Nonlinear Computational
Aeroelasticity Lab

# ACCELERATION OF A COMPUTATIONAL FLUID DYNAMICS CODE WITH GPU USING OPENACC

NICHOLSON K. KOUKPAIZAN
PHD. CANDIDATE

CREATING THE NEXT®

GPU Technology Conference 2018, Silicon Valley
March 26-29 2018

# CONTRIBUTORS TO THIS WORK

- **GT NCAEL Team members**
  - N. Adam Bern
  - Kevin E. Jacobson
  - Nicholson K. Koukpaizan
  - Isaac C. Wilbur

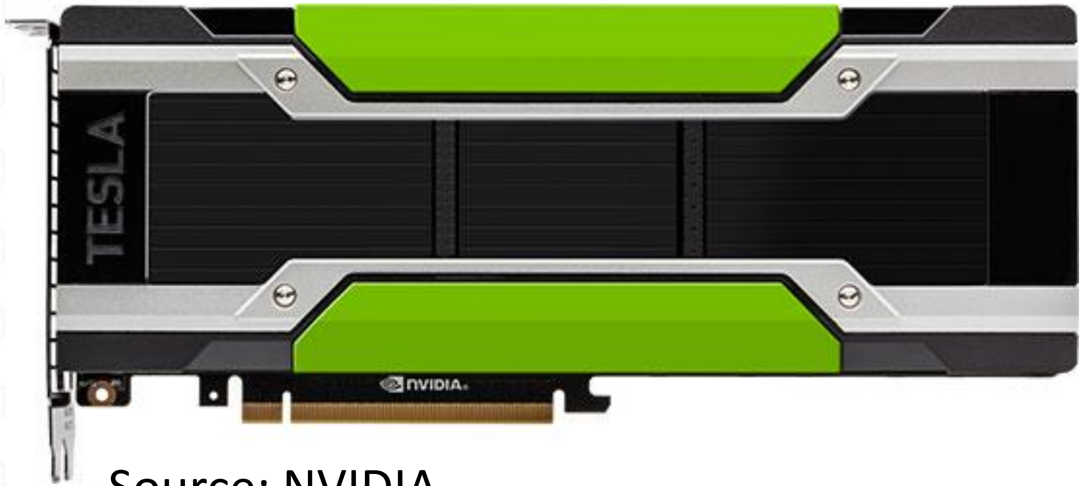
- **Mentors**
  - Matt Otten (Cornell University)
  - Dave Norton (PGI)
- **Advisor**
  - Prof. Marilyn J. Smith

- Initial work done at **the Oak Ridge GPU Hackathon** (October 9th-13th 2017)
  - "5-day hands-on workshop, with the goal that the teams leave with applications running on GPUs, or at least with a clear roadmap of how to get there." (olcf.ornl.gov)

# HARDWARE

Georgia Tech

- Access to summit-dev during the Hackathon
  - IBM Power8 CPU
  - NVIDIA Tesla P100 GPU - 16 GB
- Access to NVIDIA's psg cluster
  - Intel Haswell CPU
  - NVIDIA Tesla P100 GPU- 16 GB

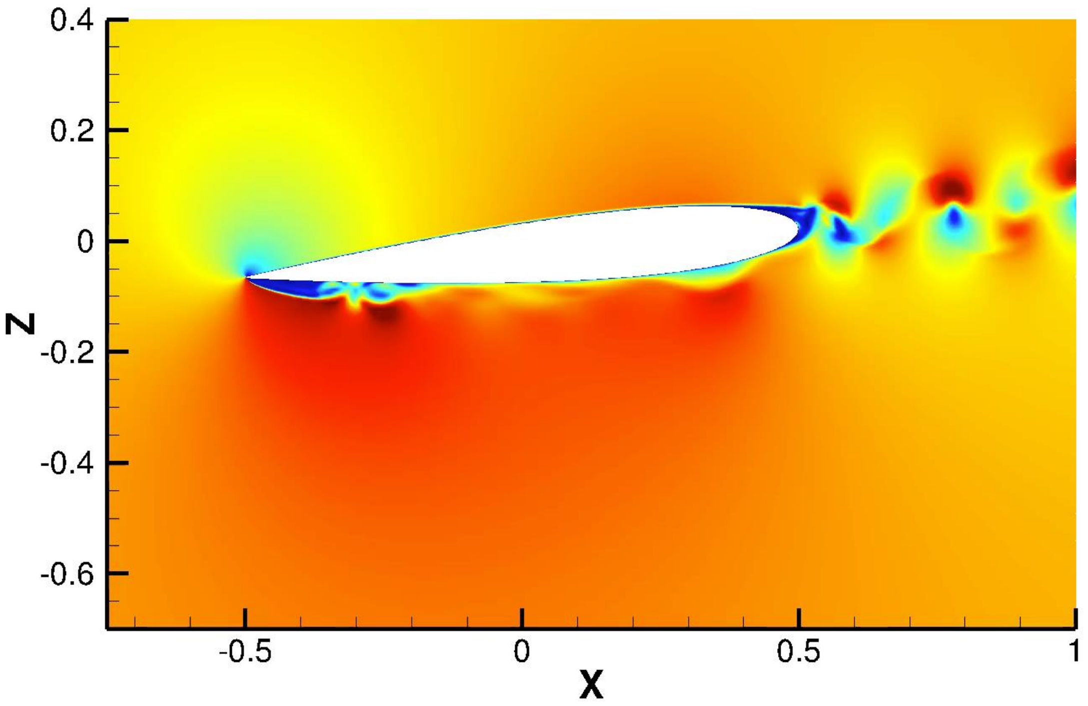



Source: NVIDIA (http://www.nvidia.com/object/tesla-p100.html)

# APPLICATION: GTSIM

Georgia Tech

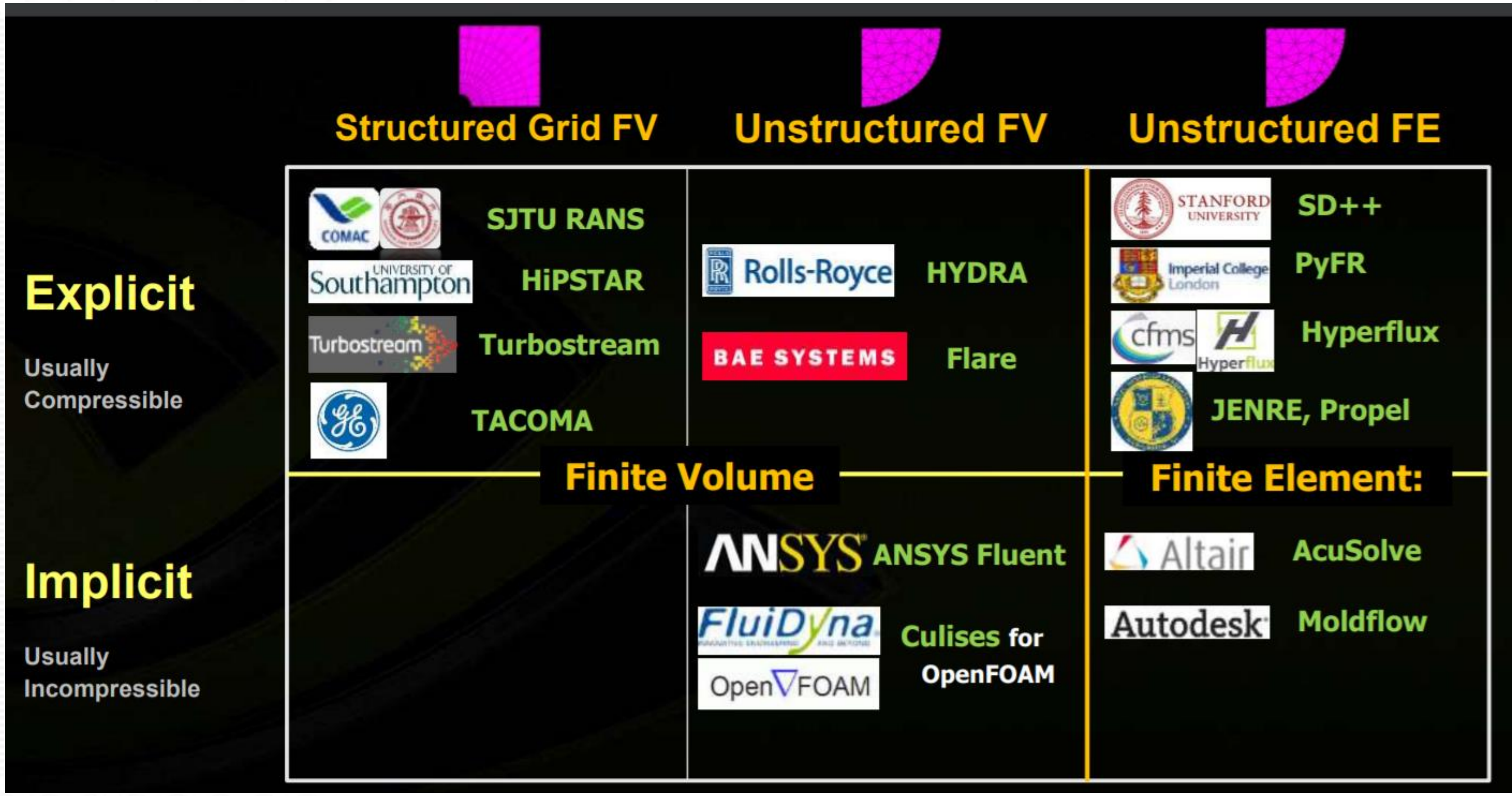
- Validated Computational Fluid Dynamics (CFD) solver
  - Finite volume discretization
  - Structured grids
  - Implicit solver
- Written in Free format Fortran 90
- MPI parallelism
- Approximately 50,000 lines of code
- No external libraries
- Shallow data structures to store the grid and solution




Reference for GTSIM: Hodara, J. PhD thesis “Hybrid RANS-LES Closure for Separated Flows in the Transitional Regime.” smartech.gatech.edu/handle/1853/54995

# WHY AN IMPLICIT SOLVER?

- Explicit CFD solvers:
  - Conditionally stable
- Implicit CFD solvers:
  - Unconditionally stable
- Courant-Friedrichs-Levy (CFL) number dictates convergence and stability




Source: Posey, S. (2015), Overview of GPU Suitability and Progress of CFD Applications, NASA Ames Applied Modeling & Simulation (AMS) Seminar – 21 Apr 2015

# PSEUDOCODE

Read in the simulation parameters, the grid and initialize the solution arrays

**_Loop_** physical time iterations

**_Loop_** pseudo-time sub-iterations

Compute the pseudo-time step based on the CFL condition

Build the left hand side ($\overline{\boldsymbol{LHS}}$) → 40 %

Compute the right hand side ($\boldsymbol{RHS}$) →31%

Use an iterative linear solver to solve for $\Delta\boldsymbol{U}$ in $\overline{\boldsymbol{LHS}} \times \Delta\boldsymbol{U} = \boldsymbol{RHS}$ →24%

Check the convergence

**_end loop_**

**_end loop_**

Export the solution ($\boldsymbol{U}$)

# LINEAR SOLVERS (1 OF 3)

- Write $\overline{\boldsymbol{LHS}} = \overline{\boldsymbol{\mathcal{L}}} + \overline{\boldsymbol{\mathcal{D}}} + \overline{\boldsymbol{\mathcal{U}}}$
- Jacobi based (Slower convergence, but more suitable for GPU)

$$\Delta \boldsymbol{U}^{k} = \overline{\boldsymbol{\mathcal{D}}}^{-1}\left(\boldsymbol{RHS}^{k-1} - \overline{\boldsymbol{\mathcal{L}}}\Delta \boldsymbol{U}^{k-1} - \overline{\boldsymbol{\mathcal{U}}}\,\Delta \boldsymbol{U}^{k-1}\right)$$

OVERFLOW solver (NAS Technical Report NAS-09-003, November 2009) used Jacobi for GPUs

- Gauss-Seidel based (one of the two following formulations)

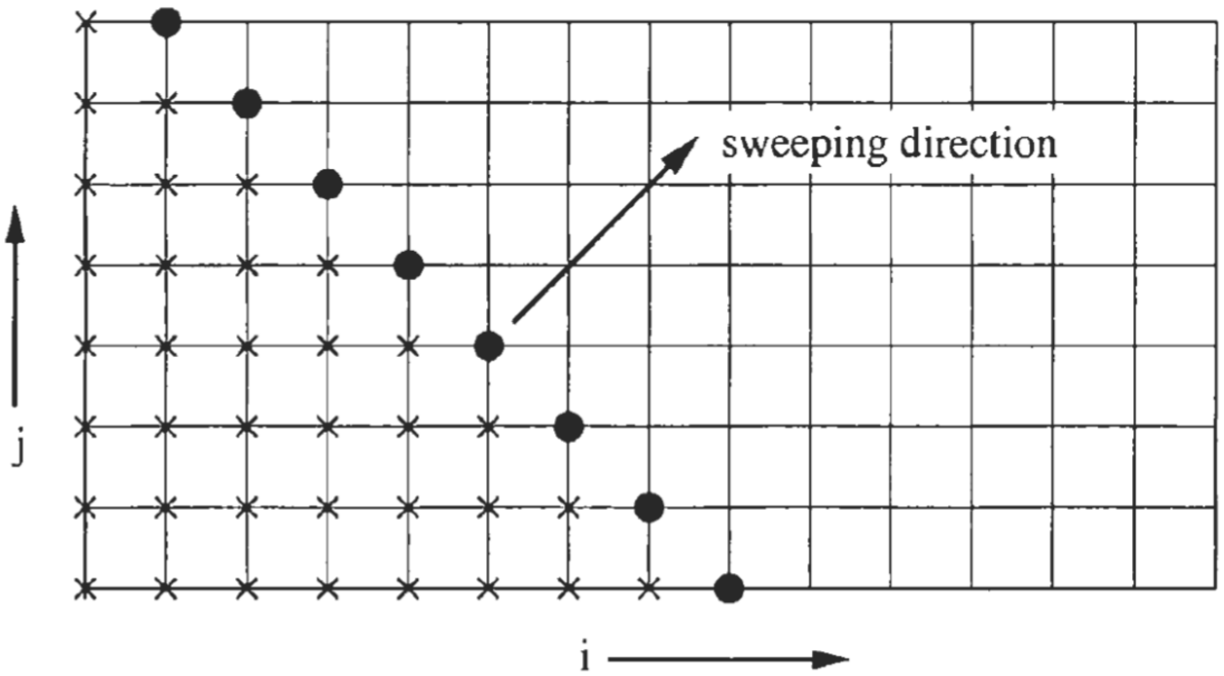
$$\Delta \boldsymbol{U}^{k} = \overline{\boldsymbol{\mathcal{D}}}^{-1}\left(\boldsymbol{RHS}^{k} - \overline{\boldsymbol{\mathcal{L}}}\,\Delta \boldsymbol{U}^{\boldsymbol{k}} - \overline{\boldsymbol{\mathcal{U}}}\,\Delta \boldsymbol{U}^{\boldsymbol{k-1}}\right)$$

$$\Delta \boldsymbol{U}^{k} = \overline{\boldsymbol{\mathcal{D}}}^{-1}\left(\boldsymbol{RHS}^{k} - \overline{\boldsymbol{\mathcal{L}}}\,\Delta \boldsymbol{U}^{\boldsymbol{k-1}} - \overline{\boldsymbol{\mathcal{U}}}\,\Delta \boldsymbol{U}^{\boldsymbol{k}}\right)$$

- Coloring scheme (red - black)
  - Red: Use the first Gauss-Seidel formulation, with previous iteration black cells data
  - Black: Use the second Gauss-Seidel formulation with the last Red update

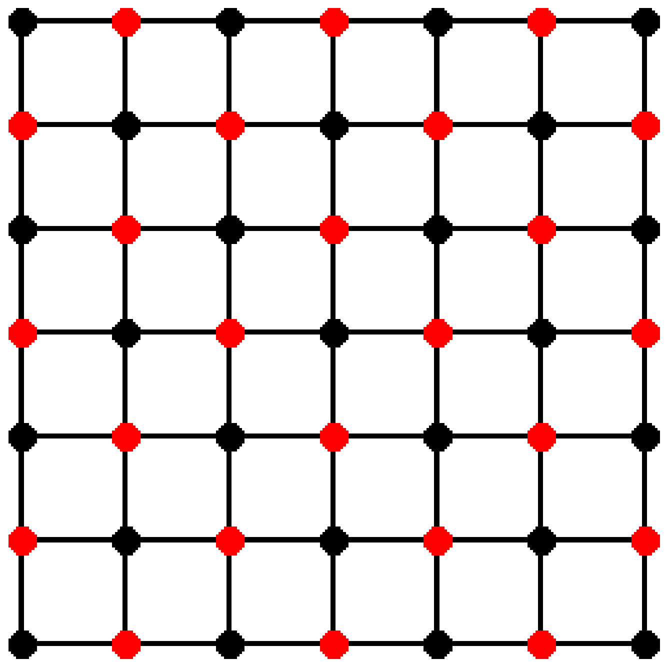
# LINEAR SOLVERS (2 OF 3)

- LU-SSOR (Lower-Upper Symmetric Successive Overrelaxation) scheme




Source: Blazek, J., Computational Fluid Dynamics: Principles and Applications. Elsevier, 2001.

- Coloring scheme (red-black)



Source: https://people.eecs.berkeley.edu/~demmel/cs267-1995/lecture24/lecture24.html

Coloring scheme is more suitable for GPU acceleration

# LINEAR SOLVERS (3 OF 3)

Georgia Tech

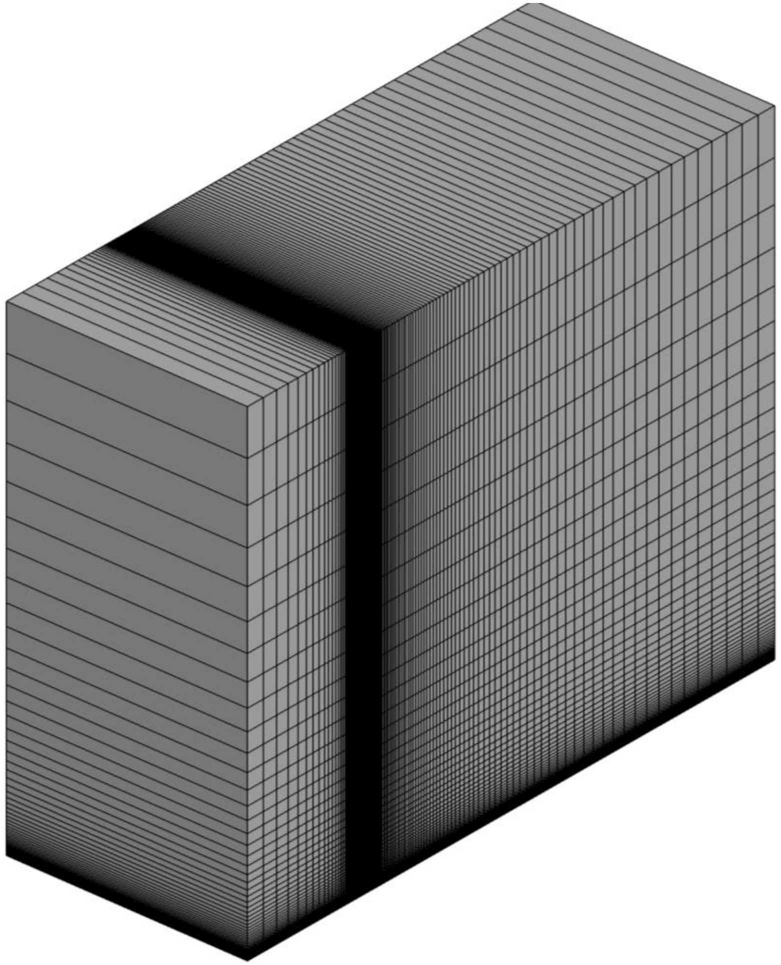
- What to consider with the red-black solver
  - Coloring scheme converges slower than LU-SSOR scheme
  - Need more linear solver iterations at each step
  - Because of the 4th order dissipation, black also depends on black!

    → potentially even slower convergence
  - Reinitializing $\Delta \boldsymbol{U}$ to zero proved to be best

Is using a GPU worth the loss of convergence in the solver?

# TEST PROBLEMS

- Laminar Flat plate
  - $Re_L = 10000$
  - $M_\infty = 0.1$
  - (2D): 161 x 2 x 65 → Initial profile
  - (3D): 161 x 31 x 65 → Hackathon
  - Other coarser/finer meshes to understand the scaling
- Define two types of speedup
  - **Speedup**: comparison to a CPU for the same algorithm
  - **"Effective" speedup**: comparison to more efficient CPU algorithm



# HACKATHON OBJECTIVES AND STRATEGY (1 OF 2)

- Port the entire application to GPU for laminar flows
- Obtain at least a 1.5 x acceleration on a single GPU compared to a CPU node, (approximately 16 cores) using OpenACC
- Extend the capability of the application using both MPI and GPU acceleration

# HACKATHON OBJECTIVES AND STRATEGY (2 OF 2)

- Data
  - !$acc data copy ()
  - Initially, data structure around all ported kernels → slowdown
  - Ultimately, only one memcopy (before entering the time loop)
- Parallel loops with collapse statement
  - !$acc parallel loop collapse(4) gang vector
  - !$acc parallel loop collapse(4) gang vector reduction
  - !$acc routine seq
  - Temporary and private variables to avoid race conditions
    - Example $rhs(i,j,k), rhs(i+1,j,k)$ updated in the same step

# RESULTS AT THE END OF THE HACKATHON

Georgia Tech

- Total run times (10 steps on a 161 x 31 x 65 grid)

| GPU | CPU (16 cores) - MPI | CPU 1 core |
| --- | --- | --- |
| 6.5 sec | 23.9 s | 89.7 s |

- Speedup
  - 13.7x versus single core
  - 3.7x versus 16 core, but this MPI test did not exhibit linear scaling
- Initial objectives not fully achieved, but encouraging results
- Postpone MPI implementation until better speedup is obtained with the serial implementation

# FURTHER IMPROVEMENTS (1 OF 2)

- Now that the code runs on GPU, what's next?
  - Can we do better?
  - What's the cost of using the coloring scheme versus the LU-SSOR scheme?
- Improve loop arrangements and data management
  - Make sure all !$acc data copy () statements have been replaced by !$acc data present () statements
  - Make sure there are no implicit data movements

# FURTHER IMPROVEMENTS (2 OF 2)

- Further study and possibly improve the speedup
- Evaluate the "effective" speedup
- Run a proper profile of the application running on GPU with pgprof

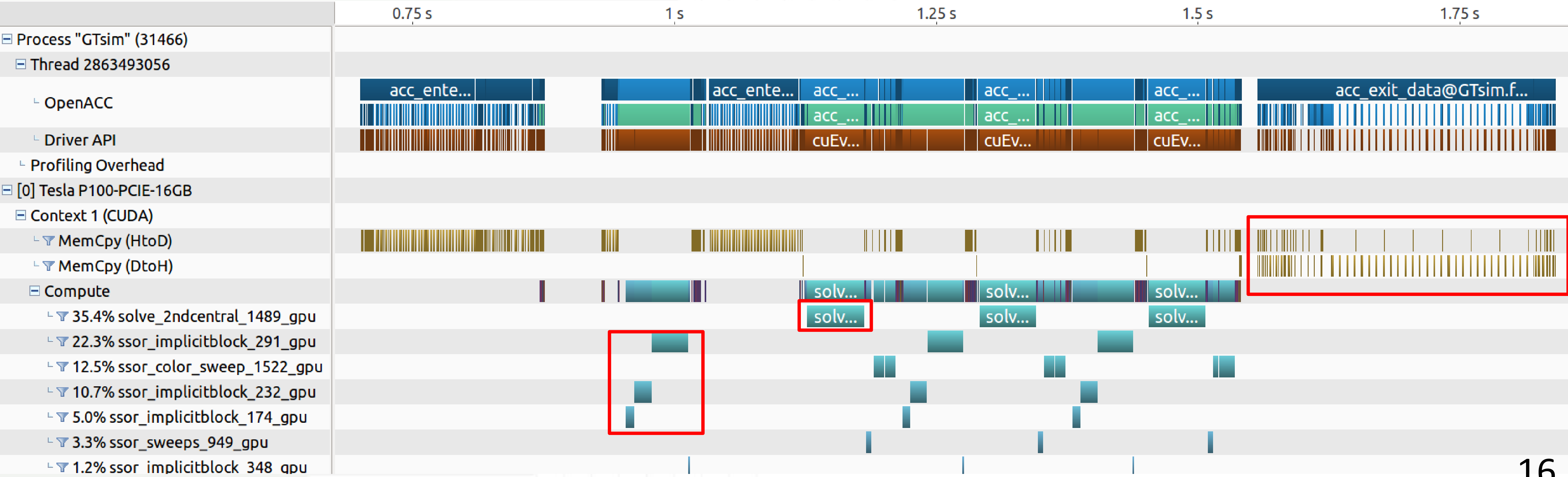
  pgprof --export-profile timeline.prof ./GTsim > GTsim.log

  pgprof --metrics achieved_occupancy,expected_ipc -o metrics.prof ./GTsim > GTsim.log

# DATA MOVEMENT

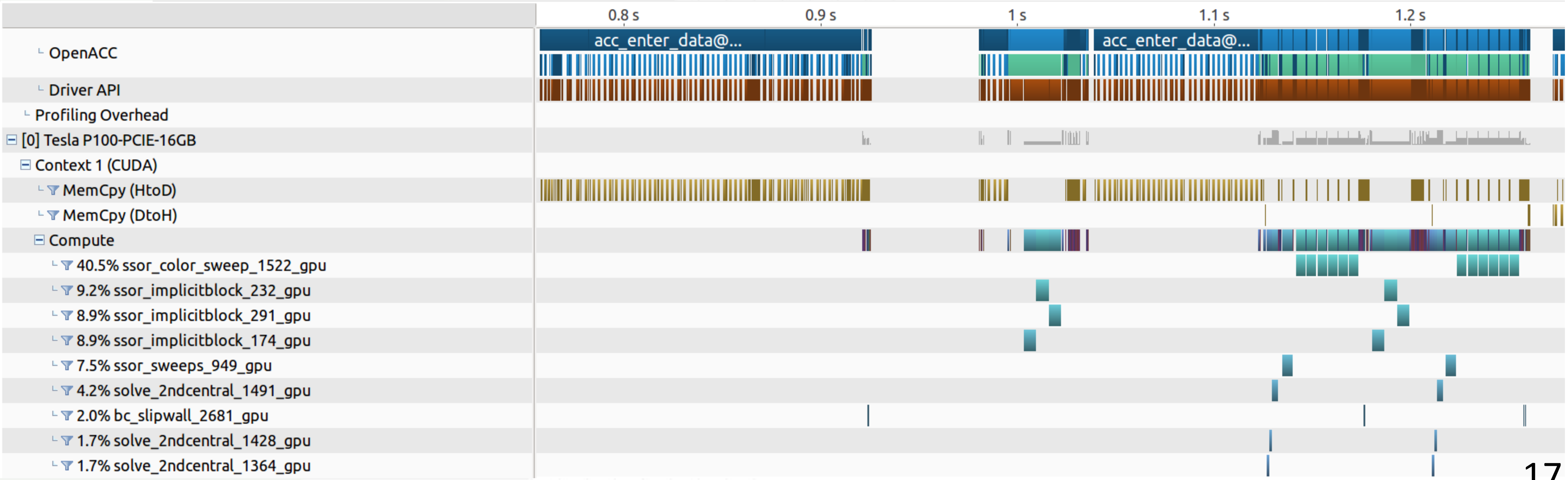


- !$acc data copy() → !$acc enter data copyin()/copyout()
- Solver blocks ($\overline{\boldsymbol{LHS}}$, $\boldsymbol{RHS}$) are not actually need back on the CPU
- Only the solution vector needs to be copied out



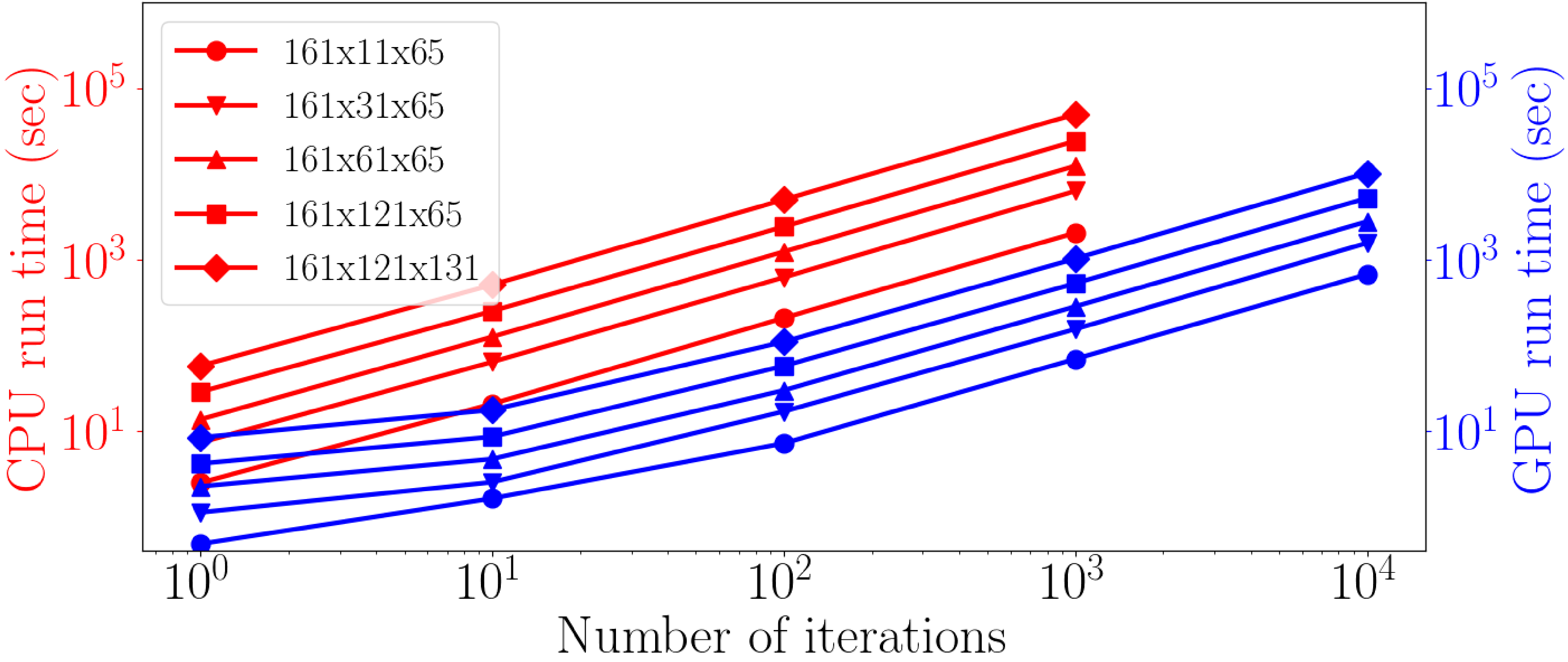
# LOOP ARRANGEMENTS

- All loop in the order k, j, I
- Limit the size of the registers to 128 → -ta=maxregcount:128
- Memory is still not accessed contiguously, especially on the red-black kernels



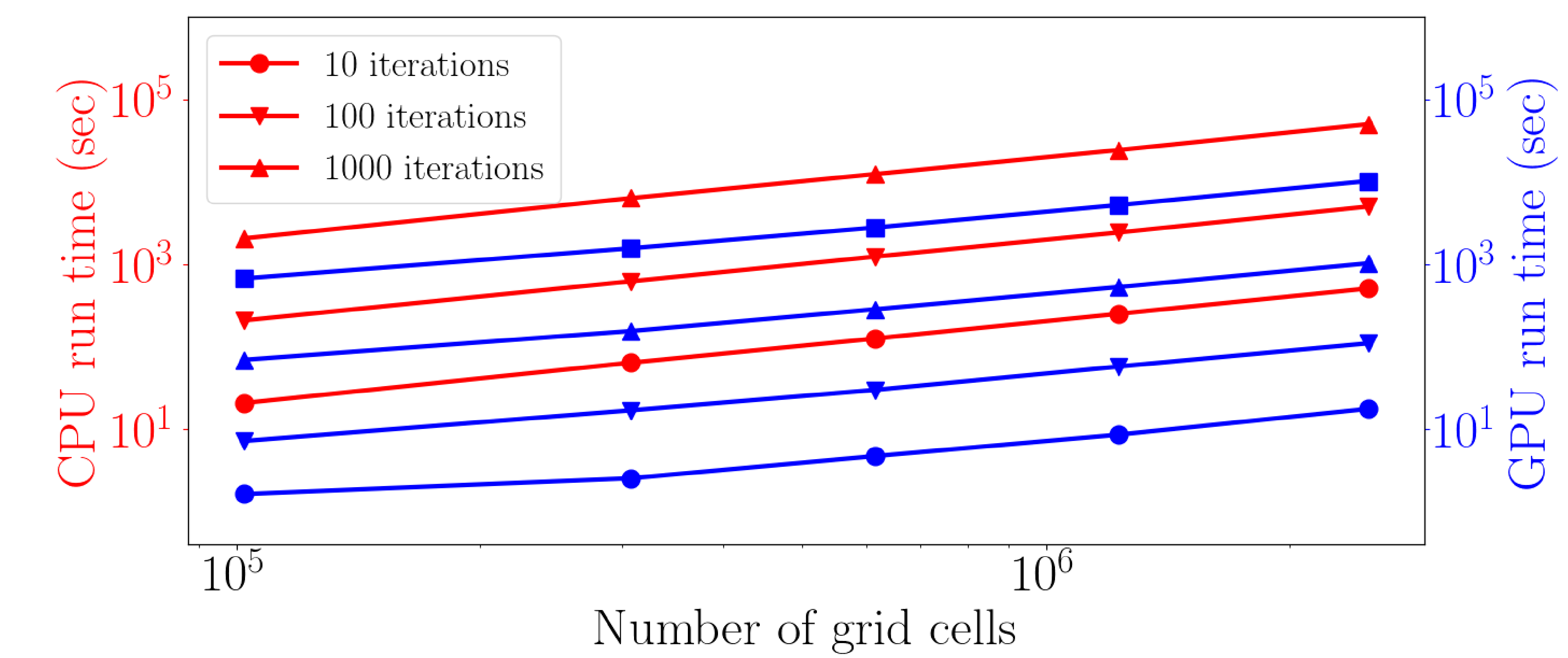
# FINAL SOLUTION TIMES

- Red-black solver with 3 sweeps, CFL 0.1
- Linear scaling with number of iterations once data movement cost is offset



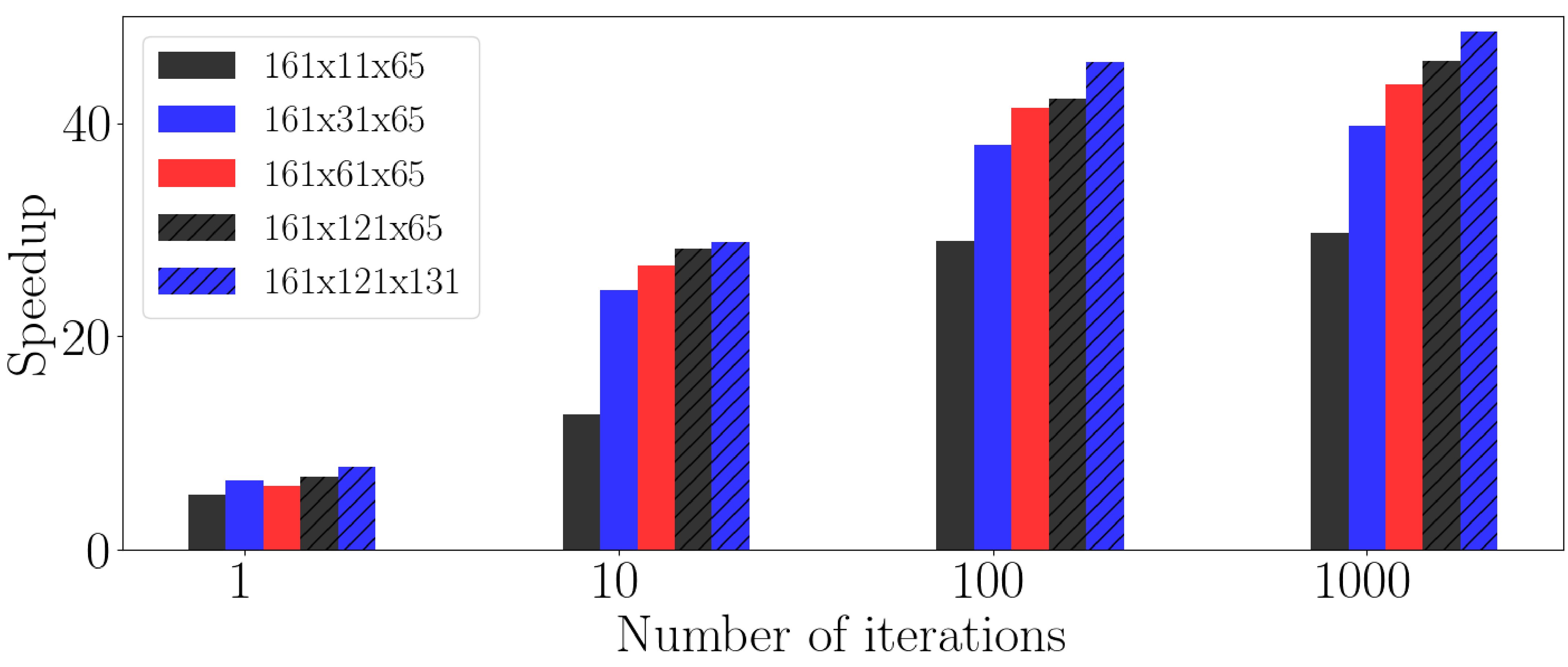
# FINAL SOLUTION TIMES

- Red-black solver with 3 sweeps, CFL 0.1
- Linear scaling with grid size once data movement cost is offset



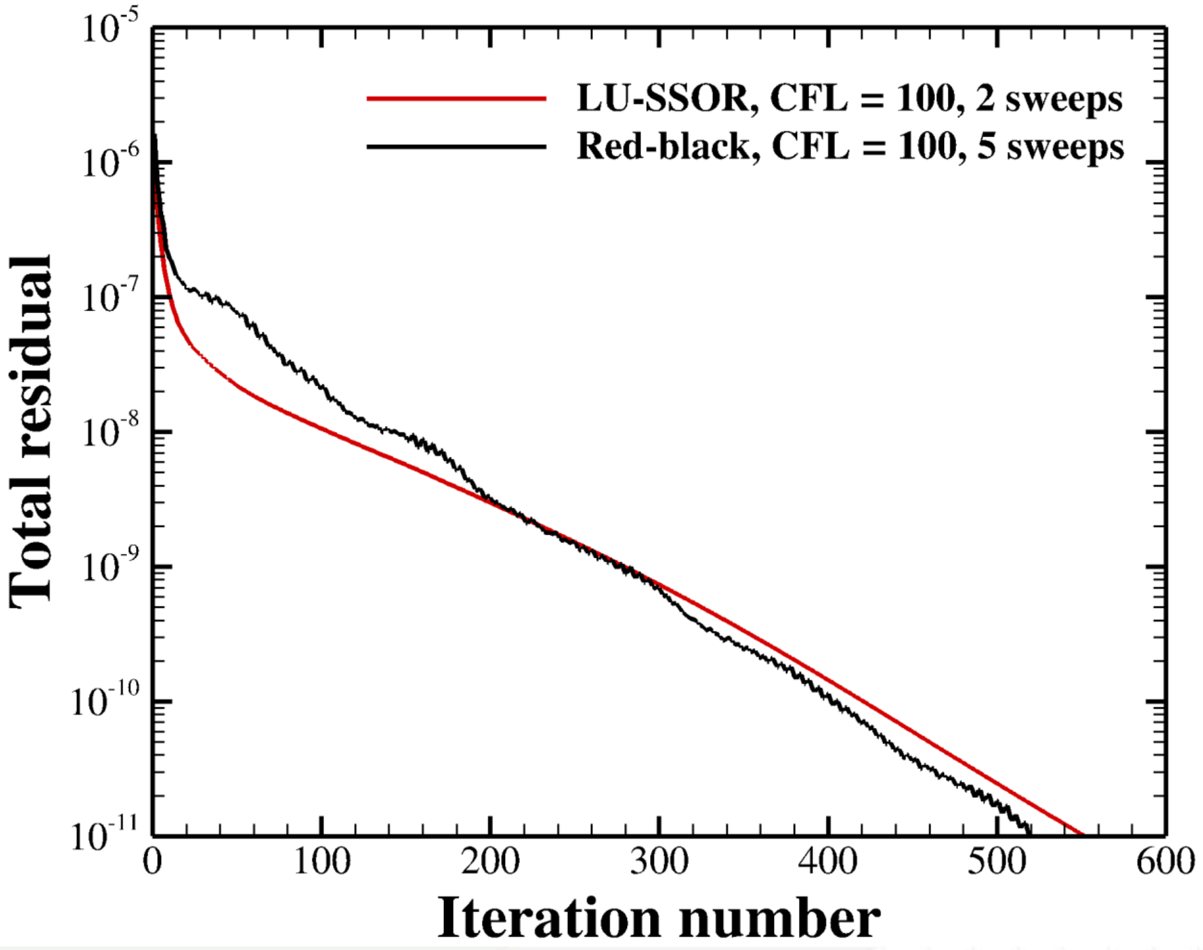
# FINAL SPEEDUP

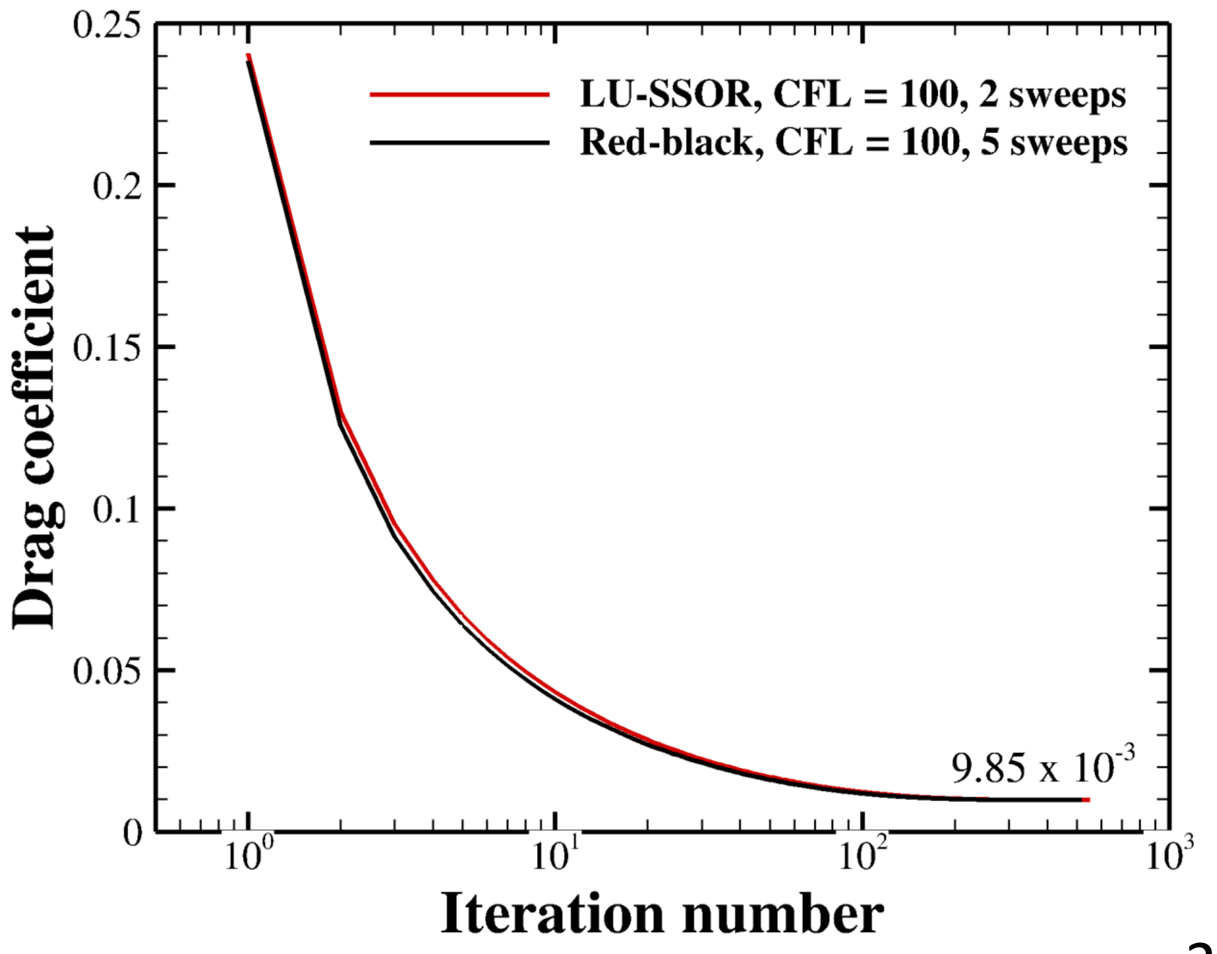
- Red-black solver with 3 sweeps, CFL 0.1
- Best speedup of 49 for a large enough grid and number of iterations



# CONVERGENCE OF THE LINEAR SOLVERS (1 OF 2)

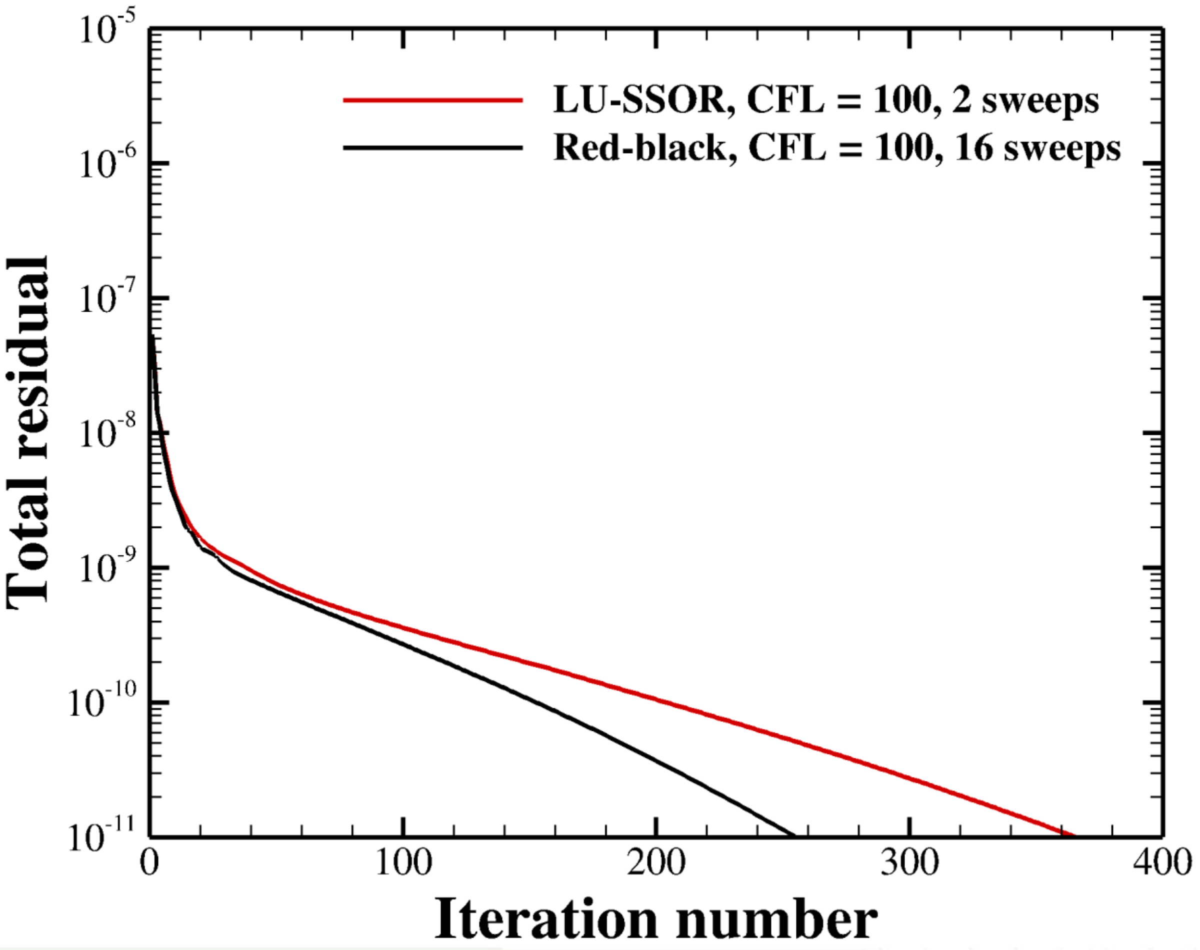
- 161 x 2 x 65 mesh, convergence to $10^{-11}$ → Same run times

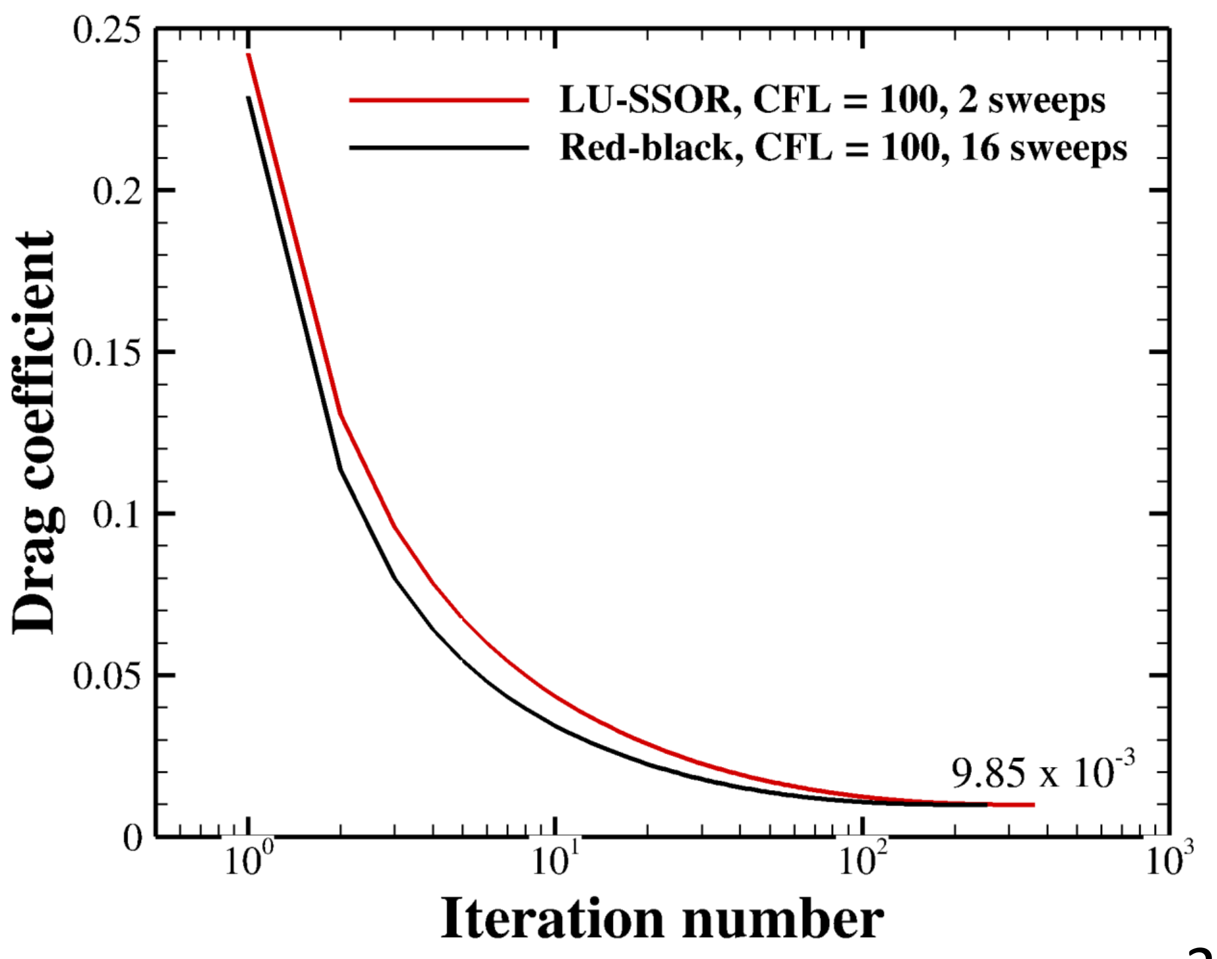


# CONVERGENCE OF THE LINEAR SOLVERS (2 OF 2)

- 161 x 31 x 65 mesh, convergence to $10^{-11}$



# EFFECTIVE SPEEDUP

- 161 x 31 x 65 mesh, convergence to $10^{-11}$

| GPU - Red-black solver | CPU - Red-black solver | CPU – SSOR solver |
| --- | --- | --- |
| 109.3 sec | 4329.6 sec | 3140.0 sec |

- Speedup of 39 compared to the same solver on CPU
- Speedup of 29 compared to the SSOR scheme on CPU

The effective speedup is the same as speedup in 2D, and lower but still good in 3D!

# CONCLUSIONS AND FUTURE WORK

- Conclusions
  - A CFD solver has been ported to GPU using OpenACC
  - Speedup on the order of 50 X compared to a single CPU core
  - Red-black solver replaced the LU-SSOR solver with little to no loss of performance
- Future work
  - Further optimization of data transfers and loops
  - Extension to MPI

# ACKNOWLEDGEMENTS


- Oak Ridge National Lab
  - Organizing and letting us participate in the 2017 GPU Hackathon
  - Providing access to Power 8 and P100 GPUs on SummitDev
- NVIDIA
  - Providing access to P100 GPUs on the psg cluster
- Everyone else who helped with this work

# CLOSING REMARKS

- Contact
  - Nicholson K. Koukpaizan
  - nicholsonkonrad.koukpaizan@gatech.edu

- Please, remember to give feedback on this session

- Question?

# BACKUP SLIDES

# GOVERNING EQUATIONS

- Navier-Stokes equations

$$\frac{\partial}{\partial t}\int_{\Omega} \boldsymbol{U} dV + \oint_{\partial\Omega} (\boldsymbol{F}_c - \boldsymbol{F}_V) dS = 0$$

$$\boldsymbol{U} = [\rho \quad \rho u \quad \rho v \quad \rho w \quad \rho E]^T$$

  - $\boldsymbol{F}_C$ , inviscid flux vector, including mesh motion if needed (Arbitrary Lagrangian-Euler formulation)
  - $\boldsymbol{F}_V$ , viscous flux vector
- Loosely coupled turbulence model equations added as needed
  - Laminar flows only in this work
  - Addition of turbulence does not change the GPU performance of the application

# DISCRETIZED EQUATIONS

- Explicit treatment of fluxes
  - 2nd order central differences with 4th order Jameson dissipation
- Implicit treatment of fluxes
  - Steger and Warming flux splitting
- Dual time stepping, with 2nd order backward difference formulation
- Form of the final equation to solve

$$\left[\frac{\Omega_{ijk}}{\Delta t}\left(\frac{\Delta t}{\Delta \tau}+\frac{3}{2}\right)\bar{\boldsymbol{I}}+\left(\frac{\partial \boldsymbol{R}}{\partial \boldsymbol{U}}\right)^{m}\right]\Delta \boldsymbol{U}^{\boldsymbol{m}}=-\boldsymbol{R}^{m}-\left(\frac{\Omega_{ijk}}{\Delta t}\right)\frac{3\boldsymbol{U}^{m}-4\boldsymbol{U}^{n}+\boldsymbol{U}^{n-1}}{2}$$

- Need a linear solver!