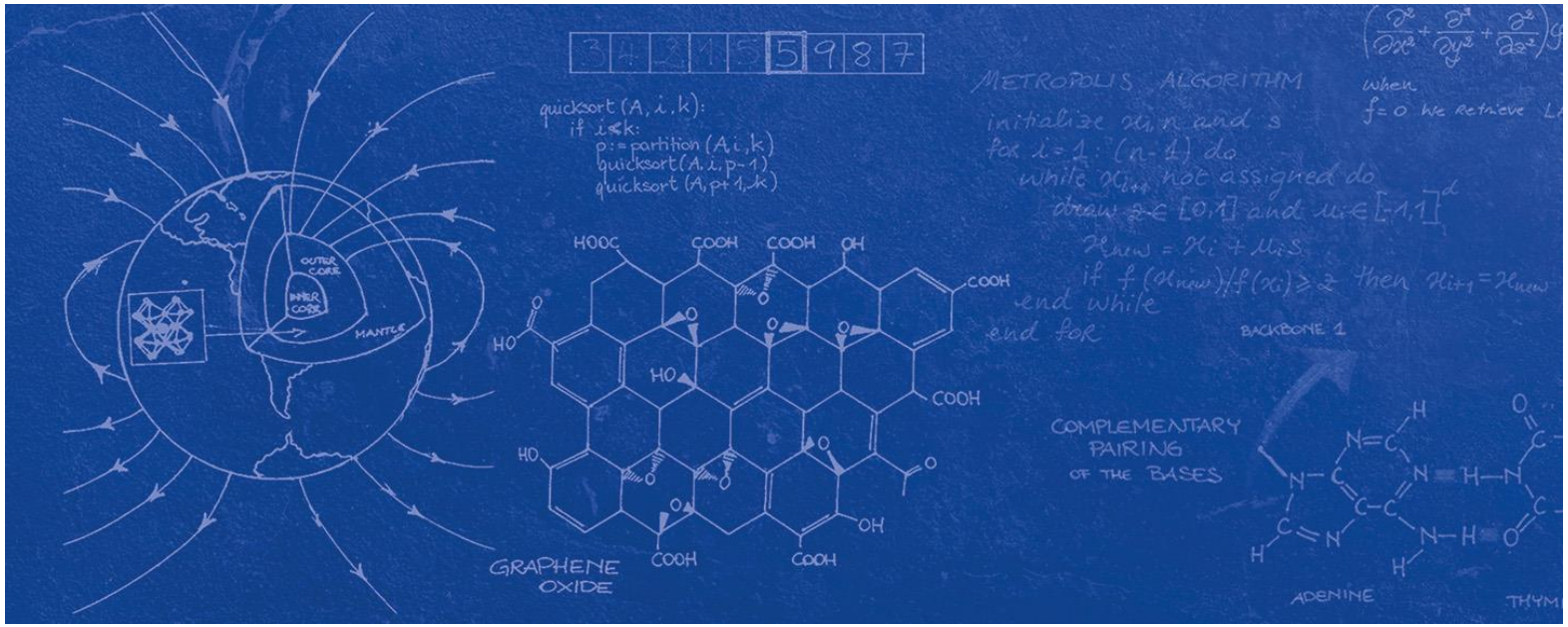
PASC 24
Zurich Switzerland | 3-5 June 2024

Enhancing GPU-Accelerated Scientific Computing in Julia with Ginkgo.jl



You Wu, Tobias Ribizel, and Hartwig Anzt

05-06-2024

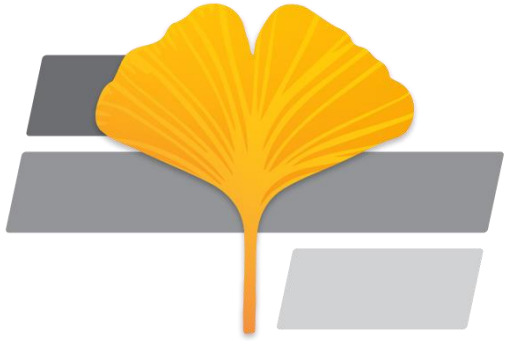
Table of Content

PASC 24
Zurich Switzerland 3-5 June 2024

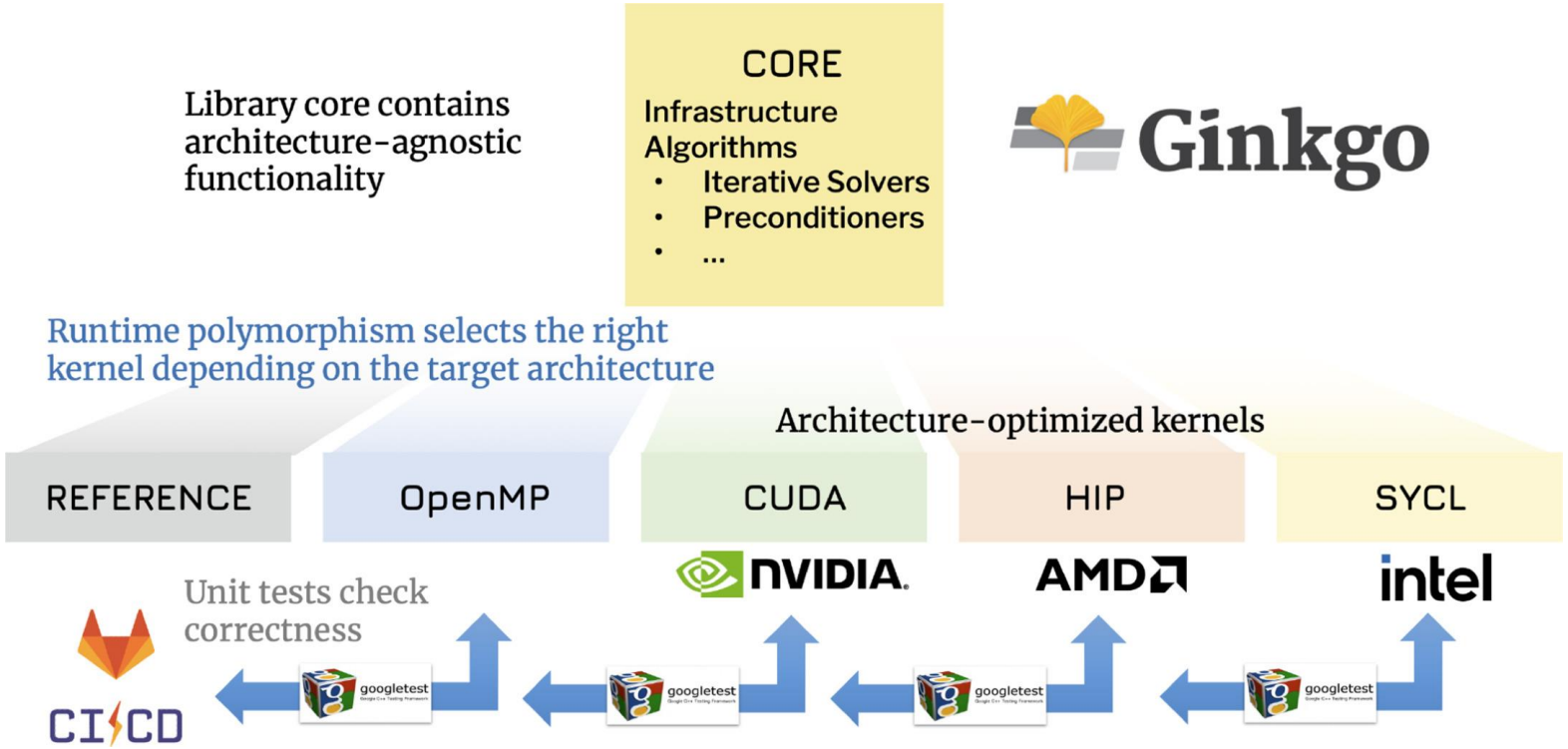
Ginkgo: A Modern Linear Operator Algebra Framework for HPC

Ginkgo

- C++ framework for sparse numerical linear algebra
- Implemented using modern C++
- Part of the *Extreme-Scale Scientific Software Stack (E4S)* and the *Extreme-Scale Software Development Kit (xSDK)*
- Emphasis on solving sparse linear systems efficiently on GPU-accelerated systems, using backends written in the respective vendor languages



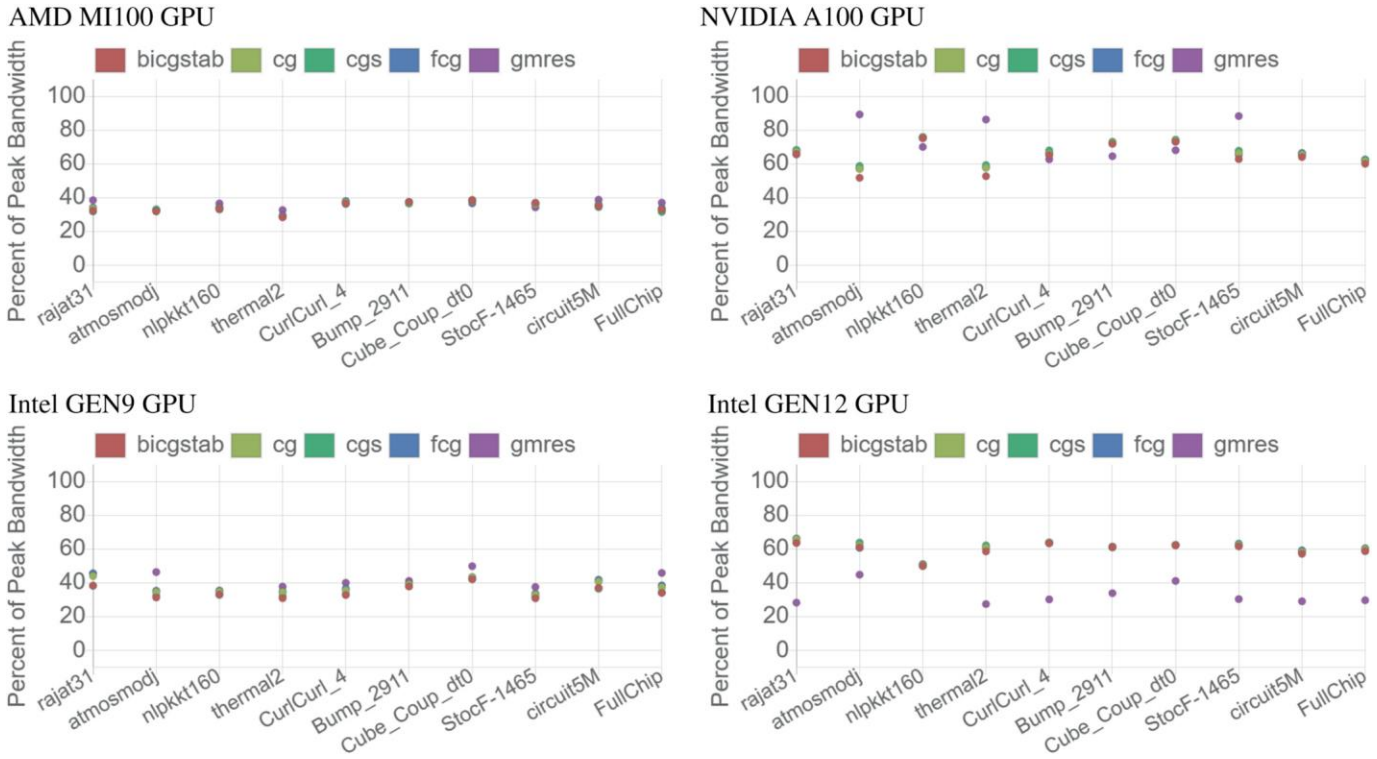
Designed for Platform Portability



	FUNCTIONALITY	OMP	CUDA	HIP	DPC++
Basic	SpMV	✓	✓	✓	✓
	SpMM	✓	✓	✓	✓
	SpGeMM	✓	✓	✓	✓
Krylov solvers	BiCG	✓	✓	✓	✓
	BiCGSTAB	✓	✓	✓	✓
	CG	✓	✓	✓	✓
	CGS	✓	✓	✓	✓
	GCR	✓	✓	✓	✓
	GMRES	✓	✓	✓	✓
	FCG	✓	✓	✓	✓
	FGMRES	✓	✓	✓	✓
	IR	✓	✓	✓	✓
	IDR	✓	✓	✓	✓
Preconditioners	Block-Jacobi	✓	✓	✓	✓
	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	✓	✓	✓	✓

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

Designed for Performance

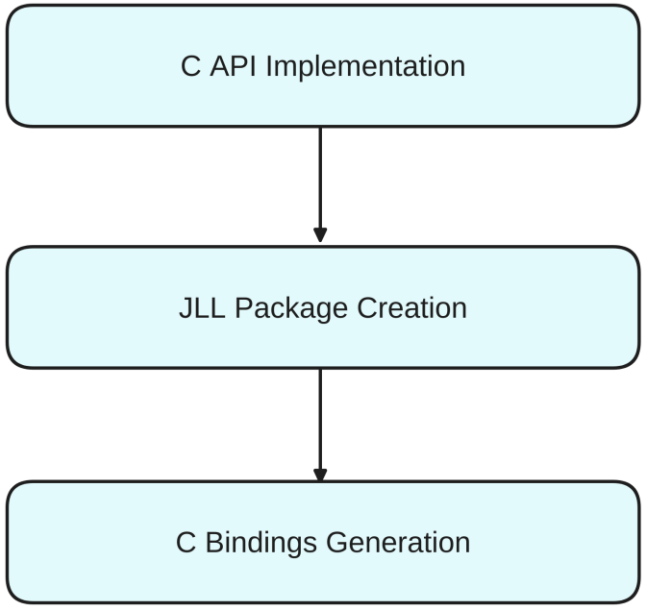


[1] Tsai Y-HM, Cojean T, Anzt H. Providing performance portable numerics for Intel GPUs.

C API & Julia Bindings

C API & Maintainable Software Development

- Implementation of a C API is more versatile and can be extended to support other languages
- Using packages such as CxxWrap.jl adds on an extra dependencies
- Calling C++ library in Julia consists of mainly three parts:



C API Implementation

- Uses the language linkage with key word **extern "c"**
- Difficulty lies in "translating" back modern C++ syntax back to C
- Appropriate use of macros largely avoid code duplication
- Naming convention should be well-established

```
// c_api.h
#define DECLARE_CSR_OVERLOAD(_ctype_value, _ctype_index, _cpptype_value,       \
                             _cpptype_index, _name, _name_dense)               \
    struct gko_matrix_csr_##_name##_st;                                        \
    typedef struct gko_matrix_csr_##_name##_st* gko_matrix_csr_##_name;        \
    gko_matrix_csr_##_name ginkgo_matrix_csr_##_name##_create(                 \
        gko_executor exec, gko_dim2_st size, size_t nnz);                      \
    gko_matrix_csr_##_name ginkgo_matrix_csr_##_name##_create_view(            \
        gko_executor exec, gko_dim2_st size, size_t nnz,                       \
        _ctype_index* row_ptrs, _ctype_index* col_idxs, _ctype_value* values); \
    void ginkgo_matrix_csr_##_name##_delete(                                   \
        gko_matrix_csr_##_name mat_st_ptr);                                    \
    gko_matrix_csr_##_name ginkgo_matrix_csr_##_name##_read(                   \
        const char* str_ptr, gko_executor exec);                               \
    void ginkgo_write_csr_##_name##_in_coo(const char* str_ptr,                \
                                           gko_matrix_csr_##_name mat_st_ptr); \
    size_t ginkgo_matrix_csr_##_name##_get_num_stored_elements(                \
        gko_matrix_csr_##_name mat_st_ptr);                                    \
    size_t ginkgo_matrix_csr_##_name##_get_num_srow_elements(                  \
        gko_matrix_csr_##_name mat_st_ptr);                                    \
    gko_dim2_st ginkgo_matrix_csr_##_name##_get_size(                          \
        gko_matrix_csr_##_name mat_st_ptr);                                    \
    const _ctype_value* ginkgo_matrix_csr_##_name##_get_const_values(          \
        gko_matrix_csr_##_name mat_st_ptr);                                    \
    const _ctype_index* ginkgo_matrix_csr_##_name##_get_const_col_idxs(        \
        gko_matrix_csr_##_name mat_st_ptr);                                    \
```

```
// c_api.h
DECLARE_CSR_OVERLOAD(float, int, float, int, f32_i32, f32)
DECLARE_CSR_OVERLOAD(float, int64_t, float, std::int64_t, f32_i64, f32)
DECLARE_CSR_OVERLOAD(double, int, double, int, f64_i32, f64)
DECLARE_CSR_OVERLOAD(double, int64_t, double, std::int64_t, f64_i64, f64)
```

Similar construct holds for DEFINE_X_OVERLOAD that resides in c_api.cpp

C API Implementation

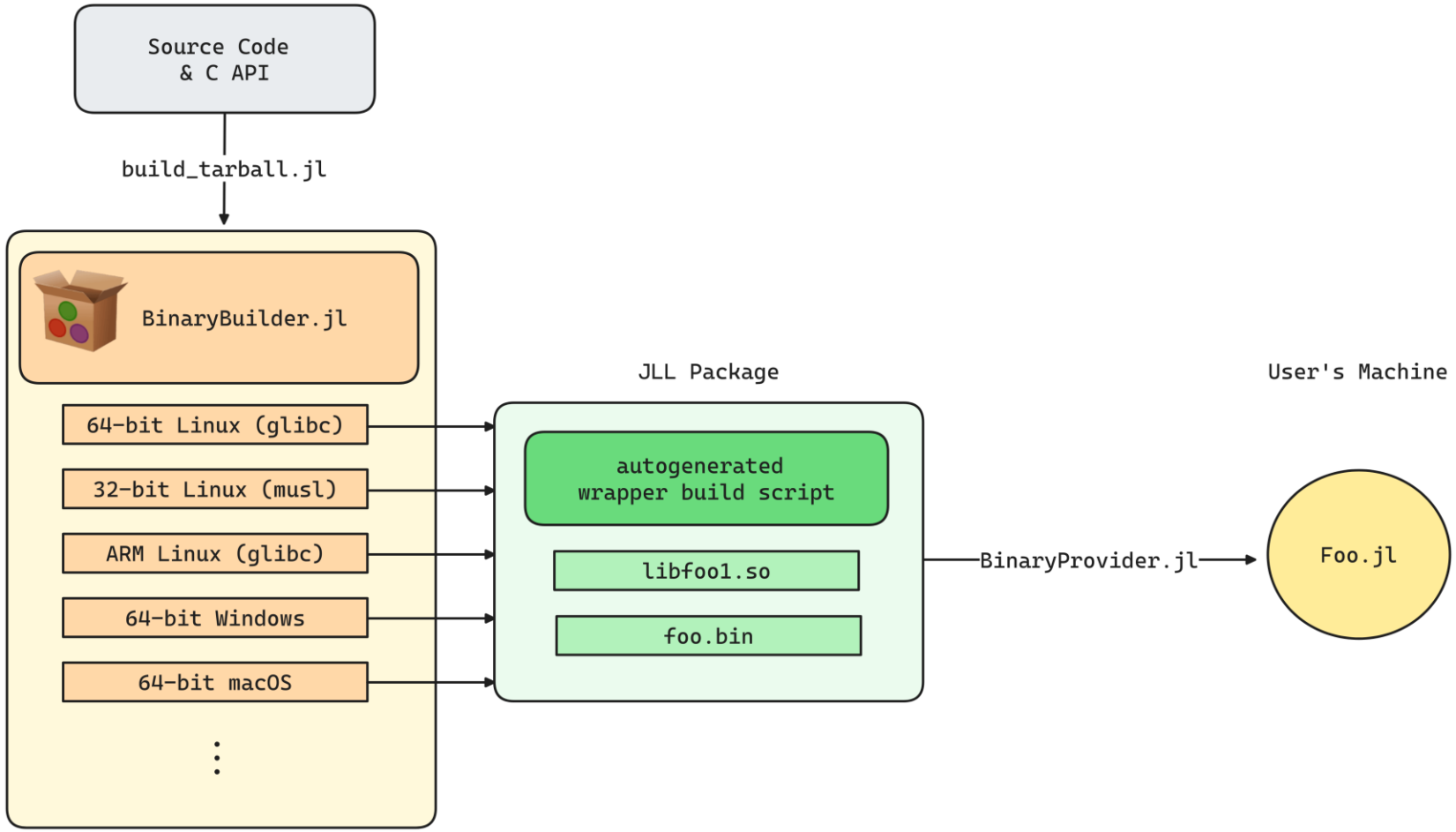

```
// c_api.cpp
struct gko_linop_st {
    std::shared_ptr<gko::LinOp> shared_ptr;
};

void ginkgo_linop_delete(gko_linop linop_st_ptr) { delete linop_st_ptr; }

void ginkgo_linop_apply(gko_linop solver, gko_linop b_st_ptr, gko_linop x_st_ptr)
{ (solver->shared_ptr)->apply(b_st_ptr->shared_ptr, x_st_ptr->shared_ptr); }

// Solver as a LinOp object
gko_linop ginkgo_linop_gmres_preconditioned_f64_create(
    gko_executor exec_st_ptr, gko_linop A_st_ptr,
    gko_deferred_factory_parameter dfp_st_ptr, double reduction, int maxiter)
{
    return new gko_linop_st{
        gko::solver::Gmres<double>::build()
            .with_criteria(
                gko::stop::Iteration::build().with_max_iters(maxiter),
                gko::stop::ResidualNorm<double>::build().with_reduction_factor(
                    reduction))
            .with_preconditioner(dfp_st_ptr->parameter)
            .on(exec_st_ptr->shared_ptr)
            ->generate(A_st_ptr->shared_ptr)};
}
```

Example: Creation of a preconditioned GMRES solver with the C API

JLL Package Creation

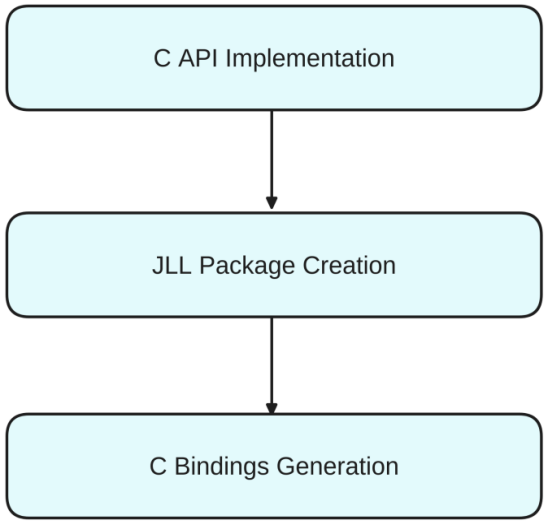
- JLL packages live under **JuliaBinaryWrappers** organization
- With build scripts hosted under **JuliaPackaging/Yggdrasil**
- **BinaryBuilder.jl** is a convenient tool for creating suitable build scripts and creating pull requests to official Julia registry for JLL packages

JLL Package Creation



C Bindings Generation

```
# An example wrap.jl file for generating src/api.jl
using Clang.Generators

import ginkgo_jll
header_dir = joinpath(ginkgo_jll.artifact_dir, "include")
isdir(header_dir) || error("$header_dir does not exist")

# wrapper generator options
options = load_options(joinpath(@__DIR__, "wrap.toml"))

# add compiler flags, e.g. "-DXXXXXXXXX"
args = get_default_args()
push!(args, "-I$header_dir")
push!(args, "-DBUILD_SHARED_LIBS")

# specifying C API header to parse
headers = [joinpath(header_dir, header) for header in readdir(header_dir) if
startswith(header, "c_api")]

# create context
ctx = create_context(headers, args, options)

# run generator
build!(ctx)
```



High-Level API Design

Code Comparison

```julia
using Ginkgo

# Type alias
const (Tv, Ti) = (Float64, Int32)

# Print ginkgo library version
version()

# Creates executor for a specific backend
const exec = create(:cuda)

# Specify executor to be passed
with(EXECUTOR => exec) do
  # Read matrix and vector from mtk files, now omit the passing of exec
  A = GkoCsr{Tv, Ti}("data/A.mtx");
  b = GkoDense{Tv}("data/b.mtx");
  x = GkoDense{Tv}("data/x0.mtx");

  solver = GkoIterativeSolver(:cg, A; maxiter = 20, reduction = 1.0e-7)
  apply!(solver, b, x)

  @info "Solution (x):"
  display(x)

  one     = number(Tv(1.0))
  neg_one = number(Tv(-1.0))
  res     = number(Tv(0.0))

  apply!(A, one, x, neg_one, b)
  norm2!(b, res)
  @info "Residual norm sqrt(r^T r):"
  display(res)
end
```

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

int main(int argc, char* argv[])
{
    // Type alias
    using Tv = double;
    using Ti = int;
    using vec = gko::matrix::Dense<Tv>;

    // Print ginkgo library version
    std::cout << gko::version_info::get() << std::endl;

    // Creates executor for a specific backend
    const auto exec = gko::CudaExecutor::create(0, gko::OmpExecutor::create());

    // Read matrix and vector from mtk files, now omit the passing of exec
    auto A = gko::share(gko::read<gko::matrix::Csr<Tv, Ti>>
                        (std::ifstream("data/A.mtx"), exec));
    auto b = gko::read<vec>(std::ifstream("data/b.mtx"), exec);
    auto x = gko::read<vec>(std::ifstream("data/x0.mtx"), exec);

    auto solver_gen =
         gko::solver::Cg<Tv>::build()
            .with_criteria(gko::stop::Iteration::build().with_max_iters(20u),
                           gko::stop::ResidualNorm<Tv>::build()
                               .with_reduction_factor(1e-7))
            .on(exec);
    auto solver = solver_gen->generate(A);
    solver->apply(b, x);

    std::cout << "Solution (x):\n";
    write(std::cout, x);

    auto one = gko::initialize<vec>({1.0}, exec);
    auto neg_one = gko::initialize<vec>({-1.0}, exec);
    auto res = gko::initialize<gko::matrix::Dense<Tv>>({0.0}, exec);
    A->apply(one, x, neg_one, b);
    b->compute_norm2(res);

    std::cout << "Residual norm sqrt(r^T r):\n";
    write(std::cout, res);
}
```

Convenient Switch of Underlying Binaries

Using underlying binaries with user-defined location

```
julia> Ginkgo.GkoPreferences.locate()
┌ Info: Location of binaries being used by Ginkgo.jl
│   ginkgo_jll.find_artifact_dir() = "/scratch/wyou/.julia/artifacts/b93004147661eb53ee77f59672c6900759a42219"
│   ginkgo_jll.get_libginkgo_path() = "/home/wyou/misc/pasc/ginkgo_forked/ginkgo/build/lib/libginkgo.so"
│   ginkgo_jll.get_libginkgo_reference_path() = "/home/wyou/misc/pasc/ginkgo_forked/ginkgo/build/lib/libginkgo_reference.so"
│   ginkgo_jll.get_libginkgo_omp_path() = "/home/wyou/misc/pasc/ginkgo_forked/ginkgo/build/lib/libginkgo_omp.so"
│   ginkgo_jll.get_libginkgo_cuda_path() = "/home/wyou/misc/pasc/ginkgo_forked/ginkgo/build/lib/libginkgo_cuda.so"
│   ginkgo_jll.get_libginkgo_hip_path() = "/home/wyou/misc/pasc/ginkgo_forked/ginkgo/build/lib/libginkgo_hip.so"
│   ginkgo_jll.get_libginkgo_dpcpp_path() = "/home/wyou/misc/pasc/ginkgo_forked/ginkgo/build/lib/libginkgo_dpcpp.so"
└   ginkgo_jll.get_libginkgo_device_path() = "/home/wyou/misc/pasc/ginkgo_forked/ginkgo/build/lib/libginkgo_device.so"

julia> Ginkgo.GkoPreferences.use_jll_binary!()
┌ Warning: ginkgo library paths changed, you will need to restart Julia for the change to take effect
│   ginkgo_jll.find_artifact_dir() = "/scratch/wyou/.julia/artifacts/b93004147661eb53ee77f59672c6900759a42219"
└ @ Ginkgo.GkoPreferences ~/misc/pasc/Ginkgo.jl/src/preferences.jl:127
```

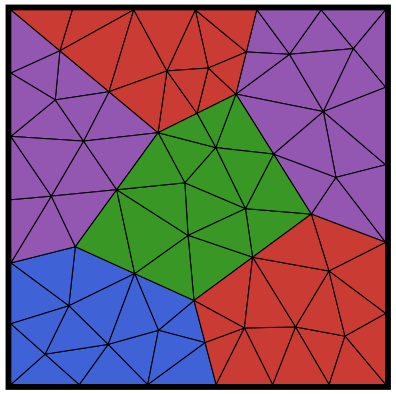
Using underlying binaries from the JLL package

```
julia> using Ginkgo; Ginkgo.GkoPreferences.locate()
┌ Info: Location of binaries being used by Ginkgo.jl
│   ginkgo_jll.find_artifact_dir() = "/scratch/wyou/.julia/artifacts/b93004147661eb53ee77f59672c6900759a42219"
│   ginkgo_jll.get_libginkgo_path() = "/scratch/wyou/.julia/artifacts/b93004147661eb53ee77f59672c6900759a42219/lib/libginkgo.so"
│   ginkgo_jll.get_libginkgo_reference_path() = "/scratch/wyou/.julia/artifacts/b93004147661eb53ee77f59672c6900759a42219/lib/libginkgo_reference.so"
│   ginkgo_jll.get_libginkgo_omp_path() = "/scratch/wyou/.julia/artifacts/b93004147661eb53ee77f59672c6900759a42219/lib/libginkgo_omp.so"
│   ginkgo_jll.get_libginkgo_cuda_path() = "/scratch/wyou/.julia/artifacts/b93004147661eb53ee77f59672c6900759a42219/lib/libginkgo_cuda.so"
│   ginkgo_jll.get_libginkgo_hip_path() = "/scratch/wyou/.julia/artifacts/b93004147661eb53ee77f59672c6900759a42219/lib/libginkgo_hip.so"
│   ginkgo_jll.get_libginkgo_dpcpp_path() = "/scratch/wyou/.julia/artifacts/b93004147661eb53ee77f59672c6900759a42219/lib/libginkgo_dpcpp.so"
└   ginkgo_jll.get_libginkgo_device_path() = "/scratch/wyou/.julia/artifacts/b93004147661eb53ee77f59672c6900759a42219/lib/libginkgo_device.so"
```

Numerical Experiments

Interoperability with Existing Packages

- Made possible by supporting implicit conversion of data types
- **Ferrite.jl** is a simple FEM toolbox written in Julia
- Using **Ferrite.jl** for FEM-based matrix assembly, and **Ginkgo.jl** for sparse LSE solution with a CG solver



2D Heat Equation on Unit Square

$$\begin{cases} -\nabla \cdot (k \nabla u) = f, x \in \Omega \\ u(x) = 0 \ , x \in \partial\Omega \end{cases} \quad \Longleftrightarrow \quad \int_\Omega \nabla \delta u \cdot \nabla u \, d\Omega = \int_\Omega \delta u \, d\Omega \quad \forall \delta u \in \mathbb{T},$$

```
################## Using Ferrite for Matrix Assembly ####################
grid = generate_grid(Quadrilateral, (20, 20));

# Trial and test functions
dim = 2
ip = Lagrange{dim, RefCube, 1}()
qr = QuadratureRule{dim, RefCube}(2)
cellvalues = CellScalarValues(qr, ip);

# Degrees of freedom
dh = DofHandler(grid)
add!(dh, :u, 1)
close!(dh);

# Creates SparseMatrixCSC
K = create_sparsity_pattern(dh)

# Essential BC
ch = ConstraintHandler(dh);
∂Ω = union(
    getfaceset(grid, "left"),
    getfaceset(grid, "right"),
    getfaceset(grid, "top"),
    getfaceset(grid, "bottom"),
);
dbc = Dirichlet(:u, ∂Ω, (x, t) -> 0)
add!(ch, dbc);
close!(ch)     # notifies constraint handler

# Assembling the linear system
K, f = assemble_global(cellvalues, K, dh);
```

2D Heat Equation on Unit Square

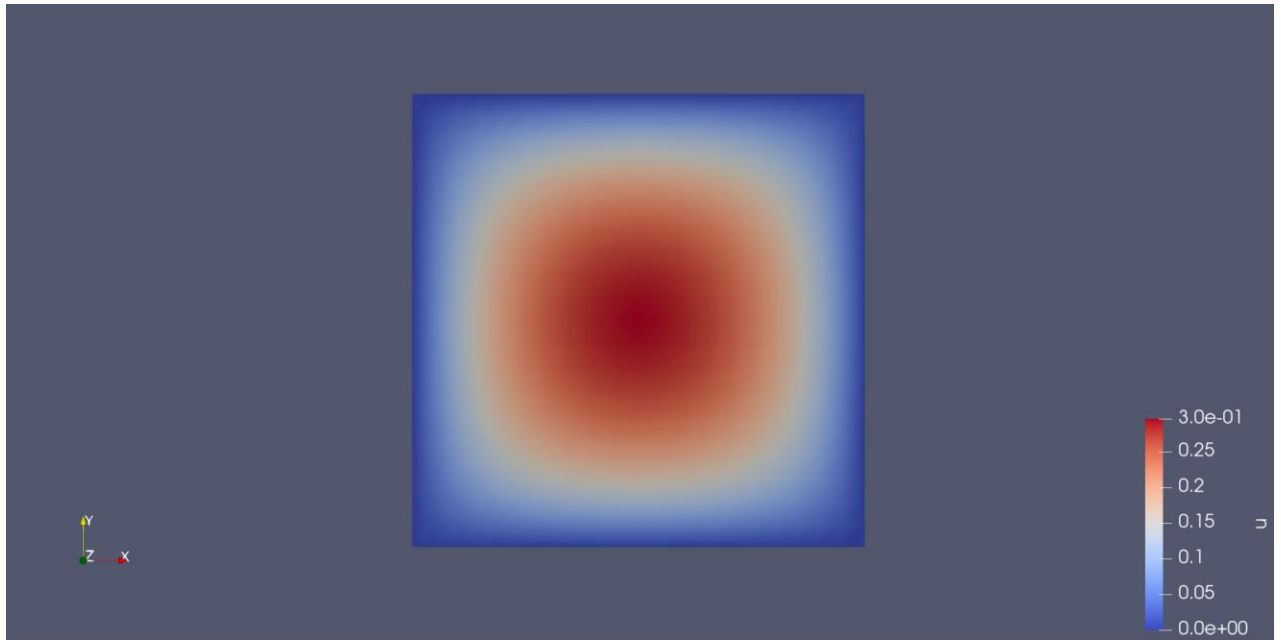
$$\begin{cases} -\nabla \cdot (k \nabla u) = f, x \in \Omega \\ u(x) = 0 \ , x \in \partial\Omega \end{cases} \quad \Longleftrightarrow \quad \int_\Omega \nabla \delta u \cdot \nabla u \, d\Omega = \int_\Omega \delta u \, d\Omega \quad \forall \delta u \in \mathbb{T},$$

```
################## Using Ferrite for Matrix Assembly ####################
grid = generate_grid(Quadrilateral, (20, 20));

# Trial and test functions
dim = 2
ip = Lagrange{dim, RefCube, 1}()
qr = QuadratureRule{dim, RefCube}(2)
cellvalues = CellScalarValues(qr, ip);

# Degrees of freedom
dh = DofHandler(grid)
add!(dh, :u, 1)
close!(dh);

# Creates SparseMatrixCSC
K = create_sparsity_pattern(dh)

# Essential BC
ch = ConstraintHandler(dh);
∂Ω = union(
    getfaceset(grid, "left"),
    getfaceset(grid, "right"),
    getfaceset(grid, "top"),
    getfaceset(grid, "bottom"),
);
dbc = Dirichlet(:u, ∂Ω, (x, t) -> 0)
add!(ch, dbc);
close!(ch)     # notifies constraint handler

# Assembling the linear system
K, f = assemble_global(cellvalues, K, dh);
```

```
################ Solve the LSE using different solvers ####################
if with_ginkgo
    # Use CG solver
    solver = GkoIterativeSolver(:cg,
                                    K,
                                exec;
                            maxiter = 20,
                        reduction = 1.0e-7)
    Ginkgo.apply!(solver, f, u, exec)
else
    u = K \ f
end
```

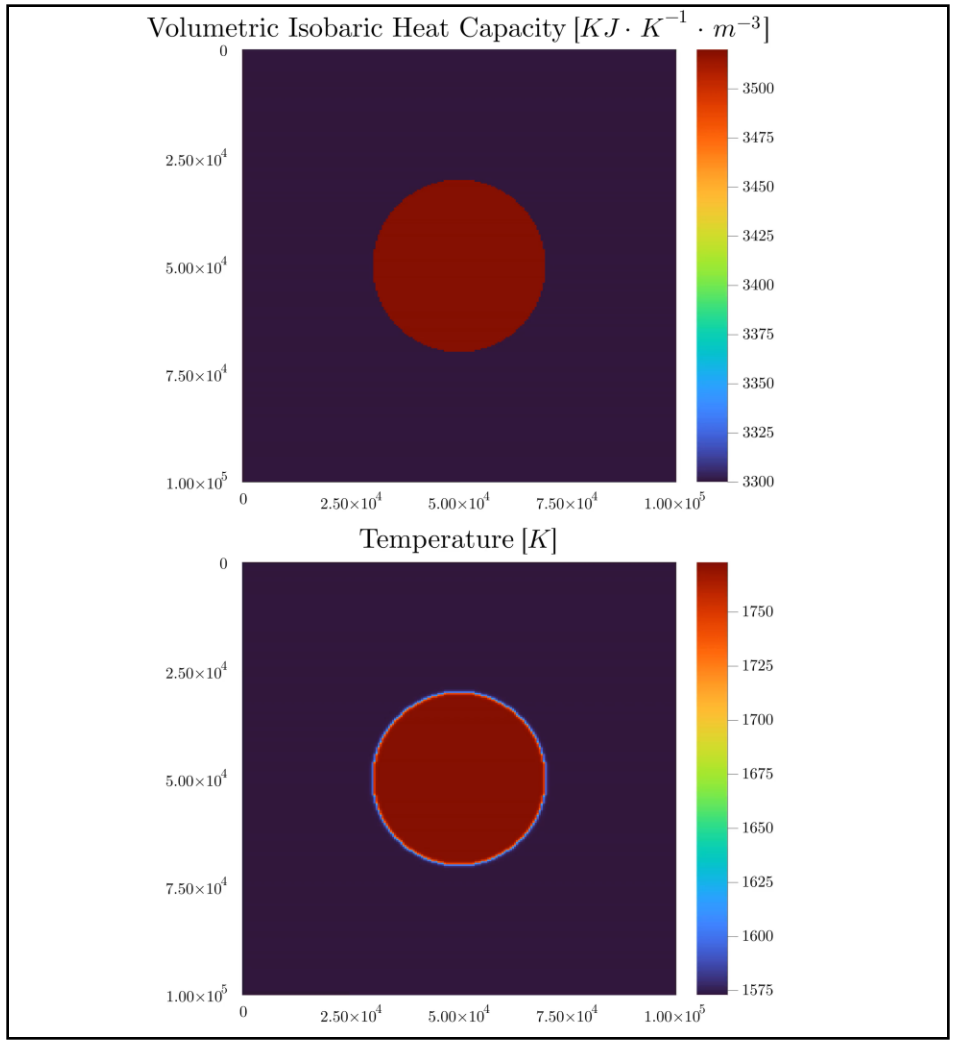
2D Heat Equation on Unit Square

$$\begin{cases} -\nabla \cdot (k \nabla u) = f, x \in \Omega \\ u(x) = 0 \ , x \in \partial\Omega \end{cases} \quad \Longleftrightarrow \quad \int_\Omega \nabla \delta u \cdot \nabla u \, d\Omega = \int_\Omega \delta u \, d\Omega \quad \forall \delta u \in \mathbb{T},$$



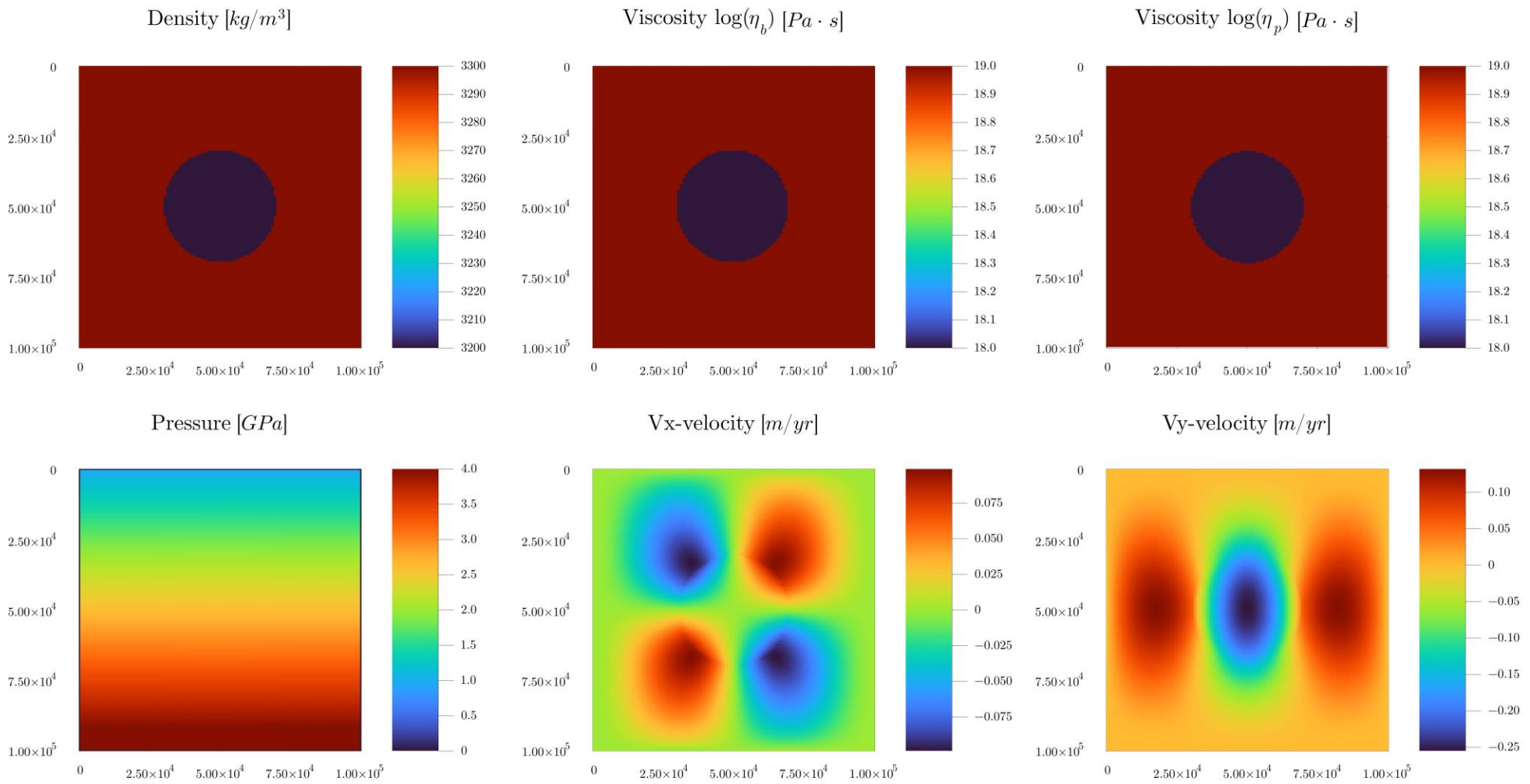
Heat Conservation

- FD method on a staggered grid
- With marker-in-cell techniques and adaptive timesteps
- Dimensionalized
- **Ginkgo.jl** sparse direct LU solver
- 10 km by 10 km square domain with 56,700 mesh cells



[2] Gerya T. Introduction to Numerical Geodynamic Modelling. 2nd ed. Cambridge University Press; 2019.

Stokes Continuity Equation



Conclusions

- Porting **Ginkgo** routines to **Ginkgo.jl** can leverage existing highly-optimized architecture-specific kernels
- C API approach is versatile but tedious, requires manual work
- For porting packages from other languages, Julia provides a good toolchain
- **Ginkgo.jl** is still under heavy development