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

TU Delft
Delft University of Technology

VON KARMAN INSTITUTE FOR FLUID DYNAMICS
Can your simulation profit from the GPU?

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

Wanna know more about GPU?

Actually, I don't care. I am an experimentalist.

You are doing CFD you should care about GPU!

Sorry boy. I have more serious problems with my turbulence model.

Fact: Postprocessing runs faster on GPU.

Fact: Faster GPU simulations shorten the parametric study many times over.
Multi-core vs many-core
Massive Parallel Systems (e.g. GPU) as a trade-off

Source: the guardian.com
How fast is it?

Performance Gain

LU    QR    SpMV    FFT    Ray tracing  Lattice Boltzmann  Image processing

How to use a GPU

OpenACC

Ease of Use

Performance Gain

GPU libraries: cuFFT, cuBLAS...

OpenCL
Airplanes are getting more efficient and engine optimization is a main contributor.
TurboLab Stator (1/4)

- \( N_{\text{blades}} = 15 \)
- Chord length fixed

- Casing fixture

\[ d=2\text{mm} \]
\[ h=20\text{mm} \]
\[ d=10\text{mm} \]
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

\[ \int_{\text{casing}}^{\text{hub}} \alpha_{\text{whirl}}^2 \]

Every point is a costly CFD optimization \( \rightarrow \) need for a HPC solution

\[
Loss_{P_0} = \frac{p_{01} - p_{02}}{p_{01} - p_1}
\]
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”
Main objective:
A more tangible GPU potential

- CFD GPU solvers
- Classification of CFD operations
- Proof-of-concept: Optimization cases
- Summary and Conclusions

*All icons 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:

\[
\frac{\partial}{\partial t} \int_{\Omega} W \, d\Omega + \int_{\partial \Omega} (F_c - F_v) \, dS = \int_{\Omega} Q \, d\Omega \quad W = \{ \rho, \rho V_x, \rho V_y, \rho V_z, \rho E \}
\]

\[
\frac{\Omega}{\Delta t} \Delta \tilde{W}^n = -\mathbf{R}^n \quad \text{Explicit Time Stepping}
\]

\[
= - \mathbf{R}^{n+1} \quad \text{Implicit Time Stepping}
\]

\[
\Delta W^n = -\frac{\Delta t}{\Omega} R^n
\]

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

![Graph showing speedup over 1 core Xeon E3 for GTX980 and K40 with N Cells in thousands on the x-axis and Speedup on the y-axis, with markers at 162 for GTX980 and 90 for K40.]

![Diagram of the solver process with steps: CPU Initialization, Send Data to GPU, convective Fluxes (Mean+Turbulent), Viscous Fluxes (Mean+Turbulent), Source Term, Time Integration, Boundaries and Interfaces Update, Convergence?, Postprocessing on CPU, Send Solution to host.]
Convective Flux Evaluation (1/3)

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

\[
W_L = f(W_i, W_{i-1}) \quad \quad W_R = f(W_{i+1}, W_{i+2})
\]

<table>
<thead>
<tr>
<th>i-2</th>
<th>i-1</th>
<th>i</th>
<th>i+1</th>
<th>i+2</th>
<th>i+3</th>
</tr>
</thead>
</table>

Active Face
Convective Flux (2/3):
Thread mapping possibilities

- Face-wise is not thread-safe
Convective Flux (2/3):
Thread mapping possibilities

- 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): Multicolored (MC) vs redundant (Red)

Red: Run 1
- Read coalesced
- Compute Residual at face i
- Store coalesced

MC: Run 1
- Read striped
- Compute Residual at face i
- Store striped

MC: Run 2
- Read striped
- Compute Residual at face i+1
- Store striped

Memory bandwidth for striped memory access on GTX780:
$$A[i*\text{stride}] = B[i*\text{stride}] + C[i*\text{stride}]$$
**Convective Flux (3/3): Multicolored (MC) vs redundant (Red)**

<table>
<thead>
<tr>
<th></th>
<th>MC</th>
<th>RED</th>
</tr>
</thead>
<tbody>
<tr>
<td>Face fluxes per call</td>
<td>N/2</td>
<td>2N</td>
</tr>
<tr>
<td>Total faces fluxes</td>
<td>N</td>
<td>2N</td>
</tr>
<tr>
<td>Time per call [ms]</td>
<td>0,28</td>
<td>0,71</td>
</tr>
<tr>
<td>Total time [ms]</td>
<td>0,56</td>
<td>0,71</td>
</tr>
<tr>
<td>Operations ratio</td>
<td>-</td>
<td>2x</td>
</tr>
<tr>
<td>Total Speedup</td>
<td><strong>1,26x</strong></td>
<td>-</td>
</tr>
</tbody>
</table>

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 coarse 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 $\rightarrow$ Oscillation in the solution.
- A smoother residual reduces the oscillation $\rightarrow$ Higher CFLs.
- Smoothing: diffusion equation $\rightarrow$ solve a tridiagonal system.

\[
\begin{align*}
\text{i: } 0 & \rightarrow \text{ Ni} \\
\end{align*}
\]
Convergence Acceleration on GPU (3/3): Implicit Residual Smoothing on GPU

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

![Diagram with grid and graph showing convergence acceleration on GPU with implicit residual smoothing](image-url)
Main objective:
A more tangible GPU potential

- CFD GPU solvers
  - Explicit time integration
  - Implicit time integration

- Proof-of-concept: Optimization cases

- Classification of CFD operations

- Summary and Conclusions
Implicit Time Stepping is more Stable but ...

\[ \frac{\Omega I}{\Delta t} \Delta \bar{W}^n = -\bar{R}^{(n+1)} \]

\[ \bar{R}^{n+1} \approx \bar{R}^n + \left( \frac{\delta \bar{R}}{\delta \bar{W}} \right) \Delta \bar{W}^n \]

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

\[ A x = b, \]

\[ A M^{-1} u = b, x \equiv M^{-1} u \]

---

**Algorithm 4** preconditioned GMRES

1. \( r_0 = b - A x_0, \beta := \| r_0 \|_2 \) and \( v_1 := r_0 / \beta \)
2. \( \textbf{while} \| r \|_2 > \epsilon \| b \|_2 \) \textbf{do}
3. \( \textbf{for} j = 1 \textbf{ to } m \textbf{ do} \)
4. \( w_j := A M^{-1} v_j \)
5. \( \textbf{for} i = 1 \textbf{ to } j \textbf{ do} \)
6. \( h_{ij} = (w_i, v_i) \)
7. \( w_j := w_j - h_{ij} v_i \)
8. \( \textbf{end for} \)
9. \( h_{j+1,j} = \| w_j \|_2 \) and \( v_{j+1} = w_j / h_{j+1,j} \)
10. \( V_m := [v_1, ..., v_m], \bar{H} = h_{ij, 1 \leq i < m + 1, 1 \leq j < m} \)
11. \( \textbf{end for} \)
12. \( y_m = \text{argmin}_y \| \beta e_1 - \bar{H}_m y \| \)\)
13. \( x_m = x_0 + M^{-1} V_m y_m \)
14. \( x_0 := x_m \)
15. \( \textbf{end while} \)

---

**Algorithm 5** ILU(0)

1. \( \textbf{for} i = 2 \textbf{ to } n \textbf{ do} \)
2. \( \textbf{for} k = 1 \textbf{ to } i - 1 \textbf{ do} \)
3. \( \textbf{if} (i, k) \in S \textbf{ then} \)
4. \( a_{ik} = a_{ik} / a_{kk} \)
5. \( \textbf{for} j = k + 1 \textbf{ to } n \textbf{ do} \)
6. \( \textbf{if} (i, j) \in S \textbf{ then} \)
7. \( a_{ij} = a_{ik} / a_{kj} \)
8. \( \textbf{end if} \)
9. \( \textbf{end for} \)
10. \( \textbf{end if} \)
11. \( \textbf{end for} \)
12. \( \textbf{end for} \)
ILIU is costly on GPU

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

- **MCILIU-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)

Solve [s]  Assemble [s]

CPU ILU: 16568  5602 
GPU ILU:  5.55x  579  3412
CPU OD-ILU: 16691  3586
GPU OD-ILU: 11.46x  591  1178

5602  3412  3586  591
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
Classification (1/2): GPUs controversy

Performance Comparison: Explicit/Implicit

![Graph showing performance comparison between Explicit and Implicit methods for different hardware (CPU and GPU) with respect to time and Nbr Cell.]
Performance Comparison: Explicit/Implicit

Normalized wall time

\[ R_C = \frac{N^\text{Exp}_{\text{ITR}}}{N^\text{Imp}_{\text{ITR}}} \]

136x
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

\[ R_c = 14 \text{ (low CFL for implicit =15)} \]

\[ R_C = \frac{N^\text{Exp}}{N^\text{Imp}} \]

\[ \frac{1}{1000} \quad 0.1 \quad 1 \quad 10 \quad 100 \quad 1000 \]
Example of a stator Optimization

```
entropy
40
34.2857
28.5714
22.8571
17.1429
11.4286
5.71429
0
```

CoordinateY

CoordinateX

S_Coef

Generation

Baseline
Optimized
Example of a stator Optimization
LS82 cascade

\[ R_c = 457 \text{ (Explicit solver bad flow convergence)} \]

\[ R_c = \frac{N_{Exp}^{ITR}}{N_{Imp}^{ITR}} \]
LS82 cascade: Results

![Graph showing entropy increase vs outflow angle with markers for 2-Level Opt., 1-level Opt., and Baseline. Accepted outflow angles are indicated by a star.](image)

![Comparison of baseline and optimized flow fields with color mapping for $\nu \sim \mu_T$.](image)
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