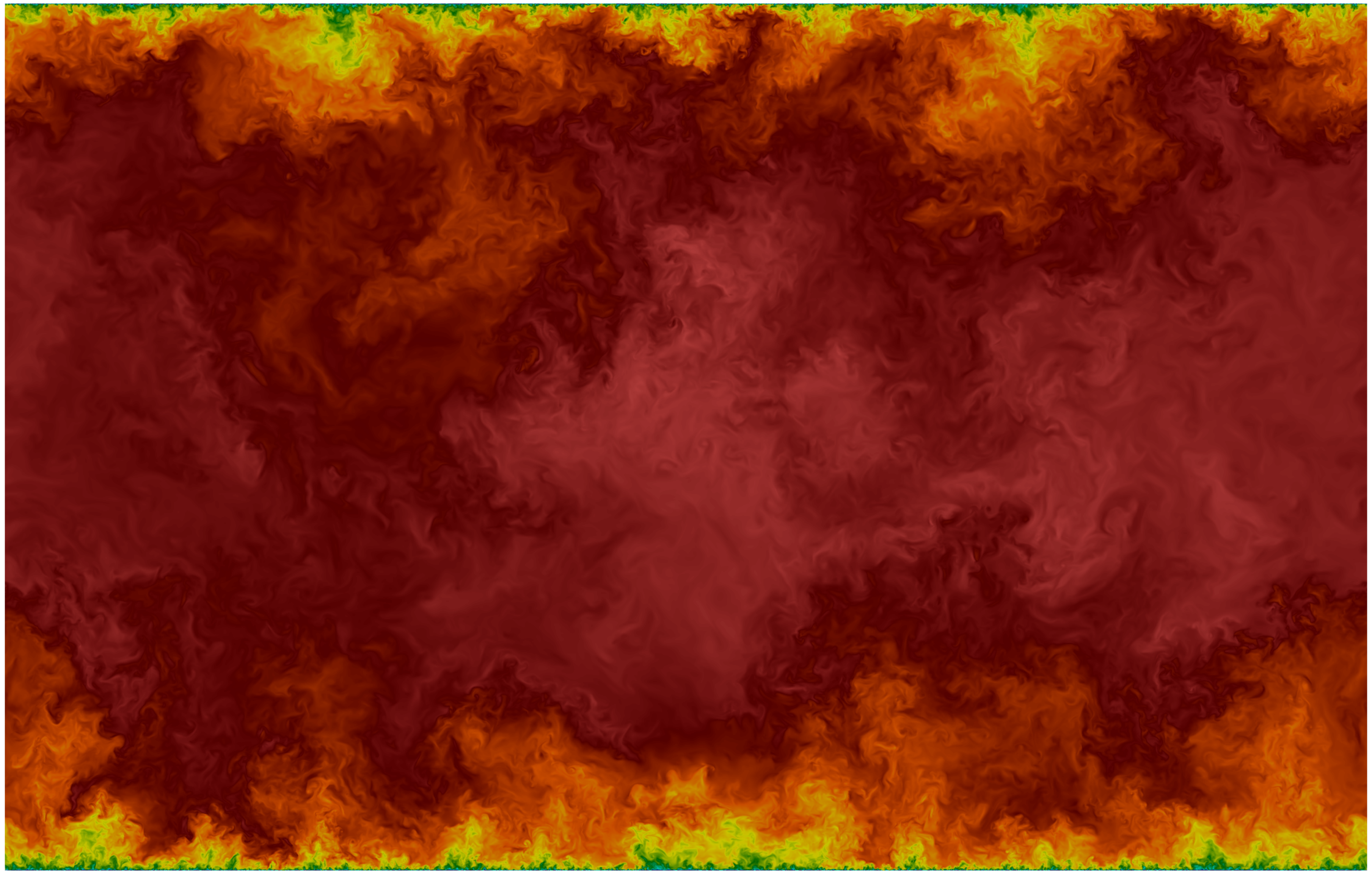
tooling up for extreme-scale simulations of turbulent flows



Pedro Costa

TU Delft, Mechanical Engineering, Process & Energy Department
P.SimoesCosta@tudelft.nl

NUM-SCARS, CWI, 23 January 2026

acknowledgements

TU Delft

J. Peeters

R. Diez

D. Palancha

NVIDIA

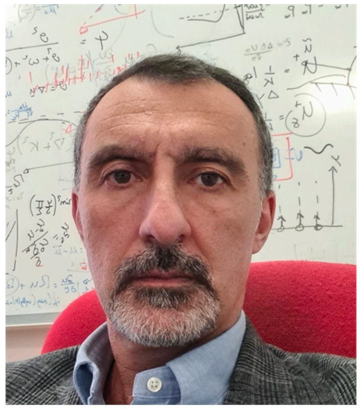
J. Romero

M. Fatica

GSSI

R. Verzicco

U. Rome La Sapeinza


S. Pirozzoli

outline

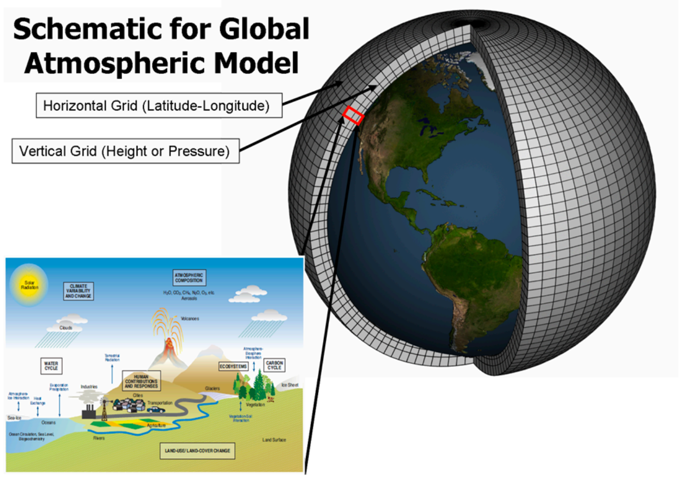
- motivation
- DNS of turbulent flows with immersed interfaces (e.g., obstacles, particles, or multi-fluid flows)
- tailoring our DNS solver for extreme-scale simulations on many GPUs
  - numerical method tailoring
  - runtime performance autotuning
  - mixed precision & distributed TDMA
  - fast solver for non-uniform grids
- summary & outlook

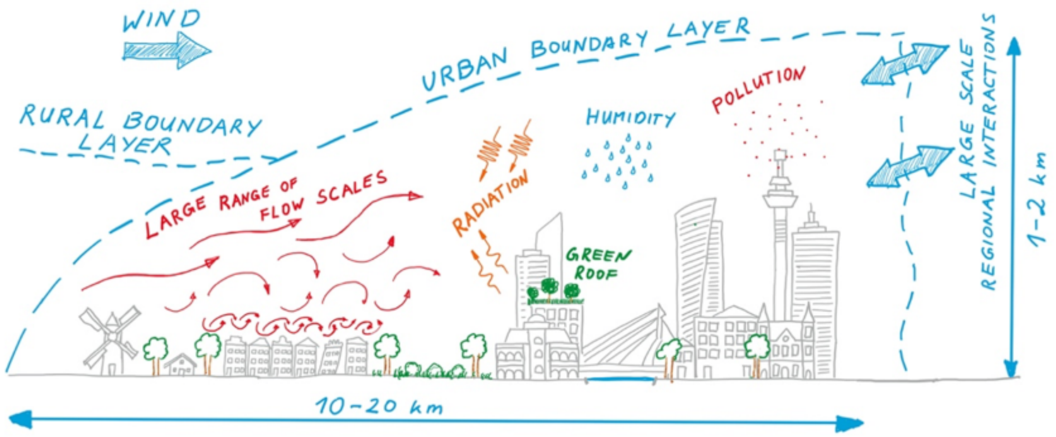
outline

- motivation
- DNS of turbulent flows with immersed interfaces (e.g., obstacles, particles, or multi-fluid flows)
- tailoring our DNS solver for extreme-scale simulations on many GPUs
  - numerical method tailoring
  - runtime performance autotuning
  - mixed precision & distributed TDMA
  - fast solver for non-uniform grids
- summary & outlook

motivation

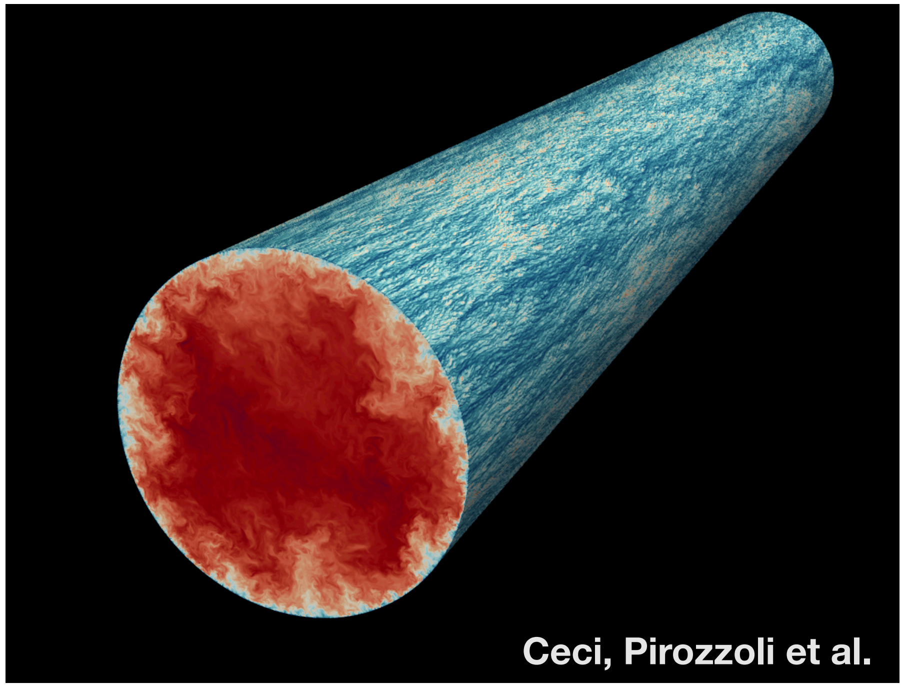
why keep pushing the limits of our numerical solvers?


Schematic for Global Atmospheric Model
Horizontal Grid (Latitude-Longitude)
Vertical Grid (Height or Pressure)



climate / energy

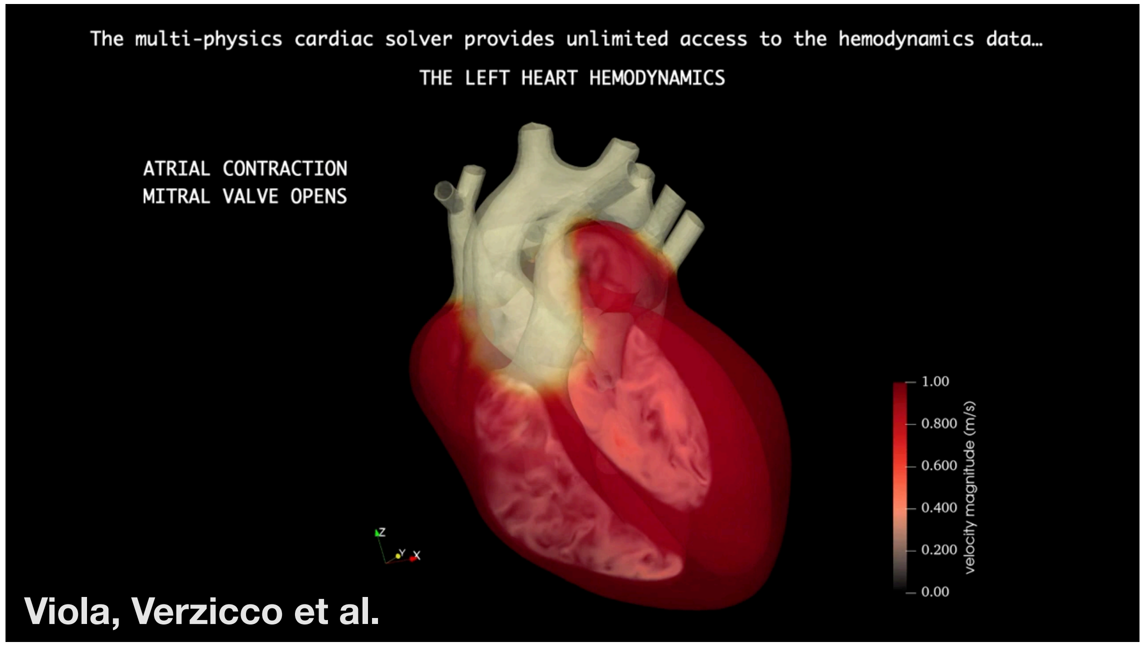
need to **increase resolution**!



Ceci, Pirozzoli et al.

wall turbulence fundamentals

need **higher Reynolds numbers**


The multi-physics cardiac solver provides unlimited access to the hemodynamics data…
THE LEFT HEART HEMODYNAMICS
ATRIAL CONTRACTION
MITRAL VALVE OPENS
1.00
0.800
0.600
0.400
0.200
0.00
velocity magnitude (m/s)
z
y
x

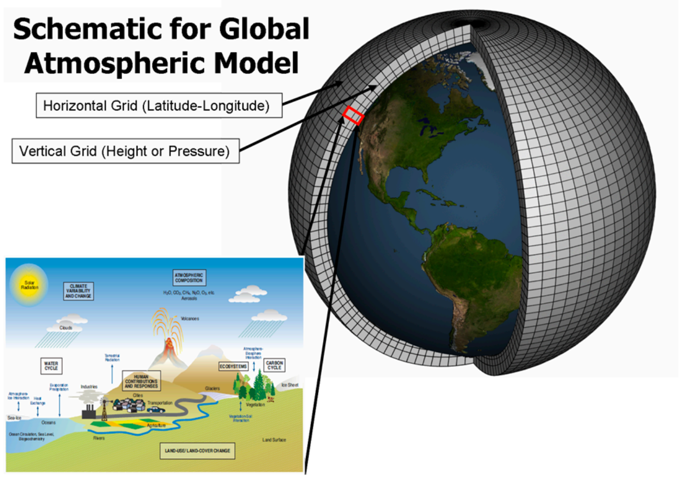
Viola, Verzicco et al.

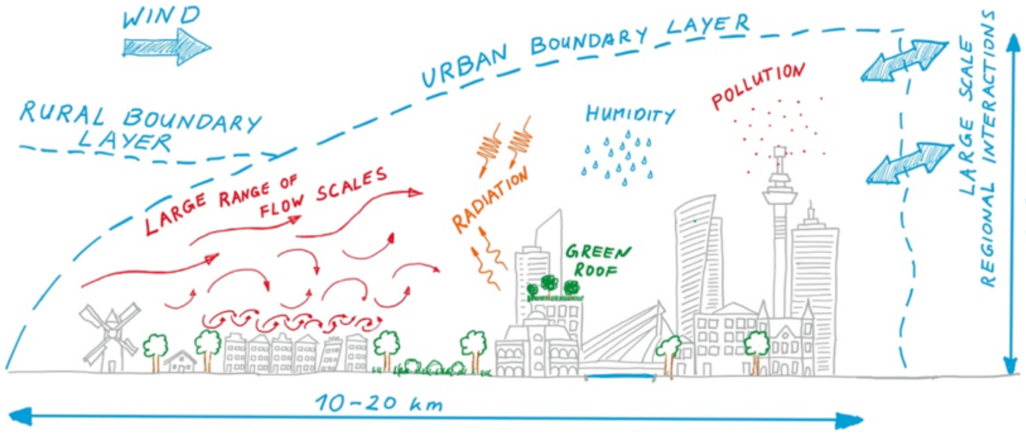
health

need for **fast** predictions and U.Q.

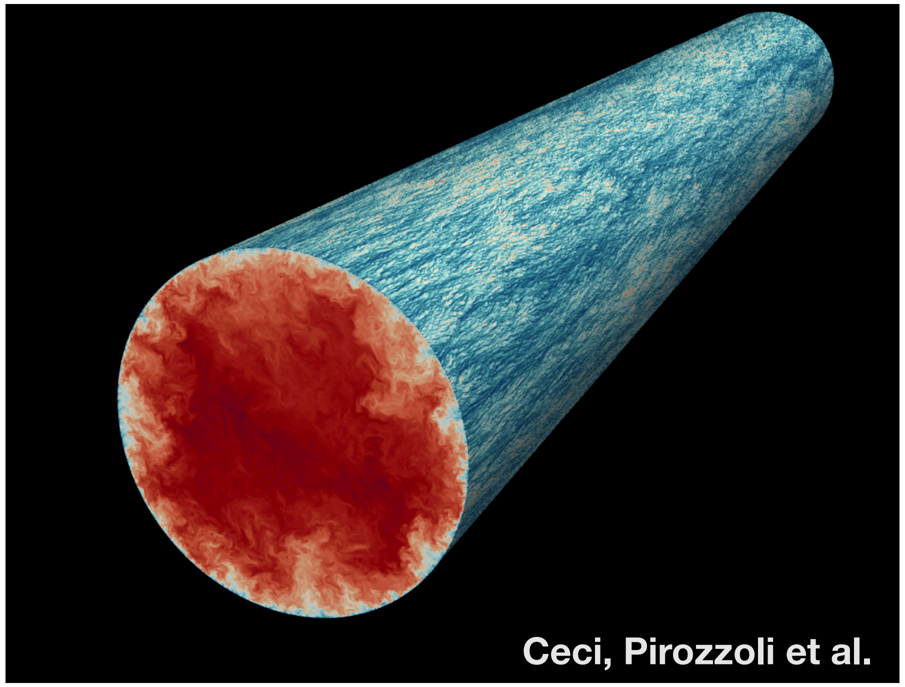
motivation

why keep pushing the limits of our numerical solvers?


Schematic for Global Atmospheric Model
Horizontal Grid (Latitude-Longitude)
Vertical Grid (Height or Pressure)

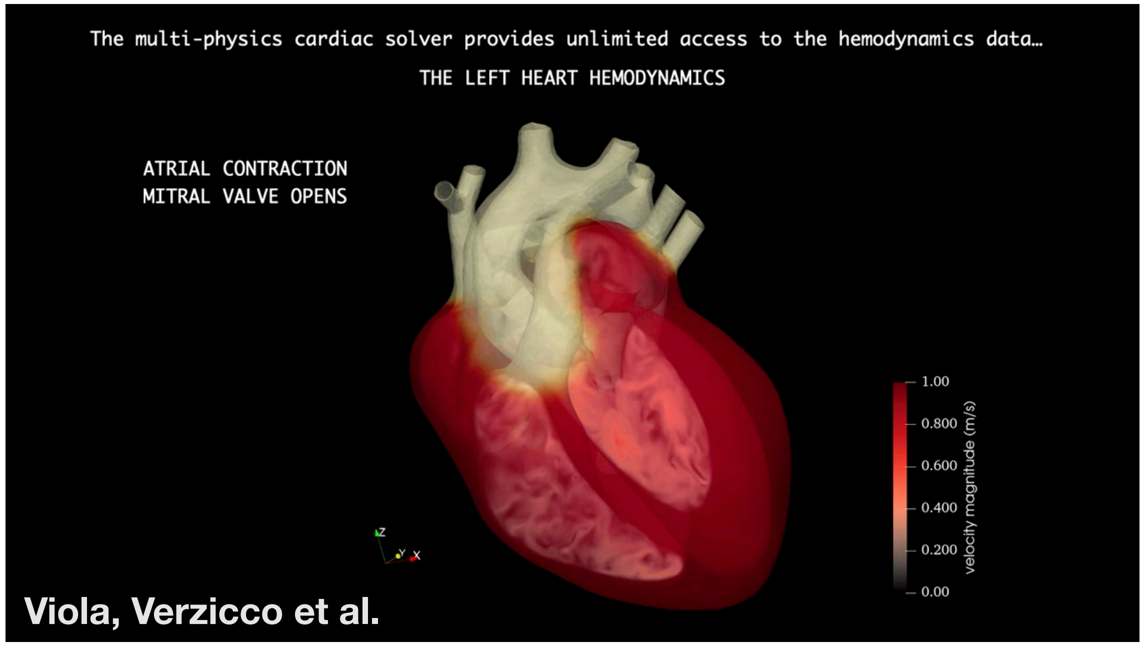


climate / energy
need to **increase resolution**!



Ceci, Pirozzoli et al.

wall turbulence fundamentals
need **higher Reynolds numbers**



The multi-physics cardiac solver provides unlimited access to the hemodynamics data…

THE LEFT HEART HEMODYNAMICS

ATRIAL CONTRACTION
MITRAL VALVE OPENS

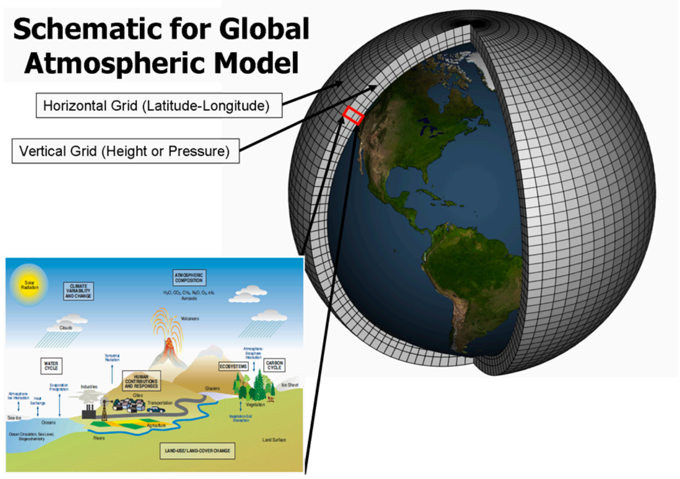
Viola, Verzicco et al.

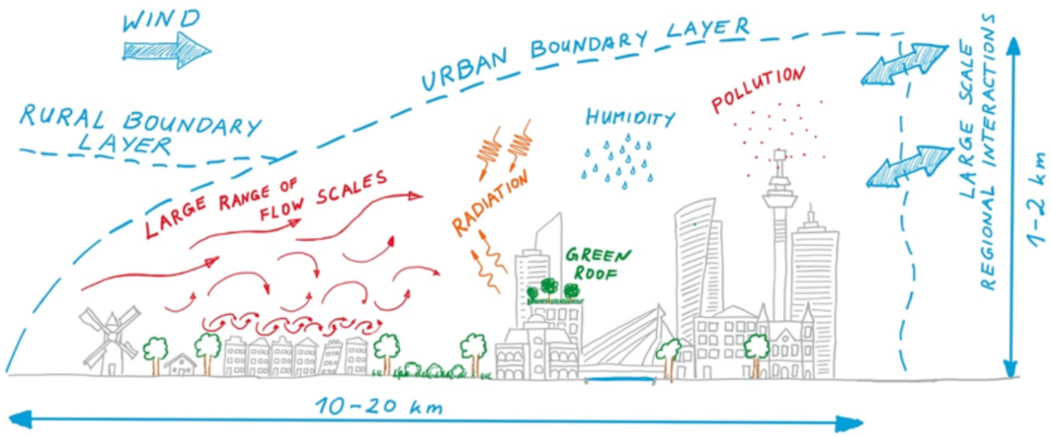
health
need for **fast** predictions and U.Q.

continuous demand for **resolving larger systems / many small systems faster**

motivation

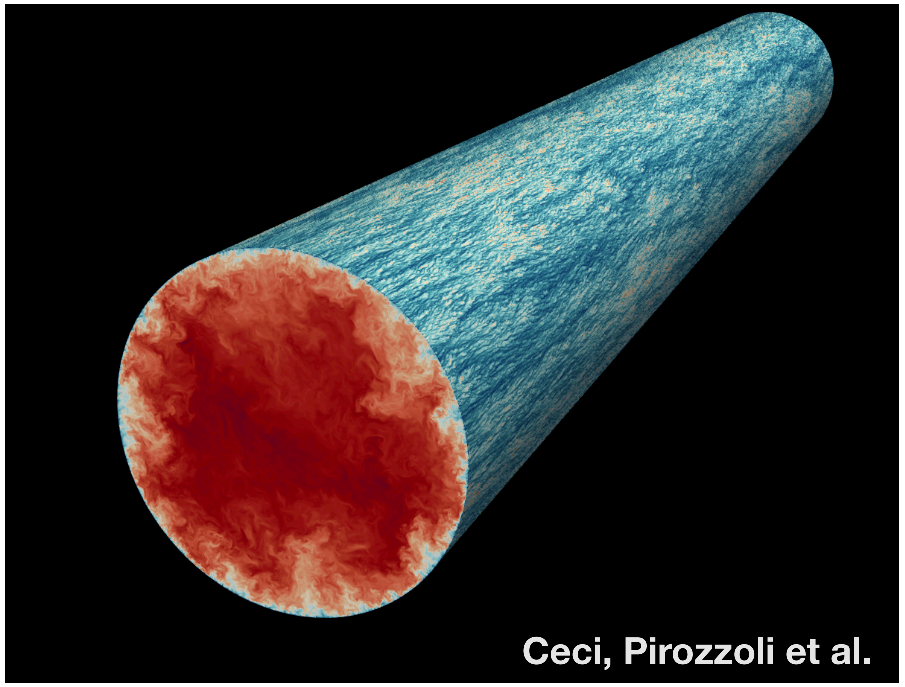
why keep pushing the limits of our numerical solvers?


Schematic for Global Atmospheric Model
Horizontal Grid (Latitude-Longitude)
Vertical Grid (Height or Pressure)



climate / energy

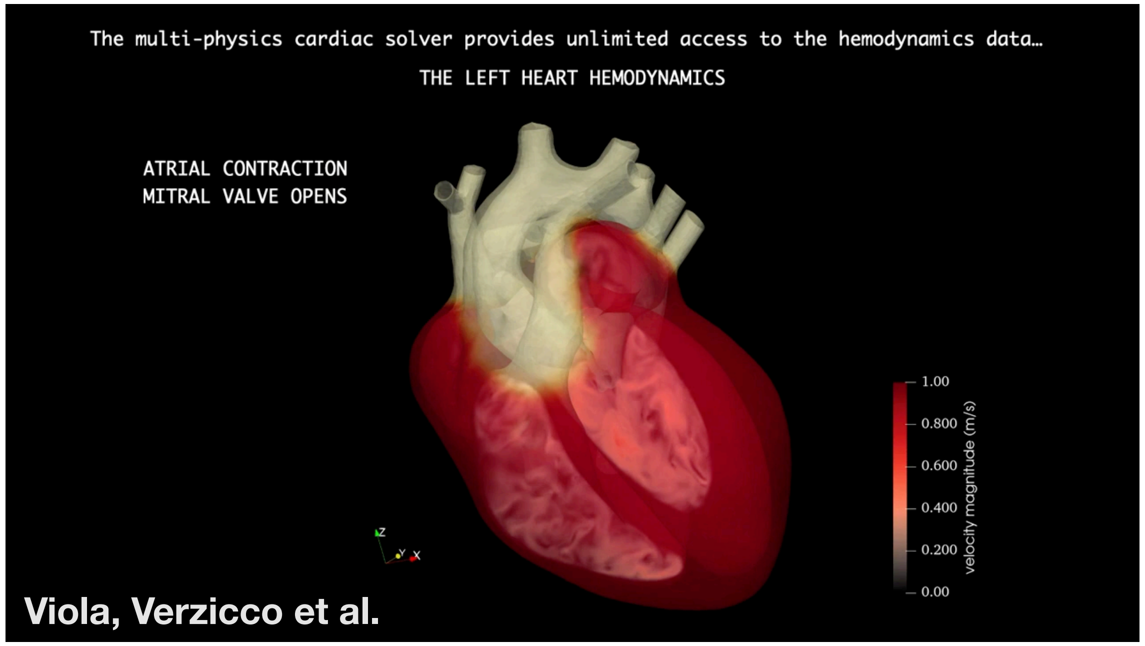
need to **increase resolution**!



Ceci, Pirozzoli et al.

wall turbulence fundamentals

need **higher Reynolds numbers**


The multi-physics cardiac solver provides unlimited access to the hemodynamics data…
THE LEFT HEART HEMODYNAMICS
ATRIAL CONTRACTION
MITRAL VALVE OPENS
1.00
0.800
0.600
0.400
0.200
0.00
velocity magnitude (m/s)
z
y
x

Viola, Verzicco et al.

health

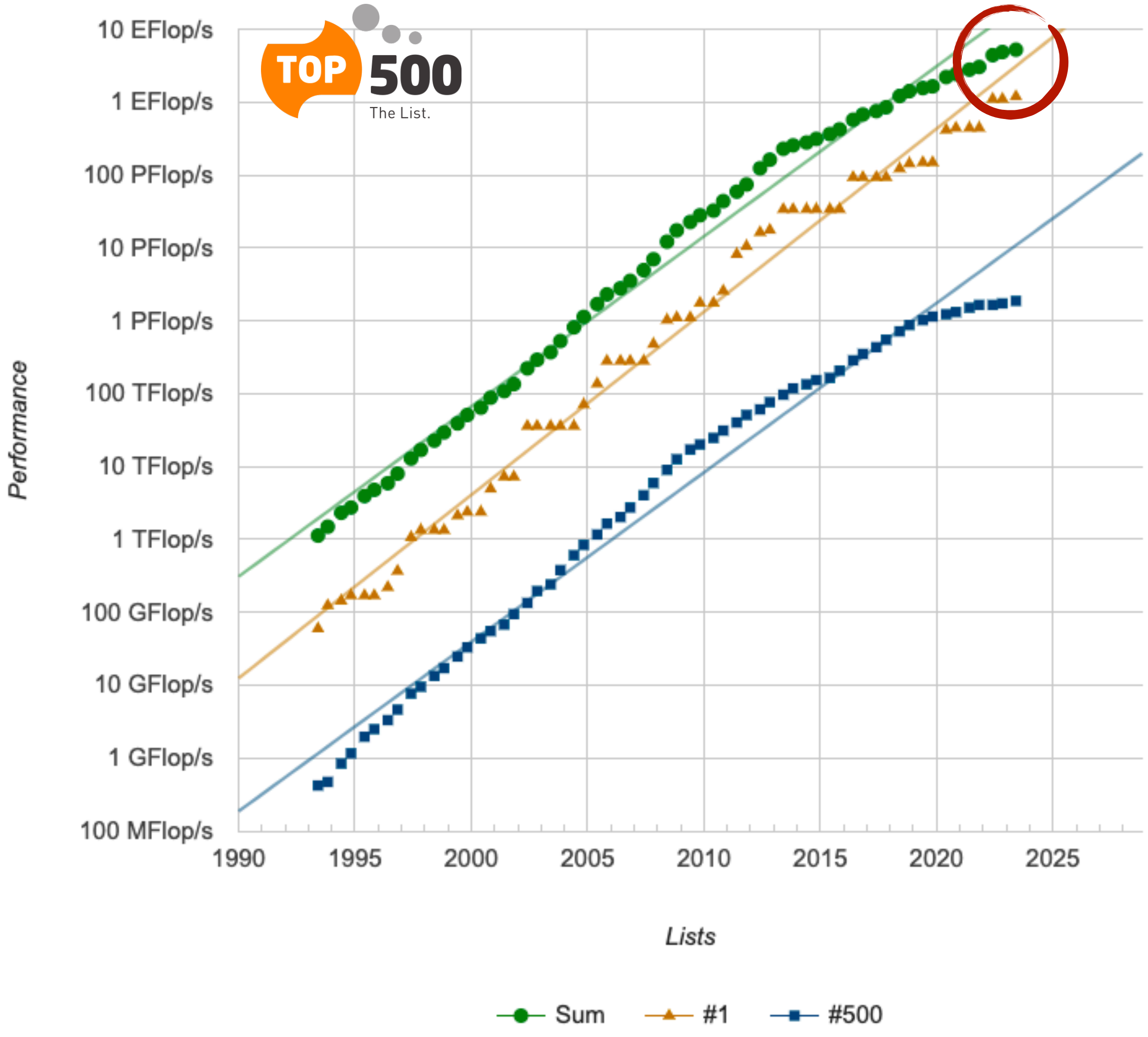
need for **fast** predictions and U.Q.

continuous demand for **resolving larger systems / many small systems faster**

change in paradigm

log scale!

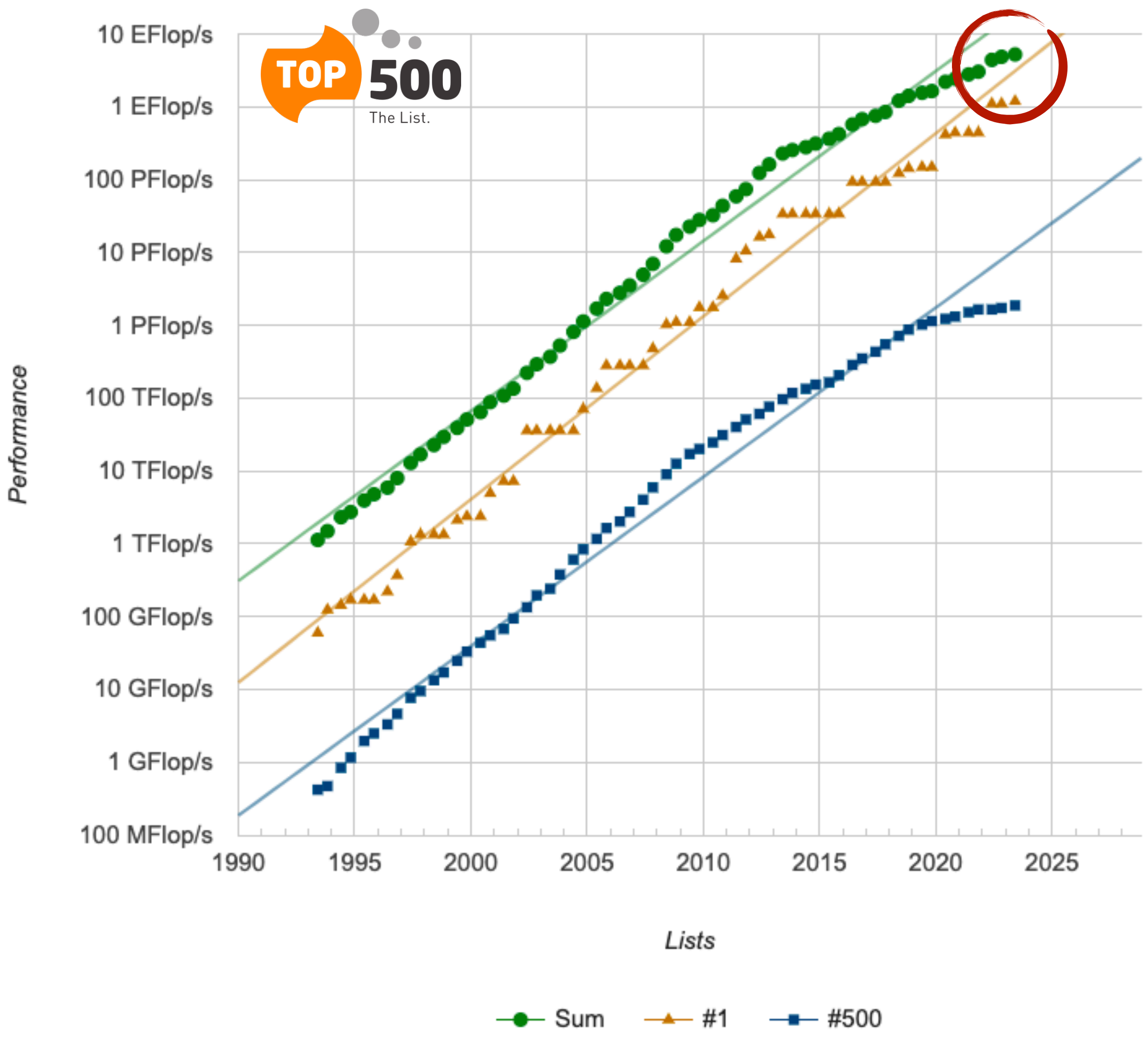
10^{18} FLOPS/s

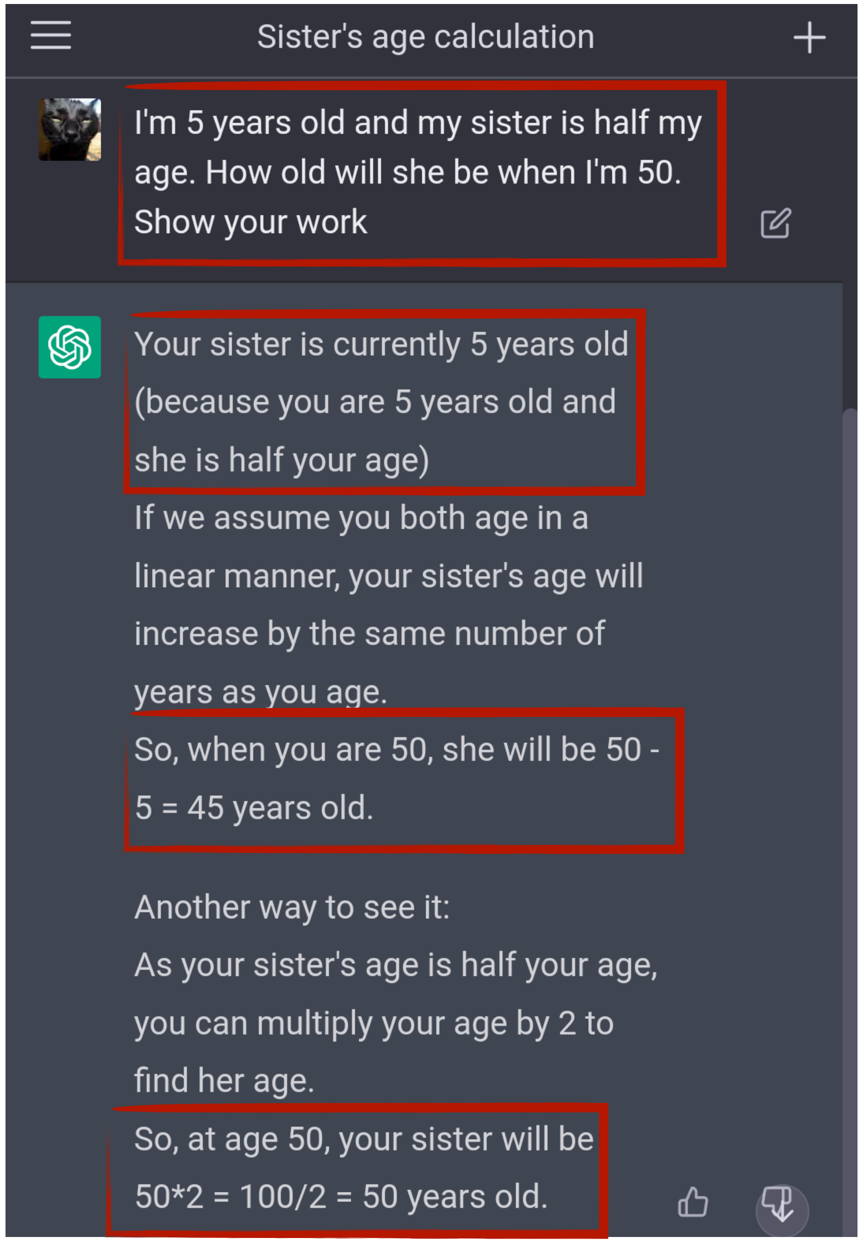


change in paradigm

log scale!

10^{18} FLOPS/s





I'm 5 years old and my sister is half my age. How old will she be when I'm 50. Show your work

Your sister is currently 5 years old (because you are 5 years old and she is half your age)

If we assume you both age in a linear manner, your sister's age will increase by the same number of years as you age.

So, when you are 50, she will be 50 - 5 = 45 years old.

Another way to see it:

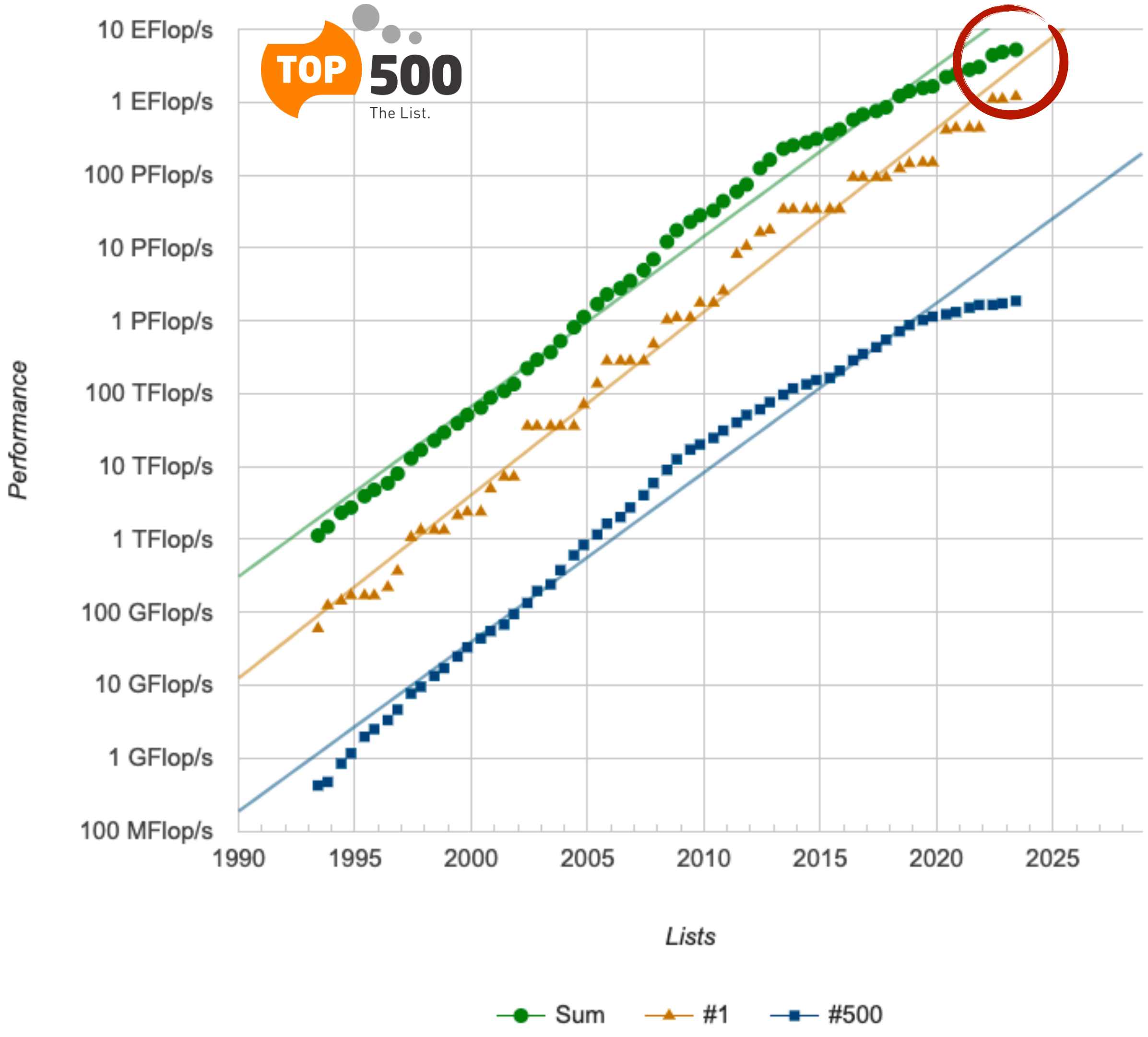
As your sister's age is half your age, you can multiply your age by 2 to find her age.

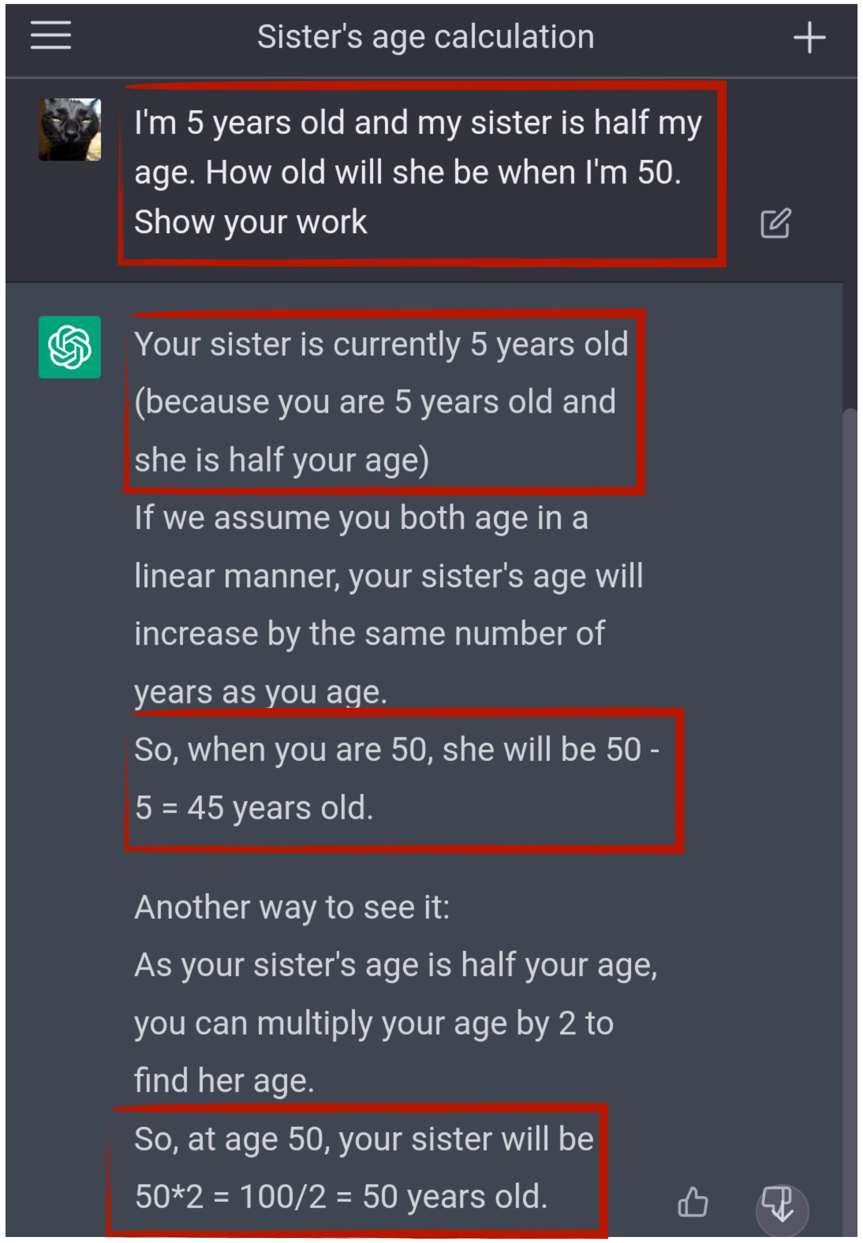
So, at age 50, your sister will be 50*2 = 100/2 = 50 years old.

change in paradigm

log scale!

10^{18} FLOPS/s





I'm 5 years old and my sister is half my age. How old will she be when I'm 50. Show your work

Your sister is currently 5 years old (because you are 5 years old and she is half your age)

If we assume you both age in a linear manner, your sister's age will increase by the same number of years as you age.

So, when you are 50, she will be 50 - 5 = 45 years old.

Another way to see it:

As your sister's age is half your age, you can multiply your age by 2 to find her age.

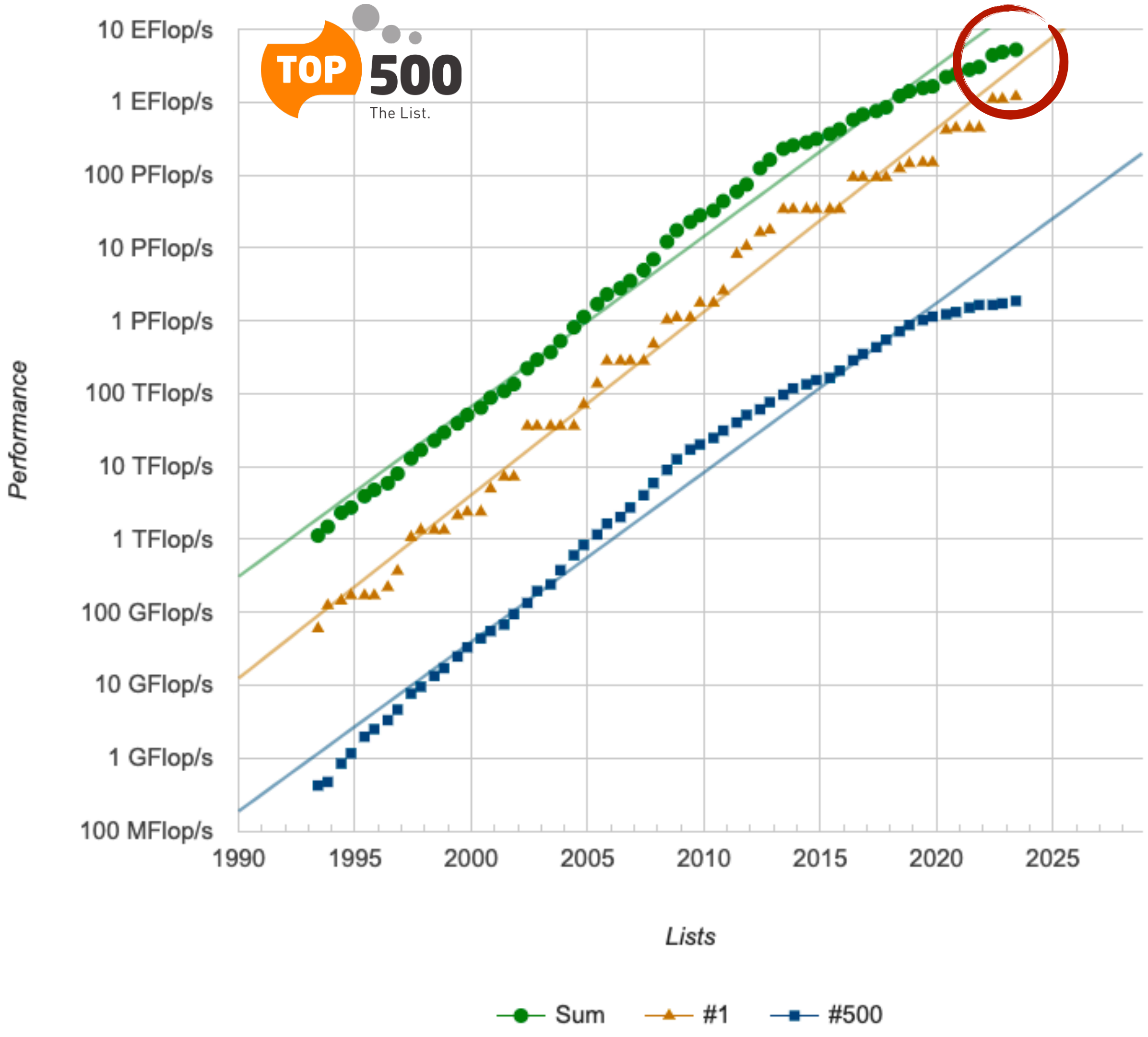
So, at age 50, your sister will be 50*2 = 100/2 = 50 years old.

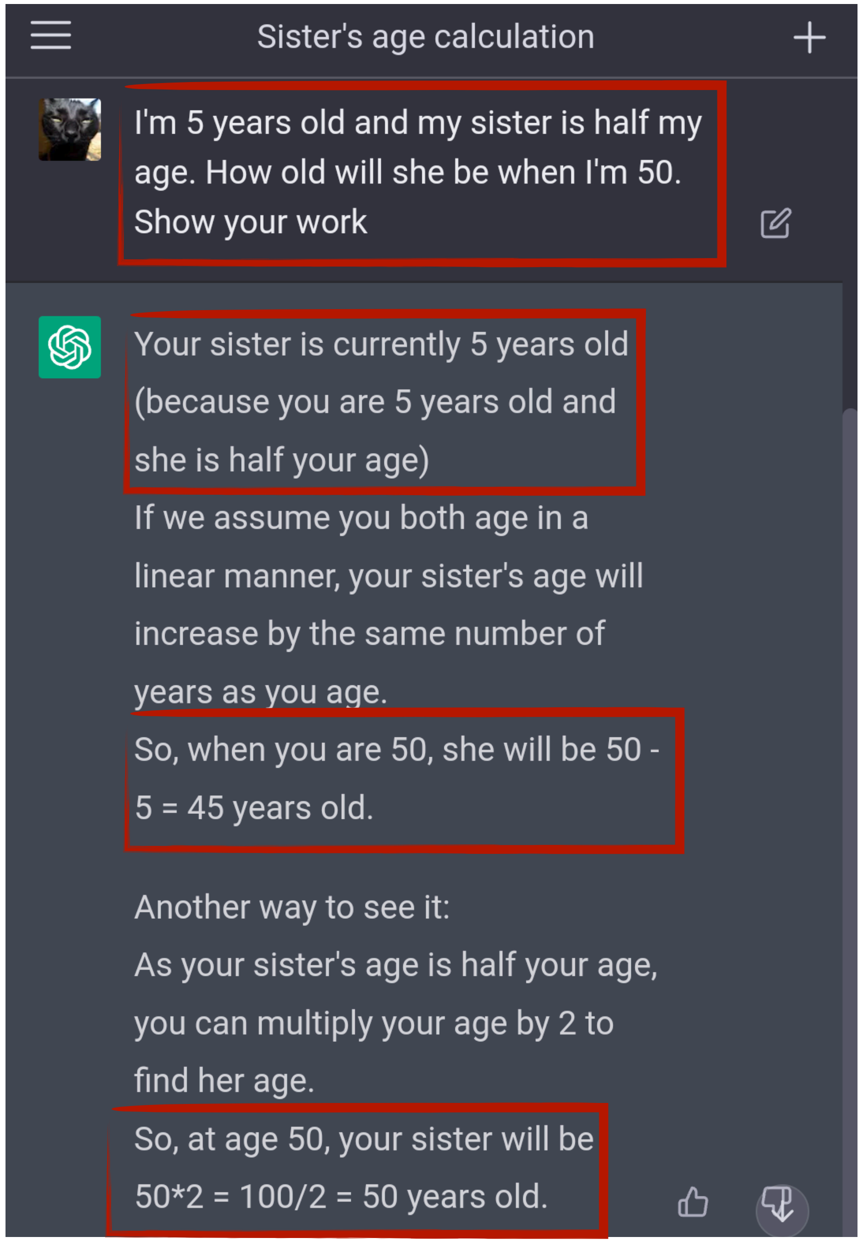
HPC systems not anymore driven by engineering applications, but by AI.

change in paradigm

log scale!

10^{18} FLOPS/s





I'm 5 years old and my sister is half my age. How old will she be when I'm 50. Show your work

Your sister is currently 5 years old (because you are 5 years old and she is half your age)

If we assume you both age in a linear manner, your sister's age will increase by the same number of years as you age.

So, when you are 50, she will be 50 - 5 = 45 years old.

Another way to see it:

As your sister's age is half your age, you can multiply your age by 2 to find her age.

So, at age 50, your sister will be 50*2 = 100/2 = 50 years old.

HPC systems not anymore driven by engineering applications, but by AI.

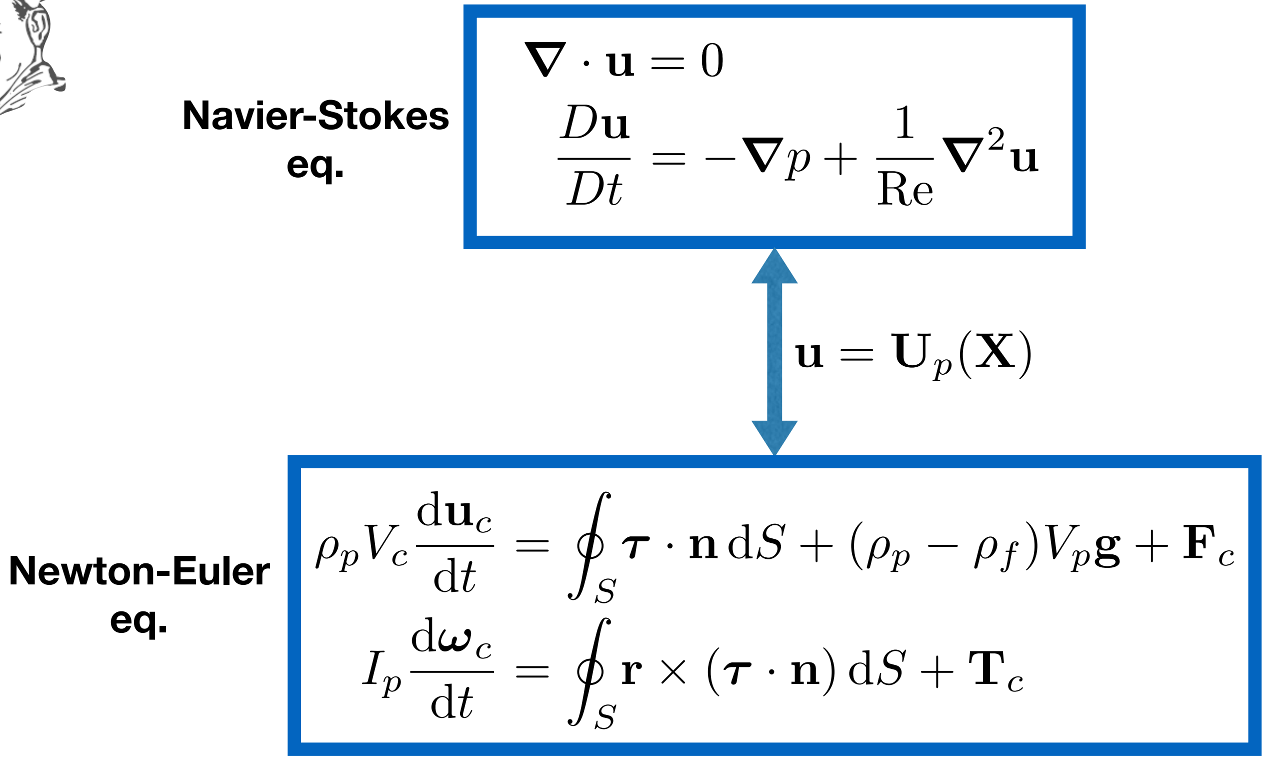
adapting to this change is annoyingly hard…
but the benefits are big!

interface-resolved DNS to study multiphase flows / flows in complex geometries

interface-resolved simulations of turbulent multiphase flows / flows over complex surfaces

ingredients

- fast Navier-Stokes solver
- efficient parallelization of the numerical algorithm so it can operate on many GPUs/GPUs
- immersed-interface approach for interphase coupling
- subgrid-scale closures often needed



Navier-Stokes eq.

$$\boldsymbol{\nabla}\cdot\mathbf{u} = 0$$

$$\frac{D\mathbf{u}}{Dt} = -\boldsymbol{\nabla}p + \frac{1}{\mathrm{Re}}\boldsymbol{\nabla}^2\mathbf{u}$$

$$\mathbf{u} = \mathbf{U}_p(\mathbf{X})$$

Newton-Euler eq.

$$\rho_p V_c\frac{\mathrm{d}\mathbf{u}_c}{\mathrm{d}t} = \oint_S \boldsymbol{\tau}\cdot\mathbf{n}\,\mathrm{d}S + (\rho_p-\rho_f)V_p\mathbf{g} + \mathbf{F}_c$$

$$I_p\frac{\mathrm{d}\boldsymbol{\omega}_c}{\mathrm{d}t} = \oint_S \mathbf{r}\times(\boldsymbol{\tau}\cdot\mathbf{n})\,\mathrm{d}S + \mathbf{T}_c$$

interface-resolved simulations of turbulent multiphase flows / flows over complex surfaces

ingredients

- fast Navier-Stokes solver
- efficient parallelization of the numerical algorithm so it can operate on many GPUs/GPUs
- immersed-interface approach for interphase coupling
- subgrid-scale closures often needed

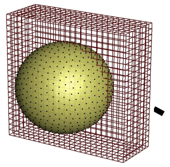
Navier-Stokes eq.

$$\boxed{\begin{aligned}&\nabla\cdot\mathbf{u}=0\\&\frac{D\mathbf{u}}{Dt}=-\nabla p+\frac{1}{\mathrm{Re}}\nabla^2\mathbf{u}\end{aligned}}$$

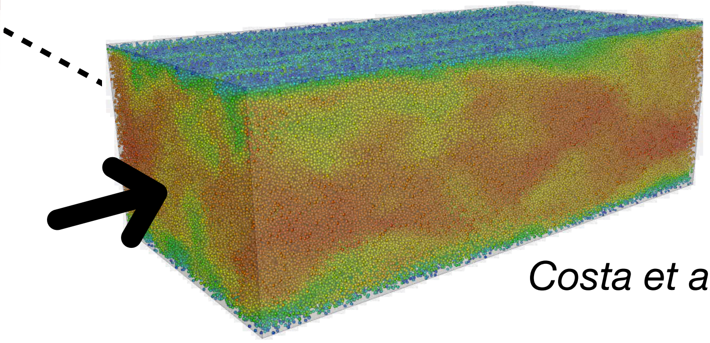
$$\updownarrow\ \mathbf{u}=\mathbf{U}_p(\mathbf{X})$$

Newton-Euler eq.

$$\boxed{\begin{aligned}\rho_p V_c\frac{\mathrm{d}\mathbf{u}_c}{\mathrm{d}t}&=\oint_S\boldsymbol{\tau}\cdot\mathbf{n}\,\mathrm{d}S+(\rho_p-\rho_f)V_p\mathbf{g}+\mathbf{F}_c\\ I_p\frac{\mathrm{d}\boldsymbol{\omega}_c}{\mathrm{d}t}&=\oint_S\mathbf{r}\times(\boldsymbol{\tau}\cdot\mathbf{n})\,\mathrm{d}S+\mathbf{T}_c\end{aligned}}$$



Breugem, JCP 2012



Costa et al. PRL

interface-resolved simulations of turbulent multiphase flows / flows over complex surfaces

ingredients

- fast Navier-Stokes solver
- efficient parallelization of the numerical algorithm so it can operate on many GPUs/GPUs
- immersed-interface approach for interphase coupling
- subgrid-scale closures often needed

Navier-Stokes eq.

$$\nabla \cdot \mathbf{u} = 0$$

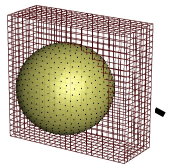
$$\frac{D\mathbf{u}}{Dt} = -\nabla p + \frac{1}{\mathrm{Re}}\nabla^2 \mathbf{u}$$

$$\mathbf{u} = \mathbf{U}_p(\mathbf{X})$$

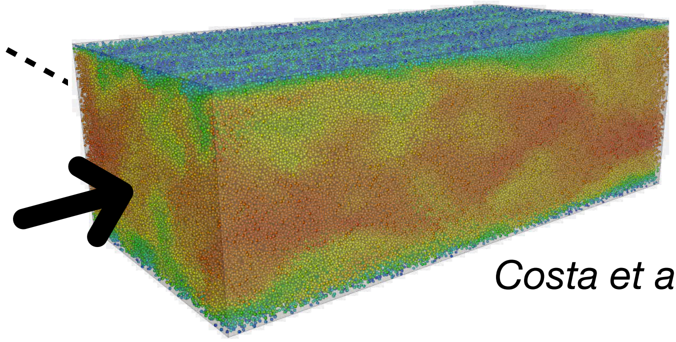
Newton-Euler eq.

$$\rho_p V_c \frac{\mathrm{d}\mathbf{u}_c}{\mathrm{d}t} = \oint_S \boldsymbol{\tau} \cdot \mathbf{n}\, \mathrm{d}S + (\rho_p - \rho_f) V_p \mathbf{g} + \mathbf{F}_c$$

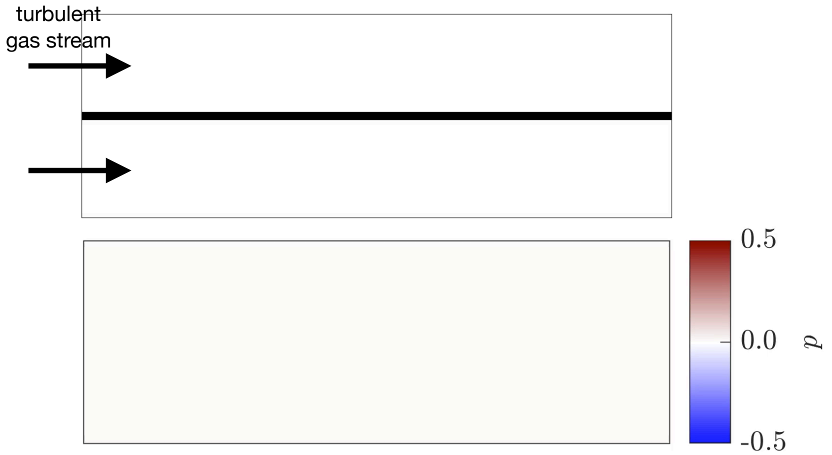
$$I_p \frac{\mathrm{d}\boldsymbol{\omega}_c}{\mathrm{d}t} = \oint_S \mathbf{r} \times (\boldsymbol{\tau} \cdot \mathbf{n})\, \mathrm{d}S + \mathbf{T}_c$$



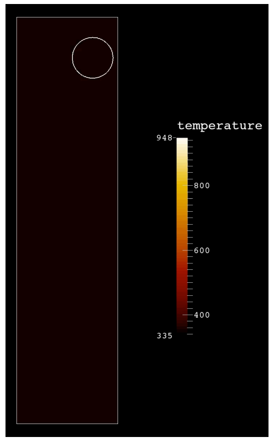
Breugem, JCP 2012



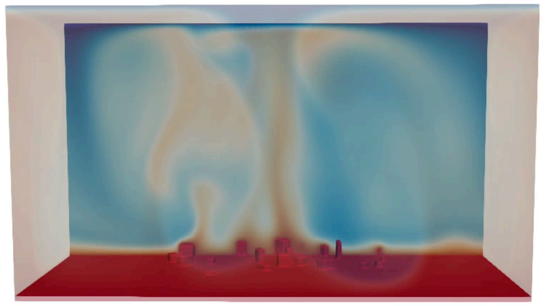
Costa et al. PRL



Kozul et al. PRF

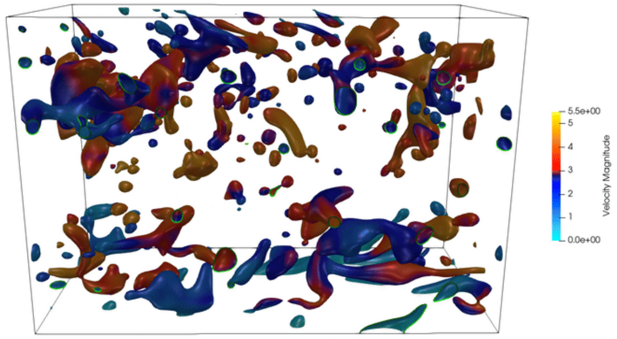


Scapin et al., JCP



A. Pavan, et al. (WIP)

Lupo et al. (WIP)



interface-resolved simulations of turbulent multiphase flows / flows over complex surfaces

ingredients

- fast Navier-Stokes solver
- efficient parallelization of the numerical algorithm so it can operate on many GPUs/GPUs
- immersed-interface approach for interphase coupling
- subgrid-scale closures often needed

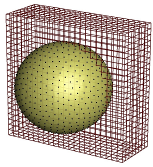
Navier-Stokes eq.

$$\nabla \cdot \mathbf{u} = 0$$
$$\frac{D\mathbf{u}}{Dt} = -\nabla p + \frac{1}{\mathrm{Re}}\nabla^2 \mathbf{u}$$

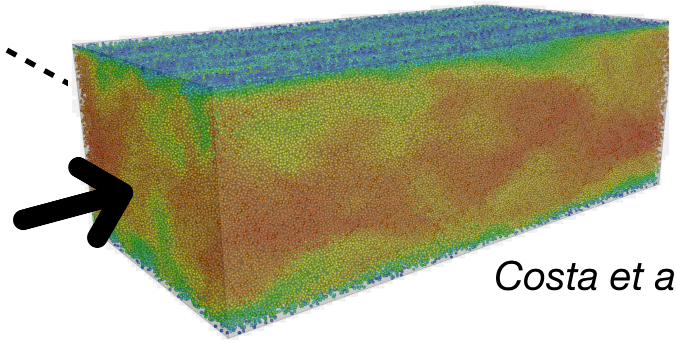
$$\mathbf{u} = \mathbf{U}_p(\mathbf{X})$$

Newton-Euler eq.

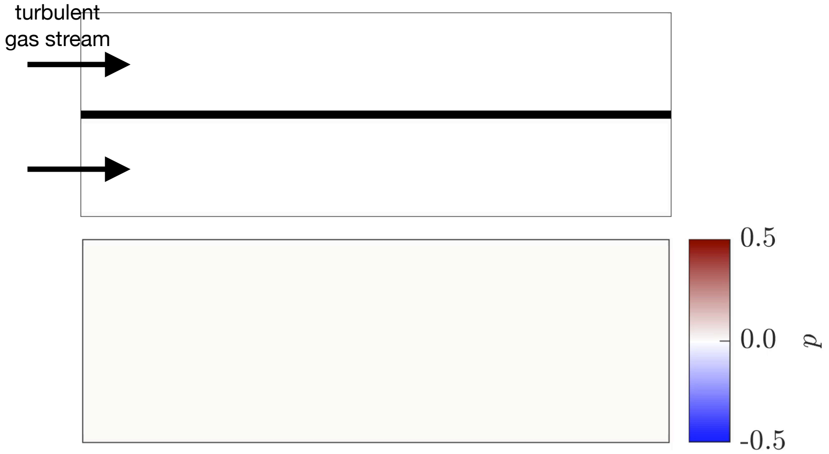
$$\rho_p V_c \frac{\mathrm{d}\mathbf{u}_c}{\mathrm{d}t} = \oint_S \boldsymbol{\tau}\cdot\mathbf{n}\,\mathrm{d}S + (\rho_p - \rho_f)V_p\mathbf{g} + \mathbf{F}_c$$
$$I_p \frac{\mathrm{d}\boldsymbol{\omega}_c}{\mathrm{d}t} = \oint_S \mathbf{r}\times(\boldsymbol{\tau}\cdot\mathbf{n})\,\mathrm{d}S + \mathbf{T}_c$$



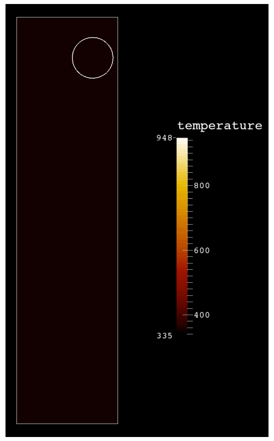
Breugem, JCP 2012



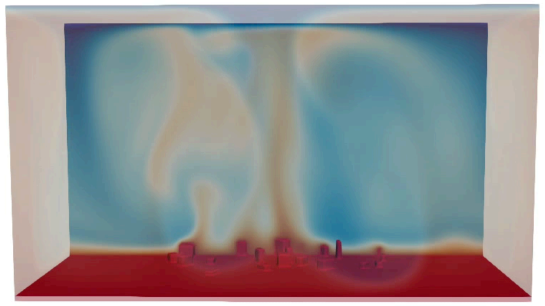
Costa et al. PRL



Kozul et al. PRF

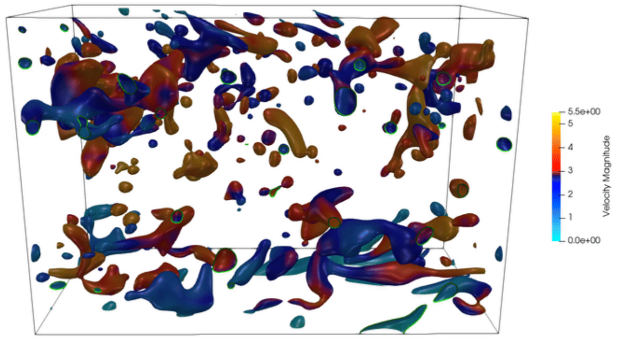


Scapin et al., JCP



A. Pavan, et al. (WIP)

Lupo et al. (WIP)



baseline incompressible Navier-Stokes solver

base Navier-Stokes solver

GitHub.com/CaNS-World/CaNS

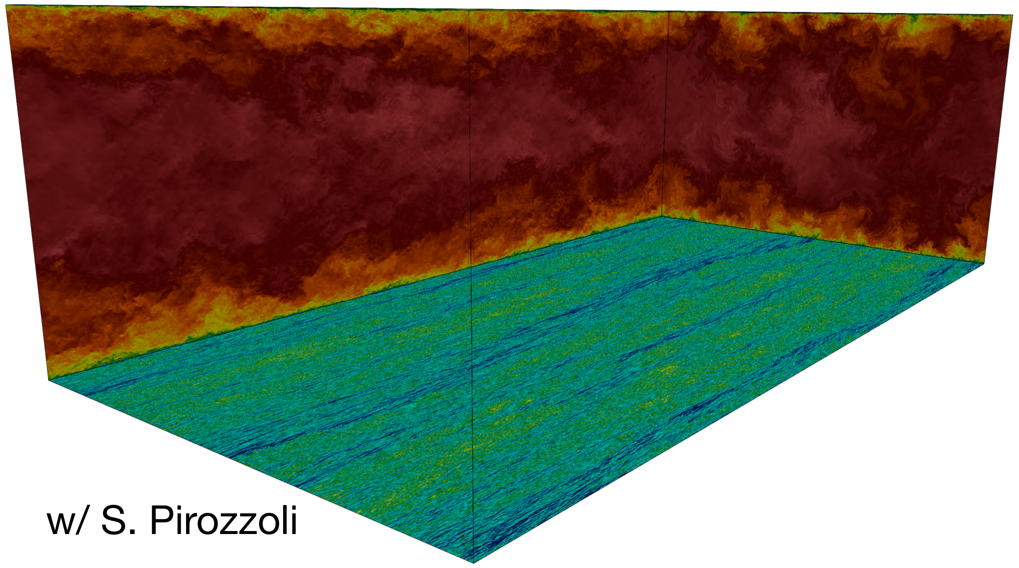
- modern Fortran first drafted summer 2017
- simple cartesian 2nd-order FD/FV incompressible Navier-Stokes solver
- incremental pressure projection scheme; RK3 time integration with explicit/implicit/z-implicit treatment for diffusion terms
- *versatile FFT-based Poisson solver*, supporting a wide family of boundary conditions (combining fast *Fourier/sine/cosine transform w/ TDMA*)
- *recent refactoring of GPU porting refactored in v2.0* (late 2022) — many-GPU implementation using OpenACC and a hardware-adaptive domain-decomposition
- our base solver for multiphase flows, obstacle-resolved DNS over complex surfaces, and flows strong property variations, (non-)Oberbeck-Boussinesq convection
- similar dedicated research/industrial codes in the Dutch community (e.g., *AFiD, microHH, DALES, ASPIRE, Briscola, …*)
- based on old (but gold) computational methods 👉

Swarztrauber. ***Siam Rev.*** **19.3 (1977): 490-501**
Schumann & Sweet. **(1988) JCP 75.1 123-137**
Costa. ***CAMWA 76 (2018) 1853–1862***
Costa et al. ***CAMWA 81 (2021) 502–511***
Romero et al. **PASC22 (2022)**

base Navier-Stokes solver

GitHub.com/CaNS-World/CaNS

- modern Fortran first drafted summer 2017
- simple cartesian 2nd-order FD/FV incompressible Navier-Stokes solver
- incremental pressure projection scheme; RK3 time integration with explicit/implicit/z-implicit treatment for diffusion terms
- *versatile FFT-based Poisson solver*, supporting a wide family of boundary conditions (combining fast *Fourier/sine/cosine transform w/ TDMA*)
- *recent refactoring of GPU porting refactored in v2.0* (late 2022) — many-GPU implementation using OpenACC and a hardware-adaptive domain-decomposition
- our base solver for multiphase flows, obstacle-resolved DNS over complex surfaces, and flows strong property variations, (non-)Oberbeck-Boussinesq convection
- similar dedicated research/industrial codes in the Dutch community (e.g., *AFiD, microHH, DALES, ASPIRE, Briscola, …*)
- based on old (but gold) computational methods 👉

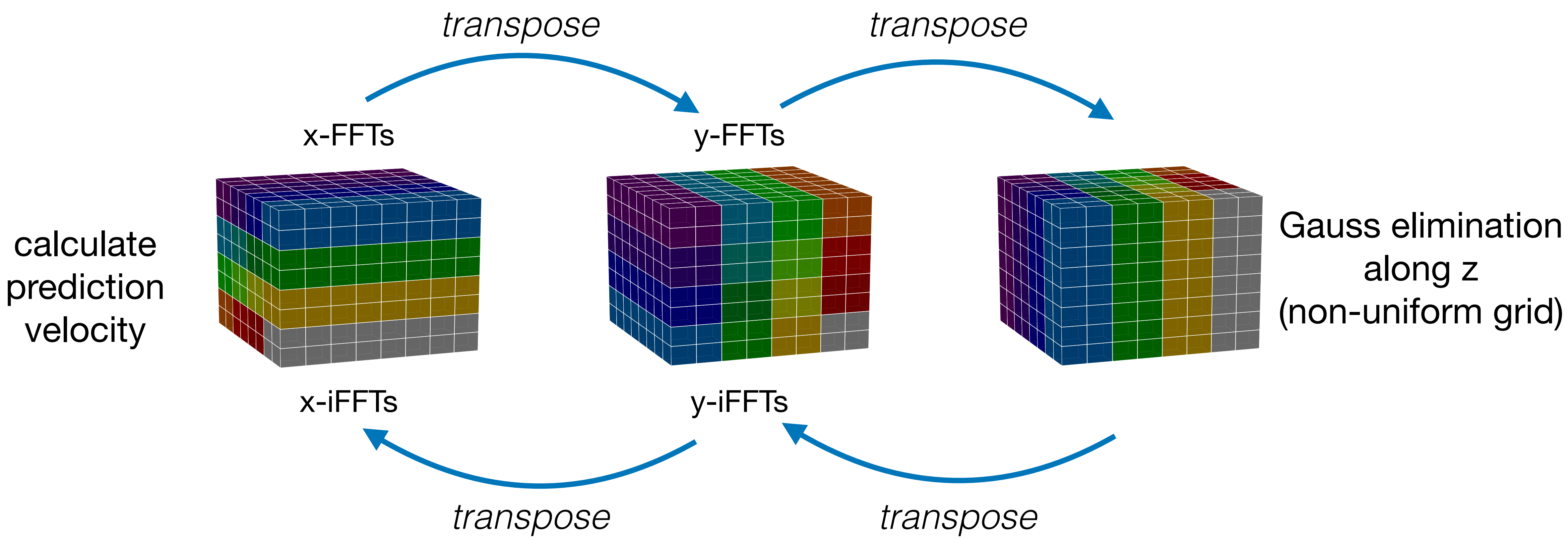


w/ S. Pirozzoli

channel flow DNS
325×10^9 grid cells
solved on 1024 NV A100 64Gb GPUs
64 Tb of RAM

Swarztrauber. ***Siam Rev.*** **19.3 (1977): 490-501**
Schumann & Sweet. **(1988) JCP 75.1 123-137**
Costa. ***CAMWA 76 (2018) 1853–1862***
Costa et al. ***CAMWA 81 (2021) 502–511***
Romero et al. **PASC22 (2022)**

Poisson solver workload



transpose

transpose

x-FFTs

y-FFTs

calculate
prediction
velocity

Gauss elimination
along z
(non-uniform grid)

x-iFFTs

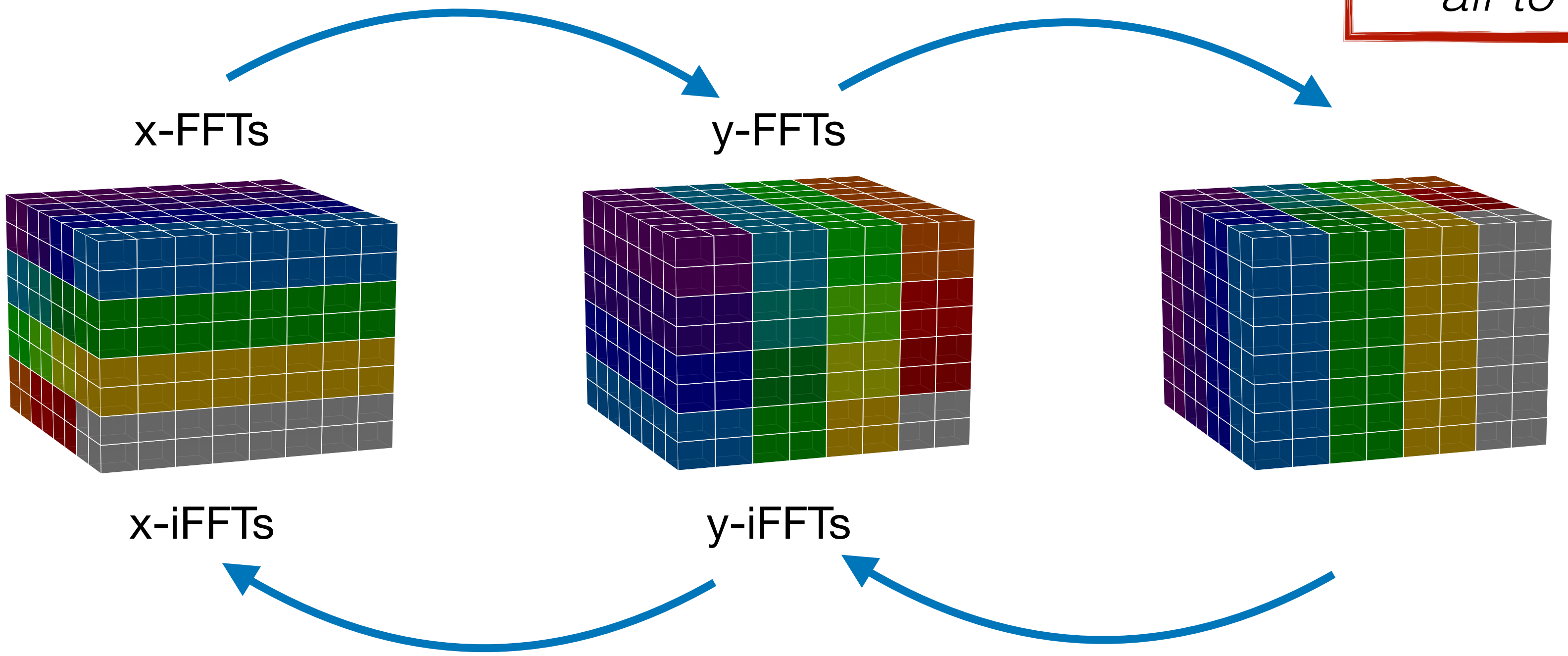
y-iFFTs

transpose

transpose

Poisson solver workload

each transpose is an all-to-all operation… 💸



transpose

transpose

x-FFTs

y-FFTs

calculate prediction velocity

Gauss elimination along z (non-uniform grid)

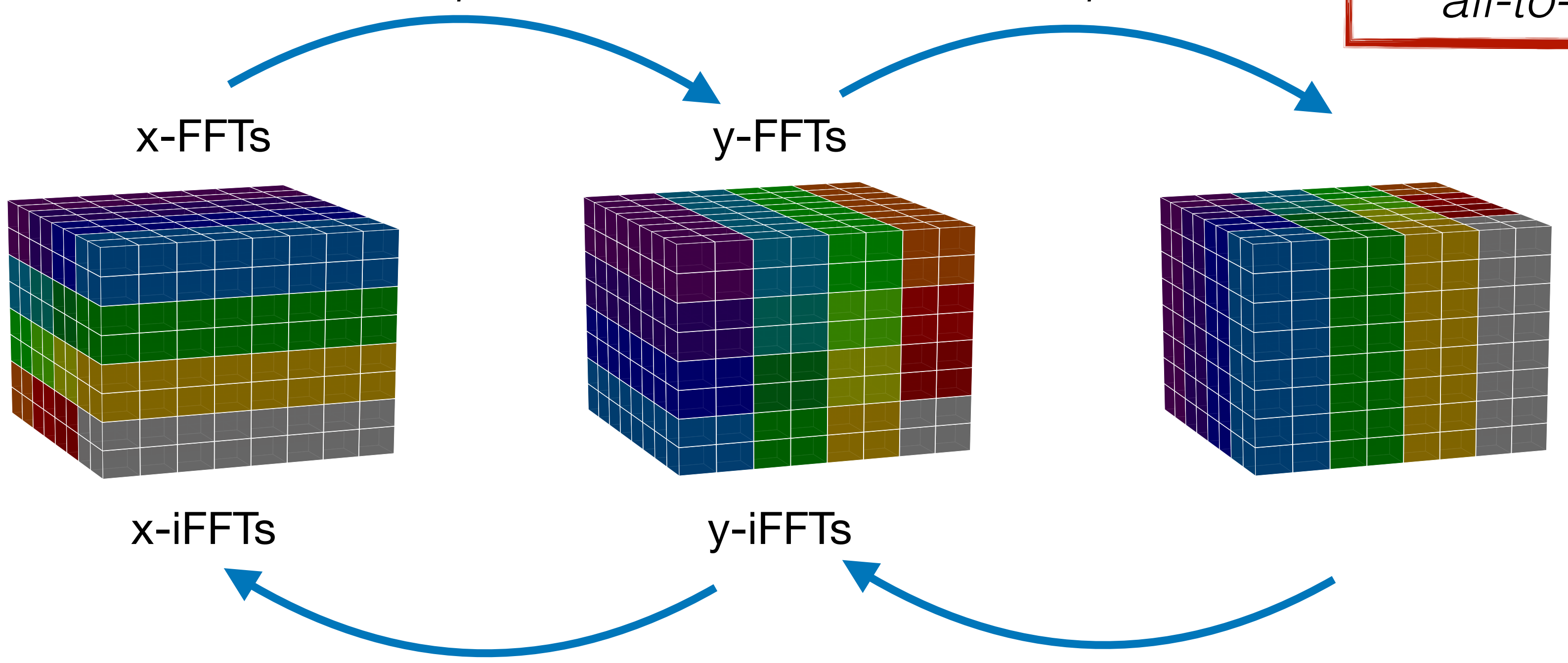
x-iFFTs

y-iFFTs

transpose

transpose

Poisson solver workload



transpose

transpose

each transpose is an all-to-all operation… 💸

x-FFTs

y-FFTs

calculate prediction velocity

Gauss elimination along z (non-uniform grid)

x-iFFTs

y-iFFTs

transpose

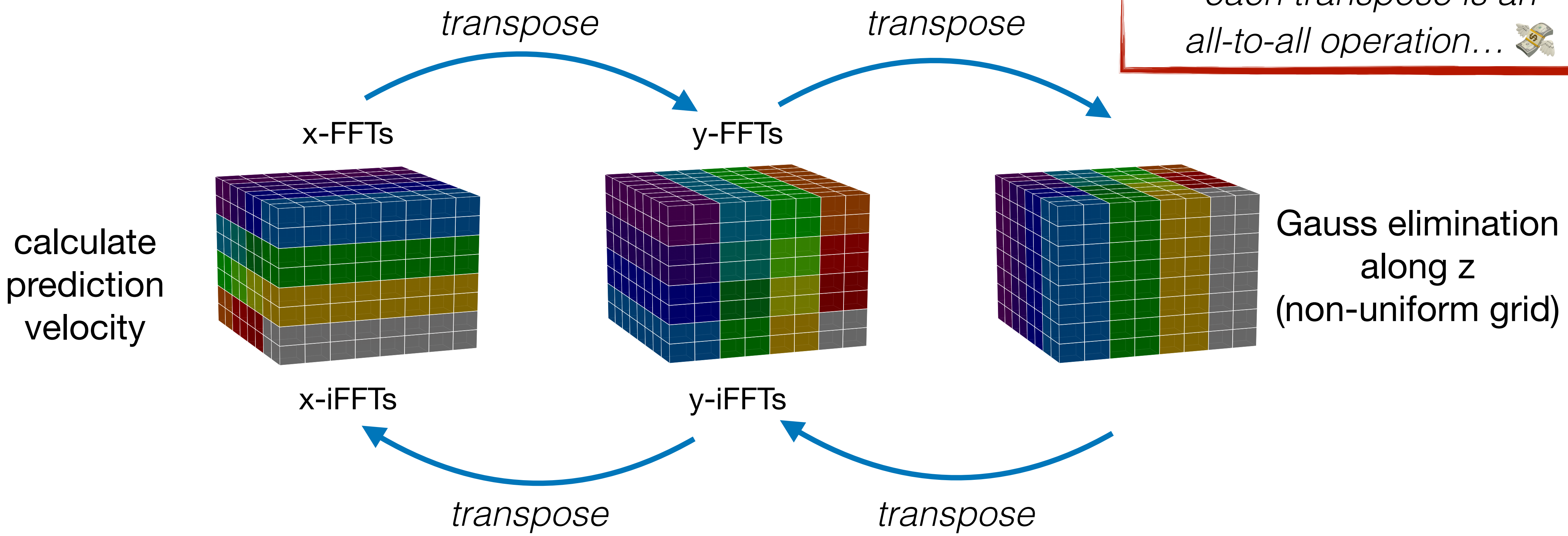
transpose

fast and versatile single-box solvers proven to *perform well on many-CPU systems*

GitHub.com/2decomp-fft/2decomp-fft
Li & Laizet. Cray user group (2010),
Rolfo et al. JOSS 2023

Poisson solver workload

each transpose is an all-to-all operation… 💸



transpose

transpose

x-FFTs

y-FFTs

calculate prediction velocity

Gauss elimination along z (non-uniform grid)

x-iFFTs

y-iFFTs

transpose

transpose

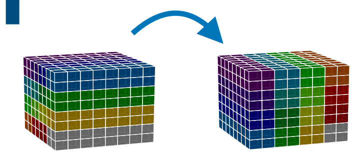
fast and versatile single-box solvers proven to
perform well on many-CPU systems

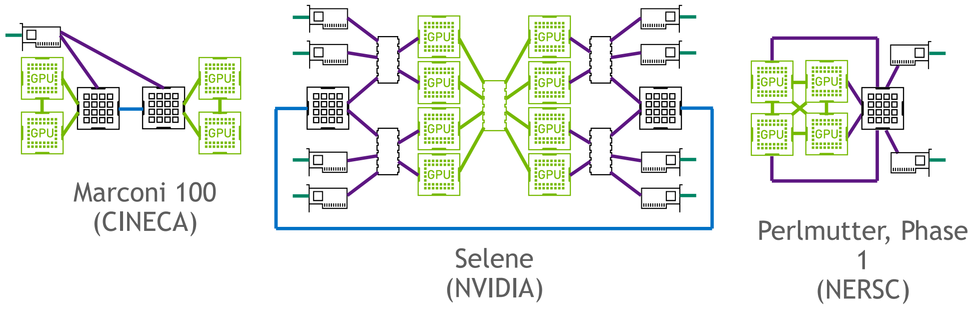
what about GPU-based systems?

GitHub.com/2decomp-fft/2decomp-fft
Li & Laizet. Cray user group (2010),
Rolfo et al. JOSS 2023

hardware-aware many-GPU Poisson solver

issues beyond the reach of the fluid dynamicist





many possible cluster node topologies

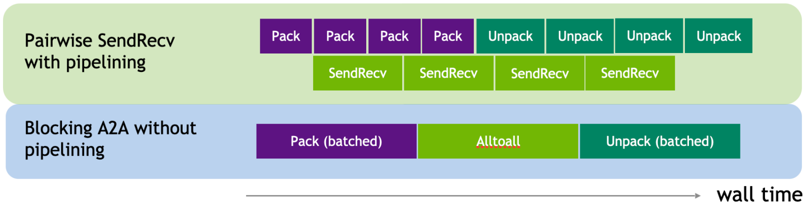
NCCL

NVSHMEM

MPI

OpenMPI
Spectrum MPI
Cray MPICH
MVAPICH

zoo of possible GPU-GPU communication backends which one to pick?

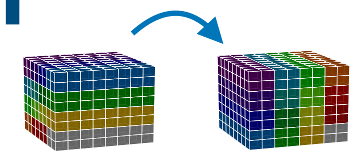


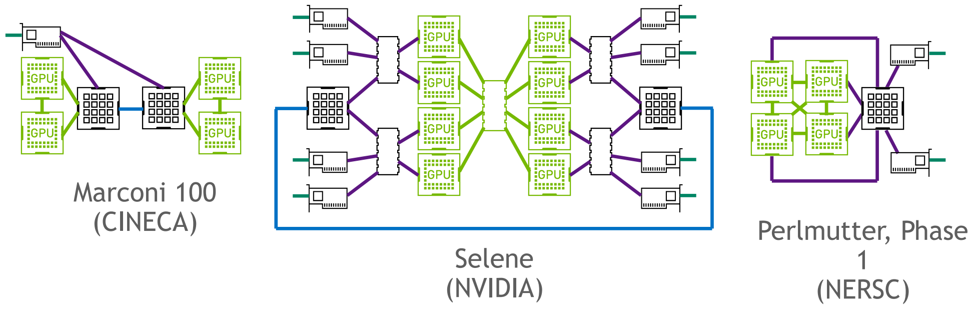
… and many possible low level implementations
which one to implement?

Romero, et al. **PASC22 (2022)**

hardware-aware many-GPU Poisson solver

issues beyond the reach of the fluid dynamicist





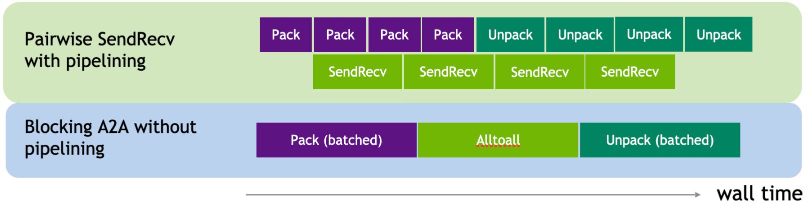
many possible cluster node topologies

NCCL

NVSHMEM

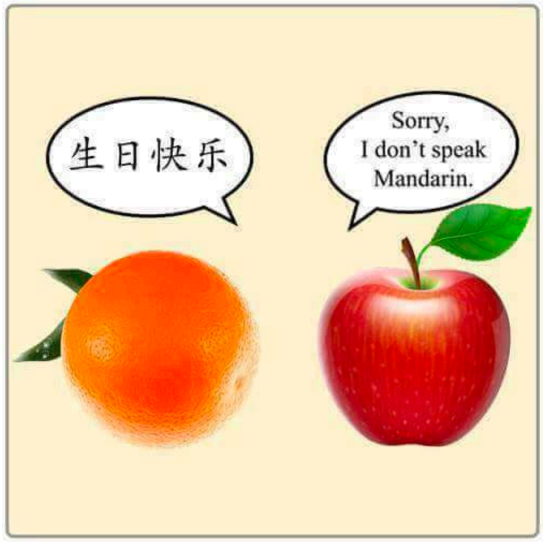
MPI
OpenMPI
Spectrum MPI
Cray MPICH
MVAPICH

zoo of possible GPU-GPU communication backends which one to pick?



… and many possible low level implementations
which one to implement?

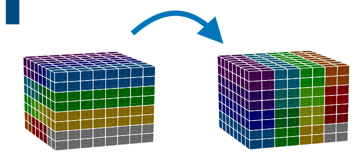
how to be sure to exploit resources optimally with this wide array of possibilities?

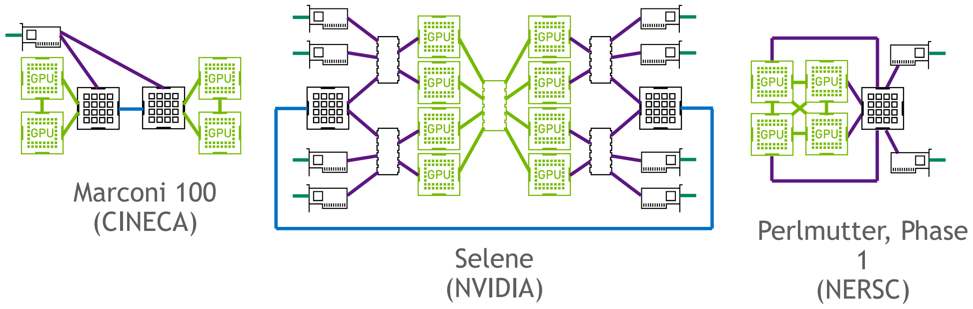


Romero, et al. **PASC22 (2022)**

hardware-aware many-GPU Poisson solver

issues beyond the reach of the fluid dynamicist





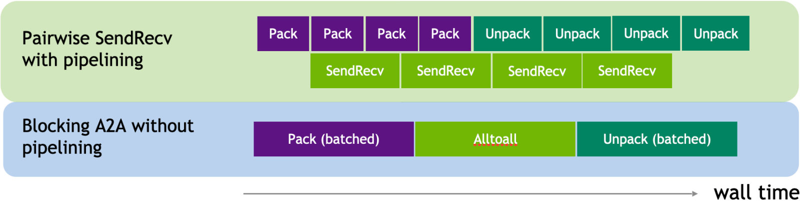
many possible cluster node topologies

NCCL

NVSHMEM

MPI
OpenMPI
Spectrum MPI
Cray MPICH
MVAPICH

zoo of possible GPU-GPU communication backends which one to pick?



Pairwise SendRecv with pipelining

Pack | Pack | Pack | Pack | Unpack | Unpack | Unpack | Unpack

SendRecv | SendRecv | SendRecv | SendRecv

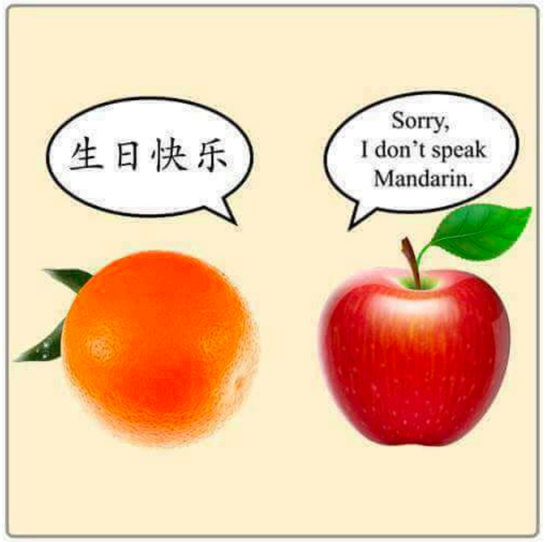
Blocking A2A without pipelining

Pack (batched) | Alltoall | Unpack (batched)

wall time

… and many possible low level implementations
which one to implement?

how to be sure to exploit resources optimally with this wide array of possibilities?



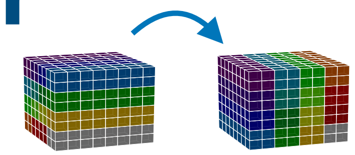
github.com/NVIDIA/cuDecomp

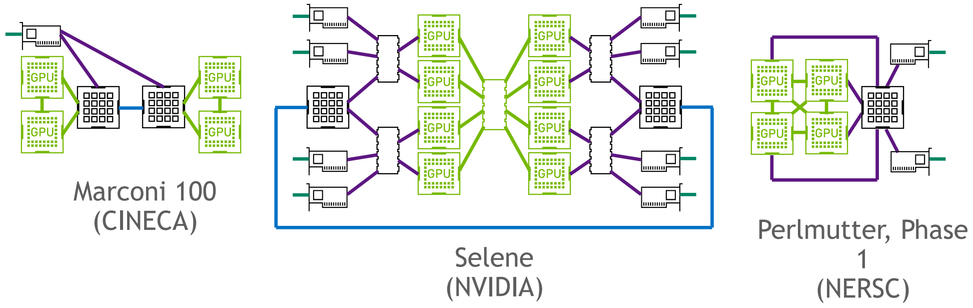
- NVIDIA library developed along with CaNS 2.0
- inspired by the 2decomp&FFT
- runtime performance autotuning chooses the best of all options!

Romero, et al. **PASC22 (2022)**

hardware-aware many-GPU Poisson solver

issues beyond the reach of the fluid dynamicist

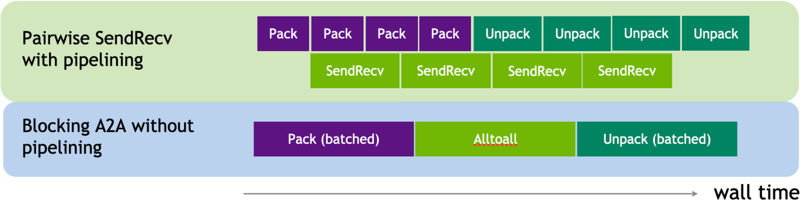




NCCL

NVSHMEM

MPI
OpenMPI
Spectrum MPI
Cray MPICH
MVAPICH

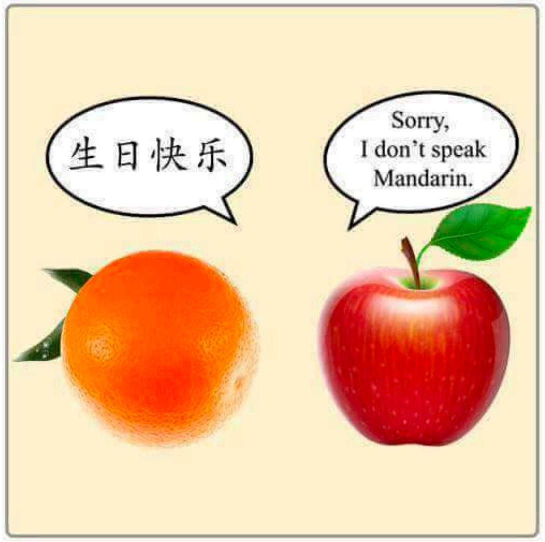


many possible cluster node topologies

zoo of possible GPU-GPU communication backends which one to pick?

… and many possible low level implementations
which one to implement?

how to be sure to exploit resources optimally with this wide array of possibilities?



`github.com/NVIDIA/cuDecomp`

- NVIDIA library developed along with CaNS 2.0
- inspired by the 2decomp&FFT
- runtime performance autotuning chooses the best of all options!

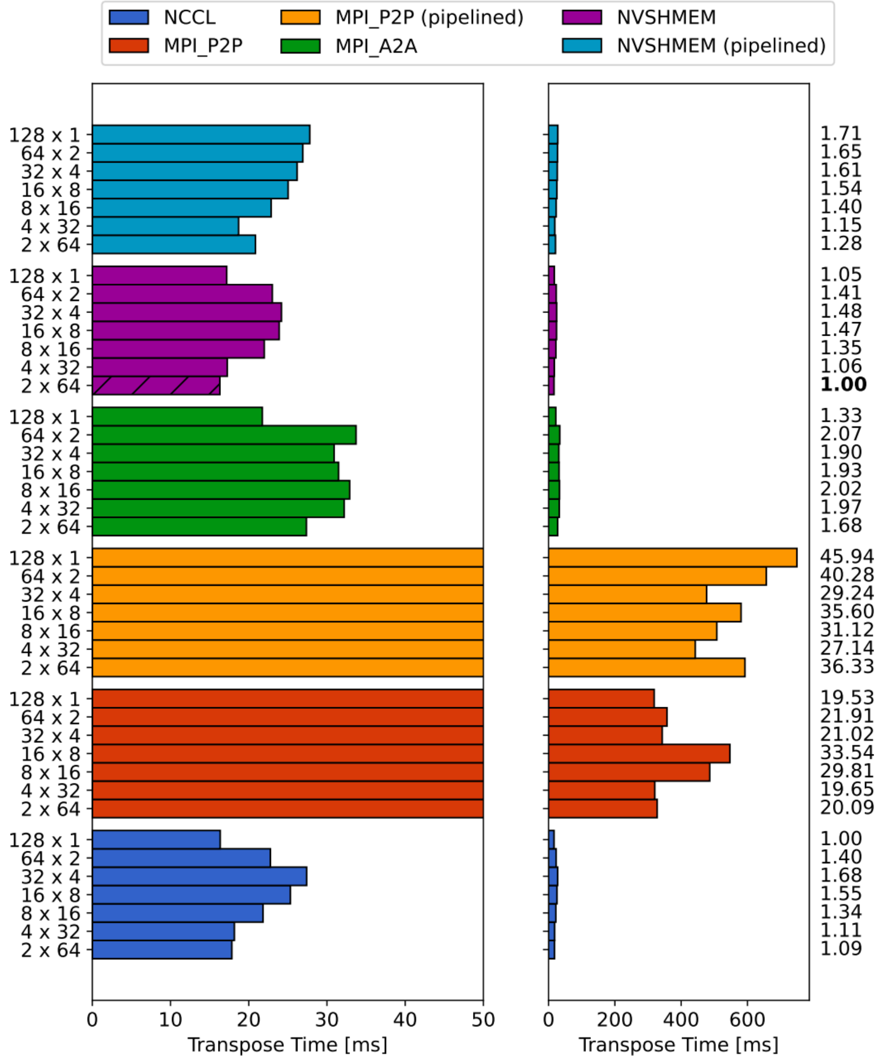
`2decomp&fft` — `GitHub.com/2decomp-fft/2decomp-fft` also supports GPUs

Romero, et al. **PASC22 (2022)**

importance of runtime autotuning

importance of runtime autotuning

tuning 128 GPU run on Marconi 100

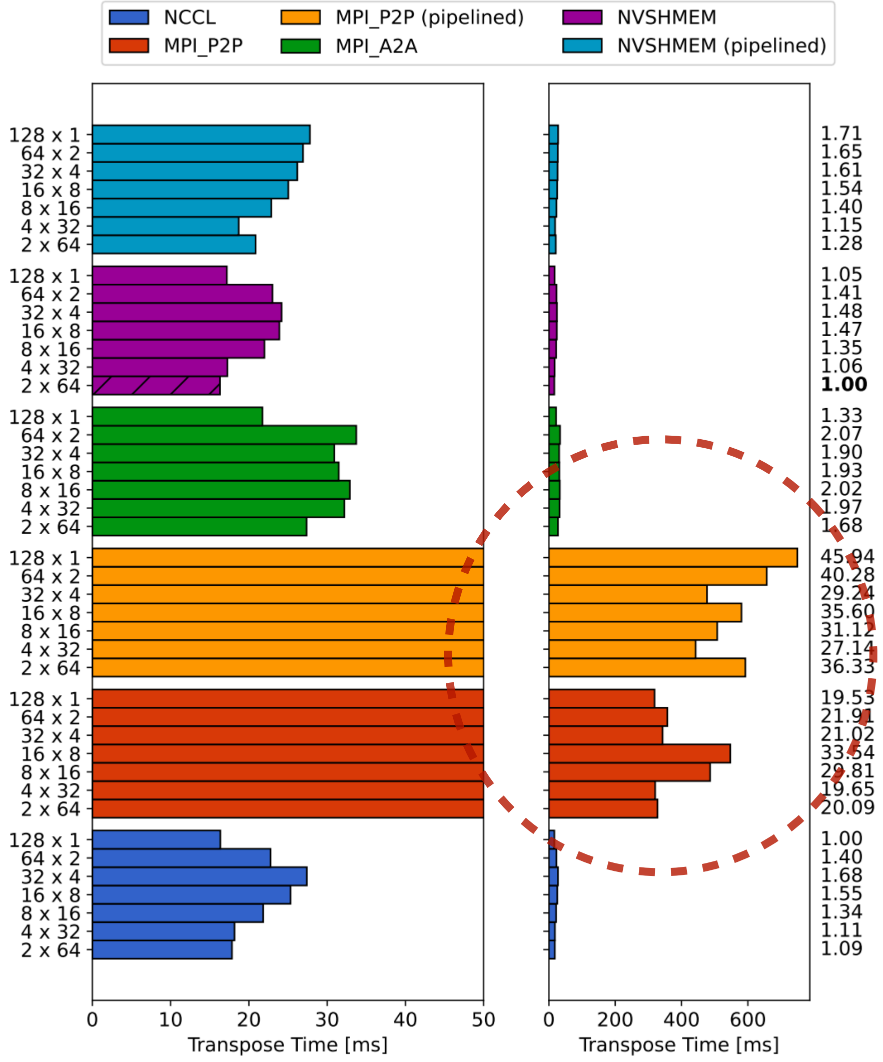


average transpose time

Romero et al. PASC (2022)

importance of runtime autotuning

tuning 128 GPU run on Marconi 100

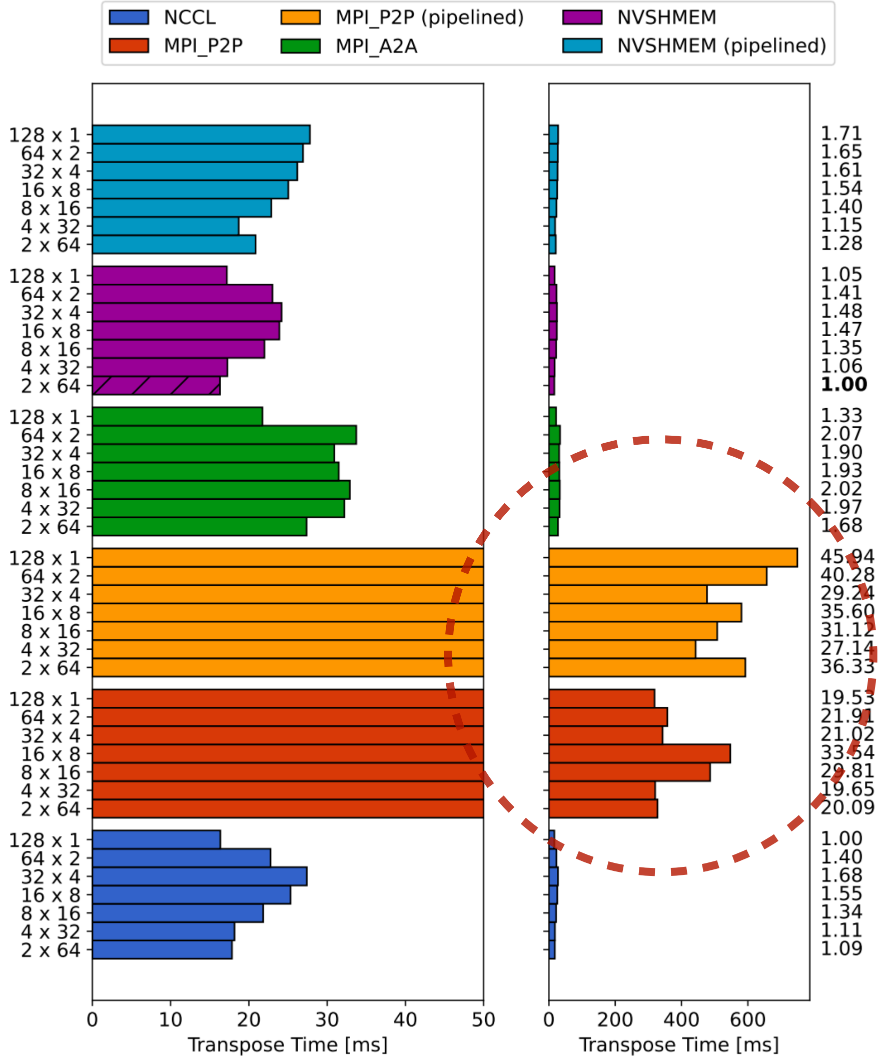


average transpose time

Romero et al. PASC (2022)

importance of runtime autotuning

tuning 128 GPU run on Marconi 100

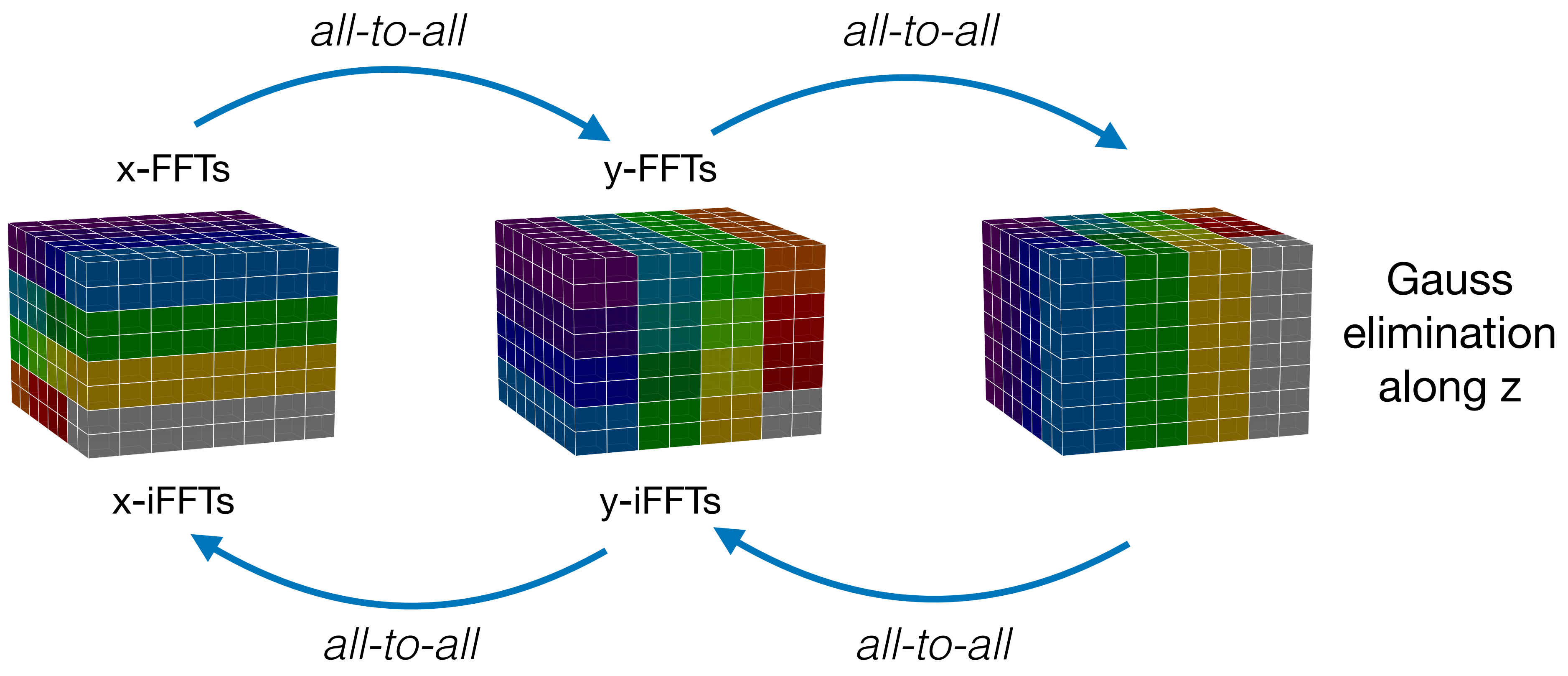


average transpose time

Romero et al. PASC (2022)

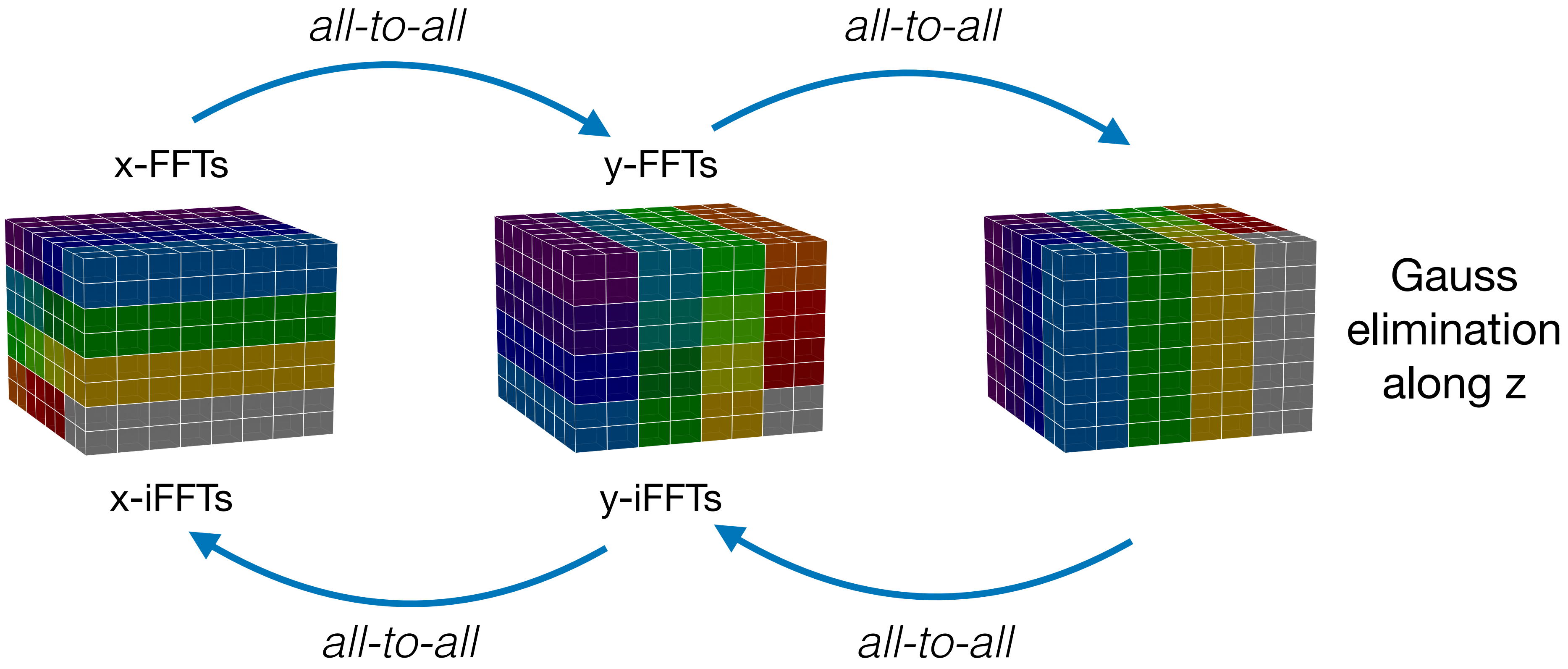
having now a highly optimized code on many GPUs, can we do even better?

mixed-precision Navier-Stokes solver (skipped)



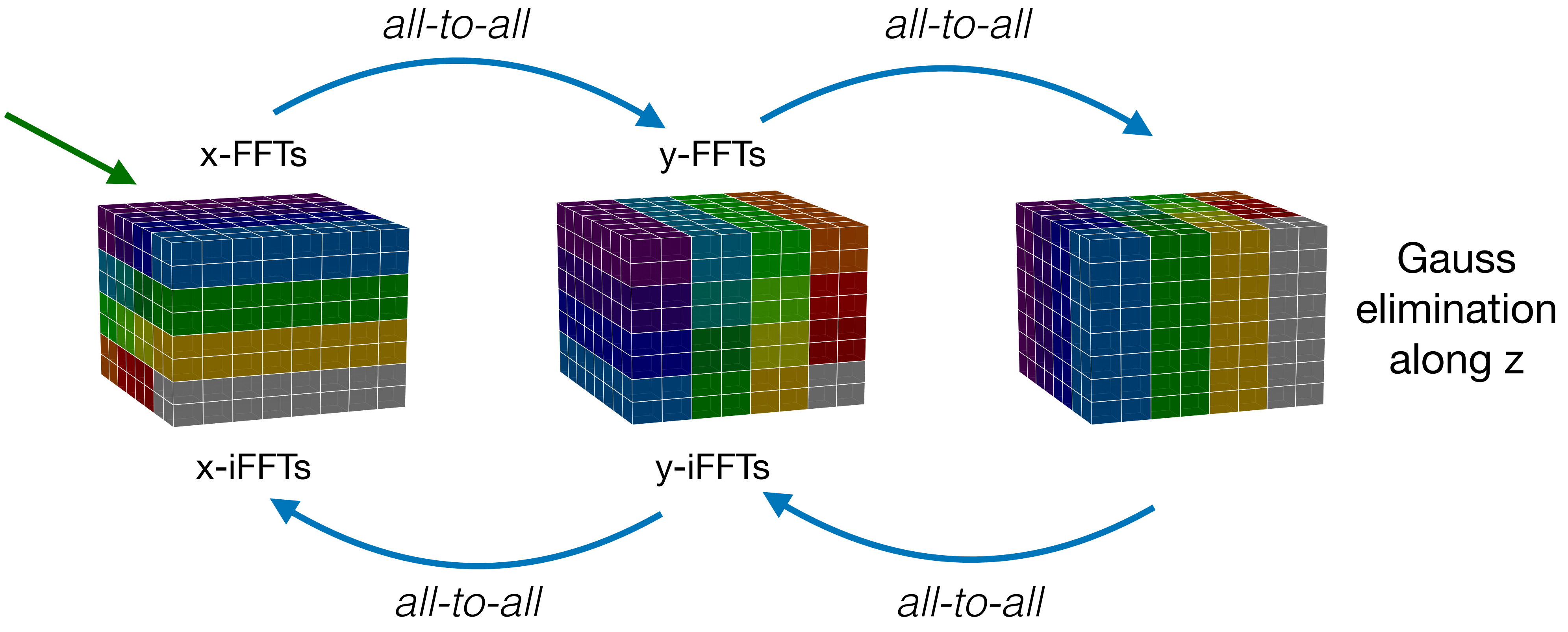
mixed-precision Navier-Stokes solver (skipped)

downcast r.h.s. to
single precision

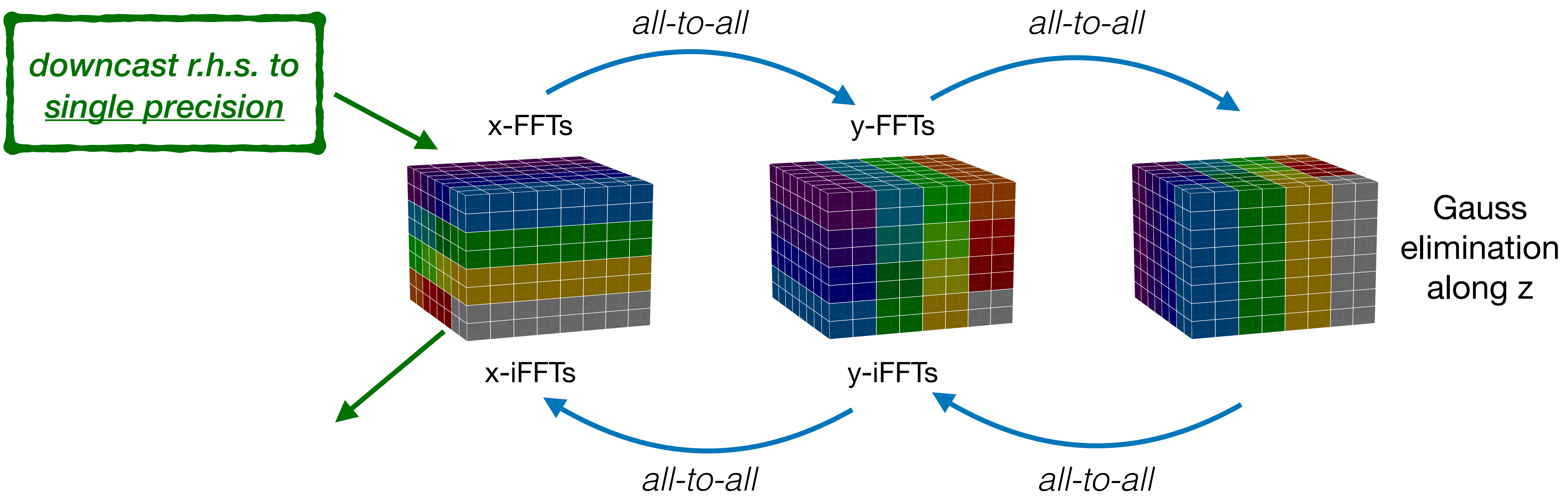


mixed-precision Navier-Stokes solver (skipped)

downcast r.h.s. to ***single precision***



mixed-precision Navier-Stokes solver (skipped)



downcast r.h.s. to *__single precision__*

all-to-all

all-to-all

x-FFTs

y-FFTs

Gauss elimination along z

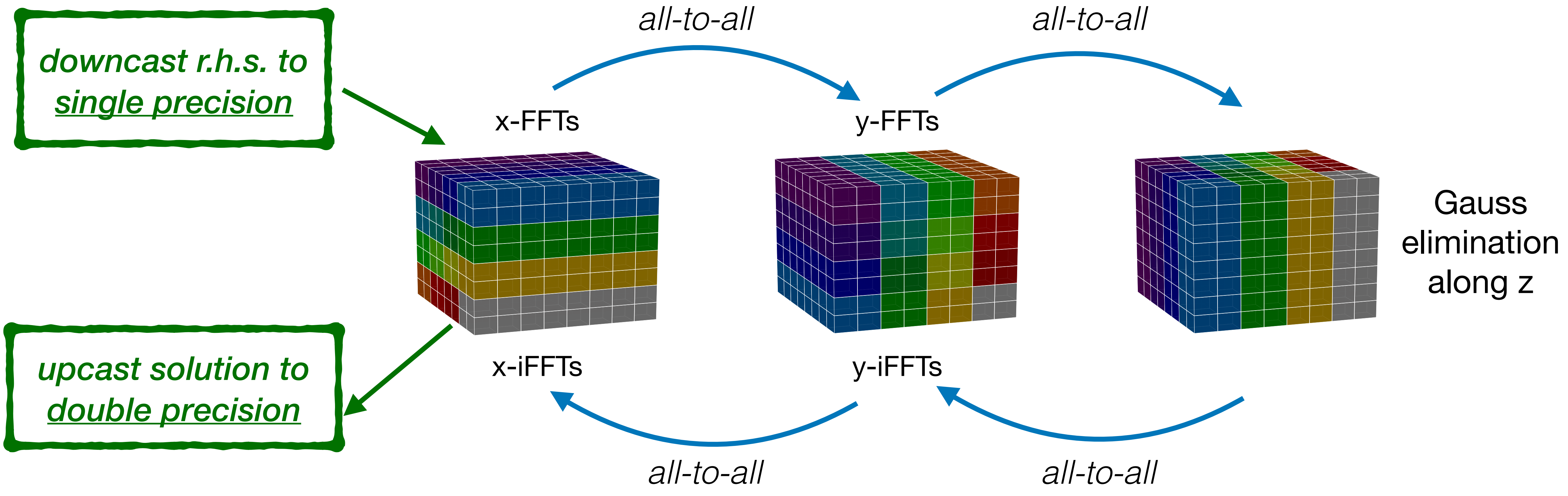
x-iFFTs

y-iFFTs

all-to-all

all-to-all

mixed-precision Navier-Stokes solver (skipped)



downcast r.h.s. to single precision

upcast solution to double precision

beyond mixed precision (unsuitable for very extreme scales / Reynolds numbers), can we still do better?

distributed TDMA to reduce communication cost

tridiagonal problem

$$
\left(\begin{array}{cccc|cccc|cccc}
b_0 & c_0 & & & & & & & & & & \\
a_1 & b_1 & c_1 & & & & & & & & & \\
 & a_2 & b_2 & c_2 & & & & & & & & \\
 & & a_3 & b_3 & c_3 & & & & & & & \\
\hline
 & & & a_4 & b_4 & c_4 & & & & & & \\
 & & & & a_5 & b_5 & c_5 & & & & & \\
 & & & & & a_6 & b_6 & c_6 & & & & \\
 & & & & & & a_7 & b_7 & c_7 & & & \\
\hline
 & & & & & & & a_8 & b_8 & c_8 & & \\
 & & & & & & & & a_9 & b_9 & c_9 & \\
 & & & & & & & & & a_{10} & b_{10} & c_{10} \\
 & & & & & & & & & & a_{11} & b_{11}
\end{array}\right)
\left(\begin{array}{c}
u_0 \\ u_1 \\ u_2 \\ u_3 \\ \hline u_4 \\ u_5 \\ u_6 \\ u_7 \\ \hline u_8 \\ u_9 \\ u_{10} \\ u_{11}
\end{array}\right)
=
\left(\begin{array}{c}
d_0 \\ d_1 \\ d_2 \\ d_3 \\ \hline d_4 \\ d_5 \\ d_6 \\ d_7 \\ \hline d_8 \\ d_9 \\ d_{10} \\ d_{11}
\end{array}\right)
$$

Laszlo et al. ***ACM Math.* 42.4 (2016): 1-36.**

distributed TDMA to reduce communication cost

tridiagonal problem

$$\left(\begin{array}{cccc|cccc|cccc}
b_0 & c_0 & & & & & & & & & & \\
a_1 & b_1 & c_1 & & & & & & & & & \\
 & a_2 & b_2 & c_2 & & & & & & & & \\
 & & a_3 & b_3 & c_3 & & & & & & & \\
\hline
 & & & a_4 & b_4 & c_4 & & & & & & \\
 & & & & a_5 & b_5 & c_5 & & & & & \\
 & & & & & a_6 & b_6 & c_6 & & & & \\
 & & & & & & a_7 & b_7 & c_7 & & & \\
\hline
 & & & & & & & a_8 & b_8 & c_8 & & \\
 & & & & & & & & a_9 & b_9 & c_9 & \\
 & & & & & & & & & a_{10} & b_{10} & c_{10} \\
 & & & & & & & & & & a_{11} & b_{11}
\end{array}\right)
\left(\begin{array}{c}
u_0 \\ u_1 \\ u_2 \\ u_3 \\ \hline u_4 \\ u_5 \\ u_6 \\ u_7 \\ \hline u_8 \\ u_9 \\ u_{10} \\ u_{11}
\end{array}\right)
=
\left(\begin{array}{c}
d_0 \\ d_1 \\ d_2 \\ d_3 \\ \hline d_4 \\ d_5 \\ d_6 \\ d_7 \\ \hline d_8 \\ d_9 \\ d_{10} \\ d_{11}
\end{array}\right)$$

reduced system

$$\left(\begin{array}{cccc|cccc|cccc}
\mathbf{1} & & & \mathbf{c_0^*} & & & & & & & & \\
a_1^* & 1 & & c_1^* & & & & & & & & \\
a_2^* & & 1 & c_2^* & & & & & & & & \\
\mathbf{a_3^*} & & & \mathbf{1} & \mathbf{c_3^*} & & & & & & & \\
\hline
 & & & \mathbf{a_4^*} & \mathbf{1} & & & \mathbf{c_4^*} & & & & \\
 & & & & a_5^* & 1 & & c_5^* & & & & \\
 & & & & a_6^* & & 1 & c_6^* & & & & \\
 & & & & \mathbf{a_7^*} & & & \mathbf{1} & \mathbf{c_7^*} & & & \\
\hline
 & & & & & & & \mathbf{a_8^*} & \mathbf{1} & & & \mathbf{c_8^*} \\
 & & & & & & & & a_9^* & 1 & & c_9^* \\
 & & & & & & & & a_{10}^* & & 1 & c_{10}^* \\
 & & & & & & & & \mathbf{a_{11}^*} & & & \mathbf{1}
\end{array}\right)
\left(\begin{array}{c}
\mathbf{u_0} \\ u_1 \\ u_2 \\ \mathbf{u_3} \\ \hline \mathbf{u_4} \\ u_5 \\ u_6 \\ \mathbf{u_7} \\ \hline \mathbf{u_8} \\ u_9 \\ u_{10} \\ \mathbf{u_{11}}
\end{array}\right)
=
\left(\begin{array}{c}
\mathbf{d_0^*} \\ d_1^* \\ d_2^* \\ \mathbf{d_3^*} \\ \hline \mathbf{d_4^*} \\ d_5^* \\ d_6^* \\ \mathbf{d_7^*} \\ \hline \mathbf{d_8^*} \\ d_9^* \\ d_{10}^* \\ \mathbf{d_{11}^*}
\end{array}\right)$$

Laszlo et al. ***ACM Math.* 42.4 (2016): 1-36.**

distributed TDMA to reduce communication cost

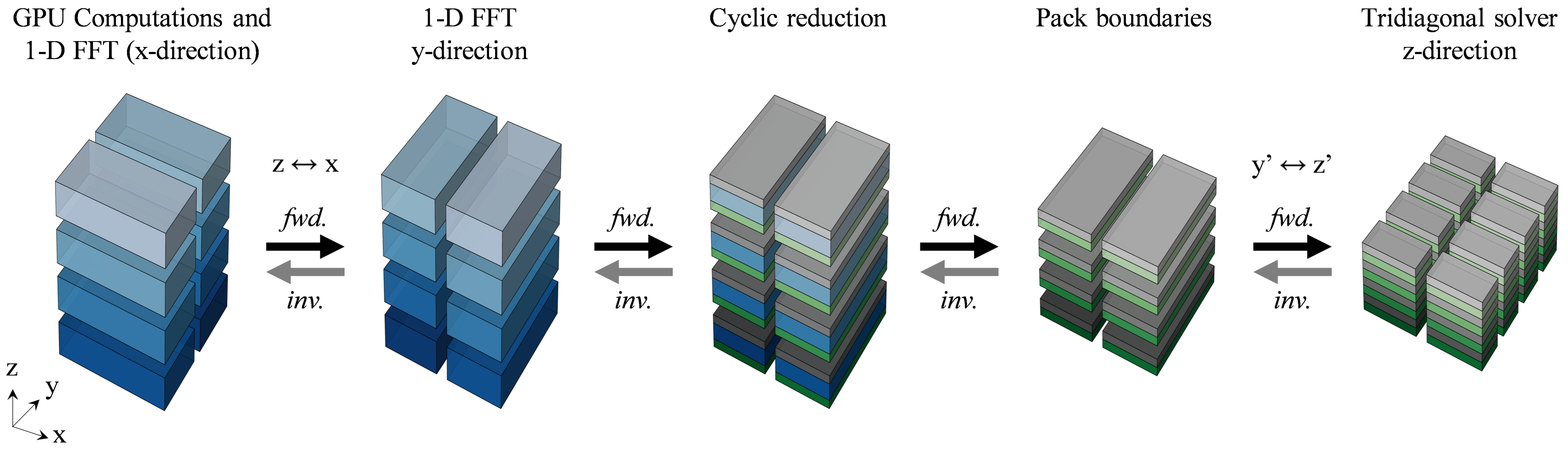
tridiagonal problem

$$\left(\begin{array}{cccc|cccc|cccc}
b_0 & c_0 & & & & & & & & & & \\
a_1 & b_1 & c_1 & & & & & & & & & \\
 & a_2 & b_2 & c_2 & & & & & & & & \\
 & & a_3 & b_3 & c_3 & & & & & & & \\ \hline
 & & & a_4 & b_4 & c_4 & & & & & & \\
 & & & & a_5 & b_5 & c_5 & & & & & \\
 & & & & & a_6 & b_6 & c_6 & & & & \\
 & & & & & & a_7 & b_7 & c_7 & & & \\ \hline
 & & & & & & & a_8 & b_8 & c_8 & & \\
 & & & & & & & & a_9 & b_9 & c_9 & \\
 & & & & & & & & & a_{10} & b_{10} & c_{10} \\
 & & & & & & & & & & a_{11} & b_{11}
\end{array}\right)
\left(\begin{array}{c}
u_0 \\ u_1 \\ u_2 \\ u_3 \\ \hline u_4 \\ u_5 \\ u_6 \\ u_7 \\ \hline u_8 \\ u_9 \\ u_{10} \\ u_{11}
\end{array}\right)
=
\left(\begin{array}{c}
d_0 \\ d_1 \\ d_2 \\ d_3 \\ \hline d_4 \\ d_5 \\ d_6 \\ d_7 \\ \hline d_8 \\ d_9 \\ d_{10} \\ d_{11}
\end{array}\right)$$

$$\Longrightarrow$$

reduced system

$$\left(\begin{array}{cccc|cccc|cccc}
\mathbf{1} & & & \mathbf{c_0^*} & & & & & & & & \\
a_1^* & 1 & & c_1^* & & & & & & & & \\
a_2^* & & 1 & c_2^* & & & & & & & & \\
\mathbf{a_3^*} & & & \mathbf{1} & \mathbf{c_3^*} & & & & & & & \\ \hline
 & & & \mathbf{a_4^*} & \mathbf{1} & & & \mathbf{c_4^*} & & & & \\
 & & & & a_5^* & 1 & & c_5^* & & & & \\
 & & & & a_6^* & & 1 & c_6^* & & & & \\
 & & & & \mathbf{a_7^*} & & & \mathbf{1} & \mathbf{c_7^*} & & & \\ \hline
 & & & & & & & \mathbf{a_8^*} & \mathbf{1} & & & \mathbf{c_8^*} \\
 & & & & & & & & a_9^* & 1 & & c_9^* \\
 & & & & & & & & a_{10}^* & & 1 & c_{10}^* \\
 & & & & & & & & \mathbf{a_{11}^*} & & & \mathbf{1}
\end{array}\right)
\left(\begin{array}{c}
\mathbf{u_0} \\ u_1 \\ u_2 \\ \mathbf{u_3} \\ \hline \mathbf{u_4} \\ u_5 \\ u_6 \\ \mathbf{u_7} \\ \hline \mathbf{u_8} \\ u_9 \\ u_{10} \\ \mathbf{u_{11}}
\end{array}\right)
=
\left(\begin{array}{c}
\mathbf{d_0^*} \\ d_1^* \\ d_2^* \\ \mathbf{d_3^*} \\ \hline \mathbf{d_4^*} \\ d_5^* \\ d_6^* \\ \mathbf{d_7^*} \\ \hline \mathbf{d_8^*} \\ d_9^* \\ d_{10}^* \\ \mathbf{d_{11}^*}
\end{array}\right)$$



GPU Computations and 1-D FFT (x-direction)

1-D FFT y-direction

Cyclic reduction

Pack boundaries

Tridiagonal solver z-direction

Laszlo et al. ***ACM Math.*** **42.4 (2016): 1-36.**

distributed TDMA to reduce communication cost

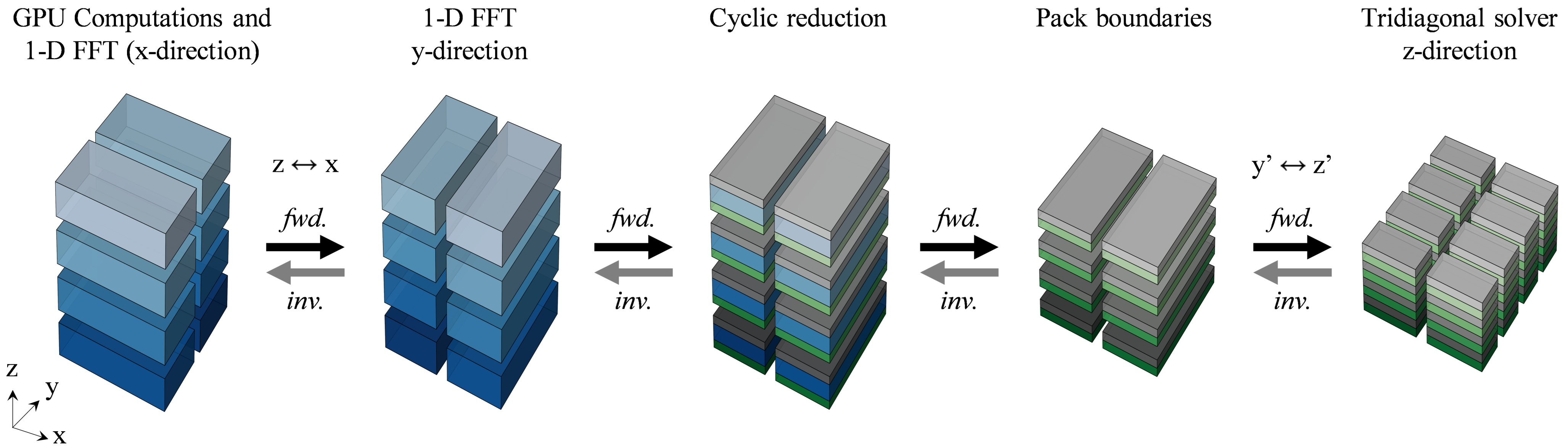
tridiagonal problem

$$\left(\begin{array}{cccc|cccc|cccc}
b_0 & c_0 & & & & & & & & & & \\
a_1 & b_1 & c_1 & & & & & & & & & \\
 & a_2 & b_2 & c_2 & & & & & & & & \\
 & & a_3 & b_3 & c_3 & & & & & & & \\
\hline
 & & & a_4 & b_4 & c_4 & & & & & & \\
 & & & & a_5 & b_5 & c_5 & & & & & \\
 & & & & & a_6 & b_6 & c_6 & & & & \\
 & & & & & & a_7 & b_7 & c_7 & & & \\
\hline
 & & & & & & & a_8 & b_8 & c_8 & & \\
 & & & & & & & & a_9 & b_9 & c_9 & \\
 & & & & & & & & & a_{10} & b_{10} & c_{10} \\
 & & & & & & & & & & a_{11} & b_{11}
\end{array}\right)
\left(\begin{array}{c} u_0 \\ u_1 \\ u_2 \\ u_3 \\ \hline u_4 \\ u_5 \\ u_6 \\ u_7 \\ \hline u_8 \\ u_9 \\ u_{10} \\ u_{11} \end{array}\right)
=
\left(\begin{array}{c} d_0 \\ d_1 \\ d_2 \\ d_3 \\ \hline d_4 \\ d_5 \\ d_6 \\ d_7 \\ \hline d_8 \\ d_9 \\ d_{10} \\ d_{11} \end{array}\right)$$

$$\Longrightarrow$$

reduced system

$$\left(\begin{array}{cccc|cccc|cccc}
\mathbf{1} & & & \mathbf{c_0^*} & & & & & & & & \\
a_1^* & 1 & & c_1^* & & & & & & & & \\
a_2^* & & 1 & c_2^* & & & & & & & & \\
\mathbf{a_3^*} & & & \mathbf{1} & \mathbf{c_3^*} & & & & & & & \\
\hline
 & & & \mathbf{a_4^*} & \mathbf{1} & & & \mathbf{c_4^*} & & & & \\
 & & & & a_5^* & 1 & & c_5^* & & & & \\
 & & & & a_6^* & & 1 & c_6^* & & & & \\
 & & & & \mathbf{a_7^*} & & & \mathbf{1} & \mathbf{c_7^*} & & & \\
\hline
 & & & & & & & \mathbf{a_8^*} & \mathbf{1} & & & \mathbf{c_8^*} \\
 & & & & & & & & a_9^* & 1 & & c_9^* \\
 & & & & & & & & a_{10}^* & & 1 & c_{10}^* \\
 & & & & & & & & \mathbf{a_{11}^*} & & & \mathbf{1}
\end{array}\right)
\left(\begin{array}{c} \mathbf{u_0} \\ u_1 \\ u_2 \\ \mathbf{u_3} \\ \hline \mathbf{u_4} \\ u_5 \\ u_6 \\ \mathbf{u_7} \\ \hline \mathbf{u_8} \\ u_9 \\ u_{10} \\ \mathbf{u_{11}} \end{array}\right)
=
\left(\begin{array}{c} \mathbf{d_0^*} \\ d_1^* \\ d_2^* \\ \mathbf{d_3^*} \\ \hline \mathbf{d_4^*} \\ d_5^* \\ d_6^* \\ \mathbf{d_7^*} \\ \hline \mathbf{d_8^*} \\ d_9^* \\ d_{10}^* \\ \mathbf{d_{11}^*} \end{array}\right)$$



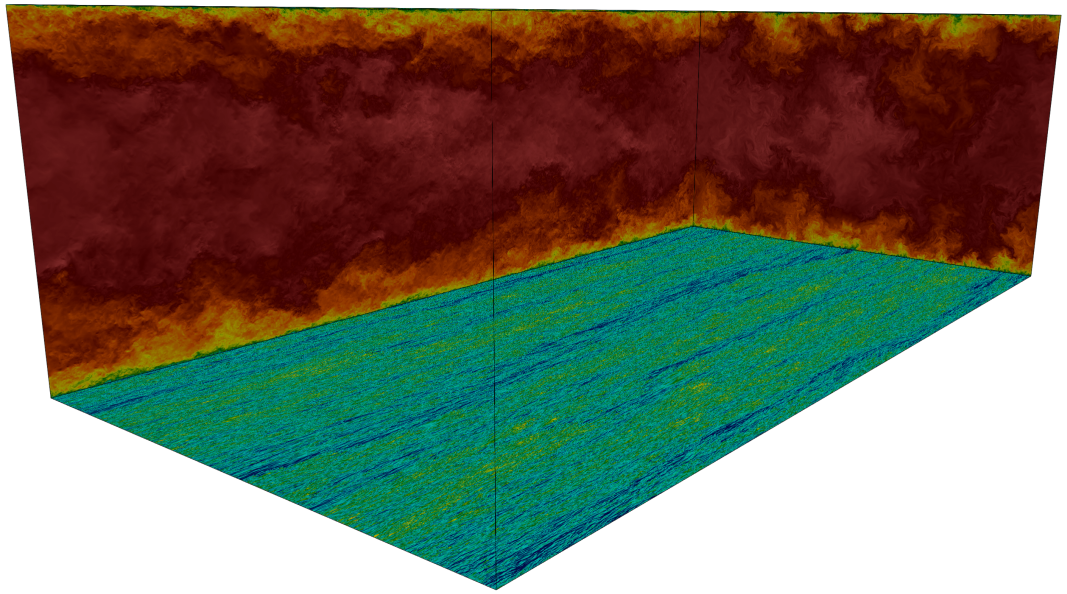
same set of *all-to-all* collective operations, but much less data to “transpose”

Laszlo et al. ***ACM Math.* 42.4 (2016): 1-36.**

Poisson solver performance at scale

turbulent channel flow setup w/ implicit Z diffusion

82 billion grid points



R. Diez, J. Peeters, & P. Costa. CPC (2025)

Poisson solver performance at scale

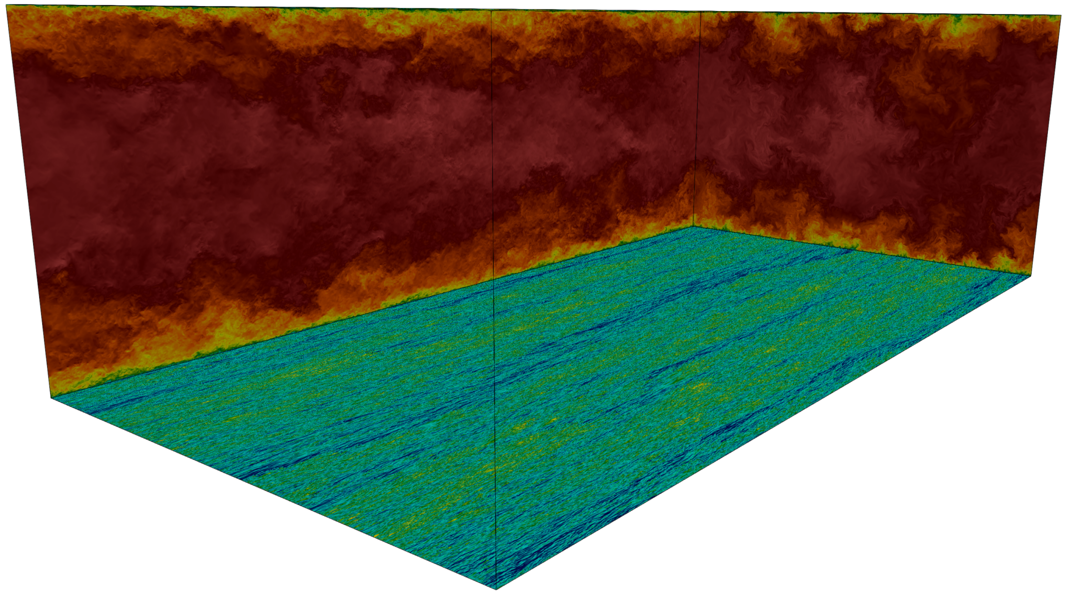
turbulent channel flow setup w/ implicit Z diffusion

82 billion grid points

LEONARDO

#GPUs (nodes)	wall-time/3RK steps [s]	wall-time/3RK steps [s] (w/
256 (64)	4.02	2.56
512 (128)	2.21	1.34
1024 (256)	1.22	0.78

$t_w N_{GPU}/(N_x N_y N_z) \approx 8\,\mathrm{ns}$

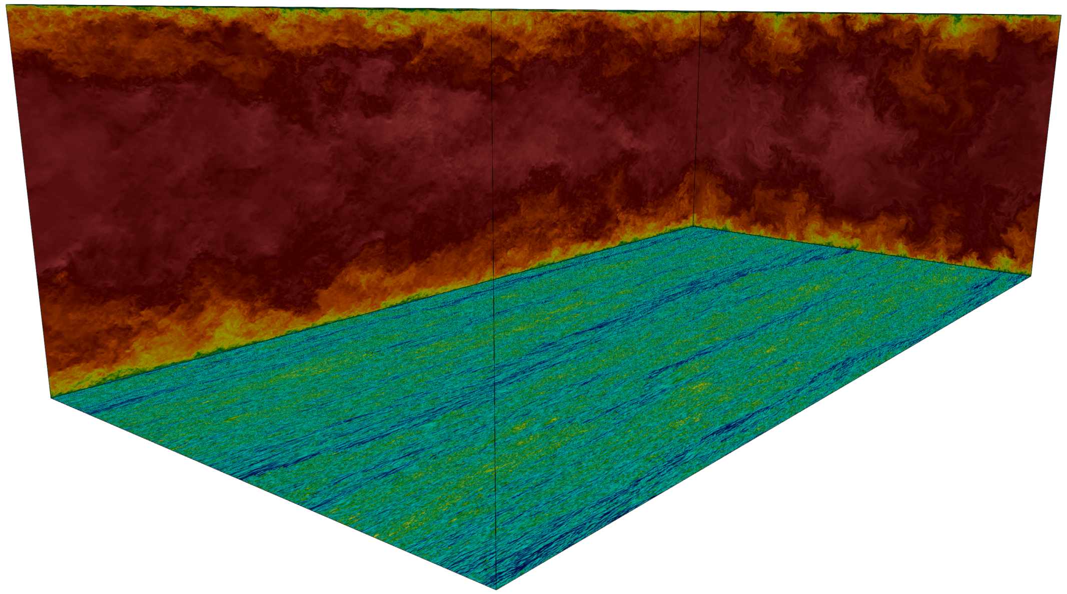


R. Diez, J. Peeters, & P. Costa. CPC (2025)

Poisson solver performance at scale

turbulent channel flow setup w/ implicit Z diffusion

82 billion grid points



LEONARDO

#GPUs (nodes)	wall-time/3RK steps [s]	wall-time/3RK steps [s] (w/
256 (64)	4.02	2.56
512 (128)	2.21	1.34
1024 (256)	1.22	0.78

$t_w N_{GPU}/(N_x N_y N_z) \approx 8\,\mathrm{ns}$

LUMI

#GCDs (nodes)	wall-time/3RK steps [s] (original)	wall-time/3RK steps [s] (w/
256 (32)	7.33	5.75
512 (64)	3.59	2.95
1024 (128)	2.03	1.61

$t_w N_{GPU}/(N_x N_y N_z) \approx 17\,\mathrm{ns}$ (~ 2.1x slower w.r.t. Leonardo)

R. Diez, J. Peeters, & P. Costa. CPC (2025)

Poisson solver performance at scale

turbulent channel flow setup w/ implicit Z diffusion

82 billion grid points

LEONARDO

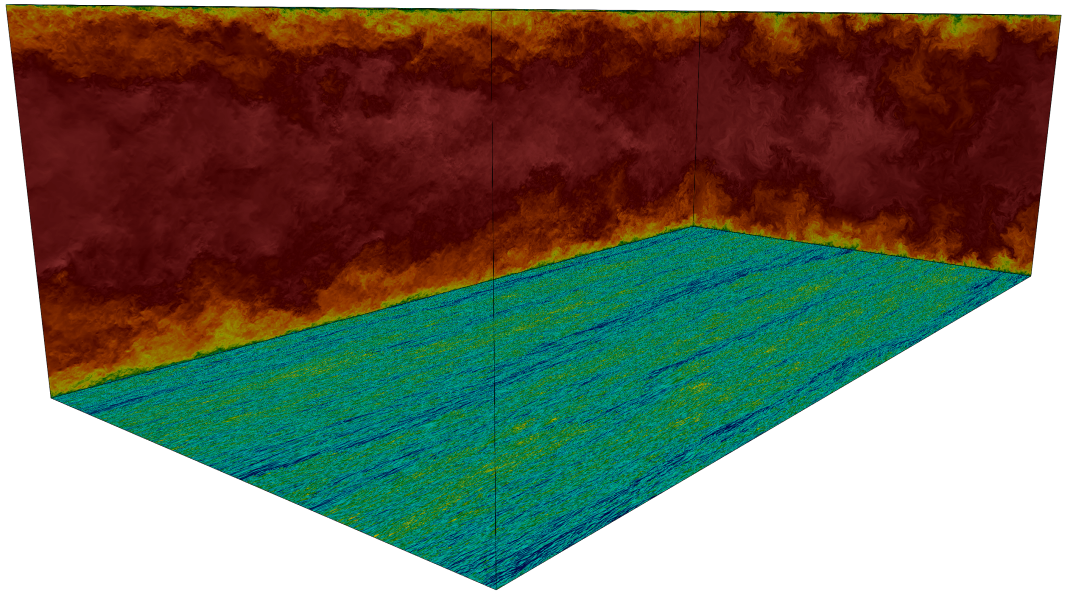
#GPUs (nodes)	wall-time/3RK steps [s]	wall-time/3RK steps [s] (w/
256 (64)	4.02	2.56
512 (128)	2.21	1.34
1024 (256)	1.22	0.78

$t_w N_{GPU}/(N_x N_y N_z) \approx 8\,\mathrm{ns}$

LUMI

#GCDs (nodes)	wall-time/3RK steps [s] (original)	wall-time/3RK steps [s] (w/
256 (32)	7.33	5.75
512 (64)	3.59	2.95
1024 (128)	2.03	1.61

$t_w N_{GPU}/(N_x N_y N_z) \approx 17\,\mathrm{ns}$ (~ 2.1x slower w.r.t. Leonardo)



- Leonardo — **speedup w/ PCR-TDMA of 2x in the Poisson solver; 1.6x in the overall numerical algorithm**
- LUMI — lower speedup w/ PCR-TDMA due to poor transpose performance (higher latency & computational overhead)
- normalised wall time on saturated GPUs — CaNS ~2x faster on Leonardo than LUMI

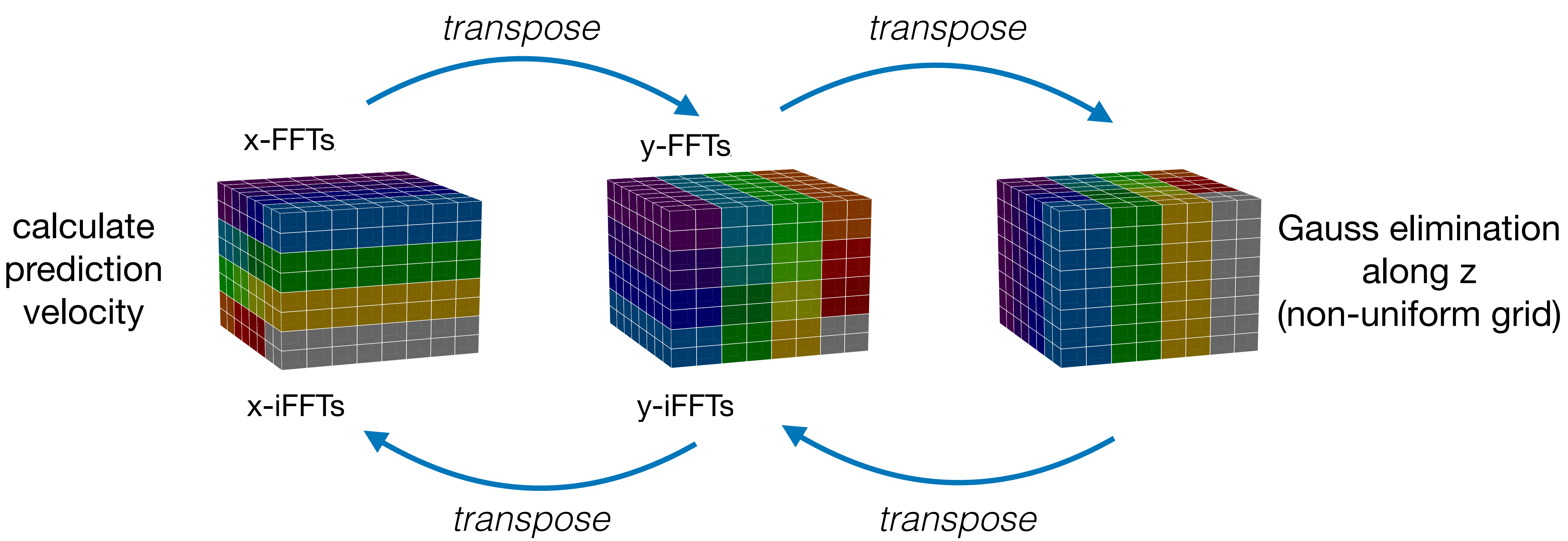
R. Diez, J. Peeters, & P. Costa. CPC (2025)

so far, we are limited to non-uniform grids along one direction

(e.g., canonical flows with at most one inhomogeneous direction)

can we generalize the approach to non-uniform grids along all directions ?

recent work — extension of fast direct solver for non-uniform grids along all directions



calculate prediction velocity

x-FFTs

x-iFFTs

transpose

y-FFTs

y-iFFTs

transpose

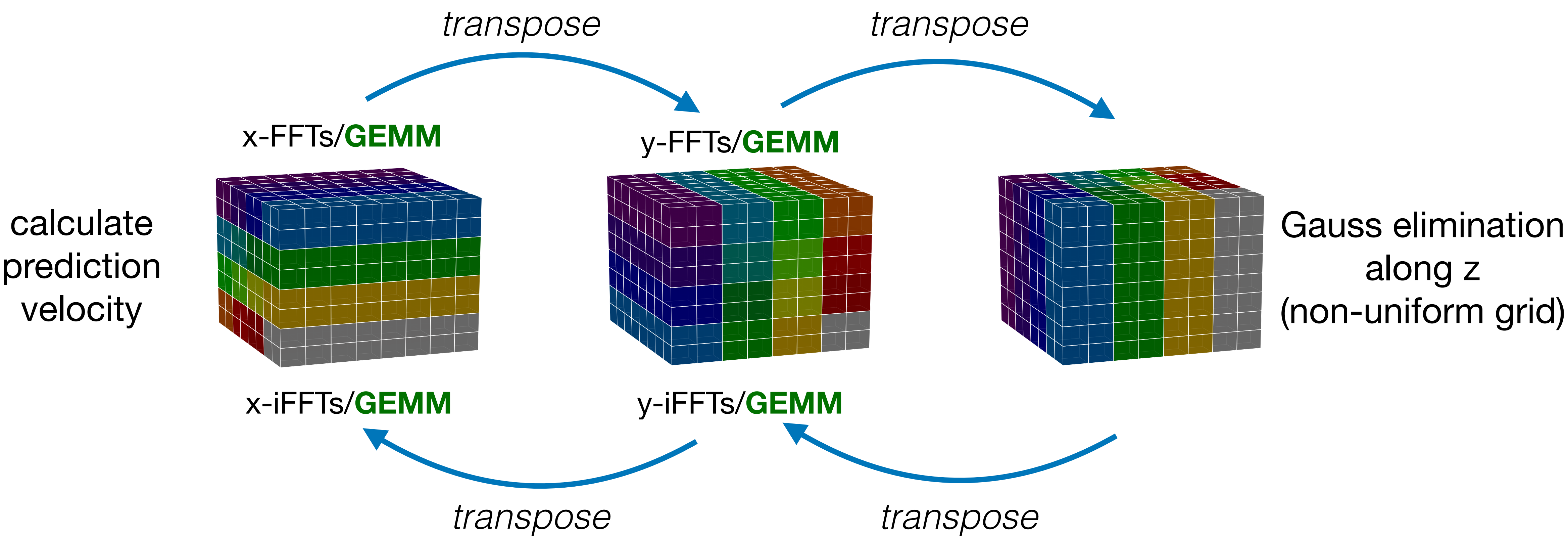
Gauss elimination along z (non-uniform grid)

transpose

transpose

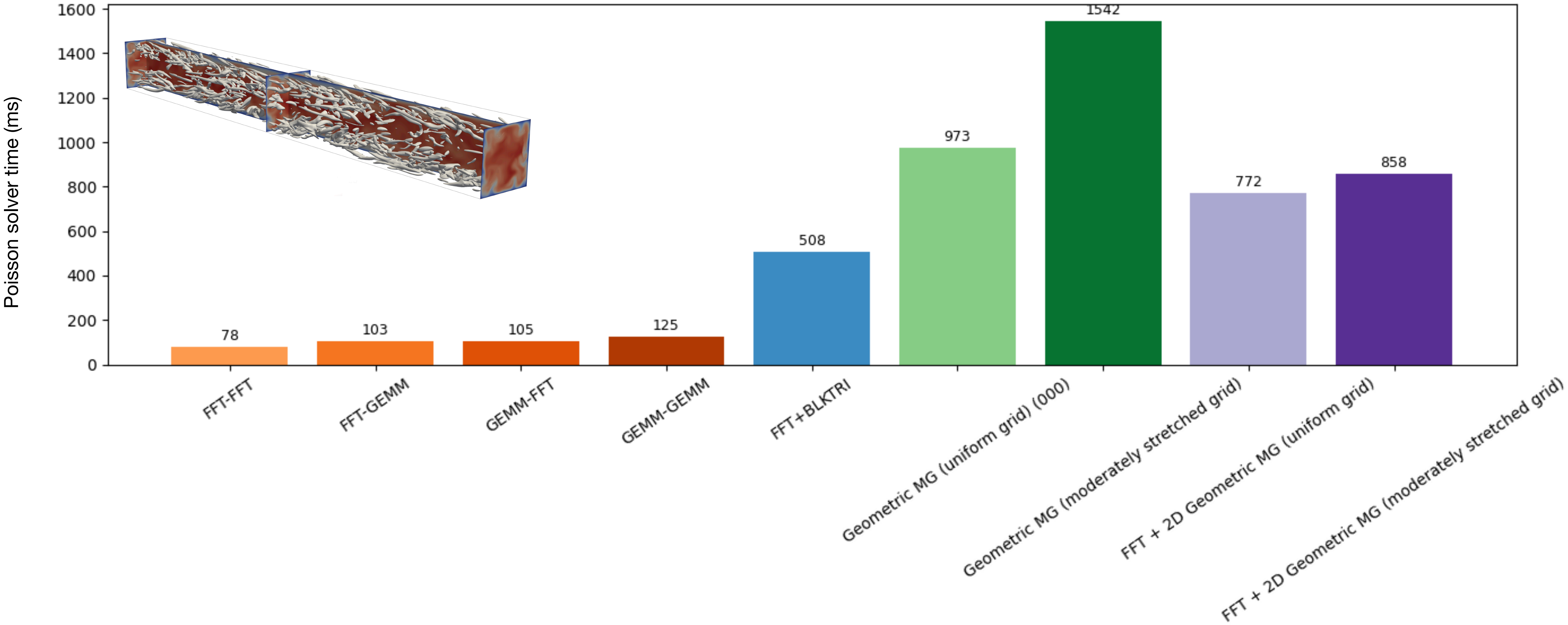
D. Palancha, J. Romero, M. Fatica, R. Verzicco & P. Costa.
A GEMM-based fast finite-difference Poisson solver for non-uniform grids

recent work — extension of fast direct solver for non-uniform grids along all directions



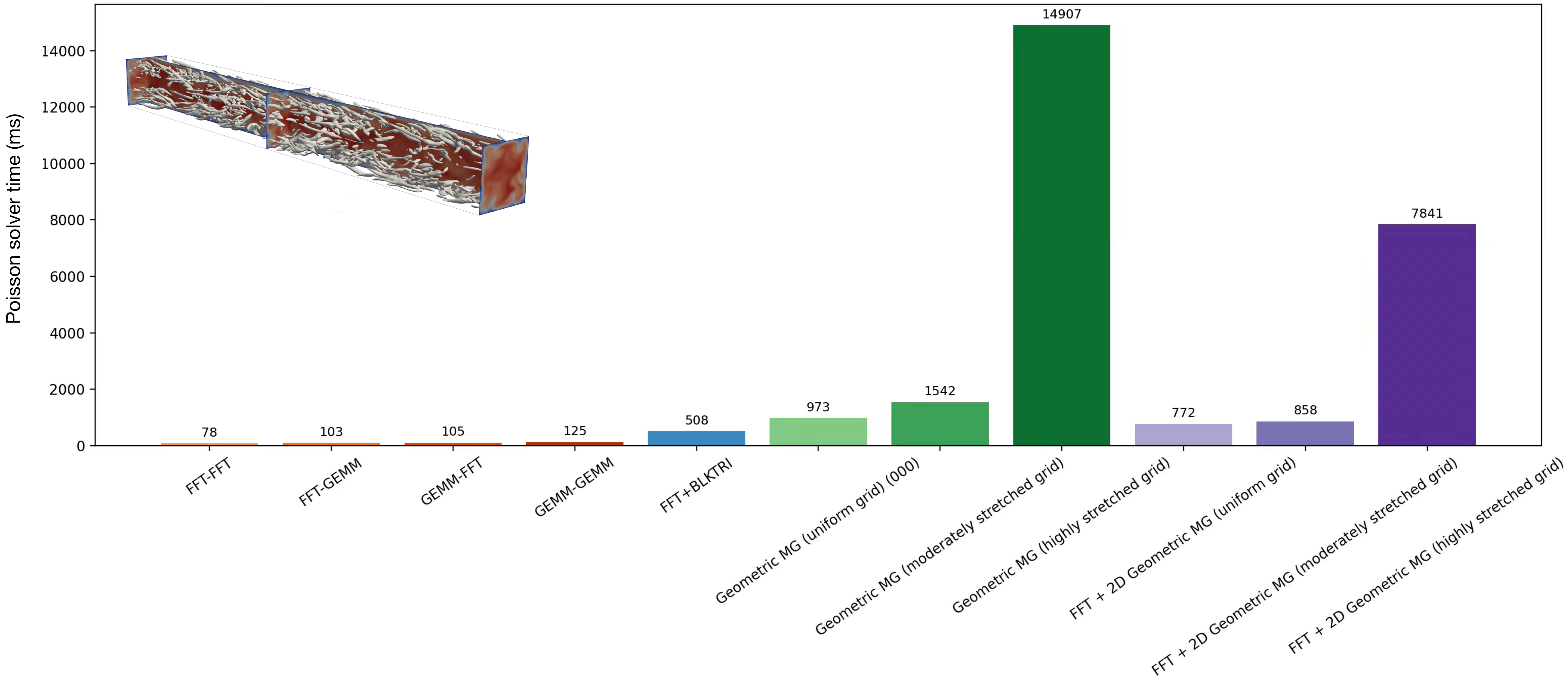
D. Palancha, J. Romero, M. Fatica, R. Verzicco & P. Costa.
A GEMM-based fast finite-difference Poisson solver for non-uniform grids

recent work — extension of fast direct solver for non-uniform grids along all directions



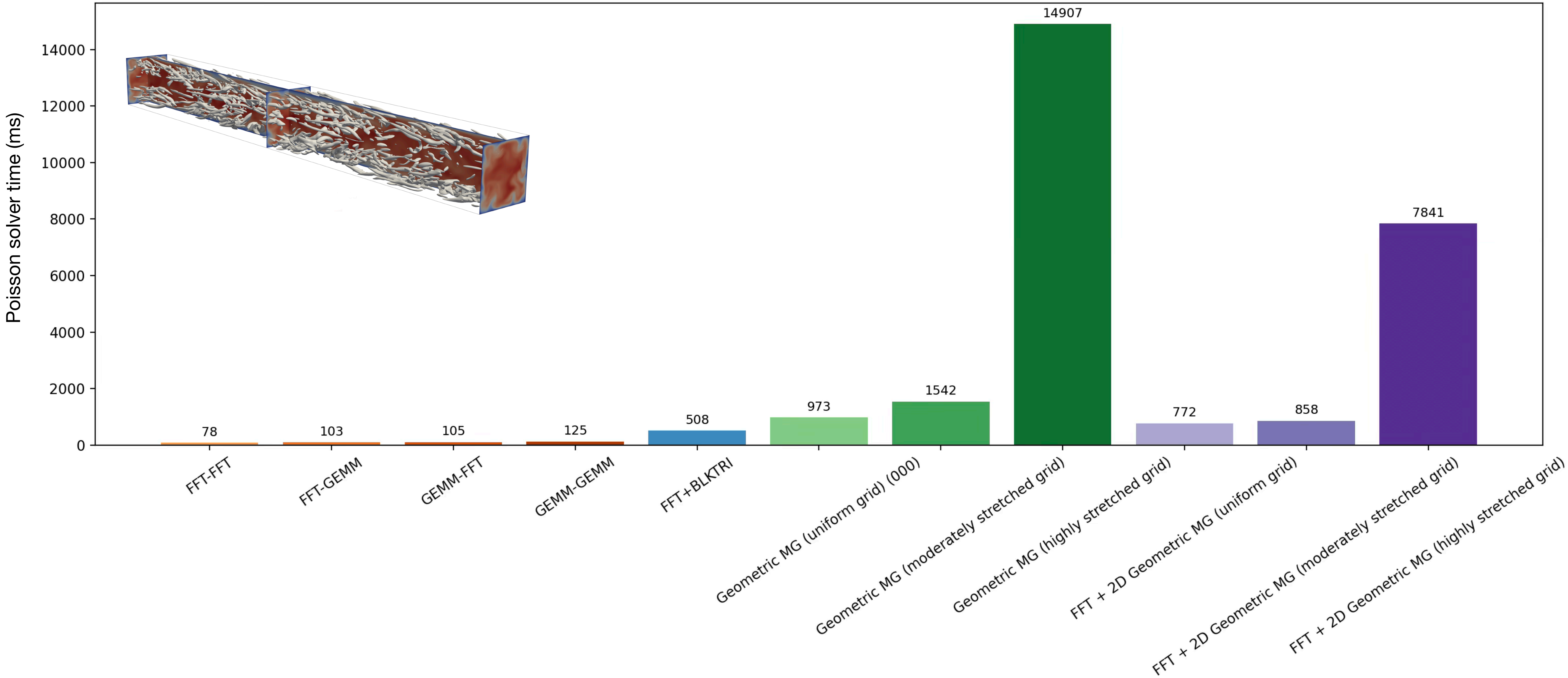
D. Palancha, J. Romero, M. Fatica, R. Verzicco & P. Costa.
A GEMM-based fast finite-difference Poisson solver for non-uniform grids

recent work — extension of fast direct solver for non-uniform grids along all directions



D. Palancha, J. Romero, M. Fatica, R. Verzicco & P. Costa.
A GEMM-based fast finite-difference Poisson solver for non-uniform grids

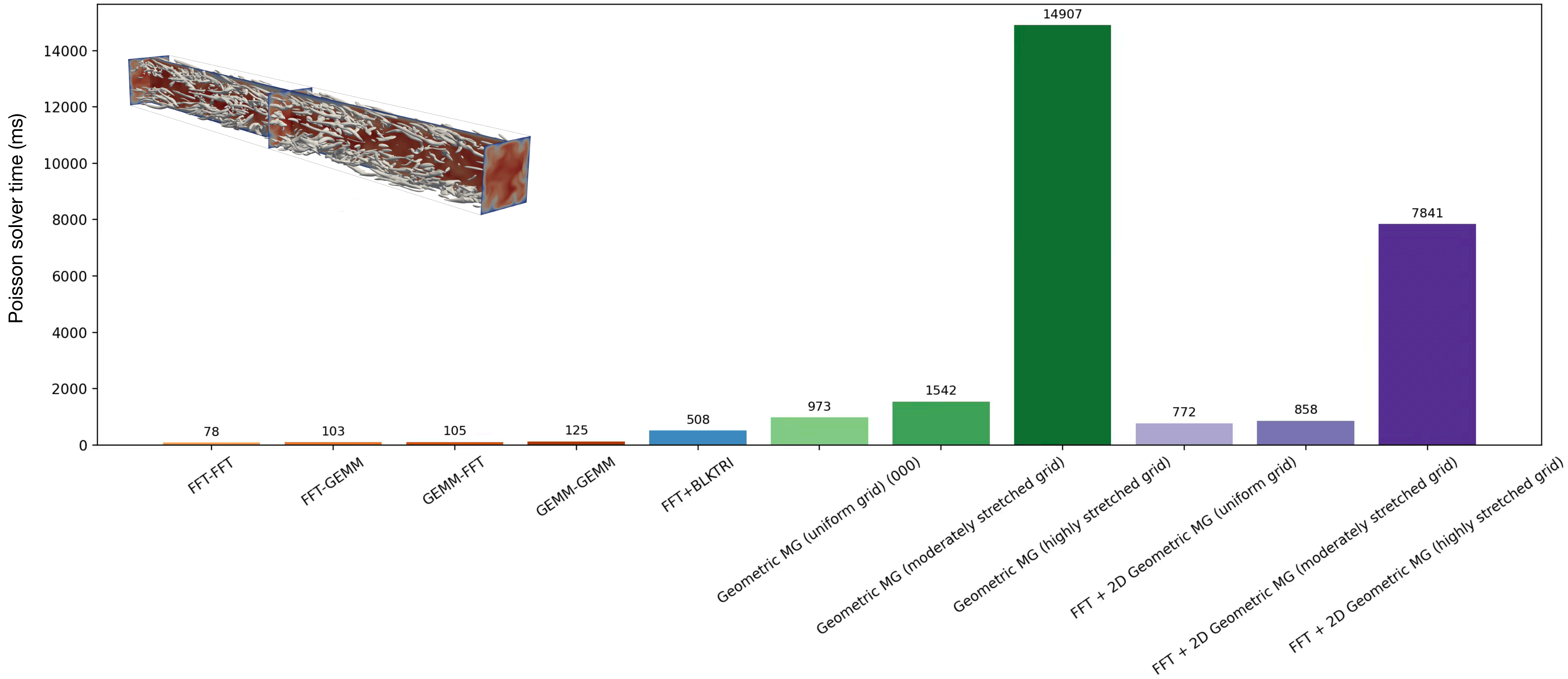
recent work — extension of fast direct solver for non-uniform grids along all directions



GEMM among the most optimized operations on GPUs, thanks to AI!

D. Palancha, J. Romero, M. Fatica, R. Verzicco & P. Costa.
A GEMM-based fast finite-difference Poisson solver for non-uniform grids

recent work — extension of fast direct solver for non-uniform grids along all directions



GEMM among the most optimized operations on GPUs, thanks to AI!

wall-clock time for a Poisson solve on a GH200 supership 2048x512x512: ~ 0.1—0.2s per billion grid cells

D. Palancha, J. Romero, M. Fatica, R. Verzicco & P. Costa.
A GEMM-based fast finite-difference Poisson solver for non-uniform grids

conclusions & outlook

- leveraging the exascale era is hard, but we can do so much more after breaking the barrier of porting our tools to work on extreme-scale GPU-based systems
- running efficiently in recent architectures benefits from tailoring the numerical solver in all three fronts — numerical method, computational algorithm, and implementation strategy
- new FFT/GEMM-based Poisson solver enable fast DNS in cartesian geometric with non-uniform grids along all directions — soon in new version of CaNS
- in multiphase flows, we will be soon able to perform fully-coupled simulations of phase changing multiphase flows near walls with minimal working assumptions
- all open-source (or soon-to-be), you are welcome to try it too!

conclusions & outlook

- leveraging the exascale era is hard, but we can do so much more after breaking the barrier of porting our tools to work on extreme-scale GPU-based systems

- running efficiently in recent architectures benefits from tailoring the numerical solver in all three fronts — numerical method, computational algorithm, and implementation strategy

- new FFT/GEMM-based Poisson solver enable fast DNS in cartesian geometric with non-uniform grids along all directions — soon in new version of CaNS

- in multiphase flows, we will be soon able to perform fully-coupled simulations of phase changing multiphase flows near walls with minimal working assumptions

- all open-source (or soon-to-be), you are welcome to try it too!

thank you for listening!

EuroHPC
Joint Undertaking

NWO