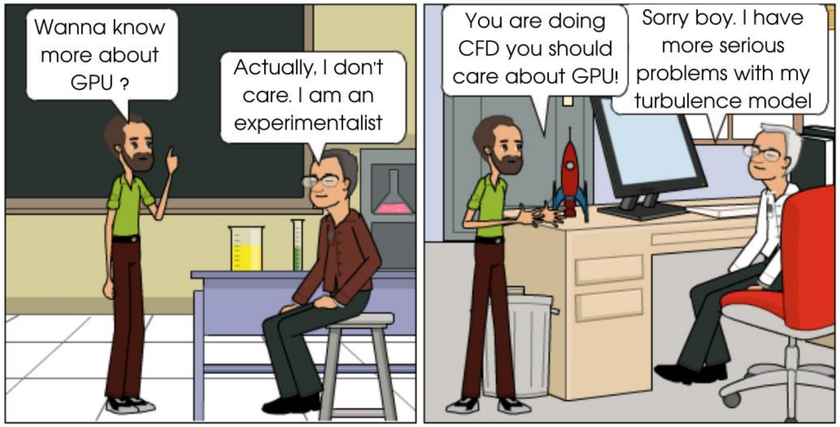
GPU-accelerated CFD Simulations for Turbomachinery Design Optimization

Mohamed H. Aissa

www.researchgate.net/profile/Mohamed_Aissa3

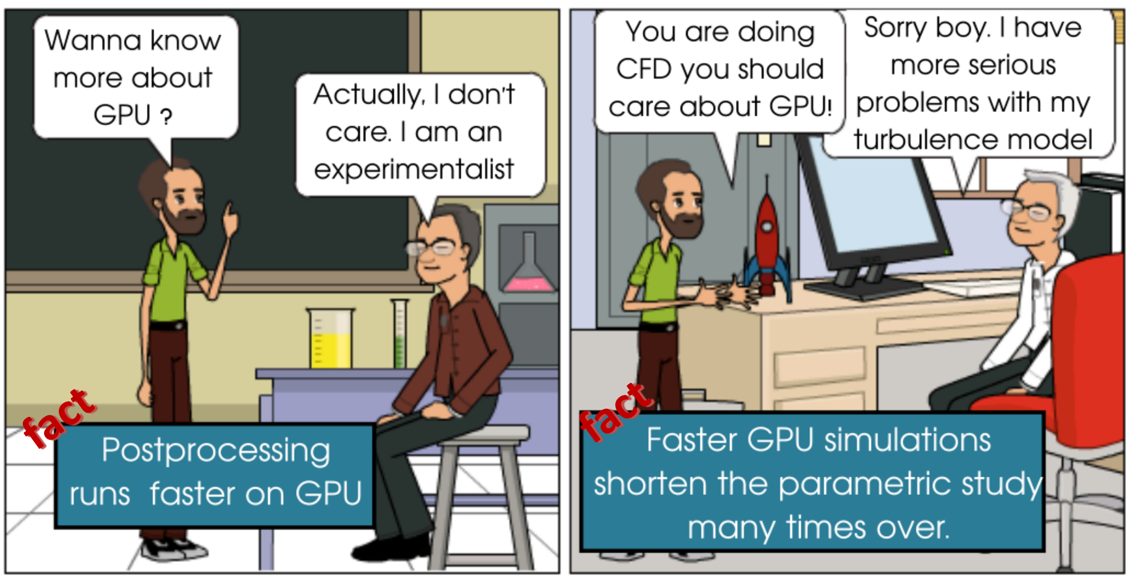
Co-promotor:Dr. Tom Verstraete

Promotor: Prof. C. Vuik



Can your simulation profit from the GPU?

- What is a GPU?
- How fast it is?
- How to use it?



Multi-core vs many-core

Control Unit

Processor Processor Processor Processor

Memory

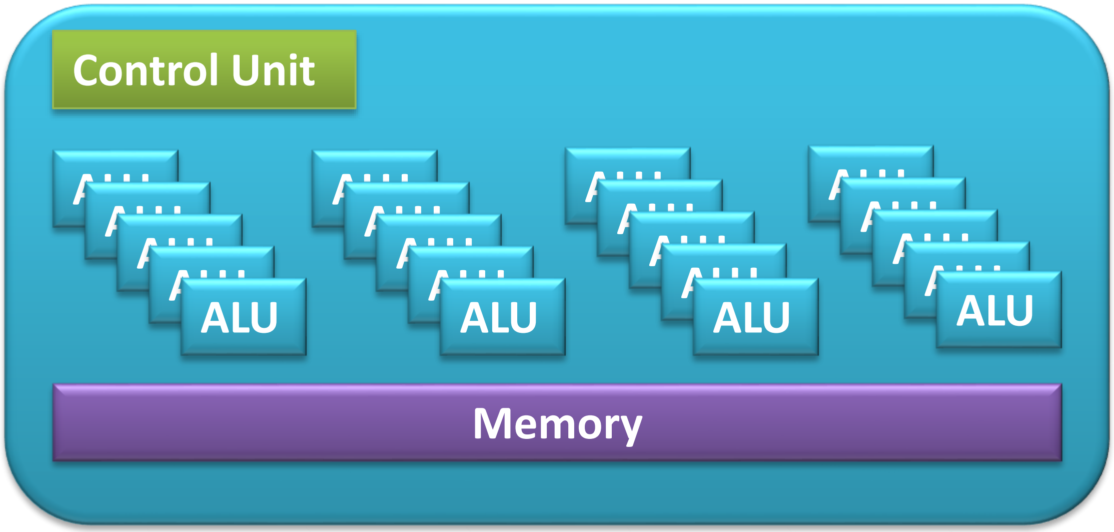
Communication Netwok

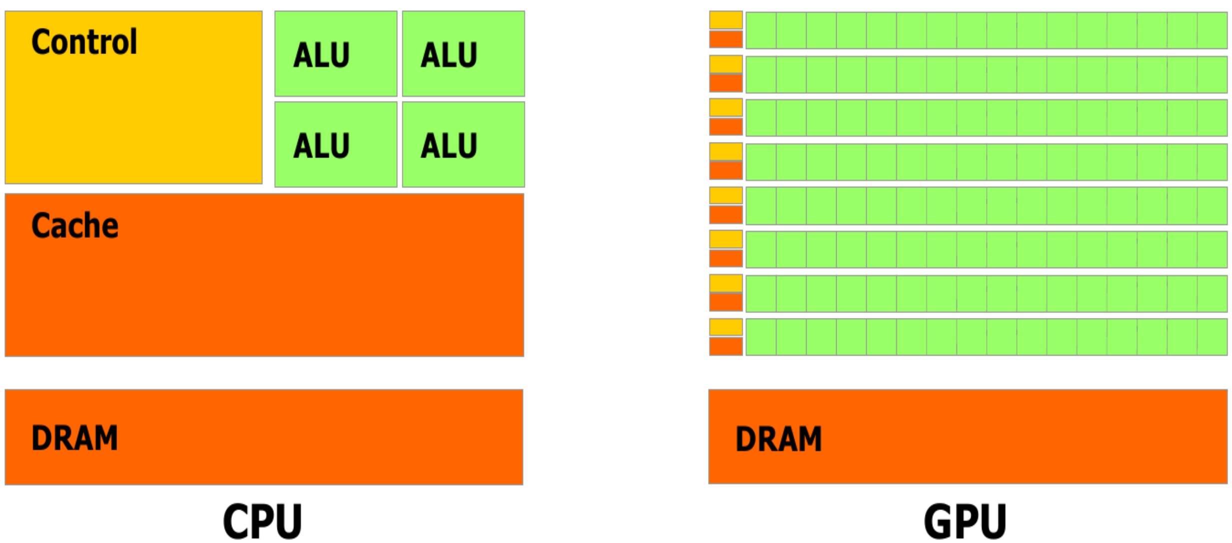
Control Unit | Control | ol Unit | Control Unit

Processor | Proce | essor | Processor

Memory | Mem | mory | Memory

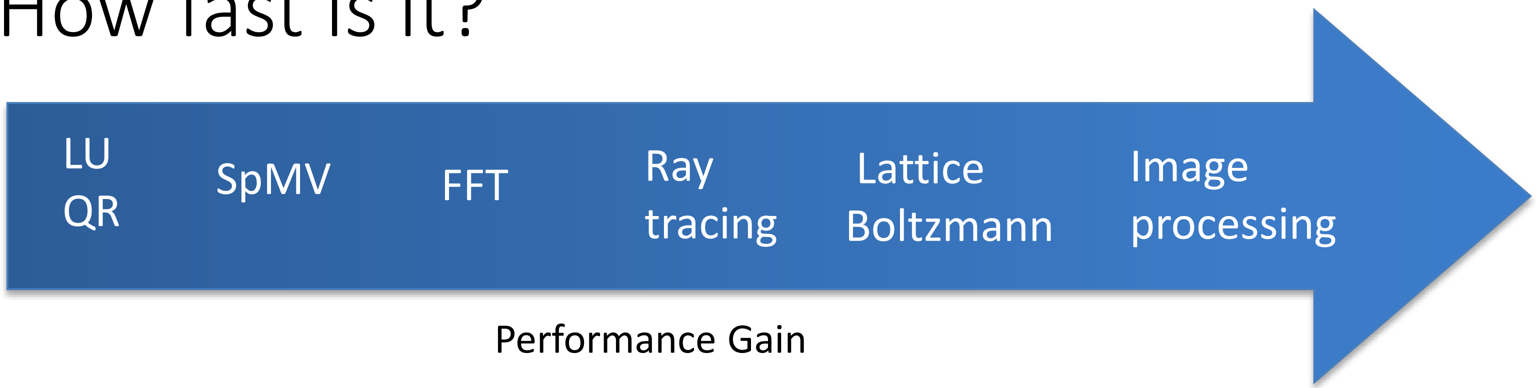
Massive Parallel Systems (e.g. GPU) as a trade-off



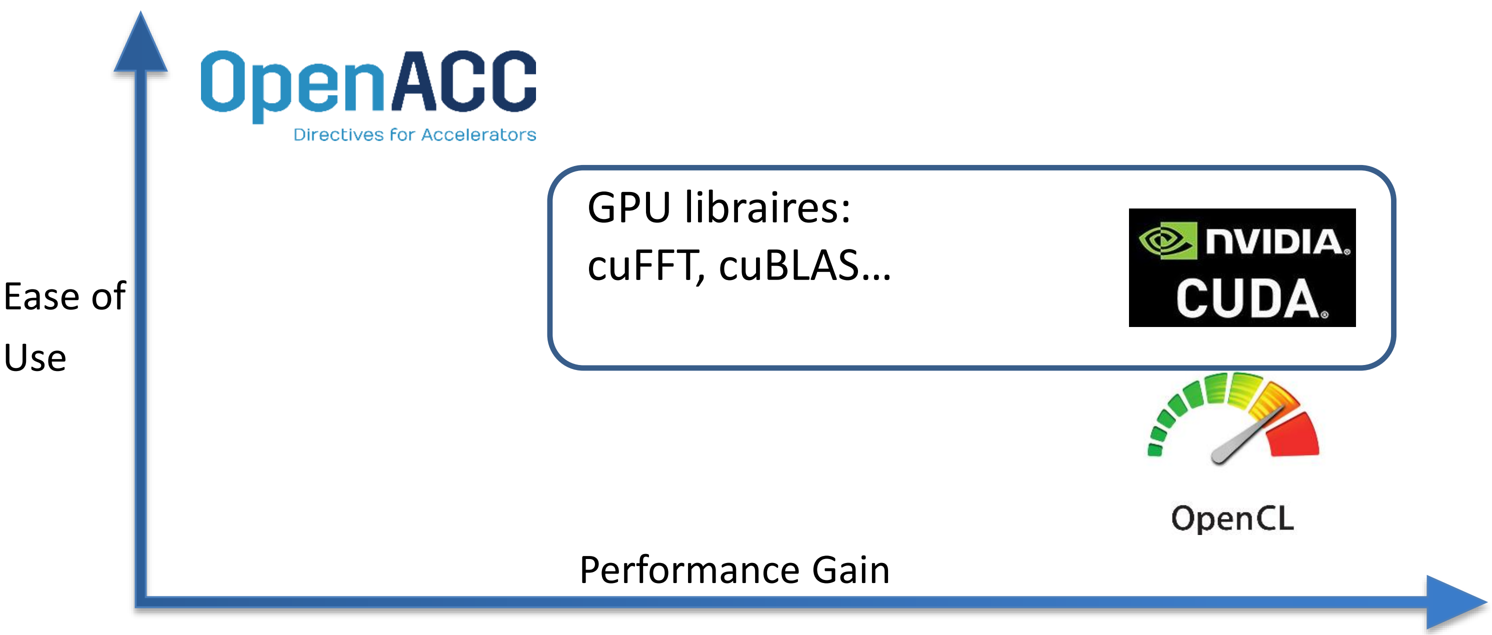


Source: the guardian.com

How fast is it?



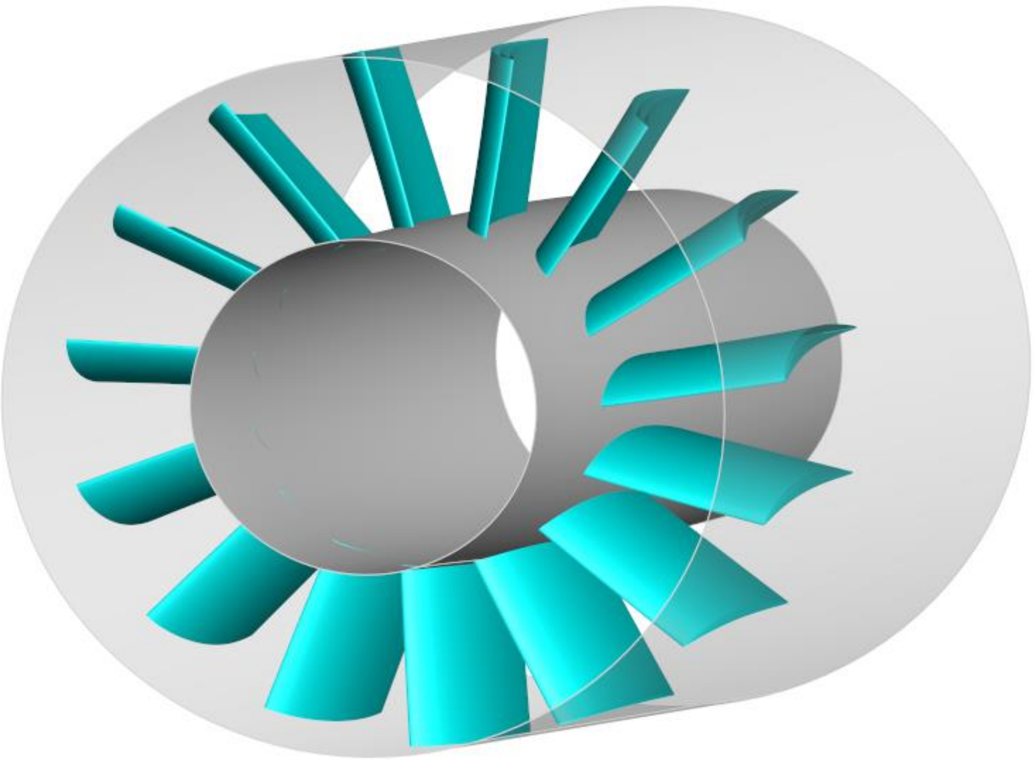
How to use a GPU



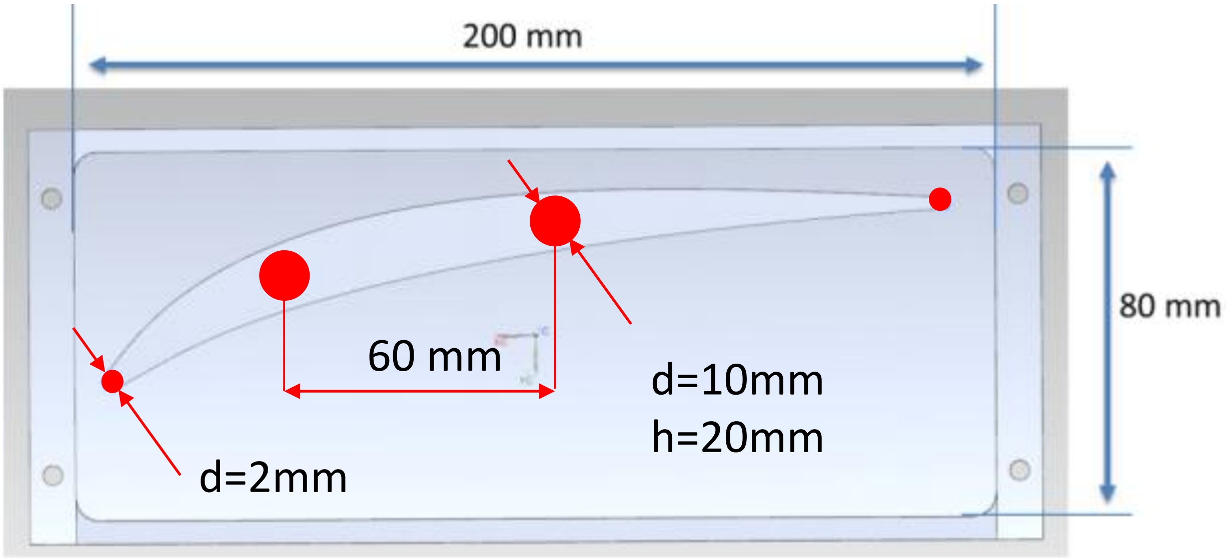
Airplanes are getting more efficient

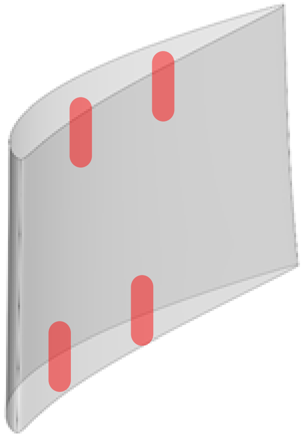
Engine optimization is a main contributor

TurboLab Stator (1/4)



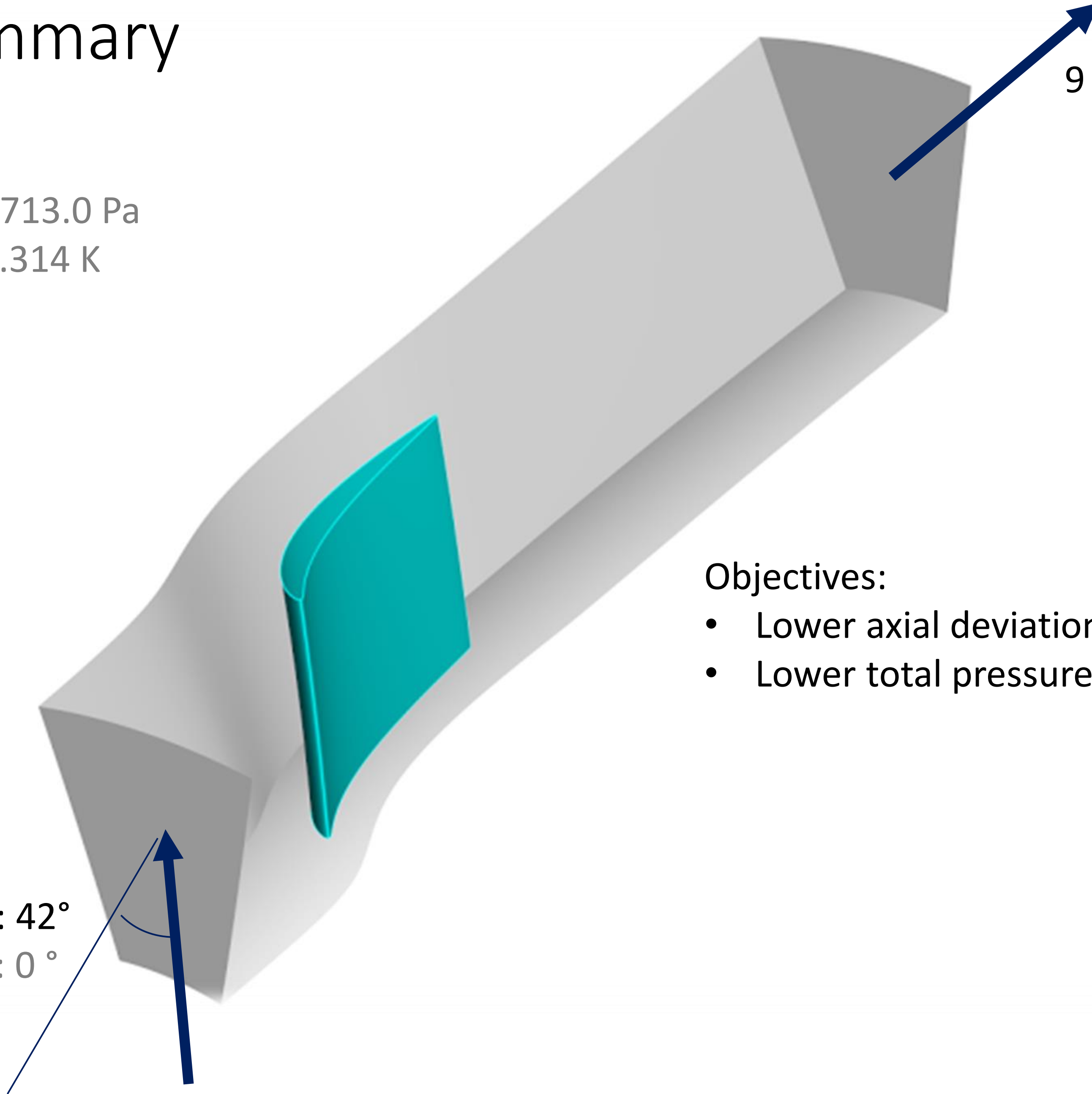
- N_{blades}= 15
- Chord length fixed
- Casing fixture





TurboLab (2/4): Boundary conditions and summary

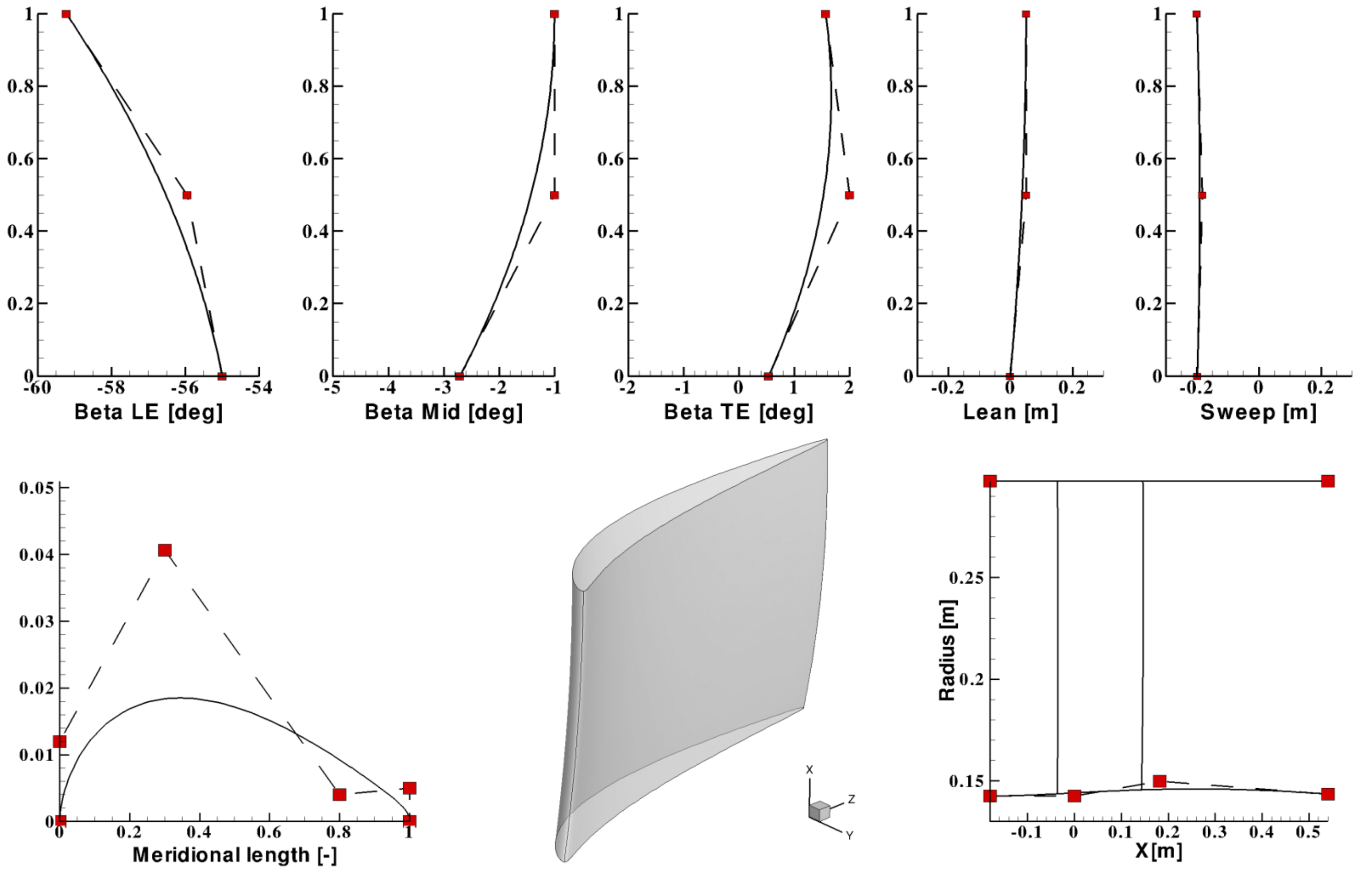
Inlet P_0: 102713.0 Pa
Inlet T_0: 294.314 K



Objectives:
- Lower axial deviation
- Lower total pressure loss

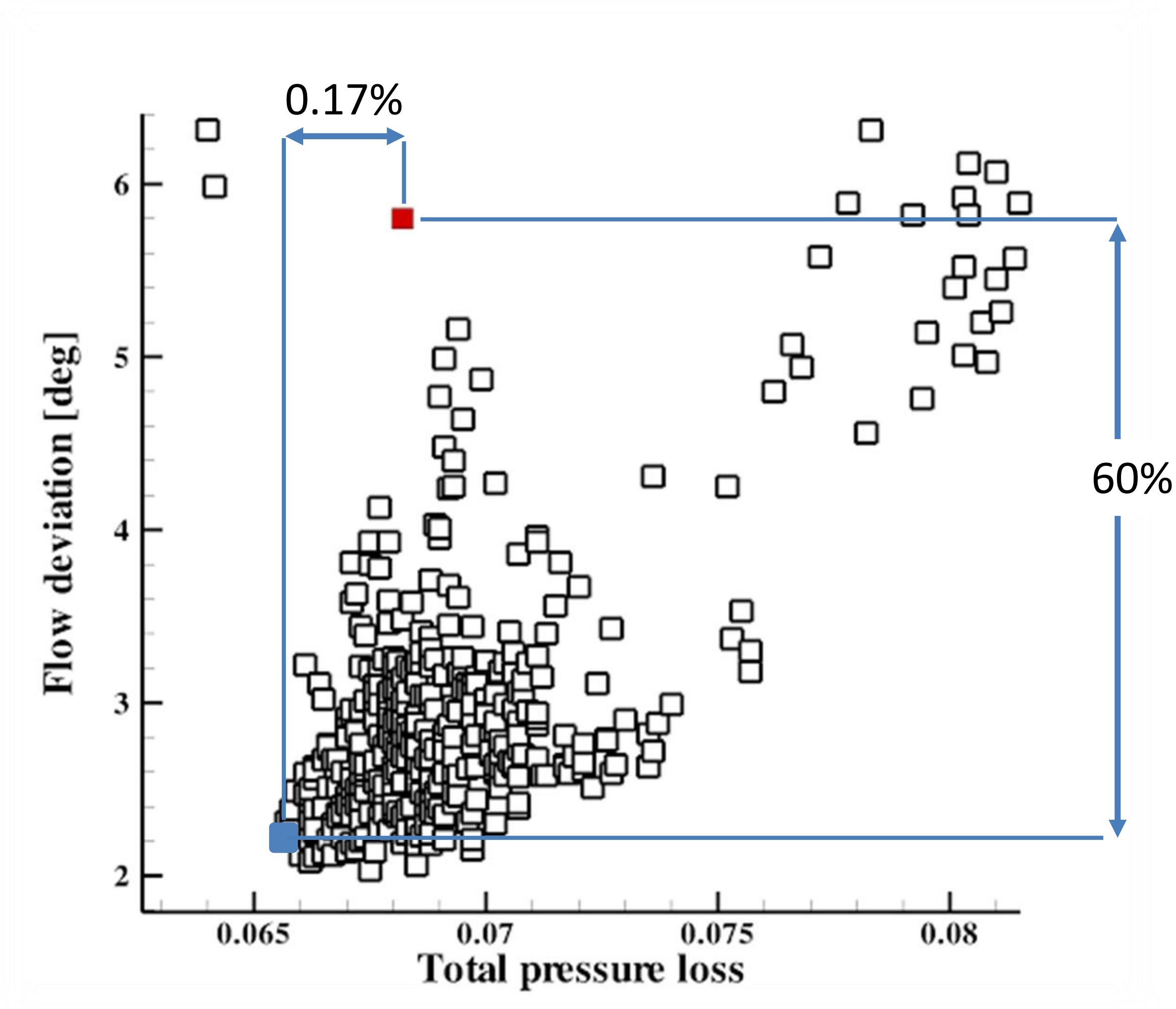
Inlet whirl angle: 42°
Inlet pitch angle: 0 °

TurboLab (3/4): Parametrization
21 Design variables



TurboLab (4/4):Optimization Results

$$\int_{hub}^{casing} \alpha_{whirl}^2$$



$$Loss_{P_0} = \frac{p_{01} - p_{02}}{p_{01} - p_1}$$

Every point is a costly CFD optimization → need for a HPC solution

How beneficial are GPUs, a quick literature check:

- Acceleration is case-dependent (from 1x to 1000x).

- Speedups are sometimes contradicting.

- Some publications are very critical to GPUs for scientific computations:
  - Lee et al “Debunking the 100x GPU vs. CPU myth”
  - Vuduc et al. “On the limits of GPU acceleration”

Main objective:
A more tangible GPU potential

CFD GPU solvers

Classification
of CFD operations

Proof-of-concept:
Optimization cases

Summary
and Conclusions

* All icon in this document from Flaticon.com

Main objective:
A more tangible GPU potential

CFD GPU solvers

Classification
of CFD operations

Proof-of-concept:
Optimization cases

Summary
and Conclusions

Numerical Scheme:

$$\frac{\partial}{\partial t}\int_{\Omega} \boldsymbol{W}\, d\Omega + \oint_{\partial\Omega} (\boldsymbol{F_c} - \boldsymbol{F_v})\, dS = \int_{\Omega} \boldsymbol{Q}\, d\Omega \qquad \boldsymbol{W} = \{\rho, \rho V_x, \rho V_y, \rho V_z, \rho E\}$$

$$\frac{\Omega}{\Delta t}\Delta \vec{W}^{n} \begin{cases} \text{Explicit Time Stepping} \\ = -\ \vec{R}^{n} \\ \text{Implicit Time Stepping} \\ = -\ \vec{R}^{(n+1)} \end{cases}$$

$$\Delta \boldsymbol{W}^{n} = -\frac{\Delta t}{\Omega}\boldsymbol{R}^{n}$$

$$\left[\frac{(\Omega I)}{\Delta t} + \left(\frac{\delta \boldsymbol{R}}{\delta \boldsymbol{W}}\right)\right]\Delta \boldsymbol{W}^{n} = -\boldsymbol{R}^{n}$$

Main objective:
A more tangible GPU potential

CFD GPU solvers

- Explicit time integration

Classification of CFD operations

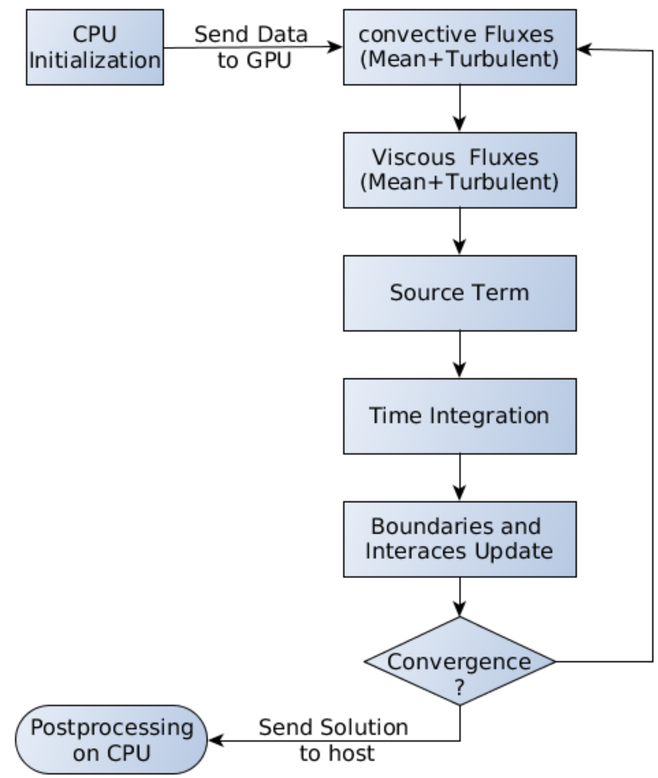
Proof-of-concept: Optimization cases

Summary and Conclusions

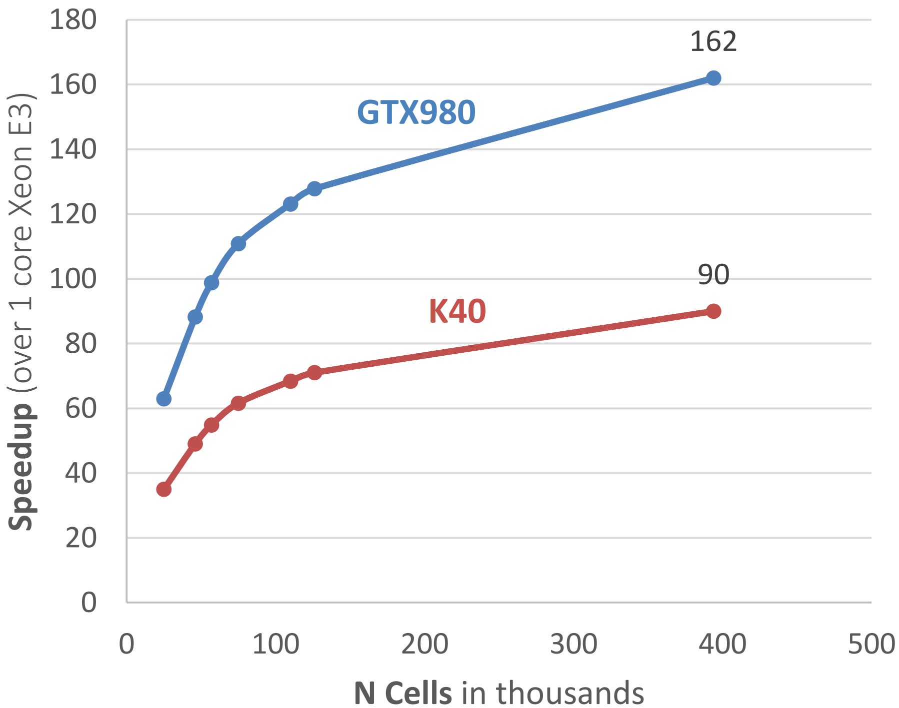
Explicit solver

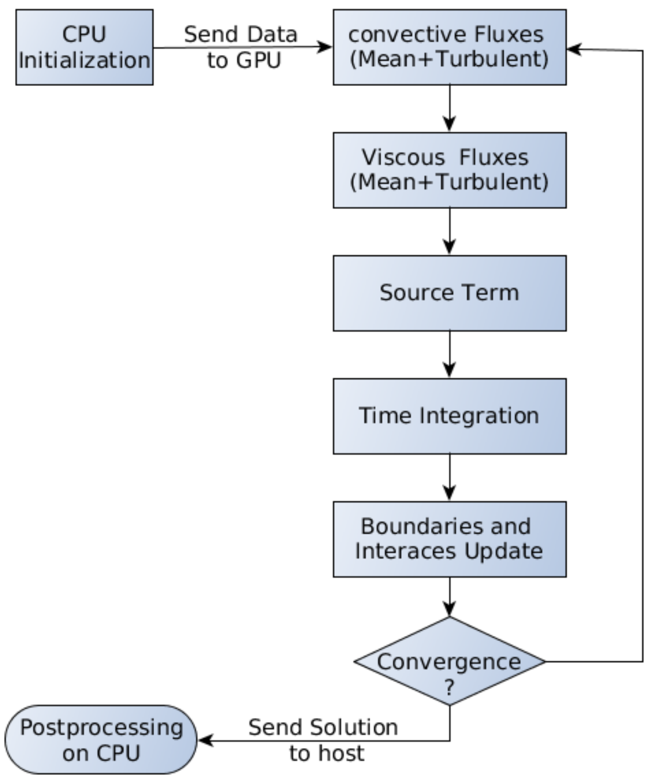
- Application:
  Steady RANS simulation
- Solved Equations:
  RANS (SA Model)
- Discretization (2nd Order):
  Roe Scheme + Flux Limiter
  Explicit RK 4 Stage
- Mesh: Multi-Block, Structured

- Acceleration:
  - 2 level Multigrid
  - Implicit Residual Smoothing



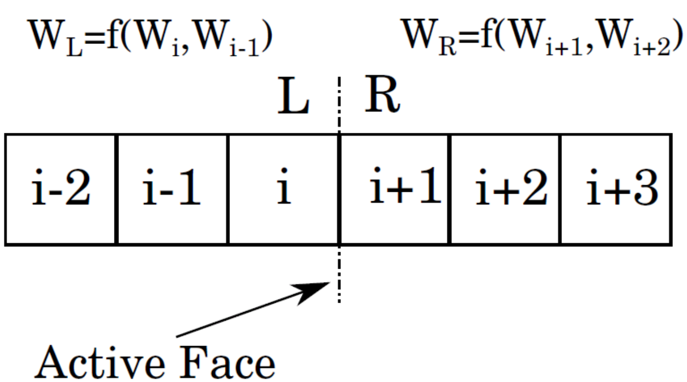
Explicit solver





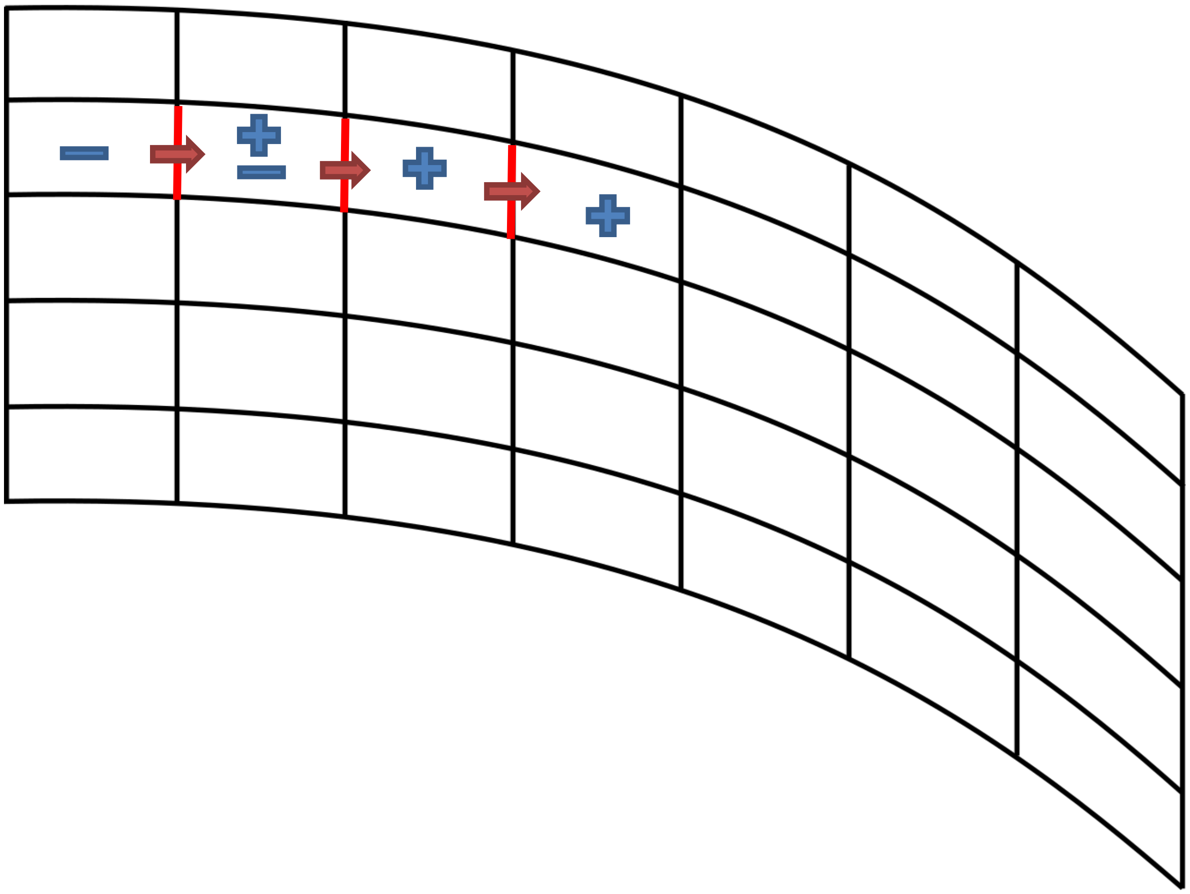
Convective Flux Evaluation (1/3)

$$(\vec{F}_c)_{I+1/2} = \frac{1}{2}[\vec{F}_c(\vec{W}_R) + \vec{F}_c(\vec{W}_L) - |\bar{A}_{Roe}|_{I+1/2}(\vec{W}_R - \vec{W}_L)]$$



Convective Flux (2/3): Thread mapping possibilities

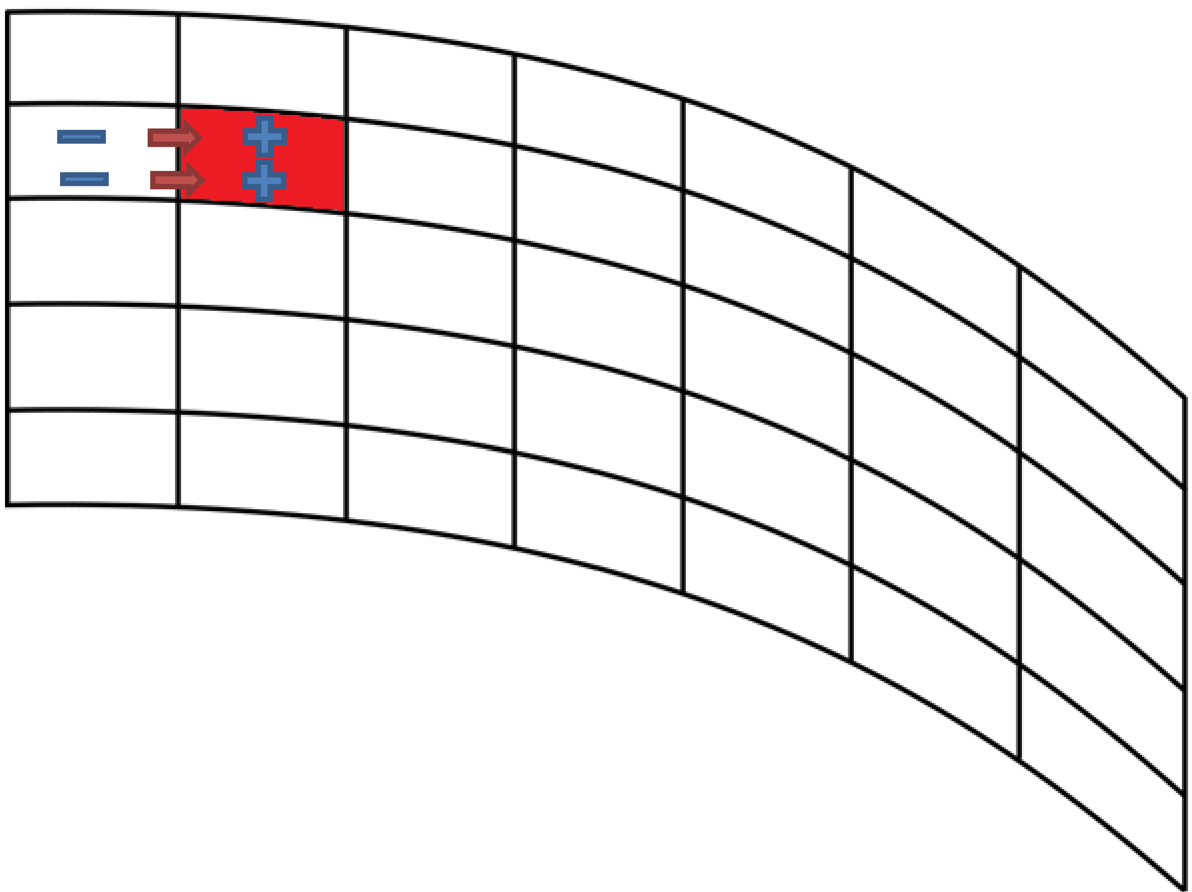
- Face-wise is not thread-safe

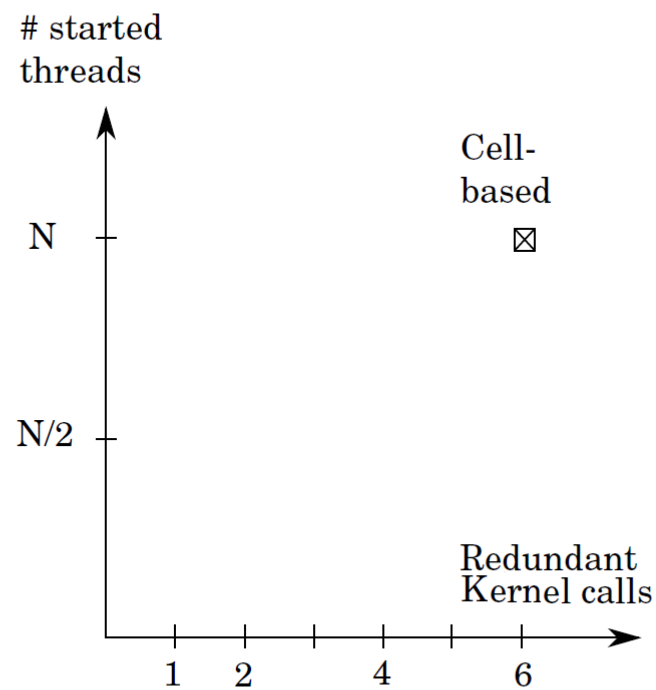


Convective Flux (2/3): Thread mapping possibilities

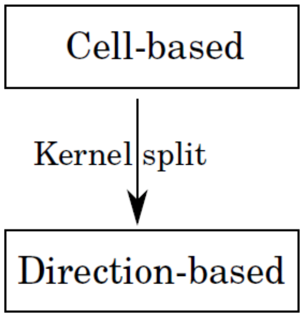
Cell-based

- Cell-based mapping thread safe but with redundancy

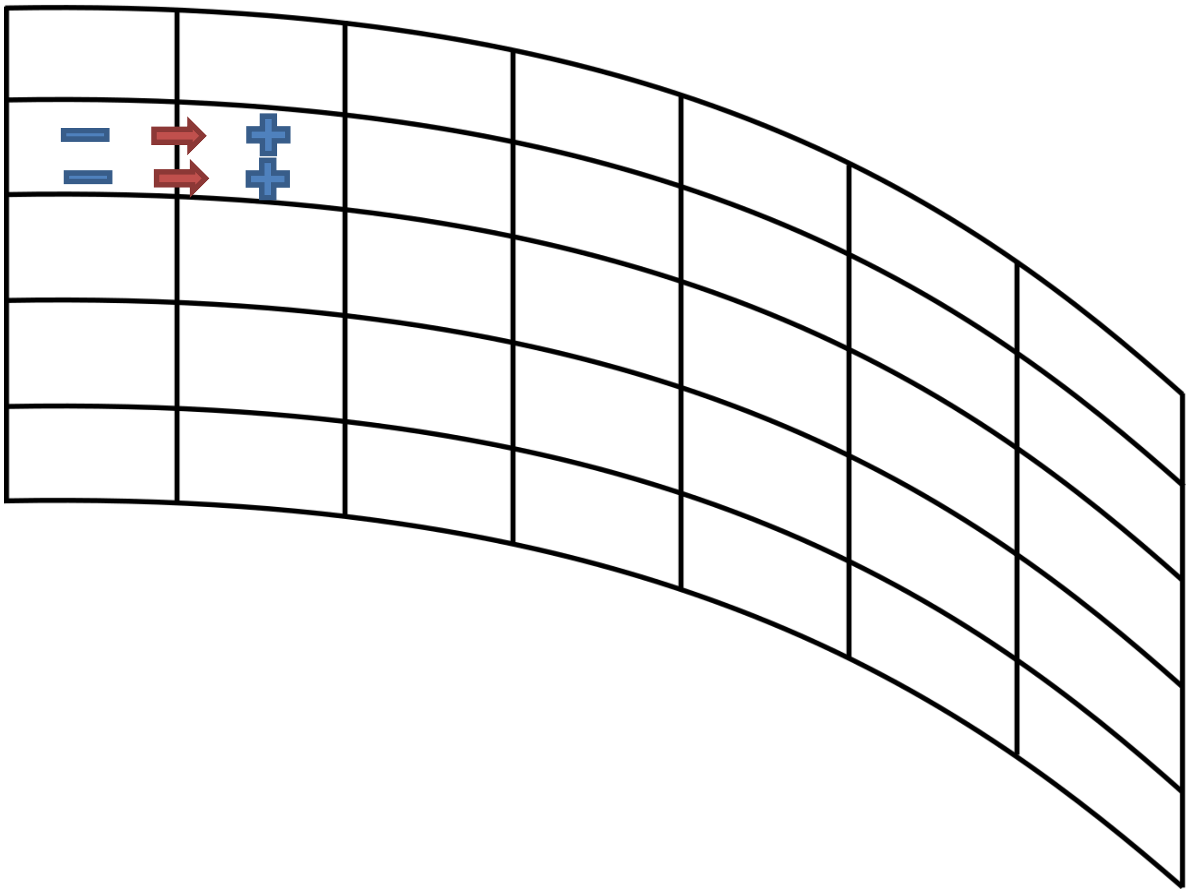


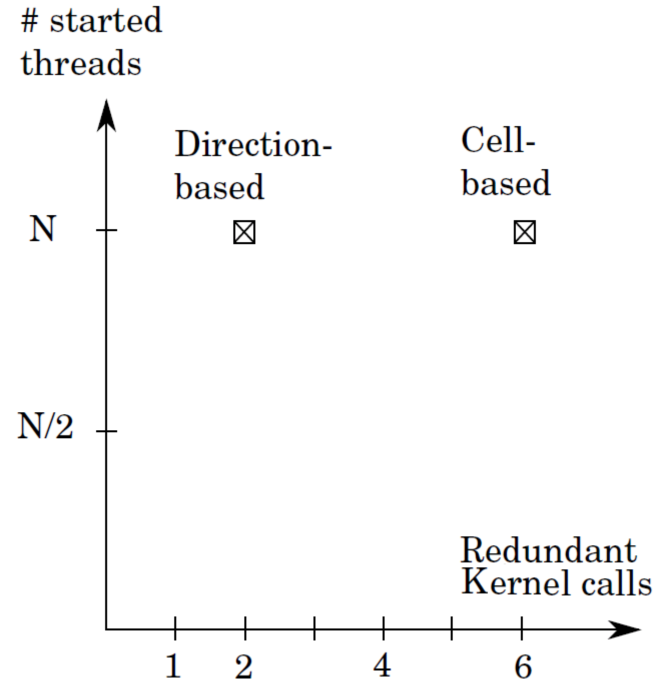


Convective Flux (2/3): Thread mapping possibilities



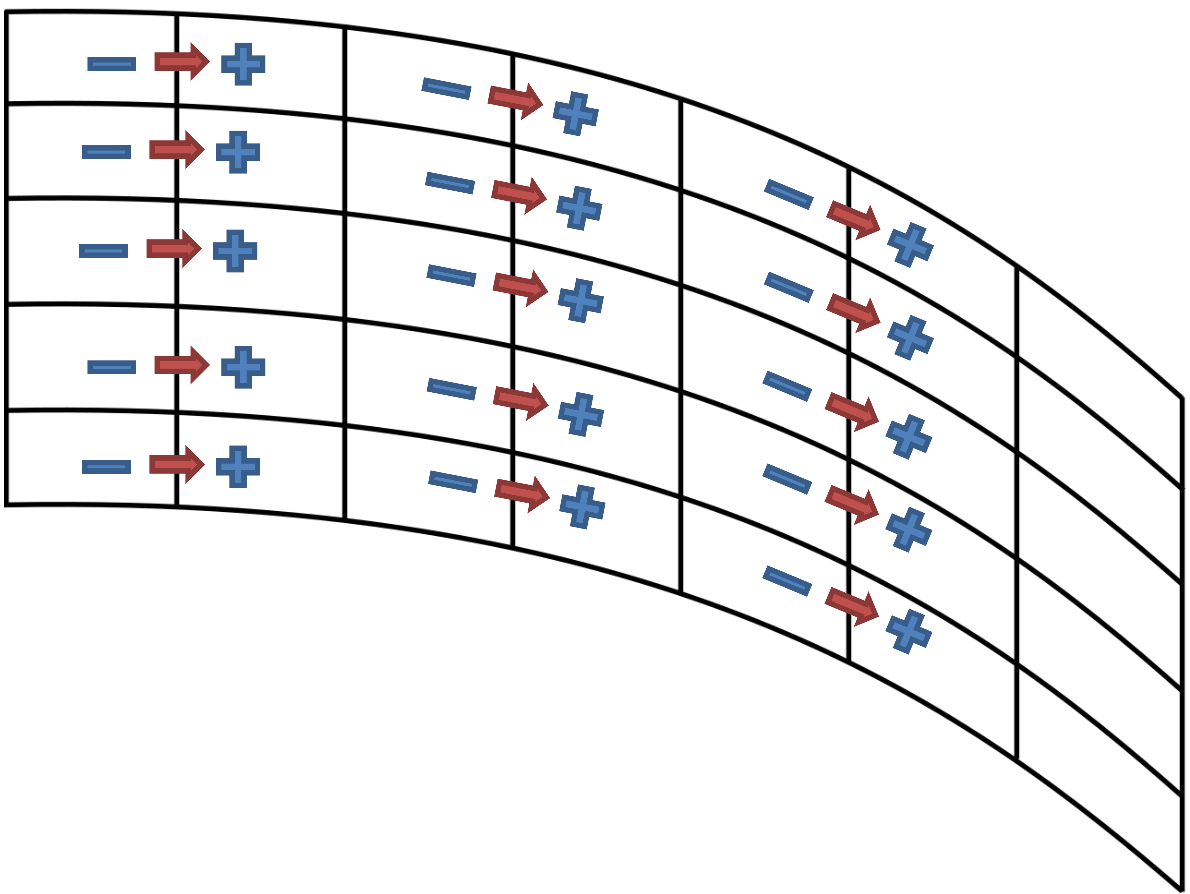
- Direction-based mapping thread safe and less redundancy

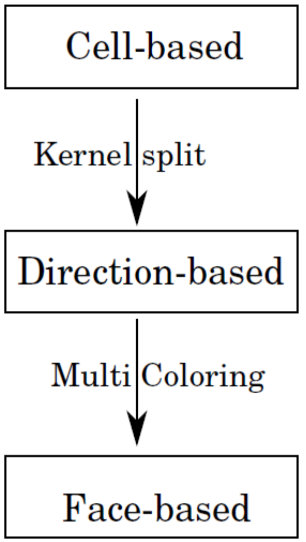


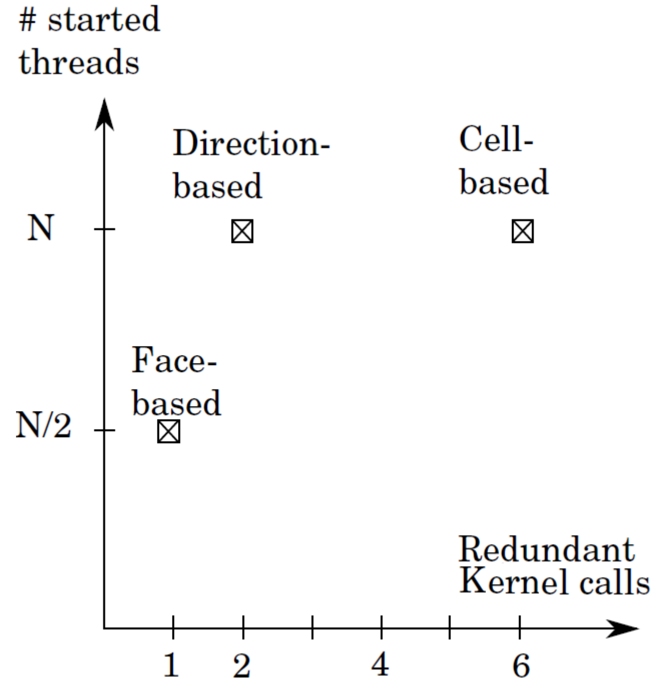


Convective Flux (2/3): Thread mapping possibilities

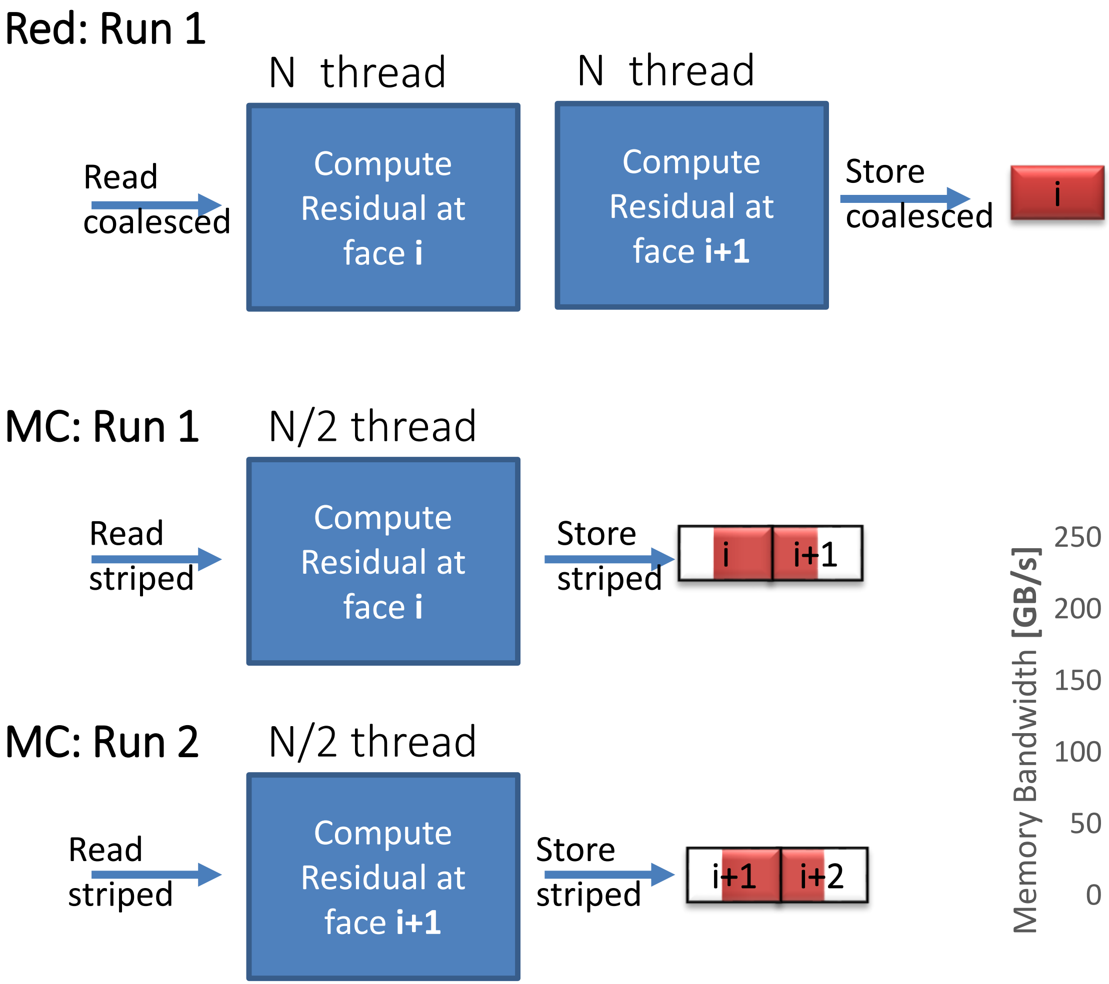
- Multicoloring (MC) Face-based mapping thread safe and No redundancy

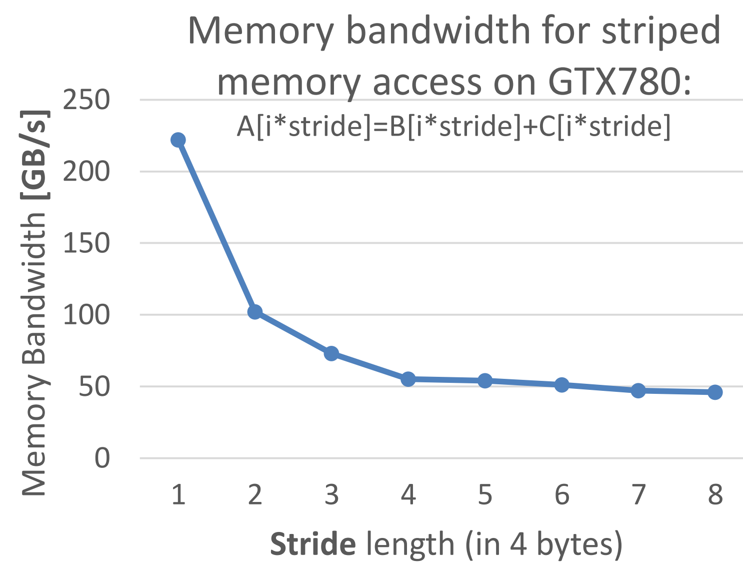






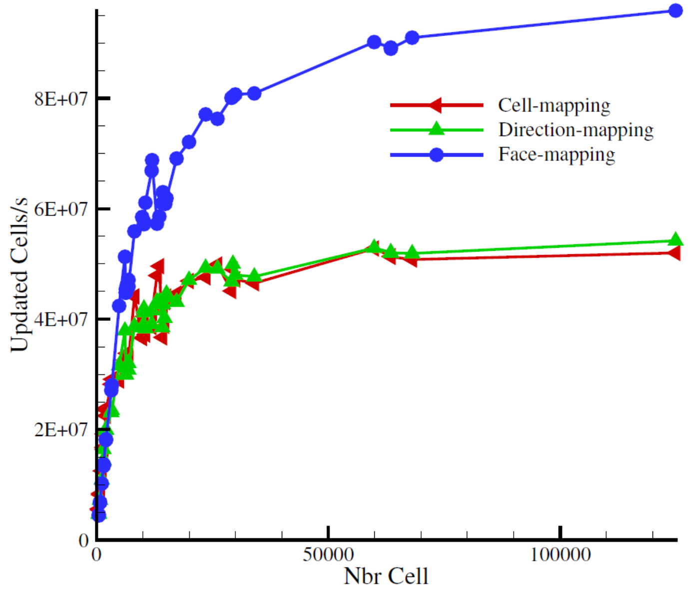
Convective Flux (3/3): Multicolred (MC) vs redundant (Red)





Convective Flux (3/3): Multicolred (MC) vs redundant (Red)

	MC	RED
Face fluxes per call	N/2	2N
Total faces fluxes	N	2N
Time per call [ms]	0,28	0,71
Total time [ms]	0,56	0,71
Operations ratio	-	2x
Total Speedup	1,26x	-



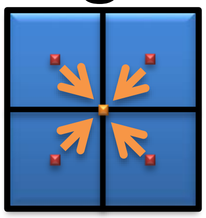
1,26x instead of 2x:
cost of striped access

Convergence Acceleration on GPU (1/3)

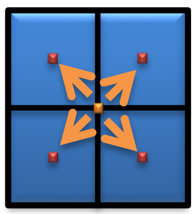
- Explicit solver is well adapted to the GPU architecture
- Flow convergence is slow (CFL limitation)
- Need for convergence acceleration.

- convergence acceleration methods on the GPU?
  - Multigrid
  - Implicit residual smoothing

Convergence Acceleration on GPU (2/3): Multigrid is also fast on the GPU

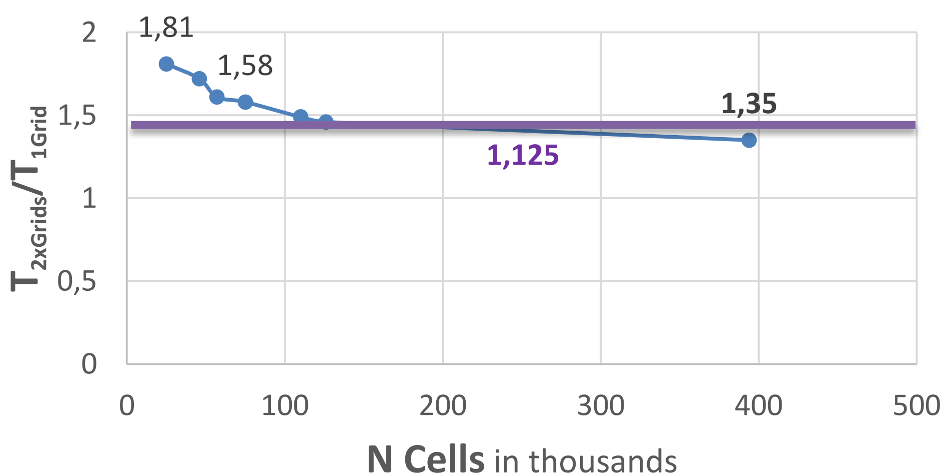


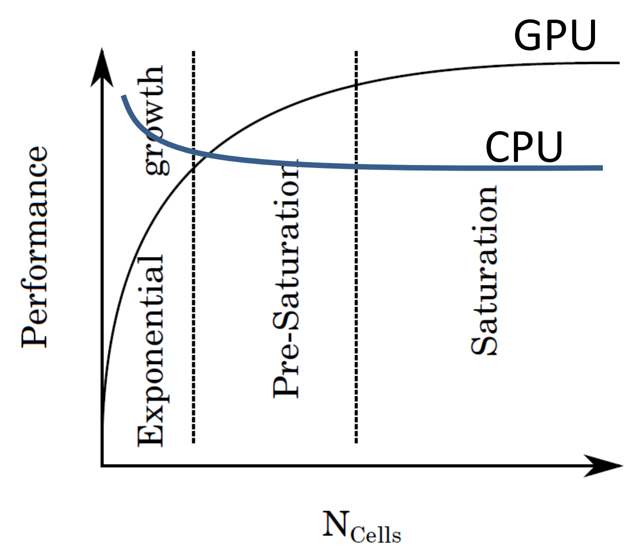
- Solve on fine grid
- Interpolate solution and residual to coarse grid



- Solve on coase grid assisted by fine residual
- Prolongate coarse correction to fine grid

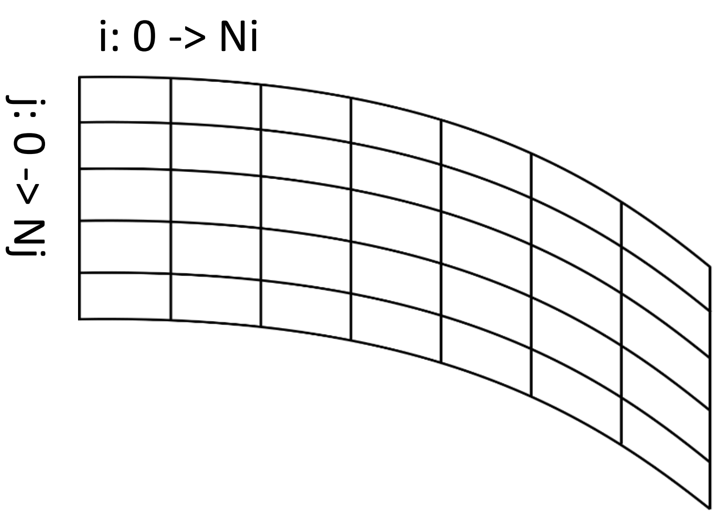
Cost of a 2-Grid scheme
converging to ideal cost of 1,125

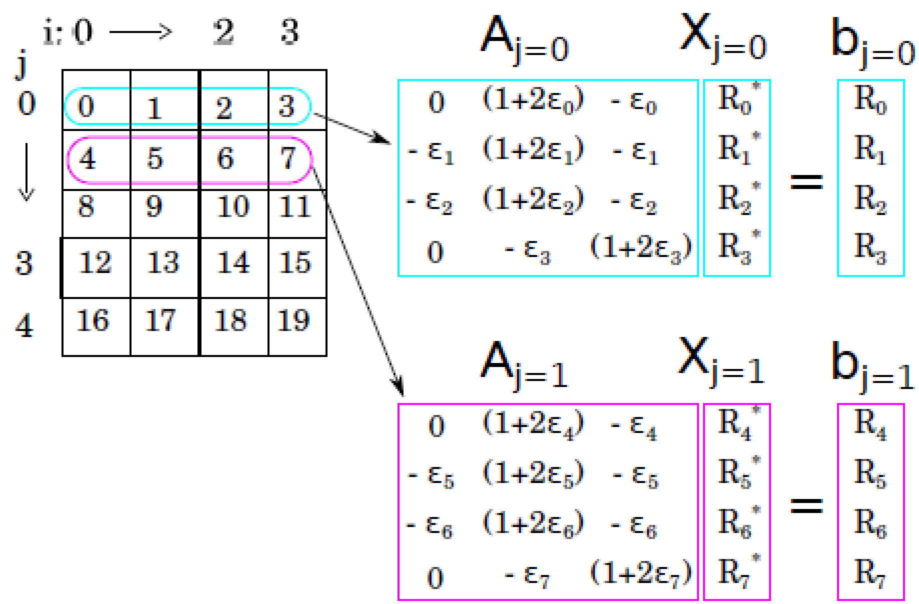




Convergence Acceleration on GPU (3/3): Implicit Residual Smoothing on GPU

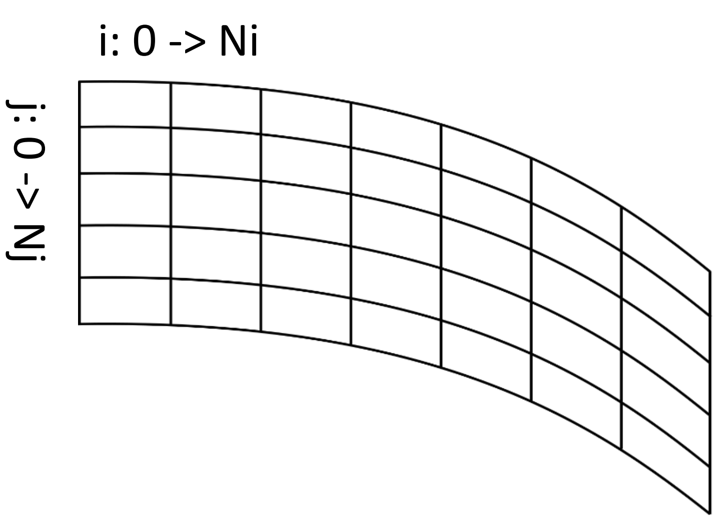
- Higher CFL → Oscillation in the solution.
- A smoother residual reduces the oscillation ->Higher CFLs.
- Smoothing: diffusion equation → solve a tridiagonal system.

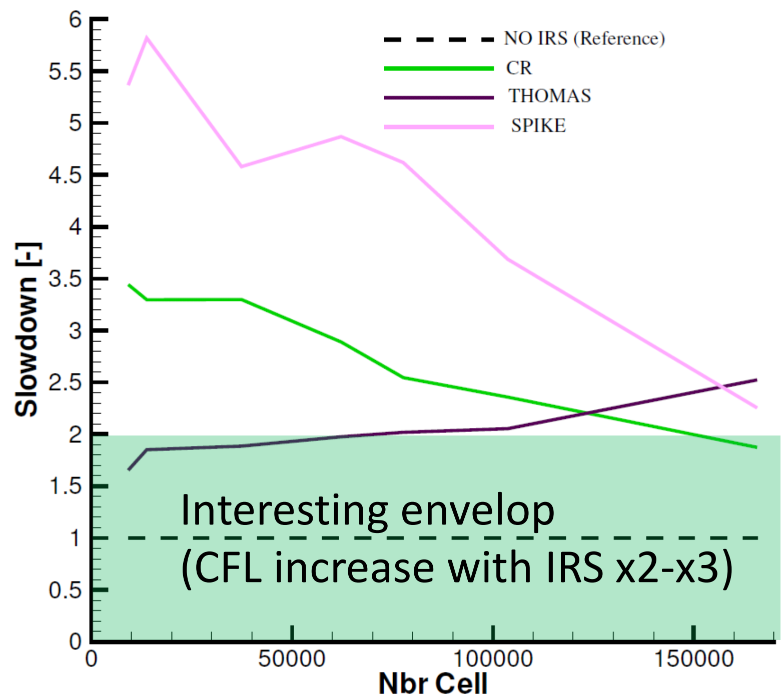




Convergence Acceleration on GPU (3/3): Implicit Residual Smoothing on GPU

- Higher CFL → Oscillation in the solution.
- A smoother residual reduces the oscillation ->Higher CFLs.
- Smoothing: diffusion equation → solve a tridiagonal system.





Main objective:
A more tangible GPU potential

CFD GPU solvers
- Explicit time integration
- Implicit time integration

Classification
of CFD operations

Proof-of-concept:
Optimization cases

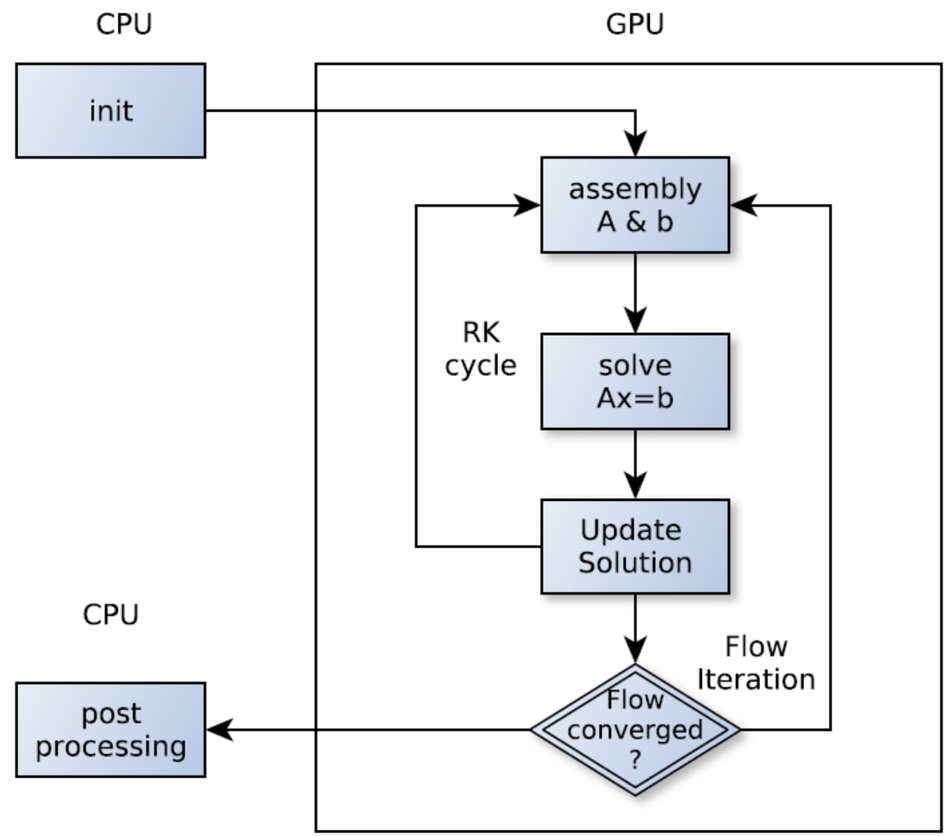
Summary
and Conclusions

Implicit Time Stepping is more Stable but …

$$\frac{\Omega I}{\Delta t}\Delta\vec{W}^n = -\vec{R}^{(n+1)}$$

$$\vec{R}^{n+1} \approx \vec{R}^n + \left(\frac{\delta\vec{R}}{\delta\vec{W}}\right)\Delta\vec{W}^n$$

$$\left[\frac{\Omega I}{\Delta t} + \left(\frac{\delta\vec{R}}{\delta\vec{W}}\right)\right]\Delta\vec{W}^n = \vec{R}^n$$



GMRES + Preconditioner

$$\left[\frac{\Omega I}{\Delta t} + \left(\frac{\delta \vec{R}}{\delta \vec{W}}\right)\right] \Delta \vec{W}^n = \vec{R}^n$$

$$Ax = b,$$

$$AM^{-1}u = b, x \equiv M^{-1}u$$

Algorithm 4 preconditioned GMRES

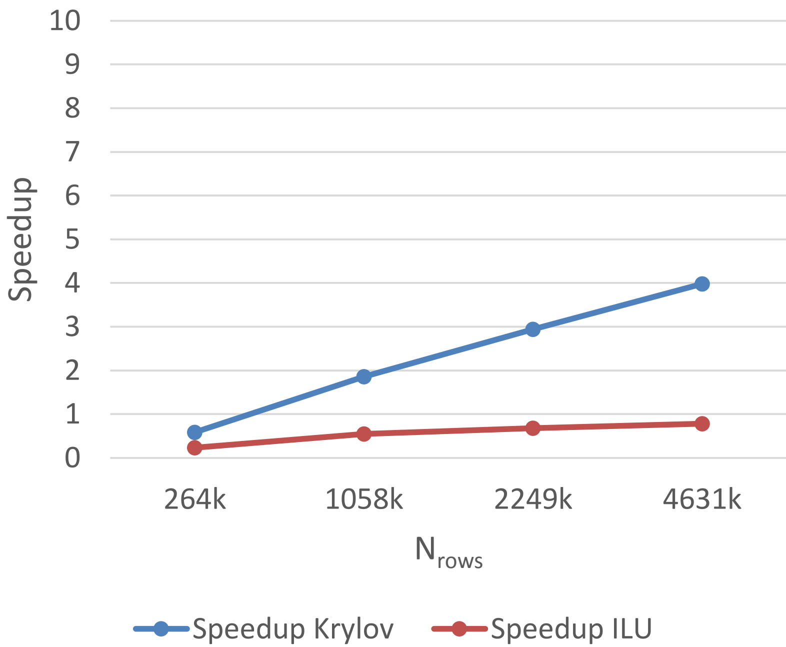
$r_0 = b - Ax_0$, $\beta := ||r_0||_2$ and $v_1 := r_0/\beta$
while $||r||_2 > \epsilon ||b||_2$ **do**
    for j=1 **to** m **do**
        $w_j := AM^{-1}v_j$
        for i=1 **to** j **do**
            $h_{ij} = (w_i, v_i)$
            $w_j := w_j - h_{ij}v_i$
        end for
        $h_{j+1,j} = ||w_j||_2$ and $v_{j+1} = w_j/h_{j+1,j}$
        $V_m := [v1, ..., v_m]$, $\bar{H} = h_{ij_{1\le i<m+1, 1\le j<m}}$
    end for
    $y_m = \text{argmin}_y ||\beta e_1 - \bar{H}_m y||)$
    $x_m = x_0 + M^{-1}V_m y_m$
    $x_0 := x_m$
end while

Algorithm 5 ILU(0)

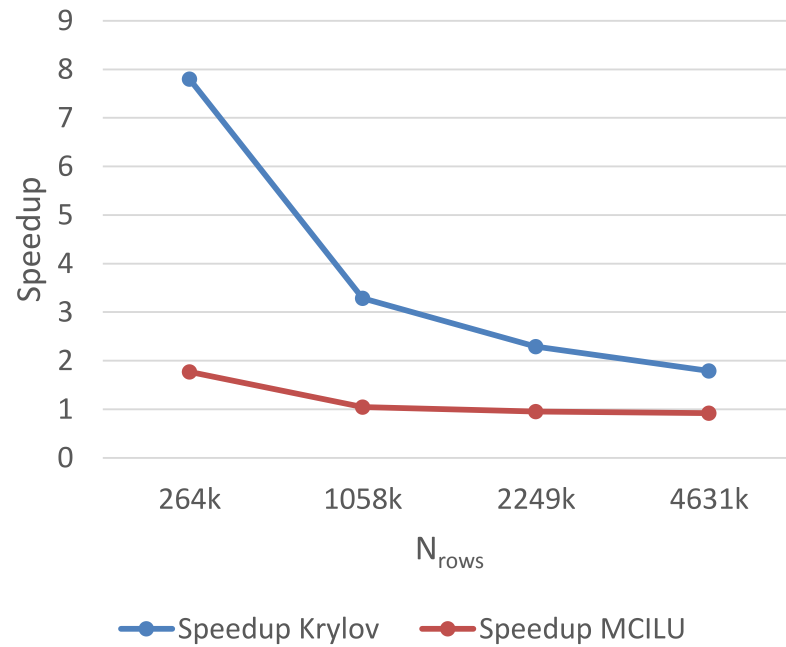
for i=2 **to** n **do**
    for k=1 **to** i-1 **do**
        if $(i,k) \in S$ **then**
            $a_{ik} = a_{ik}/a_{kk}$
            for j=k+1 **to** n **do**
                if $(i,j) \in S$ **then**
                    $a_{ij} = a_{ik}/a_{kj}$
                end if
            end for
        end if
    end for
end for

ILU is costly on GPU

- **ILU-GMRES:** Small gain on every iteration but ILU setup is slow:



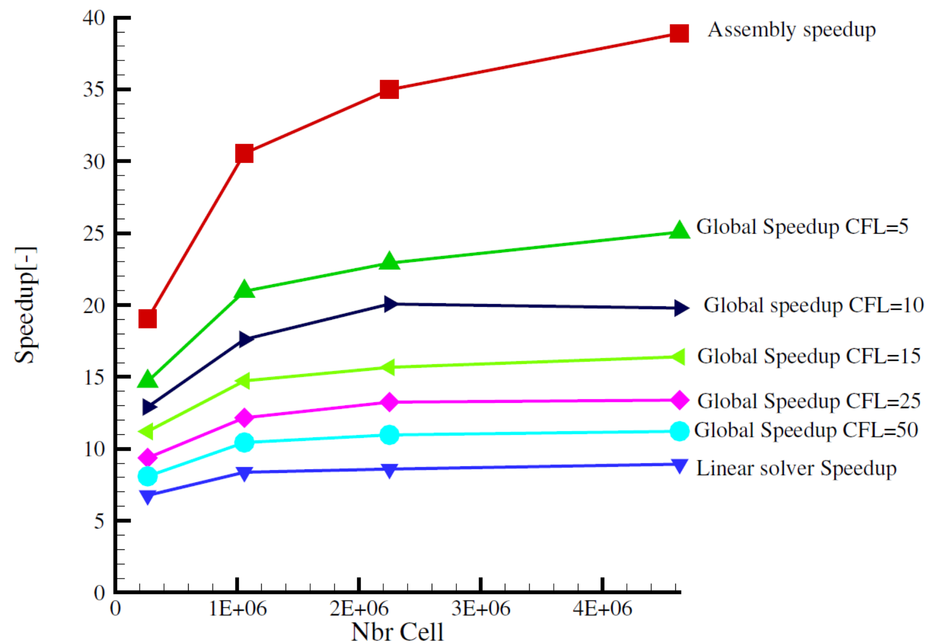
- **MCILU-GMRES:** Multi-colored ILU fast only for small problems.



Why not Jacobi PC

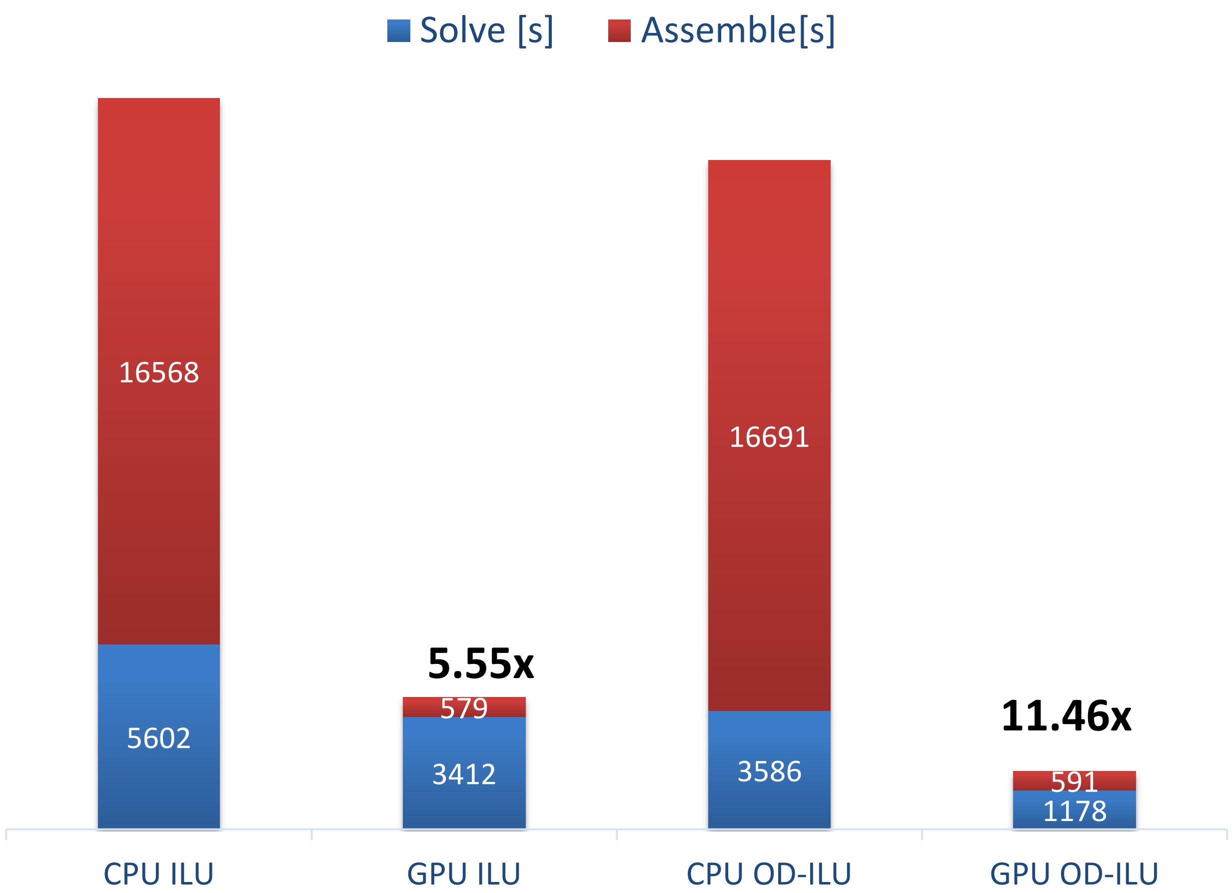
- **Jacobi-GMRES:** very fast but stable only for small time steps

- **Jacobi-GMRES:** Speedup decreases for higher CFLs



On-demand factorization

```
if (itr> MAX_ITR ) M <-LU_Factorization (A)
(x, itr) <- FGMRES (A,M,b)
```



Main objective:
A more tangible GPU potential

CFD GPU solvers

Classification
of CFD operations

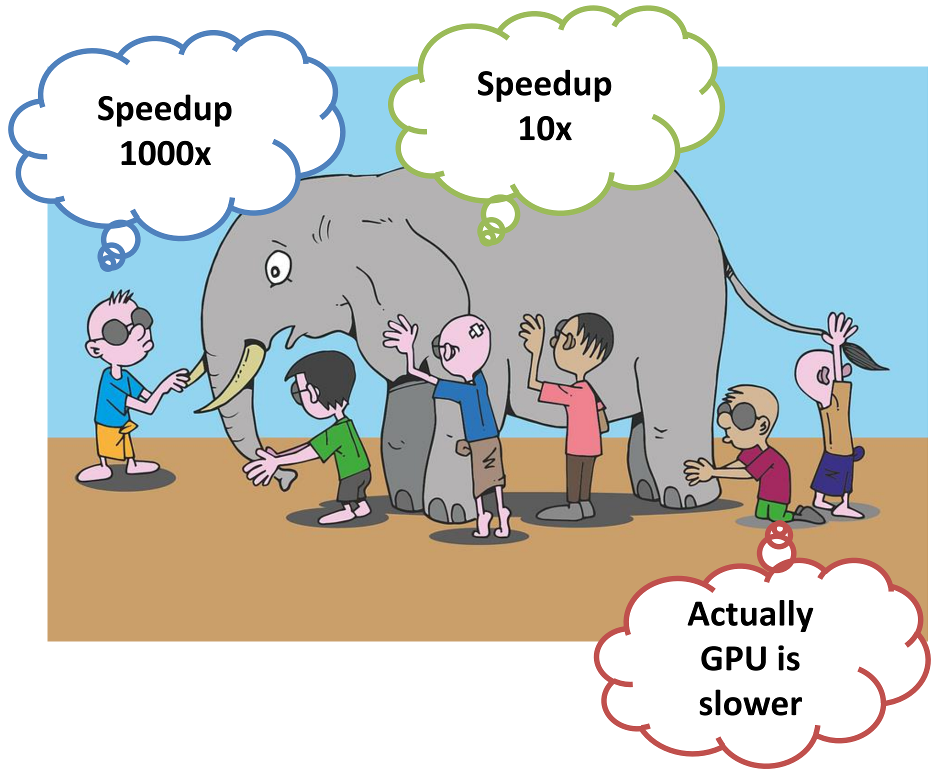
Proof-of-concept:
Optimization cases

Summary
and Conclusions

Classification (1/2): GPUs controversy

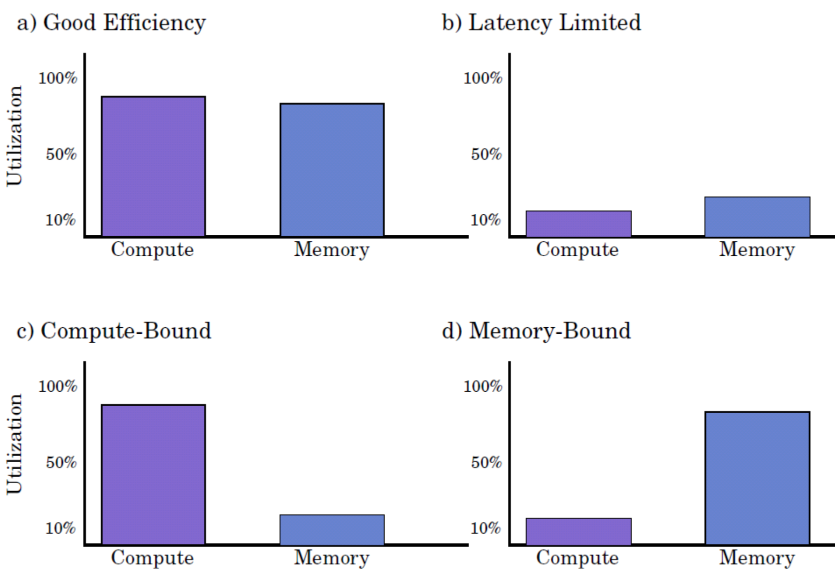
- GPU thousands of lightweight cores.
- Explicit solver: 10x to 100x speedup.
- Implicit solver: 1x to 10x speedup

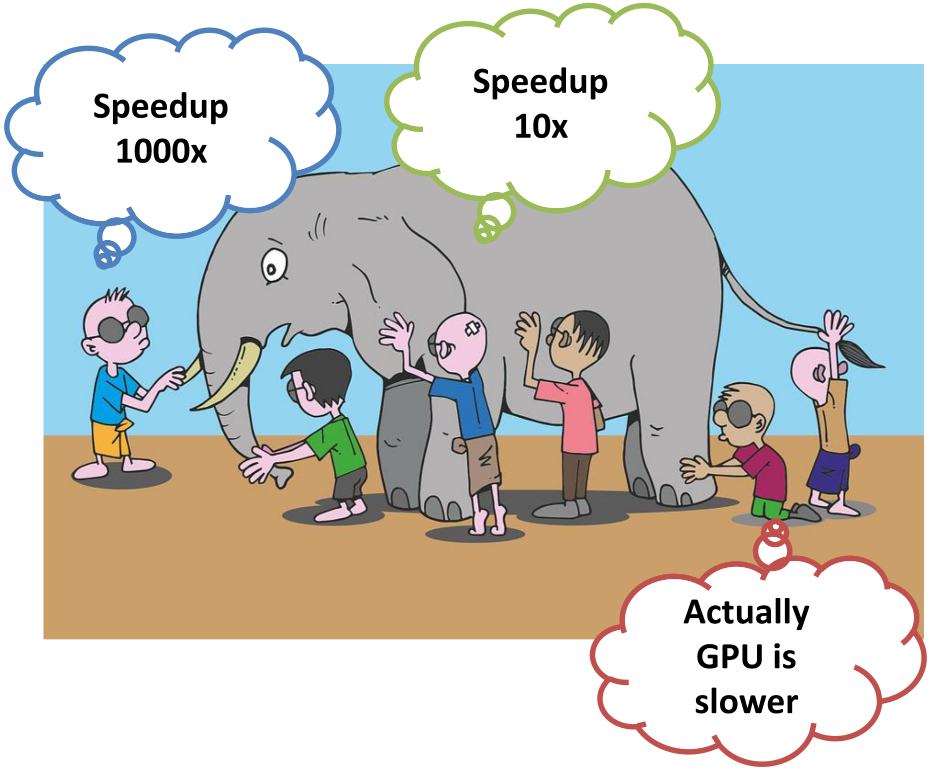
→ We need a classification



Picture modified from :https://pixabay.com/en/blind-men-elephant-story-feel-see-1458438/

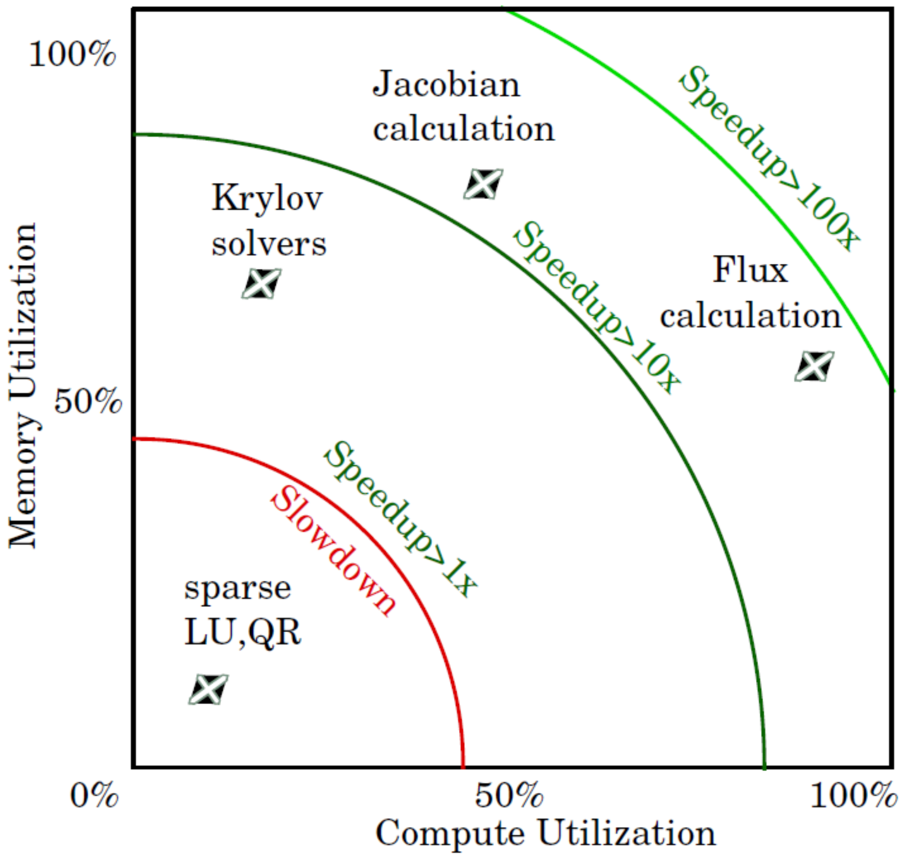
Classification (1/2): GPUs controversy

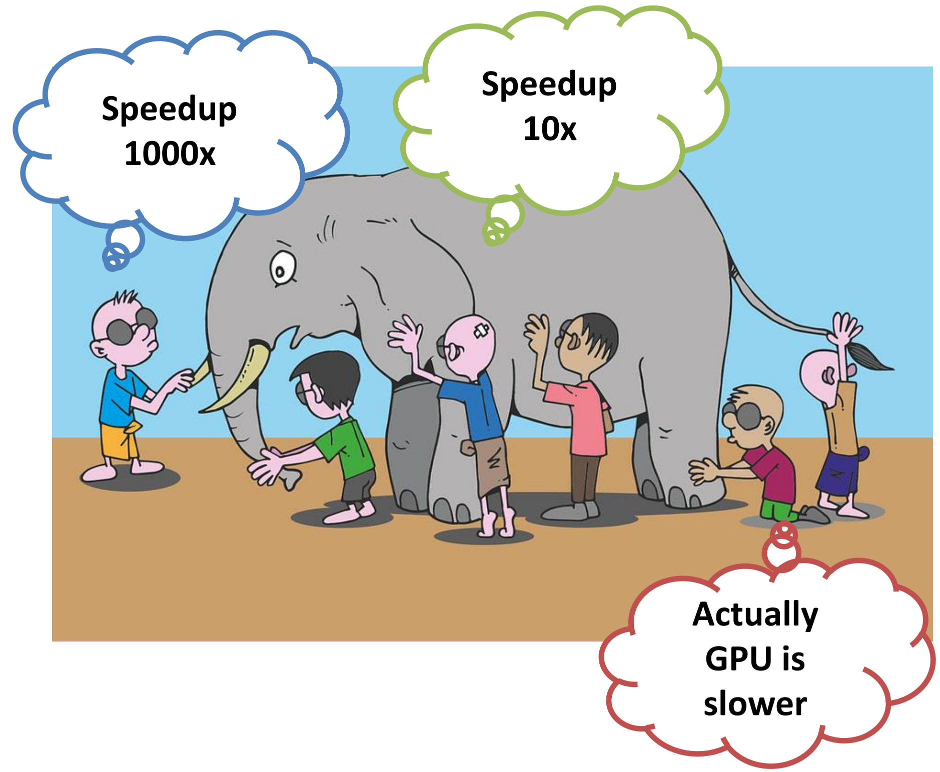




Picture modified from :https://pixabay.com/en/blind-men-elephant-story-feel-see-1458438/

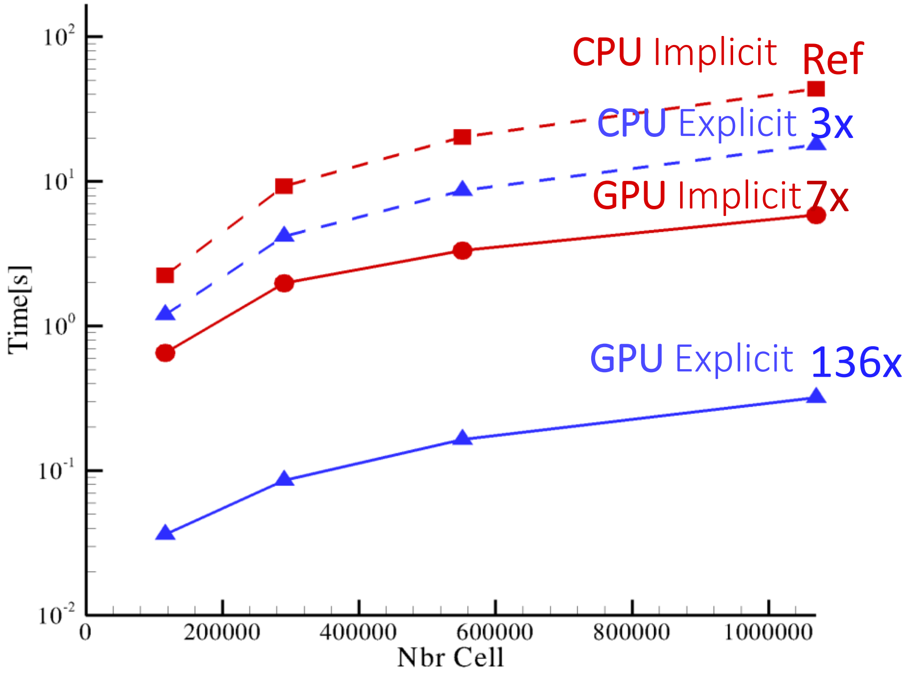
Classification (2/2): CFD operations





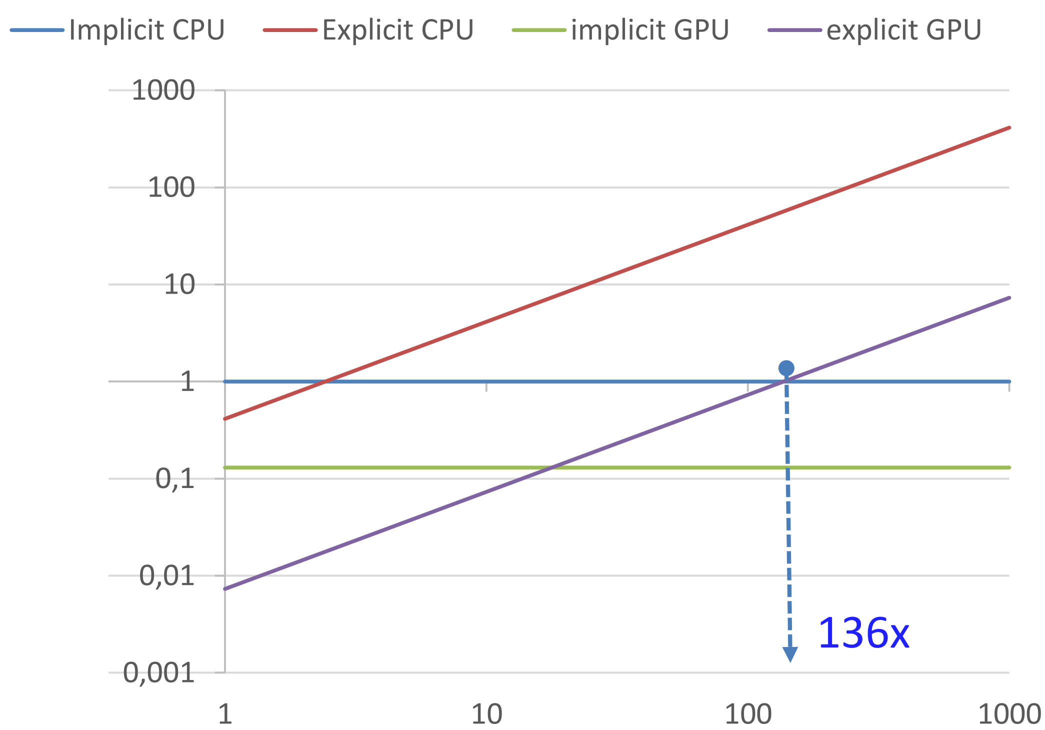
Full article:
Aissa M. , Verstraete ,T., and Vuik, c.. "*Toward a GPU-aware comparison of explicit and implicit CFD simulations on structured meshes.*" Computers & Mathematics with Applications 74.1 (2017): 201-217.

Performance Comparison: Explicit/Implicit



Performance Comparison: Explicit/Implicit

Normalized wall time



$$R_C = \frac{N_{ITR}^{Exp}}{N_{ITR}^{Imp}}$$

Main objective:
A more tangible GPU potential

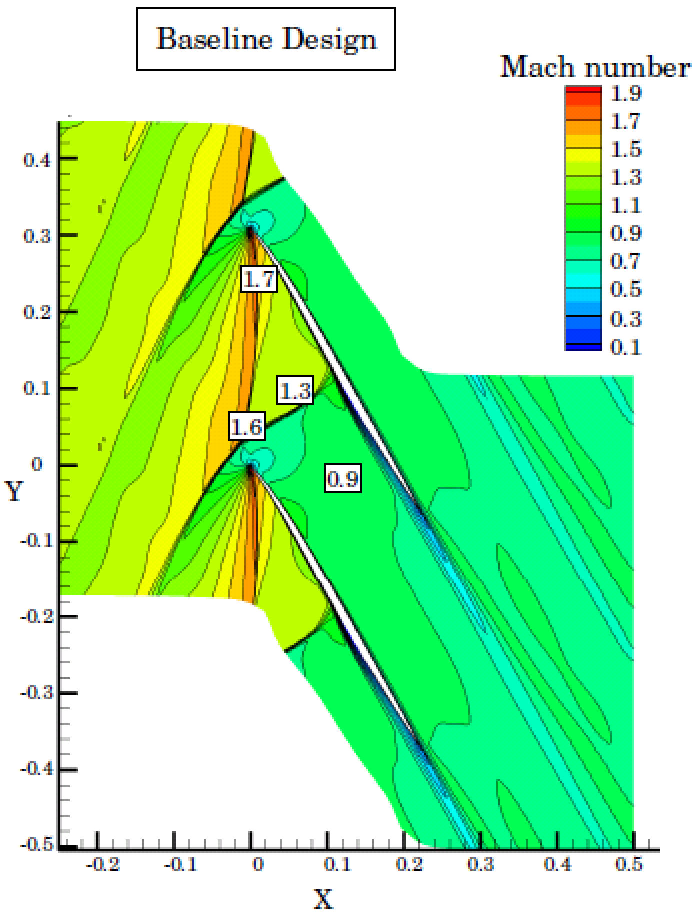
CFD GPU solvers

Classification
of CFD operations

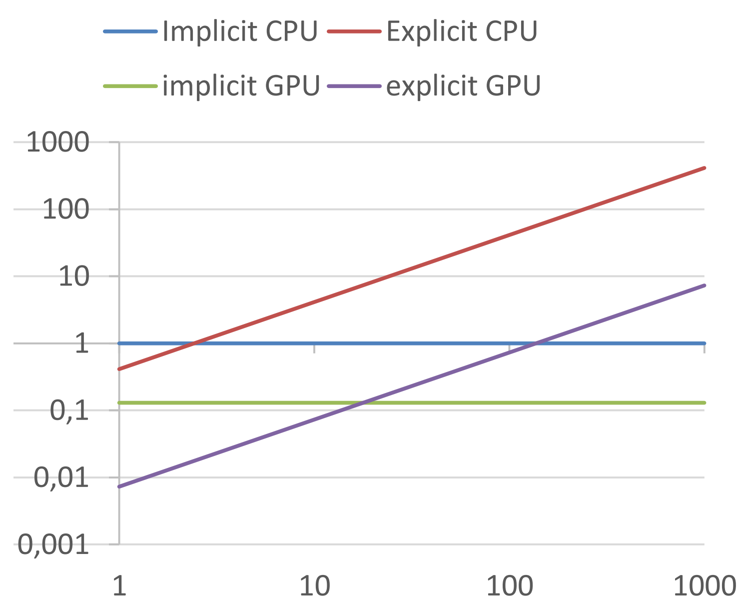
Proof-of-concept:
Optimization cases

Summary
and Conclusions

Example of a stator Optimization

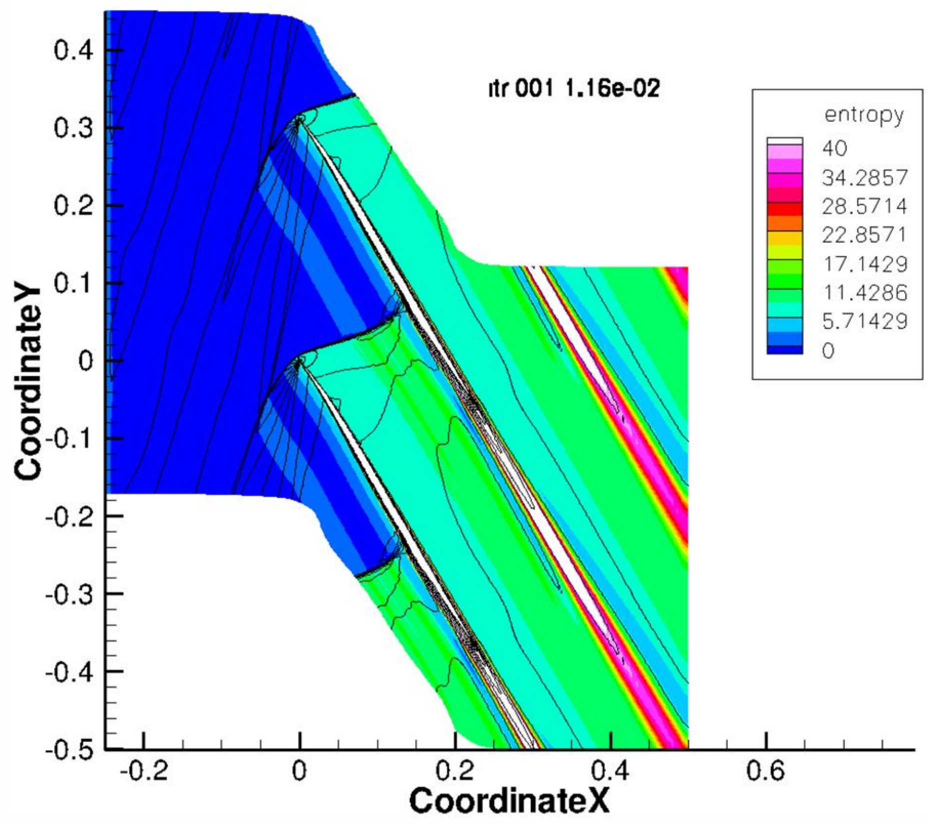


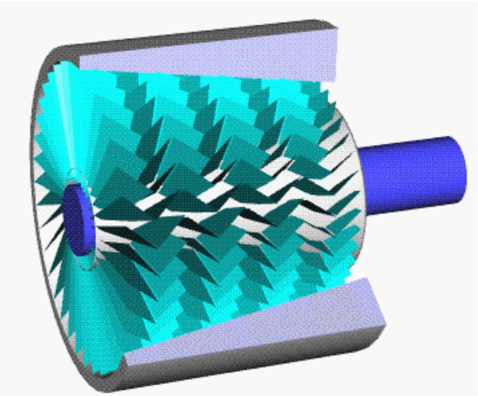
R_c=14 (low CFL for implicit =15)

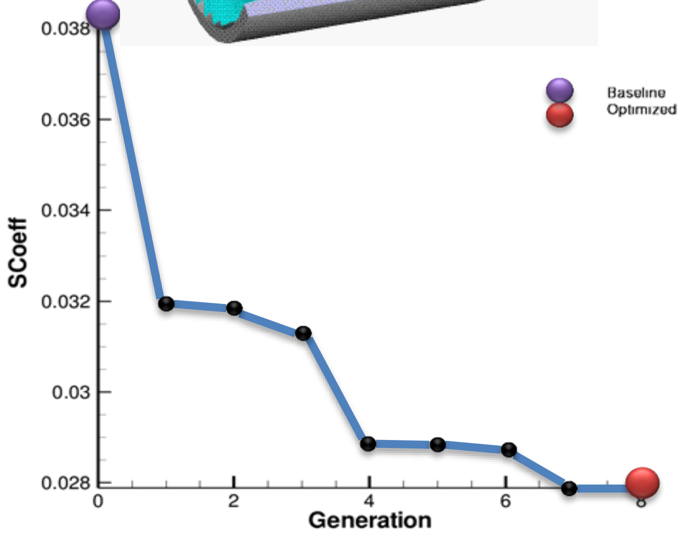


$$R_C = \frac{N_{ITR}^{Exp}}{N_{ITR}^{Imp}}$$

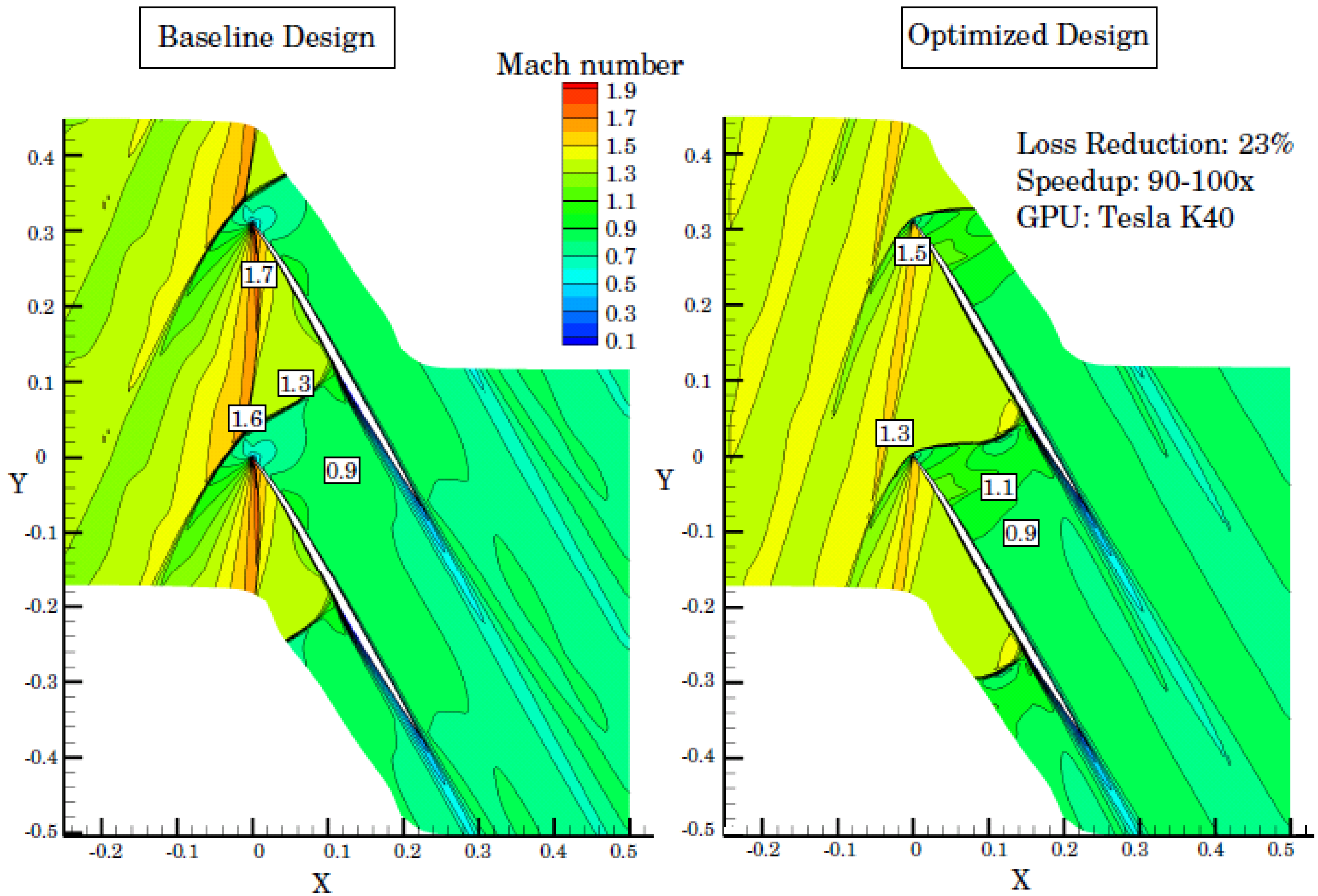
Example of a stator Optimization





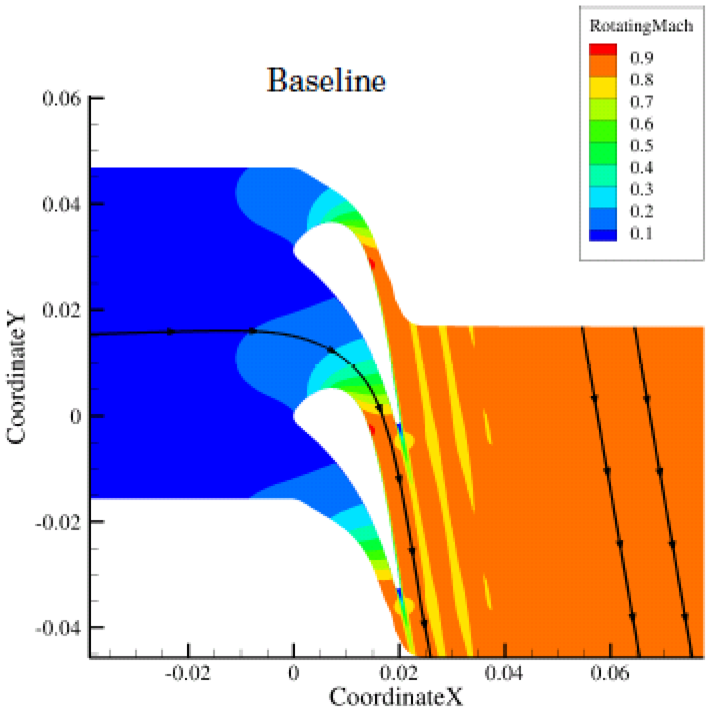


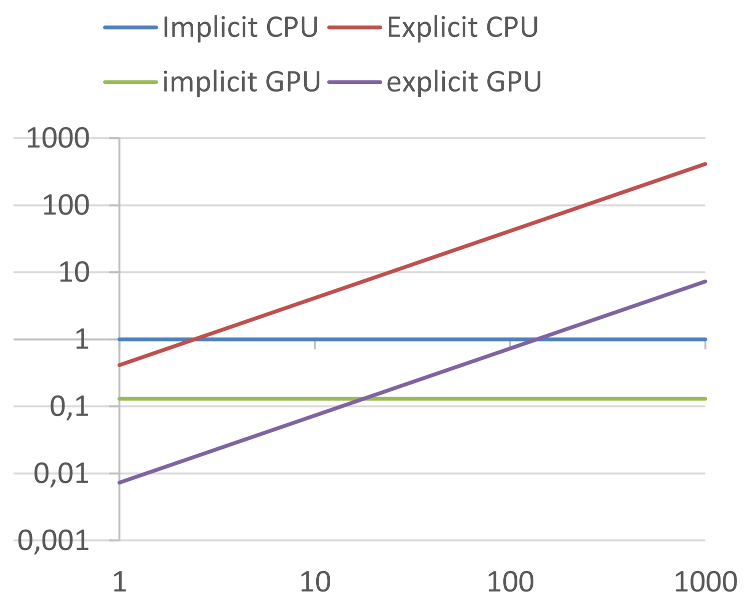
Example of a stator Optimization



LS82 cascade

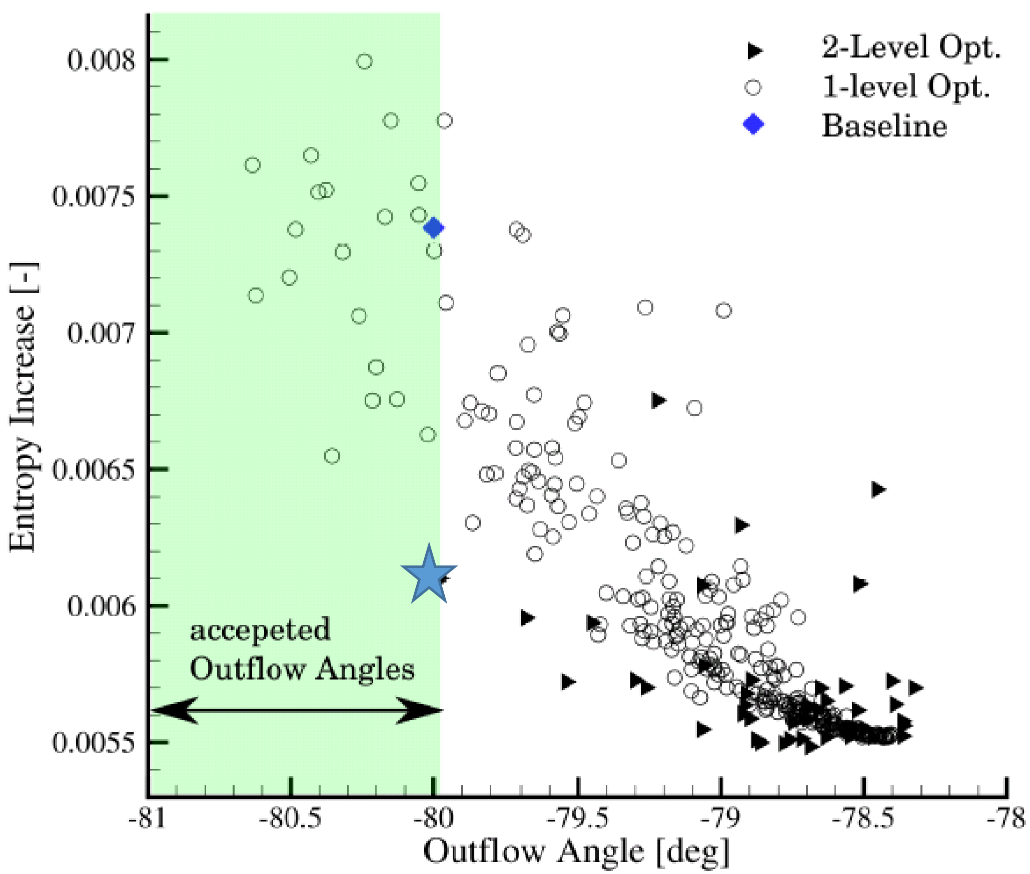
R_c=457 (Explicit solver bad flow convergence)

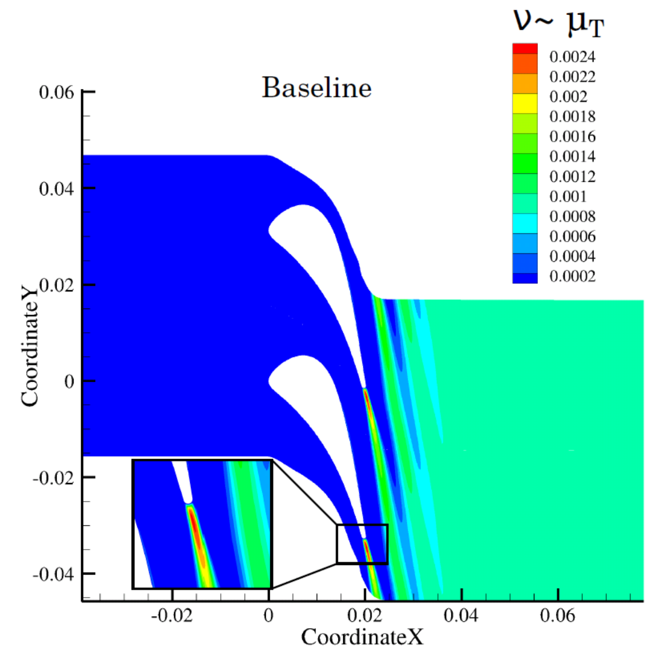


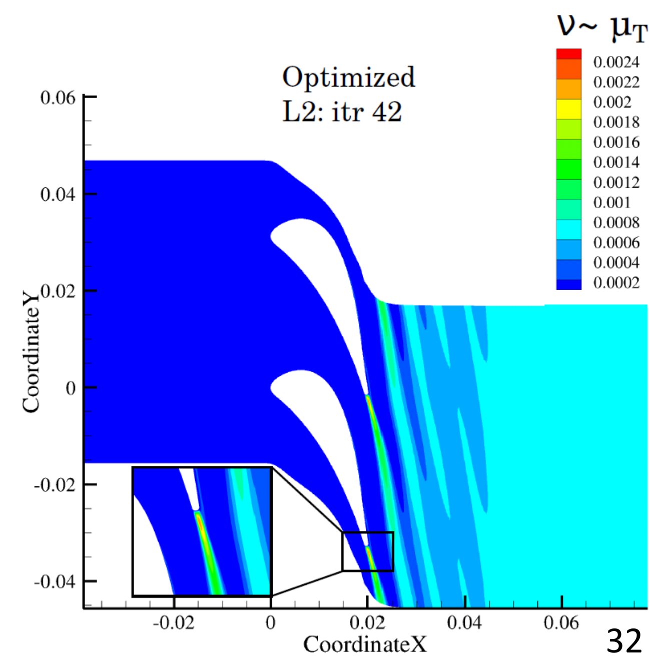


$$R_C = \frac{N_{ITR}^{Exp}}{N_{ITR}^{Imp}}$$

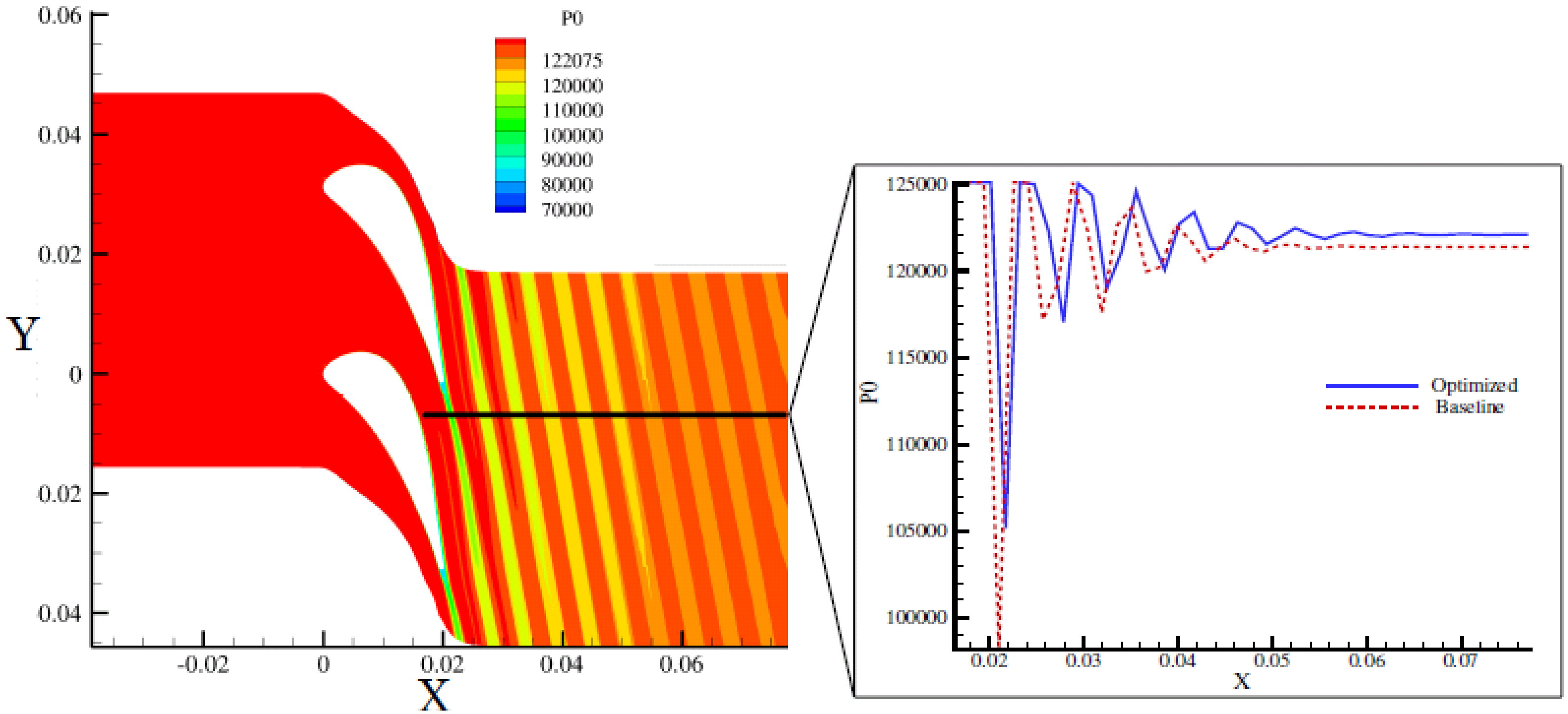
LS82 cascade: Results







LS82 cascade: Optimized blade



Main objective:
A more tangible GPU potential

CFD GPU solvers

Classification
of CFD operations

Proof-of-concept:
Optimization cases

Summary
and Conclusions

Summary

Explicit RANS:
100x-180x speedup.

Implicit RANS: 10x-20x speedup
(due to slow preconditioning.

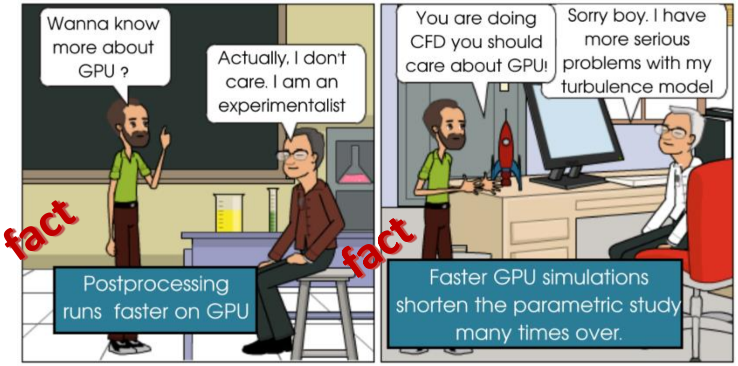
On-demand preconditioning:
x3 faster but GPU-friendlier
preconditioner is needed.

The classification:
an operation-specific
acceleration offers more insights.

Choice Explicit/Implicit:
Convergence ratio is decisive.

Can your simulation profit from the GPU?

- Where you situate your algorithm (slide 4: QR to ray-tracing)?
- Do you need double precision (for half-precision FPGA is faster)?
- ready to code (otherwise openACC is easier to use)?
- Anyone provided a classification for operation used in your field?



Thanks for your attention

Dr. Mohamed H. Aissa
Turbomachinery & Propulsion Department
Email: **aissa@vki.ac.be**

www.researchgate.net/profile/Mohamed_Aissa3