NEKO: A Modern, Portable, and Scalable Framework for High-Fidelity Computational Fluid Dynamics

Niclas Jansson

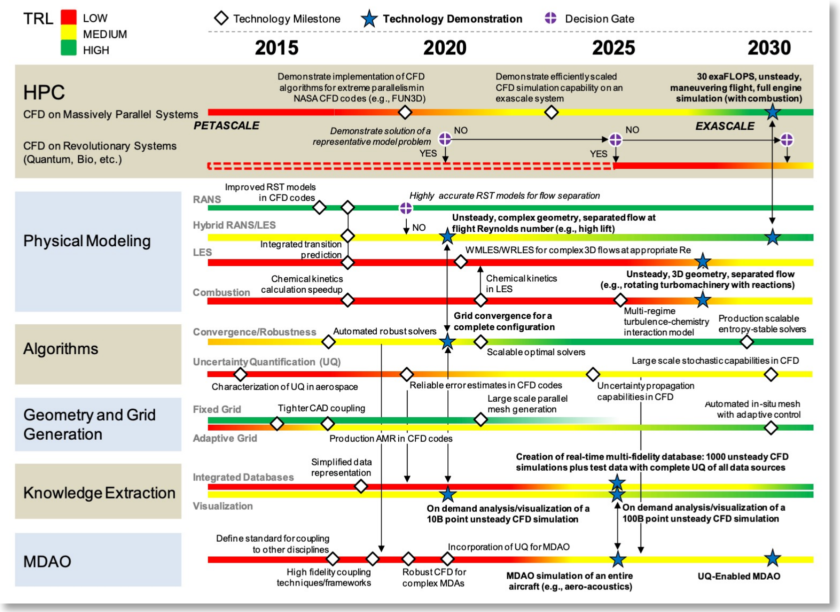
PDC Center for High Performance Computing
KTH Royal Institute of Technology

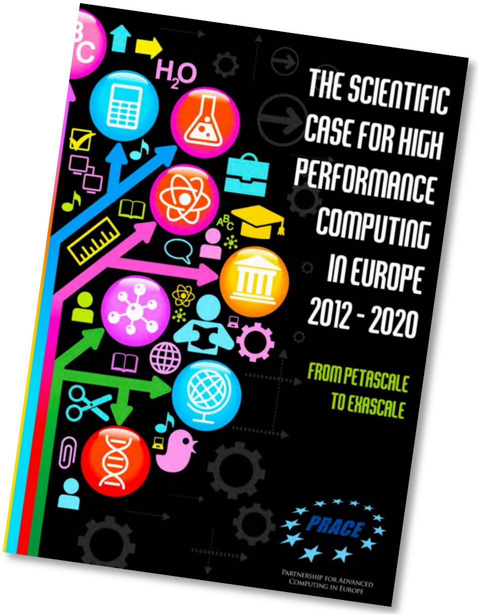
CEEC

Centre of Excellence in Exascale CFD

Introduction

About 10% of the energy use in the world is spent overcoming turbulent friction





No upper limit in fluid dynamics to the size of the systems to be studied via simulations

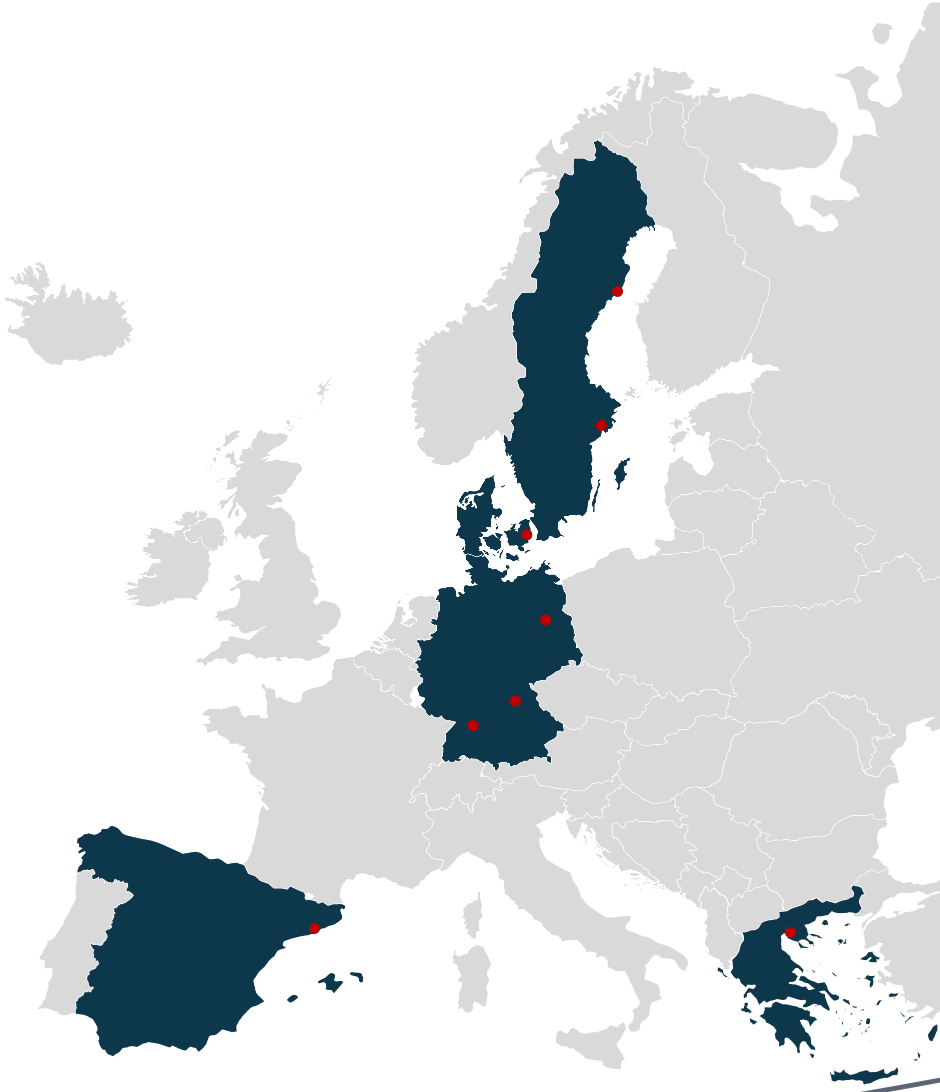
Computational Fluid Dynamics is one of the areas with a clear need and **great potential to reach exascale**

CEEC

Centre of Excellence in Exascale CFD

The main goal of CEEC is to address the extreme-scale computing challenge to enable the use of accurate and cost-efficient high fidelity computational fluid dynamics (CFD) simulations at exascale

- Implement **exascale-ready workflows** for addressing grand challenge scientific problems
- Develop **new or improved algorithms** that can efficiently exploit exascale systems.
- Significantly improve **energy efficiency** of simulations
- Demonstrate workflows on **lighthouse cases** relevant for both academia and industry



KTH VETENSKAP OCH KONST

Universität Stuttgart

FAU Friedrich-Alexander-Universität Erlangen-Nürnberg

DTU

BAM

BSC Barcelona Supercomputing Center Centro Nacional de Supercomputación

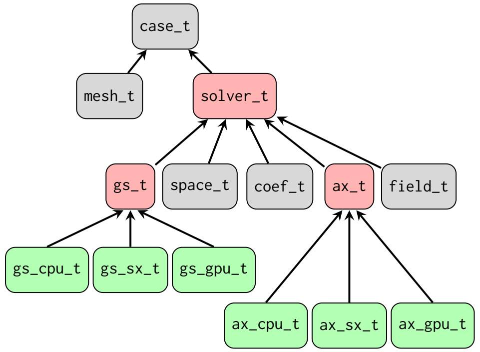
UMEÅ UNIVERSITET

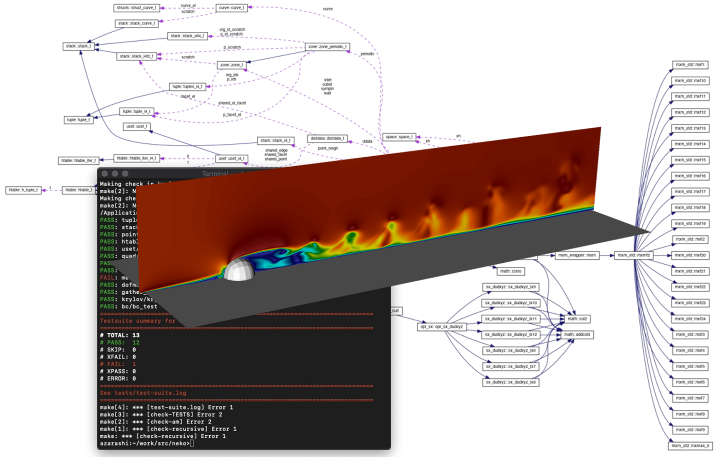
ceec-coe.eu

Portable Spectral Element Framework NEKO

- High-order spectral element flow solver
  - Incompressible Navier-Stokes equations
  - Matrix-free formulation, **small tensor products**
  - **Gather-scatter** operations between elements
- Modern **object-oriented** approach (Fortran 2008)

```fortran
! Base type for a matrix-vector product providing Ax
type, abstract :: ax_t
 contains
   procedure(ax_compute), nopass, deferred :: compute
end type ax_t

! Abstract interface for computing Ax
abstract interface
   subroutine ax_compute(w, u, coef, msh, Xh)
     implicit none
     type(space_t), intent(inout) :: Xh
     type(mesh_t),  intent(inout) :: msh
     type(coef_t),  intent(inout) :: coef
     real(kind=dp), intent(inout) :: w(:,:,:,:)
     real(kind=dp), intent(inout) :: u(:,:,:,:)
   end subroutine ax_compute
end interface
```

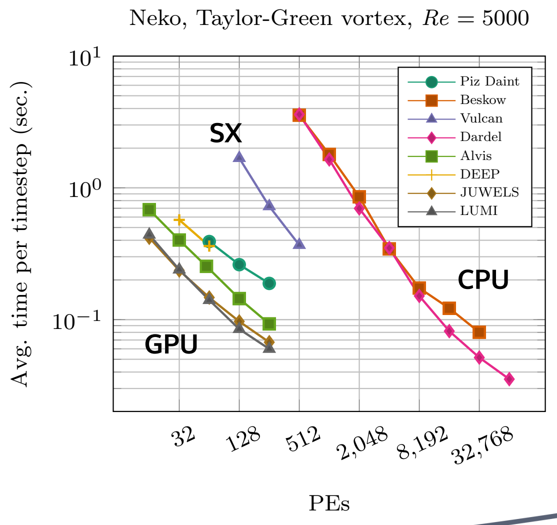




- Various hardware-backends
  - CPUs, GPUs down to exotic vector processors and FPGAs
    - **Device abstraction layer** for accelerators (CUDA/HIP/OpenCL)
- Modern Software Engineering (pFUnit, ReFrame, **Spack**)

```
> spack install neko+cuda
```

ExtremeFLOW/neko



www.neko.cfd

Device Abstraction Layer

CEEC

How to interface Fortran with accelerators?

- Native CUDA/HIP/OpenCL implementation via C-interfaces
- Device pointers in each derived type

```
type field_t
   real(kind=rp), allocatable :: x(:,:,:,:) !< Field data
   type(space_t), pointer :: Xh     !< Function space
   type(mesh_t), pointer :: msh     !< Mesh
   type(dofmap_t), pointer :: dof   !< Dofmap
   type(c_ptr) :: x_d = C_NULL_PTR !< Device pointer
end type field_t
```

- Abstraction layer hiding memory management
- Hash table associating x with x_d
- Kernels invoked from the object hierarchy via C interfaces (Ax, vector ops)
  - **Wrapper functions** for each supported accelerator backend
  - **Templated** (CUDA/HIP) or **pre-processor macros** (OpenCL) for runtime parameters
- **Auto/runtime tuning** based on polynomial order

```
src/
|
|-- math
|    `-- bcknd
|         |-- cpu
|         |-- device
|         |    |-- cuda
|         |    |-- hip
|         |    `-- opencl
|         |-- sx
|         `-- xsmm
```

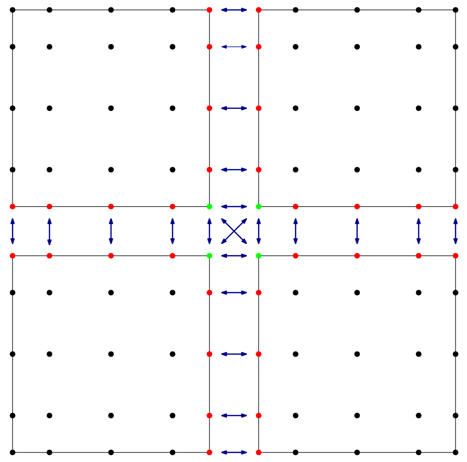
```
!> Enum @a hipError_t
enum, bind(c)
   enumerator :: hipSuccess = 0
   ...
end enum

!> Enum @a hipMemcpyKind
enum, bind(c)
   enumerator :: hipMemcpyHostToHost = 0
   enumerator :: hipMemcpyHostToDevice = 1
   ...
end enum

interface
   integer (c_int) function hipMalloc(ptr_d, s) &
        bind(c, name='hipMalloc')
     use, intrinsic :: iso_c_binding
     implicit none
     type(c_ptr) :: ptr_d
     integer(c_size_t), value :: s
   end function hipMalloc
end interface
```

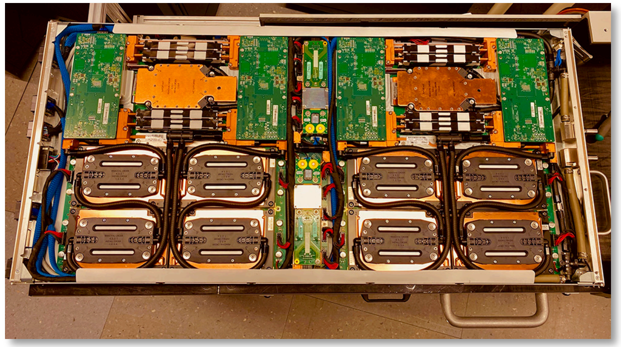
```
subroutine field_init(f,…)
type(field_t) :: f
...
call allocate(f%x(…,…,…,…,)
call device_alloc(f%x_d, size)
call device_associate(f%x, f%x_d)
```

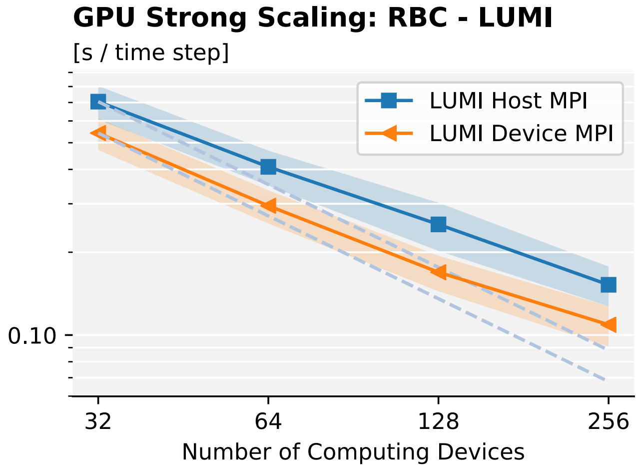
```
cudaMalloc    hipMalloc    clCreateBuffer
```

Gather-Scatter

- Uses indirect addressing and are (mostly) non-injective
- Topology aware optimisations
  - Facets (single neighbour), red points
    - Injective, **vectorizable** (always operating on **sorted** tuples)
  - Non facets (arbitrary number of neighbours), green points
    - **Cannot** be made injective, **not vectorizable** (small amount)
- Multiple levels of overlapping communication and computation
  - Overlapping with **non-blocking MPI** (device aware)
  - **Asynchronous** GPU kernels (neighbours in streams)
  - **Auto/runtime** tuning of all combinations

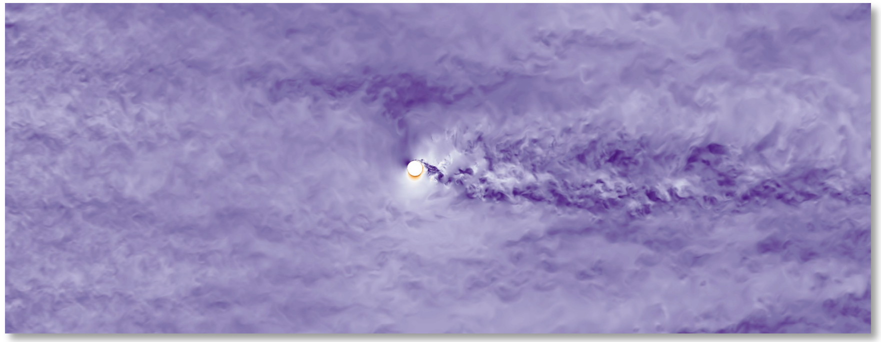


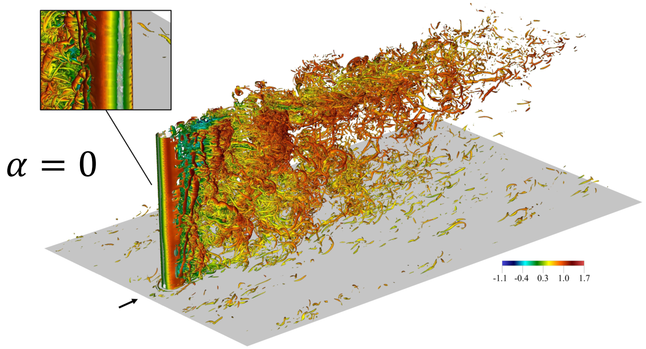


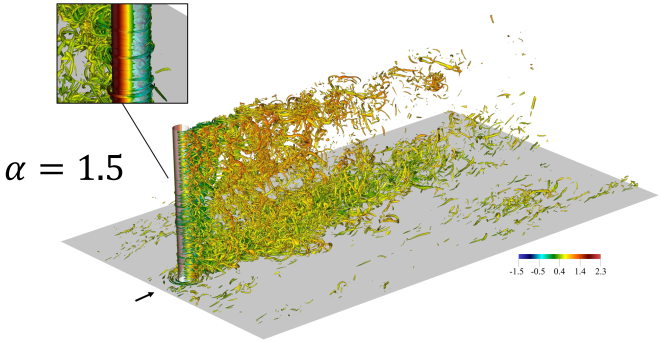


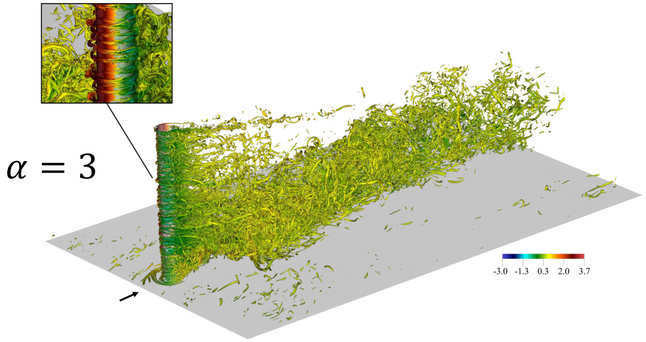
Large-scale DNS of turbulence with applications in sustainable shipping

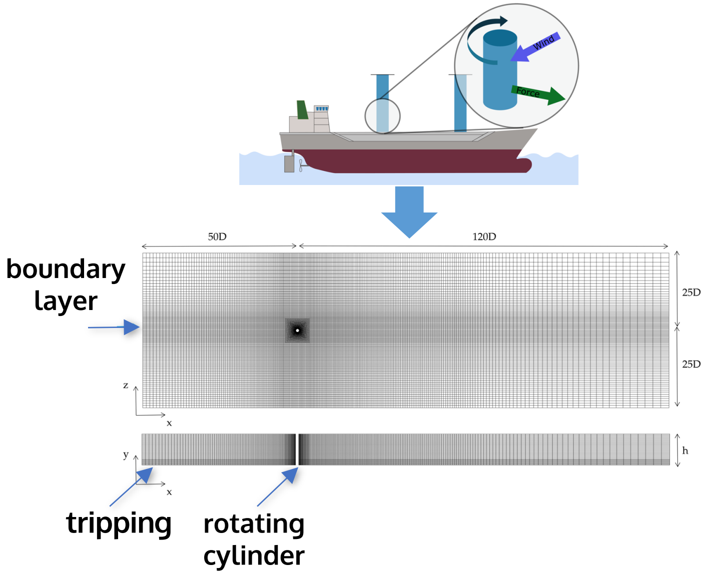
- DNS of the flow around a Flettner rotor at $Re_D = 3000$ in a turbulent boundary layer, for three different spinning ratios α
- **Less than two days** on LUMI-G (> two weeks on LUMI-C)

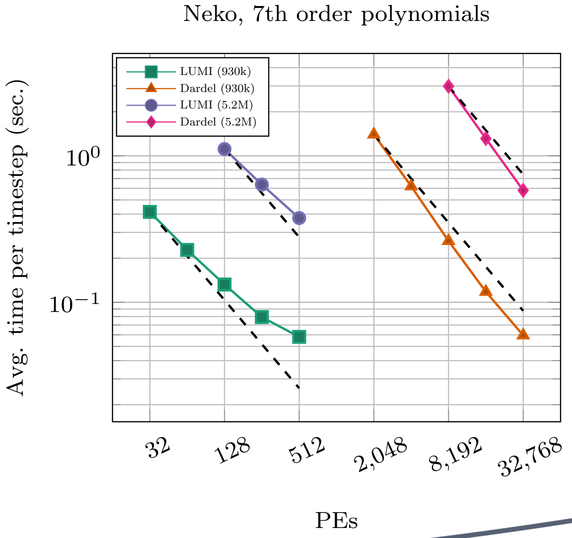












Numerical Method $P_N - P_N$

- Time integration is performed using an implicit-explicit scheme (BDFk/EXTk)

$$\sum_{j=0}^{k} \frac{b_j}{dt} u^{n-j} = -\nabla p^n + \frac{1}{Re} \nabla^2 u^n + \sum_{j=1}^{k} a_j \left(u^{n-j} \cdot \nabla u^{n-j} + f^n \right)$$

with b_k and a_k coefficients of the implicit-explicit scheme, solving at time-step n

$$\Delta p^n = \nabla \cdot \left(\sum_{j=1}^{k} a_j \left(u^{n-j} \cdot \nabla u^{n-j} + f^n \right) \right)$$

$$\frac{1}{Re} \Delta u^n - \frac{b_0}{dt} u^n = \nabla p^n + \sum_{j=1}^{k} \left(\frac{b_j}{dt} u^{n-j} + a_j \left(u^{n-j} \cdot \nabla u^{n-j} + f^n \right) \right)$$

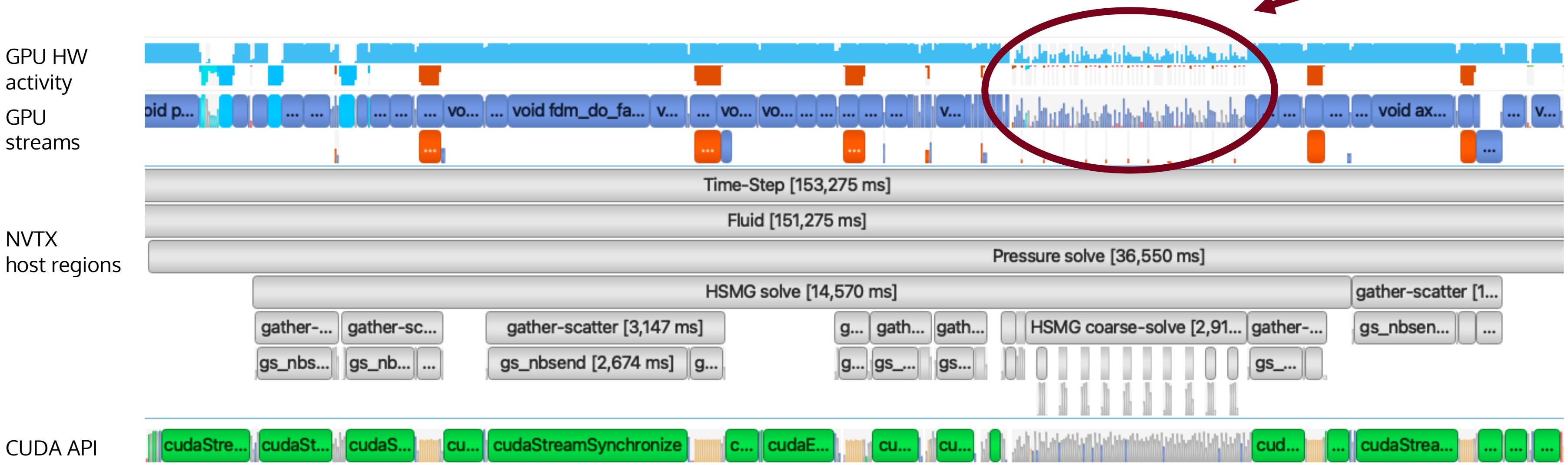
- Three velocity solves using CG with block Jacobi preconditioner (**fast**)
- One Pressure solve using GMRES with an additive overlapping Schwarz preconditioner (**expensive**)

$M_0^{-1} = \underbrace{R_0^T A_0^{-1} R_0}_{\text{Coarse grid (linear elements)}} + \sum_{k=1}^{K} R_k^T \tilde{A}_k^{-1} R_k$, key is to have a **scalable coarse grid solver**

Additiver Schwarz Preconditioner on GPUs

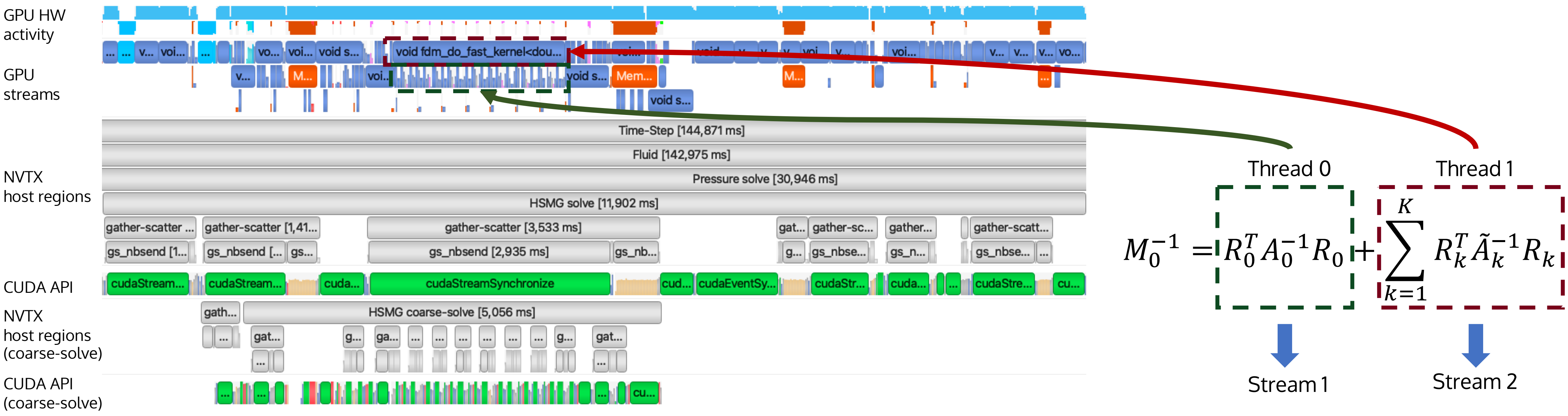
- Coarse grid solved using an approximate Krylov solver
  - Preconditioned Pipelined Conjugate Gradient with a low, maximum iteration limit
- Low computational efficiency on GPUs
  - A_0 is on linear elements, too little data to keep the GPU busy.
  - Many small kernels, dominated by kernel launch latency

$$M_0^{-1} = R_0^T A_0^{-1} R_0 + \sum_{k=1}^{K} R_k^T \tilde{A}_k^{-1} R_k$$



Task-decomposed Overlapped Preconditioner

- Exploit available **task-parallelism**
  - Launch the left and right part of M_0^{-1} in parallel on the device
  - Launch independent work in parallel from **different threads** in an OpenMP region
  - Launch tasks in **separate streams** to allow overlap and increase GPU utilization
  - Maximise kernel overlap using **stream priority** to ensure progress in both stream

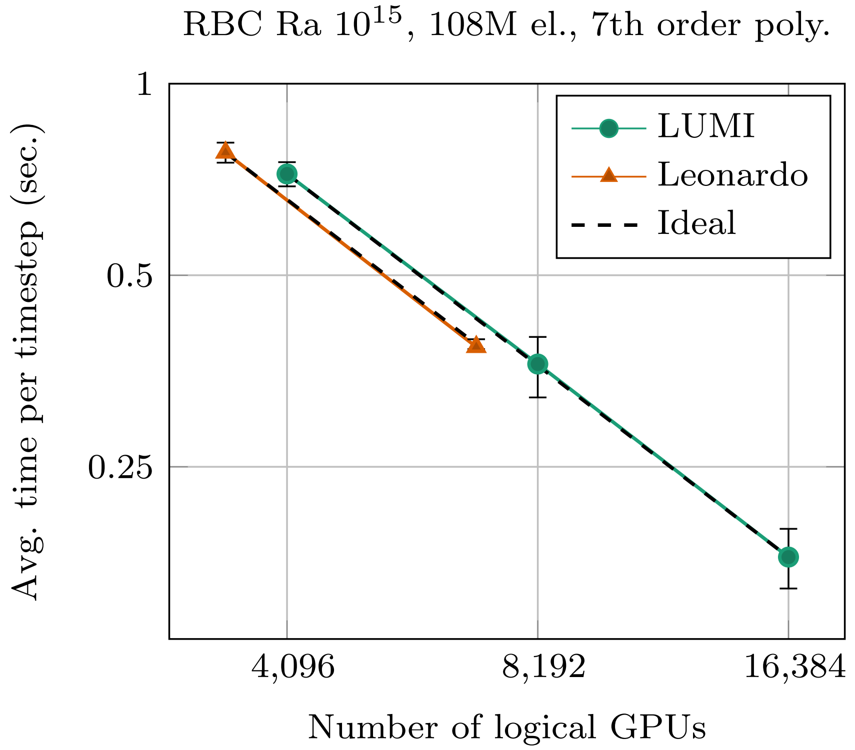


Extreme-scale High-Fidelity Simulations of Turbulent thermal convection

CEEC

- Exploring the ultimate regime of turbulent Rayleigh-Bénard convection
- Performance measurements on two of the EuroHPC-JU pre-exascale supercomputers **LUMI** and **Leonardo**
- Close to perfect parallel efficiency with less than 7000 elements per logical GPU
- Significantly reducing the smallest required problem size for strong scalability limits
- Improvements mainly due to the new overlapped pressure preconditioner

ACM Gordon Bell Prize Finalist 2023



99% confidence intervals is illustrated as error bars

MAX PLANCK COMPUTING & DATA FACILITY
MPCDF
FAU Friedrich-Alexander-Universität Erlangen-Nürnberg
TECHNISCHE UNIVERSITÄT ILMENAU
KTH VETENSKAP OCH KONST

Summary

- Computational Fluid Dynamics is one of the areas with a clear need **and great potential to reach exascale**
- High-order methods are essential on current HPC machines
  - **Better suited for current hardware**, improved accuracy for "free"
- The heterogenous HPC landscape is a nightmare
  - Find a suitable level of abstraction
  - Use the best tools, **mix languages and programming models**
- Modern software engineering approaches to ensure portability
  - **Verification & validation**
- Exploit all the **available concurrency** of the application
  - Key ingredient to achieve good strong scalability on LUMI and Leonardo

Funded by the European Union. This work has received funding from the European High Performance Computing Joint Undertaking (JU) and Sweden, Germany, Spain, Greece, and Denmark under grant agreement No 101093393.

Co-funded by the European Union

EuroHPC Joint Undertaking

Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European High Performance Computing Joint Undertaking (JU) and Sweden, Germany, Spain, Greece, and Denmark. Neither the European Union nor the granting authority can be held responsible for them.

CEEC

Centre of Excellence in Exascale CFD