Task-decomposed Overlapped Pressure Preconditioner for Sustained Strong Scalability on Accelerated Exascale Systems

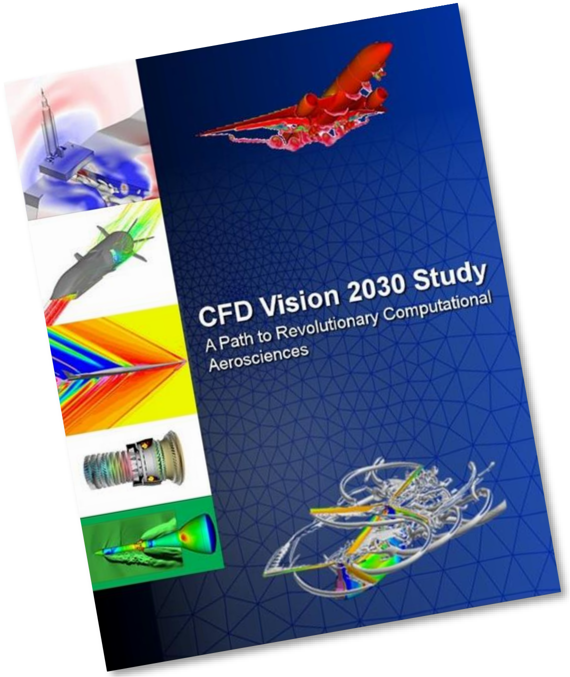
Niclas Jansson[1], Martin Karp[1], Szilárd Páll[1], Philipp Schlatter[2,1], Stefano Markidis[1]

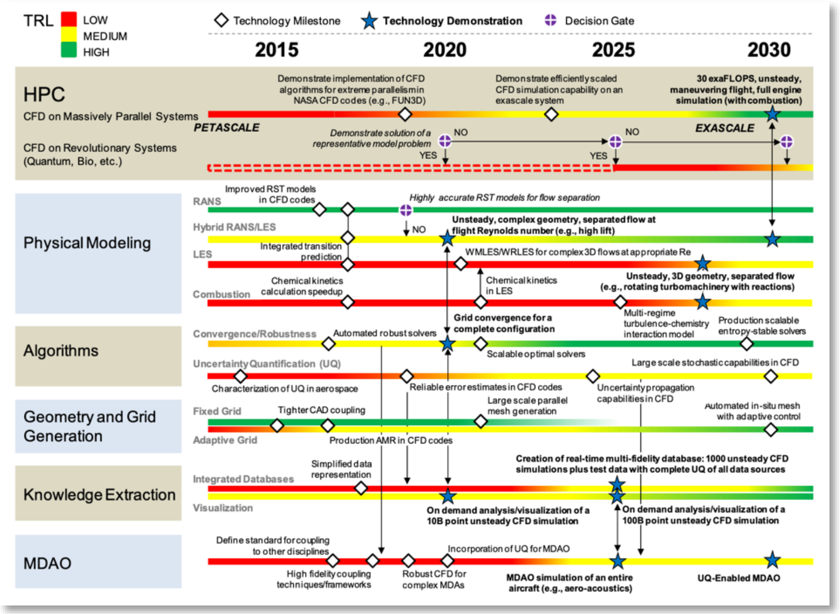
[1]KTH Royal Institute of Technology,
[2]Friedrich-Alexander-Universität Erlangen-Nürnberg

Introduction

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

THE SCIENTIFIC CASE FOR HIGH PERFORMANCE COMPUTING IN EUROPE 2012 - 2020
FROM PETASCALE TO EXASCALE
PRACE
PARTNERSHIP FOR ADVANCED COMPUTING IN EUROPE




TRL
LOW
MEDIUM
HIGH
Technology Milestone
Technology Demonstration
Decision Gate
2015
2020
2025
2030
HPC
CFD on Massively Parallel Systems
CFD on Revolutionary Systems (Quantum, Bio, etc.)
PETASCALE
EXASCALE
Demonstrate implementation of CFD algorithms for extreme parallelism in NASA CFD codes (e.g., FUN3D)
Demonstrate efficiently scaled CFD simulation capability on an exascale system
30 exaFLOPS, unsteady, maneuvering flight, full engine simulation (with combustion)
Demonstrate solution of a representative model problem
NO
YES
NO
YES
Physical Modeling
RANS
Hybrid RANS/LES
LES
Combustion
Improved RST models in CFD codes
Highly accurate RST models for flow separation
NO
Unsteady, complex geometry, separated flow at flight Reynolds number (e.g., high lift)
Integrated transition prediction
WMLES/WRLES for complex 3D flows at appropriate Re
Chemical kinetics calculation speedup
Chemical kinetics in LES
Unsteady, 3D geometry, separated flow (e.g., rotating turbomachinery with reactions)
Multi-regime turbulence-chemistry interaction model
Algorithms
Convergence/Robustness
Automated robust solvers
Grid convergence for a complete configuration
Scalable optimal solvers
Production scalable entropy-stable solvers
Uncertainty Quantification (UQ)
Characterization of UQ in aerospace
Reliable error estimates in CFD codes
Uncertainty propagation capabilities in CFD
Large scale stochastic capabilities in CFD
Geometry and Grid Generation
Fixed Grid
Adaptive Grid
Tighter CAD coupling
Large scale parallel mesh generation
Automated in-situ mesh with adaptive control
Production AMR in CFD codes
Knowledge Extraction
Integrated Databases
Visualization
Simplified data representation
Creation of real-time multi-fidelity database: 1000 unsteady CFD simulations plus test data with complete UQ of all data sources
On demand analysis/visualization of a 10B point unsteady CFD simulation
On demand analysis/visualization of a 100B point unsteady CFD simulation
MDAO
Define standard for coupling to other disciplines
High fidelity coupling techniques/frameworks
Robust CFD for complex MDAs
Incorporation of UQ for MDAO
MDAO simulation of an entire aircraft (e.g., aero-acoustics)
UQ-Enabled MDAO

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

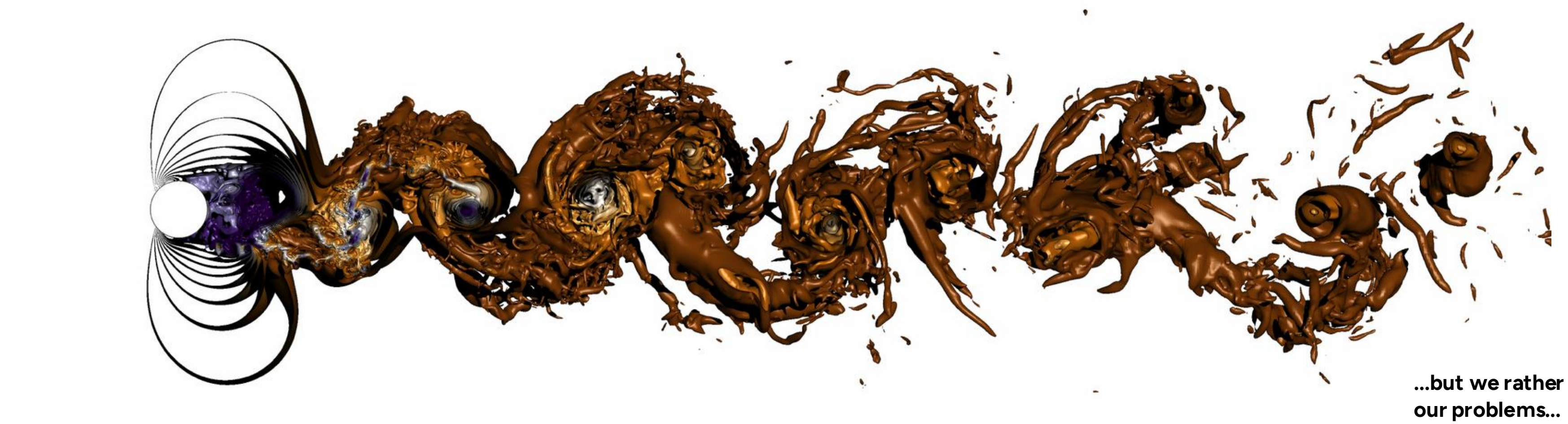
Computational Fluid Dynamics is one of the areas with a clear need and **great potential to reach exascale**

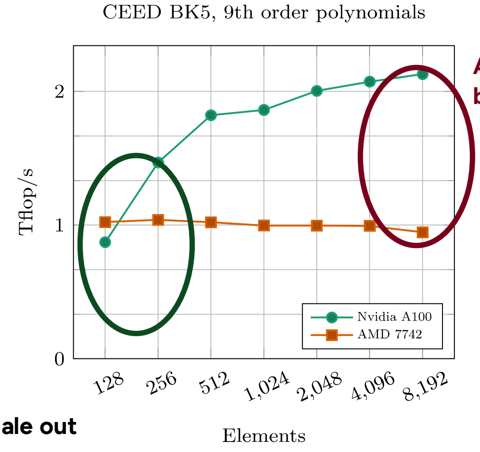
Introduction

- Exascale will require either **unreasonably large problem** sizes or **significantly improved efficiency** of current methods
  - Finite-Volume LES of a full car on the entire K computer (京) required **more than 100 billion grid points** to run efficiently
  - What problem size is needed to fill the 379 PFlop/s LUMI…
- High-order methods
  - Attractive numerical properties, **small dispersion** errors and more “accuracy” per degree of freedom
  - Better suited to take advantage of **modern hardware** (accelerators)

京: 82944 nodes, 663552 Cores, **10 PFlop/s**

Dardel: 56 nodes, 448 MI250X GCDs, ≈**10 PFlop/s**





Accelerators works best with a lot of data!

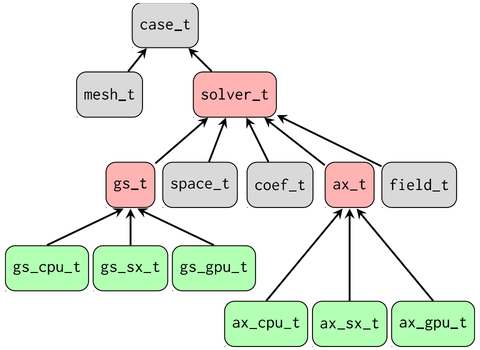
…but we rather scale out our problems…

Portable Spectral Element Framework NEKO

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

```
! 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
```

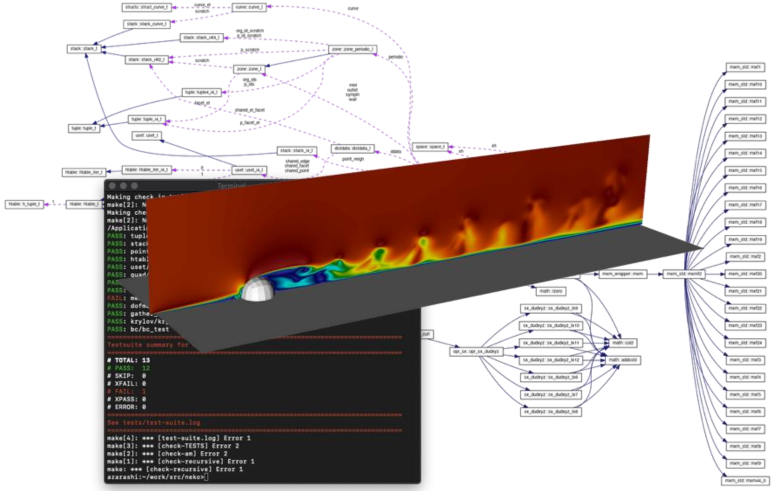


- Various hardware-backends
  - CPUs, GPUs down to exotic vector processors and FPGAs
    - **Device abstraction layer** for accelerators (CUDA/HIP/OpenCL)
  - Modern software engineering (pFUnit, ReFrame, **Spack**)

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

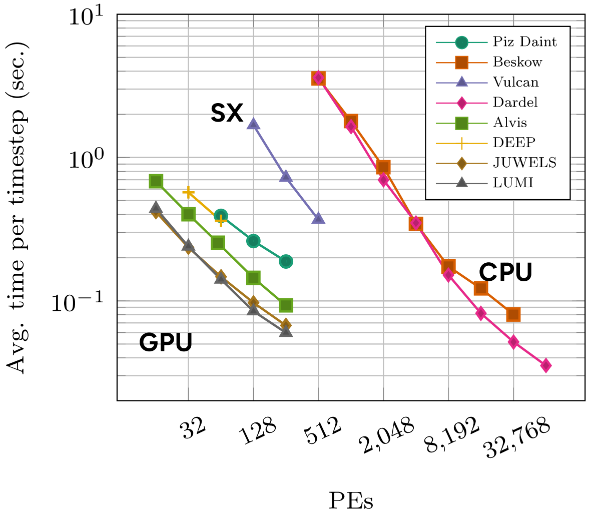
ExtremeFLOW/neko

www.neko.cfd



```
Making check in
make[2]: N
Making che
make[2]: N
/Applicati
PASS: tupl
PASS: stack
PASS: point
PASS: htabl
PASS: uset/
PASS: quad/
PAS
PASS:
FAIL: me
PASS: dofm
PASS: gathe
PASS: krylov/kr
PASS: bc/bc_test
============================================================================
Testsuite summary for
============================================================================
# TOTAL: 13
# PASS:  12
# SKIP:  0
# XFAIL: 0
# FAIL:  1
# XPASS: 0
# ERROR: 0
============================================================================
See tests/test-suite.log
============================================================================
make[4]: *** [test-suite.log] Error 1
make[3]: *** [check-TESTS] Error 2
make[2]: *** [check-am] Error 2
make[1]: *** [check-recursive] Error 1
make: *** [check-recursive] Error 1
azarashi:~/work/src/neko>
```

Neko, Taylor-Green vortex, $Re = 5000$



Device Abstraction Layer

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
```

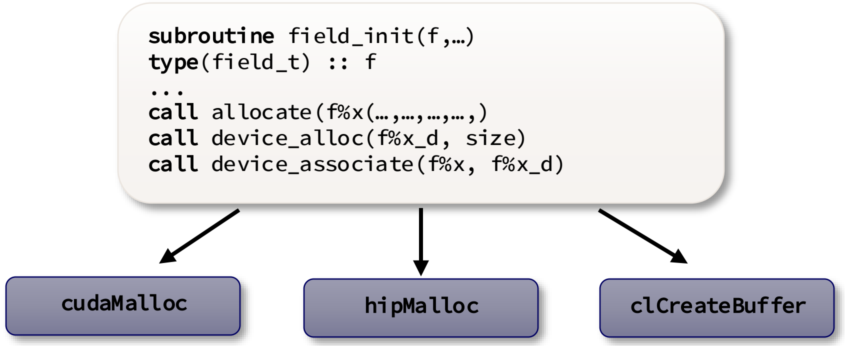
- Abstraction layer hiding memory management
- Hash table associating x with x_d
- Kernels invoked from the object hierarchy via C interfaces (Ax, vector ops)
  - **Wrapper functions** for each supported accelerator backend
  - **Templated** (CUDA/HIP) or **pre-processor macros** (OpenCL) for runtime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

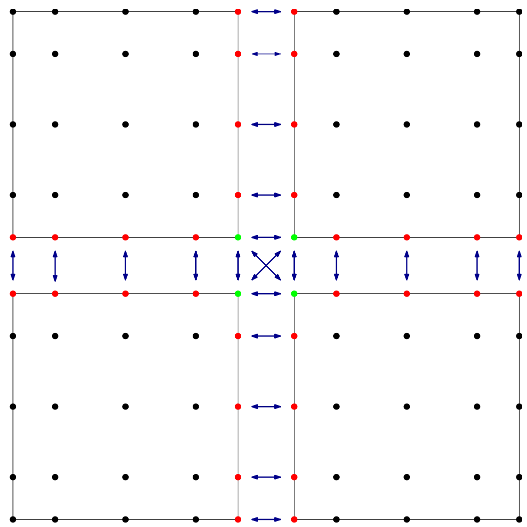
cudaMalloc

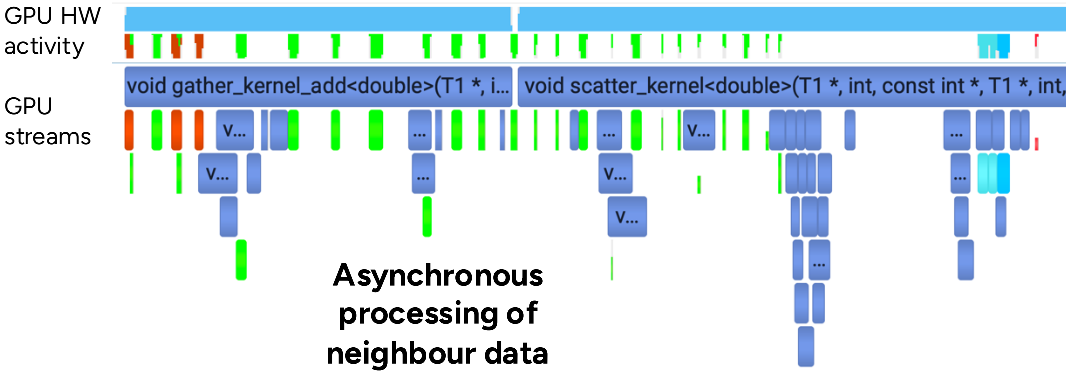
hipMalloc

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

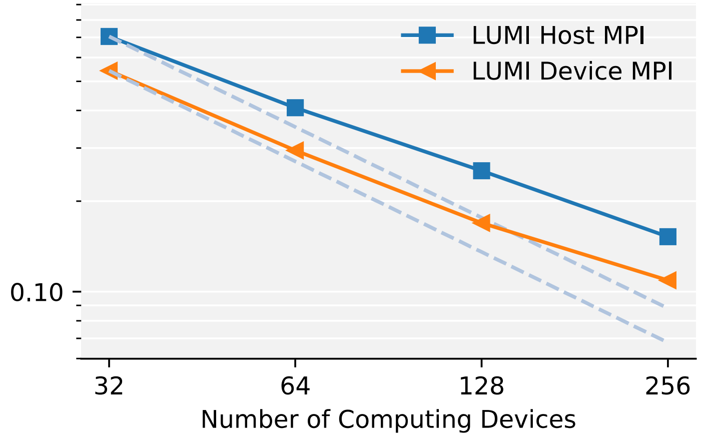



GPU HW activity
GPU streams
void gather_kernel_add<double>(T1 *, i...
void scatter_kernel<double>(T1 *, int, const int *, T1 *, int,
v...
...

Asynchronous processing of neighbour data

GPU Strong Scaling: RBC - LUMI

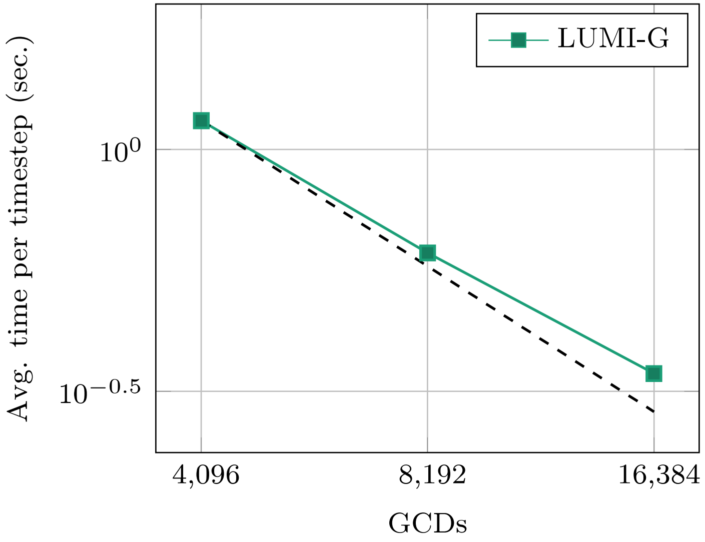
[s / time step]



Performance Baseline

- Full machine runs towards the end of the LUMI-G pilot phase
- DNS of flow past a circular cylinder at $Re = 50{,}000$
  - 113M elements
  - 7th order polynomials (8 GLL points)
- Simulation restarted from prebaked low-order runs
  - Restart checkpoint: 453GB
  - Extrapolated to 7th order polynomials
  - Computed solution (snapshot): 1.5TB
- Preliminary results
  - Achieved close to 80% parallel efficiency
  - Using 20%, 40% and 80% of the entire machine

Cylinder Re 50k, 113M el., 7th order poly.



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**)

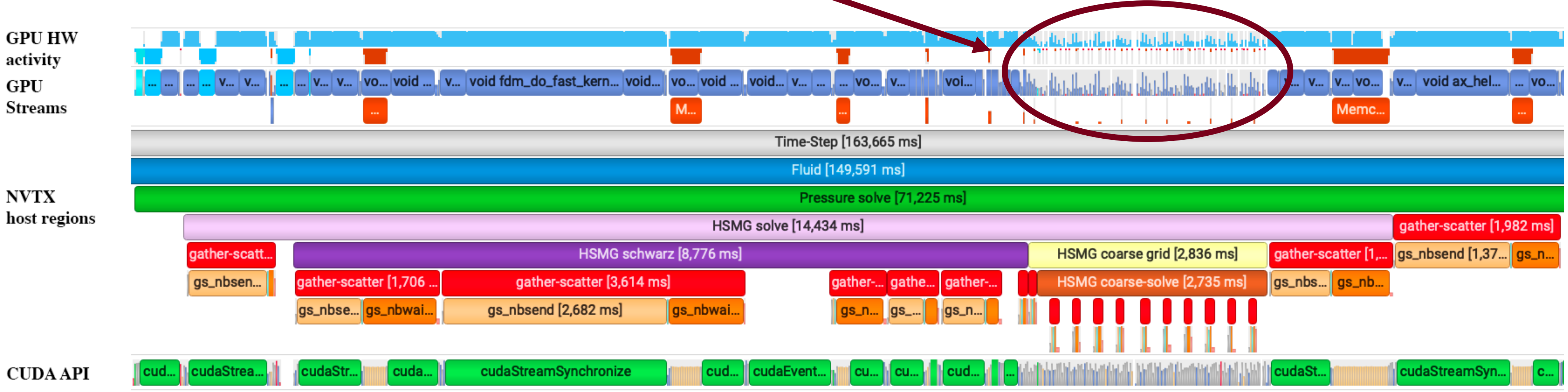
$M_0^{-1} = \underbrace{R_0^T A_0^{-1} R_0}_{\text{Coarse grid (linear elements)}} + \sum_{k=1}^{K} R_k^T \tilde{A}_k^{-1} R_k$, key is to have a **scalable coarse grid solver**

[1] G.E. Karniadakis, M. Israeli, S.A. Orszag, High-order splitting methods for the incompressible Navier-Stokes equations, J. Comput Phys, 1991

Additive Schwarz Preconditioner on GPUs

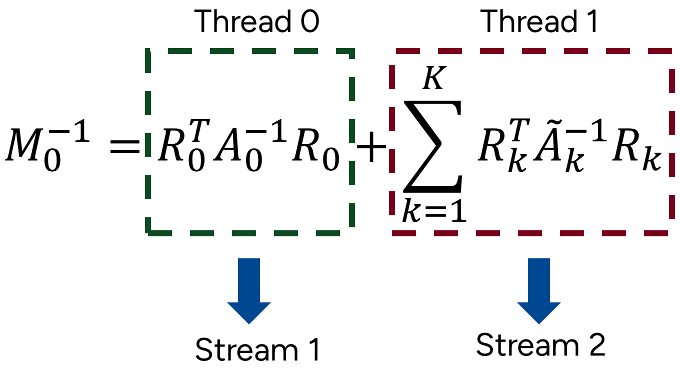
- Coarse grid solved using an approximate Krylov solver
  - Preconditioned Pipelined Conjugate Gradient with a low, maximum iteration limit
- Low computational efficiency on GPUs
  - A_0 is on linear elements, too little data to keep the GPU busy.
  - Many small kernels, dominated by kernel launch latency

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

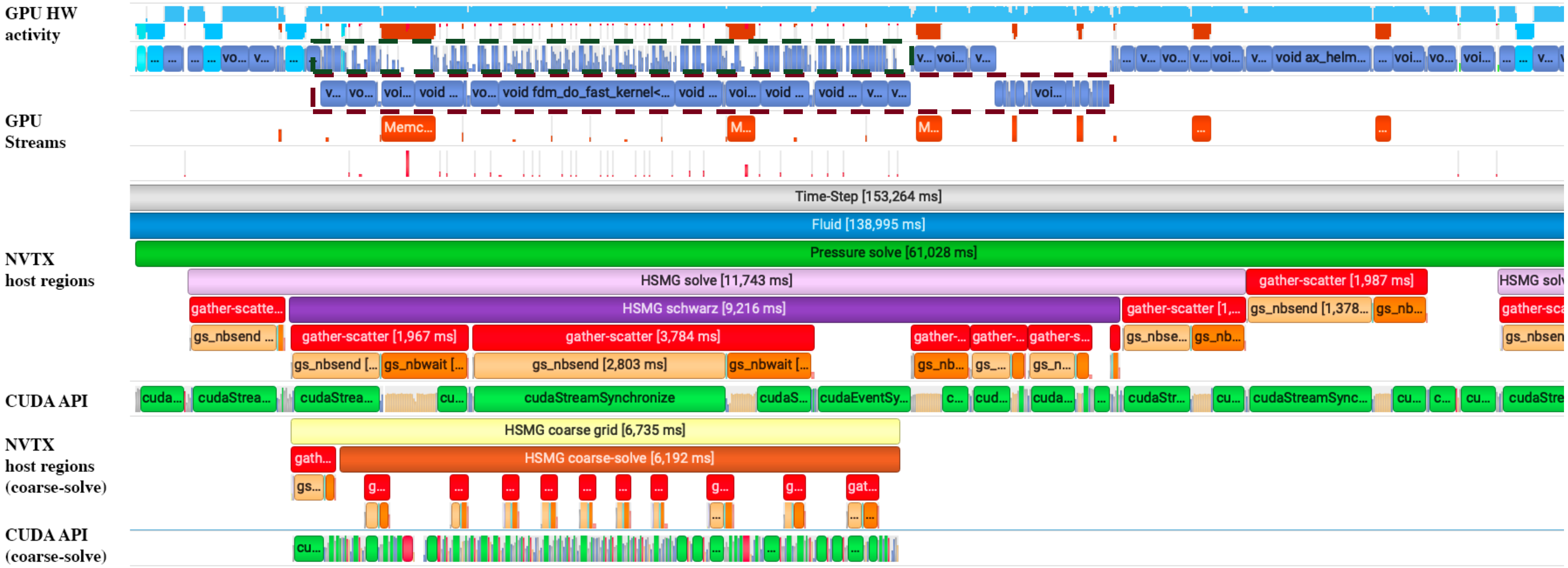


Task-decomposed Overlapped Preconditioner

- 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

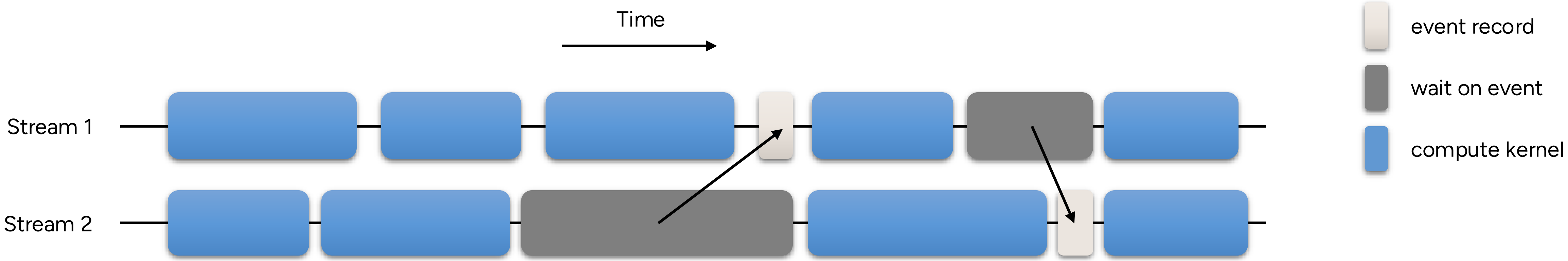


$$M_0^{-1} = \underbrace{R_0^T A_0^{-1} R_0}_{\text{Thread 0}} + \underbrace{\sum_{k=1}^{K} R_k^T \tilde{A}_k^{-1} R_k}_{\text{Thread 1}}$$



Task-decomposed Overlapped Preconditioner

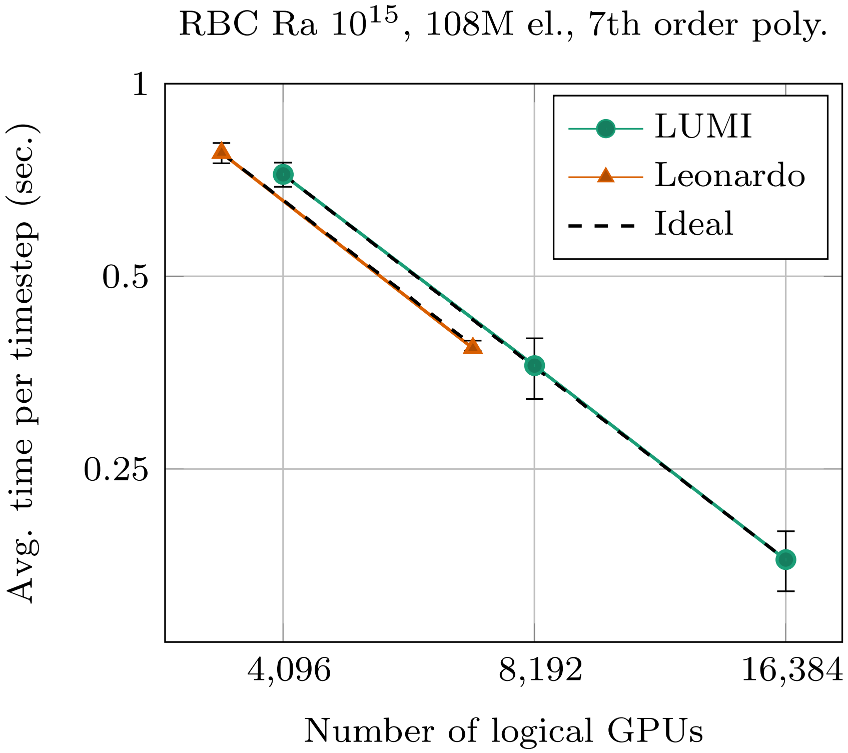
- How to synchronise between dependencies in different streams and/or host
  - Device syntonisation **too costly** (`hipDeviceSynchronize`, `cudaDeviceSynchronize`)
- Fine grained synchronisation using events
  - Events are recorded in a stream once dependencies are ready
  - Other streams can wait on specific events in other streams
  - Allow for one-to-one, one-to-many and many-to-one synchronisation without halting the device



Performance Results

- Performance measurements on two of the EuroHPC-JU pre-exascale supercomputers **LUMI** and **Leonardo**
  - RBC in a cylinder with aspect ratio 1:10, $Ra = 10^{15}$
  - 108M elements, 7th order polynomials
  - 37B unique grid points and more than 148B degrees of freedom
  - Figure of merit: strong scalability, average time per timestep (after transient)
- 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



RBC Ra 10^{15}, 108M el., 7th order poly.
Avg. time per timestep (sec.)
1
0.5
0.25
4,096
8,192
16,384
Number of logical GPUs
LUMI
Leonardo
Ideal

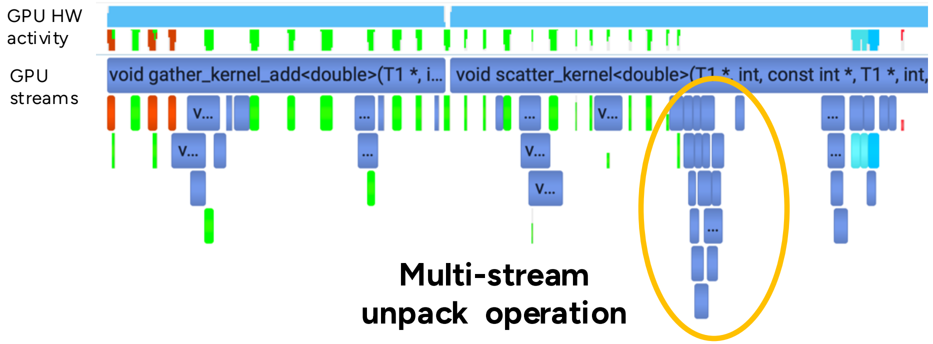
99% confidence intervals is illustrated as error bars

Illustration of the canonical problem at $Ra = 10^{13}$, iso-surfaces of temperature

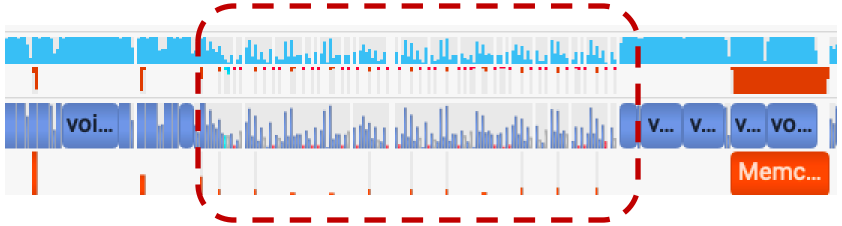
N. Jansson et al., Exploring the Ultimate Regime of Turbulent Rayleigh-Bénard Convection Through Unprecedented Spectral-Element Simulations, SC '23: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, 2023.

Performance Evaluation

- Only considering the pressure solver (GMRES)
  - Measure throughput (GDoF/s) per GMRES iteration
- Test case
  - Taylor-Green Vortex, Reynolds number $Re = 1600$
  - Two different meshes, 128^3 and 256^3 elements
  - 7th order polynomials ≈1G and 8.5G DoFs
- Performance measured for several combinations
  - Non-overlapped (*nt1*) and new overlapped preconditioner (*nt2*)
  - Four different gather-scatter strategies:
    - *gs1:* single stream pack/unpack, *gs2:* multi-stream pack
gs3: multi-stream unpack, *gs4:* multi-stream pack/unpack
  - Three different coarse-grid solvers
    - Standard preconditioned conjugate gradient (*standard PCG*)
    - Pipelined preconditioned conjugate gradient (*pipelined PCG*)
    - Standard PCG using kernel fusion (Fused kernels PCG)


GPU HW activity
GPU streams
void gather_kernel_add<double>(T1 *, i...
void scatter_kernel<double>(T1 *, int, const int *, T1 *, int,
v...
...

Multi-stream unpack operation

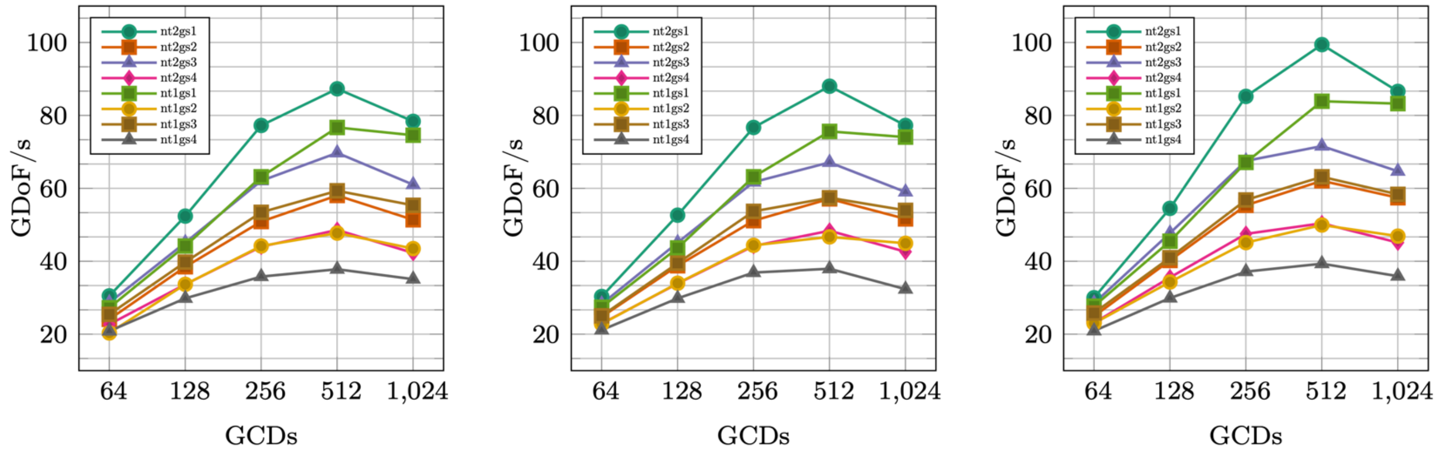


Fuse kernels (if possible)

Performance Results

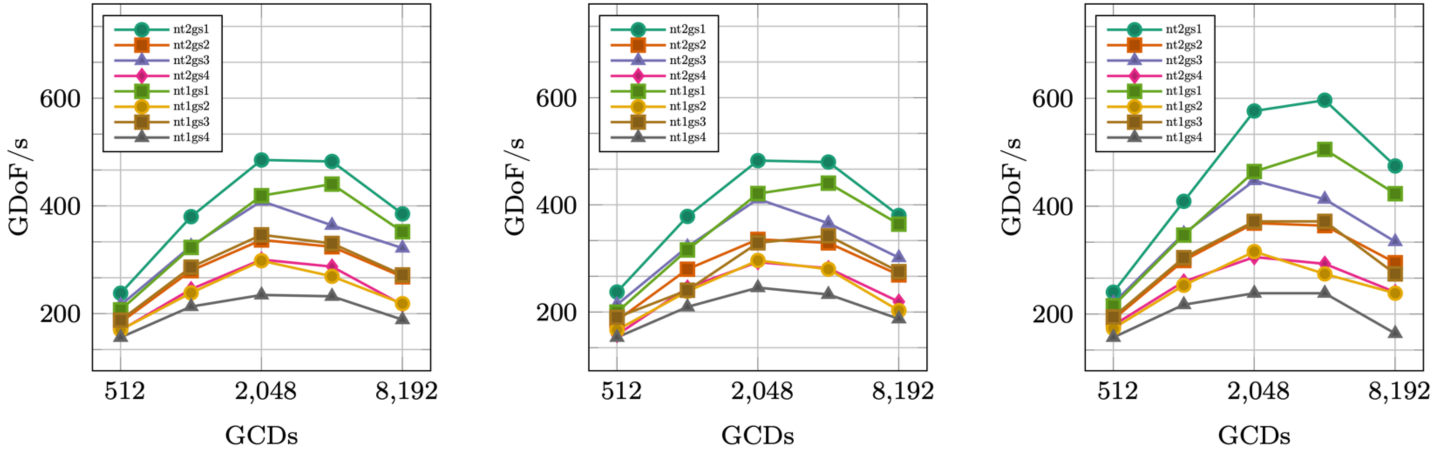
- Highest throughput with the new formulation using a single stream (*nt2gs1*)
- No apparent benefit from using multi-stream pack/unpack
    - **Unexpected result**, early results on NVIDIA GPU platforms indicate a benefit
    - More detailed analysis necessary (**tools…**)
- No apparent benefit from using the fused kernels PCG
    - Possibly **scheduling issues**
- Clear benefit from pipelined PCG.
- Strong scalability is lost around 7000 elements per GCD
- Similar throughput trends for both the small and the large test case

128^3 elements



(a) **Standard PCG.** (b) **Fused kernels PCG.** (c) **Pipelined PCG.**

256^3 elements



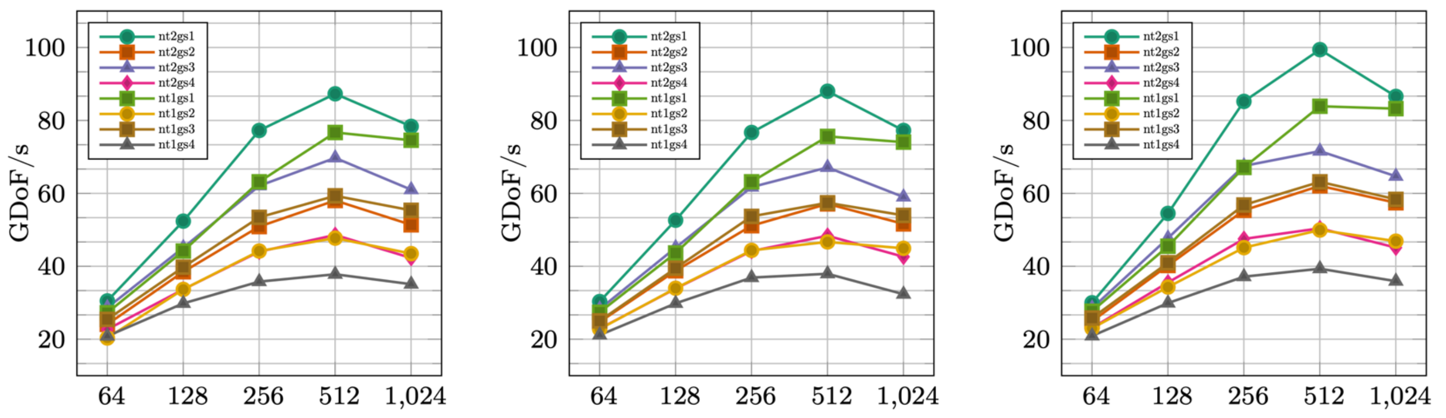
(a) **Standard PCG.** (b) **Fused kernels PCG.** (c) **Pipelined PCG.**

Performance Results

- LUMI's ROCm 5.2.3 is old (Aug. 2022)
  - Scheduling of work in the GPU stream could have been improved in later releases
- Tested the 128^3 element case on Frontier
  - ROCm 6.2.0 (Aug. 2024)
- More than **25% - 50% higher throughput**
- Still no benefit from using multi-stream pack/unpack
- Still no benefit from the fused CG version
- At scale, large benefit from pipelined PCG

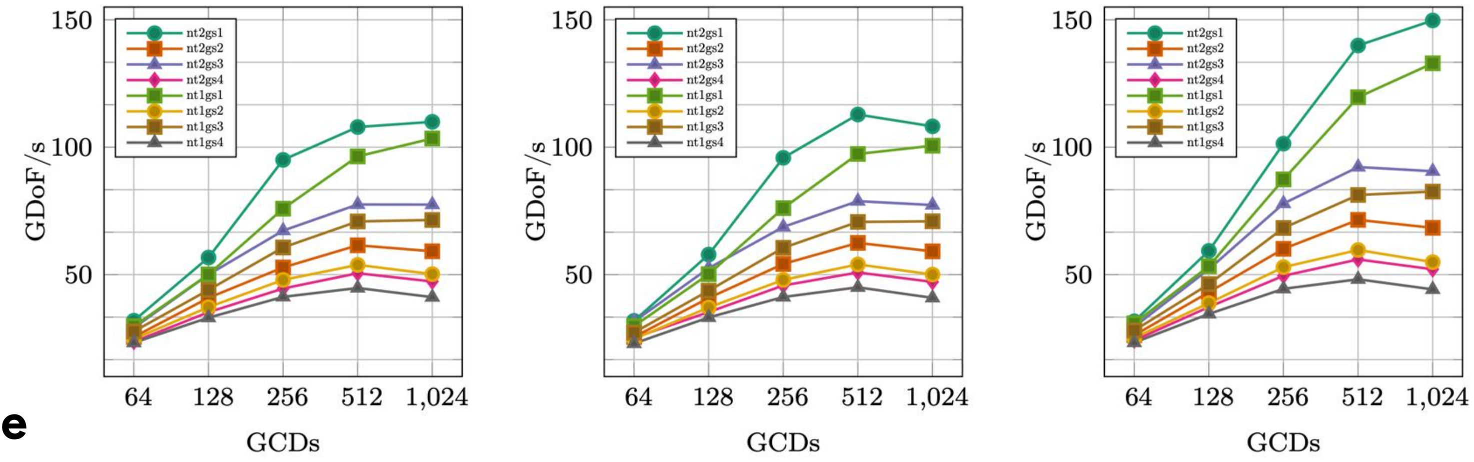
Driver/runtime is important for performance

LUMI



(a) Standard PCG. (b) Fused kernels PCG. (c) Pipelined PCG.

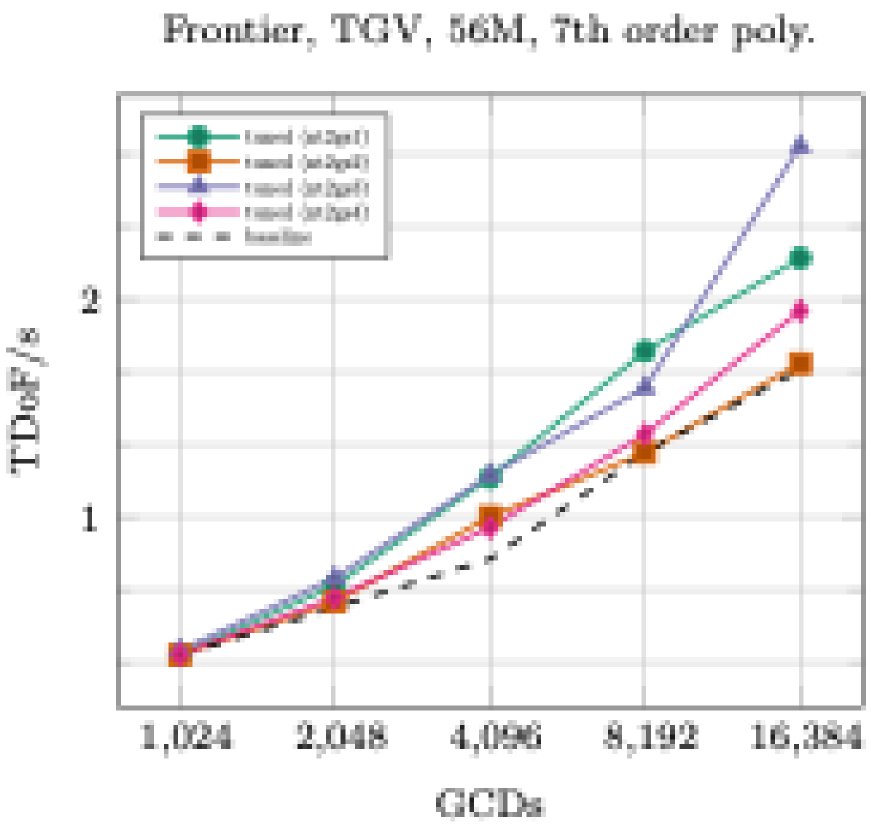
Frontier



(a) Standard PCG. (b) Fused kernels PCG. (c) Pipelined PCG.

Extreme-scale Performance

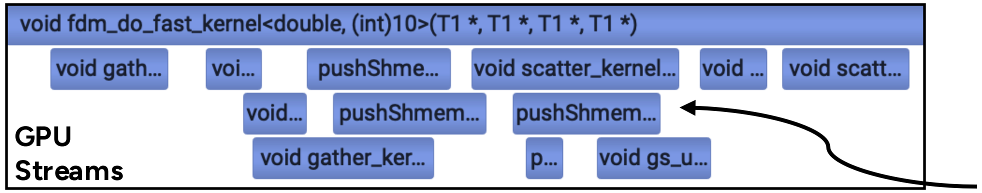
- Large scale performance evaluation on Frontier
  - Only considering the pressure solver (GMRES)
  - Measure throughput (TDoF/s) per GMRES iteration
  - Pipelined PCG as coarse grid solver
- Similar test case as before
  - Taylor-Green Vortex, Reynolds number $Re = 1600$
  - Larger meshes, 56M elements
  - 7th order polynomials ≈29G DoFs
- Some benefit from using multi-stream pack/unpack at scale
- Improvements with less than 4000 elements per GCD
- More than **60% higher throughput at scale**



Further Performance Engineering

GPU-initiated communication

- The pressure preconditioner needs frequent, fine-grained communication
  - MPI calls are issued from the CPU ⟹ **GPU execution will stall**
- New NVSHMEM based gather-scatter backend
  - Non-blocking signalling put operations (p2p)



void fdm_do_fast_kernel<double, (int)10>(T1 *, T1 *, T1 *, T1 *)

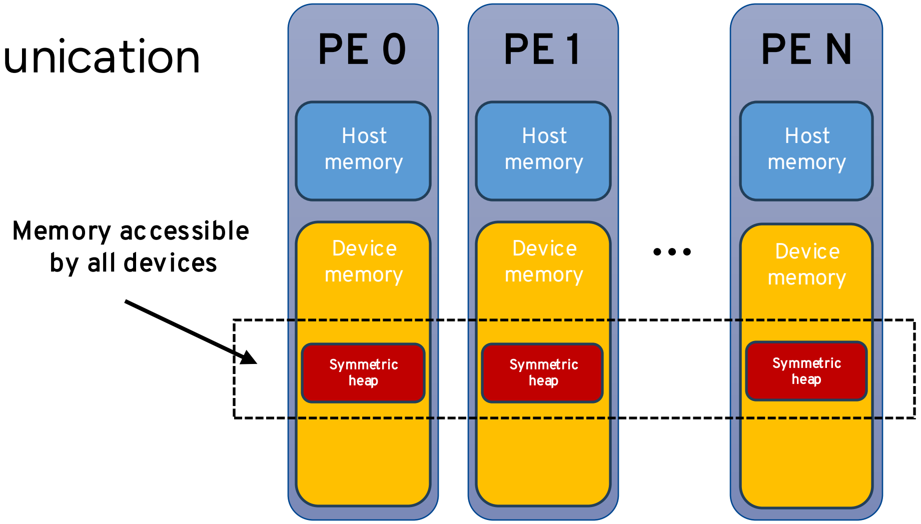
void gath... voi... pushShme... void scatter_kernel... void ... void scatt...

void... pushShmem... pushShmem...

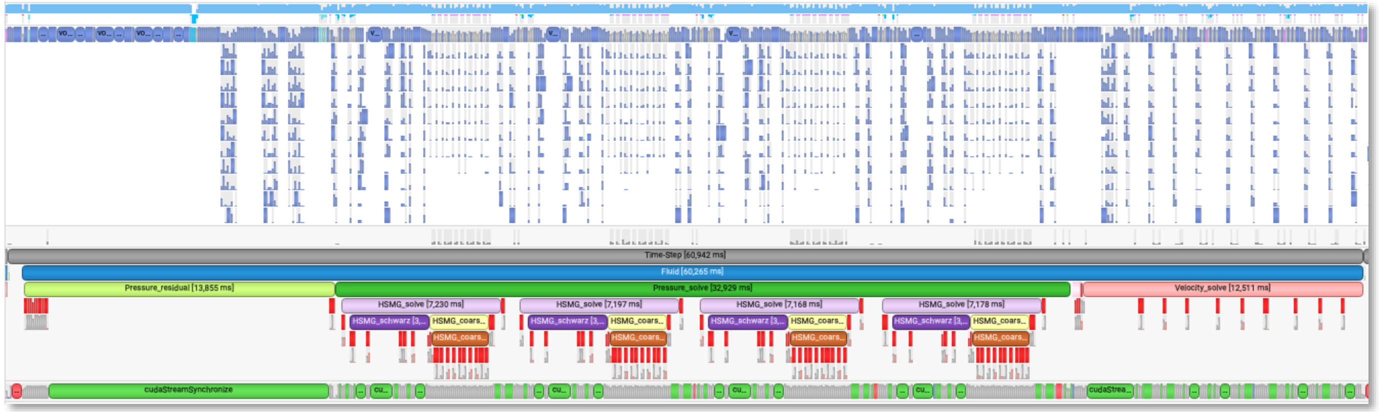
void gather_ker... p... void gs_u...

GPU Streams

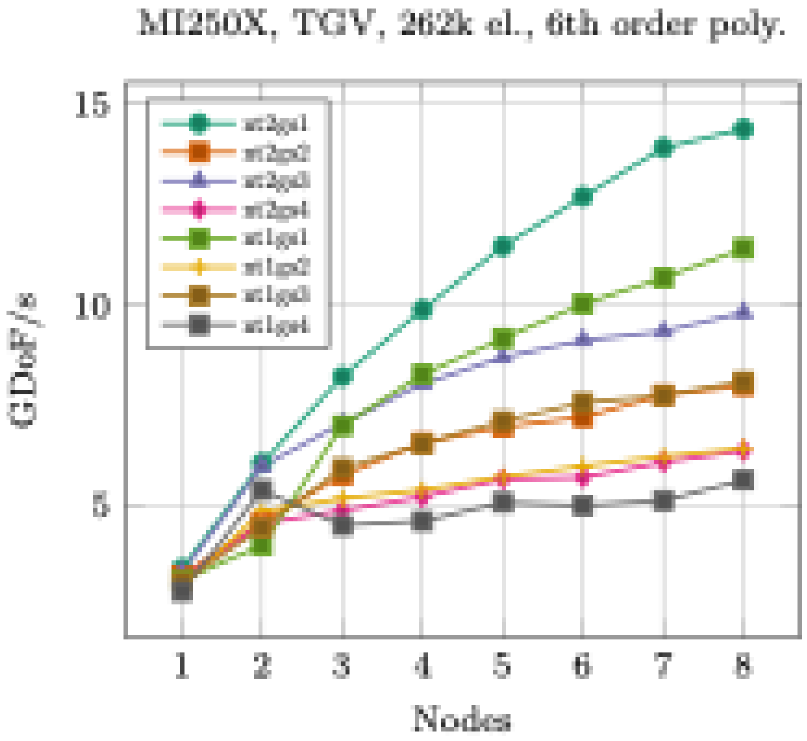
SHMEM comm. as device kernels

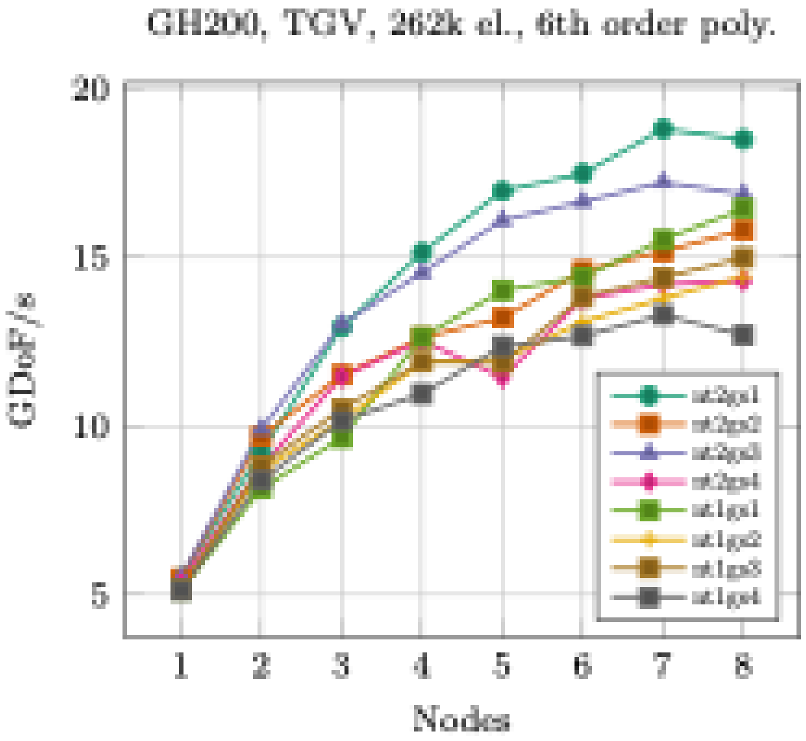


- Reducing latency, no “communication stalls”
  - `cudaStreamSynchronize` replaced with `cudaEventSynchronize`
- Both computation and communication “scheduled by the accelerator”
- Performance evaluation ongoing



Comparing AMD and NVIDIA blades





- Node to node comparsion between EX235a and EX254n blades
  - MI250x: GCC 13.2, ROCm 6.2.4
  - GH200: GCC 13.2, CUDA 12.3

Combining AMD and NVIDA

- Can we use both node types in a single simulation?
  - Works out of the box! (…some MPI I/O issues)

```
export MPICH_GPU_SUPPORT_ENABLED=1
export OMP_NUM_THREADS=2

if [ `uname -m` == x86_64 ]; then
   ./neko_x86 tgv_iccfd.case
fi

if [ `uname -m` == aarch64 ]; then
   ./neko_aarch64 tgv_iccfd.case
fi
```

  - 4xGH200 + 4xMI250x nodes
- Any performance benefits?

Nodes	GDoF/s
4xGH200 + 4xMI250X	15.4151
8xMI250X	14.3775
8xGH200	18.4984

```
    _  __  ____  __ __  ____
   / |/ / / __/ / //_/ / __ \
  /    / / _/  / ,<   / /_/ /
 /_/|_/ /___/ /_/|_|  \____/

(version: 0.9.99)
(build: 2025-01-17 on x86_64-unknown-linux-gnu using gnu)

-------Job Information--------
Start time: 09:16 / 2025-01-17
Running on: 48 MPI ranks, using 2 thrds each
CPU type  :
     (rank:   0 )AMD EPYC 7A53 64-Core Processor
     (rank:   1 )AMD EPYC 7A53 64-Core Processor
     (rank:   2 )AMD EPYC 7A53 64-Core Processor
     (rank:   3 )AMD EPYC 7A53 64-Core Processor
     ...
     (rank:  44 )ARM Neoverse V2
     (rank:  45 )ARM Neoverse V2
     (rank:  46 )ARM Neoverse V2
     (rank:  47 )ARM Neoverse V2
Bcknd type:
     (rank:   0 )Accelerator (HIP)
     (rank:   1 )Accelerator (HIP)
     (rank:   2 )Accelerator (HIP)
     (rank:   3 )Accelerator (HIP)
            ...
     (rank:  44 )Accelerator (CUDA)
     (rank:  45 )Accelerator (CUDA)
     (rank:  46 )Accelerator (CUDA)
     (rank:  47 )Accelerator (CUDA)
Dev. name :
     (rank:   0 )AMD Instinct MI250X
     (rank:   1 )AMD Instinct MI250X
     (rank:   2 )AMD Instinct MI250X
     (rank:   3 )AMD Instinct MI250X
     ...
     (rank:  44 )NVIDIA GH200 120GB
     (rank:  45 )NVIDIA GH200 120GB
     (rank:  46 )NVIDIA GH200 120GB
     (rank:  47 )NVIDIA GH200 120GB
```

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**
  - **Drivers/runtime** important for performance
- Exploit all the **available concurrency** of the application
  - Key ingredient to achieve good strong scalability on LUMI and Frontier

SeRC Swedish e-Science Research Centre

Swedish Research Council

CEEC

INNO4SCALE

www.neko.cfd

KTH

VETENSKAP OCH KONST