::parallel_for(
        Kokkos::TeamThreadRange(member, 0, n),
        [&](const int& j) {
            const int rowLength = row_ptr(j + 1) - row_ptr(j);

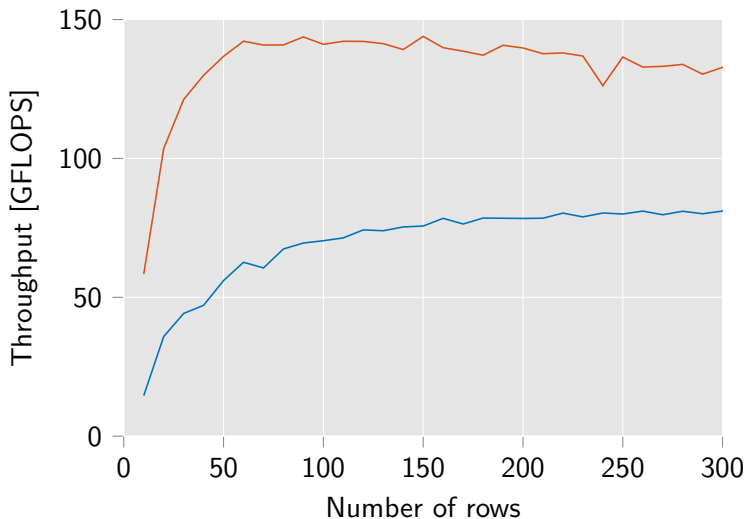
            ValueType sum = 0;
            Kokkos::parallel_reduce(
                Kokkos::ThreadVectorRange(member, row_length),
                [&](const ordinal_type& l, value_type& lsum) {
                    lsum += values(k, row_ptr(j) + l) *
                        X(k, colIndices(row_ptr(j) + l));
                }, sum);

            sum *= alpha(k);
            Y(k, j) = beta(k) * Y(k, j) + sum;
        });
}
```



- ▶ A for loop over the matrices within the team.
- ▶ A thread parallel for over the rows.
- ▶ A vector parallel reduce over the non-zeros.

Team batched SPMV: performance



— Base implementation
— Team approach

- ▶ On V100,
- ▶ $N = 51200$,
- ▶ 7 non-zero values per row.

Depending on the number of rows per matrix, the team approach can double the achieved throughput of the batched SPMV.



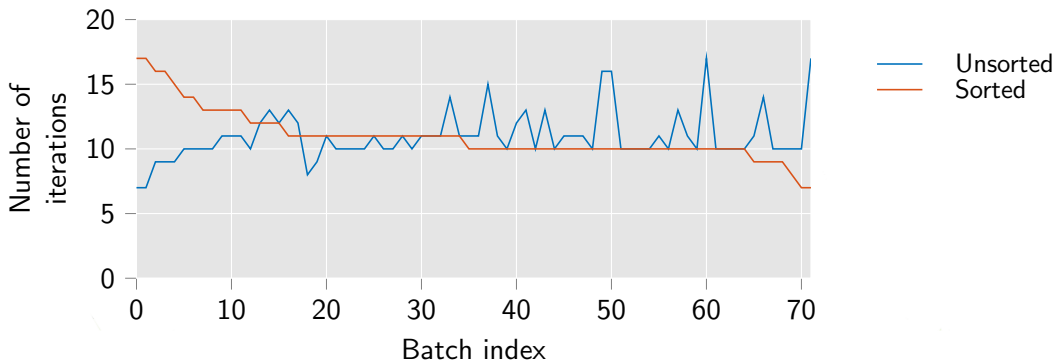
- ▶ Uses batched BLAS kernels: SPMV, AXPY, DOT, COPY, and GEMV,
- ▶ Continues the GMRES while the m systems have not converged,
- ▶ Stops the update of converged system to avoid underflow,
- ▶ Evaluated on devices without communication with the host.

```
for (size_t j = 0; j < maximum_iteration; ++j) {  
    A.apply(member, subview(V, ALL, j, ALL), W);  
    member.team_barrier();  
    P.apply(member, W, W);  
  
    for (size_t i = 0; i < j + 1; ++i) {  
        member.team_barrier();  
        auto V_i = subview(V, ALL, i, ALL);  
        TeamVectorDot<MemberType>::invoke  
            (member, W, V_i, tmp);  
        member.team_barrier();  
        TeamVectorCopy1D::invoke  
            (member, tmp, subview(H, ALL, i, j));  
        member.team_barrier();  
        parallel_for(  
            TeamVectorRange(member, 0, m),  
            [&](const OrdinalType& ii) {  
                tmp(ii) = -tmp(ii);  
            });  
        member.team_barrier();  
        TeamVectorAxy<MemberType>::invoke  
            (member, tmp, V_i, W);  
    } //...
```

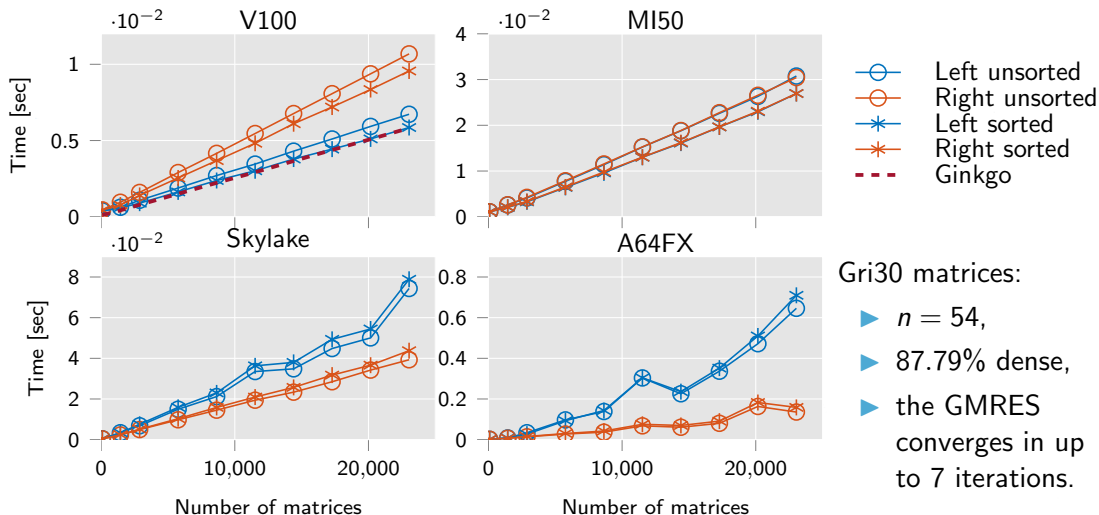
Batched GMRES performance: Impact of the grouping



- ▶ The grouping of the systems into subsets influences the measured performance,
- ▶ Best to group systems that need the same number of iterations to converge; but those numbers are unknown a priori,
- ▶ Two tested ordering for the systems: the unsorted and the sorted orders.



Batched GMRES performance: Pele gri30 matrices

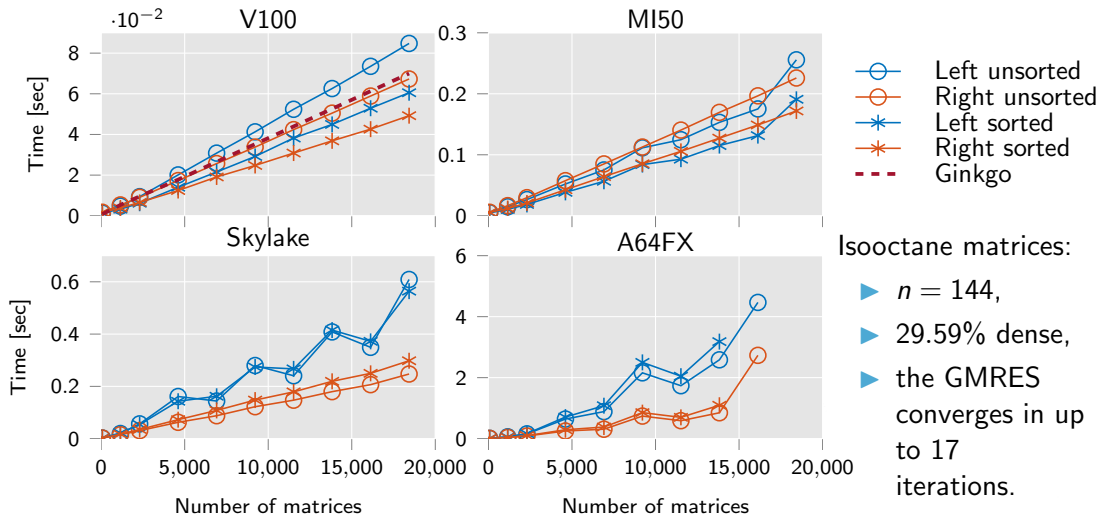


Gri30 matrices:

- ▶ $n = 54$,
- ▶ 87.79% dense,
- ▶ the GMRES converges in up to 7 iterations.

Good performance achieved on GPUs. Ordering has a limited impact.

Batched GMRES performance: Pele iso-octane matrices



Iso-octane matrices:

- ▶ $n = 144$,
- ▶ 29.59% dense,
- ▶ the GMRES converges in up to 17 iterations.

Good performance achieved on GPUs. Ordering has a larger impact.



Conclusions:

- ▶ We discussed main strategies for a performance portable batched sparse linear solver;
- ▶ We discussed the implementation of a batched SPMV and its performance;
- ▶ We briefly illustrate how kernels can be combined at the team level to write an efficient solver;
- ▶ We briefly illustrate the performance of the batched GMRES on four different architectures and the impact of the grouping.

Future work:

- ▶ Investigate the performance on CPU architectures (especially the left layout);
- ▶ Evaluate the performance of the batched GMRES compared to the performance of batched dense solvers;
- ▶ Evaluate the performance of the discussed kernels and solvers on other architectures;
- ▶ Evaluate the performance on larger application matrices;
- ▶ Develop other batched linear solvers.

- ▶ Ginkgo team: H. Anzt, A. Kashi, P. Nayak, et al.
Provided access to the Ginkgo source code for performance comparison of the batched GMRES,
- ▶ SUNDIALS team: C. Balos, David G. , C. Woodward,
Provided batched matrices associated with chemical species in reacting Navier-Stokes equations,
- ▶ M. Adams, Lawrence Berkeley Laboratory,
- ▶ This research was supported by the Exascale Computing Project (17-SC-20-SC), a joint project of the U.S. Department of Energy's Office of Science and National Nuclear Security Administration, responsible for delivering a capable exascale ecosystem, including software, applications, and hardware technology, to support the nation's exascale computing imperative.