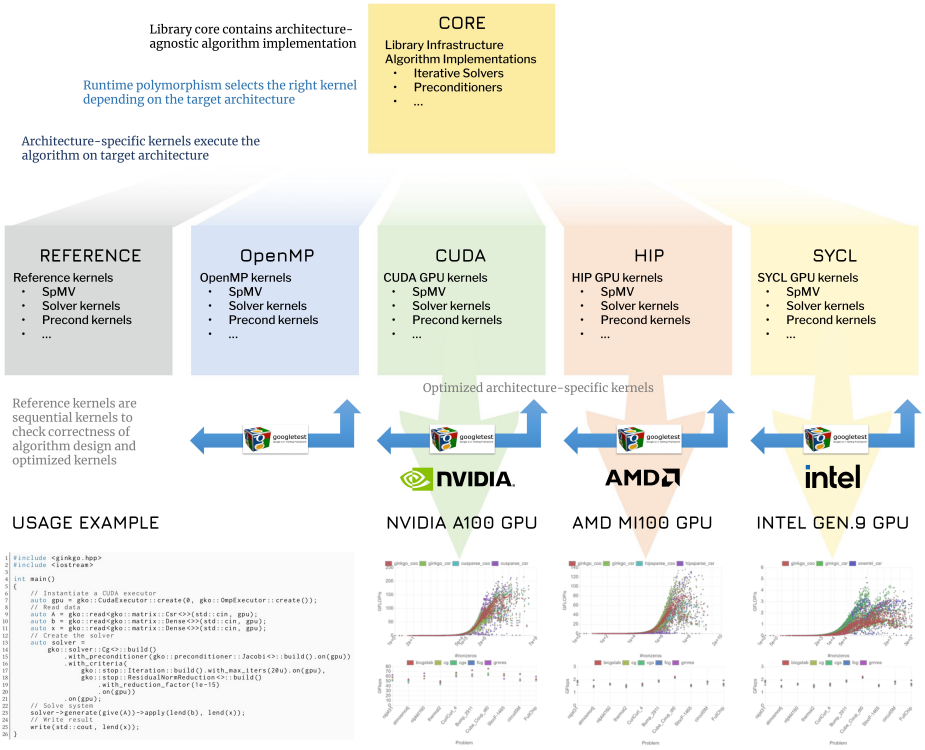
Ginkgo

FIND OUT MORE AT https://github.com/ginkgo-project/ginkgo

DESIGN

Ginkgo is a C++ framework for sparse numerical linear algebra. Using a universal linear operator abstraction, Ginkgo provides basic building blocks such as 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, Intel CPU/GPUs, NVIDIA GPUs, and OpenMP-supporting architectures. Core functionality is separated from hardware-specific kernels for easy extension to other architectures, with runtime polymorphism selecting the correct kernels.



SUSTAINABILITY

Ginkgo is part of the Extreme-scale Scientific Software Stack (E4S) and the extreme-scale Software Development Kit (xSDK), and adopts 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. Code development in Ginkgo is realized in a Continuous Integration / Continuous Benchmarking framework. GitLab runners are used on private servers and HPC clusters with Docker/Enroot providing different compilation and execution environments. Unit tests cover all functionality and use the Google Test framework.

SPONSORED BY

Exascale Computing Project

EuroHPC Joint Undertaking

Federal Ministry of Education and Research

HELMHOLTZ RESEARCH FOR GRAND CHALLENGES

HARDWARE PARTNERS

AMD

intel

NVIDIA

INTEGRATION

deal.II

OpenFOAM

sundials

HYTEG

OpenCARP

Spack

xSDK

EESSI

EasyBuild

E4S

preCICE

Karlsruher Institut für Technologie

Technical University of Munich

THE UNIVERSITY OF TENNESSEE KNOXVILLE

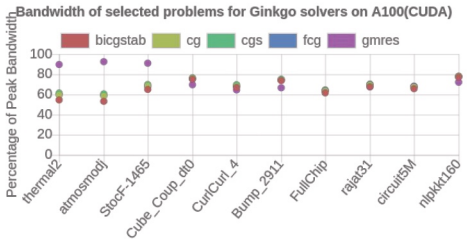
This research was supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration, the Horizon2020 EuroHPC Project MICROCARD, and the Helmholtz Impuls und Vernetzungsfond VH-NG-1241.

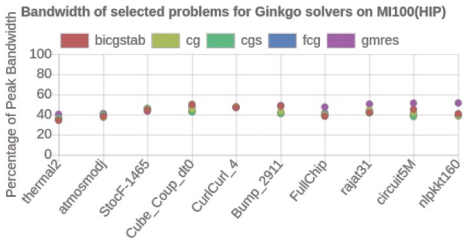
Ginkgo

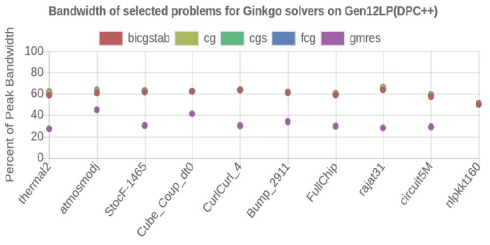
FIND OUT MORE AT https://github.com/ginkgo-project/ginkgo

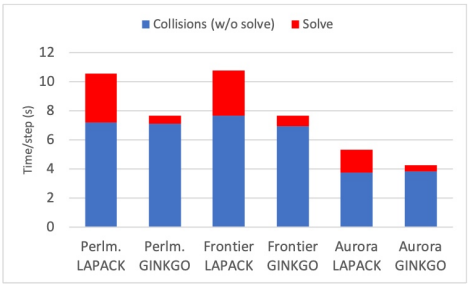
PERFORMANCE

NVIDIA AMD intel





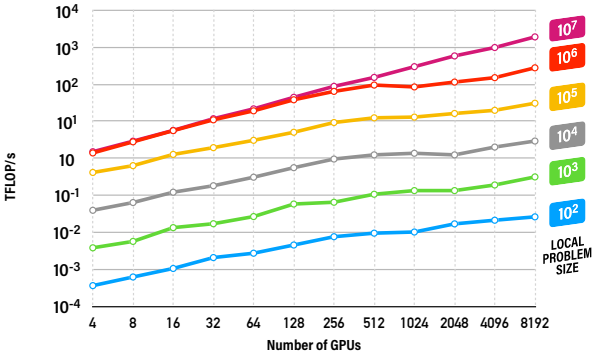




For plasma simulations with the X-point included Gyrokinetic Code (XGC), Ginkgo's batched iterative solvers reduce linear solve time by ~85%, 80%, and 70% on NERSC Perlmutter (AMD EPYC CPU / 4 NVIDIA A100 GPUs per node), OLCF Frontier (AMD EPYC CPU / 4 AMD MI250X GPUs per node), and ALCF Aurora (dual-socket Intel Max CPU / 6 Intel Max 1550 GPUs per node).

Weak scaling of distributed sparse matrix-vector (SpMV) multiplication

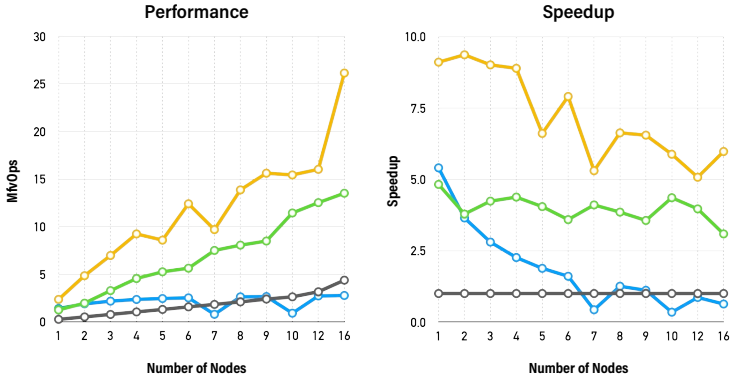
on Jülich Supercomputing Centre JUPITER (4 NVIDIA Grace Hopper GH200s per node)



Accelerating an OpenFOAM 3D lid-driven cavity simulation with 630^3 total cells

2 Intel Xeon Platinum 8368/ 4 NVIDIA A100s per node

CPU OpenFOAM CPU reference · GPUURR1 1 rank/GPU · GPUOSR1 18 ranks/GPU · GPUOSRI6 72 ranks CPU assembly, 1 rank/GPU solve



FUNCTIONALITY

Routines available in IEEE fp64/fp32/fp16, bfloat16, and complex variants.

		OMP	CUDA	HIP	SYCL
Basic	SpMV	✓	✓	✓	✓
	SpMM	✓	✓	✓	✓
	SpGeMM	✓	✓	✓	✓
Krylov solvers	BiCG	✓	✓	✓	✓
	BiCGSTAB	✓	✓	✓	✓
	CG	✓	✓	✓	✓
	CGS	✓	✓	✓	✓
	GCR	✓	✓	✓	✓
	GMRES	✓	✓	✓	✓
	FCG	✓	✓	✓	✓
	FGMRES	✓	✓	✓	✓
	IR	✓	✓	✓	✓
	IDR	✓	✓	✓	✓
	Chebyshev	✓	✓	✓	✓
	MinRES	✓	✓	✓	✓
Preconditioners	Block-Jacobi	✓	✓	✓	✓
	Gauss-Seidel/SSOR	✓	✓	✓	✓
	ILU/IC	✓	✓	✓	
	Parallel ILU/IC	✓	✓	✓	✓
	Parallel ILUT/ICT	✓	✓	✓	✓
	ISAI	✓	✓	✓	✓
Batched	Batched BiCGSTAB	✓	✓	✓	✓
	Batched CG	✓	✓	✓	✓
	Batched GMRES	✓	✓	✓	✓
	Batched ILU	✓	✓	✓	✓
	Batched ISAI	✓	✓	✓	✓
	Batched Block-Jacobi	✓	✓	✓	✓
AMG	AMG preconditioner	✓	✓	✓	✓
	AMG solver	✓	✓	✓	✓
	Parallel Graph Match	✓	✓	✓	✓
Sparse direct	Symbolic Cholesky	✓	✓	✓	✓
	Numeric Cholesky	✓	✓	✓	
	Symbolic LU	✓	✓	✓	✓
	Numeric LU	✓	✓	✓	
	Sparse TRSV	✓	✓	✓	
Utilities	On-Device Matrix Assembly	✓	✓	✓	✓
	MC64/RCM reordering	✓	✓	✓	✓
	Wrapping user data	✓	✓	✓	✓
	Logging	✓	✓	✓	✓
	PAPI counters (SDE)	✓	✓	✓	✓
	Config file support	✓	✓	✓	✓

✓ *MPI Support* ✓ *Single-GPU Support*