Direct numerical flow simulation on vector and massively-parallel supercomputers

Johannes Peter, Markus Kloker
31st Workshop on Sustained Simulation Performance
18.03.2021
Overview

- Introduction
- Simulation of supersonic film cooling
  - Recent results
- Performance aspects on vector and massively-parallel systems
  - Kernel performance and scaling
  - Performance comparison: Hawk vs. Aurora
  - Weak scaling
- Conclusions
Introduction
Who we are and what we do

• Working group „Boundary layers, transition, and turbulence“
  • @ the Institute of Aerodynamics and Gas Dynamics, University of Stuttgart
  • 2 senior researchers, 2 postdocs, 10 PhD students

• Research on boundary-layer phenomena:
  • Boundary-layer instability and transition control
  • Turbulent boundary-layer flows
  • Fundamental turbulence properties
  • From incompressible to super-/hypersonic flows
What we do and how we do it

• Turbulence is governed by complex interaction and a large range of scales
  • Modelling is still far from perfect

• Direct Numerical Simulation: Calculate a flow without any modelling of turbulence
  • All turbulent scales are resolved: spatially and temporally
  ➢ Requires high accuracy in space and time
    • Requires a large amount of grid points and time steps
    • High-order and accurate methods
  ➢ High computational demands

• My cases:
  • Up to 5 billion grid points
  • ~500k to >1000k time steps
  • ~100-300 Gbyte Data per full 3D flowfield output
How we do it

Code information

• NS3D: in-house developed Fortran code
  • Solves the compressible unsteady Navier-Stokes equations
• Spatial: high-order **finite differences** up to \( O^8 \)
• Temporal: **explicit** Runge-Kutta \( O^4 \)
• De-aliasing: compact filter \( O^{10} \)
• Block-structured grids
• Parallelized using hybrid **MPI/OpenMP**
• Available for flows of one and two gas components
Supersonic Film Cooling
What I do
Vulcain 2 – Ariane 5 main engine
What I do

Rocket engines are cool, but very hot

• Vulcain 2 specifications:
  • Combustion temperature $T_0 \approx 3500 \text{ K}$
  • Combustion pressure $p_0 \approx 115 \text{ bar}$
  • Thermal power $\sim 3 \text{ GW}$
  • Vacuum thrust $\sim 1.35 \text{ MN}$

• An increase in efficiency, power, reliability, reusability is desired

➢ Improved active cooling is necessary
Supersonic Film Cooling

- Film cooling: a (cold) **secondary gas is injected** close to the wall into a hot main flow
- Effective cooling method in a rocket nozzle, where the main flow is supersonic
- My simulations
  - Freestream $gH_2O$ at $Ma_\infty = 3.3$ and $T_\infty = 1980$ K ($T_0 = 3600$ K)
  - Coolant helium at $Ma_c = 1.8$ and $T_{0,c} = 330$ K
Supersonic Film Cooling

- Why DNS?
  - **Understand** the complex mixing behavior in detail
  - Provide **design-guidlines** for film-cooling applications
  - Provide **data & reference cases** for turbulence modelling
  - Can relatively easy **vary flow conditions**, especially temperature
- Experimental limits:
  - „cold“ → long time → temperatures too low
    → accessible for measurement
  - „hot“ → short time → wall temperature stays at ambient initial condition
    → measurements are very difficult
Supersonic Film Cooling

Turbulent transport

- Turbulent fluctuations cause additional transport
- The ratio of turbulent transport of momentum to energy/mass is given by the turbulent Prandtl/Schmidt number:

\[
Pr_t = \frac{\overline{\rho u'' v''}}{\overline{\rho u'' T''}} \frac{\partial \tilde{T}}{\partial y} \frac{\partial \tilde{u}}{\partial y} \\
Sc_t = \frac{\overline{\rho u'' v''}}{\overline{\rho u'' c_1''}} \frac{\partial \tilde{c}_1}{\partial y} \frac{\partial \tilde{u}}{\partial y}
\]

- Naturally derived from DNS data (post-processing)
- Important modelling parameters in RANS turbulence models
  - Usually assumed constant, \( \approx 0.7 - 0.9 \)
Supersonic Film Cooling

Turbulent transport

- Turbulent Prandtl number varies substantially
  - Between 0.4 and 1.5
  - Large gradients exist
- Schmidt number behaves comparably
- DNS data can now help in developing models for varying $Pr_t, Sc_t$
  - $Pr_t, Sc_t$ not strongly affected by the coolant mass flux
Supersonic Film Cooling
Influence of upstream wall temperature

- Adiabatic wall temperature: $T_{\text{rec}} \approx 3480$ K
- In a real application, the upstream wall must be cooled: e.g. by regenerative cooling
  - Reference case: upstream wall is adiabatic
  - Cooled cases: upstream wall is isothermal at $\frac{T_W}{T_{\text{rec}}} \approx 0.5$, $\frac{T_W}{T_{\text{rec}}} \approx 0.35$
Supersonic Film Cooling

Influence of upstream wall temperature

- Close to the slot: the pre-cooling leads to a decrease in effectiveness
- Further downstream: effectiveness is increased with upstream wall cooling
- Cooling leads to a higher shear and a stronger instability in the shear and mixing layer between main flow and coolant
  - Enhanced transport of heat and mass
  - Turbulent Prandtl and Schmidt number are also influenced
Performance Aspects
Code details

• NS3D: in-house developed Fortran code
  • Solves the compressible unsteady Navier-Stokes equations
    • Spatial: high-order finite differences up to $O_8$
    • Temporal: explicit Runge-Kutta $O_4$
    • De-aliasing: compact filter $O_{10}$

• Block-structured grids
• Parallelized using hybrid MPI/OpenMP

• Available for flows of one and two gas components
Code details
Spatial derivatives

• Schemes for spatial derivatives:
  • Explicit finite differences (EFD) O8
    
  • Compact finite differences (CFD) O6
    • Requires the solution of a tri-diagonal equation system
      
      ➢ (i−1) loop dependency (for x-derivative)
      
      ➢ **Not possible to vectorize innermost loop** (for x-derivative)
    • Global decoupling between MPI ranks is achieved by replacing „out-of-domain“ derivatives with EFD
Basic code structure

1. Calculate spatial derivatives: $\frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial}{\partial z}$ from state $Q = (\rho, \rho\vec{u}, E)$.

2. Calculate temporal derivative: $\frac{\partial Q}{\partial t} = f\left(\frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial}{\partial z}\right)$ from Navier-Stokes equations.

3. Update flow state: $Q^n \rightarrow Q^{n+1}$ with Runge-Kutta sub-step

➢ Three computational kernels

➢ All in the basic form of three nested loops:

```
    do i = 1, nx
        do j = 1, ny
            do k = 1, nz
```
Kernel scaling

- Kernel scaling w.r.t. domain size?
- Code striped of everything but computational kernels

- Comparison of one „Cluster Element“ (CE):
  - Hawk: 1 Node = 2 x 64 cores AMD Epyc 7742
  - Aurora: 1 Vector Engine = 8 cores SX-Aurora Tsubasa, Type 10B
  - (Smallest unit to allocate and (very) roughly comparable in cost and power)

- Each CE is fully used, i.e.:
  - 32 MPI processes at 4 OpenMP threads for Hawk (⇒ one process per CCX ⇒ sharing L3 cache)
  - 8 MPI processes at 1 OpenMP threads for Aurora
Kernel scaling

Hawk – CFD

- Very high performance for very small grid sizes
- Performance levels off for Runge-Kutta and d/dt kernels ⇒ memory bandwidth
- Performance reduces with increasing size for spatial derivatives ⇒ caching
Kernel scaling
Aurora – CFD

- Strong **performance variations** for tiny changes in grid size
- Wrong domain extend can lead to **bad memory access** patterns
Kernel scaling

Aurora – CFD

- Very low performance for small grid sizes ⇒ short vector length
- Performance for all kernels levels off
- Spatial derivatives need long innermost AND middle loop length (especially d/dx)
Kernel scaling
Hawk & Aurora – CFD

- RuKu- and d/dt-Kernels significantly faster on Aurora
- For spatial derivatives comparison depends on grid size
Kernel scaling
Hawk & Aurora – EFD

- Principally **same picture as for CFD** – just with higher overall performance
- RuKu- and d/dt-Kernels significantly faster on Aurora
- For spatial derivatives comparison depends on grid size
Basic code structure

1. Calculate spatial derivatives: \( \frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial}{\partial z} \) from state \( Q = (\rho, \rho u, E) \).
   • MPI data exchange and related operations

2. Calculate temporal derivative: \( \frac{\partial Q}{\partial t} = f\left(\frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial}{\partial z}\right) \) from Navier-Stokes equations.

3. Update flow state: \( Q^n \rightarrow Q^{n+1} \) with Runge-Kutta sub-step
   • Set boundary conditions

4. Additional stuff, e.g.:
   • Auxiliary functions, e.g. calculation of viscosity
   • Filtering
   • I/O
   • Calculation of postprocessing values (e.g. mean)

All the icky stuff:
• Short loops
• Conditional loops
• Indirect array access
• Idle times
• etc.
Performance comparison

- **Realistic** simulation:
  - Supersonic turbulent boundary-layer
  - Use of complex **boundary conditions**, filtering, etc.
  - Comparing 32 VE vs. 32 nodes Hawk

- Four testcases:
  - Small-1: Small # of grid points, innermost loop does **not fill vector length**
  - Small-2: Same # of grid points as Small-1, modified to **fill vector length** on innermost loop
  - Medium: 2 x # of grid points from small cases, comparable to very large cases on Hawk
  - Large: Full efficiency on Aurora
Performance comparison

- Four testcases:
  - Small-1: Small # of grid points, innermost loop does not fill vector length
  - Small-2: Same # of grid points as Small-1, modified to fit vector length on innermost loop
  - Medium: Doubled # of grid points from small cases
  - Large: Full efficiency on Aurora

- Relative performance based on „Iterations/s“: **higher is better**

<table>
<thead>
<tr>
<th></th>
<th>CFD</th>
<th>EFD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hawk</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>Aurora</td>
<td>73%</td>
<td>83%</td>
</tr>
<tr>
<td></td>
<td>Small-1</td>
<td>Small-1</td>
</tr>
<tr>
<td></td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td></td>
<td>120%</td>
<td>130%</td>
</tr>
<tr>
<td></td>
<td>Medium</td>
<td>Medium</td>
</tr>
<tr>
<td></td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td></td>
<td>148%</td>
<td>154%</td>
</tr>
<tr>
<td></td>
<td>Large</td>
<td>Large</td>
</tr>
<tr>
<td></td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td></td>
<td>185%</td>
<td>195%</td>
</tr>
</tbody>
</table>
Weak scaling

- Perfect scaling on Hawk
- Very good weak scaling on Aurora, 95% on 64 VE
  - Peculiar dips for non-$2^n$-configurations
Weak scaling

- Scaling with ~98% efficiency from 1 to 1024 nodes on Hawk
- However large performance variations still occur
Conclusions
Conclusions 1/2
Supersonic film cooling

- DNS of supersonic film cooling have been performed
  - High fidelity DNS requires high computational power
  - DNS can offer valuable **insights into the flow physics**

- The **upstream wall temperature** has a non-negligible **influence** on the film-cooling behavior

- **Turbulence modelling parameters** can be evaluated from DNS
  - Turbulent Prandtl and Schmidt numbers are *not* constant in the mixing region
Conclusions 2/2
Performance aspects

• One **Aurora** VE can **outperform** one node **Hawk**
  • Certain **minimum grid sizes** need to be met
    ➢ Disadavantageous for strong scaling
  • Choosing the wrong domain sizes can lead to **bad memory access patterns**
  • „Only“ 48 GByte memory per VE

• **Aurora sustained performance:**
  • ~10% of peak (CFD)
  • ~13% of peak (EFD)
  • Probably still potential for optimization

• **Weak scaling is very good** on Aurora, almost perfect on Hawk
Thank you!

Johannes Peter

e-mail  johannes.peter@iag.uni-stuttgart.de
phone   +49 (0) 711 685-63429
www.iag.uni-stuttgart.de

University of Stuttgart
Institute of Aerodynamics and Gas Dynamics
Pfaffenwaldring 21