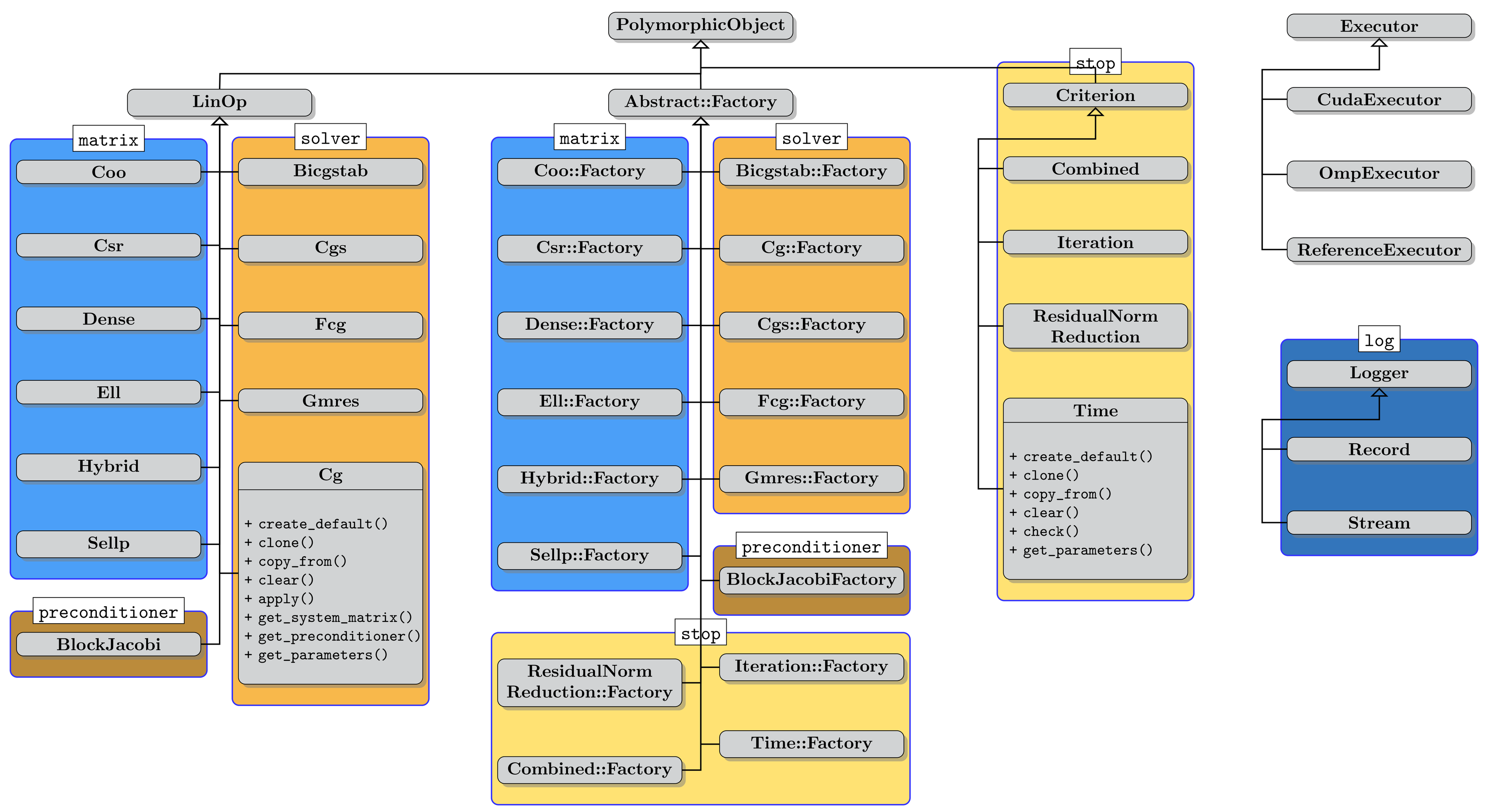
Ginkgo - A Linear Operator Library for Sparse Problems

Hartwig Anzt, Terry Cojean, YenChen Chen, Goran Flegar, Pratik Nayak, Yuhsiang M. Tsai, Enrique S. Quintana-Orti

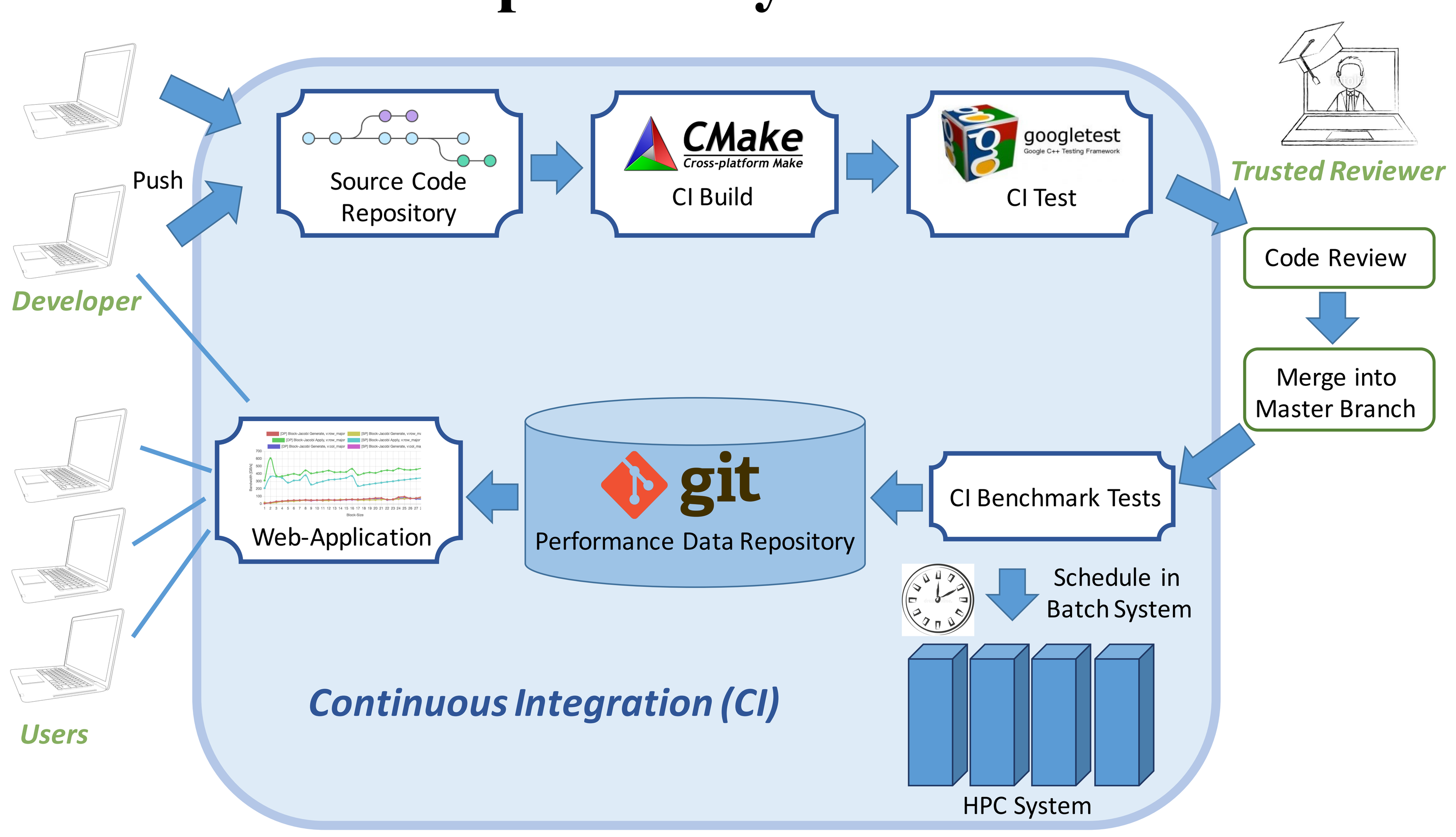
Karlsruhe Institute of Technology
The University of Tennessee Knoxville
Universitat Jaume I

Design of the Ginkgo Ecosystem

Ginkgo[1] is a C++ framework for sparse linear algebra. It 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 CUDA devices and OpenMP-supporting architectures. Runtime polymorphism is used to invoke the hardware-specific kernels. The library design separating core functionality from these kernels can easily be extended to other architectures. A template parameter specifying the types of elements in matrices and vectors enables to use of not only the IEEE754 standard floating point formats, but also customized precisions such as those provided by the FloatX[1] library.



Software Development Cycle

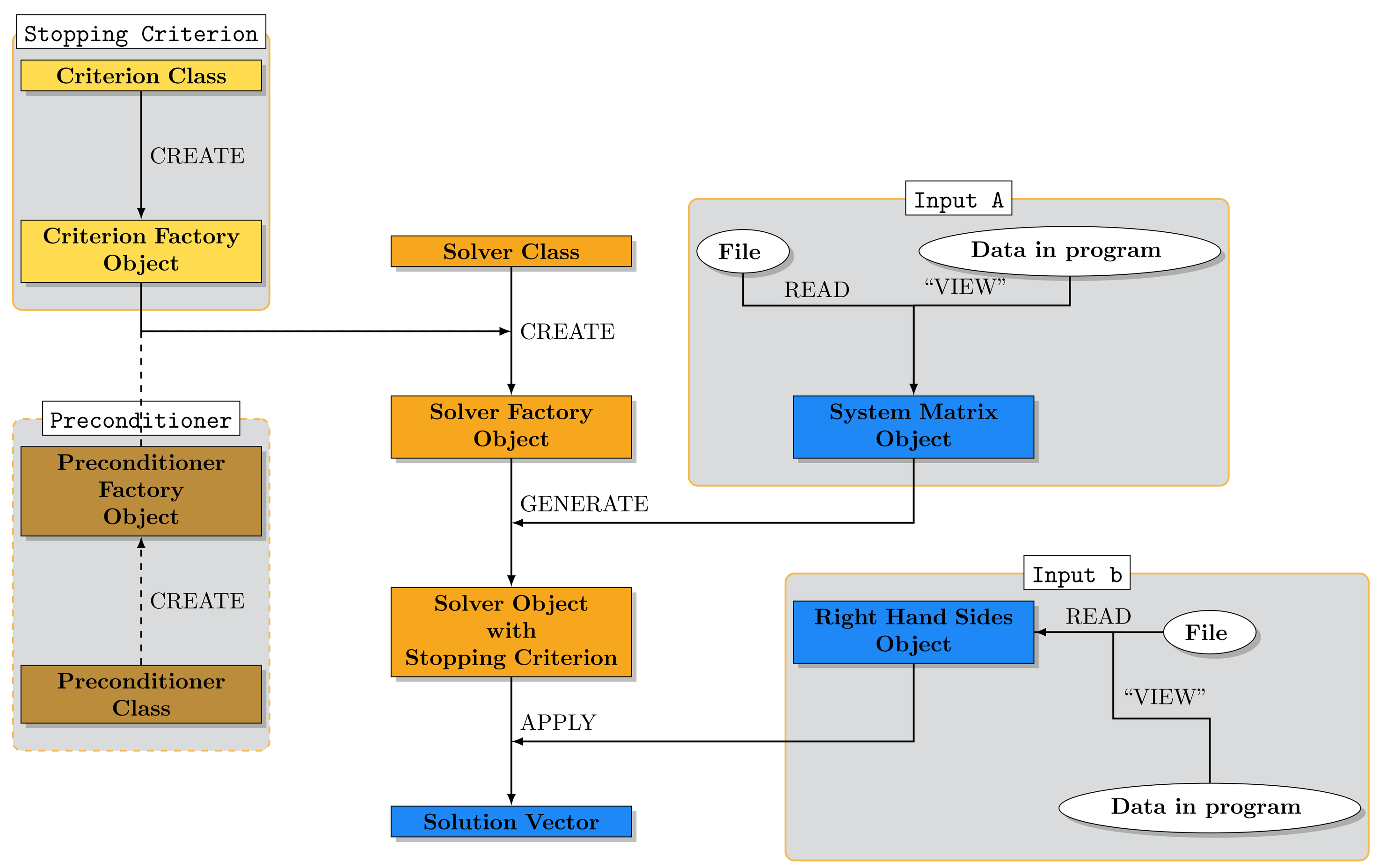


Code Example

This is a minimal example that uses Ginkgo to solve a linear system of equations. The system matrix, right-hand side and the initial guess are all read from the standard input. The system is solved on a GPU using the Conjugate Gradient (CG) method enhanced with a block-Jacobi preconditioner. Two stopping criteria are combined to limit the number of iterations and set the desired relative residual. The solution is written to the standard output.

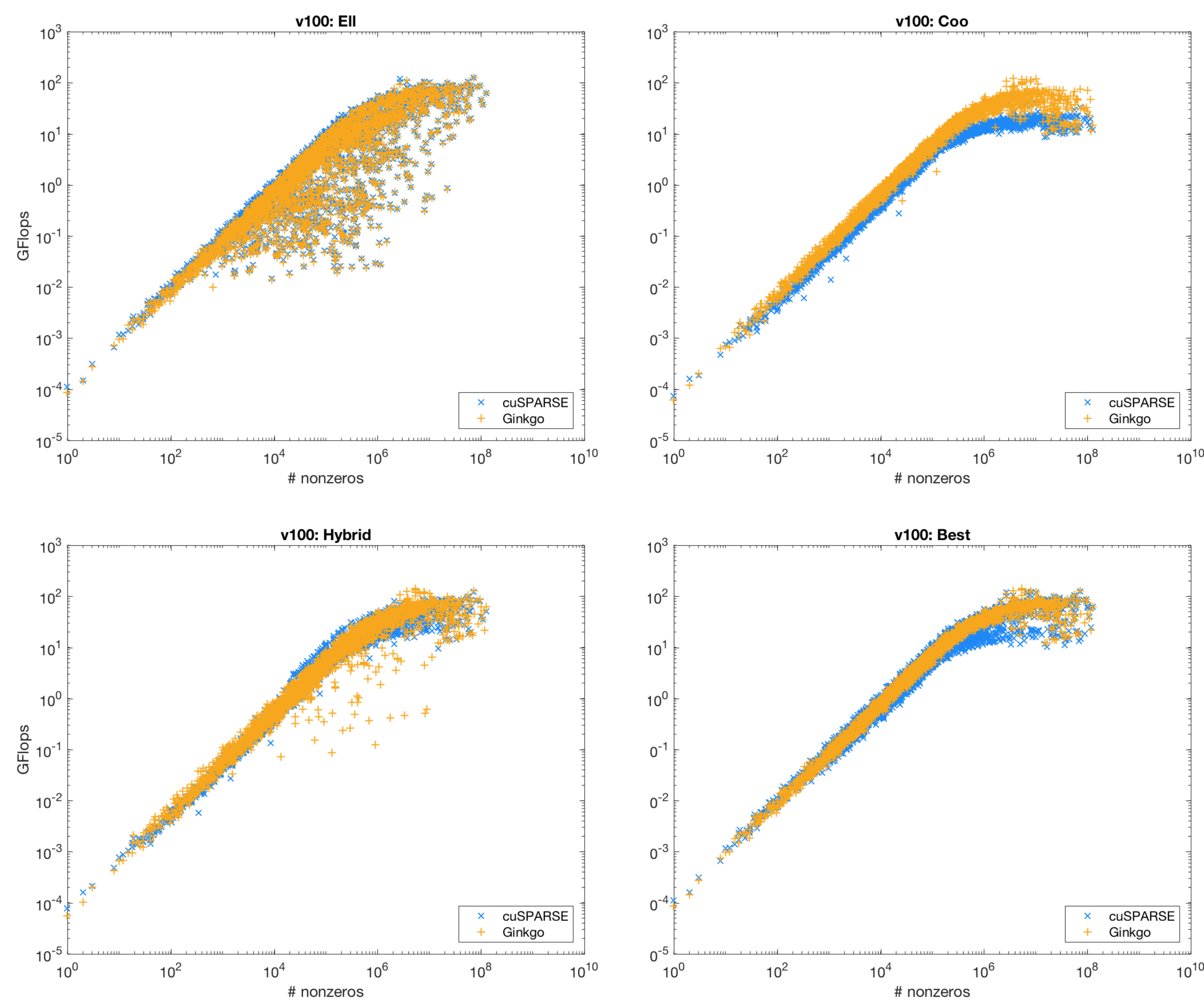
```
int main()
{
  // Instantiate a CUDA executor
  auto exec = gko::CudaExecutor::create(0, gko::OmpExecutor::create());
  // Read data
  auto A = gko::read<gko::matrix::Csr<>>(std::cin, exec);
  auto b = gko::read<gko::matrix::Dense<>>(std::cin, exec);
  auto x = gko::read<gko::matrix::Dense<>>(std::cin, exec);
  // Create the solver
  auto solver = gko::solver::Cg<>::Factory::create()
    .with_preconditioner(
      gko::preconditioner::BlockJacobiFactory<>::create(exec, 32))
    .with_criterion(gko::stop::Combined::Factory::create()
      .with_criteria(
        gko::stop::Iteration::Factory::create()
          .with_max_iters(20u)
          .on_executor(exec),
        gko::stop::ResidualNormReduction<>::Factory::create()
          .with_reduction_factor(1e-15)
          .on_executor(exec))
      .on_executor(exec))
    .on_executor(exec);
  // Solve system
  solver->generate(give(A))->apply(lend(b), lend(x));
  // Write result
  write(std::cout, lend(x));
}
```

The general workflow for solving a linear system is:



Performance Evaluation

Ginkgo is specifically designed to efficiently leverage the compute power of the latest hardware architectures. The performance evaluation on an NVIDIA Volta V100 GPU compares the performance of different sparse matrix vector kernels available in Ginkgo with counterparts of NVIDIA's cuSPARSE[3] library. The test matrices coming from problems in computational science, circuit design problems, optimization, and big data analytics are taken from the Suite Sparse matrix collection[4].



References

[1]Float eXtended, FloatX: https://github.com/oprecomp/FloatX
[2]xSDK: Extreme-scale Scientific Software Development Kit: https://xsdk.info/
[3]NVIDIA cuSPARSE library: https://docs.nvidia.com/cuda/cusparse/index.html/
[4]The Suite Sparse matrix collection: https://sparse.tamu.edu/

https://github.com/ginkgo-project/ginkgo

https://ginkgo-project.github.io/gpe/