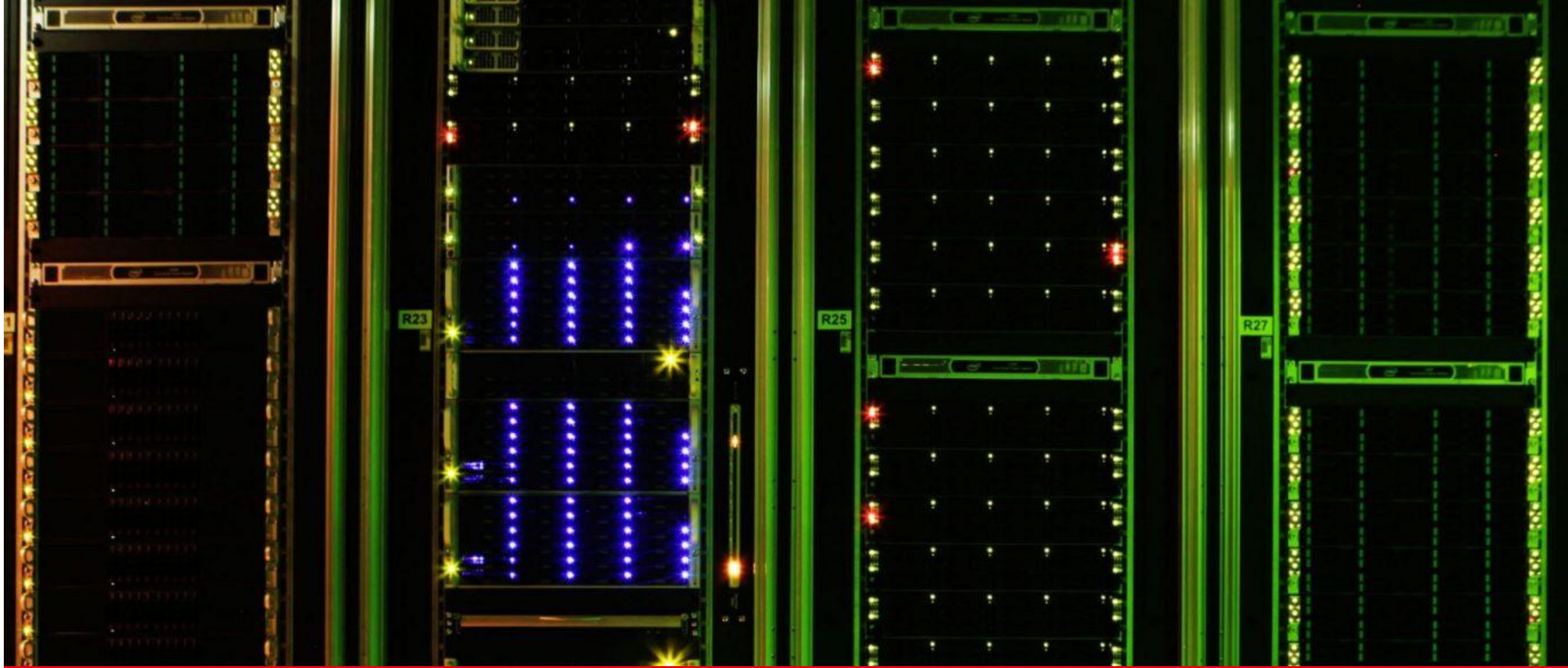


Progress on the ORB5 and GENE3D codes

E. Lanti and P. Panchal

2nd Annual Meeting of EUROfusion HPC ACHs
Wed. 27th of November 2024, Barcelona

ORB5

Let’s jump back ten years ago

“An investigation into the library-based approach, using Kokkos, from a “mini-app” such as GK-Engine [...]”

A little bit of ORB5 history

- In 2015 began a big work of merge, refactoring, optimization, and GPU porting
- Requirements were portability, single source code, and “ease of use”
- OpenMP and OpenACC chosen for multithreading and GPU support
- We chose to use mini apps
  - Pic-Engine: very simple app retaining the PIC main routines (push, deposition, solve, and get field)
  - GK-Engine: mini version of ORB5
- We used the experience from the mini app into ORB5
  - ORB5 ran on 90% of Summit with good scaling up to ~6000 GPUs [Ohana, *et al.* (2019)]

ORB5

Back to today

Lessons learned

- OpenACC is not portable (mainly works with NVIDIA)
  - OpenMP offload being implemented (T. Ewart (Intel), V-M. Yli-Suutala (CSC))
  - But compiler support is OK^{ish}
- While both Open{MP,ACC} allow a single source code, in reality it is not always the case
  - Annotations are simply ignored if not using multi-threading or GPUs

Explore a library-based approach using Kokkos

- Kokkos is designed to be performance portable
- They used an abstract model of a compute node that must be accounted for since the beginning
  - Abstract engine
  - Abstract memory placement
  - Abstract memory layout

ORB5

Back to today

Fortran is not an option anymore

- Fortran just works™
  - It benefits from decades of experience and is amazingly performant on CPU
- Fortran and Kokkos don't mix well together [N. Moschüring (2021)]
- But it lacks an ecosystem
  - Support for new hardware and new language features are very slow
  - Very few (good) tools around the language (linter, formatter, etc.)

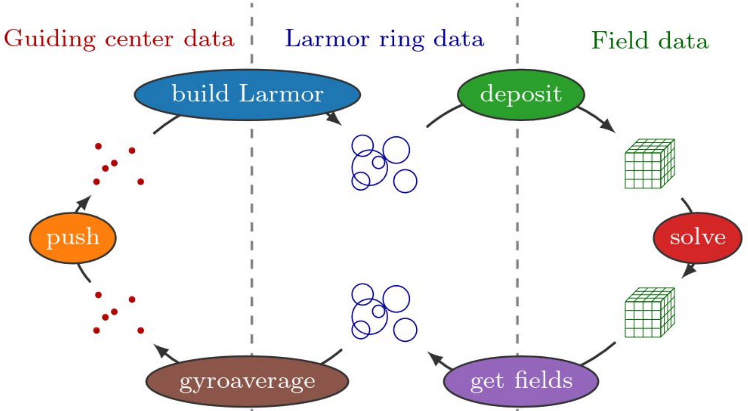
We chose C++ as a replacement

- Zero-cost abstraction
- Big momentum to make the language evolve towards scientific computing
- Community guidelines to write safer, cleaner, and faster code
- A lot of tooling
- Quick implementation of the standard (you can already try C++26!)

ORB5

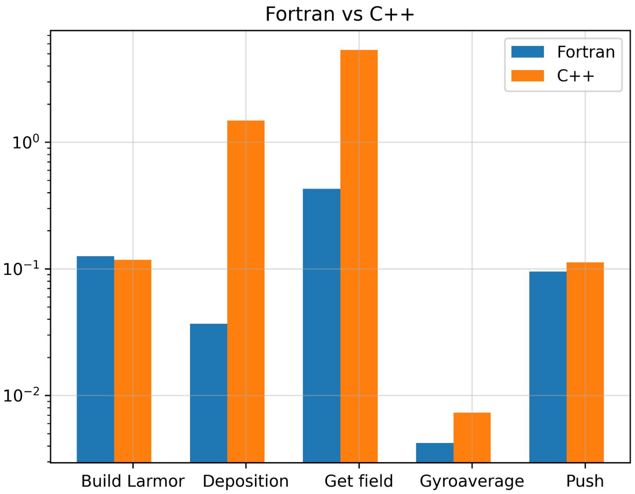
GkEngine++

- C++ version of the Gk-Engine (limited to ES physics with adiabatic e$^-$)
  - Build system using Modern CMake
  - Testing using Catch2 framework
  - Reproduces Gk-Engine results
- Experiment with different design approaches
- Two iterations
  - One with custom “NDField” emulating Fortran arrays
  - One with Kokkos (currently only for CPU)



ORB5

GkEngine++ vs Gk-Engine

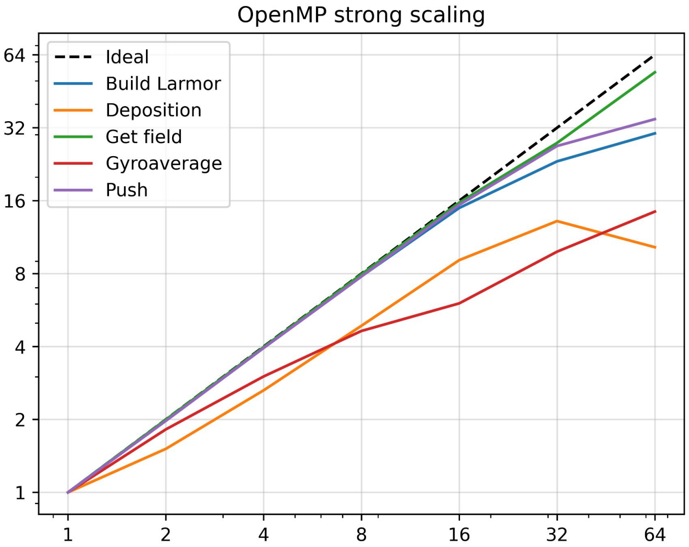


- Fortran is very performant out of the box!
- Once tackled an inlining problem with the C++ version performance is comparable
- Not a totally fair comparison because not all the languages capabilities were accounted for

ACH 2024 - ORB5 and GENE3D - Barcelona

ORB5

An OpenMP scaling using Kokkos



- Recompiled with Kokkos OpenMP backend
- “Naive” implementation, no optimizations made
- Ran on the Jed production cluster @ EPFL
  - 2 x 36 cores compute nodes

ACH 2024 - ORB5 and GENE3D - Barcelona

ORB5

Conclusions

- Implemented a new C++ test-bed based on the Gk-Engine
- Can be used to test various technologies such as Kokkos

- First experience in porting a production Fortran code to another language
- Performance are comparable, but need to be careful with C++ features we use

- Finish the Kokkos implementation to work on GPUs
- Polish the implementation
  - Better C++ (no copy/paste of Fortran code)
  - Optimize the Kokkos porting
- Try C++ std::algorithm which could make the code more declarative

EPFL

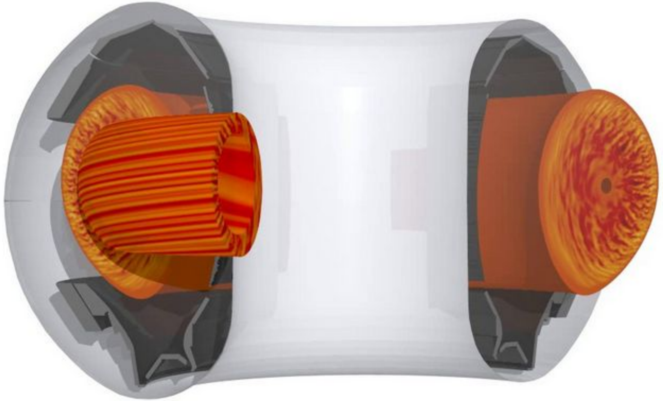
Improving Linear Solve Times

Linear System

- Independent induction linear systems (67584 X 67584, GENE3D)

$$\mathbf{A}\mathbf{x}_{t+1} = \mathbf{b}_t$$

- Invertible, symmetric system matrix. Multiple solves.
- Goal: fastest multi solve using one GPU (one node)

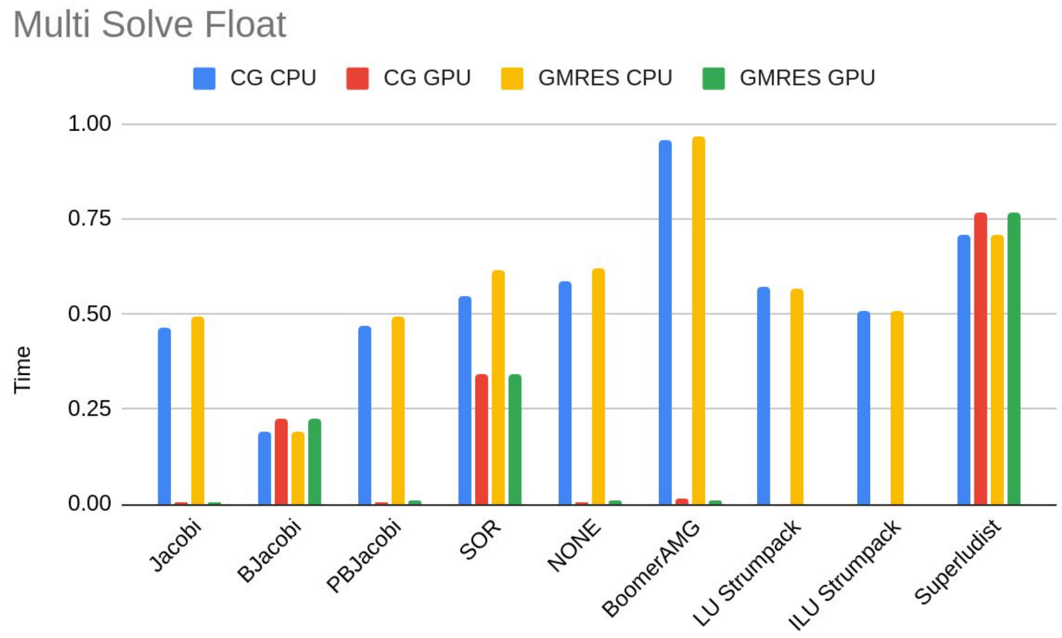


Solvers & Preconditioners in Petsc

- Direct solver
    - LU: superludist, matsolverstrumpack
- Indirect solver
    - CG
    - GMRES
- Preconditioners
    - Jacobi
    - BJacobi
    - PBJacobi
    - SOR
    - None
    - BoomerAMG: Hypre
    - ILU: Matsolverstrumpack

Results

10 solve reps, 0 initial guess, RTOL 1e-5, 8 OMP threads

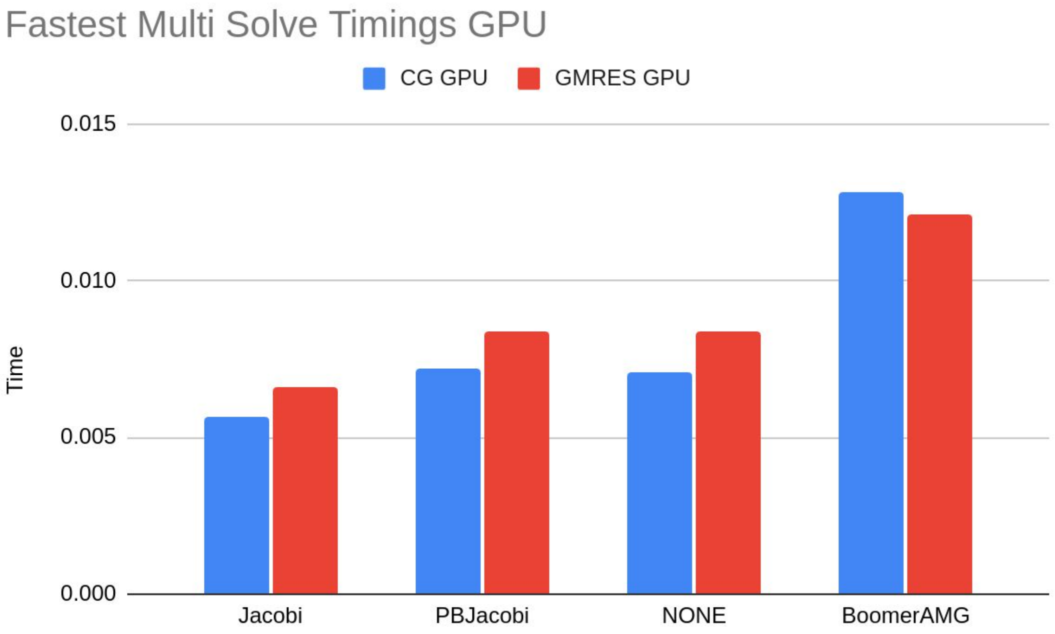


No GPU + float support in strumpack!

- Superludist, CPU: 0.708914
- Jacobi, GPU
  - CG: 0.005638 (125X)
  - GMRES: 0.006632 (107X)
- Boomeramg, GPU
  - CG: 0.012828 (55X)
  - GMRES: 0.012116 (58X)
- None, GPU
  - CG: 0.007063 (100X)
  - GMRES: 0.008353 (85X)

Results

10 solve reps, 0 initial guess, RTOL 1e-5, 8 OMP threads

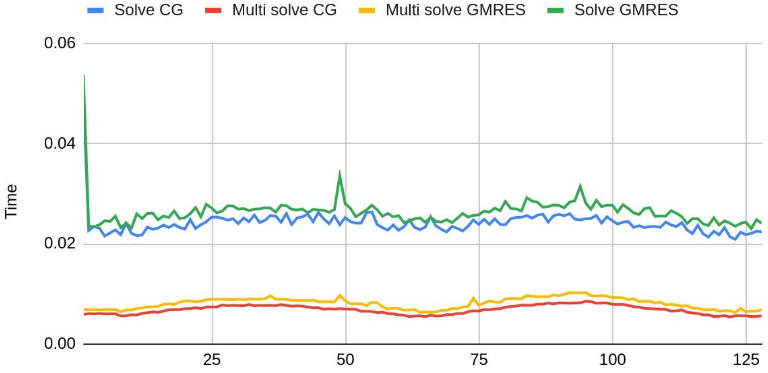


- Superludist, CPU: 0.708914
- Jacobi, GPU
  - CG: 0.005638 (125X)
  - GMRES: 0.006632 (107X)
- Boomeramg, GPU
  - CG: 0.012828 (55X)
  - GMRES: 0.012116 (58X)
- None, GPU
  - CG: 0.007063 (100X)
  - GMRES: 0.008353 (85X)

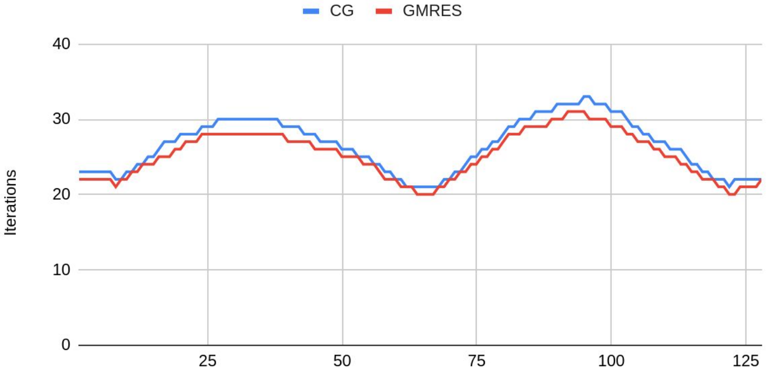
ACH 2024 - ORB5 and GENE3D - Barcelona

Detailed Picture

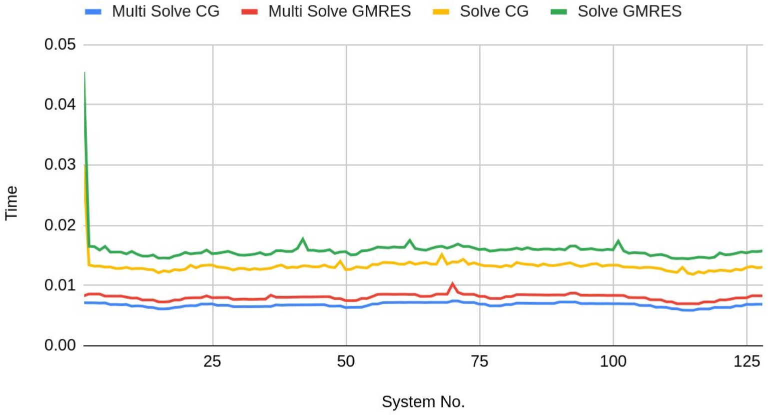
Jacobi GPU Timings



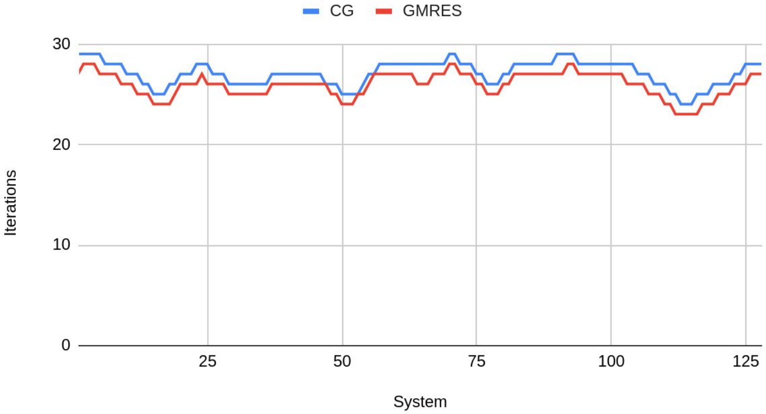
Jacobi GPU Iterations



Timing GPU None

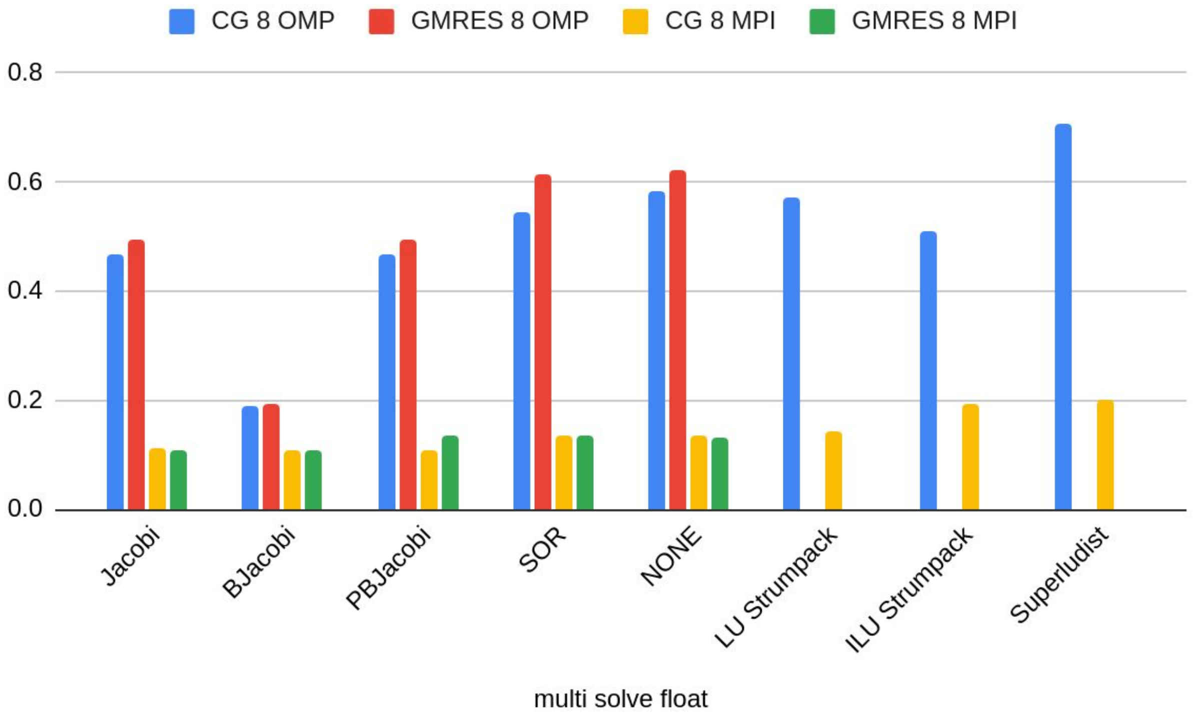


Iterations GPU None



ACH 2024 - ORB5 and GENE3D - Barcelona

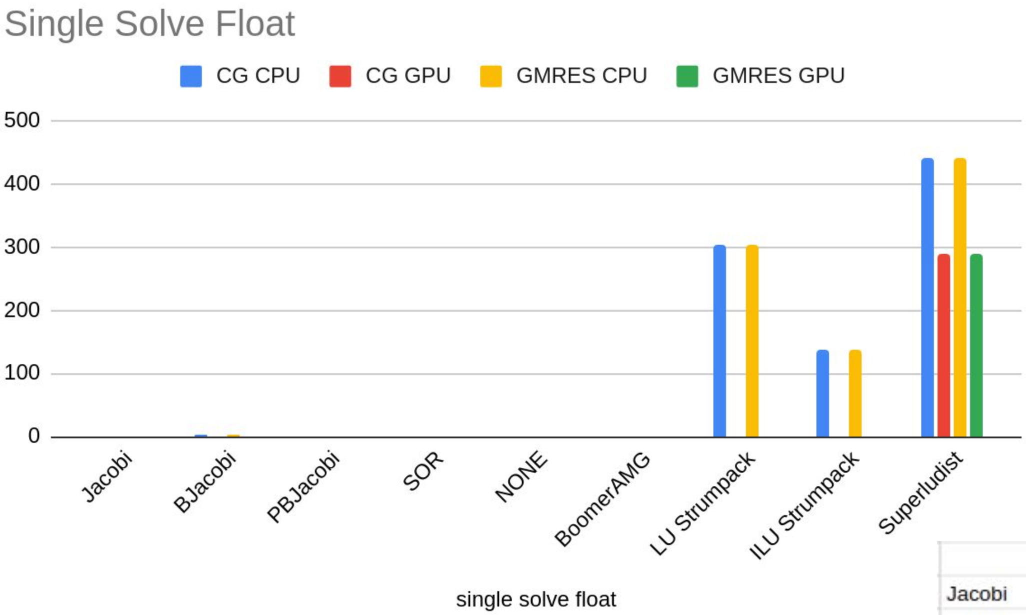
MPI vs OpenMP

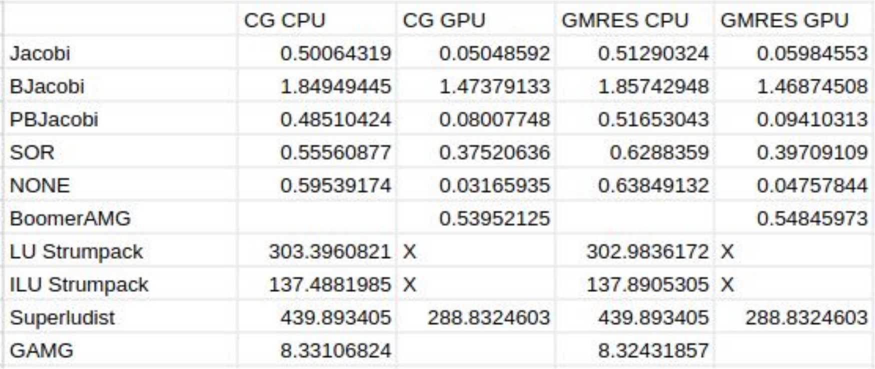


MPI parallelization gives better scaling than OpenMP. Still slower than GPU.

ACH 2024 - ORB5 and GENE3D - Barcelona

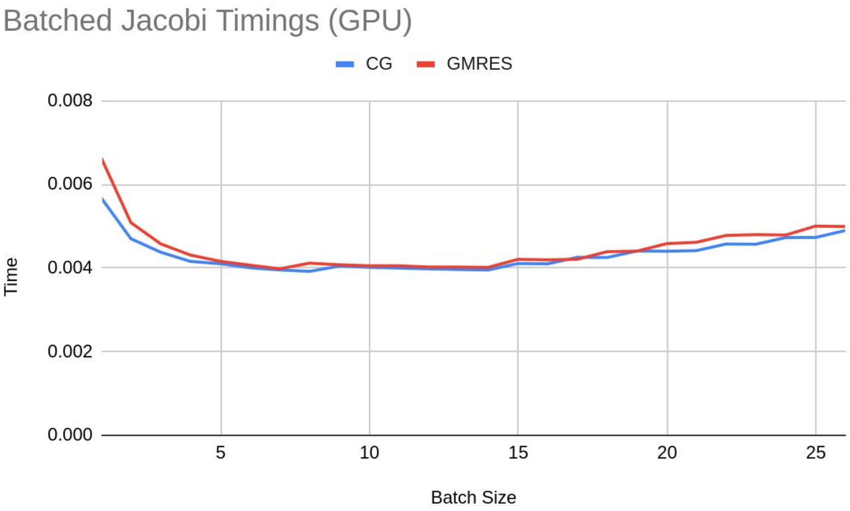
Single Solve Times

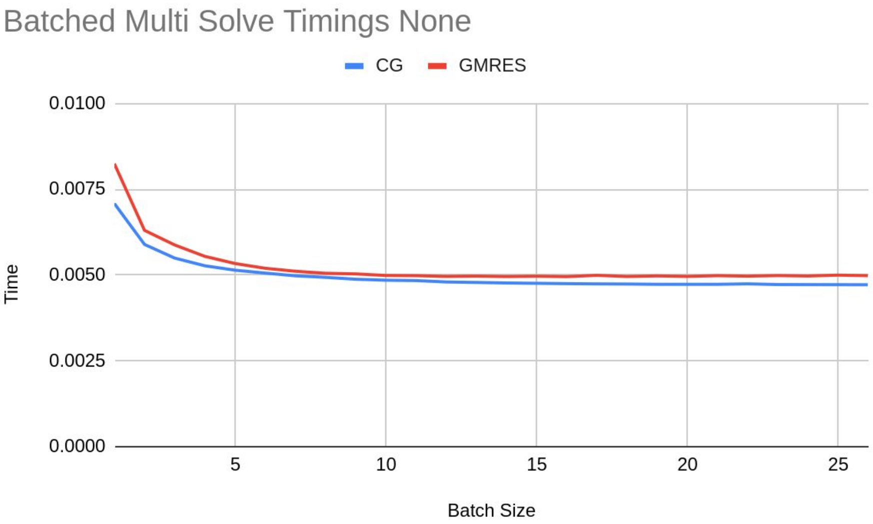




	CG CPU	CG GPU	GMRES CPU	GMRES GPU
Jacobi	0.50064319	0.05048592	0.51290324	0.05984553
BJacobi	1.84949445	1.47379133	1.85742948	1.46874508
PBJacobi	0.48510424	0.08007748	0.51653043	0.09410313
SOR	0.55560877	0.37520636	0.6288359	0.39709109
NONE	0.59539174	0.03165935	0.63849132	0.04757844
BoomerAMG		0.53952125		0.54845973
LU Strumpack	303.3960821	X	302.9836172	X
ILU Strumpack	137.4881985	X	137.8905305	X
Superludist	439.893405	288.8324603	439.893405	288.8324603
GAMG	8.33106824		8.32431857	

Combining Linear Systems





~30% improvement. Best batch size depends on hardware and problem size.

Summary

- Well conditioned systems. Performance gain with iterative solver.
- Lowest timings with Jacobi preconditioner on GPU
- Batching systems can improve performance per system
- Similar results for double precision (RTOL=1e-12)

Thank you!

ACH 2024 - ORB5 and GENE3D - Barcelona