SPLISS

A Sparse Linear System Solver for Transparent Integration of Emerging HPC Technologies into CFD Solvers

January 24th, 2023, CaDS Seminar, IAS, Jülich Supercomputing Centre

Arne Rempke, Olaf Krzikalla, Jasmin Mohnke, Johannes Wendler, Michael Wagner, Thomas Gerhold
Institute of Software Methods for Product Virtualization, High Performance Computing, German Aerospace Center (DLR)

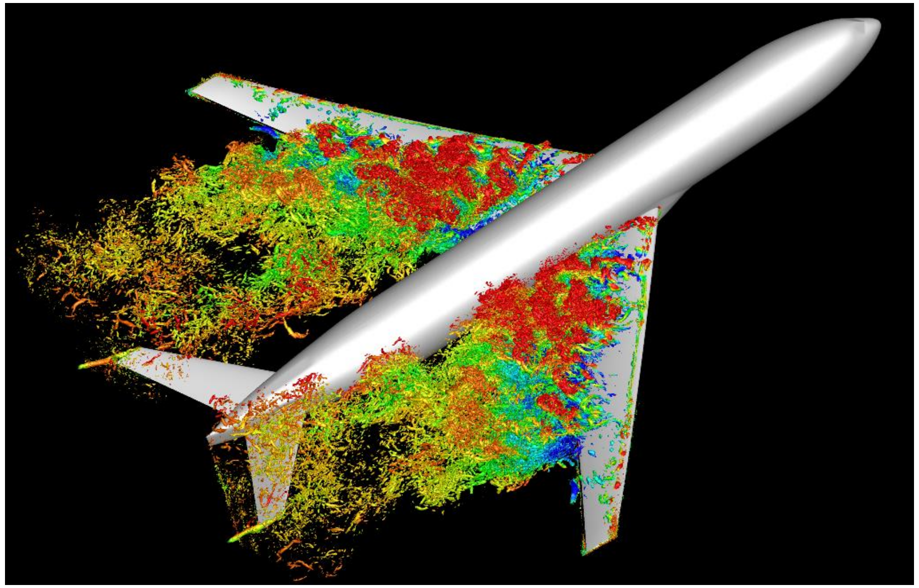
DLR

WHAT‘S THE CONTEXT

Arne Rempke, German Aerospace Center (DLR), Institute of Software Methods for Product Virtualization, January 24th, 2023

What we (DLR aerospace) do

- Wind tunnel experiments
- Flight tests
- …
- Computational Fluid Dynamics
  - Numerically solving nonlinear partial differential equations
  - For implicit schemes the most expensive part is solving linear equation systems
  - Industrial relevant cases require efficient use of HPC (turbulence is difficult)



Our challenges/chances:

- Try to make use of current (and be ready for future) hardware technology, but codes are often complex, large, calibrated to physical measurements and quality assured, so it is not so easy to adopt fast
- Due to recent changes in hardware technology (Many-core, SIMD, GPU, …), we have worked on new implementations

Software Approach to tackle these challenges

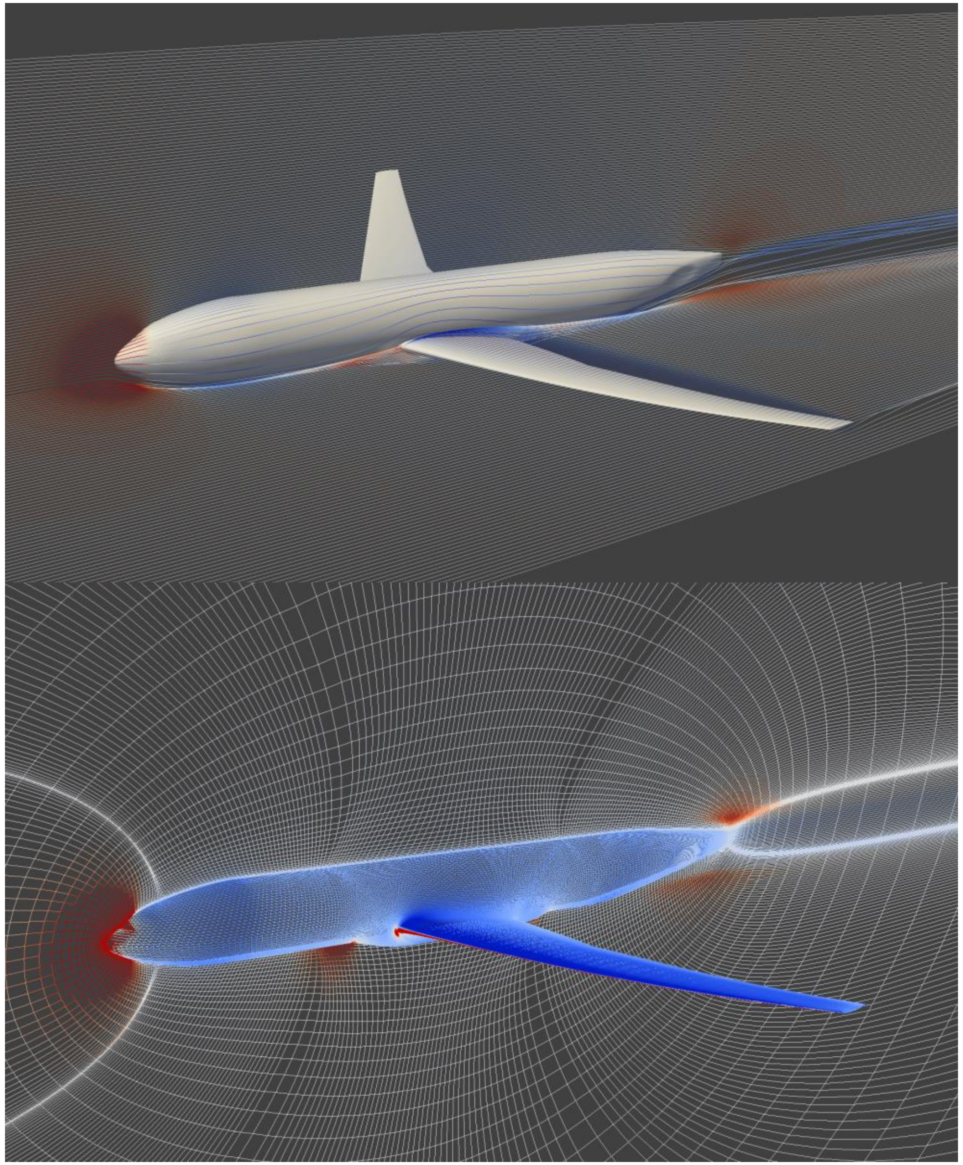
- Different CFD solvers for specific flow characteristics
  - TRACE for turbomachinery
  - CODA for aerodynamics
  - …
  - Contain physical modeling, handling of boundary conditions, nonlinear relations, wind-tunnel calibration, transsonic/hypersonic/… flow regime, …

- Common library for (approximatively) solving a linear equation system with characteristics from aeronautical CFD
  - More focus on low-level performance and hardware technologies
  - May adapt to specific technologies more easily due to its comparably limited functional range

SPLISS OVERVIEW

Arne Rempke, German Aerospace Center (DLR), Institute of Software Methods for Product Virtualization, January 24th, 2023

Key features of a linear solver for aeronautical CFD



Sparse matrices

- Dense blocks with a fixed block size or variable block sizes
- Mixed data types: e.g. some entries are complex, others real, some multiscalars

Solver

- Different components should be combinable (as preconditioner)
- Robust methods for stiff CFD problems:
  - Direct inversion of (generalized) diagonal blocks (LU/Thomas-Algorithm)
  - Jacobi, Gauss-Seidel, GMRES, linear multigrid, …

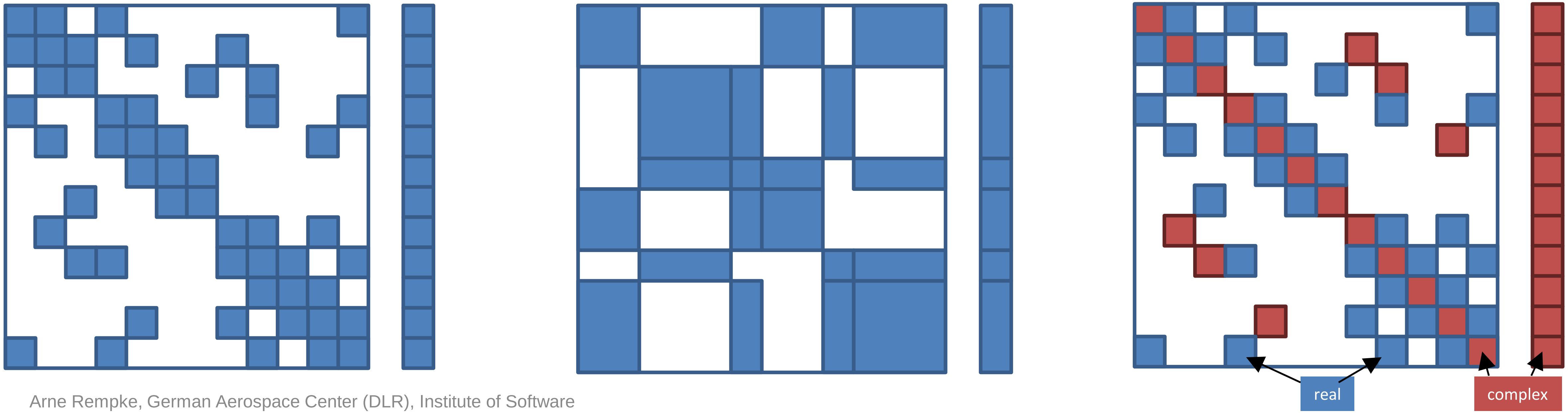
Efficient parallelization for HPC

- Distributed memory (GASPI, MPI)
- Shared memory (Threading)
- GPU support
- Vector instructions (SIMD)

Matrix Structure

Sparse matrices with dense blocks

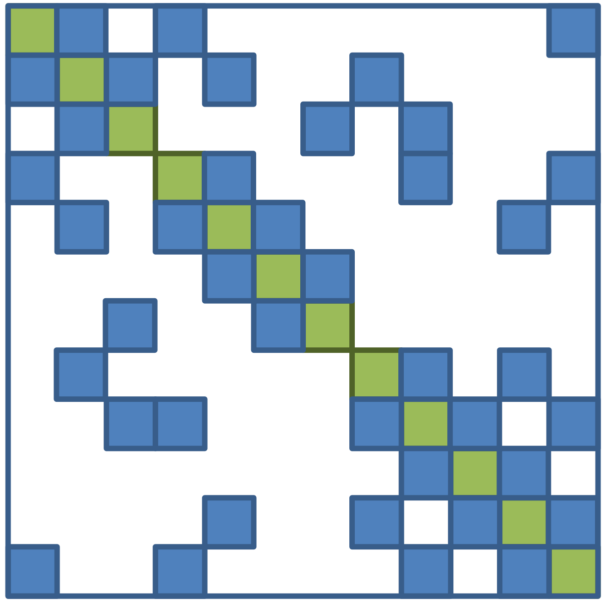
- Blocks of fixed size (e.g. 5x5, 7x7, 12x12 for all blocks within a single sparse matrix) (finite-volume Euler or RANS method)
- Blocks of variable sizes within one sparse matrix (e.g. 12x12, 48x48, 120x120 and 240x240 in one sparse matrix) (mixed-order Discontinuous-Galerkin method)
- Mixed data types: e.g. some entries are **complex**, others **real** (time-spectral/harmonic balance method)

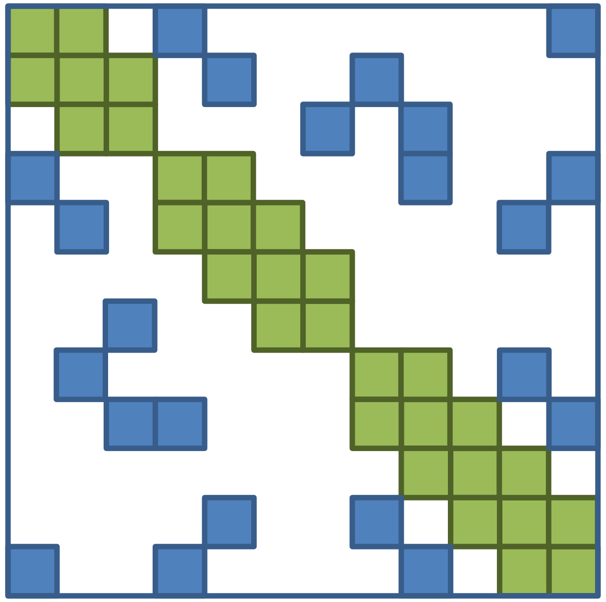


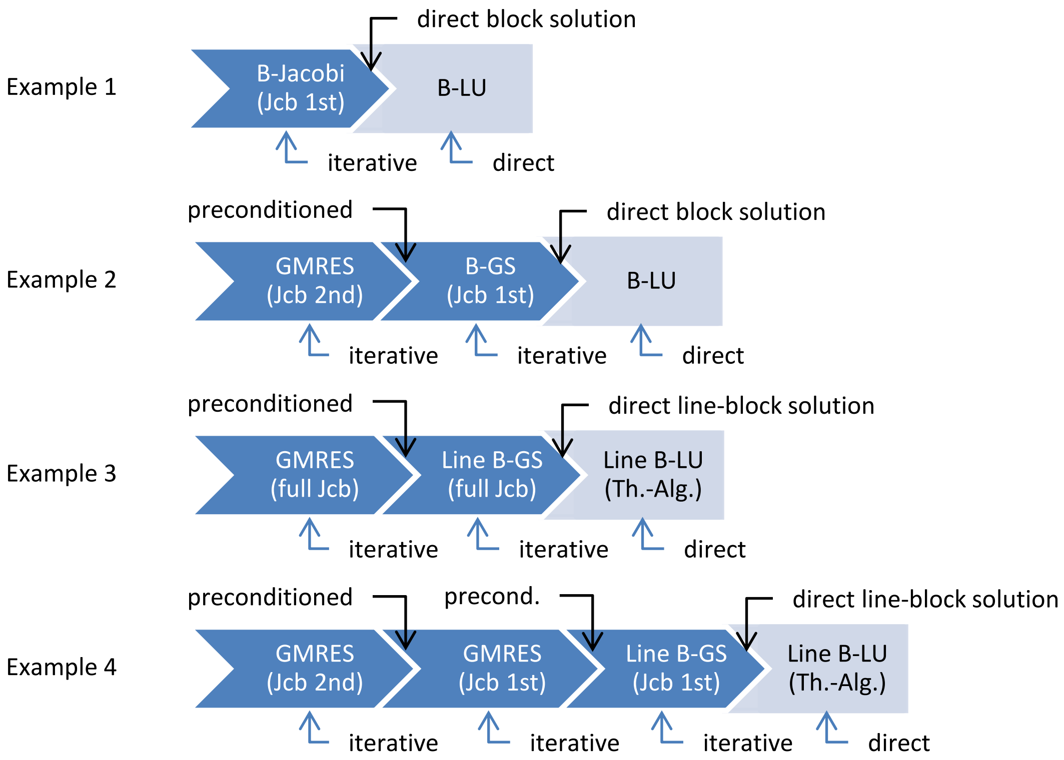
Solver Structure

Robust methods for stiff CFD problems:

- Block- and line-implicit methods relying on a direct solution of diagonal blocks (LU) or tridiagonal blocks (=lines, Thomas-Algorithm)
- Jacobi, Gauss-Seidel, GMRES, linear multigrid, …



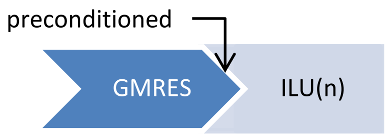




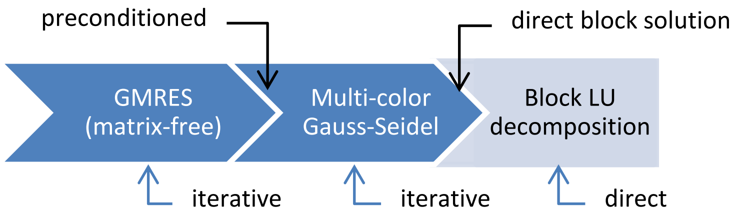
ALGORITHMICAL FEATURES

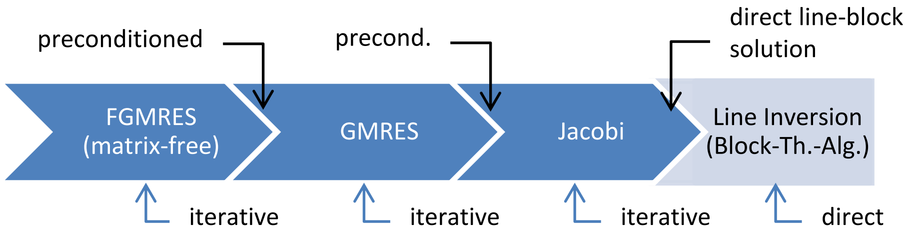
Flexible solver components

- Standard linear algebra packages provide solver/preconditioner combination:



- Spliss supports to chain multiple solver components, even with different linear operators:





Solver chaining

right preconditioned

direct diagonal solution

GMRes(A) → Jacobi(A) → LU(A_{Diagonal})

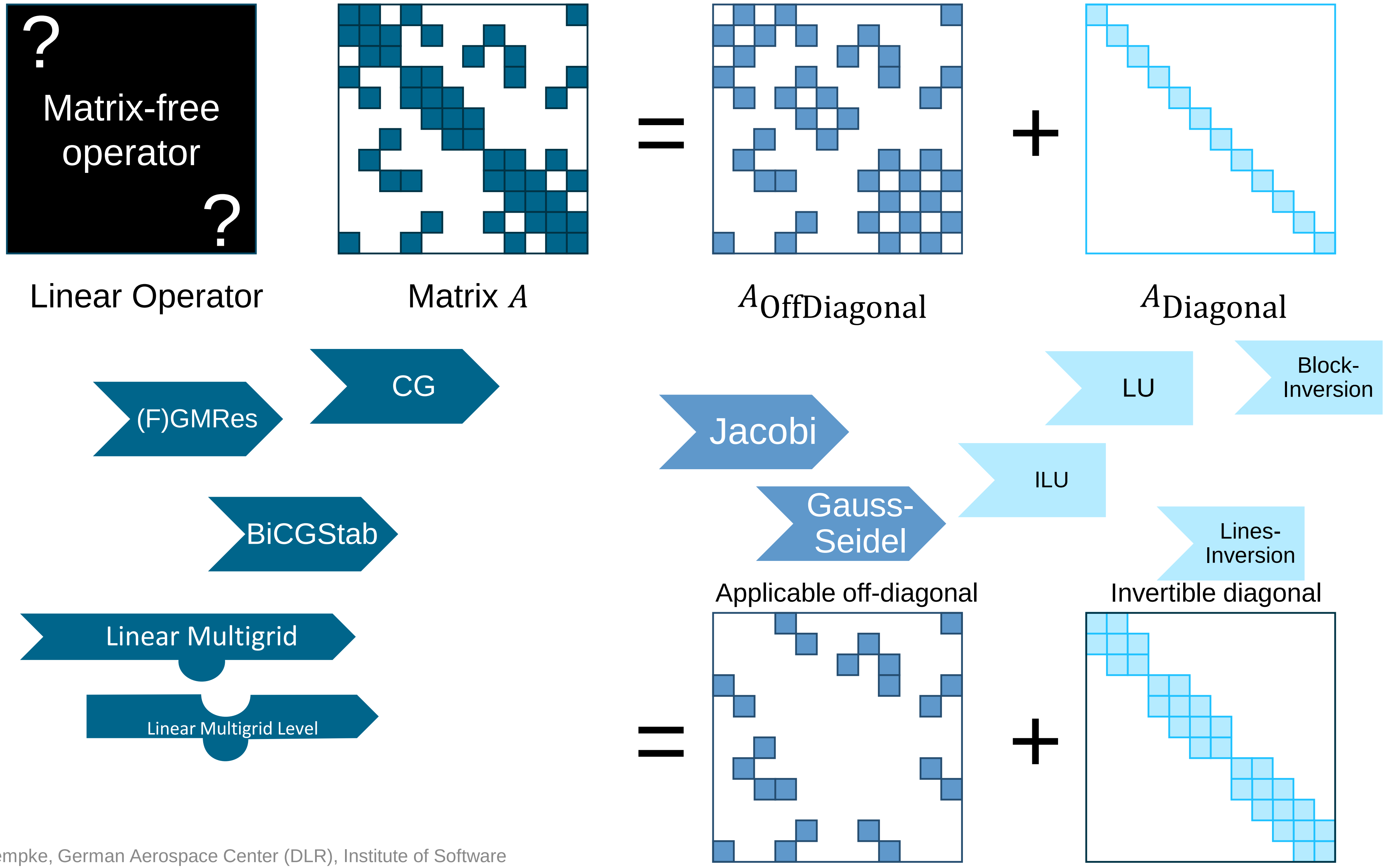
`x = GMRes(A).Apply(b):`
- $v_0 = b - A\,x$
- `for` $i = 0, \ldots, \text{maxIts}$:
  - $w = A(\text{Successor}(v_i))$
  - $v_{i+1} = \text{Orthonormalize}(w)$
  - $\text{Update}(H, \gamma)$
- `Solve` $H\,y = \gamma$
- $w = \sum_i y_i v_i$
- $x \mathrel{+}= \text{Successor}(w)$

`x = Jacobi(A).Apply(b):`
- `if` $(A - A_{\text{OffDiagonal}} = \text{Successor}_{\text{Matrix}})$:
  - `for` $i = 0, \ldots, \text{maxIts}$:
    - $r = b - A_{\text{OffDiagonal}}\,x$
    - $x = (1 - \lambda)x + \lambda\,\text{Successor}(r)$
- `else:`
  - `for` $i = 0, \ldots, \text{maxIts}$:
    - $r = b - A\,x$
    - $x \mathrel{+}= \lambda\,\text{Successor}(r)$

`x = LU(D).Apply(b):`
- $x = D^{-1} b$

Note that in case the matrices for different solver components match really well, an optimized version is applied

Featured Solver Components



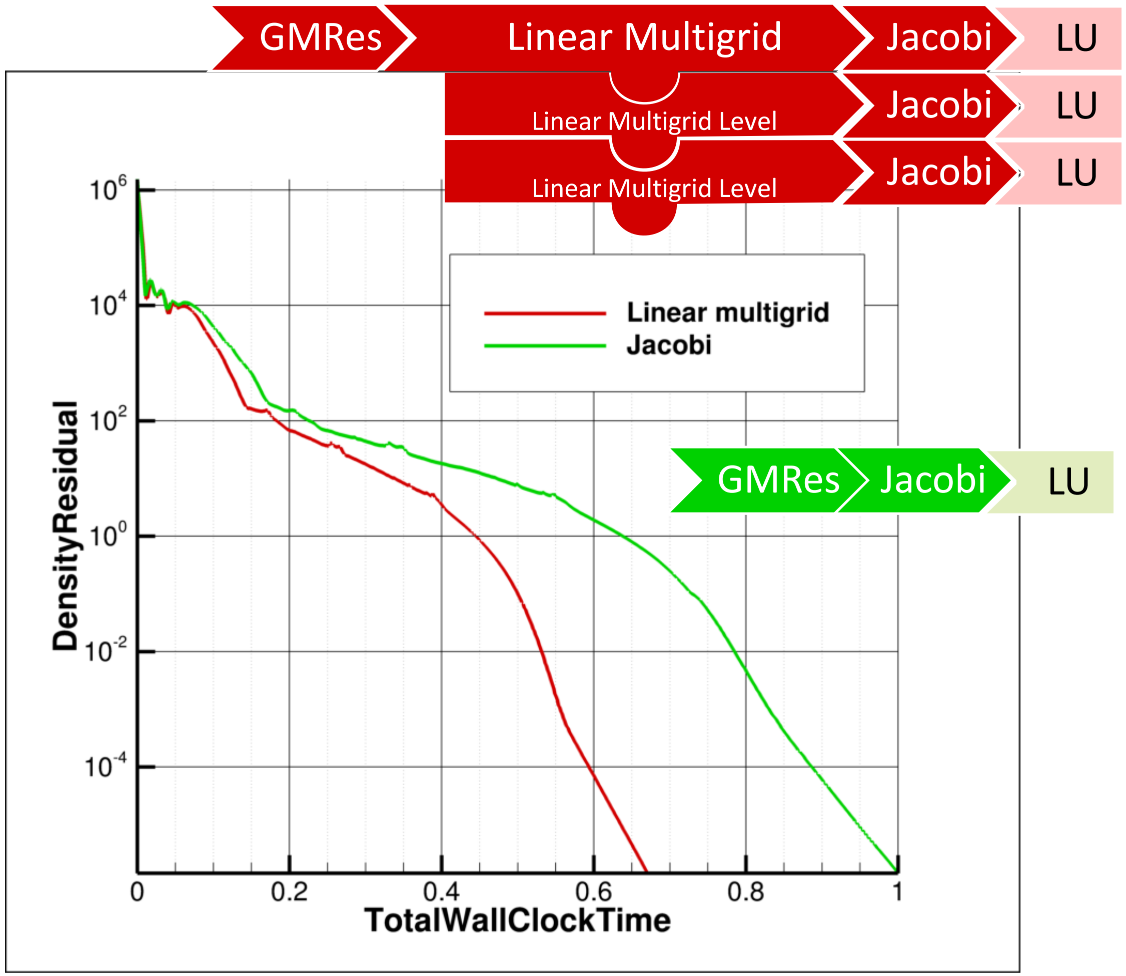
Multigrid Solver Component

Flexible integration

- Each level can use its own smoother
- Transfer operators can be user-provided

RAE2822 65k elements, CODA

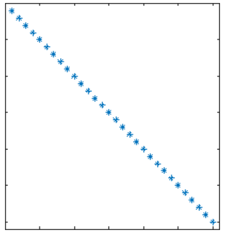
- Reduction of time to solution by 1/3 already for very small test case



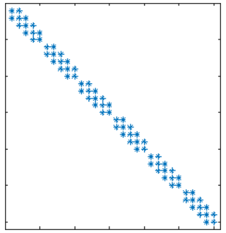
Lines Inversion / Thomas Algorithm

- Jacobi-method uses a diagonal inversion:

$$x^{(i+1)} := x^{(i)} + D^{-1}\left(b - Ax^{(i)}\right)$$



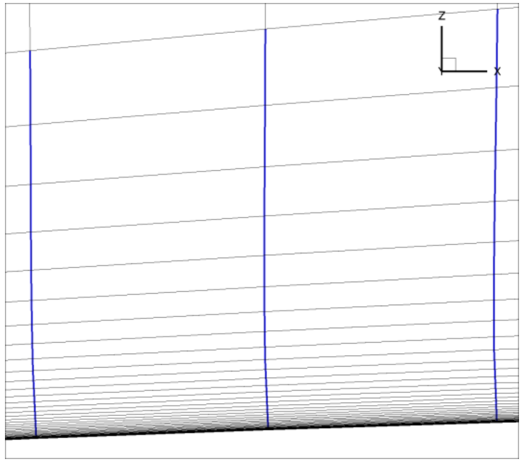
where



- $D := \mathrm{diag}(A)$ (point-implicit)

or

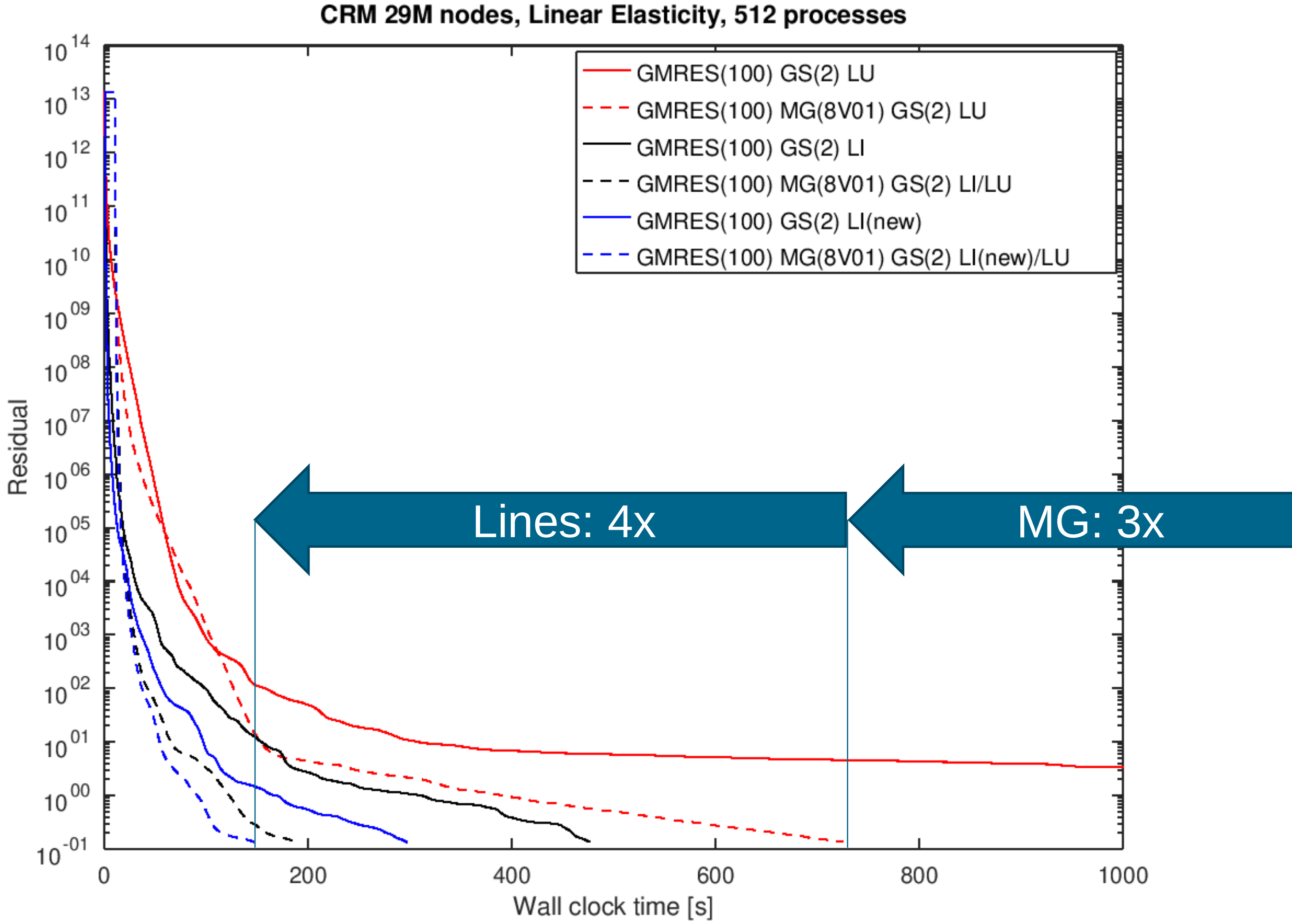
- $D := \mathrm{tridiag}(A)$ (lines-implicit)



- Especially favourable/needed when mesh has very anisotropic cells, aspect ratios ≥5000:1

Efficiency of the tailored solver components

- Red solid curve is a „standard linear solver“
- Multigrid gives speedup of 2-3 (dashed)
- LinesInversion gives additional speedup of 3-4 (black/blue)



PARALLELIZATION ASPECTS

Main Operation during Solving: d = A · s

With:

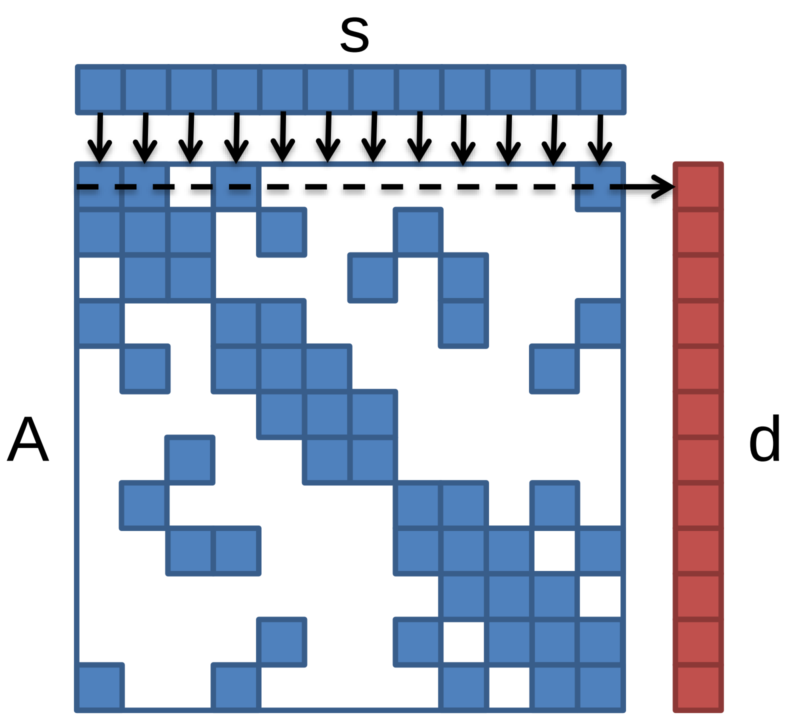
s: Source Vector

d: Destination Vector

A: Matrix

Formula: $\boldsymbol{d_i} = \sum_{j=0}^{N} \boldsymbol{A_{ij}} \cdot \boldsymbol{s_j}$

→ All rows can be computed independently.



Distributed Memory Parallelization

Program run:

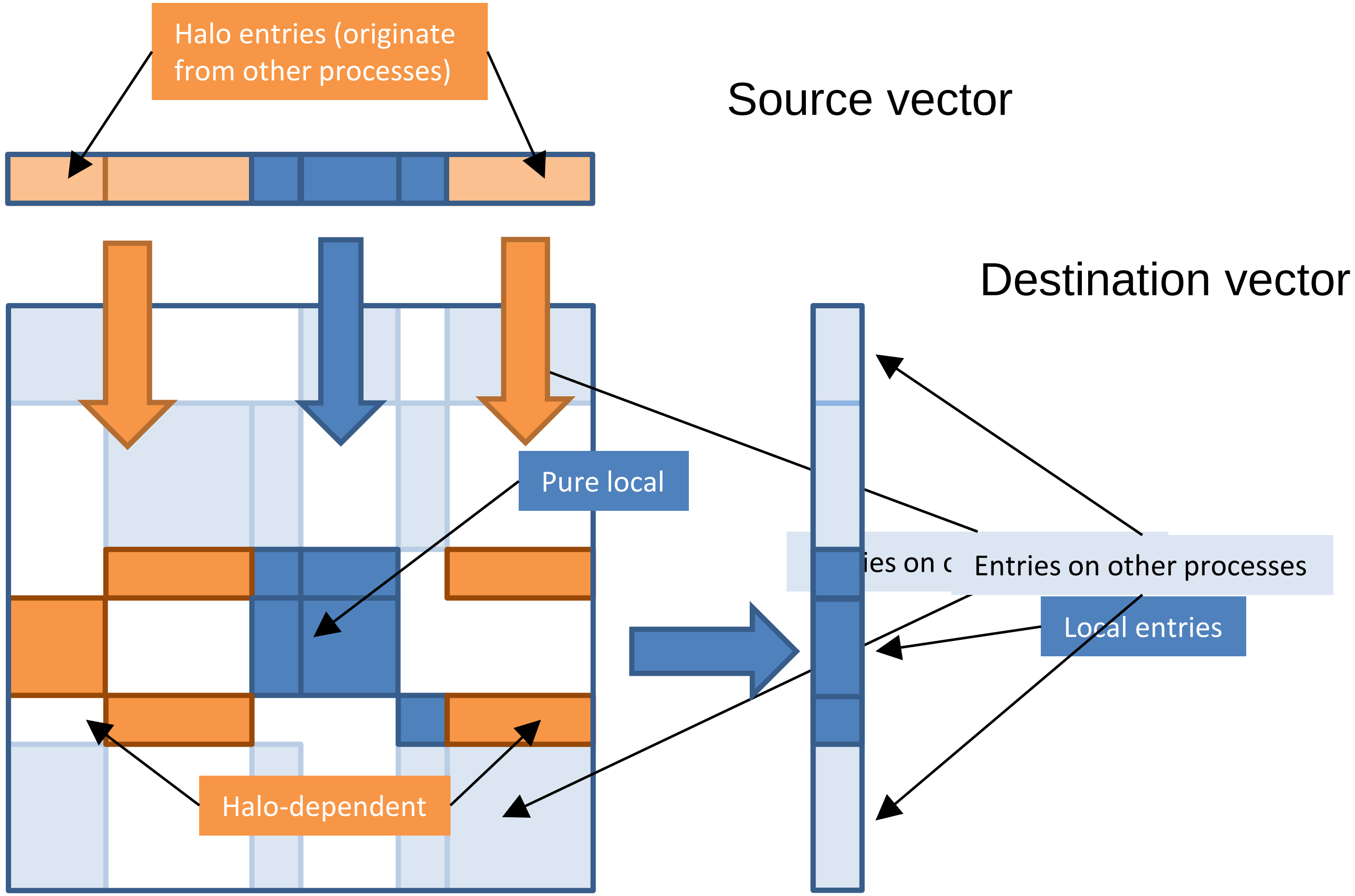
Start sending halo data

↓

Computation of pure local parts

↓

Computation of halo parts



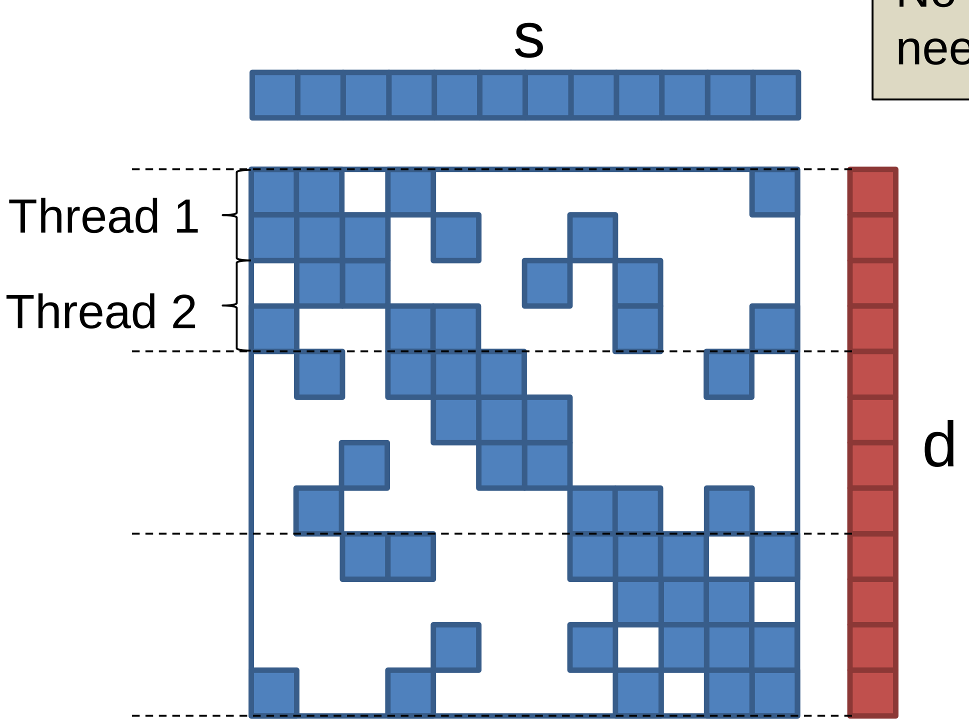
Arne Rempke, German Aerospace Center (DLR), Institute of Software Methods for Product Virtualization, January 24th, 2023

Shared Memory Parallelization

Straightforward derived from cluster level parallelization

- Every thread computes some rows
- Same strategy on CPU and GPU

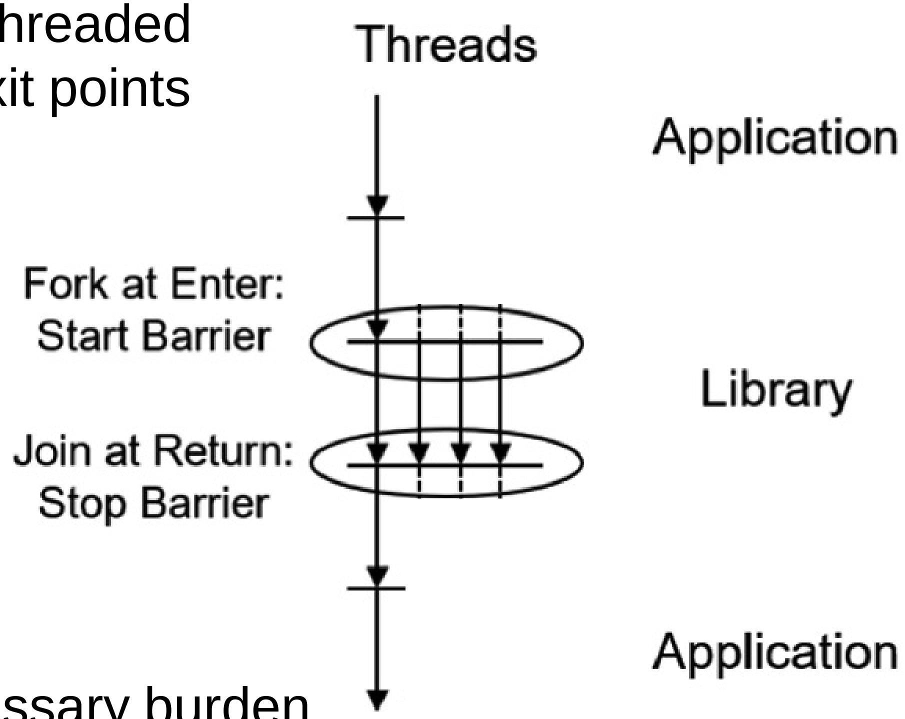
No synchronization needed for write



Arne Rempke, German Aerospace Center (DLR), Institute of Software Methods for Product Virtualization, January 24th, 2023

Threading model

- Typical design of a library
  - Single threaded entry/exit points



  - Unnecessary burden if the user code also uses threads

- Spliss design
  - Allows to enter/exit with all threads

GPU Parallelization

- Similar as for Multithreading
- Using alpaka* allows us to write a single Kernel to be executed on CPU or GPU
- Spliss hides the CUDA backend/compiler/… from user code:
  - Explicit template instantiations of CUDA-dependent classes on Spliss compilation
  - No necessity to use nvcc for user code
- Since Spliss is a C++ template library, user calls to small functions, e.g.

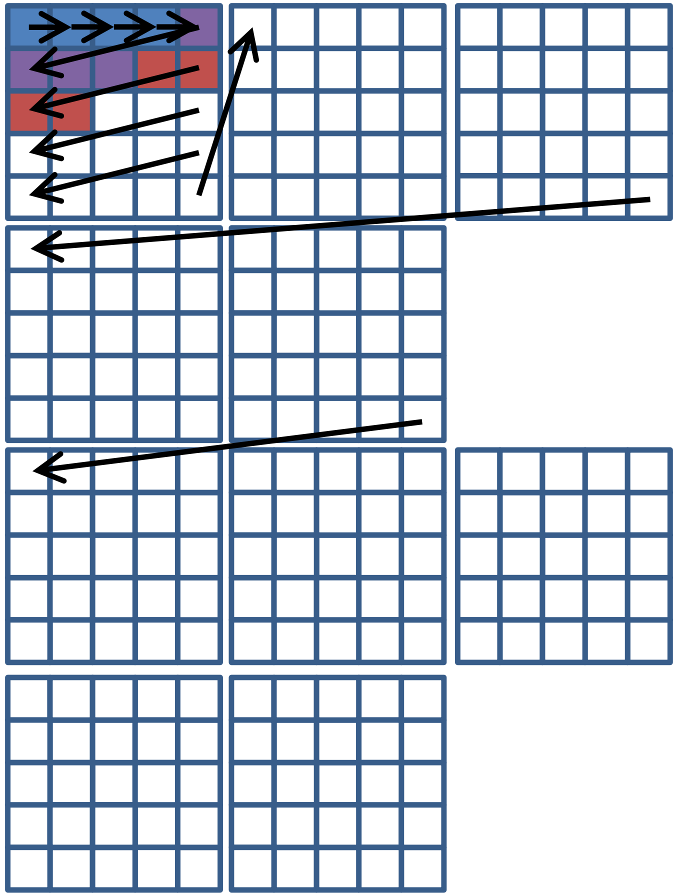
  ```
  A[row][col] += myContribution;
  ```

  can still be inlined, allowing a seamless integration while capsulating the actual memory layout

* https://github.com/alpaka-group/alpaka

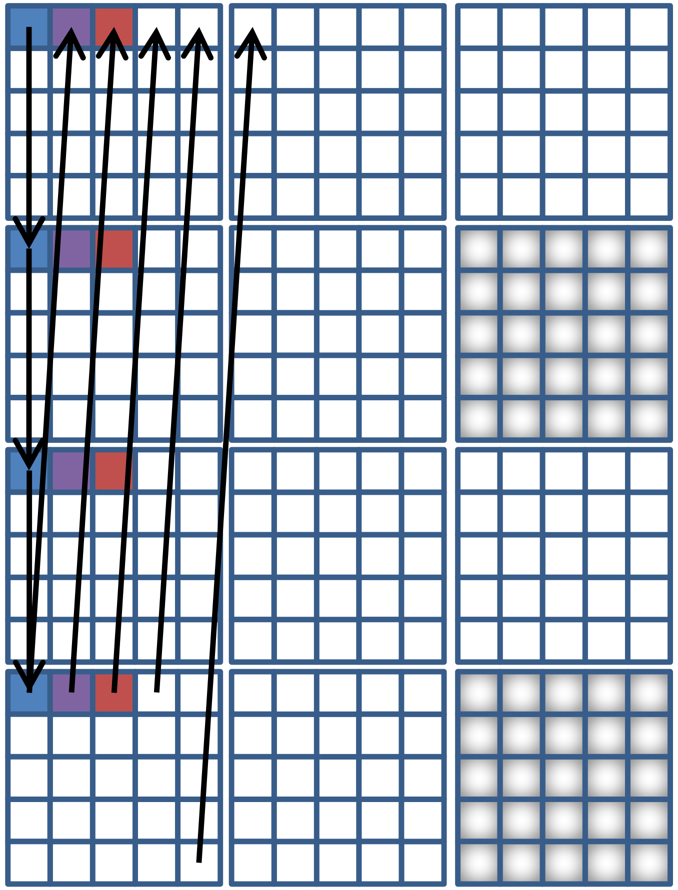
SIMD Parallelization / Memory Layout

Compact Block Layout



- All entries of one matrix block are consecutively stored
- All matrix blocks are row-wise consecutively stored

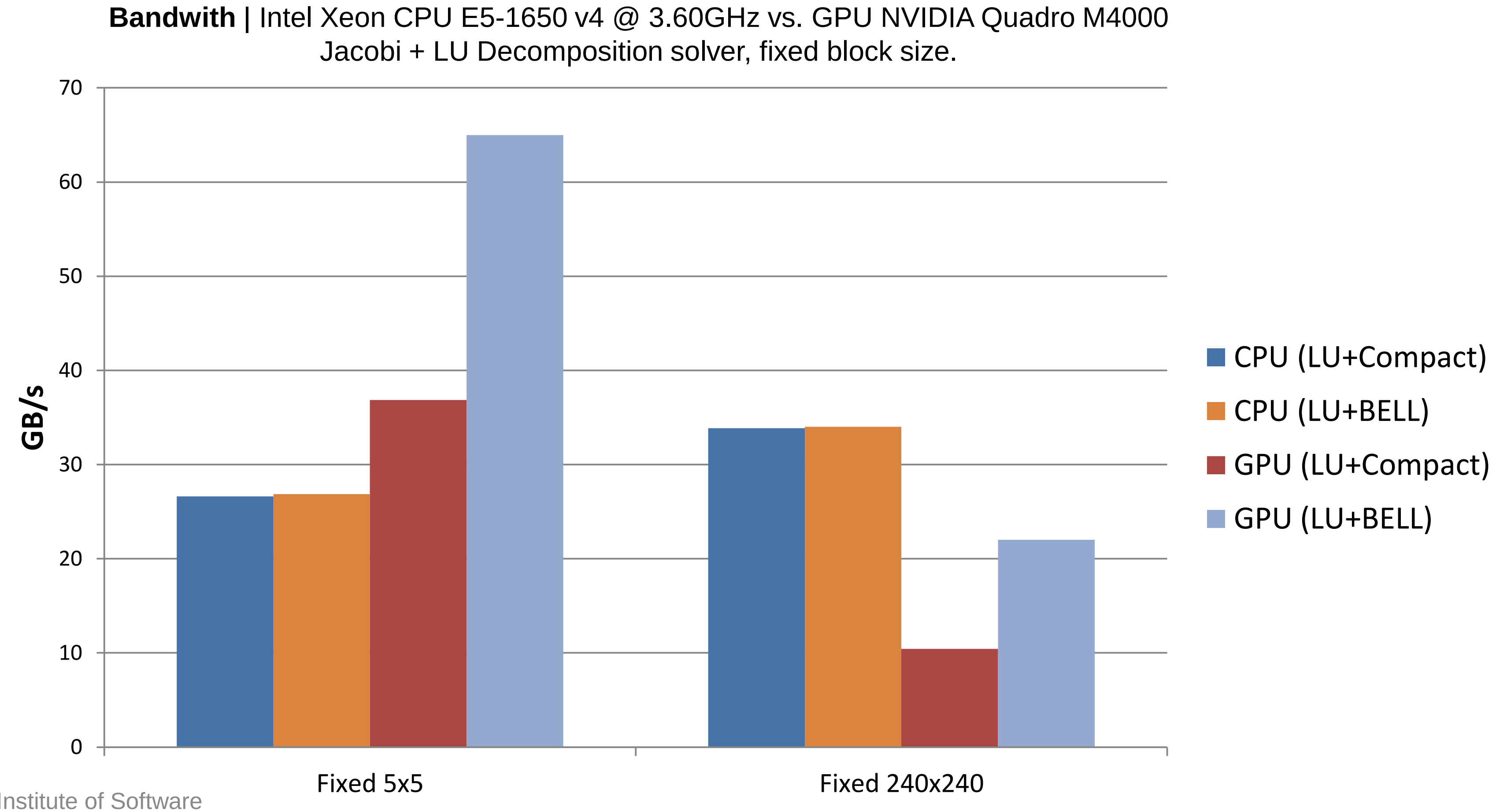
BELL Layout for fixed block sizes



- Matrix block entries are interlaced stored
- A number (usually the SIMD vector size) of consecutively stored entries belong to the same coordinate of matrix blocks
- A matrix block row may end with padding blocks.

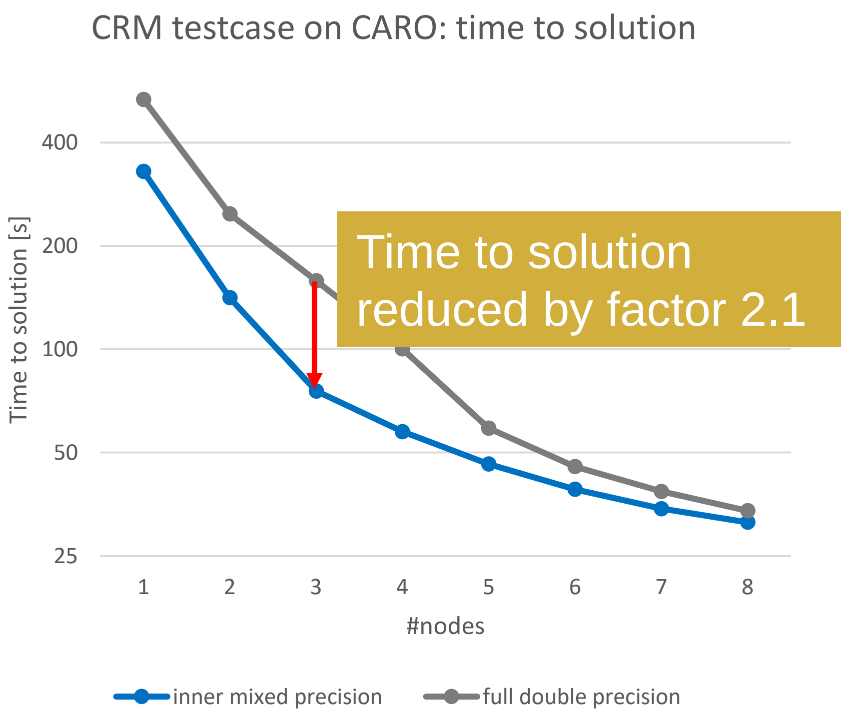
SIMD Parallelization: Performance

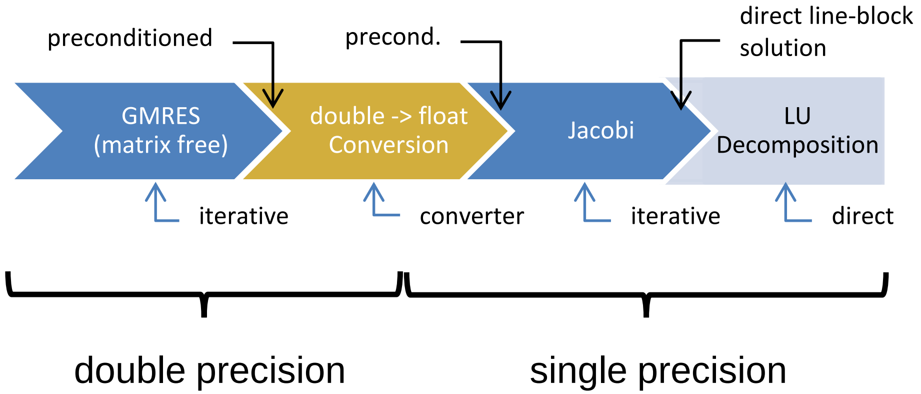
BELL has nearly no effect on CPU, but huge effects on GPU



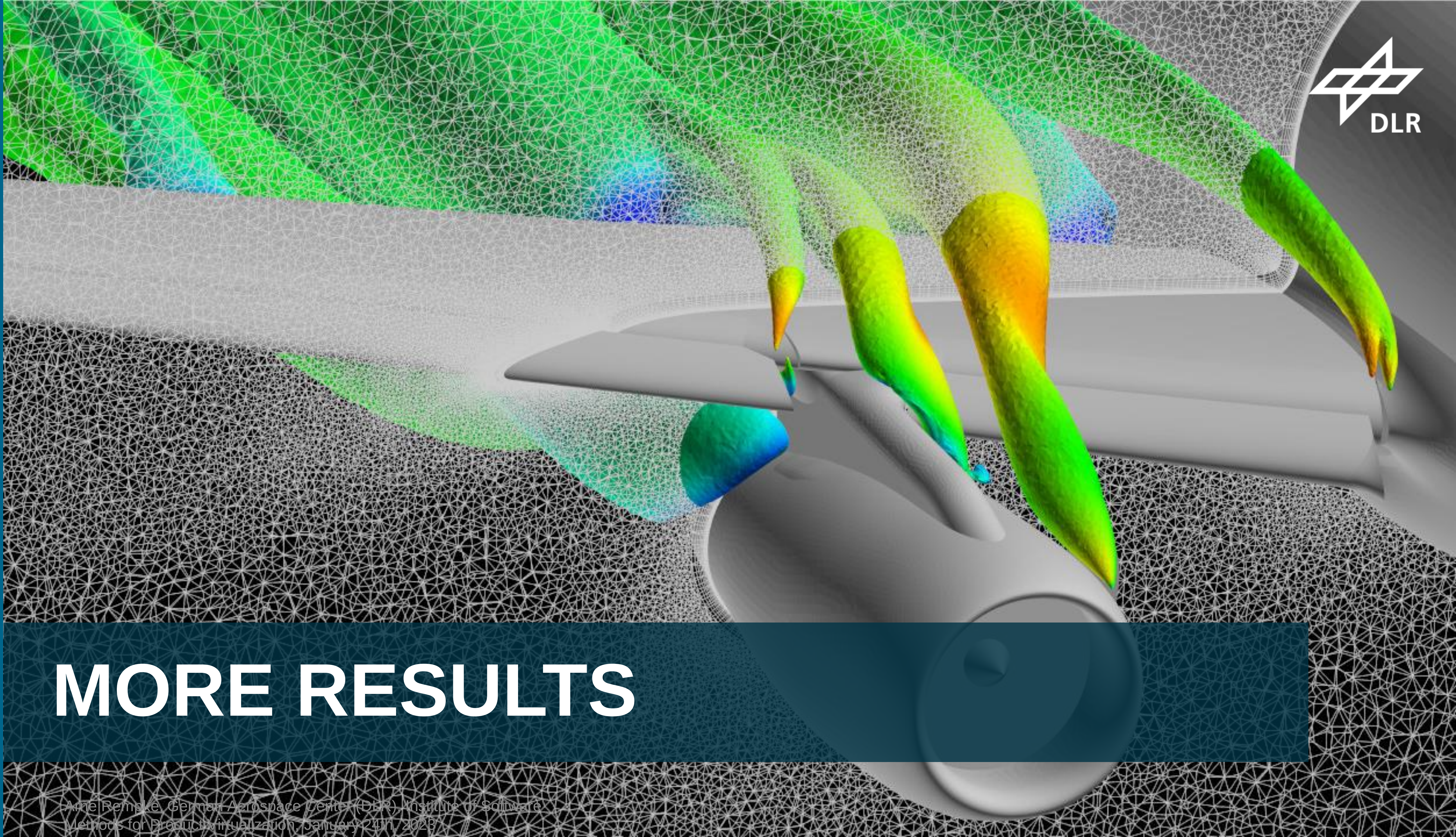
Mixed precision

- Idea: Reduce memory footprint of inner hot loops since performance is memory bound





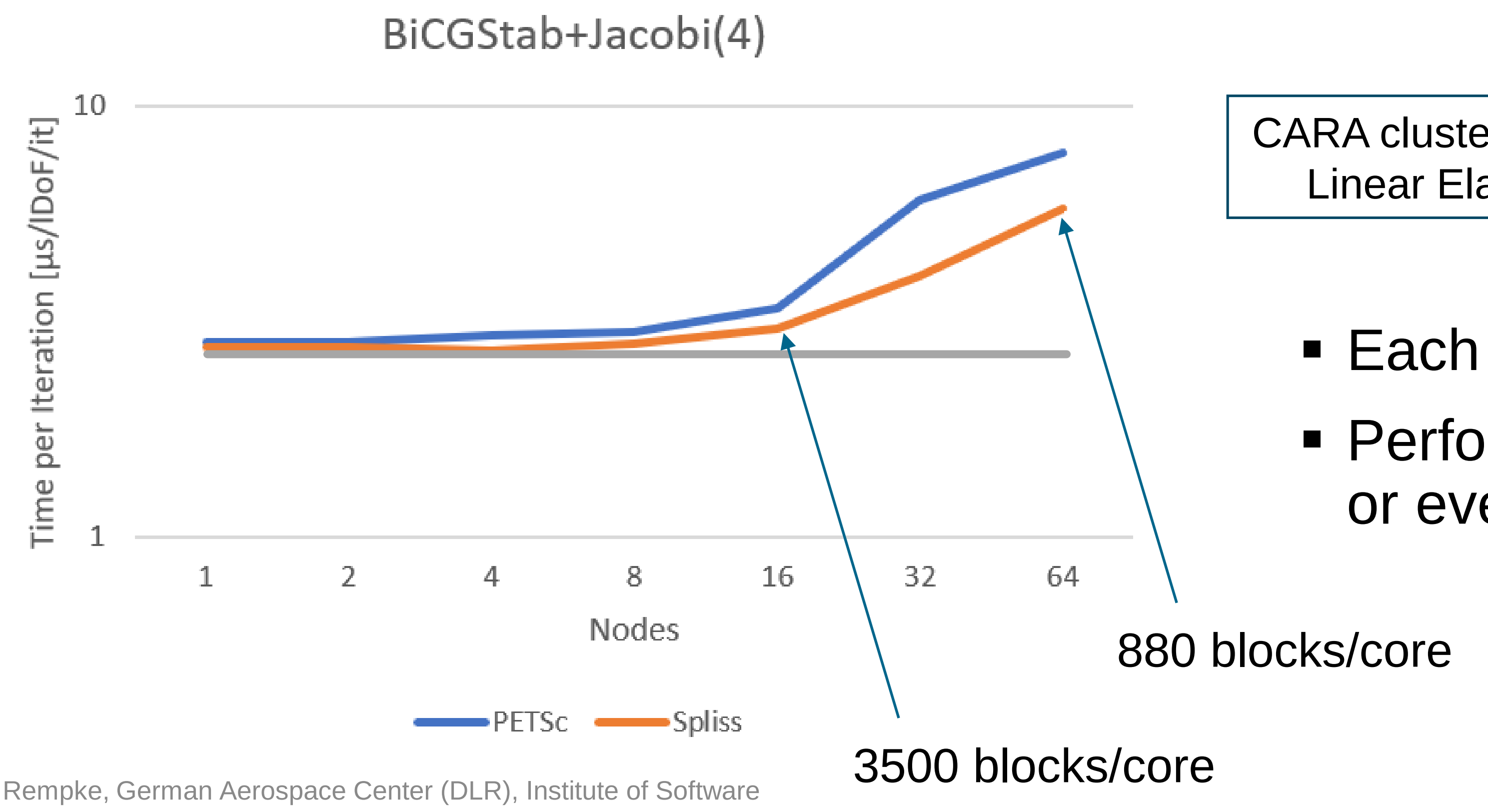
- User still provides matrix / input vectors and receives solution vector in double precision
- Inner Spliss solver components operate in float precision



MORE RESULTS

Comparison to PETSc

- Problem size: 3.6M x3 degrees of freedom
- Solve single linear equation system for residual reduction of 1e-14
- Runtime per DoF and iteration

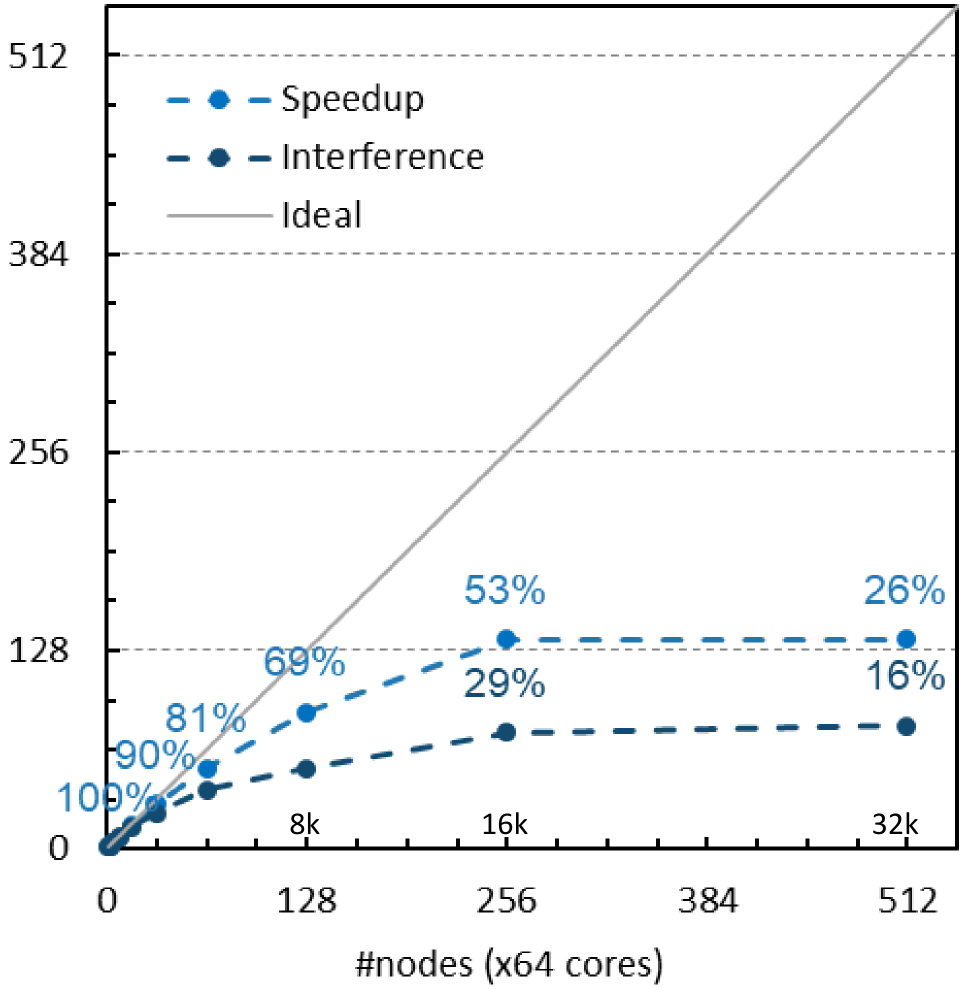


CARA cluster, Spliss 1.3.0, PETSc 3.13.1
Linear Elasticity CRM 3.6M vertices

- Each node runs 64 processes
- Performance and scaling similar or even better

Scalability Evaluation on CARA
CODA release 2022.04, Spliss release 2.0.1

Speedup | CARA (AMD Epyc 7601)
CRM, 24.1M elements, implicit Euler, Jacobi + LU



Observation

- 93% efficiency at 1k cores, 53% efficiency at 16k cores
- No additional gain for 32k cores
- Good scaling for small mesh (735 elements/core at 32k)
- Setup 16 processes / 4 threads per node scales best

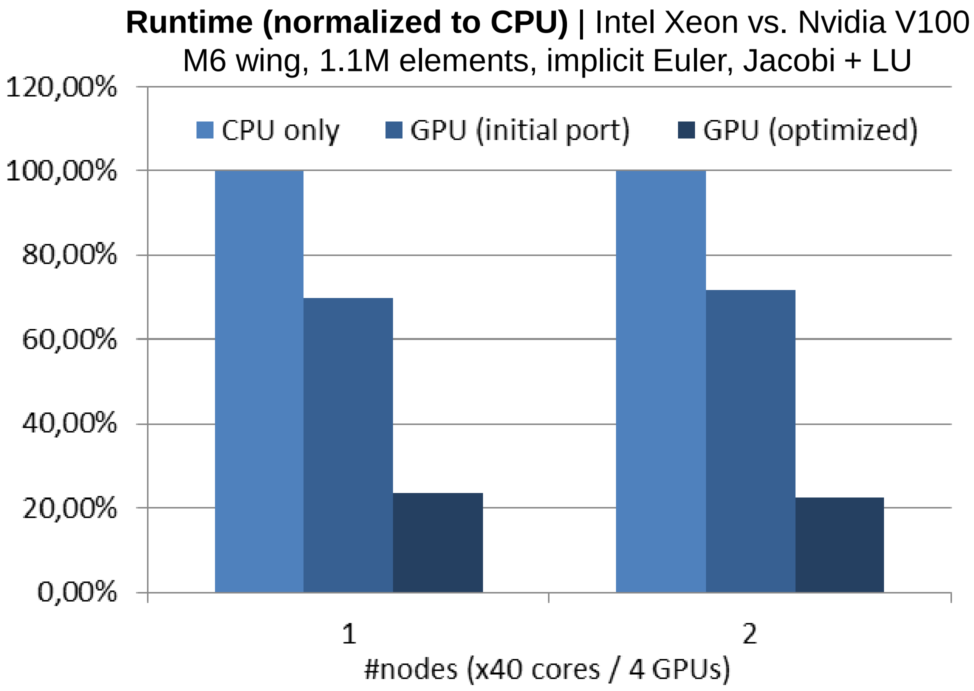
- Up to 1.8x slow-down with network interference*
- ParMetis about 2-5% slower than Zoltan

* Reproducible with e.g. other CODA runs on close-by nodes

GPU Development
From initial GPU support to Spliss 2.0

Improvements of GPU (optimized) vs. GPU (initial)

- Using CUDA-aware MPI communication to eliminate unnecessary device-to-host transfers
- GPUDirect accelerations at runtime allow communication without involvement of host memory
- Nvidia MPS (multi-process service) allows multiple processes to efficiently offload to the same GPU
- Optimized host-to-device transfers



GPU (initial): all data transfers via host & device

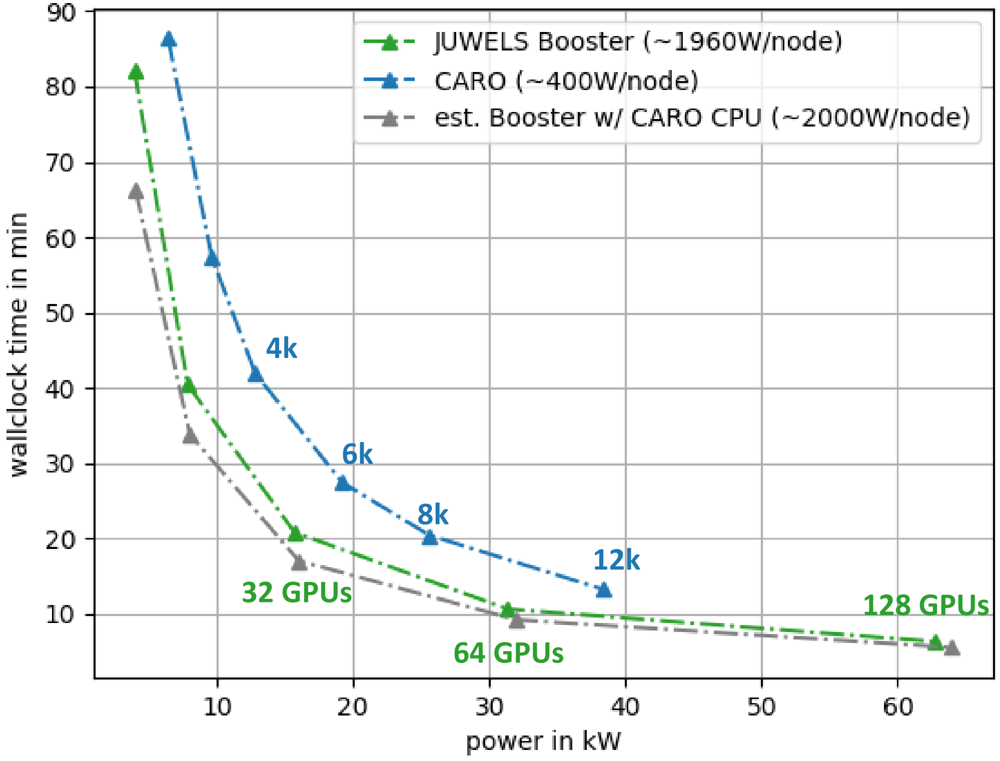
GPU Development
Next gen GPUs

Juwels Booster (Jülich)

- 4x Nvidia Tesla A100 per node

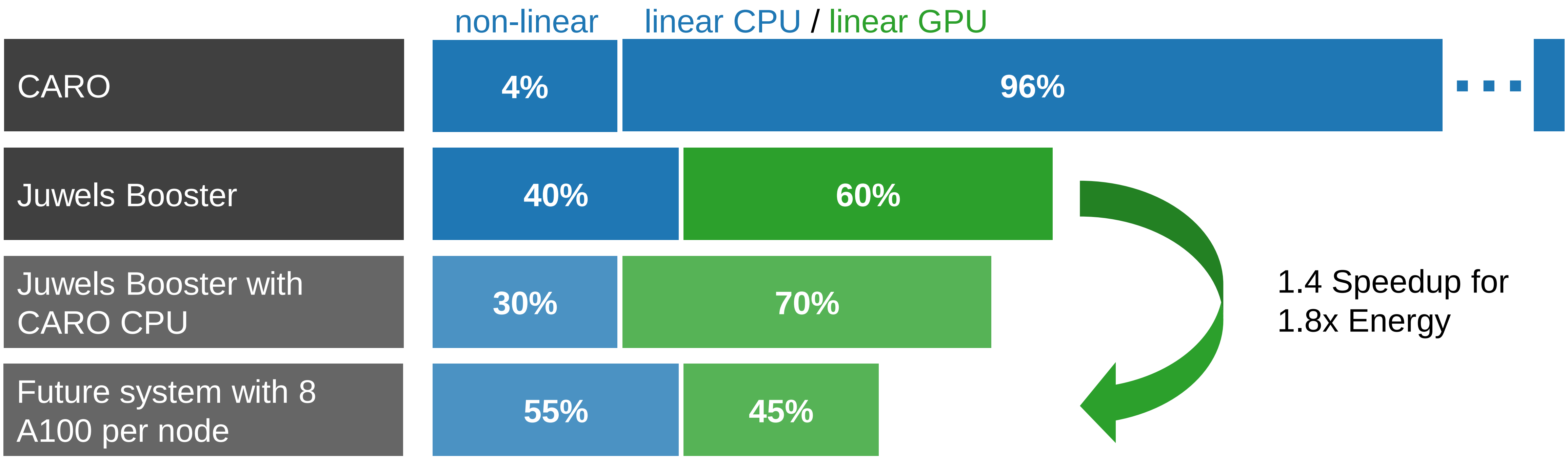
- Time to solution: speedup of 8-9 for same number of nodes on Juwels
  - Rather unfair, since on Juwels every process uses a GPU **in addition** to the CPU
- Energy comparison (seconds per used Watt): speedup of 1.6-1.9 on Juwels
- Hypothetical Juwels Booster node with CARO CPU: 1.8-2.3 speedup (energy-wise)

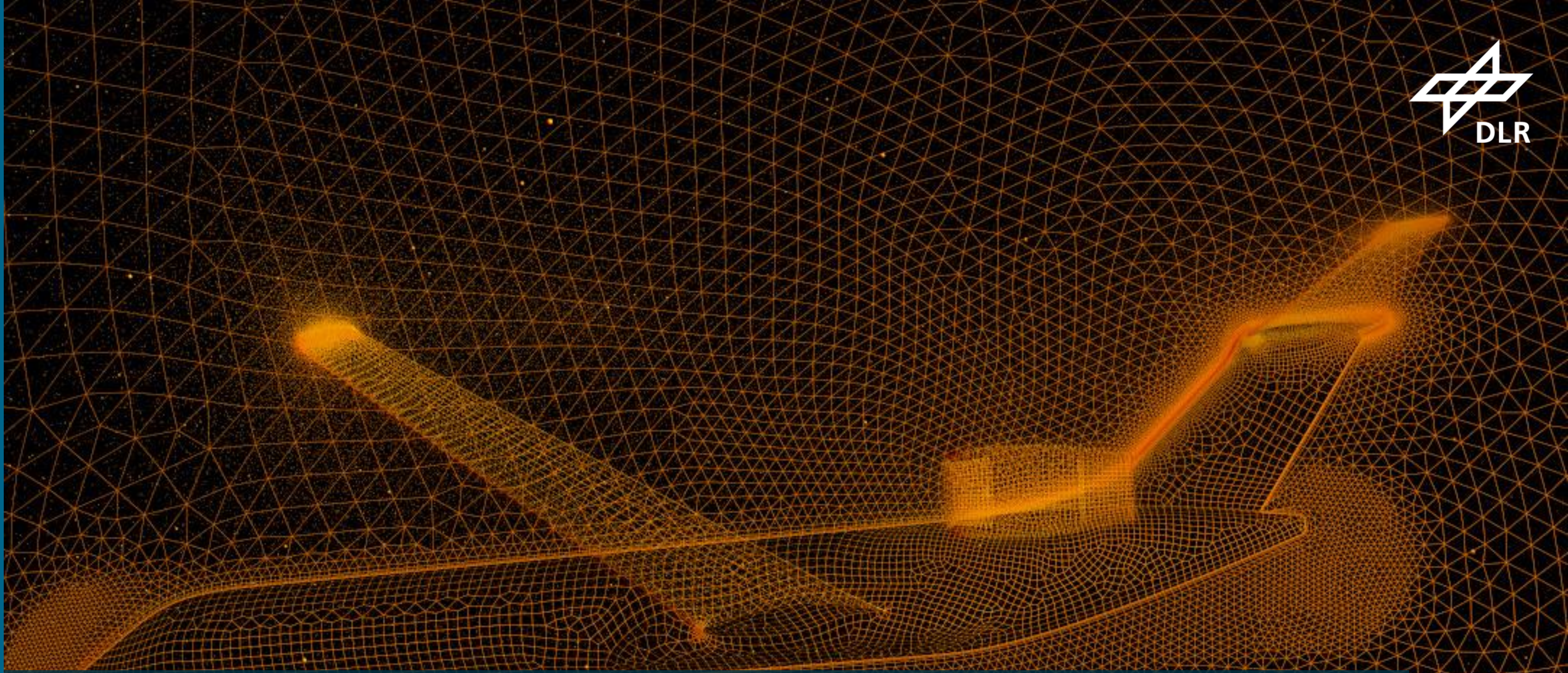
Runtime | CARO (AMD Rome) vs. Juwels (4x Nvidia A100)
M6 wing, 69.2M elements, implicit Euler, Jacobi + Block Inv.



GPU Development
Impact of GPU support of Spliss on application wallclock time

- For implicit methods in CODA, linear equation systems are solved via Spliss
- Thus, only the linear part of CODA benefits from GPUs
- For future system with more or more powerful GPUs the non-linear part may become bottleneck





QUESTIONS?