# Ginkgo

## A NODE-LEVEL SPARSE LINEAR ALGEBRA LIBRARY FOR HPC

### ECP PEEKS

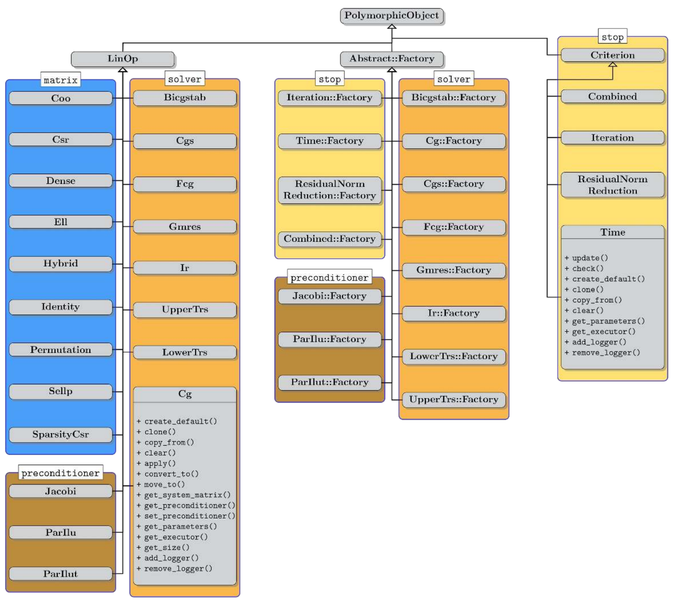
**The PEEKS Project part of ECP's CLOVER umbrella project aimed at delivering production-ready high performance preconditioned Krylov solvers for Exascale Computing. For these solvers to efficiently exploit extreme-scale hardware, both the solver algorithms and the implementations must be redesigned to address challenges like extreme concurrency, complex memory hierarchies, costly data movement, and heterogeneous node architectures. One important aspect is the implementation-readiness for GPU accelerators that are expected to provide a significant performance share in Exascale Computing. One product of the PEEKS project is the Ginkgo software library, which addresses the efficient use of GPU technology.**

### PERFORMANCE

Ginkgo is specifically designed to efficiently leverage the compute power of the latest hardware architectures. The performance evaluations on an AMD RadeonVII GPU and an NVIDIA Volta V100 GPU compares the performance of different sparse matrix vector kernels available in Ginkgo with counterparts of AMD's hipSPARSE[2] and NVIDIA's cuSPARSE[3] library. The performance profiles evaluate all test matrices available in the Suite Sparse matrix collection[4].

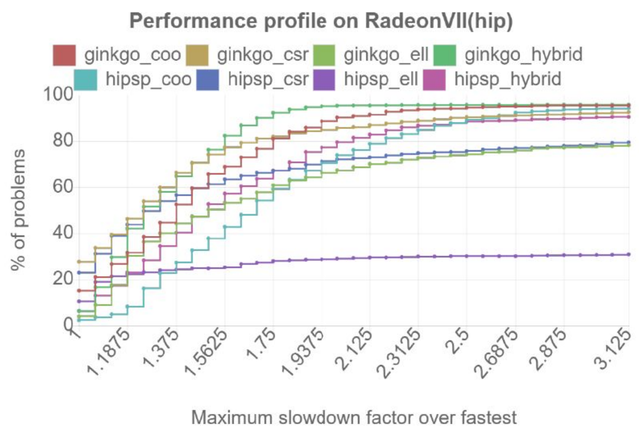
### DESIGN

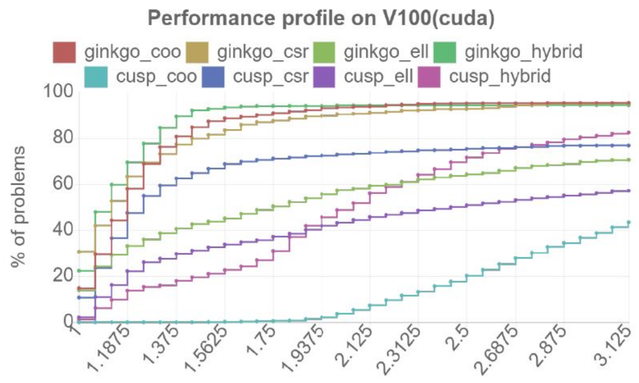
Ginkgo[1] is a C++ framework for sparse linear algebra. Using a universal linear operator abstraction, Ginkgo provides basic building blocks like the sparse matrix vector product for a variety of matrix formats, iterative solvers, and preconditioners. Ginkgo targets multi- and many-core systems, and currently features back-ends for AMD GPUs, NVIDIA GPUs, and OpenMP-supporting architectures. Runtime polymorphism is used to invoke the hardware-specific kernels, separating core functionality from these kernels allows for easy extension to other architectures.



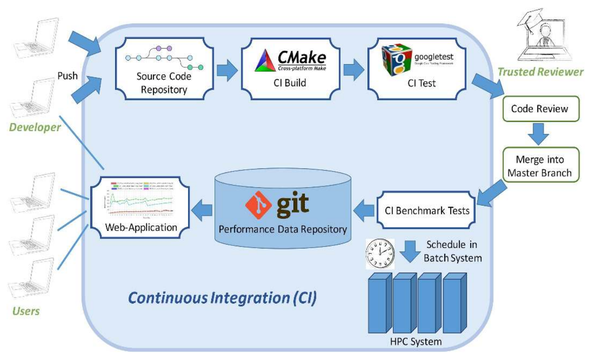

```cpp
#include <ginkgo.hpp>
#include <iostream>

int main()
{
    // Instantiate a CUDA executor
    auto gpu = gko::CudaExecutor::create(0, gko::OmpExecutor::create());
    // Read data
    auto A = gko::read<gko::matrix::Csr<>>(std::cin, gpu);
    auto b = gko::read<gko::matrix::Dense<>>(std::cin, gpu);
    auto x = gko::read<gko::matrix::Dense<>>(std::cin, gpu);
    // Create the solver
    auto solver =
        gko::solver::Cg<>::build()
            .with_preconditioner(gko::preconditioner::Jacobi<>::build().on(gpu))
            .with_criteria(
                gko::stop::Iteration::build().with_max_iters(20u).on(gpu),
                gko::stop::ResidualNormReduction<>::build()
                    .with_reduction_factor(1e-15)
                    .on(gpu))
            .on(gpu);
    // Solve system
    solver->generate(give(A))->apply(lend(b), lend(x));
    // Write result
    write(std::cout, lend(x));
}
```




### SUSTAINABLE SOFTWARE DEVELOPMENT




Ginkgo is part of the extreme-scale Software Development Kit (xSDK[6]), and deploys the xSDK community policies for sustainable software development and high software quality. The source code of the Ginkgo library can be accessed in a public git repository on GitHub. To preserve intellectual property of new developments, the public repository is mirrored into a repository hosted on GitLab that features private branches. Code development in Ginkgo is realized in a Continuous Integration / Continuous Benchmarking framework. GitLab runners are used on a private server where Docker images are used to provide different execution environments in terms of Compilers and Third-Party Libraries. To test the correct execution, each functionality is complemented by unit tests. The unit testing is realized using the Google Test framework.

### PERFORMANCE RESULTS DATABASE

All performance results of benchmark runs are archived in a publicly accessible performance data repository[5] based on git. Furthermore, the state of the machine, the environment, and even the compiled binaries are archived as GitLab artifacts. The intention is to not only provide users and developers with feedback about the performance of basic functionality, but also using performance data archiving to monitor the performance of central functionality over time to detect performance degradations.

### REFERENCES


1] Ginkgo: https://ginkgo-project.github.io/

2] hipSPARSE: https://github.com/ROCmSoftwarePlatform/hipSPARSE

3] cuSPARSE: https://docs.nvidia.com/cuda/cusparse/index.html

4] Suite Sparse Matrix Collection: https://sparse.tamu.edu/

5] Ginkgo Performance Explorer: https://ginkgo-project.github.io/gpe/

6] xSDK: https://xsdk.info



INNOVATIVE COMPUTING LABORATORY

THE UNIVERSITY OF TENNESSEE KNOXVILLE

Karlsruher Institut für Technologie

U.S. DEPARTMENT OF ENERGY | Office of Science

SPONSORED BY

EXASCALE COMPUTING PROJECT

National Nuclear Security Administration