Performance portable batched sparse linear solvers in Kokkos Kernels

Presented by:

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

Kokkos User Group, December 12, 2023
SAND2023-14340C

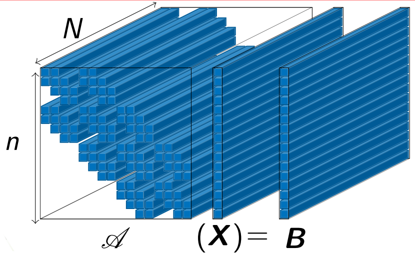
Outline

Introduction: Batched Sparse Linear systems

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.

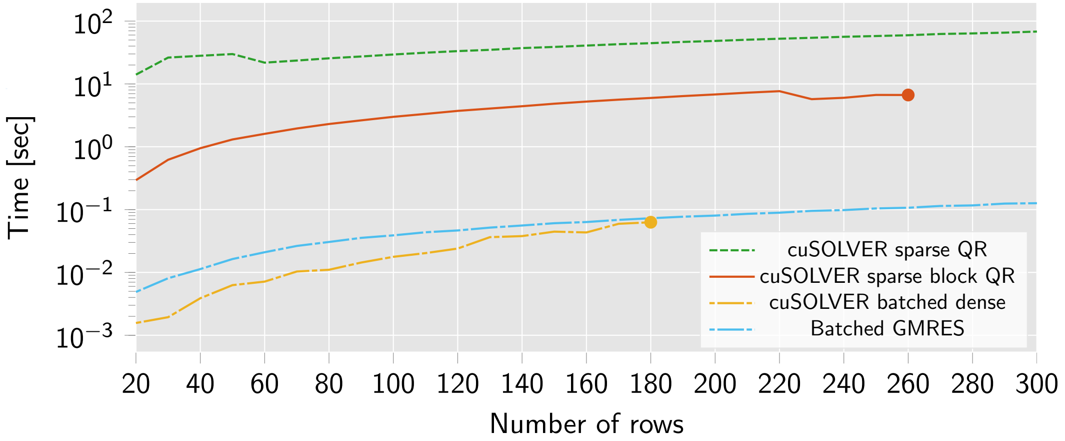
Need for a **performance portable strategy** to **solve large numbers** of relatively **small sparse linear systems**.



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

Liegeois, K. et al. "Performance Portable Batched Sparse Linear Solvers." IEEE Transactions on Parallel and Distributed Systems 34.5 (2023): 1524-1535.

Why do we need batched sparse linear solvers?



While usual sparse solvers or dense batched solvers are available and can be used in vendor libraries, their are not well suited for batched sparse linear systems.

Introduction: Kokkos and Kokkos Kernels

Kokkos:

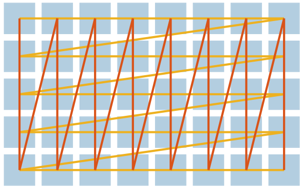
- C++ performance portability library;
- Enables single source performance portable codes;
- Provides programming models for shared-memory parallelism;
- Provides 3 levels of hierarchical parallelism: team level, thread level, vector level;
- Provides data abstractions for performance portability.

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.

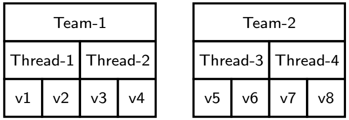
Introduction: Kokkos views

- 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.



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

Strategies for batched Krylov methods

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.

Strategies for batched Krylov methods

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.

Strategies for batched Krylov methods

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.

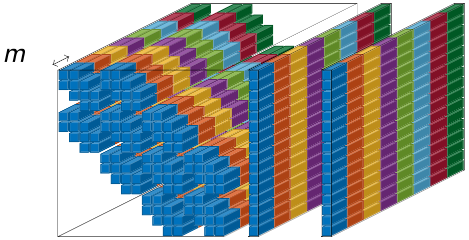
Strategies for batched Krylov methods

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

Liegeois, K. et al. "GMRES with embedded ensemble propagation for the efficient solution of parametric linear systems in uncertainty quantification of computational models." Computer Methods in Applied Mechanics and Engineering 369 (2020): 113188.

Performance portable batched SPMV at the team level

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

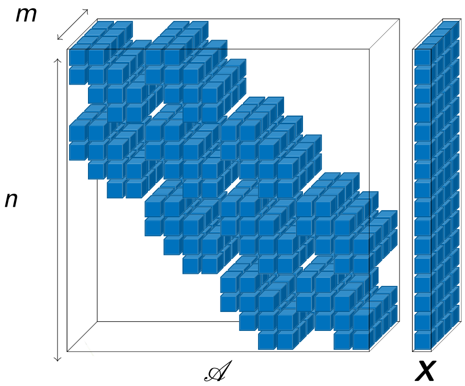
$$\mathbf{y}_{\ell:} = \alpha_\ell \, \mathbf{A}_{\ell::} \, \mathbf{x}_{\ell:} + \beta_\ell \, \mathbf{y}_{\ell:} \quad \text{for all} \quad \ell = 1, \dots, m.$$

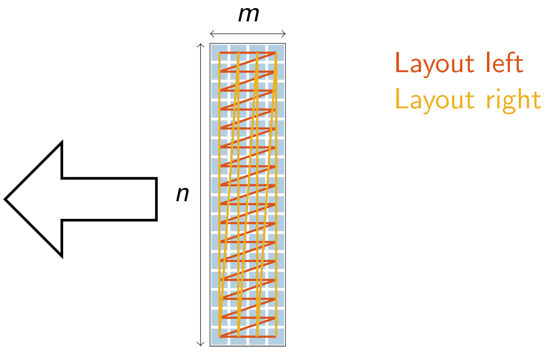
Targeted properties:

- 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.

Team batched SPMV

- nm independent products between $\boldsymbol{a}_{\ell j:}$ and $\boldsymbol{x}_{\ell :}$,
- 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



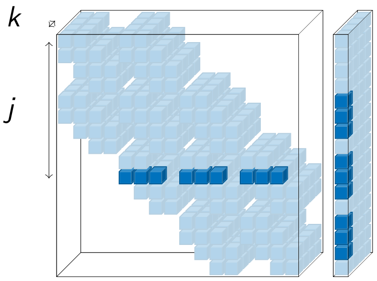


Team batched SPMV

```
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;
  });
```

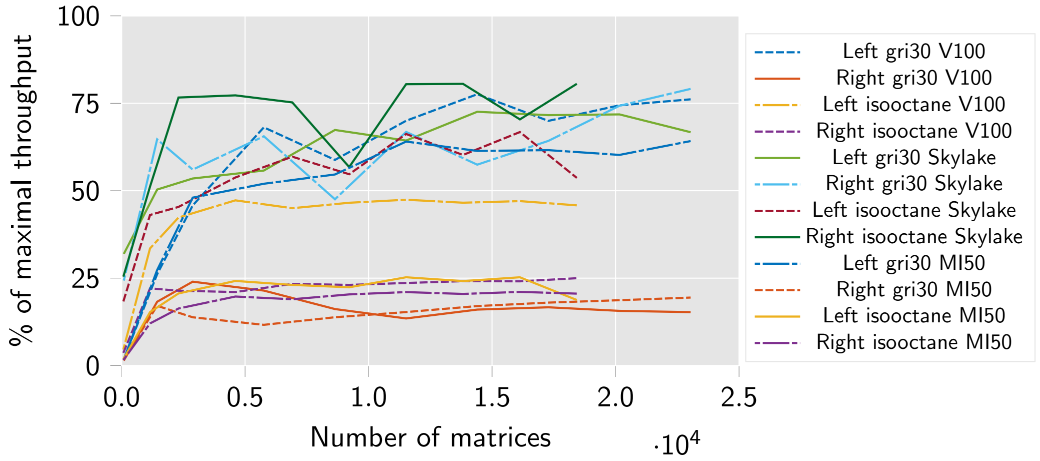
where:

```
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.

Team batched SPMV: performance portability



The maximal throughput is computed assuming that **every** data used more than once is reused from cache. Very good performance for left layout except for the isooctane on MI50.

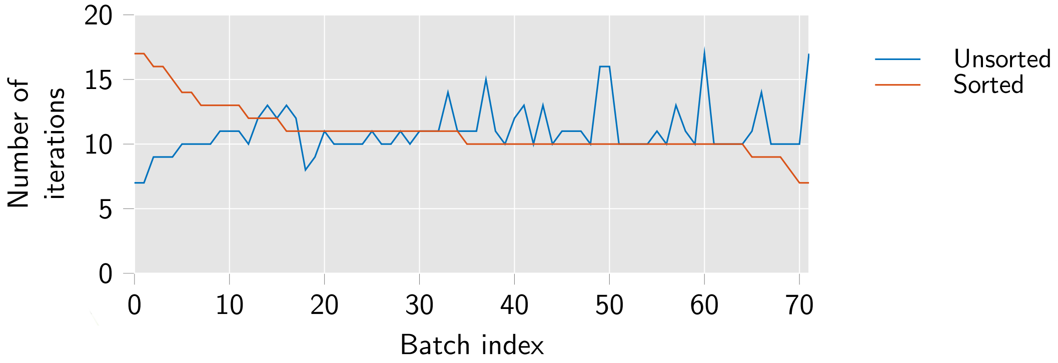
Team batched GMRES

- 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.

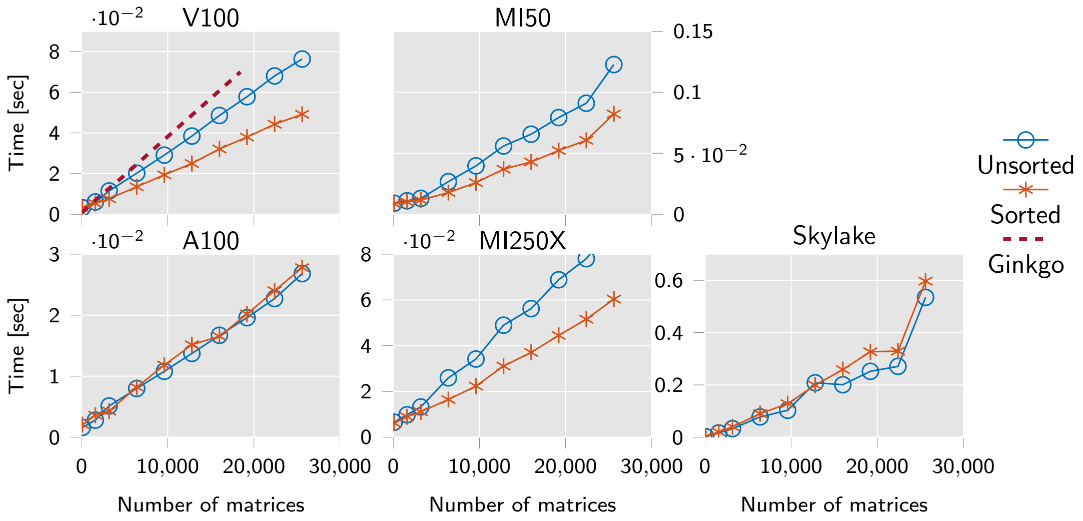
```
for (size_t j = 0; j < maximum_iteration; ++j) {
  A.apply(member, subview(V, ALL, j, ALL), W);
  member.team_barrier();
  P.apply(member, W, W);

  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();
    TeamVectorAxpy<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 isooctane matrices



Good performance achieved on GPUs. Faster than Ginkgo on V100.

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 five different architectures and the impact of the grouping.

Acknowledgment

- 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.

ECP EXASCALE COMPUTING PROJECT