SAND2022-5867C

Kokkos Kernels (Sake project)

Kokkos Kernels Overview

Kokkos Kernels is the portable math library within the Kokkos ecosystem. It supports a wide variety of math algorithms roughly categorized as
• Dense linear algebra kernels,
• Sparse linear algebra kernels,
• Batched linear algebra and solvers,
• Graph algorithms.

Using the data structures and hierarchical parallelism approach provided by Kokkos Core, the algorithms are implemented on all architectures of interest for ECP and the DoE scientific applications.

The development team partners with applications and computing facilities to ensure that other ECP libraries and applications can benefit from the library's constant improvements.

Finally algorithmic co-design with vendors and other libraries ensure collective success and helps enhance the state of computing on the most advanced architectures.

Early Access Platforms status

Kokkos Kernels supports all the platforms of interest for ECP and supports the backends of Kokkos Core. On each backend, architecture specific optimizations are implemented and vendor libraries (TPLs) are interfaced when available.

On host all backends are fully supported:
1. Serial, OpenMP and Threads (renamed Pthreads backend)

On device the following backends are supported
1. CUDA: production support and performance optimization
2. HIP: still in experimental namespace, close to production (see recent performance improvement in Figure 1 below)
3. SYCL: experimental support, waiting for more stable software ecosystem and wider availability of hardware
4. OpenMP Target: insufficient compiler/linker support to allow significant support

To facilitate utilization of Kokkos Kernels on early access platforms, new HIP recipes are available to build using Spack (tested on Spock) and experimental SYCL recipes are under development (modules available at JLSE).

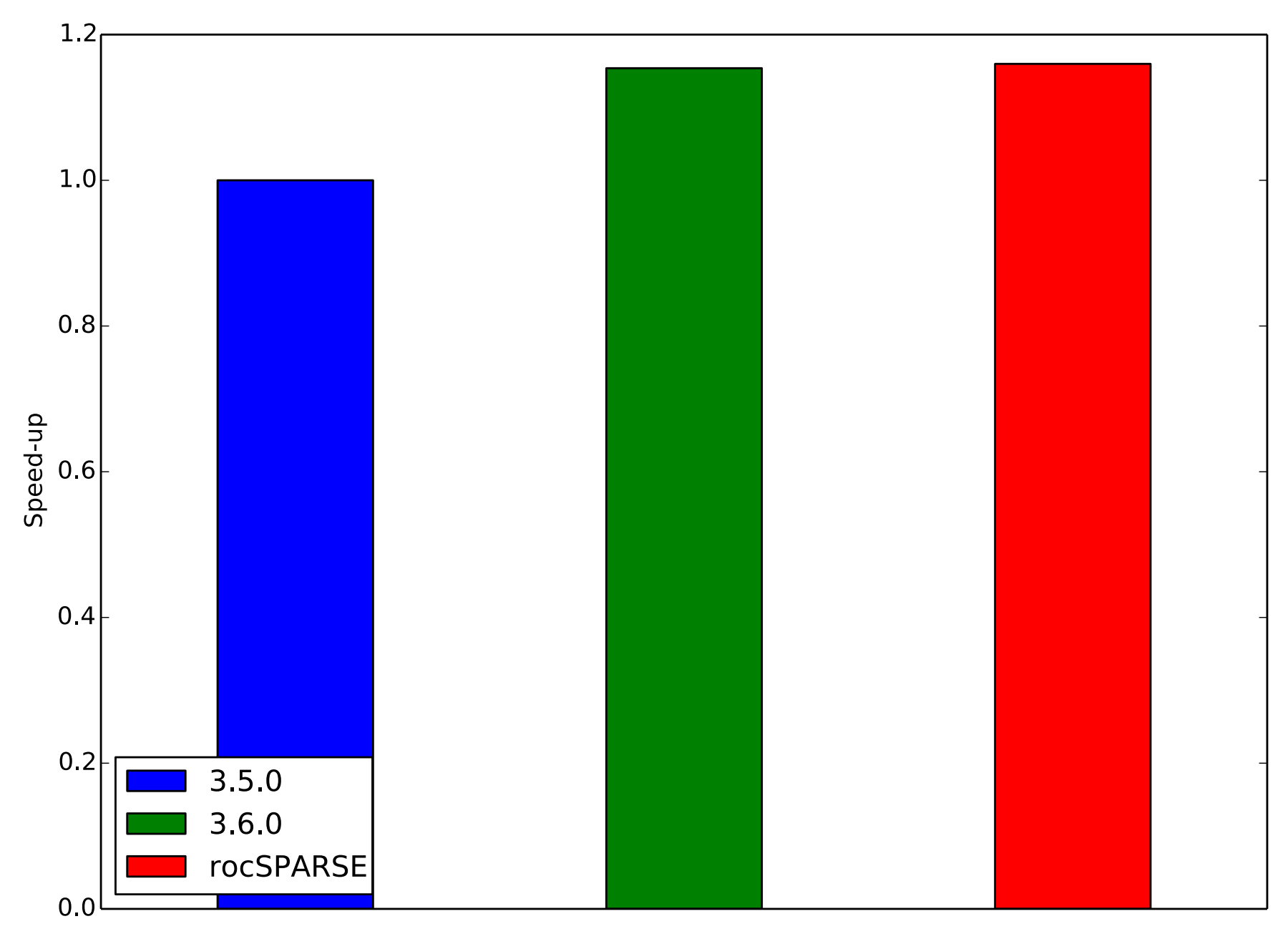

Figure 1: performance improvement of SpMV in the HIP backend between Kokkos Kernels 3.5.0 and 3.6.0

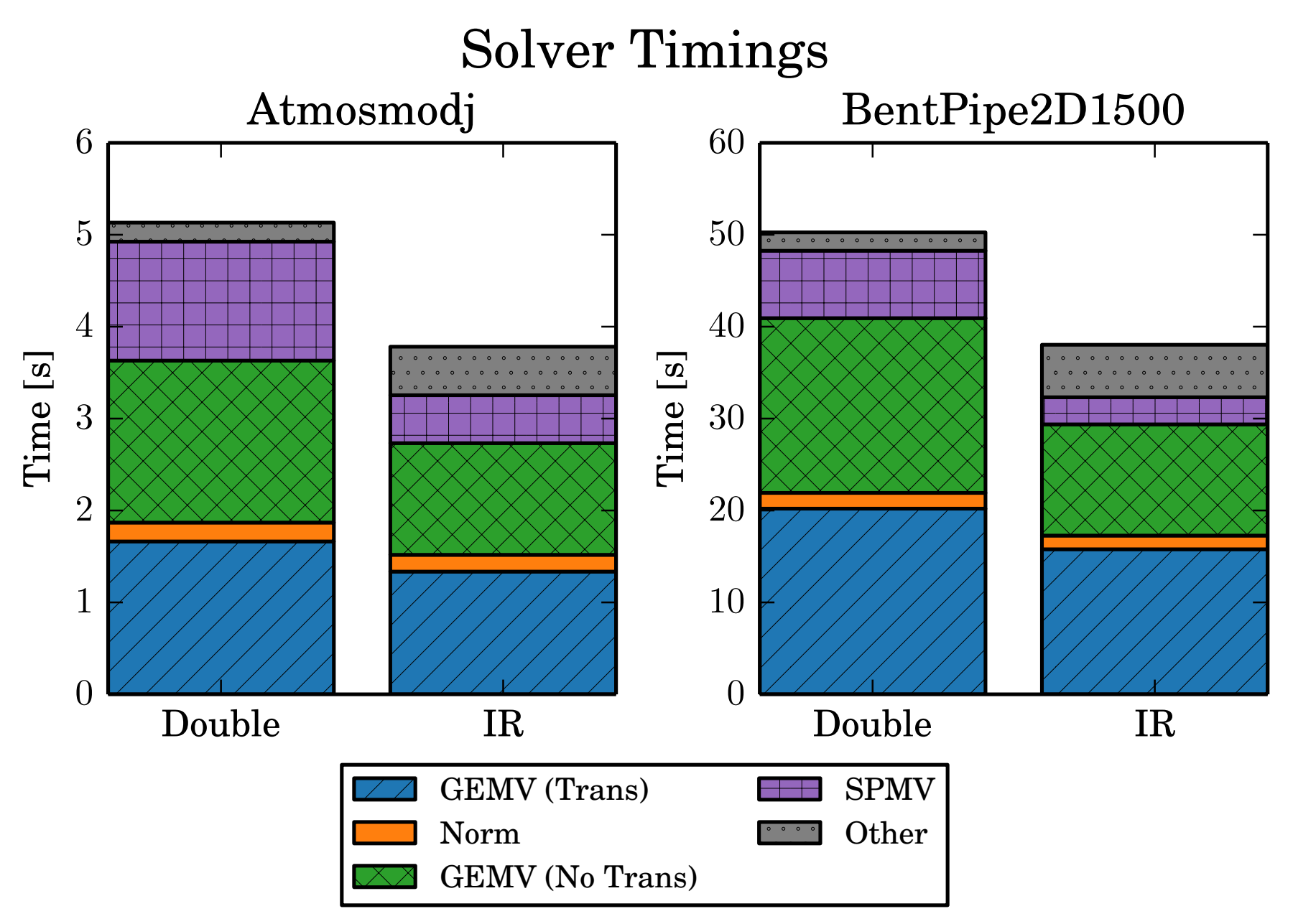

Figure 2: Solve times for GMRES(50) double vs. Iterative Refinement

Highlights from 2021/22

New batched linear solvers are being developed in support of applications. They allow a variety of physics code to solve multiple problems on GPU for collisions detection, chemical reactions, material constitutive law evolution, etc.
The approach has been tested on multiple compute architectures and good scaling with respect to the number of problems to solve is observed, see Figure 3 below.

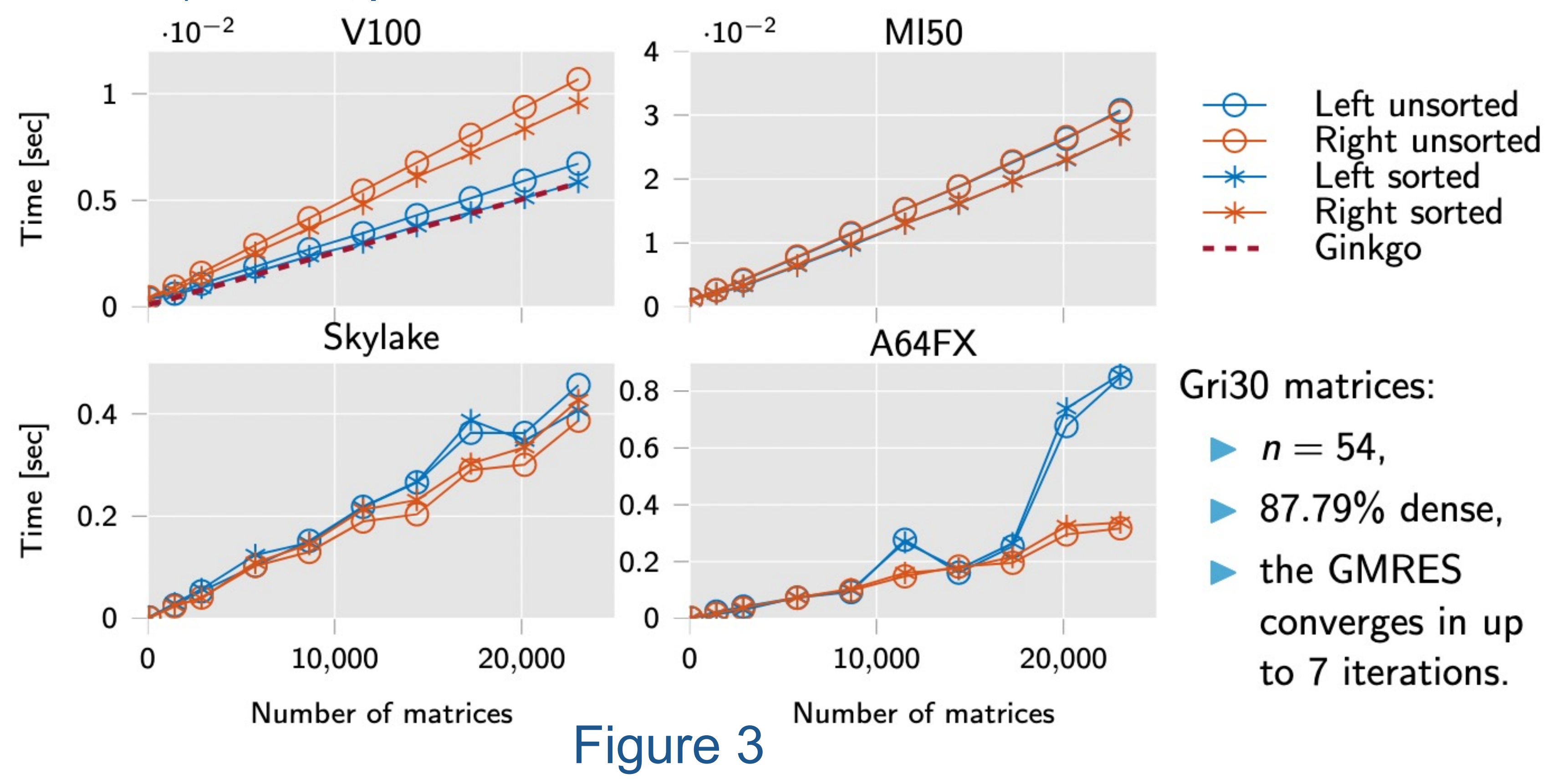

Figure 3

A double buffering technique has been implemented to improve the performance of our batched GEMM algorithm.
The technique improves the compute rate up to 1.25x. Good scaling is observed up to a size of 32, further improvements are possible for larger matrices.
Overall performance is comparable to other state of the art libraries see Figure 4.

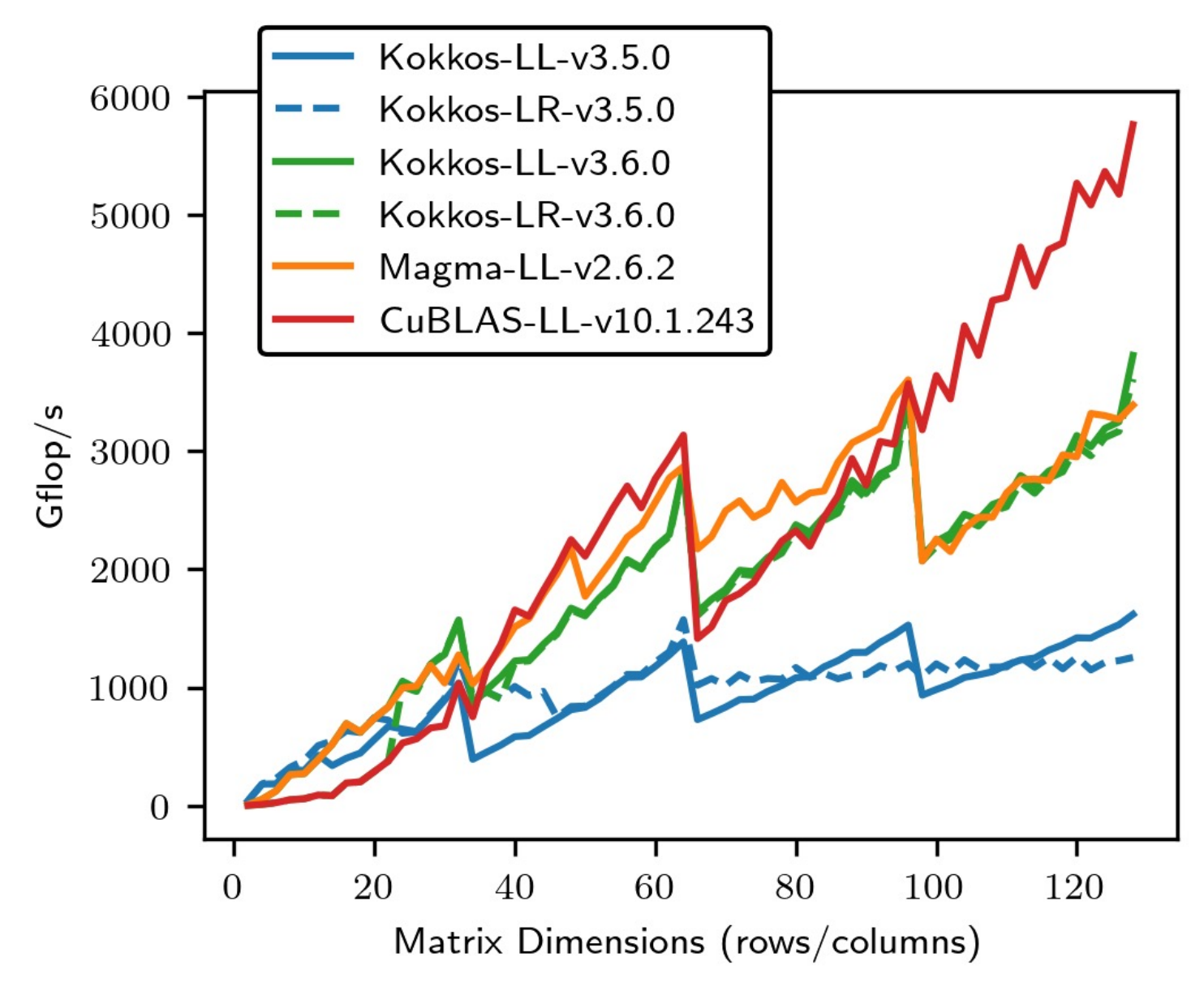

Figure 4

Mixed precision solvers implemented using iterative refinement have been tested (see Figure 2) and provide speed-up compared to uniform precision count-part. Experimentation with fp16 are also underway.

Finally support for block compressed sparse row matrices (BsrMatrix) has been added for the following algorithms:
• Block SpMV
• Block SpGEMM
• Block Gauss-Seidel

Future directions

Looking ahead, the following activities are planned for the upcoming 2022/23 year.

1. Additional capabilities in batched solvers (new preconditioners, data format)
2. Batched ODE integrators explicit and implicit
3. SYCL backend performance improvements and wider support for oneMKL (device MKL)
4. Uniform support of stream interface for BLAS algorithms
5. Unified interface for host/devices callable kernels

Additionally libraries/applications support, Spack package improvements and accessibility and performance on early access platforms work will continue.

ECP 6th Annual Meeting, May 2022

This research was supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration.