Readiness of HPC Extreme-scale Applications (2nd Edition)
ISC HPC 2025 Workshop
Friday, June 13, 2025, 2:00pm - 6:00pm

STREAmS: portability, performance, maintainability.

Can they coexist?

F. Salvadore

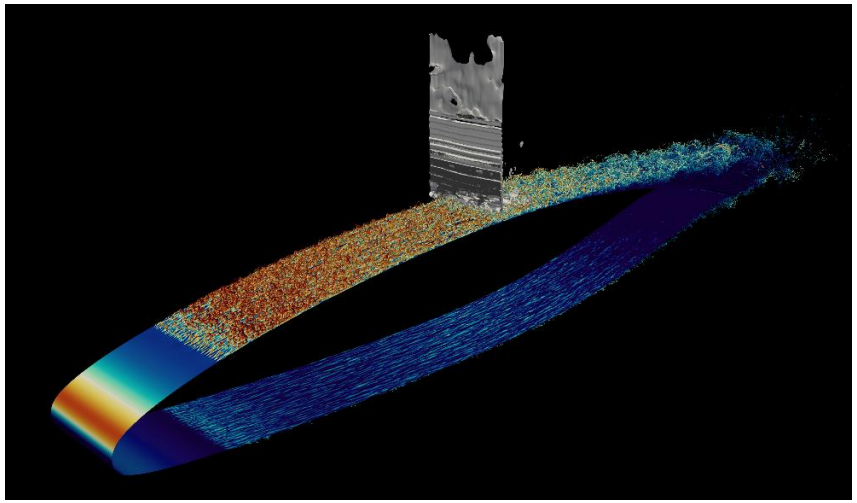
HPC Department, CINECA

EXCELLERAT Center of Excellence

- The European Centre of Excellence for Engineering Applications
  - https://www.excellerat.eu/
  - The EXCELLERAT project is a single point of access for expertise on how data management, data analytics, visualisation, simulation-driven design and co-design with high-performance computing (HPC) can benefit engineering.
  - funded by EuroHPC

EuroHPC JU is a joint initiative between the EU, European countries and private partners to develop a World Class Supercomg Ecosystem in Europe.

- Application software developed and used within EXCELLERAT:
  - AVBP / Alya / CODA / STREAmS / Neko / m-AIA / OpenFOAM
- **Use Case 6 - CINECA & Sapienza University team**:
  - active flow control for drag reduction (DR) of transonic airfoils
  - aims to perform DNS at cutting-edge Reynolds number of uncontrolled and controlled (streamwise-travelling waves) airfoils
  - code is **STREAmS-v2.1** using recent curvilinear grid implementation
  - team: **S. Pirozzoli, G. Soldati, M. Bernardini (Sapienza University of Rome)**



Supercomputers today

- Modern **High Performance computing** (HPC) are increasingly becoming heterogenous to achieve exascale computing goals
  - Involves using multiple cores of more than one type of processor
  - offloading to accelerators not only led to increased speedups but also to energy efficiency
- Top500 list: 9 out of top 10 supercomputers are built utilizing GPU acceleration:
  - **5 AMD, 4 NVIDIA, 1 INTEL GPUs**
- Pre-**Exascale** systems currently in EU, Exascale system coming shortly
  - **Leonardo** (CINECA): NVIDIA A100 GPUs
  - **LUMI** (CSC): AMD MI250X GPUs
  - **MareNostrum5** (BSC): NVIDIA H100 GPUs
  - **JUPITER** (JSC): NVIDIA GH200 GPUs, under finalization, it will be exascale
- Exascale architectures already in production in US
  - **El Captain/Frontier**, AMD GPUs - **Aurora**, Intel GPUs

*TOP 500, Nov 2022

Heterogenous computing challenges

- Accelerating existing **CPU** parallel codes using **GPU** has been the norm in the last few years
- Classic and new challenges :
  - **performance**: algorithm / implementation / scalability
  - **sustainability**: portability / maintainability
- Portability: ability of code to run on different systems with minimal or no modifications
  - extended meaning: there are code modifications but integrated in a unique platform architecture
  - unfortunately: different programming paradigms still needed to get best performance on different vendors' devices
- Programming paradigms:
  - Multiple Vendors/paradigms (CUDA, HIP): portability?
  - Open paradigms (OpenMP, OpenACC): compiler support?
  - Performance portability libraries (kokkos, Raja, alpaca): maintenance?

OpenACC
More Science, Less Programming

NVIDIA CUDA

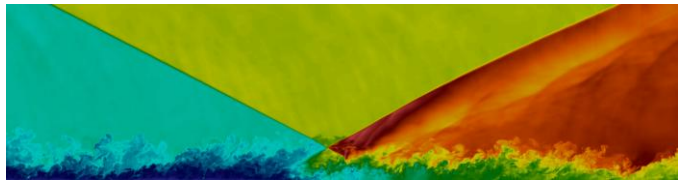
OpenCL™

oneAPI

AMD ROCm

OpenMP

STREAmS solver



- **S**upersonic **T**u**R**bul**E**nt **A**ccelerated N**S** solver
- Finite-Difference code for DNS of high-speed flows
  - Oriented to canonical cases: boundary layer, channel, compression ramp, airfoil
- Numerical approach
  - Kinetic energy preserving (KEP) schemes
  - WENO reconstructions for shock capturing
  - Explicit third-order low-storage RK scheme for time advancement
  - Immersed boundary approach for complex geometries
- Modern Fortran with object oriented framework
  - Multiple backends for CPUs, NVIDIA, AMD and Intel GPUs
- Open-Source GPL 3 license
  - https://github.com/STREAmS-CFD/STREAmS-2

Legacy

- Written in Fortran 77
- CPU parallel only
- Over 15 years of development history

v1

- Refactored in 2021
- Written in Fortran 90
- Both CPU and GPU modes supported
- CUDA Fortran for GPU
- Procedurally programmed with *ifdef*

v2

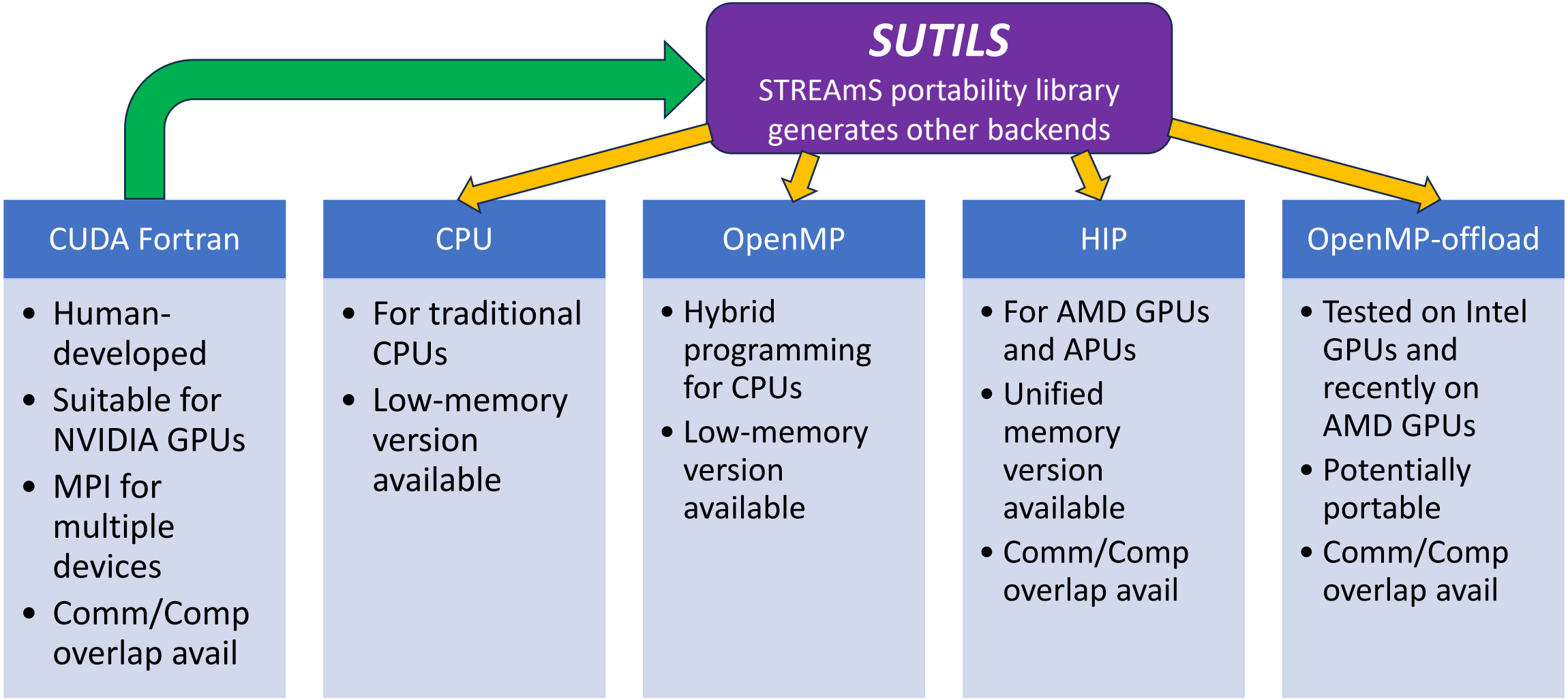
- Refactored in 2023
- Written in Fortran 2008
- Utilises Modern Fortran OOP features
- Based on a multi-backend/multi-equation approach

v2.1

- OpenMP, HIP, OpenMP-offload backends added
- Curvilinear grids (including C-mesh for airfoils)
- Open-source extended
- Published in 2025

[a] *Bernardini et al., CPC, 2021*
[b] *Bernardini et al., CPC, 2023*

STREAmS programming paradigms



sutils

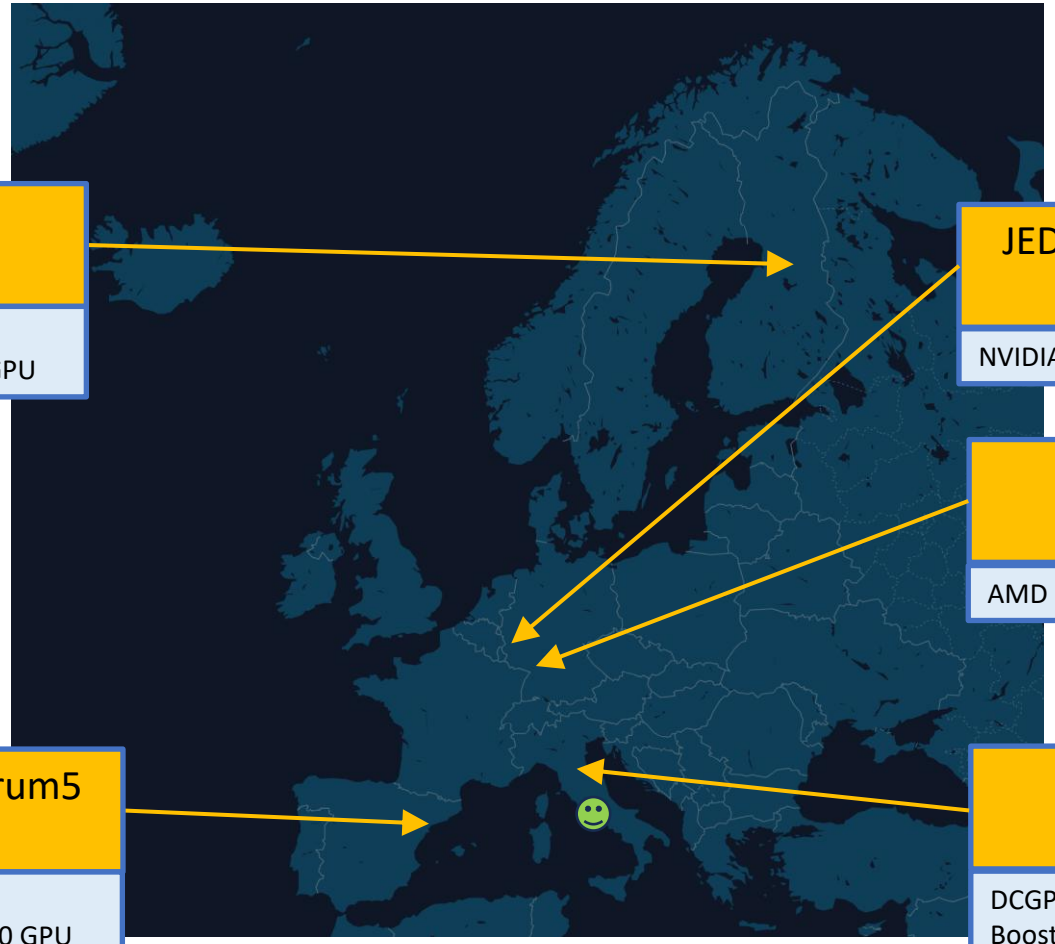
- How can *sutils* automatically generate all the backends?
  - STREAmS is based on object oriented code architecture
  - clear separation (different files) of backend-dependent and backend-independent parts
  - strict programming policies (not hard to follow after code development started)
- *sutils* is a in-house Python library
  - analyze code saving information (mostly) using Python dictionaries
  - process/replace/adapt code substantially (for HIP a C++ layer is created)
  - produces a perfectly readable code so that possible bugs can be investigated
- **major advantage**: minimize the effort of domain scientific experts during code development

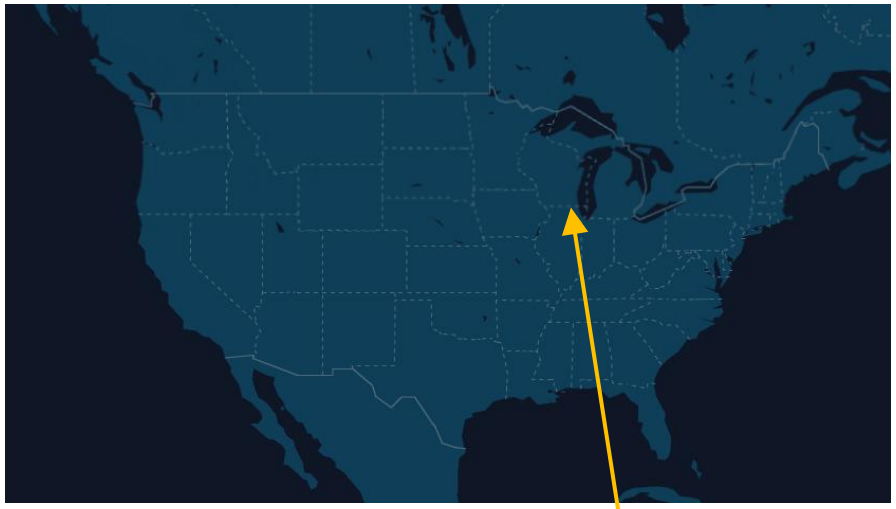
sutils / 2

- The core section of sutils employs Python Mako templating engine to produce the transformed code
  - OpenMP-offload kernel template
- sutils may need to be updated when major changes of the code are implemented
  - a relatively small update was needed to support STREAmS v2.1 curvilinear grids
  - It is possible to manually define an input file for sutils containing specific tuning for kernels, e.g., loop order, number of parallelized loops

```
% if kernel_type=="global":
% if local_arrays == True:
!$omp target data map(alloc:${",".join(larrays)})
% endif
<%
final_string=f"!$omp target teams distribute parallel do collapse({num_loop})
has_device_addr({','.join(gpu_arrays)}) {'private('+','.join(larrays)+')'
if local_arrays == True else ''} {'&' if is_reduction == True else ''}"
%>\
${final_string}
% if is_reduction == True:
% for redn_id,redn in enumerate(all_reductions):
% if len(all_reductions) > 1 and redn_id != len(all_reductions)-1:
!$omp& reduction(${redn[0]}:${redn[1]}) &
% else:
!$omp& reduction(${redn[0]}:${redn[1]})
% endif
% endfor
% endif
% elif kernel_type == "device":
!$omp declare target
% endif
% if kernel_type=="global":
% for idx in range(num_loop):
do ${index_list[idx]} = ${size[idx][0]},${size[idx][1]}
% endfor
% endif
${serial_part.strip()}
% if kernel_type=="global":
${'enddo\n'*num_loop}
% endif
% if kernel_type == "global" and local_arrays == True:
!$omp end target data
% endif
```

HPC pilgrimage





LUMI
CSC

C: AMD CPU
G: AMD MI250X GPU

JEDI (JUPITER)
JSC

NVIDIA GH200

HUNTER
HLRS

AMD MI300A APU

AURORA
ALCF

Intel Data Max 1550 GPU

MareNostrum5
BSC

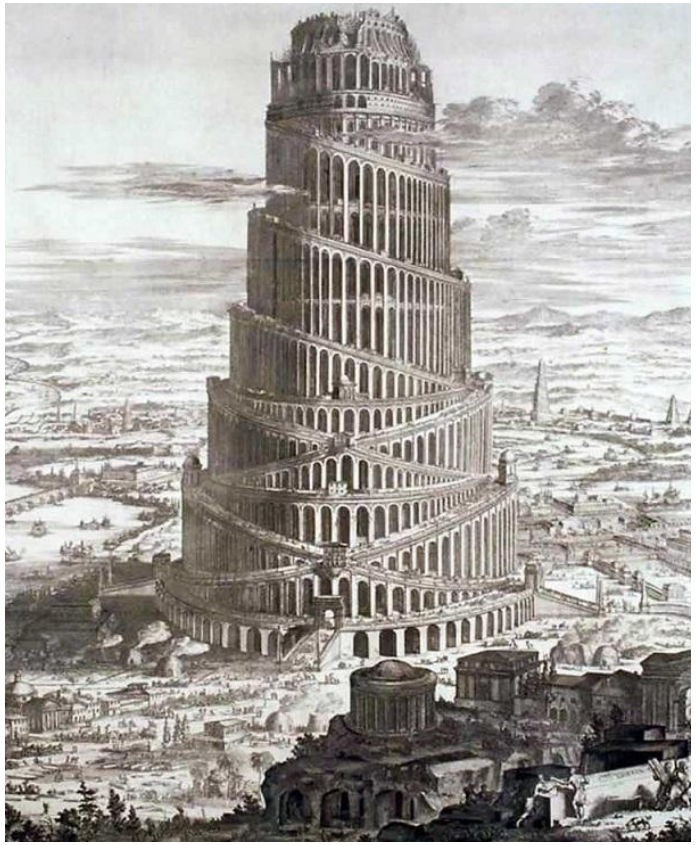
GP: Intel CPU
ACC: NVIDIA H100 GPU

Leonardo
CINECA

DCGP: Intel CPU
Booster: NVIDIA A100 GPU

Uniquely different

- Enumerating what is common among systems is much shorter than what is different
  - ssh access
  - module system
- Different:
  - hardware: devices (GPUs), network,...
  - authentication method: public key, password, smallstep,...
  - network limitations: VPN on access, cannot exit,...
  - software stack: cray, non-cray modules,...
  - scheduling and configuration: SLURM, PBS,...
  - project monitoring tools
  - getting computing time: EuroHPC, national projects,...



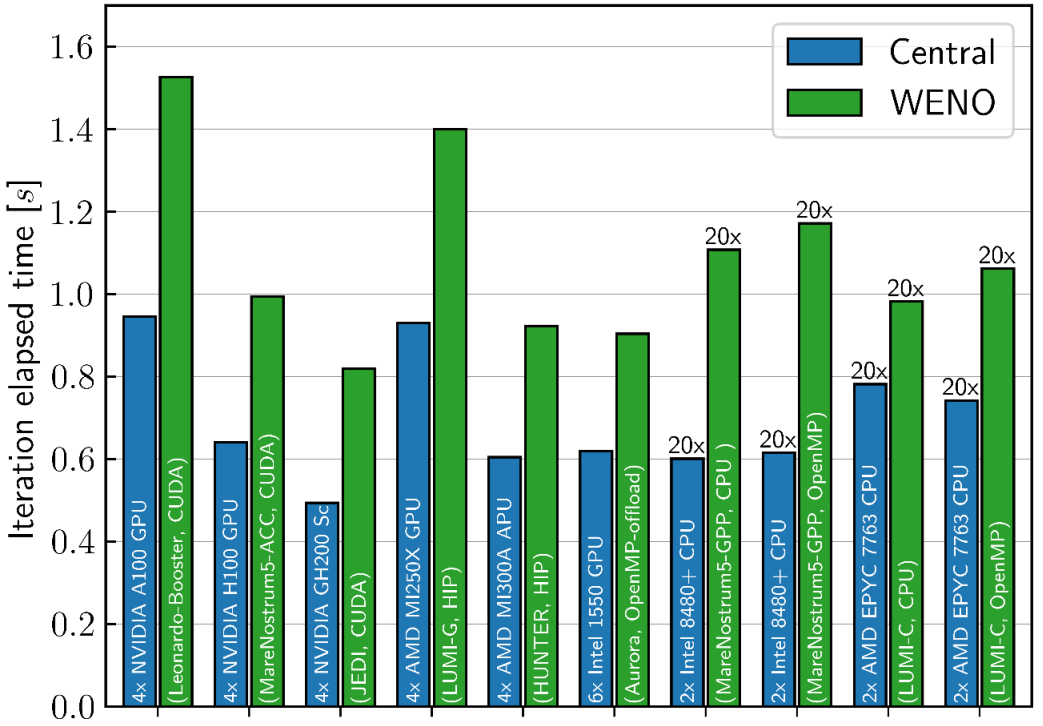
HPC systems and STREAmS paradigms

#	Cluster	Partition	Type	PU	#PU	Backend	Compiler	Ver	MPI	Ver	BW	FLOPs	Nodes
S1	Leonardo	Booster	GPU	NVIDIA A100 (SXM4 64 GB)	4	CUDA Fortran	NVIDIA	24.3	OpenMPI	4.1.6	1635 x 4	20 x 4	1024
S2	Marenostrum5	ACC	GPU	NVIDIA H100 (64GB HBM2)	4	CUDA Fortran	NVIDIA	24.5	OpenMPI	4.1.7	2000 x 4	26 x 4	64
S3	JEDI	-	Superchip	NVIDIA GH200 (96GB, 4TB/s)	4	CUDA Fortran	NVIDIA	25.1	OpenMPI	5.0.5	4000 x 4	34 x 4	32
S4	LUMI	G	GPU	AMD MI250X	4 (8 GCDs)	HIP	GNU/ROCm	13.2.1/6.0.3	Cray-MPICH	8.1.29	3200 x 4	48 x 4	2048
S5	Hunter	GPU	APU	AMD MI300A (128GB)	4	HIP	Flang/ROCm	18.0.0/6.2.2	Cray-MPICH	8.1.30	5300 x 4	61 x 4	64
S6	Aurora	-	GPU	Intel 1550 (128GB)	6 (12 Tiles)	OpenMP offload	Intel	2024.07.30.002	MPICH	4.3.0rc3	3277 x 6	52 x 6	2048
S7	MareNostrum5	GPP	CPU	Intel Xeon Platinum 8480p	2	CPU	Intel	2023.2.0	IntelMPI	2021.10.0	600	9	128
S8	MareNostrum5	GPP	CPU	Intel Xeon Platinum 8480p	2	OpenMP	Intel	2023.2.0	IntelMPI	2021.10.0	600	9	512
S9	LUMI	C	CPU	AMD EPYC 7763	2	CPU	GNU (Flang)	13.2.1 (17.0.0)	Cray-MPICH	8.1.29	410	5	1024
S10	LUMI	C	CPU	AMD EPYC 7763	2	OpenMP	GNU (Flang)	13.2.1 (17.0.0)	Cray-MPICH	8.1.29	410	5	1024

- STREAmS has predefined **`makefile`** configurations to easily prepare all these HPC environments

Reference benchmark case

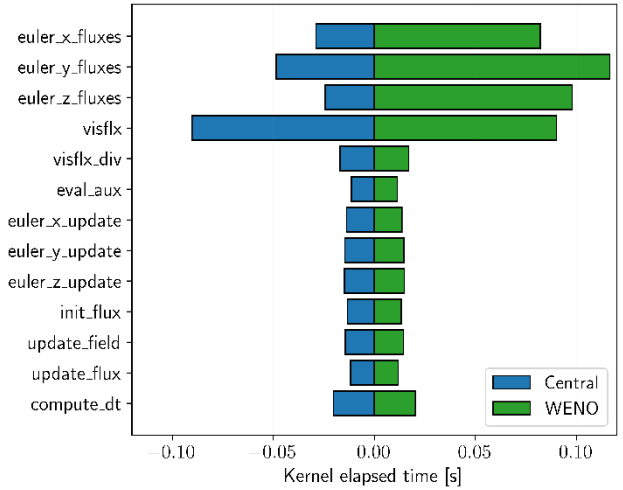
- Airfoil case (C-mesh)
- Computational grid:
  - 4096 x 286 x 276 ≈ 550M points
  - memory occupation around 180GB well below maximum values for recent systems: done in view of realistic **time-to-solutions** and to have a common case
- 8 systems compared:
  - 2 CPU based: Intel and AMD
    - pure MPI and MPI+OpenMP compared
  - 6 GPU based: NVIDIA, AMD, Intel
    - CUDA Fortran for NVIDIA, HIP for AMD, OpenMP-offload for Intel
- additional combinations could be addressed
  - OpenMP-offload currently under testing for AMD GPUs (paper submitted to IWOMP 2025)

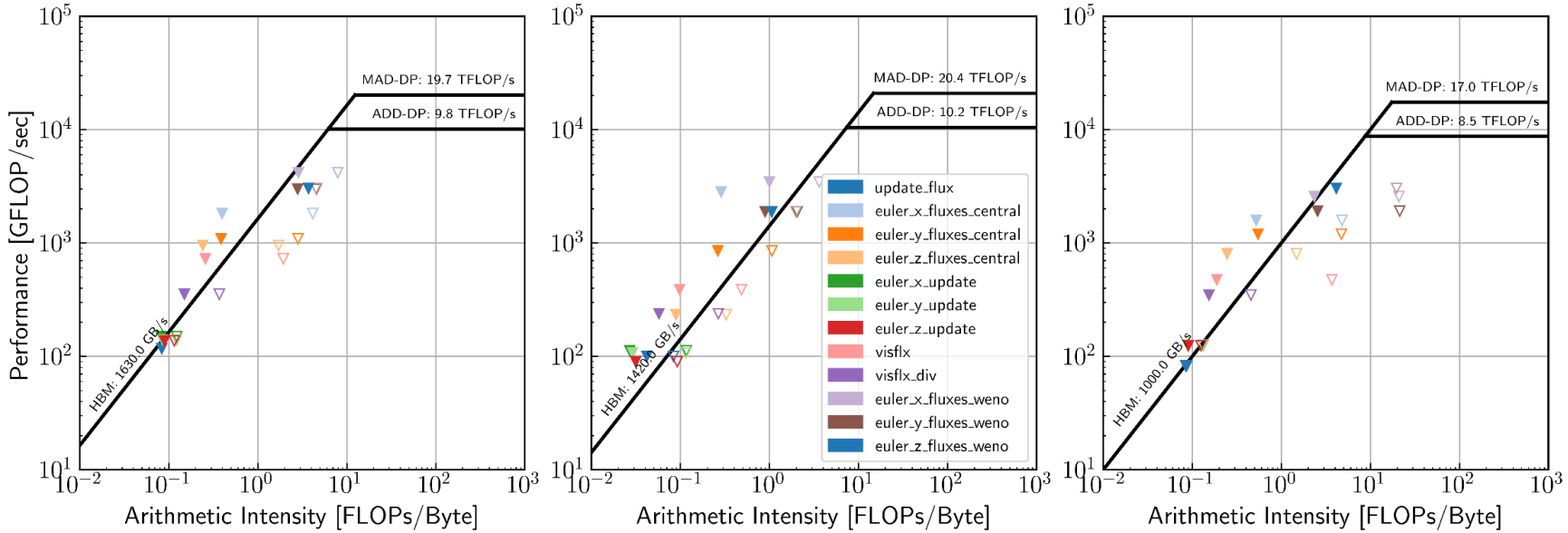


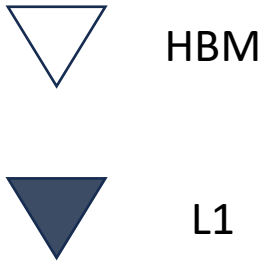
- Results mostly follow release dates from different vendors and (more loosely) device peak performance

Hierarchical (L1 and HBM) roofline analysis

- Performed on single GPU devices
  - NVIDIA A100, AMD MI250X, Intel 1550
  - Profilers: NVIDIA Nsight Compute, AMD Roc profiler, Intel Advisor
- Kernels mostly in the memory bound region
- Simple kernels already close to peak bandwidth
- Other kernels beyond HBM peaks thanks to L1 cache







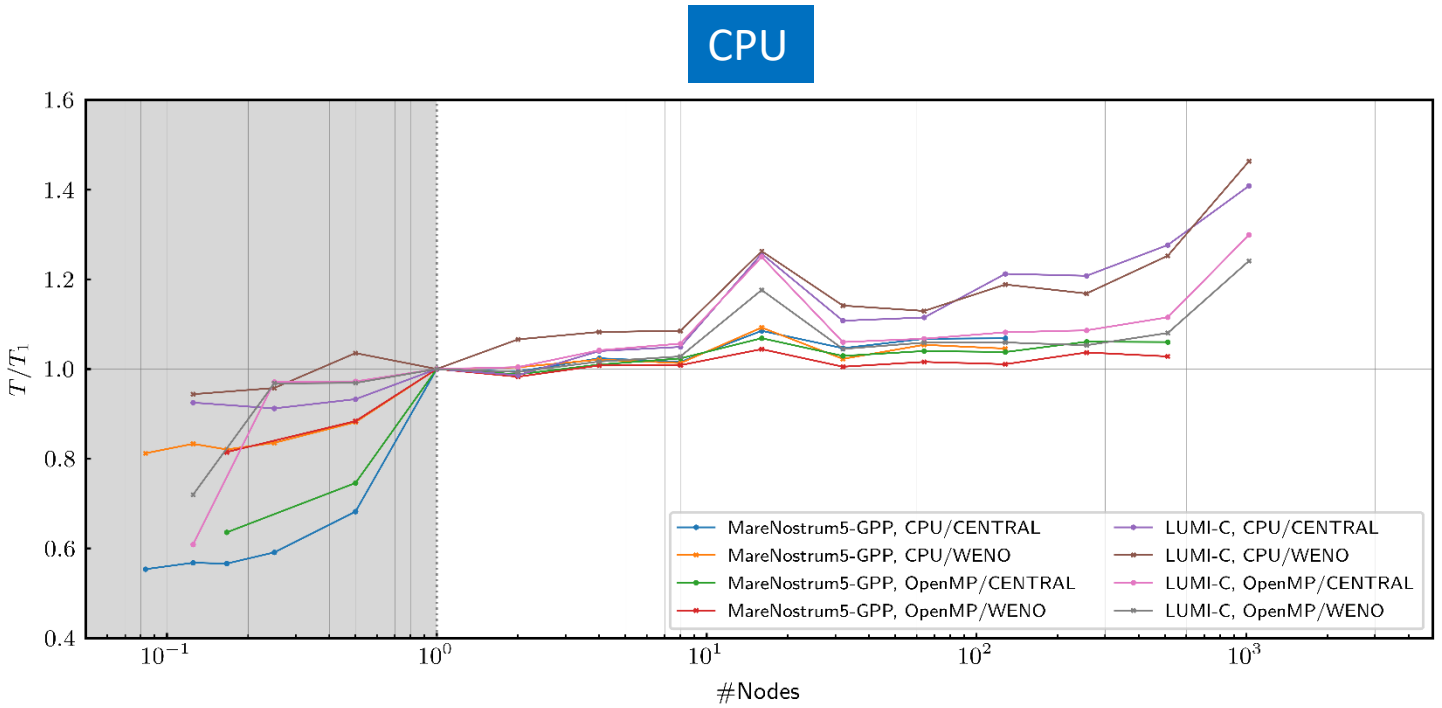
"Physical" scalability

- Performance measured in physically meaningful cases
  - For weak scaling, a set of C-mesh grids corresponding to increasingly higher Reynolds
  - the number of points in each direction is increased consistently with the needed refinement.
  - reference case is C12 and is associated to single-node case
  - for strong scaling C10 case grid is studied comparing 4, 8, 16, 32 nodes
- Finest mesh for Reynolds=6M case, realistic value for business jets
  - on the right, the green region includes 1000 wall-normal mesh points and the red region contains 1000x1000 grid points

#	Reynolds	Nx	Ny	Nz	Ntot [G]	Nodes
C1	5957609	65536	3731	4608	1126.724	2048
C2	4519941	57344	3198	3072	563.362	1024
C3	3427473	40960	2755	2496	281.661	512
C4	2597869	32768	2238	1920	140.803	256
C5	1968169	24576	1756	1632	70.430	128
C6	1489946	18432	1531	1248	35.218	64
C7	1127465	14336	1279	960	17.602	32
C8	852570	11264	1018	768	8.806	16
C9	644399	9216	829	576	4.401	8
C10	486531	7168	710	432	2.199	4
C11	367190	5632	582	336	1.101	2
C12	276904	4096	486	276	0.549	1
C13	208708	3200	398	216	0.275	1/2
C14	157213	2688	328	156	0.138	1/4
C15	118148	2048	280	120	0.069	1/8
C16	133094	2176	301	140	0.092	1/6
C17	99998	1792	237	108	0.046	1/12

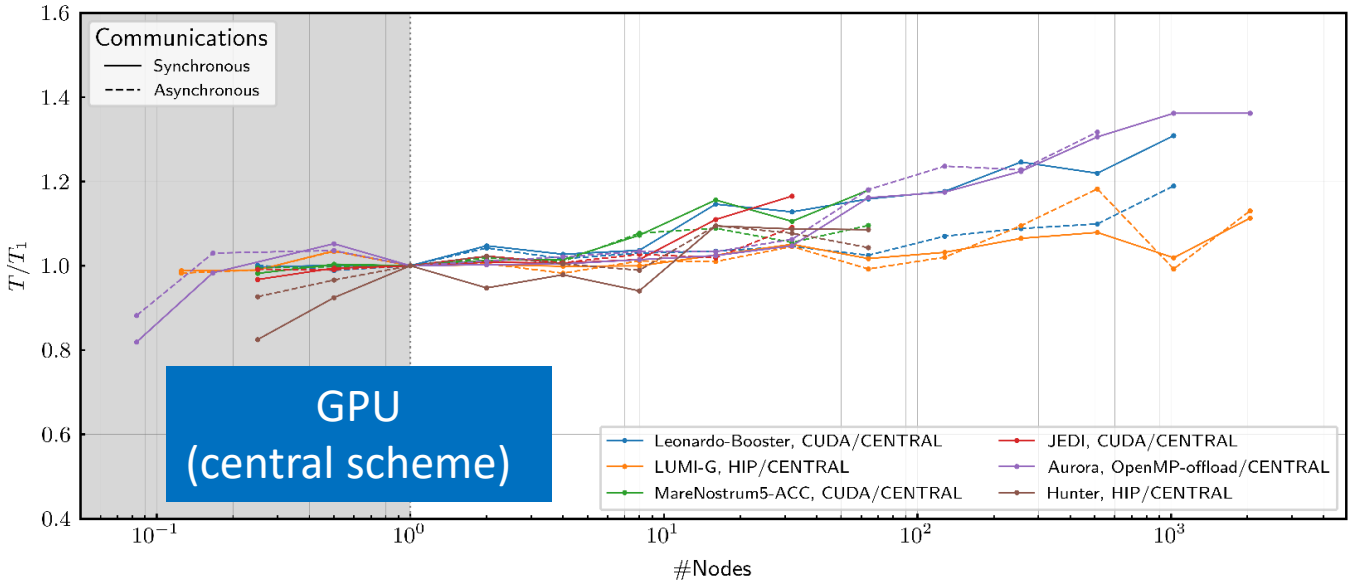
Weak scalings

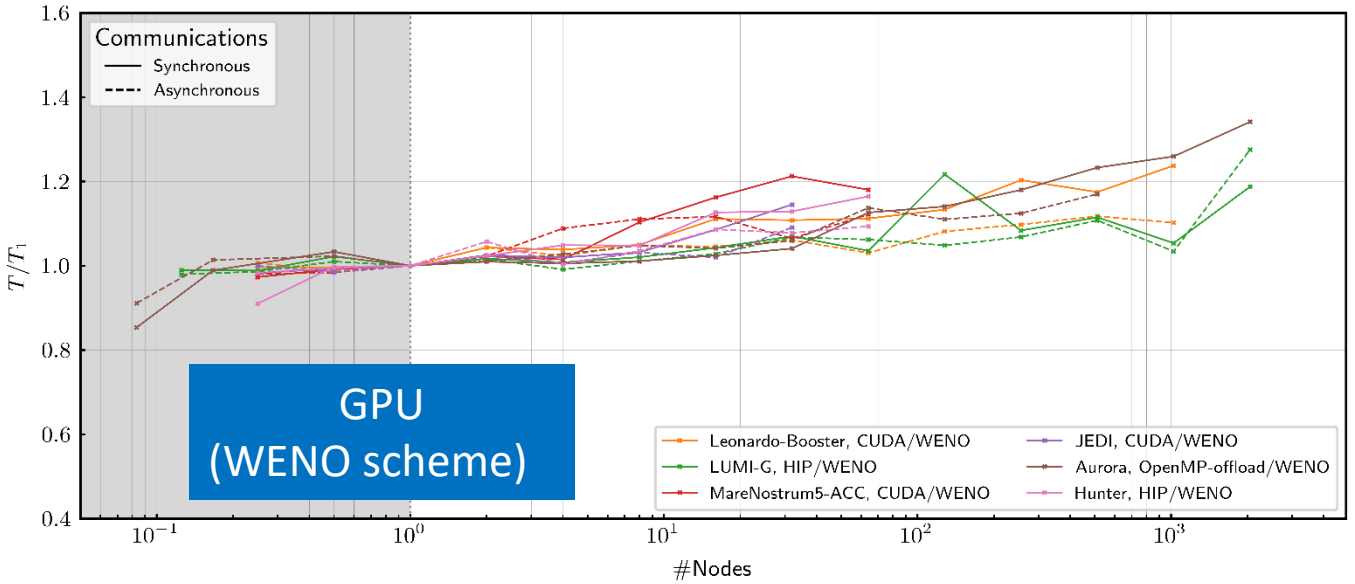
- Intra-node and inter-node scalings are good
  - intra-CPU scalings limited due to bandwidth usage
  - intra-GPU scaling are good except for Int GPUs where there is power capping
- Role of asynchrnous patterns is different for diverse GPU systems
  - always good for NVIDIA and very useful
- Memory occupation is well below the limits for recent GPU systems but this is to be closer to realistic time-to-solutions for high Reynolds



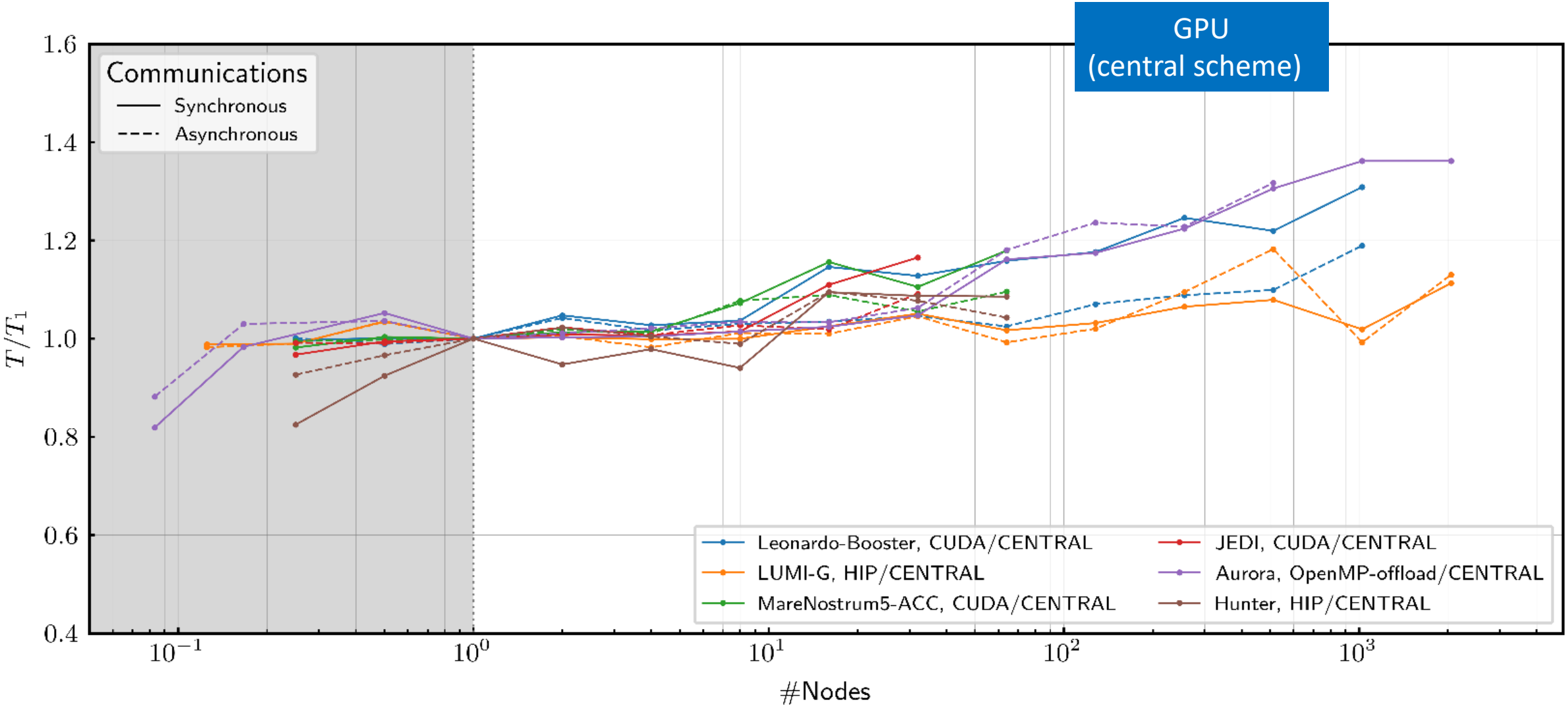
Weak scalings / 2

- Intra-node and inter-node scalings are good
  - intra-CPU scalings limited due to bandwidth usage
  - intra-GPU scaling are good except for Intel GPUs where there is power capping
- Role of asynchrnous patterns is different for diverse systems
  - always good for NVIDIA and very useful
- Memory occupation is well below the limits for recent GPU systems but this is to be closer to realistic time-to-solutions for high Reynolds





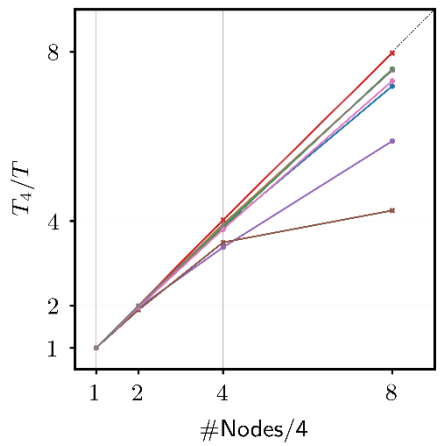
Weak scalings / 3

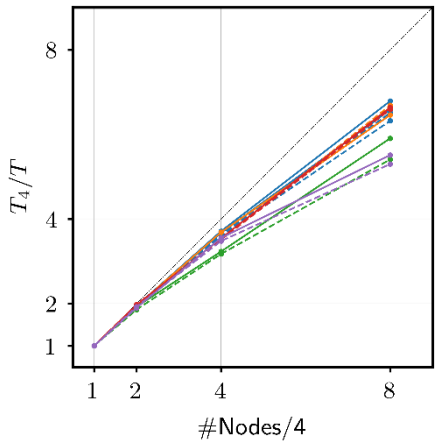


Strong scalings

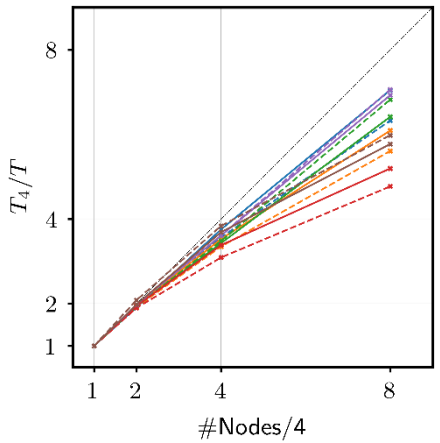
- Despite (much) higher decompositions CPU scalings mostly keep a good scaling
- For GPUs, scaling between 1 and 2 nodes is very good confirming that the reference case is not too small to adequately exploit GPU
- For higher decompositions, communication times limit scalability compared to the ideal one
- Time-to-solution is a challenge for next future
  - increasing grid refinement, reducing time step implies more iterations needed

CPU





GPU
(central scheme)



GPU
(WENO scheme)

Exascale readiness: pipeline

Different paths to accomplish the same task depending on the size of the problem

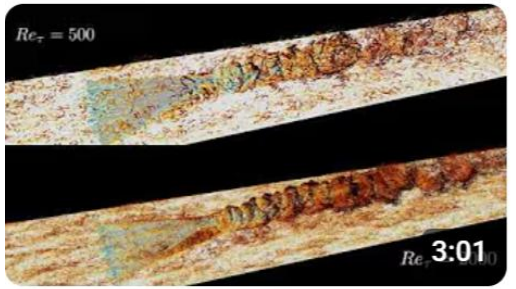
- some paths implemented for cases with more than 100 billions of cells

- **Workflow**
  - **Automated Workflow Preparation:** Initial mesh and field setup uses a preliminary RANS run; automation reduces user effort and improves reproducibility
- **Grid**
  - **Enhanced Grid Generation with Construct2D:** Improved C-mesh shaping near trailing edges and refined control of grid spacing in wall-normal/tangent directions.
  - **Support for Large Grid Restarts:** Construct2D now supports restart functionality for very large grids, improving robustness for long simulations.
  - **Alternative Grid Refinement Path:** Optional sparse grid generation with post-refinement allows scalability beyond EXCELLERAT needs.
- **Visualization**
  - **Efficient Visualization Options:** Slice-based plot3D output added; Catalyst2 enables in situ visualization during simulation runtime.
- **Statistics**
  - **Run-Time Statistical Averages:** Spanwise and time averages can be computed during runtime, reducing post-processing for large-scale runs.
  - **Run-Time Spectra with Welch Method:** Time-spectra calculation implemented in runtime mode with overlapping windowing strategy.
  - **Post-Processing Tools:** Tools for statistics and spectral analysis are complete; tailored functionalities developed for airfoil case studies.
- **Input/Output**
  - **Checkpointing:** Two checkpointing modes: MPI-I/O (single file) or per-process (preferred for large cases).
  - **Parallel Grid Input Management:** Fully parallel 2D grid handling and decomposition prior to simulation enable fast, distributed reads.
- **Continuous benchmarking**
  - **Basic pipeline:** implemented using GitLab+Jacamar+exaCB+JUBE
  - **Developed in-house tool TEBE:** to replace JUBE and simplify benchmarking automation

In situ visualization

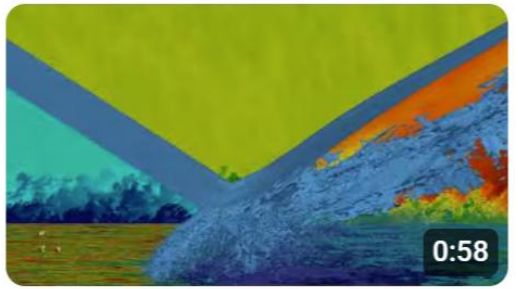
- STREAmS v2.1 implements in situ through Catalyst2
  - CONDUIT grid/field management
    - Dictionary defines memory/field layouts
    - Structured Grid mesh for Cartesian and curvilinear grids
- STREAmS-Catalyst2 from the user side
  - no STREAmS source code modification needed
  - just input fields, mainly variables to pass and frequency
  - Python pipeline script case-dependent
- APS Gallery of Fluid motion award (2022):
  - https://gfm.aps.org/meetings/dfd-2022/631f5e75199e4c2da9a94822
- Curvilinear cases under preparation, example available:
  - https://github.com/STREAmS-CFD/STREAmS-2/tree/main/examples/curvcha_moser

STREAmS YouTube channel:
https://www.youtube.com/@streamscfd6365



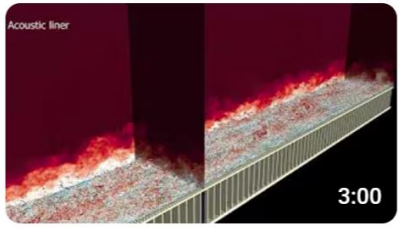
Supersonic turbulent boundary layer over a micro...

2.7K views • 2 years ago



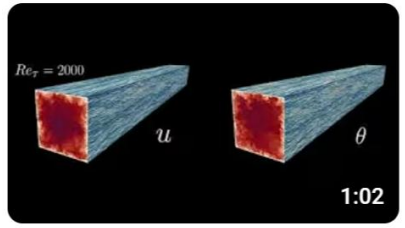
DNS of a shock wave/turbulent boundary-...

2K views • 2 years ago



Turbulent boundary layer over acoustic liners

921 views • 2 years ago



Forced thermal convection in square duct flow

162 views • 2 years ago

Conclusions

- STREAmS is a compressible fluid dynamics solver oriented to high-fidelity simulations of canonical cases
- Thanks to HPC oriented design of recent versions, 5 programming paradigms are implemented to fully exploit the diversity of HPC systems
- Portability is ensured by sutils library which allows the developer to keep their standard way of doing and periodically generate other backends
- Pipeline tools, including in situ visualization, oriented to exascale-size
- Extensive benchmarking performed on 6 clusters, 8 partitions, using a total of 10 backend combinations shows very good weak scaling up to 1 trillion points using 12K GPUs
- **Production runs ongoing on LUMI thanks to 1M node-hours awared by EuroHPC Extreme Call project**
- User guide (with references): https://streams-cfd.github.io/STREAmS-2/