Karlsruhe Institute of Technology

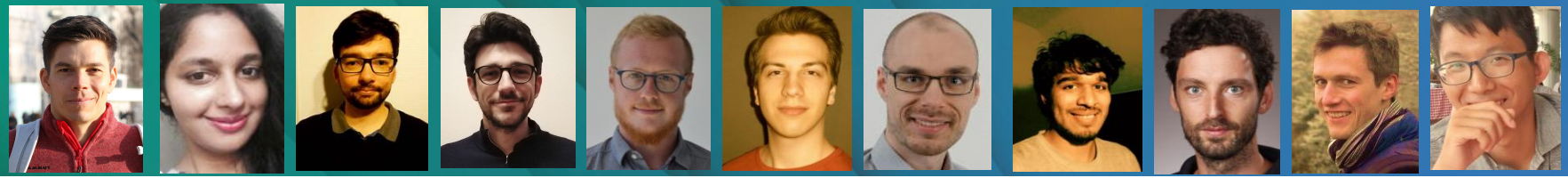
EuroHPC Joint Undertaking

MICROCARD

Recent and upcoming developments in Ginkgo.

Pratik Nayak

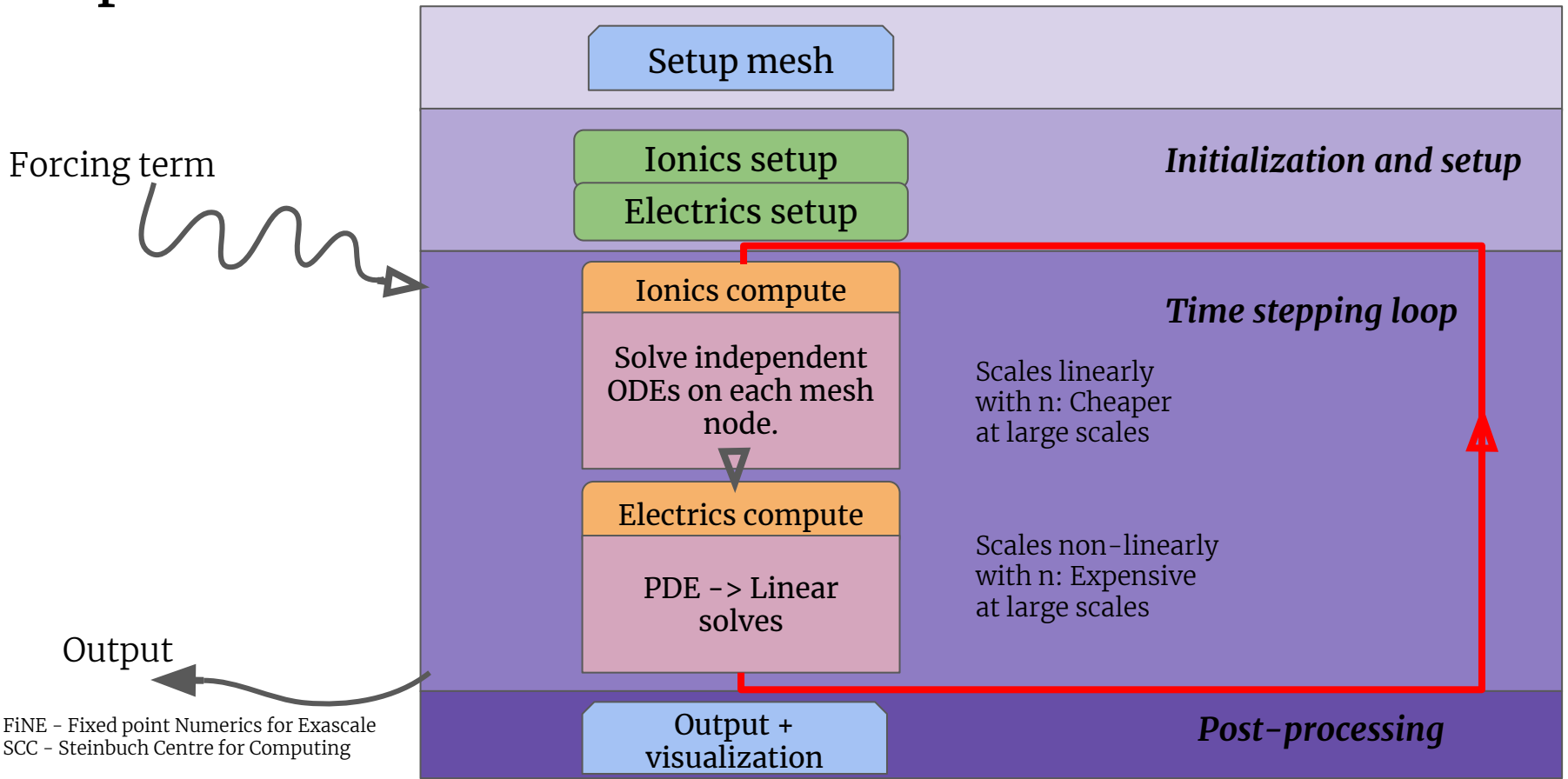
MICROCARD Workshop, Strasbourg, July 4-July 5, 2023



Outline

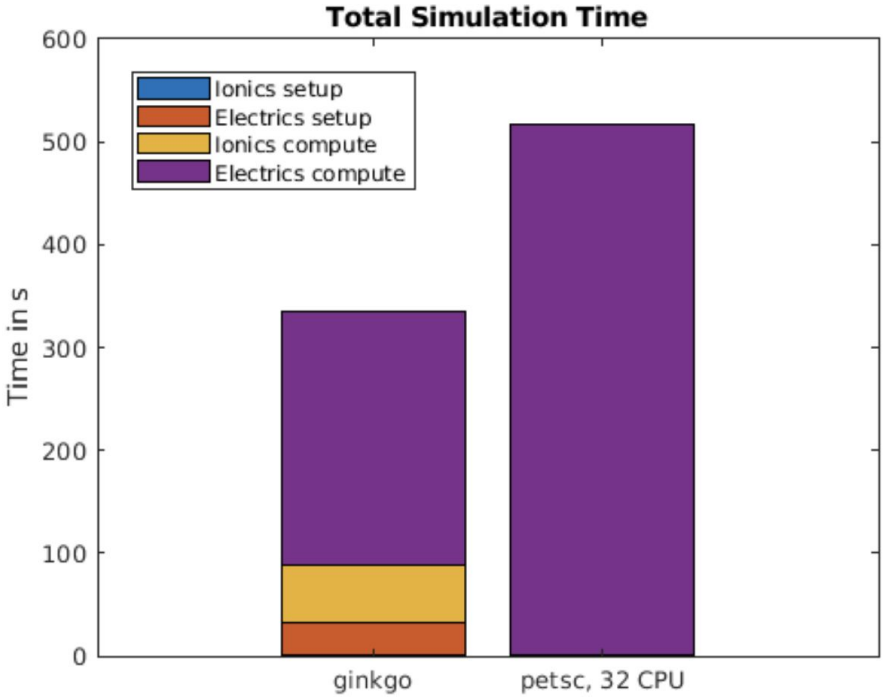
- Motivation and objectives
- Ginkgo ?
- Design philosophy and features
- A quick overview of Release 1.6.0, features and performance
- Features in development and possibly in Release 1.7.0
- Future outlook

OpenCARP simulation workflow



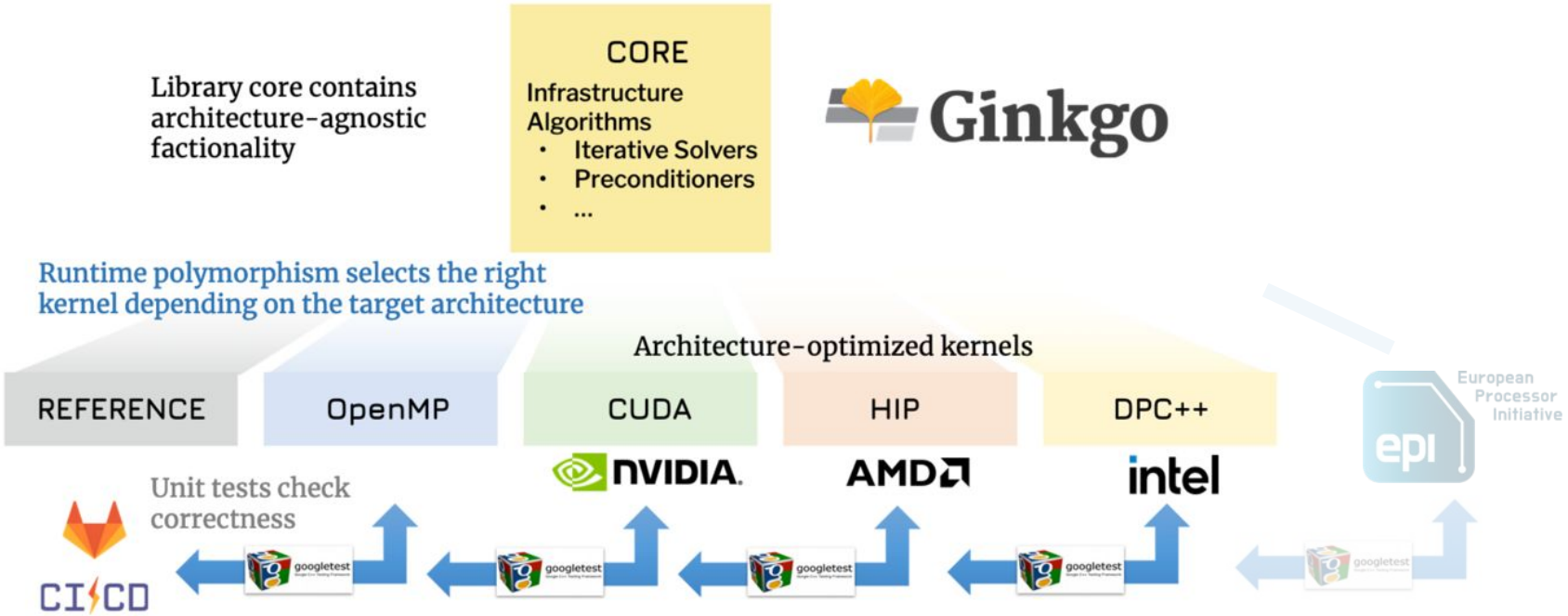
Why Ginkgo ?

- Significant part of the total time spent in linear solver.
- Crucial to optimize this bottleneck, particularly for the cell-by-cell model.
- Move computation to GPU, minimize data movement to and from GPU.

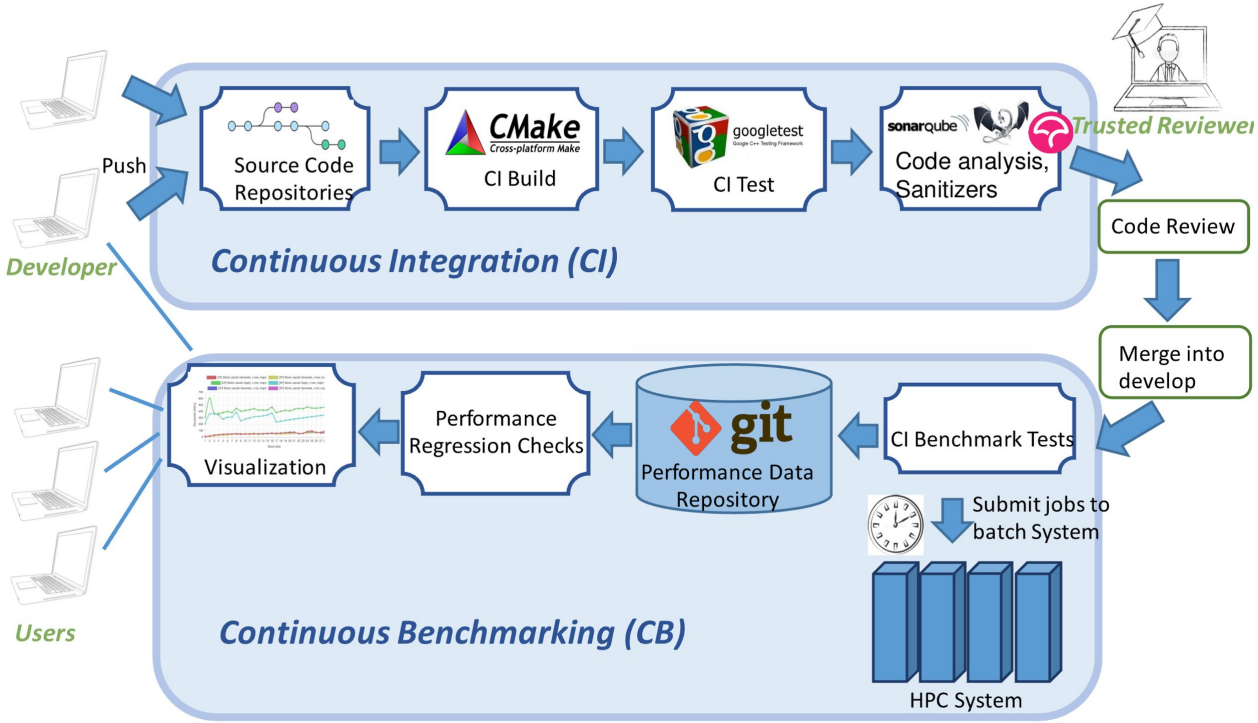


Bi-domain model: Ginkgo (1 GPU) v/s PETSc (32 CPUs)

Ginkgo: A high performance library



Ginkgo: Sustainable development



Ginkgo: Latest release - 1.6.0

- Support for sparse direct factorizations + solvers.
- Improved AMG performance.
- Profiler annotations: NVTX, ROCTX, VTune, TAU
- Stream support: Executors can now take a stream argument
- New distributed Schwarz preconditioner.
- Matrix reorderings: METIS (nested dissection), AMD (Fill-in reducing)
- Mixed precision SpMV with CSR.

More details: https://github.com/ginkgo-project/ginkgo/releases/tag/v1.6.0

Overview of features: Single executor

	Functionality	OMP	CUDA	HIP	DPC++
Basic	SpMV	✓	✓	✓	✓
	SpMM	✓	✓	✓	✓
	SpGeMM	✓	✓	✓	✓
Krylov solvers	BiCG	✓	✓	✓	✓
	BiCGSTAB	✓	✓	✓	✓
	CG	✓	✓	✓	✓
	CGS	✓	✓	✓	✓
	GMRES	✓	✓	✓	✓
	IDR	✓	✓	✓	✓
Preconditioners	(Block-)Jacobi	✓	✓	✓	✓
	ILU/IC		✓	✓	✓
	Parallel ILU/IC	✓	✓	✓	✓
	Parallel ILUT/ICT	✓	✓	✓	✓
	Sparse Approximate Inverse	✓	✓	✓	✓

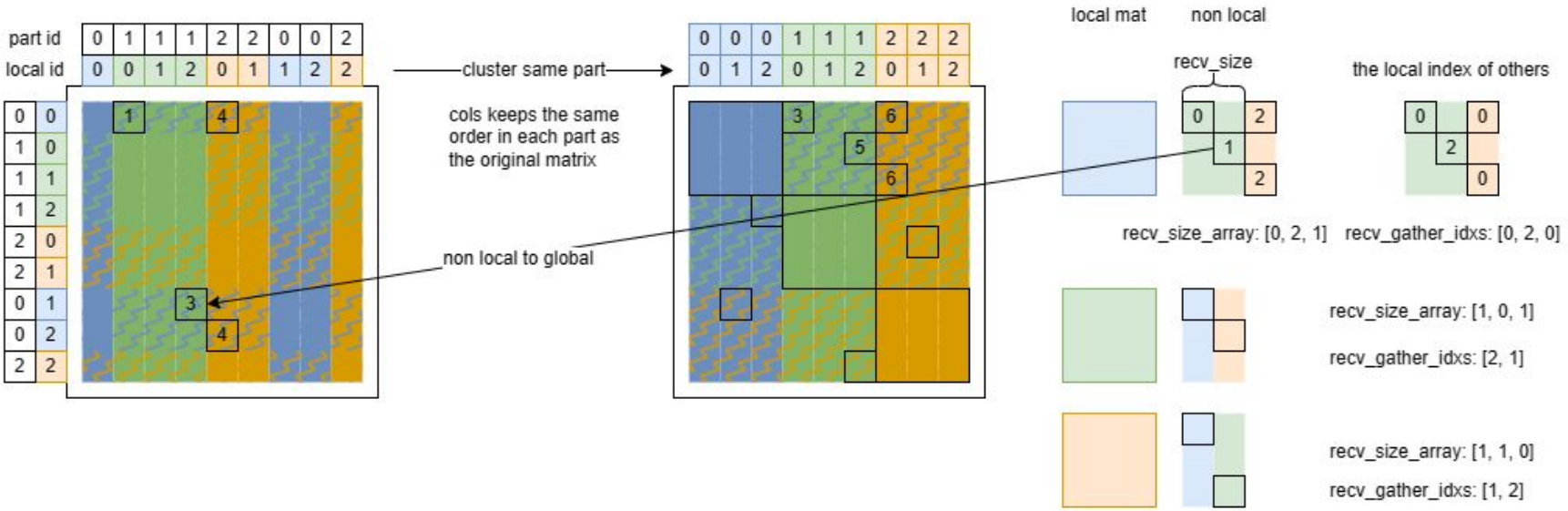
	Functionality	OMP	CUDA	HIP	DPC++
AMG	AMG preconditioner	✓	✓	✓	✓
	AMG solver	✓	✓	✓	✓
	Parallel Graph Match	✓	✓	✓	✓
Sparse direct	Symbolic Cholesky	✓	✓	✓	✓
	Numeric Cholesky	✓	✓	✓	
	Symbolic LU	✓	✓	✓	✓
	Numeric LU	✓	✓	✓	
	Sparse TRSV	✓	✓	✓	
Utilities	On-Device Matrix Assembly	✓	✓	✓	✓
	MC64/RCM reordering	✓			
	Wrapping user data	✓			
	Logging	✓			
	PAPI counters	✓			

Overview of features: Distributed

- All crucial solvers and operations supported.
- Local block preconditioners supported through the distributed Schwarz preconditioner.
- Compose matrix formats between diagonal and off-diagonal matrices.

	Functionality	OMP	CUDA	HIP	DPC++
Basic	SpMV	✓	✓	✓	✓
	SpMM	✓	✓	✓	✓
	SpGeMM	✓	✓	✓	✓
Krylov solvers	BiCG	✓	✓	✓	✓
	BiCGSTAB	✓	✓	✓	✓
	CG	✓	✓	✓	✓
	CGS	✓	✓	✓	✓
	GMRES	✓	✓	✓	✓
	IDR	✓	✓	✓	✓
Preconditioners	(Block-)Jacobi	✓	✓	✓	✓
	ILU/IC		✓	✓	✓
	Parallel ILU/IC	✓	✓	✓	✓
	Parallel ILUT/ICT	✓	✓	✓	✓
	Sparse Approximate Inverse	✓	✓	✓	✓

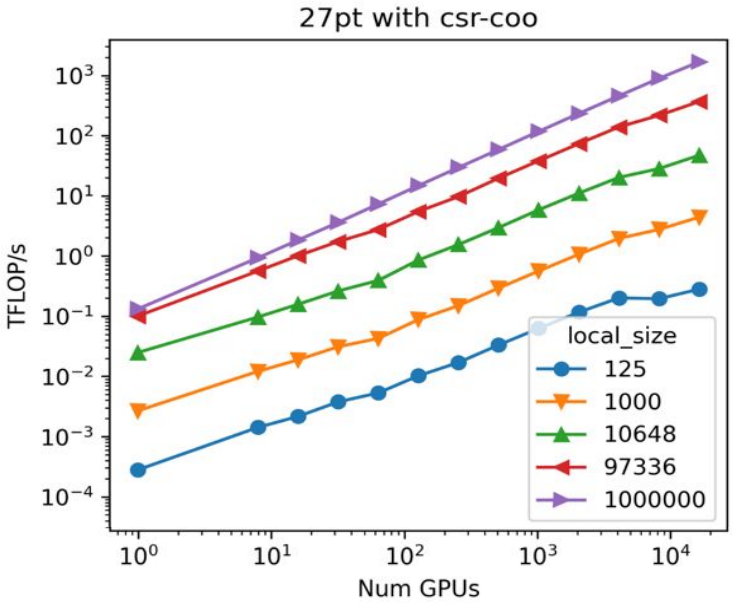
Ginkgo: Distributed data distribution

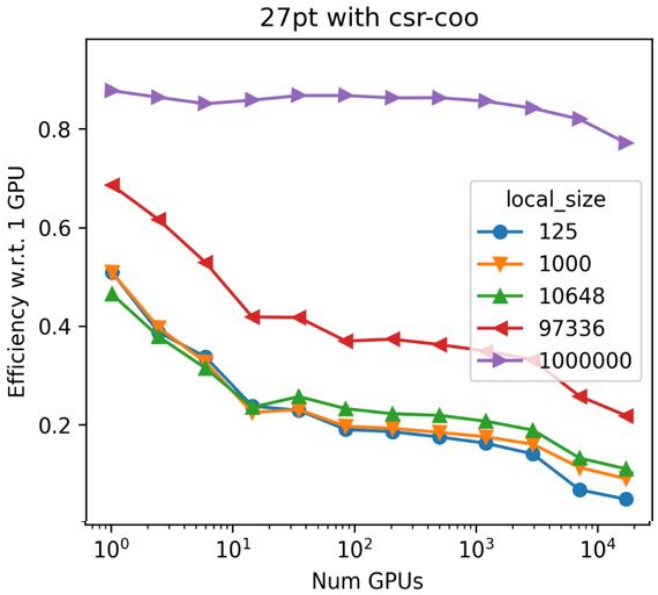


Sketch created by
Yu-Hsiang Tsai

Weak scaling: SpMV on Frontier

Cray MPICH + AMD MI250X on 16k GCDs

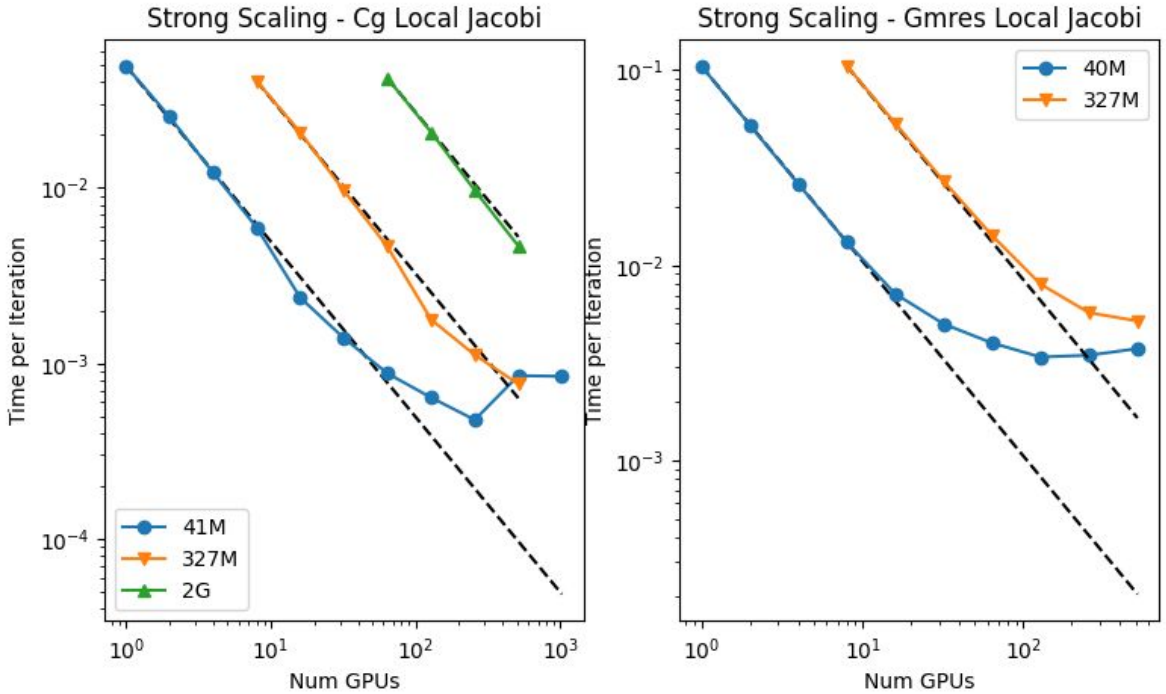




Weak scaling: problem size increases with parallel resources

Strong scaling: Solvers on Frontier

Cray MPICH + AMD MI250X

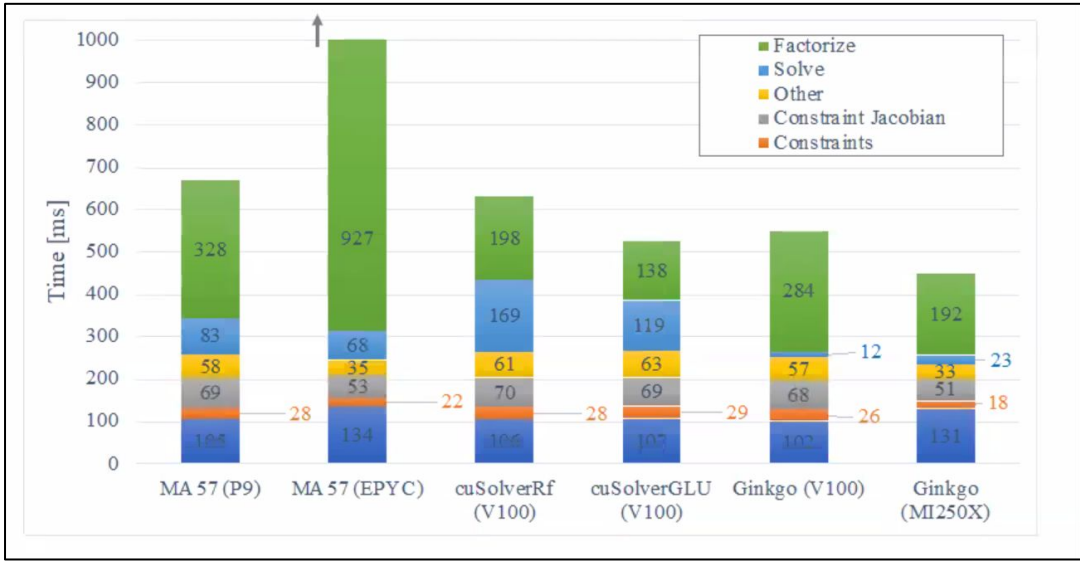


Strong scaling: constant problem size

GPU resident sparse direct solvers

- Power Grid Simulations
- All GPU solvers outperform CPU solvers
- Ginkgo first GPU-resident solver
- Enables factorization on AMD and also on Intel GPUs (No vendor alternatives available)

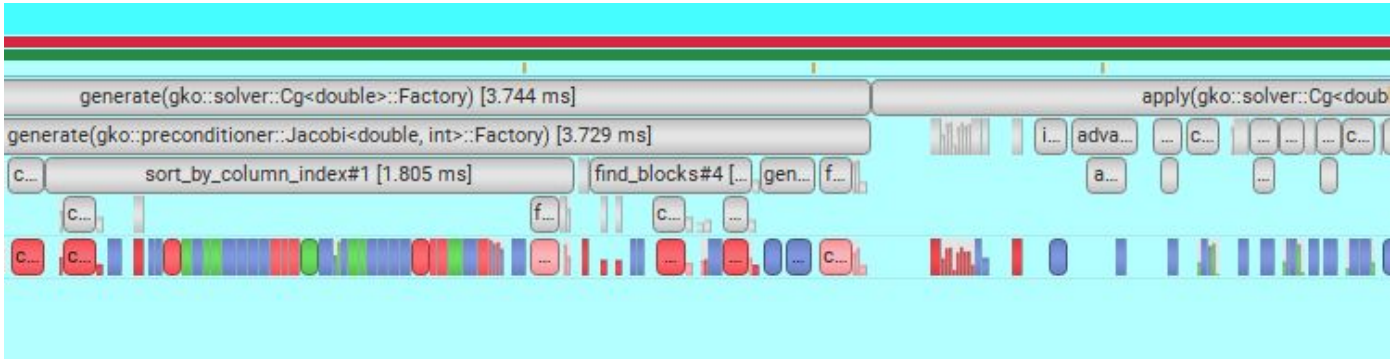
EXASGD



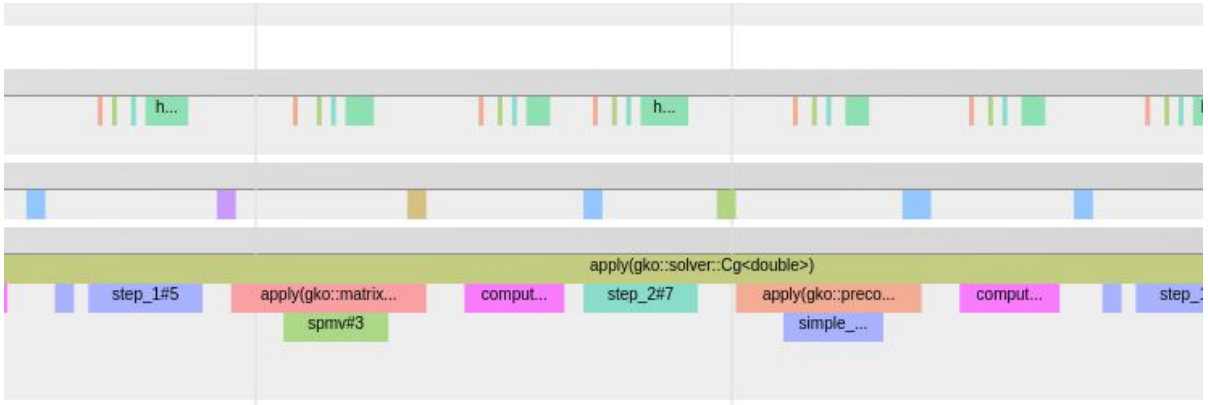
© Slaven Peles

Some utilities: Profiler annotations

NVTX



ROCTX



Some utilities: Profiler annotations

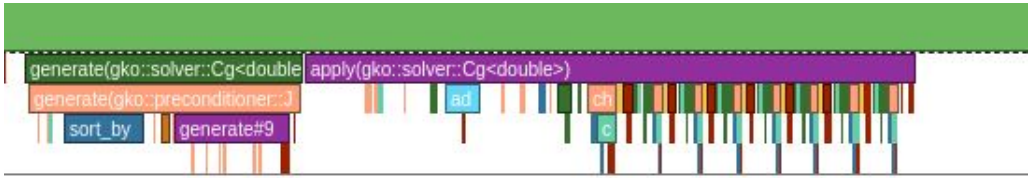
TAU

Metric: TIME
Value: Exclusive
Units: seconds

0.188	apply(gko::solver::Cg<double>)
0.098	spmv#3
0.079	apply(gko::matrix::Csr<double, int>)
0.075	apply(gko::preconditioner::Jacobi<double, int>)
0.055	compute_conj_dot#4
0.033	step_1#5
0.031	simple_apply#8
0.026	step_2#7
0.021	check(gko::stop::Iteration)
0.015	.TAU application
3.5E-4	sort_by_column_index#1
3.0E-4	generate(gko::preconditioner::Jacobi<double, int>::Factory)
3.0E-4	generate#9
1.7E-4	advanced_apply(gko::matrix::Csr<double, int>)
8.7E-5	fill_array#3
7.1E-5	aos_to_soa#2
3.9E-5	generate(gko::solver::Cg<double>::Factory)
3.9E-5	find_blocks#4
3.1E-5	advanced_spmv#5
2.5E-5	fill_in_matrix_data#2
2.5E-5	initialize#8
2.5E-5	set_all_statuses#3
2.4E-5	compute_norm2#3
1.9E-5	fill#2
1.5E-5	convert_idxs_to_ptrs#4

Tuning and Analysis Utilities

Intel VTune

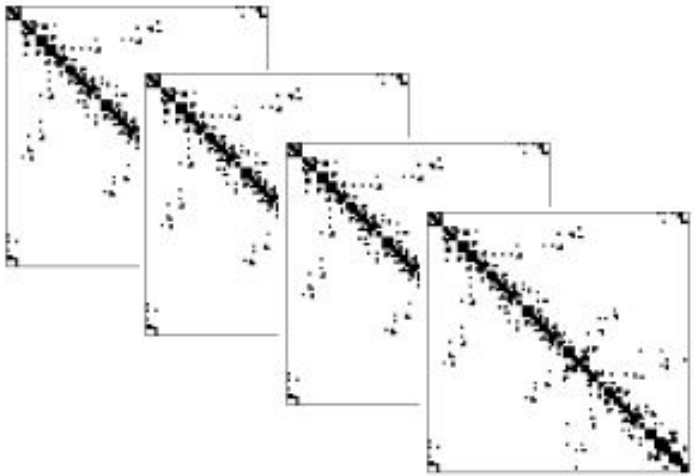


What's in development and maybe in 1.7.0 ?

- Batched iterative methods.
- More distributed preconditioners:
  - Distributed AMG (Coming soon)
  - Distributed BDDC (See Fritz's talk)
  - Two-level Schwarz methods.
- Easy solver configuration with JSON (already in use in openCARP)
- Local-only distributed matrix format: Abstract matrix format to describe and operate on overlapping partitions.
- Custom allocator support: Get around large allocation overheads with memory pool allocators (in applications).

Batched methods ?

- Batching: Independent computations that can be scheduled in parallel.
- Are highly suitable for GPUs and processors with many parallel computing units.
- Can maximize utilization of the GPU, due to excellent scalability.



Batched methods in Combustion

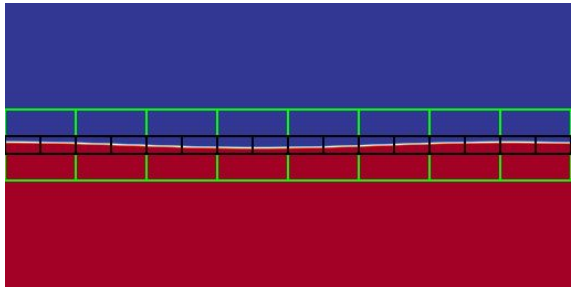
- Low mach number, non-compressible flow: leads to very stiff ODEs
- Linear multi-step time stepping with BDF

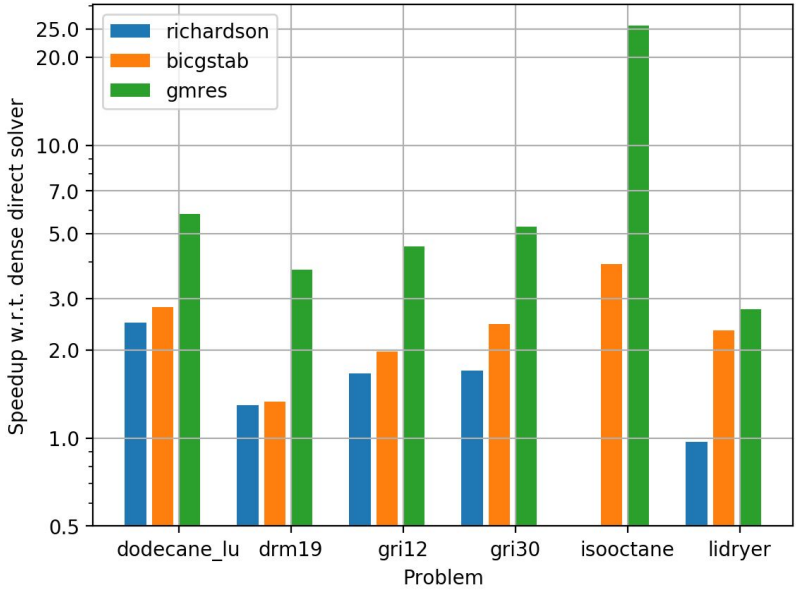
$$(I - h_n \beta_{n,0} \frac{df}{d\phi})[\phi^{n(m+1)} - \phi^{n(m)}] = -G(\phi^{n(m)})$$

- Sparsity pattern, $\mathbf{A} \equiv (I - h_n \beta_{n,0} \frac{df}{d\phi})$

  same for each cell.

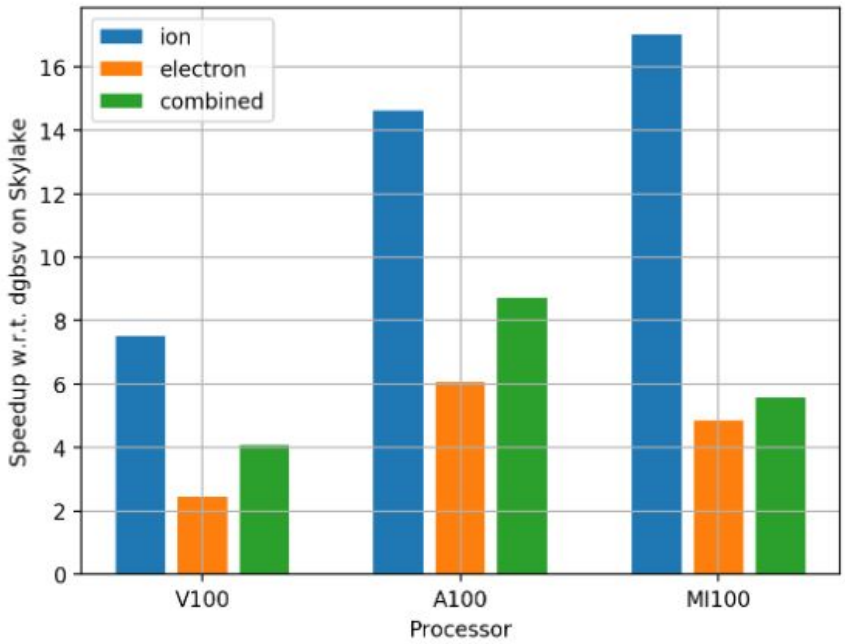
- Hence independent systems to be solved at each cell of the order of the number of cells 200k ~ 2e6 on each GPU





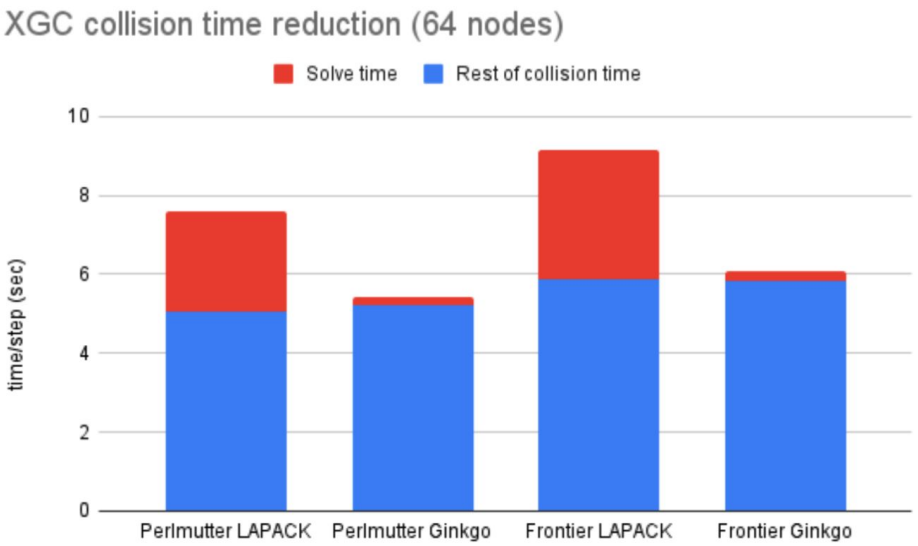
Batched methods in Fusion Plasma

- Similar idea as before, but for Particle in cell code (PIC) XGC simulating gyrokinetic plasma in a tokamak.
- Reduction in linear solve time by about 90%.
- Scales extremely well.



Batched methods in Fusion Plasma

- Similar idea as before, but for Particle in cell code (PIC) XGC simulating gyrokinetic plasma in a tokamak.
- Reduction in linear solve time by about 90%.
- Scales extremely well, due to the embarrassing parallelism.



Easy configuration for Ginkgo

- User provides a simple JSON config file at runtime.
- Solver and interface within OpenCARP configured automatically from these settings.

```
[
    {
        "name": "solver",
        "base": "CgFactory",
        "criteria": [
            {
                "name": "AbsTol",
                "base": "ResidualNormFactory",
                "baseline": "absolute",
                "reduction_factor": 1e-8
            },
            {
                "name": "Iter",
                "base": "IterationFactory",
                "max_iters": 100
            }
        ],
        "exec": "executor"
    },
    {
        "name": "precond",
        "base": "JacobiFactory",
        "max_block_size": 32,
        "exec": "executor"
    }
]
```

```
{
    "name": "precond",
    "base": "MultigridFactory",
    "criteria": [
        {
            "base": "IterationFactory",
            "max_iters": 1
        }
    ],
    "max_levels": 9,
    "min_coarse_rows": 64,
    "post_uses_pre": true,
    "zero_guess": true,
    "pre_smoother": [
        {
            "base": "IrFactory",
            "criteria": [
                {
                    "base": "IterationFactory",
                    "max_iters": 1
                }
            ],
            "solver": {
                "base": "JacobiFactory",
                "max_block_size": 1
            },
            "relaxation_factor": 0.9
        }
    ],
    "mg_level": [
        {
            "base": "AmgxPgmFactory",
            "deterministic": false
        }
    ],
    "coarsest_solver": {
        "base": "IrFactory",
```

Easy configuration for Ginkgo

```
 83  template <typename T, typename S>
 84  void ginkgo_solver<T, S>::setup_solver(abstract_matrix<T, S>& mat, double tol, int max_it, short norm,
 85                      const char* name, bool has_nullspace, void* void_logger,
 86                      const char* solver_opts_file, const char* default_opts)
 87  {
 88    using RM = gko::extension::resource_manager::ResourceManager;
 89    auto &gko_mat = dynamic_cast<ginkgo_matrix<T,S>&>(mat);
 90    auto exec = gko_mat.data->get_executor();
 91    auto non_const_exec = std::const_pointer_cast<gko::Executor>(exec);
 92    auto str_name = std::string{name} + " (Ginkgo)";
 93    this->name = str_name.c_str();
 94    this->options_file = solver_opts_file;
 95    auto logger = reinterpret_cast<opencarp::FILE_SPEC>(void_logger);
103    RM resource_manager;
104    std::ifstream options_json(this->options_file);
105    rapidjson::IStreamWrapper options_in(options_json);
106    rapidjson::Document f_options;
107    f_options.ParseStream(options_in);
108    resource_manager.insert_data<gko::Executor>("executor", non_const_exec);
109    resource_manager.read(f_options);
115
116    auto solver_factory = resource_manager.search_data<gko::LinOpFactory>("solver");
117    this->solver = solver_factory->generate(gko_mat.data);
118    dynamic_cast<gko::Preconditionable *>(this->solver.get())->set_preconditioner(this->precond);
119    this->set_stopping_criterion(normtype, tol, max_it, verbose, logger);
120  }
```

Conclusion

- Various new features available in 1.6.0
  - Distributed block preconditioners.
  - Profiling annotations.
  - Improved on on GPU AMG performance.
  - Scaling on Frontier.
- New upcoming features (1.7.0 and later)
  - Batched iterative solvers.
  - Distributed preconditioners (BDDC, AMG)
  - Easy configuration with JSON files
- Happy to discuss any additional feature requests.

Acknowledgements

EuroHPC
Joint Undertaking

This project has received funding from the European High-Performance Computing Joint Undertaking EuroHPC (JU) under grant agreement No 955495.
The JU receives support from the European Union's Horizon 2020 research and innovation programme and France, Italy, Germany, Austria, Norway, and Switzerland.
The project was co-funded by the French National Research Agency ANR, the German Federal Ministry of Education and Research, the Italian ministry of economic development, the Swiss State Secretariat for Education, Research and Innovation, the Austrian Research Promotion Agency FFG, and the Research Council of Norway.

Ginkgo

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