A hierarchical parallel implementation model for algebra-based CFD simulations on hybrid supercomputers

Xavier Álvarez-Farré

SURF

Spoiler

Spoiler

$$\mathbf{q} = -\lambda \nabla T$$

Spoiler

$$\mathbf{q} = -\lambda \nabla T$$

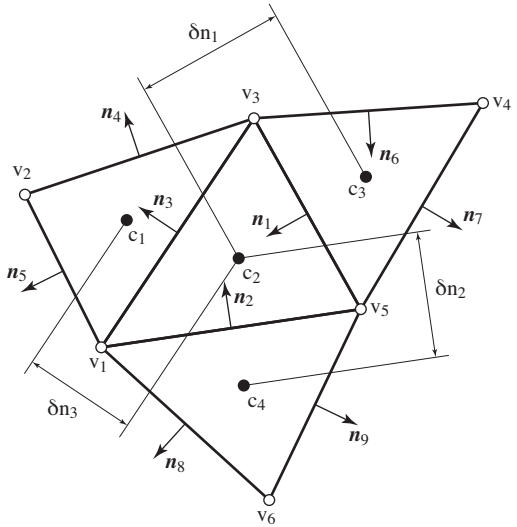
$$\downarrow$$

```
1 #include "hpc2.h"
2 int main () {
3   MPI_Init();
4   mat G("G.bin");
5   vec q(0.), T("T.bin");
6   q = -k*G*T;
7   MPI_Finalize();
8 }
```

$$\mathbf{q} = -\lambda \nabla T$$

$$\downarrow$$

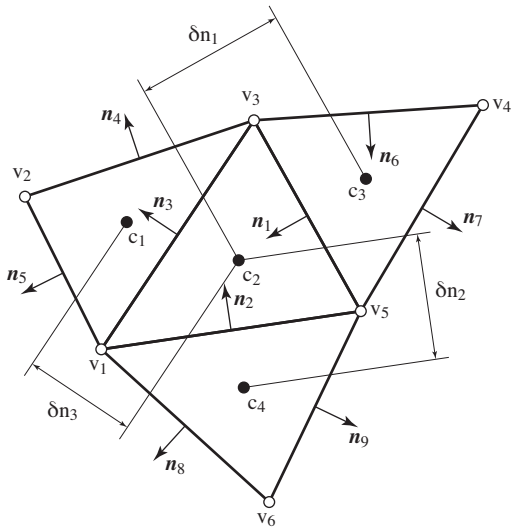
```
1 #include "hpc2.h"
2 int main () {
3   MPI_Init();
4   mat G("G.bin");
5   vec q(0.), T("T.bin");
6   q = -k*G*T;
7   MPI_Finalize();
8 }
```



$$\mathbf{q} = -\lambda \nabla T$$

$$\downarrow$$

```
1 #include "hpc2.h"
2 int main () {
3   MPI_Init();
4   mat G("G.bin");
5   vec q(0.), T("T.bin");
6   q = -k*G*T;
7   MPI_Finalize();
8 }
```



$$\begin{bmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \\ q_5 \\ q_6 \\ q_7 \\ q_8 \\ q_9 \end{bmatrix} = -\lambda \begin{bmatrix} 0 & -1 & +1 & 0 \\ 0 & -1 & 0 & +1 \\ -1 & +1 & 0 & 0 \\ +1 & 0 & 0 & 0 \\ +1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & +1 & 0 \\ 0 & 0 & 0 & +1 \\ 0 & 0 & 0 & +1 \end{bmatrix} \begin{bmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \end{bmatrix}$$

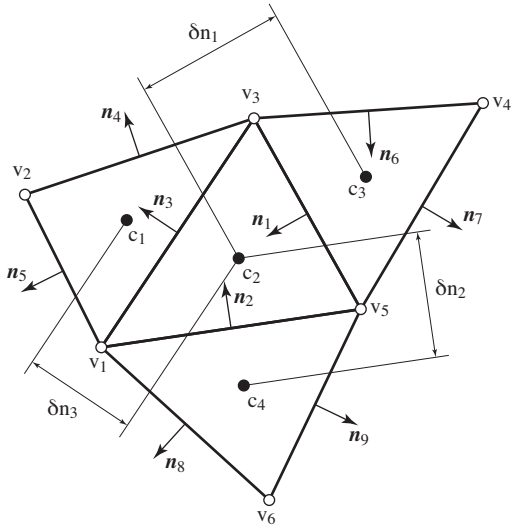
$$\downarrow$$

$$\mathbf{q}_s = -\lambda \mathbf{G} \mathbf{T}_c$$

$$\mathbf{q} = -\lambda \nabla T$$

$$\downarrow$$

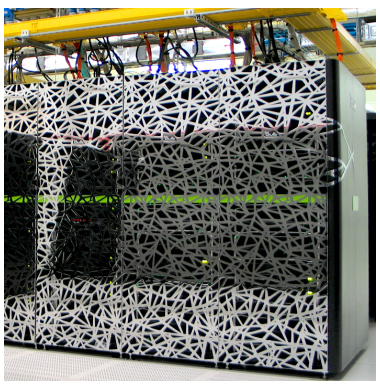
```
1 #include "hpc2.h"
2 int main () {
3   MPI_Init();
4   mat G("G.bin");
5   vec q(0.), T("T.bin");
6   q = -k*G*T;
7   MPI_Finalize();
8 }
```



$$\begin{bmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \\ q_5 \\ q_6 \\ q_7 \\ q_8 \\ q_9 \end{bmatrix} = -\lambda \begin{bmatrix} 0 & -1 & +1 & 0 \\ 0 & -1 & 0 & +1 \\ -1 & +1 & 0 & 0 \\ +1 & 0 & 0 & 0 \\ +1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & +1 & 0 \\ 0 & 0 & 0 & +1 \\ 0 & 0 & 0 & +1 \end{bmatrix} \begin{bmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \end{bmatrix}$$

$$\downarrow$$

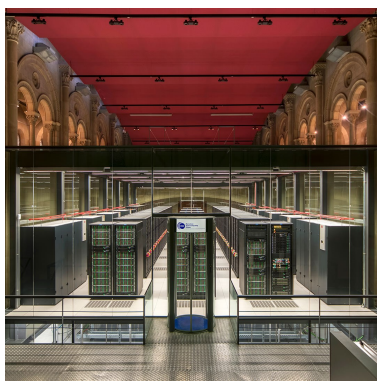
$$\mathbf{q}_s = -\lambda \mathbf{G} \mathbf{T}_c$$



(a) Snellius

(b) Lomonosov 2

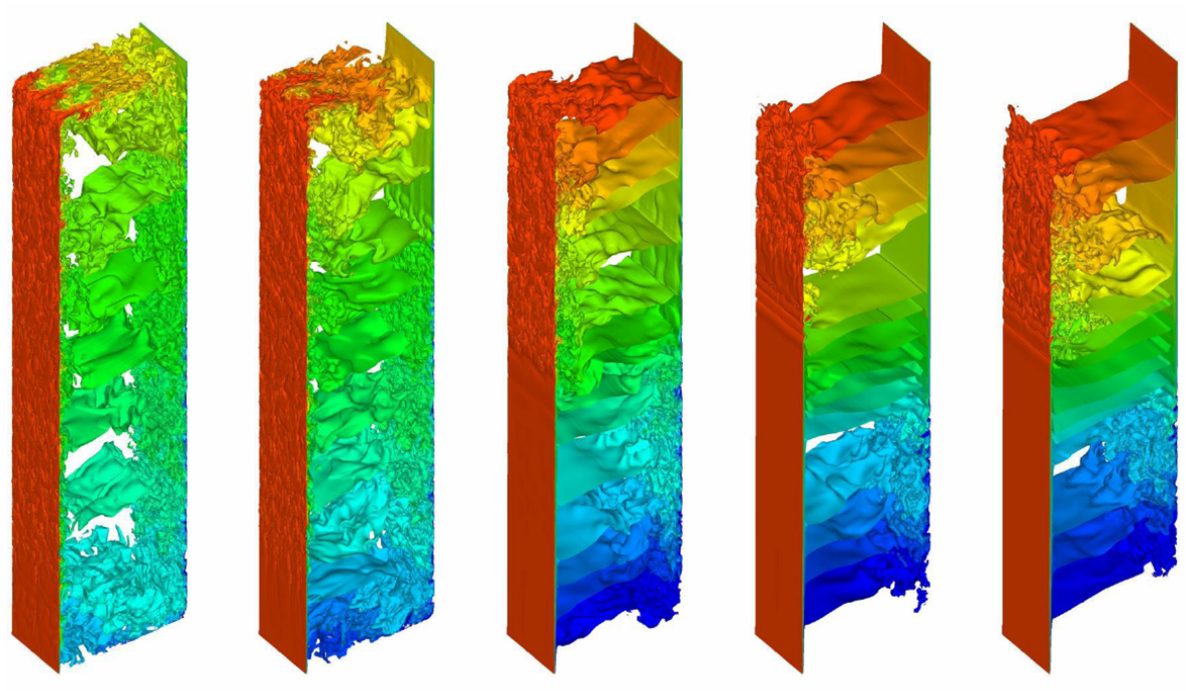
(c) TSUBAME3.0



(d) MareNostrum IV

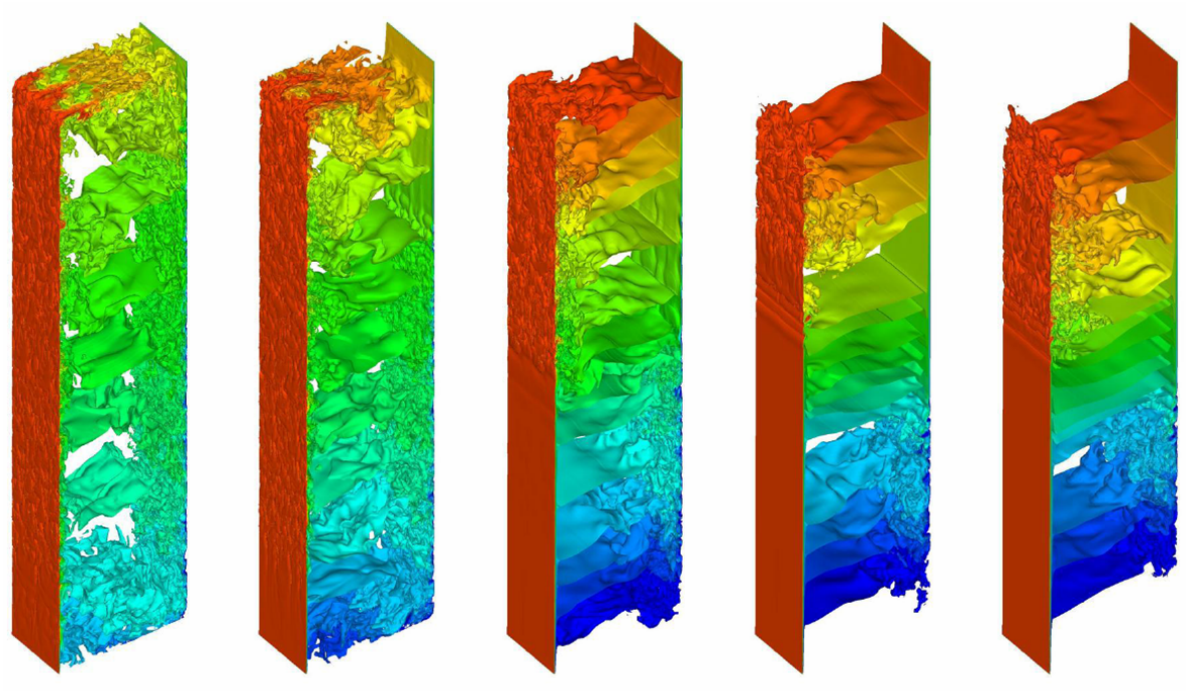
Background

The **Heat and Mass Transfer Technological Center (CTTC)** research group:



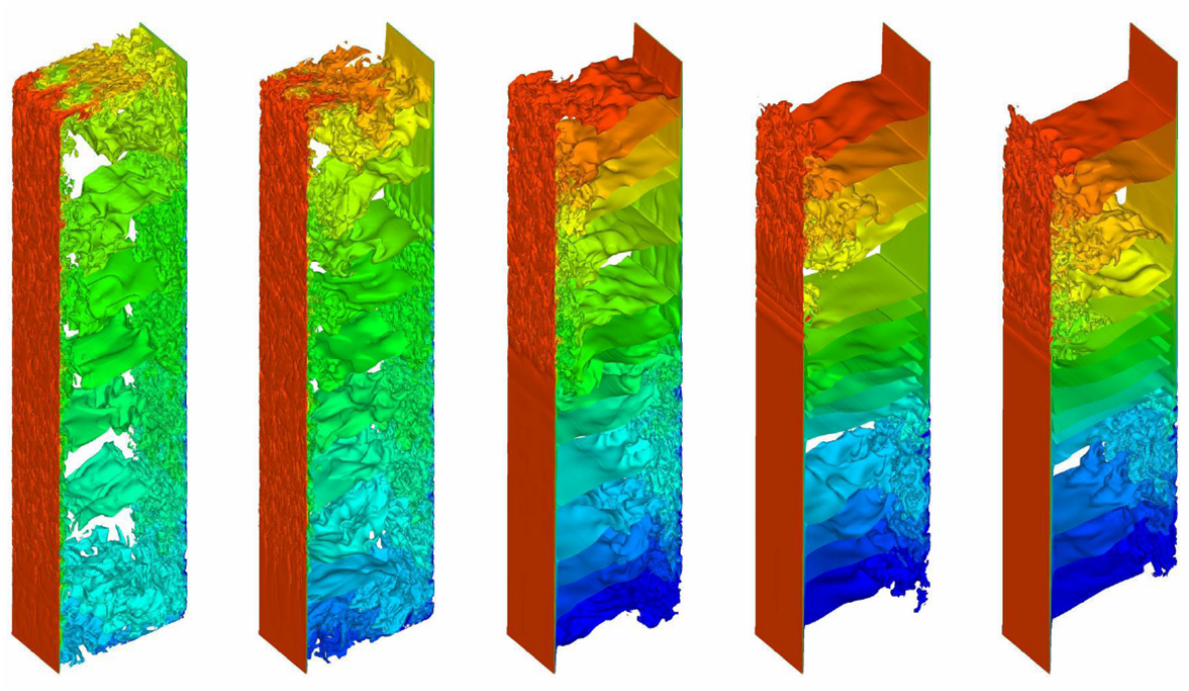
The **Heat and Mass Transfer Technological Center (CTTC)** research group:

- More than 35 researchers.



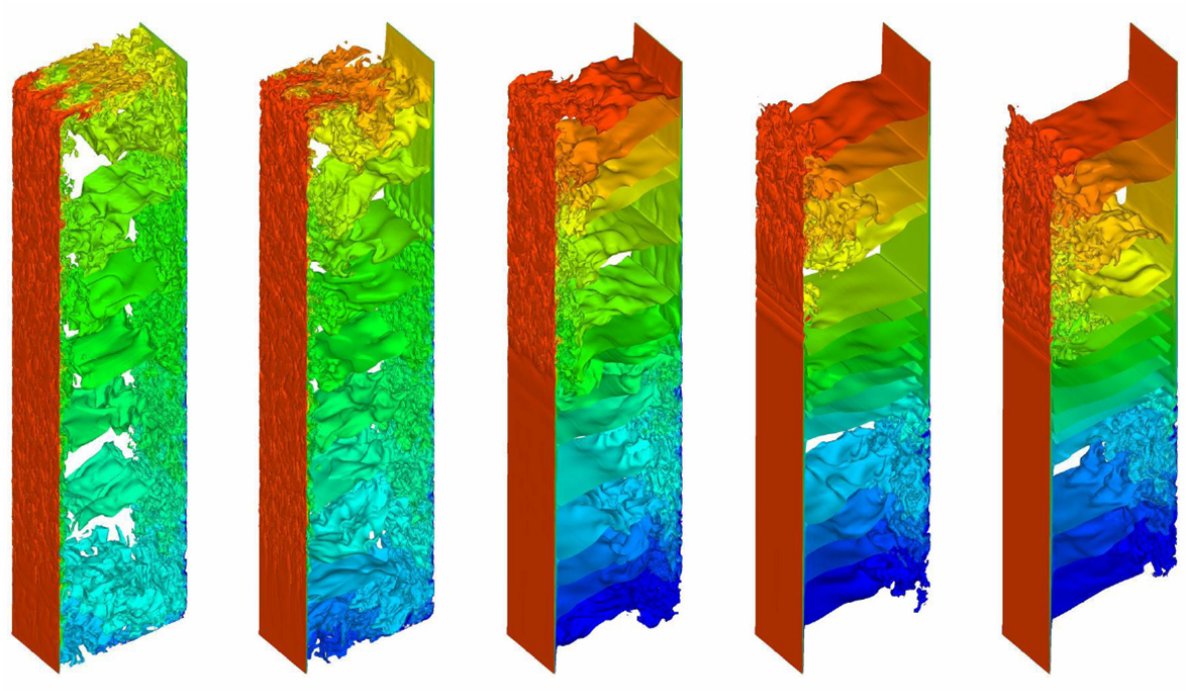
The **Heat and Mass Transfer Technological Center (CTTC)** research group:

- More than 35 researchers.
- Highly concerned about the environmental sustainability.



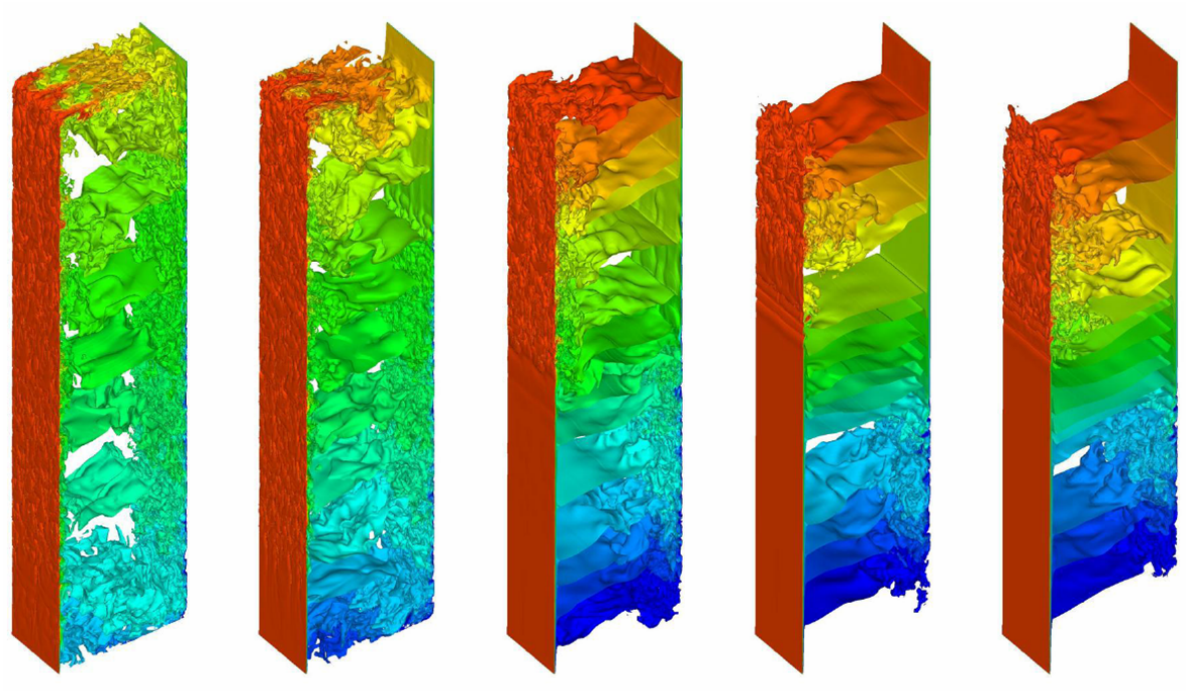
The **Heat and Mass Transfer Technological Center (CTTC)** research group:

- More than 35 researchers.
- Highly concerned about the environmental sustainability.
- Focused on experimental and numerical fluid mechanics.



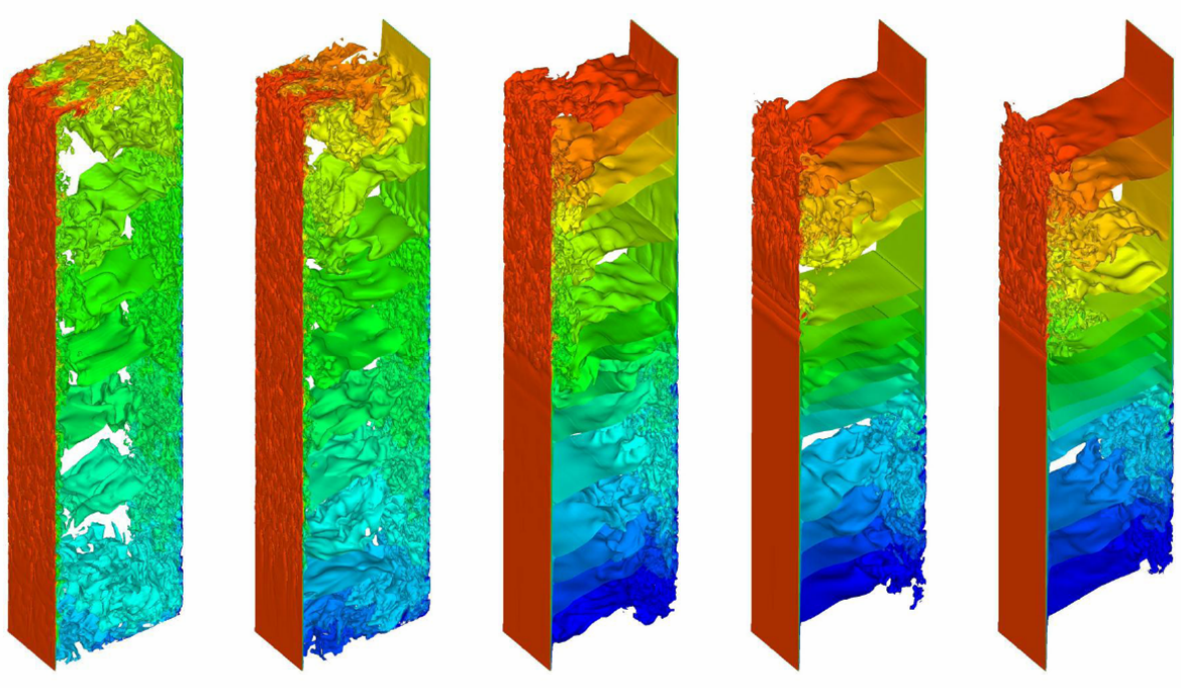
The **Heat and Mass Transfer Technological Center (CTTC)** research group:

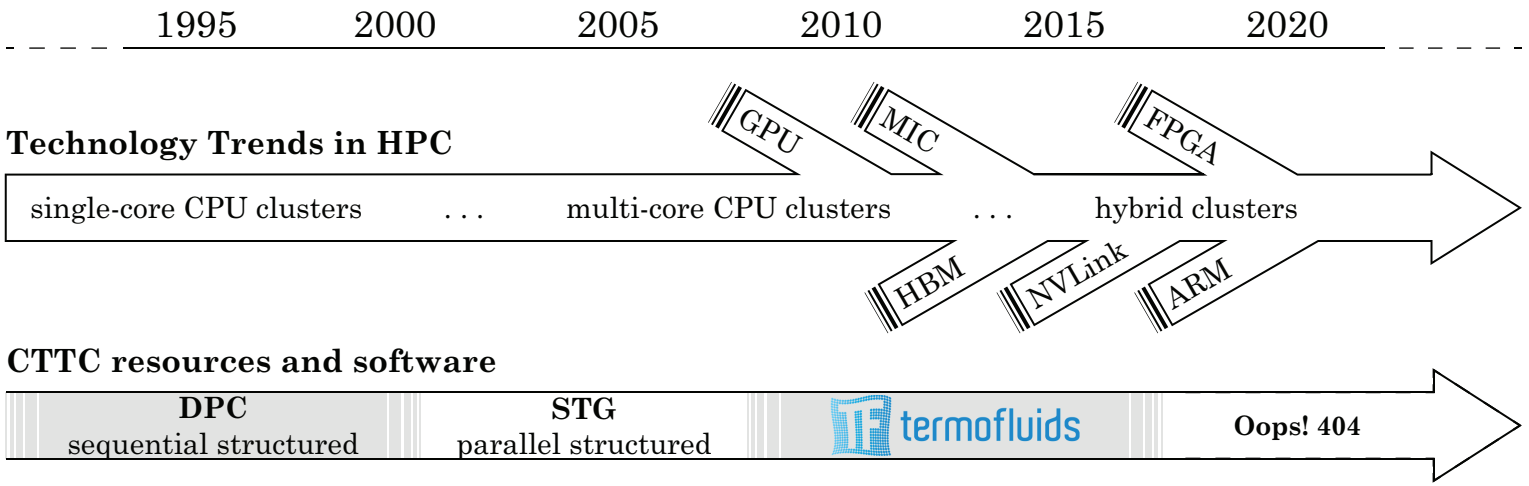
- More than 35 researchers.
- Highly concerned about the environmental sustainability.
- Focused on experimental and numerical fluid mechanics.
- Enrolled in both fundamental and applied research.

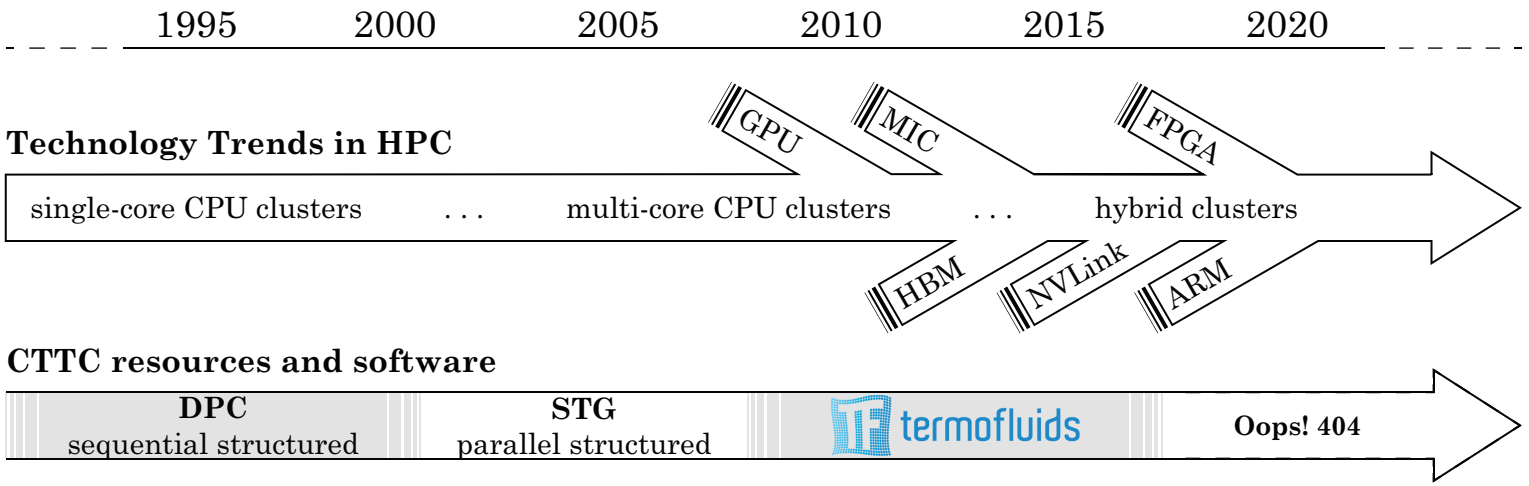


The **Heat and Mass Transfer Technological Center (CTTC)** research group:

- More than 35 researchers.
- Highly concerned about the environmental sustainability.
- Focused on experimental and numerical fluid mechanics.
- Enrolled in both fundamental and applied research.
- Studying several phenomena: natural and forced convection, multi-phase flow, aerodynamics, combustion, fluid-structure interaction, among others.

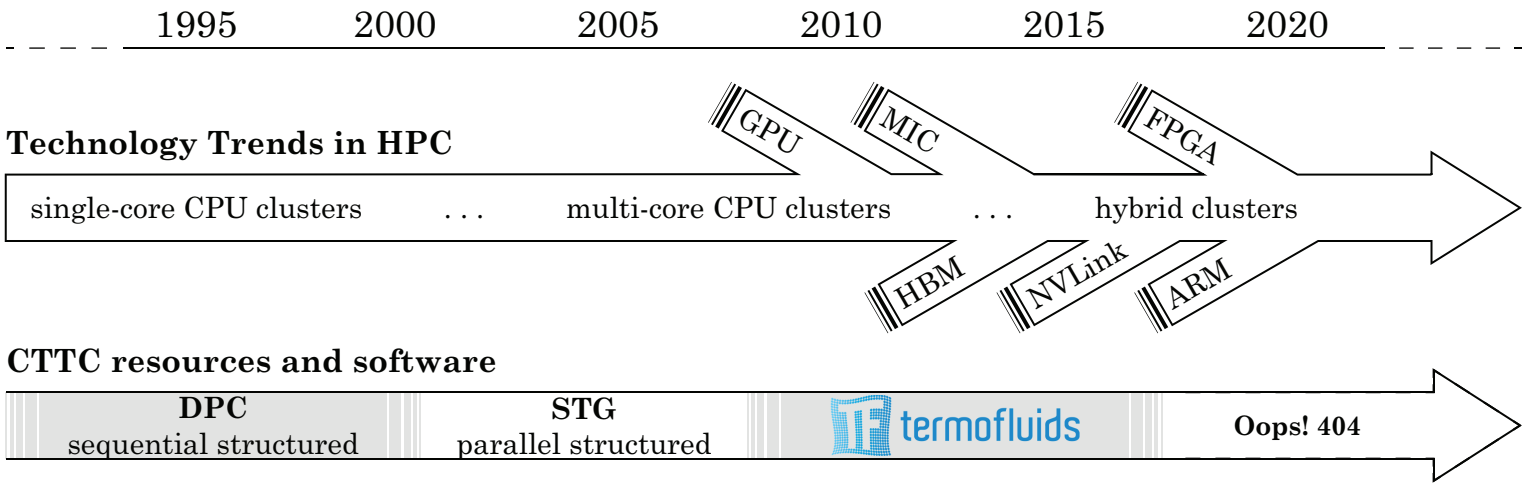



1995
2000
2005
2010
2015
2020
Technology Trends in HPC
GPU
MIC
FPGA
single-core CPU clusters
. . .
multi-core CPU clusters
. . .
hybrid clusters
HBM
NVLink
ARM
CTTC resources and software
DPC
sequential structured
STG
parallel structured
termofluids
Oops! 404



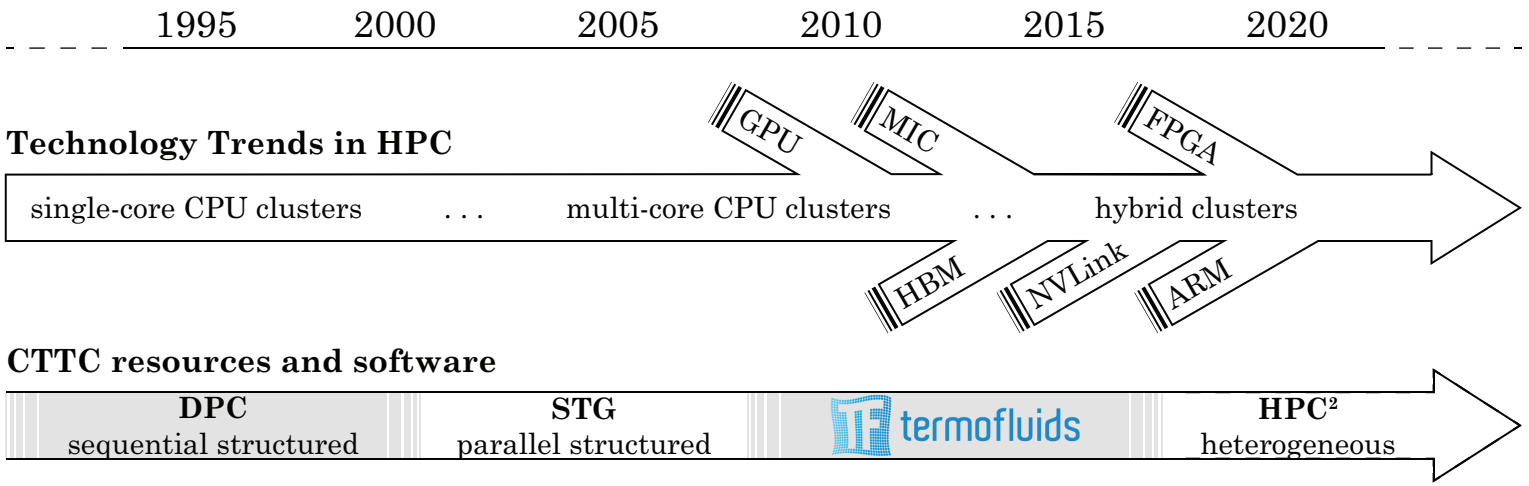
The evolution in hardware technologies

enables scientific computing to advance incessantly and reach further aims. Nowadays, the use of HPC systems is rather common on the solution of both industrial and academic scale problems.



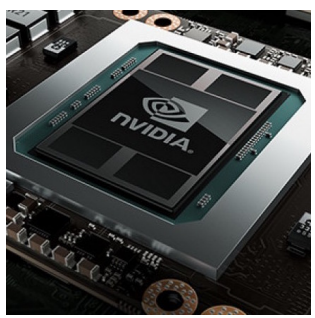
Since the beginning,

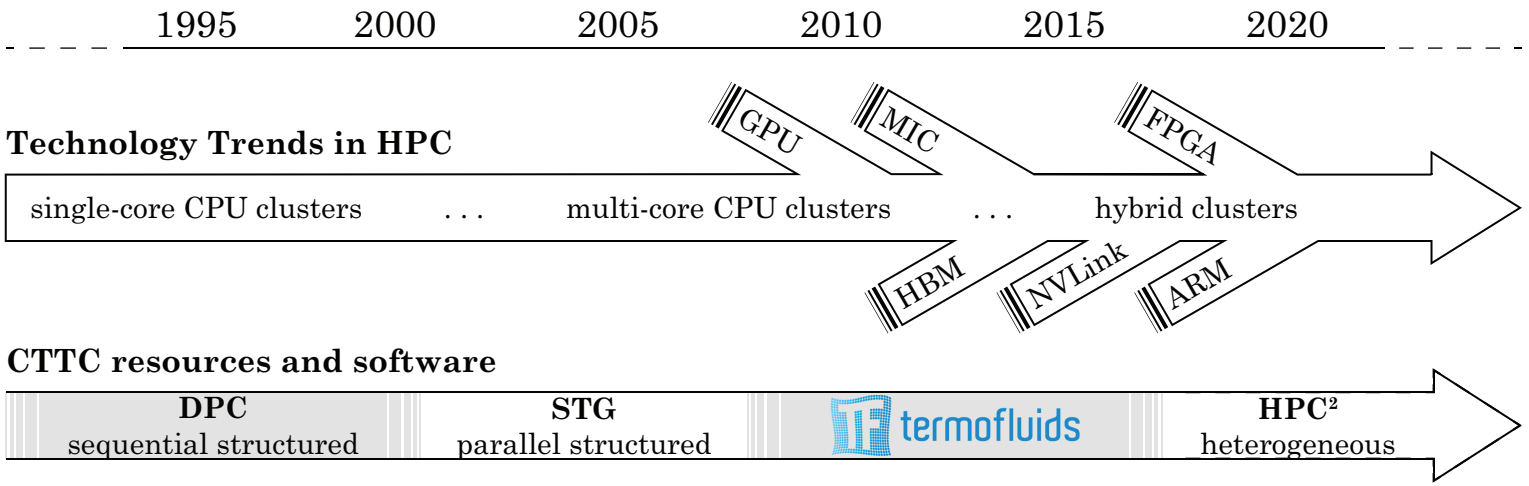
researchers of CTTC is devoted to develop and adapt CFD codes for the state-of-the art computer resources, from sequential structured to parallel unstructured applications.



Massively-parallel devices

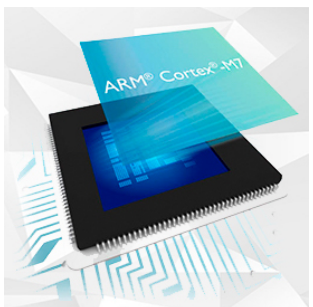
of various architectures are incorporated into modern supercomputers, causing the hybridisation of HPC systems and making the design of computing applications a rather complex problem: the kernels conforming the algorithms must be compatible with distributed- and shared-memory SIMD and MIMD parallelism, and stream processing.

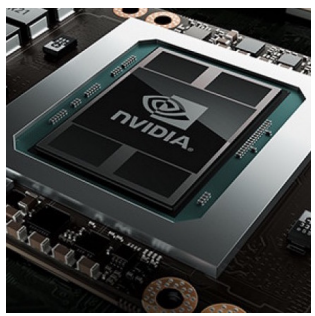




Currently,

a fully-portable, algebra-based framework for heterogeneous computing is being developed. Namely, the traditional stencil data structures and sweeps are replaced by algebraic data structures and kernels, and the discrete operators and mesh functions are then stored as sparse matrices and vectors, respectively.





Ideally,

CFD researchers do CFD stuff.

Ideally,

CFD researchers do CFD stuff.

However,

this is NOT true at all.

Ideally,

CFD researchers do CFD stuff.

However,

this is NOT true at all.

50%

Actual science. Study and develop numerical methods, analyze results, read papers, contribute to community.

Ideally,

CFD researchers do CFD stuff.

However,

this is NOT true at all.

50%

Actual science. Study and develop numerical methods, analyze results, read papers, contribute to community.

25%

Burrocracy. Justify projects and grants, write internal reports. Re:generate information and documentation. Multiplicity.

Ideally,

CFD researchers do CFD stuff.

However,

this is NOT true at all.

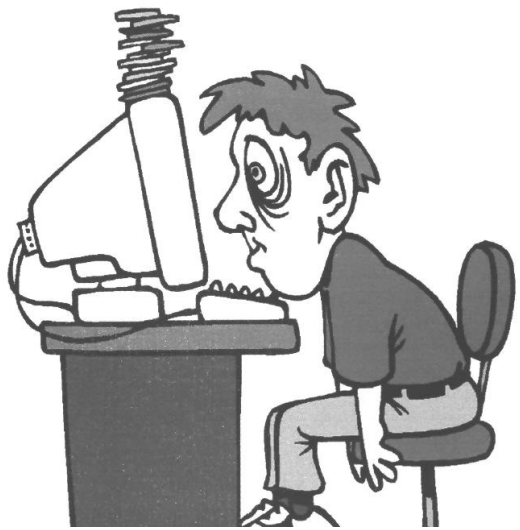
50%

Actual science. Study and develop numerical methods, analyze results, read papers, contribute to community.

25%

Burrocracy. Justify projects and grants, write internal reports. Re:generate information and documentation. Multiplicity.

25%



Develop and debug codes. Adapt frameworks to new clusters. Struggle with SLURM, NUMA, nodes, cores...

Ideally,

CFD researchers do CFD stuff.

However,

this is NOT true at all.

50%

Actual science. Study and develop numerical methods, analyze results, read papers, contribute to community.

25%

Burrocracy. Justify projects and grants, write internal reports. Re:generate information and documentation. Multiplicity.

25%

Develop and debug codes. Adapt frameworks to new clusters. Struggle with SLURM, NUMA, nodes, cores...

The algebraic approach

Stencil

Traditionally, the development of scientific computing software is based on calculations in iterative stencil loops over a discretized geometry—the mesh. Despite being intuitive and versatile, the interdependency between algorithms and their computational implementations in stencil applications usually introduces an inevitable complexity when it comes to portability and sustainability.

Algebraic

By casting discrete operators and mesh functions into sparse matrices and vectors, it has been shown that all the calculations in a typical CFD algorithm for the DNS and LES of incompressible turbulent flows boil down to a minimalist set of algebraic subroutines.

The idea is to use the stencils just for building data and leave the calculations to an algebraic framework; thus, legacy codes may be maintained indefinitely as preprocessing tools, and the calculation engines become easy to port and optimize.

Continuous, dimensionless Navier–Stokes equations read:

$$\nabla \cdot \mathbf{u} = 0, \qquad \partial_t \mathbf{u} + (\mathbf{u} \cdot \nabla)\mathbf{u} - \frac{1}{Re}\Delta \mathbf{u} + \nabla p = 0.$$

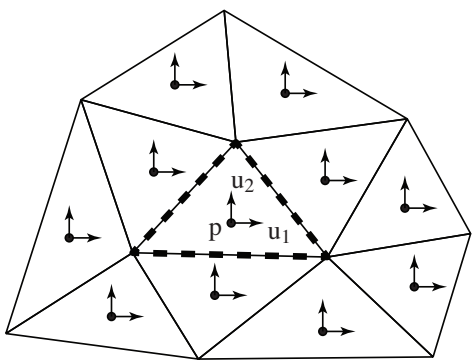
[1] Trias et al., Symmetry-preserving discretization of Navier-Stokes equations on collocated unstructured grids, J.Comp.Phys., 258, 246-267, 2014.

Continuous, dimensionless Navier–Stokes equations read:

$$\nabla \cdot \mathbf{u} = 0, \qquad \partial_t \mathbf{u} + (\mathbf{u} \cdot \nabla)\mathbf{u} - \frac{1}{Re}\Delta \mathbf{u} + \nabla p = 0.$$

Finite-volume, algebra-based discretization on arbitrary collocated mesh[1]:

$$\mathsf{M}\boldsymbol{u}_s = \mathbf{0}_c, \qquad \Omega_c^{3d} d_t \boldsymbol{u}_c + \mathsf{C}_c^{3d}(\boldsymbol{u}_s)\boldsymbol{u}_c + \mathsf{D}_c^{3d}\boldsymbol{u}_c - \Omega_c^{3d}\mathsf{G}_c\boldsymbol{p}_c = \mathbf{0}_c,$$



[1] Trias et al., Symmetry-preserving discretization of Navier-Stokes equations on collocated unstructured grids, J.Comp.Phys., 258, 246-267, 2014.

Continuous, dimensionless Navier–Stokes equations read:

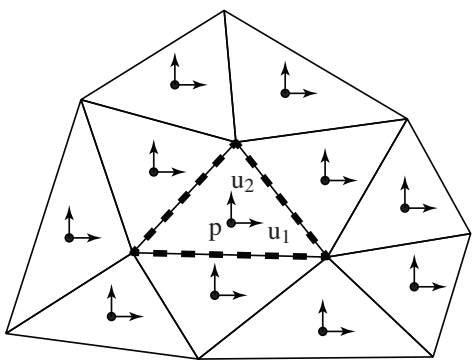
$$\nabla \cdot \mathbf{u} = 0, \qquad \partial_t \mathbf{u} + (\mathbf{u} \cdot \nabla)\mathbf{u} - \frac{1}{Re}\Delta \mathbf{u} + \nabla p = 0.$$

Finite-volume, algebra-based discretization on arbitrary collocated mesh[1]:

$$\mathsf{M}\boldsymbol{u}_s = \mathbf{0}_c, \qquad \Omega_c^{3d} d_t \boldsymbol{u}_c + \mathsf{C}_c^{3d}(\boldsymbol{u}_s)\boldsymbol{u}_c + \mathsf{D}_c^{3d}\boldsymbol{u}_c - \Omega_c^{3d}\mathsf{G}_c\boldsymbol{p}_c = \mathbf{0}_c,$$

where:

$$\text{cells} \in \mathbb{R}^n \quad \text{faces} \in \mathbb{R}^m$$



[1] Trias et al., Symmetry-preserving discretization of Navier-Stokes equations on collocated unstructured grids, J.Comp.Phys., 258, 246-267, 2014.

Continuous, dimensionless Navier–Stokes equations read:

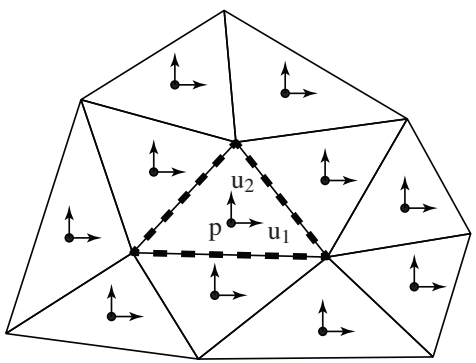
$$\nabla \cdot \mathbf{u} = 0, \qquad \partial_t \mathbf{u} + (\mathbf{u} \cdot \nabla)\mathbf{u} - \frac{1}{Re}\Delta \mathbf{u} + \nabla p = 0.$$

Finite-volume, algebra-based discretization on arbitrary collocated mesh[1]:

$$\mathsf{M}\boldsymbol{u}_s = \mathbf{0}_c, \qquad \Omega_c^{3d} d_t \boldsymbol{u}_c + \mathsf{C}_c^{3d}(\boldsymbol{u}_s)\boldsymbol{u}_c + \mathsf{D}_c^{3d}\boldsymbol{u}_c - \Omega_c^{3d}\mathsf{G}_c\boldsymbol{p}_c = \mathbf{0}_c,$$

where:

$$\text{cells} \in \mathbb{R}^n \quad \text{faces} \in \mathbb{R}^m$$



$$\boldsymbol{p}_c \in \mathbb{R}^n \quad \boldsymbol{u}_c \in \mathbb{R}^{3n} = (\boldsymbol{u}_1, \boldsymbol{u}_2, \boldsymbol{u}_3)^T \quad \boldsymbol{u}_s \in \mathbb{R}^m$$

[1] Trias et al., Symmetry-preserving discretization of Navier-Stokes equations on collocated unstructured grids, J.Comp.Phys., 258, 246-267, 2014.

Continuous, dimensionless Navier–Stokes equations read:

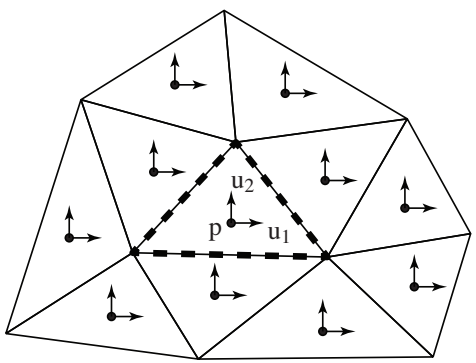
$$\nabla \cdot \mathbf{u} = 0, \qquad \partial_t \mathbf{u} + (\mathbf{u} \cdot \nabla)\mathbf{u} - \frac{1}{Re}\Delta \mathbf{u} + \nabla p = 0.$$

Finite-volume, algebra-based discretization on arbitrary collocated mesh[1]:

$$\mathsf{M}\boldsymbol{u}_s = \mathbf{0}_c, \qquad \Omega_c^{3d} d_t \boldsymbol{u}_c + \mathsf{C}_c^{3d}(\boldsymbol{u}_s)\boldsymbol{u}_c + \mathsf{D}_c^{3d}\boldsymbol{u}_c - \Omega_c^{3d}\mathsf{G}_c\boldsymbol{p}_c = \mathbf{0}_c,$$

where:

$$\text{cells} \in \mathbb{R}^n \quad \text{faces} \in \mathbb{R}^m$$



$$\boldsymbol{p}_c \in \mathbb{R}^n \quad \boldsymbol{u}_c \in \mathbb{R}^{3n} = (\boldsymbol{u}_1, \boldsymbol{u}_2, \boldsymbol{u}_3)^T \quad \boldsymbol{u}_s \in \mathbb{R}^m$$

$$\mathsf{G}_c \in \mathbb{R}^{3n \times n} \quad \mathsf{M} \in \mathbb{R}^{n \times m}$$

[1] Trias et al., Symmetry-preserving discretization of Navier-Stokes equations on collocated unstructured grids, J.Comp.Phys., 258, 246-267, 2014.

Continuous, dimensionless Navier–Stokes equations read:

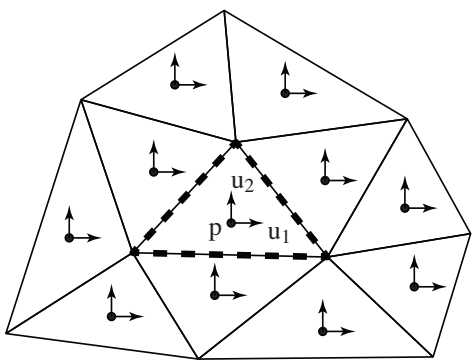
$$\nabla \cdot \mathbf{u} = 0, \qquad \partial_t \mathbf{u} + (\mathbf{u} \cdot \nabla)\mathbf{u} - \frac{1}{Re}\Delta \mathbf{u} + \nabla p = 0.$$

Finite-volume, algebra-based discretization on arbitrary collocated mesh[1]:

$$\mathsf{M}\boldsymbol{u}_s = \mathbf{0}_c, \qquad \Omega_c^{3d} d_t \boldsymbol{u}_c + \mathsf{C}_c^{3d}(\boldsymbol{u}_s)\boldsymbol{u}_c + \mathsf{D}_c^{3d}\boldsymbol{u}_c - \Omega_c^{3d}\mathsf{G}_c\boldsymbol{p}_c = \mathbf{0}_c,$$

where:

$$\text{cells} \in \mathbb{R}^n \quad \text{faces} \in \mathbb{R}^m$$



$$\boldsymbol{p}_c \in \mathbb{R}^n \quad \boldsymbol{u}_c \in \mathbb{R}^{3n} = (\boldsymbol{u}_1, \boldsymbol{u}_2, \boldsymbol{u}_3)^T \quad \boldsymbol{u}_s \in \mathbb{R}^m$$

$$\mathsf{G}_c \in \mathbb{R}^{3n \times n} \quad \mathsf{M} \in \mathbb{R}^{n \times m}$$

$$\Omega_c^{3d} \in \mathbb{R}^{3n \times 3n} = \mathsf{I}_3 \otimes \Omega_c \quad \mathsf{C}_c^{3d}(\boldsymbol{u}_s) \in \mathbb{R}^{3n \times 3n} = \mathsf{I}_3 \otimes \mathsf{C}_c(\boldsymbol{u}_s) \quad \mathsf{D}_c^{3d} \in \mathbb{R}^{3n \times 3n} = \mathsf{I}_3 \otimes \mathsf{D}_c$$

[1] Trias et al., Symmetry-preserving discretization of Navier-Stokes equations on collocated unstructured grids, J.Comp.Phys., 258, 246-267, 2014.

Algorithm to solve one time-integration step. The particular choice of the time-integration scheme is not relevant. Here, for the sake of simplicity, we have adopted a second-order Adams-Bashforth. Similarly, the particular choice of the Poisson solver will eventually depend on many factors. We have considered the Conjugate Gradient. Nevertheless, most existing sparse linear solvers algorithms rely on basic linear algebra operations.

Algorithm 1 Algorithm to solve one time-integration step of Navier–Stokes equations.

1: Compute the convective and the diffusive terms of momentum Equation:
$\mathsf{R}\left(\boldsymbol{u}_s^n, \boldsymbol{u}_c^n\right) \equiv -\mathsf{C}_c^{3d}\left(\boldsymbol{u}_s^n\right)\boldsymbol{u}_c^n - \mathsf{D}_c^{3d}\boldsymbol{u}_c^n$

Algorithm to solve one time-integration step. The particular choice of the time-integration scheme is not relevant. Here, for the sake of simplicity, we have adopted a second-order Adams-Bashforth. Similarly, the particular choice of the Poisson solver will eventually depend on many factors. We have considered the Conjugate Gradient. Nevertheless, most existing sparse linear solvers algorithms rely on basic linear algebra operations.

Algorithm 2 Algorithm to solve one time-integration step of Navier–Stokes equations.

1: Compute the convective and the diffusive terms of momentum Equation:
$\mathsf{R}\left(\boldsymbol{u}_s^n, \boldsymbol{u}_c^n\right) \equiv -\mathsf{C}_c^{3d}\left(\boldsymbol{u}_s^n\right)\boldsymbol{u}_c^n - \mathsf{D}_c^{3d}\boldsymbol{u}_c^n$
2: Compute the predictor velocity:
$\boldsymbol{u}_c^p = \boldsymbol{u}_c^n + \Delta t\left\{\frac{3}{2}\mathsf{R}\left(\boldsymbol{u}_s^n, \boldsymbol{u}_c^n\right) - \frac{1}{2}\mathsf{R}\left(\boldsymbol{u}_s^{n-1}, \boldsymbol{u}_c^{n-1}\right)\right\}$

Algorithm to solve one time-integration step. The particular choice of the time-integration scheme is not relevant. Here, for the sake of simplicity, we have adopted a second-order Adams-Bashforth. Similarly, the particular choice of the Poisson solver will eventually depend on many factors. We have considered the Conjugate Gradient. Nevertheless, most existing sparse linear solvers algorithms rely on basic linear algebra operations.

Algorithm 3 Algorithm to solve one time-integration step of Navier–Stokes equations.

1: Compute the convective and the diffusive terms of momentum Equation:
$\mathsf{R}\left(\boldsymbol{u}_s^n, \boldsymbol{u}_c^n\right) \equiv -\mathsf{C}_c^{3d}\left(\boldsymbol{u}_s^n\right)\boldsymbol{u}_c^n - \mathsf{D}_c^{3d}\boldsymbol{u}_c^n$
2: Compute the predictor velocity:
$\boldsymbol{u}_c^p = \boldsymbol{u}_c^n + \Delta t\left\{\frac{3}{2}\mathsf{R}\left(\boldsymbol{u}_s^n, \boldsymbol{u}_c^n\right) - \frac{1}{2}\mathsf{R}\left(\boldsymbol{u}_s^{n-1}, \boldsymbol{u}_c^{n-1}\right)\right\}$
3: Solve the Poisson equation:
$\mathsf{L}\tilde{\boldsymbol{p}}_c^{n+1} = \mathsf{M}\boldsymbol{u}_s^p$ where $\boldsymbol{u}_s^p = \Gamma_{c\to s}\boldsymbol{u}_c^p$

Algorithm to solve one time-integration step. The particular choice of the time-integration scheme is not relevant. Here, for the sake of simplicity, we have adopted a second-order Adams-Bashforth. Similarly, the particular choice of the Poisson solver will eventually depend on many factors. We have considered the Conjugate Gradient. Nevertheless, most existing sparse linear solvers algorithms rely on basic linear algebra operations.

Algorithm 4 Algorithm to solve one time-integration step of Navier–Stokes equations.

1: Compute the convective and the diffusive terms of momentum Equation:
$\mathsf{R}\left(\boldsymbol{u}_s^n, \boldsymbol{u}_c^n\right) \equiv -\mathsf{C}_c^{3d}\left(\boldsymbol{u}_s^n\right)\boldsymbol{u}_c^n - \mathsf{D}_c^{3d}\boldsymbol{u}_c^n$
2: Compute the predictor velocity:
$\boldsymbol{u}_c^p = \boldsymbol{u}_c^n + \Delta t\left\{\frac{3}{2}\mathsf{R}\left(\boldsymbol{u}_s^n, \boldsymbol{u}_c^n\right) - \frac{1}{2}\mathsf{R}\left(\boldsymbol{u}_s^{n-1}, \boldsymbol{u}_c^{n-1}\right)\right\}$
3: Solve the Poisson equation:
$\mathsf{L}\tilde{\boldsymbol{p}}_c^{n+1} = \mathsf{M}\boldsymbol{u}_s^p$ where $\boldsymbol{u}_s^p = \Gamma_{c\to s}\boldsymbol{u}_c^p$
4: Correct the staggered velocity field:
$\boldsymbol{u}_s^{n+1} = \boldsymbol{u}_s^p - \mathsf{G}\tilde{\boldsymbol{p}}_c^{n+1}$ where $\mathsf{G} = -\Omega_s^{-1}\mathsf{M}^T$

Algorithm to solve one time-integration step. The particular choice of the time-integration scheme is not relevant. Here, for the sake of simplicity, we have adopted a second-order Adams-Bashforth. Similarly, the particular choice of the Poisson solver will eventually depend on many factors. We have considered the Conjugate Gradient. Nevertheless, most existing sparse linear solvers algorithms rely on basic linear algebra operations.

Algorithm 5 Algorithm to solve one time-integration step of Navier–Stokes equations.

1: Compute the convective and the diffusive terms of momentum Equation:
$\mathsf{R}\left(\boldsymbol{u}_s^n, \boldsymbol{u}_c^n\right) \equiv -\mathsf{C}_c^{3d}\left(\boldsymbol{u}_s^n\right)\boldsymbol{u}_c^n - \mathsf{D}_c^{3d}\boldsymbol{u}_c^n$
2: Compute the predictor velocity:
$\boldsymbol{u}_c^p = \boldsymbol{u}_c^n + \Delta t\left\{\frac{3}{2}\mathsf{R}\left(\boldsymbol{u}_s^n, \boldsymbol{u}_c^n\right) - \frac{1}{2}\mathsf{R}\left(\boldsymbol{u}_s^{n-1}, \boldsymbol{u}_c^{n-1}\right)\right\}$
3: Solve the Poisson equation:
$\mathsf{L}\tilde{\boldsymbol{p}}_c^{n+1} = \mathsf{M}\boldsymbol{u}_s^p$ where $\boldsymbol{u}_s^p = \Gamma_{c\to s}\boldsymbol{u}_c^p$
4: Correct the staggered velocity field:
$\boldsymbol{u}_s^{n+1} = \boldsymbol{u}_s^p - \mathsf{G}\tilde{\boldsymbol{p}}_c^{n+1}$ where $\mathsf{G} = -\Omega_s^{-1}\mathsf{M}^T$
5: Correct the cell centered velocity field:
$\boldsymbol{u}_c^{n+1} = \boldsymbol{u}_c^p - \mathsf{G}_c\tilde{\boldsymbol{p}}_c^{n+1}$ where $\mathsf{G}_c = -\Gamma_{s\to c}\Omega_s^{-1}\mathsf{M}^T$

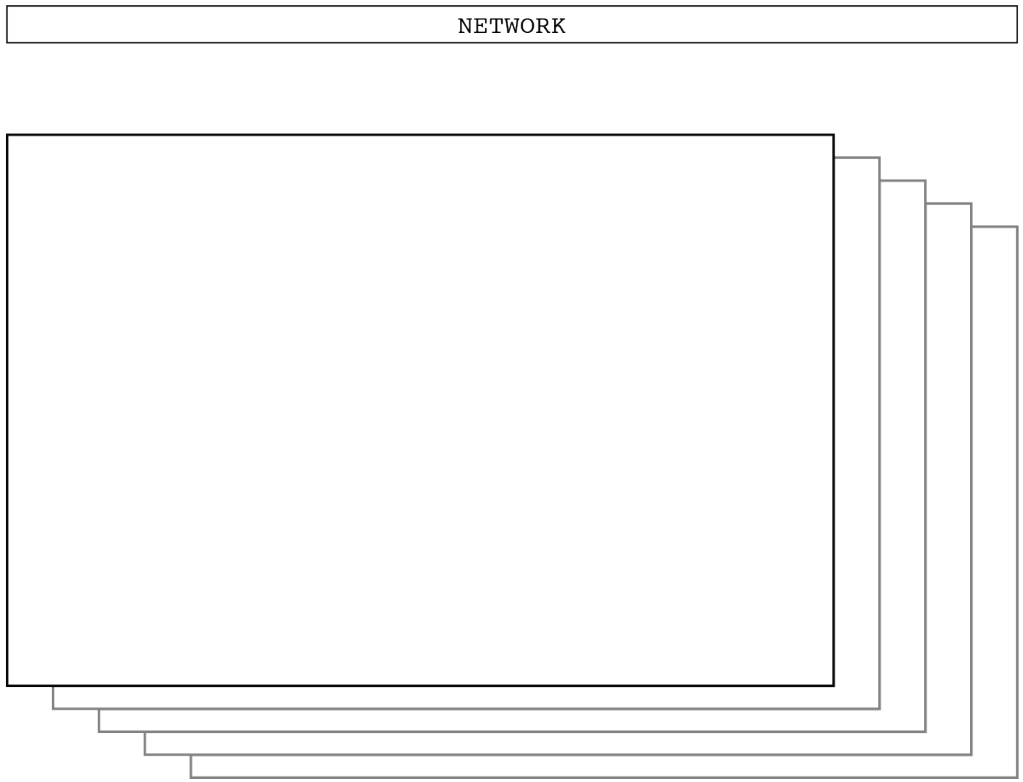
Algorithm to solve one time-integration step. The particular choice of the time-integration scheme is not relevant. Here, for the sake of simplicity, we have adopted a second-order Adams-Bashforth. Similarly, the particular choice of the Poisson solver will eventually depend on many factors. We have considered the Conjugate Gradient. Nevertheless, most existing sparse linear solvers algorithms rely on basic linear algebra operations.

Step of Algorithm	SpMV	axpy	dot
1. Compute the convective–difusive terms	12	0	0
2. Predictor velocity	0	6	0
3. Poisson equation (r.h.s)	4	2	0
*. Poisson equation (per iteration)	2	3	2
4. Velocity correction (staggered)	1	1	0
5. Velocity correction (centred)	3	3	0
Total outside Poisson solver	20	12	2

Implementation

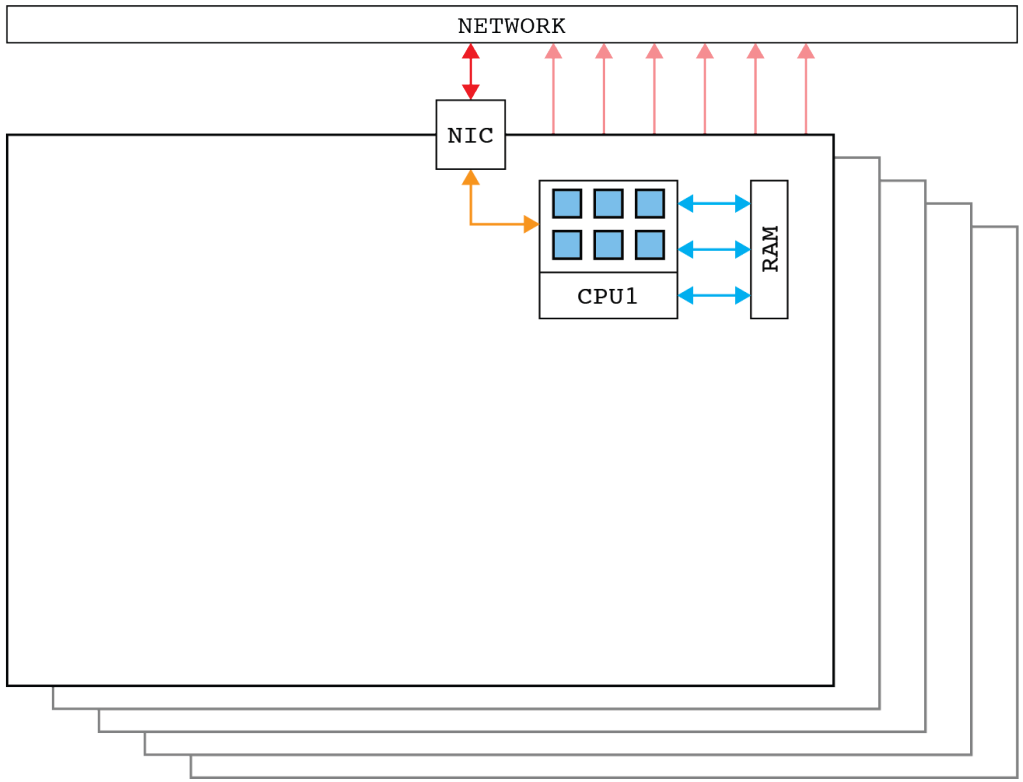
Hybrid HPC systems

Modern HPC systems consist of multiple hybrid computing nodes interconnected via a communication infrastructure. The nodes are composed of many hardware devices of different architectures, such as central processing unit (CPU) or graphics processing unit (GPU), among others.



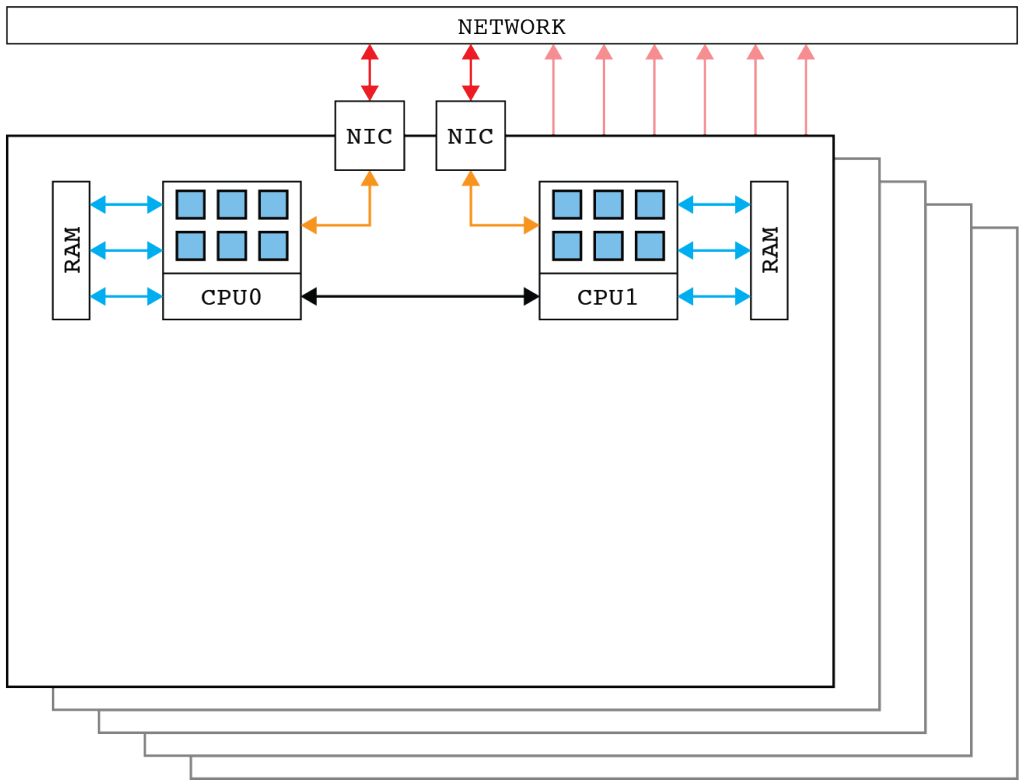
Hybrid HPC systems

Modern HPC systems consist of multiple hybrid computing nodes interconnected via a communication infrastructure. The nodes are composed of many hardware devices of different architectures, such as central processing unit (CPU) or graphics processing unit (GPU), among others.



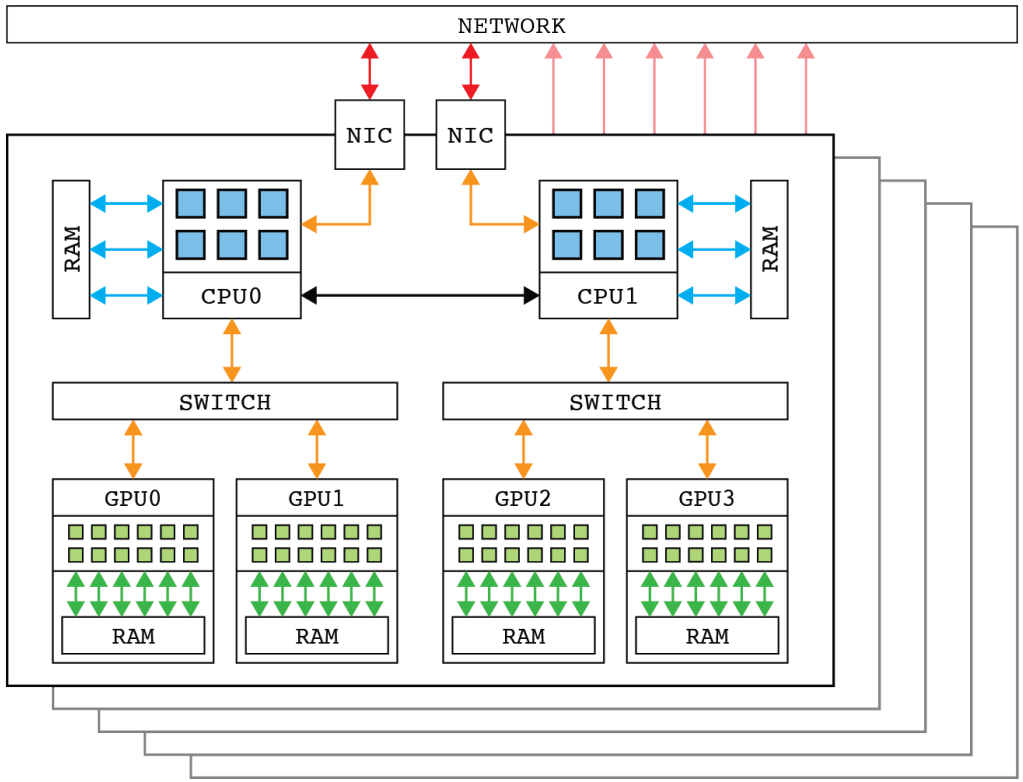
Hybrid HPC systems

Modern HPC systems consist of multiple hybrid computing nodes interconnected via a communication infrastructure. The nodes are composed of many hardware devices of different architectures, such as central processing unit (CPU) or graphics processing unit (GPU), among others.



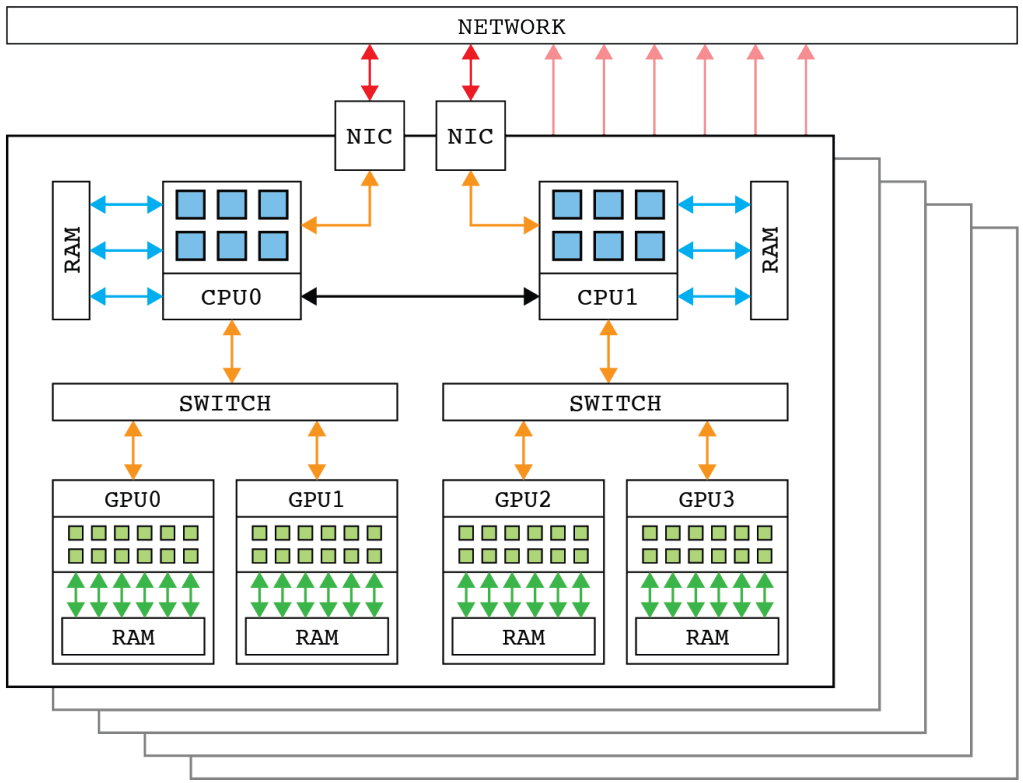
Hybrid HPC systems

Modern HPC systems consist of multiple hybrid computing nodes interconnected via a communication infrastructure. The nodes are composed of many hardware devices of different architectures, such as central processing unit (CPU) or graphics processing unit (GPU), among others.



Hybrid HPC systems

Modern HPC systems consist of multiple hybrid computing nodes interconnected via a communication infrastructure. The nodes are composed of many hardware devices of different architectures, such as central processing unit (CPU) or graphics processing unit (GPU), among others.



The algorithms must be compatible with distributed- and shared-memory multiple instruction, multiple data (DMMIMD and SMMIMD, respectively) parallelism, and more importantly, with stream processing (SP).

The **hpc2lib**'s implementation model is based on three types of objects:

The **hpc2lib**'s implementation model is based on three types of objects:

- Shaper: some sort of instructions manual that allows transforming any initial sequential input into a set of hierarchical partitions enabled for parallel processing.

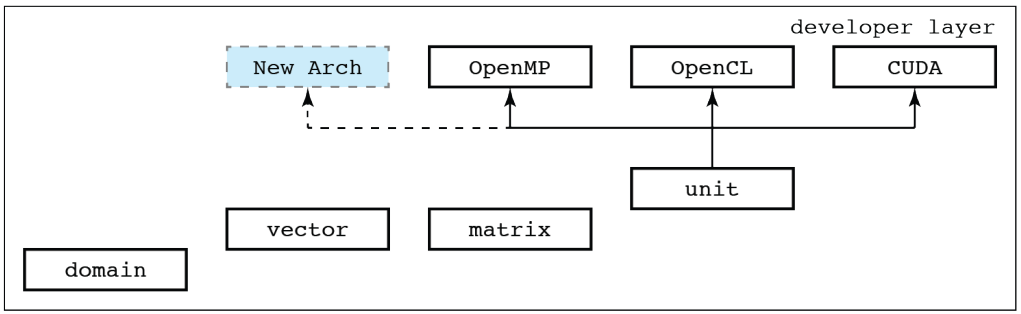
The **hpc2lib**'s implementation model is based on three types of objects:

- Shaper: some sort of instructions manual that allows transforming any initial sequential input into a set of hierarchical partitions enabled for parallel processing.
- Container: data storage objects that stock such hierarchical partitions.

The **hpc2lib**'s implementation model is based on three types of objects:

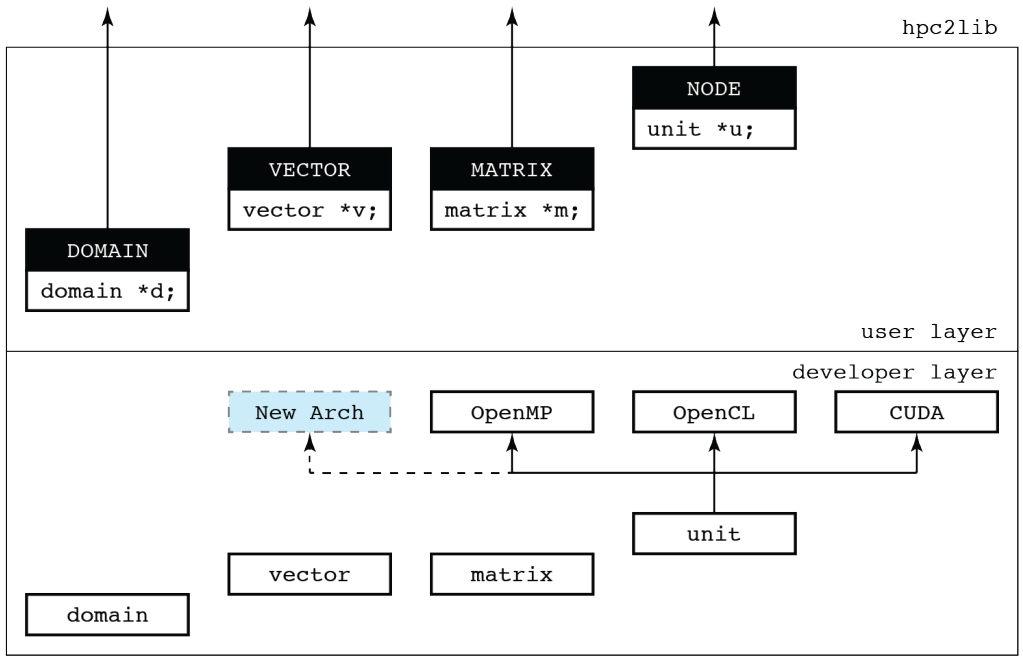
- Shaper: some sort of instructions manual that allows transforming any initial sequential input into a set of hierarchical partitions enabled for parallel processing.
- Container: data storage objects that stock such hierarchical partitions.
- Actuator: provides with methods and functions to firstly create shapes and then manipulate and operate containers

hpc2lib



The **hpc2lib**'s implementation model is based on three types of objects:

- Shaper: some sort of instructions manual that allows transforming any initial sequential input into a set of hierarchical partitions enabled for parallel processing.
- Container: data storage objects that stock such hierarchical partitions.
- Actuator: provides with methods and functions to firstly create shapes and then manipulate and operate containers

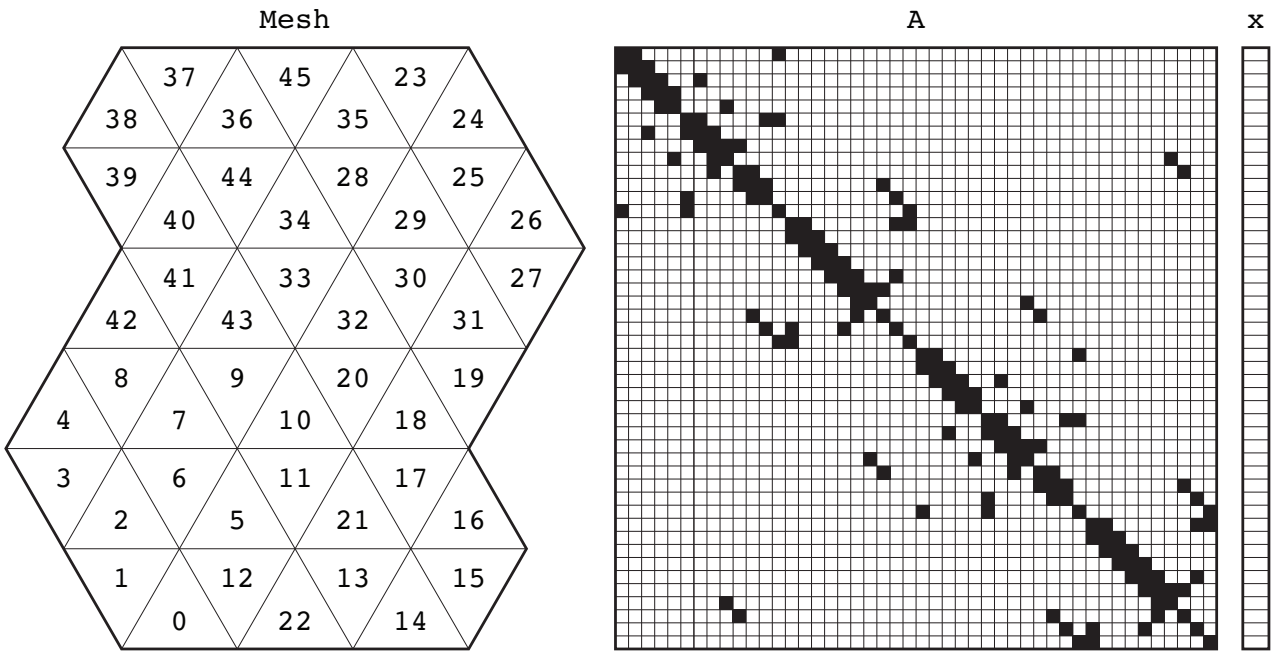


At the highest level, only wrappers or black boxes are offered to the user. Applications are system-agnostic. Users can develop codes in a sequential fancy, like playing with std::vectors.

```
1  #include "hpc2lib.h"
2
3  /* NODE object is a singleton and is accessed through a pointer */
4  NODE *Node;
5
6  int main(int argc, char** argv){
7      /* initialize MPI stuff */
8      int prov, req = MPI_THREAD_MULTIPLE;
9      MPI_Init_thread(&argc, &argv, req, &prov);
10
11     /* initialize NODE with command line arguments */
12     Node->Init(&argc, &argv);
13
14     /* initialize HPC2 objects using plain sequential data in binary files */
15     DOMAIN Cells, Faces;
16     Node->CreateDomain(Cells, Faces, "input_G.bin");
17
18     VECTOR q = Node->BuildVector(Faces, 0.0);
19     VECTOR T = Node->BuildVector(Cells, "input_T.bin");
20     MATRIX G = Node->BuildMatrix(Cells, Faces, "input_G.bin");
21
22     /* compute the gradient */
23     double k = 0.598;
24     Node->SpMV(G, T, q, -k);
25
26     MPI_Finalize();
27 }
```

Multilevel domain decomposition

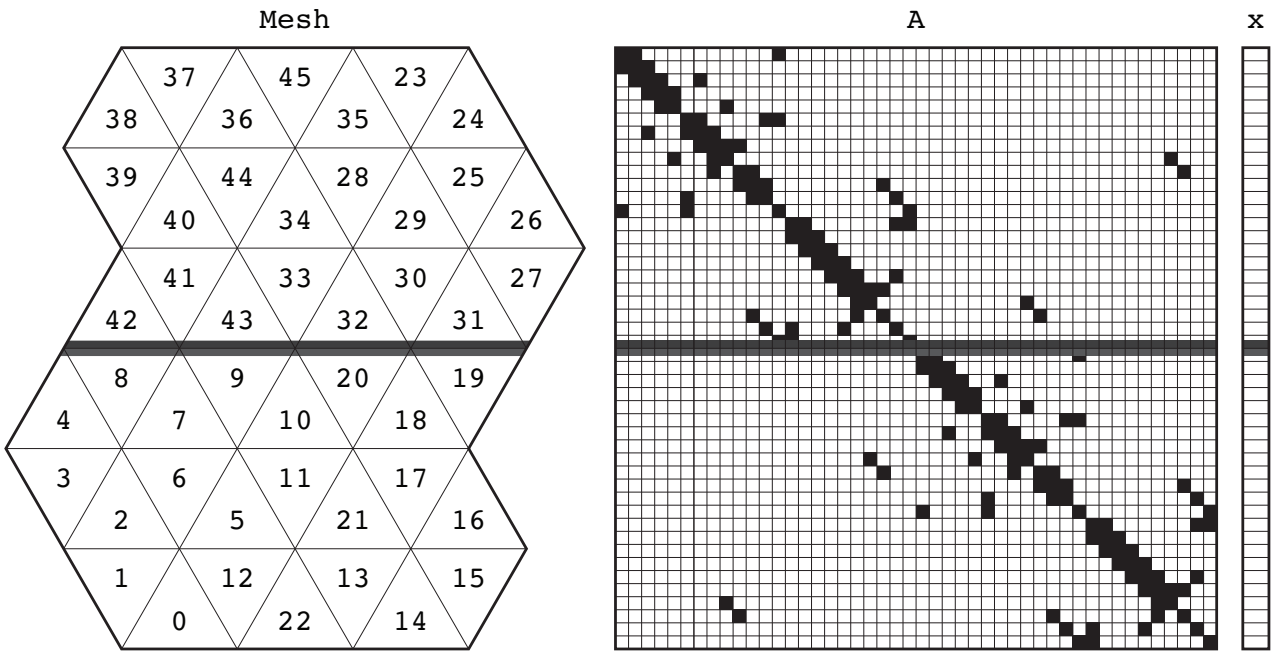
Multilevel workload distribution consists of dividing the computational domain (mesh) into subsets recursively to distribute it among the hardware of a computing system.



Multilevel domain decomposition

Multilevel workload distribution consists of dividing the computational domain (mesh) into subsets recursively to distribute it among the hardware of a computing system.

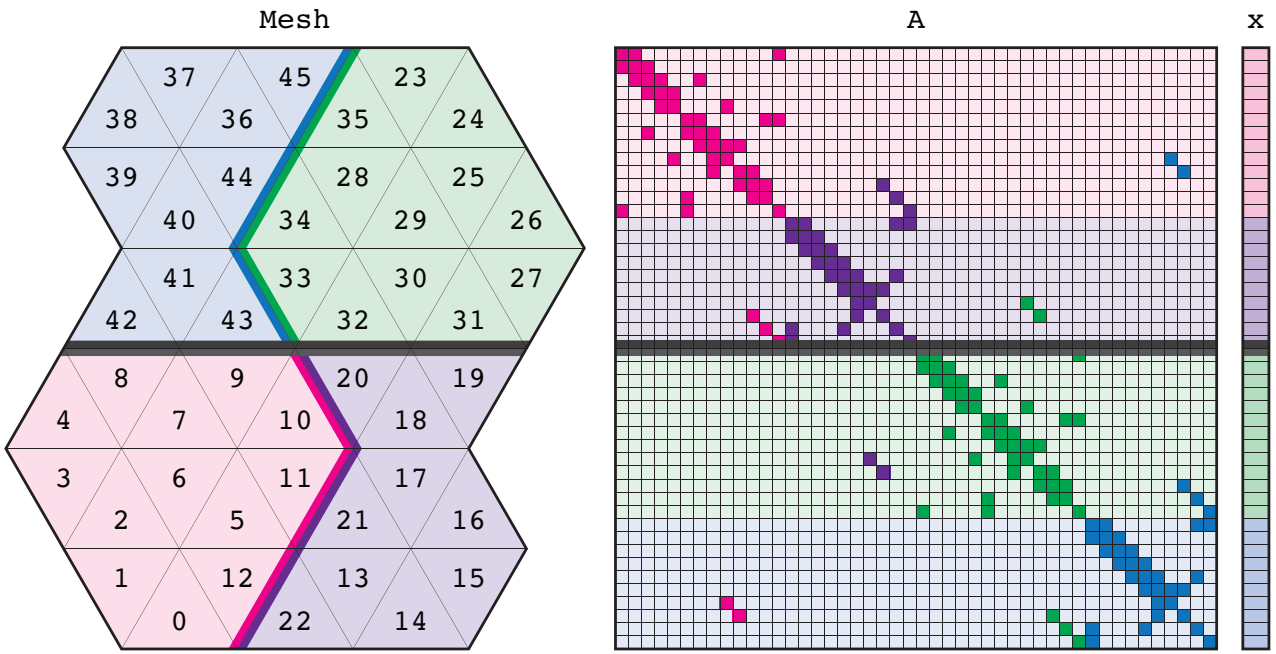
- First-level among computing nodes, *i.e.*, MPI processes.



Multilevel domain decomposition

Multilevel workload distribution consists of dividing the computational domain (mesh) into subsets recursively to distribute it among the hardware of a computing system.

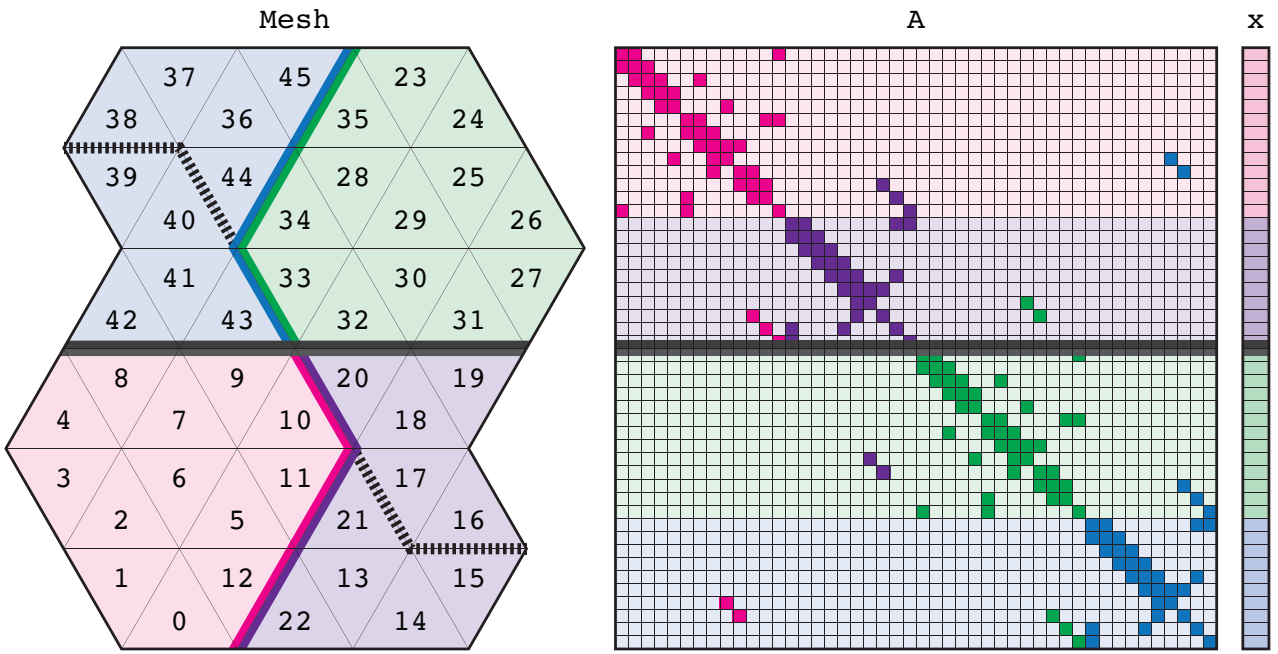
- First-level among computing nodes, *i.e.*, MPI processes.
- Second-level among computing units, *i.e.*, host and accelerators.



Multilevel domain decomposition

Multilevel workload distribution consists of dividing the computational domain (mesh) into subsets recursively to distribute it among the hardware of a computing system.

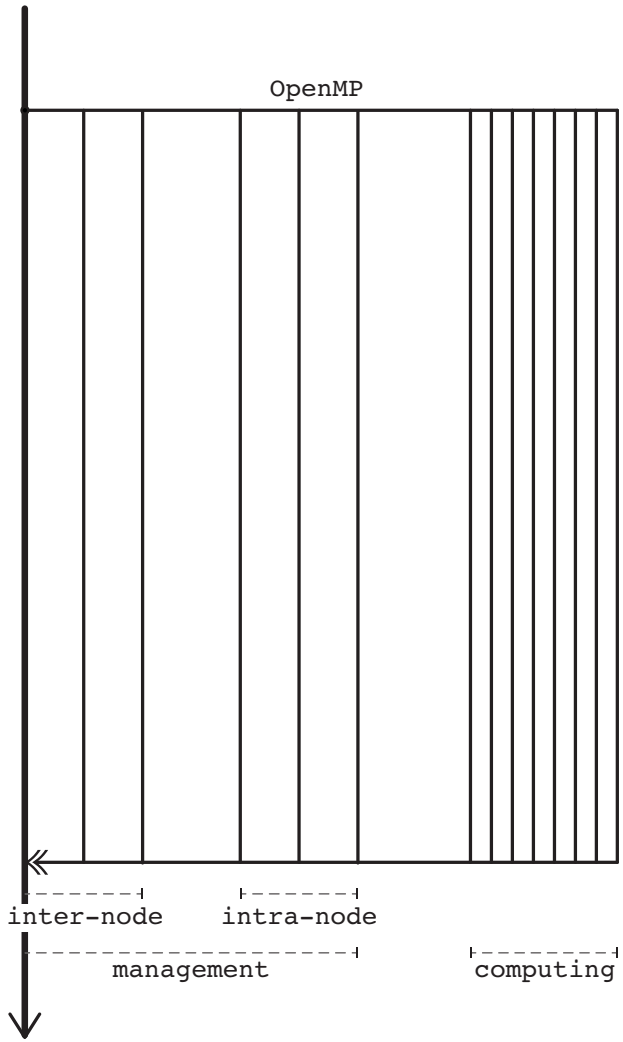
- First-level among computing nodes, *i.e.*, MPI processes.
- Second-level among computing units, *i.e.*, host and accelerators.
- Third-level among threads in NUMA shared-memory spaces.



Every thread is assigned a fixed chunk of data. Thread affinity, malloc, and first touch policy grant memory locality.

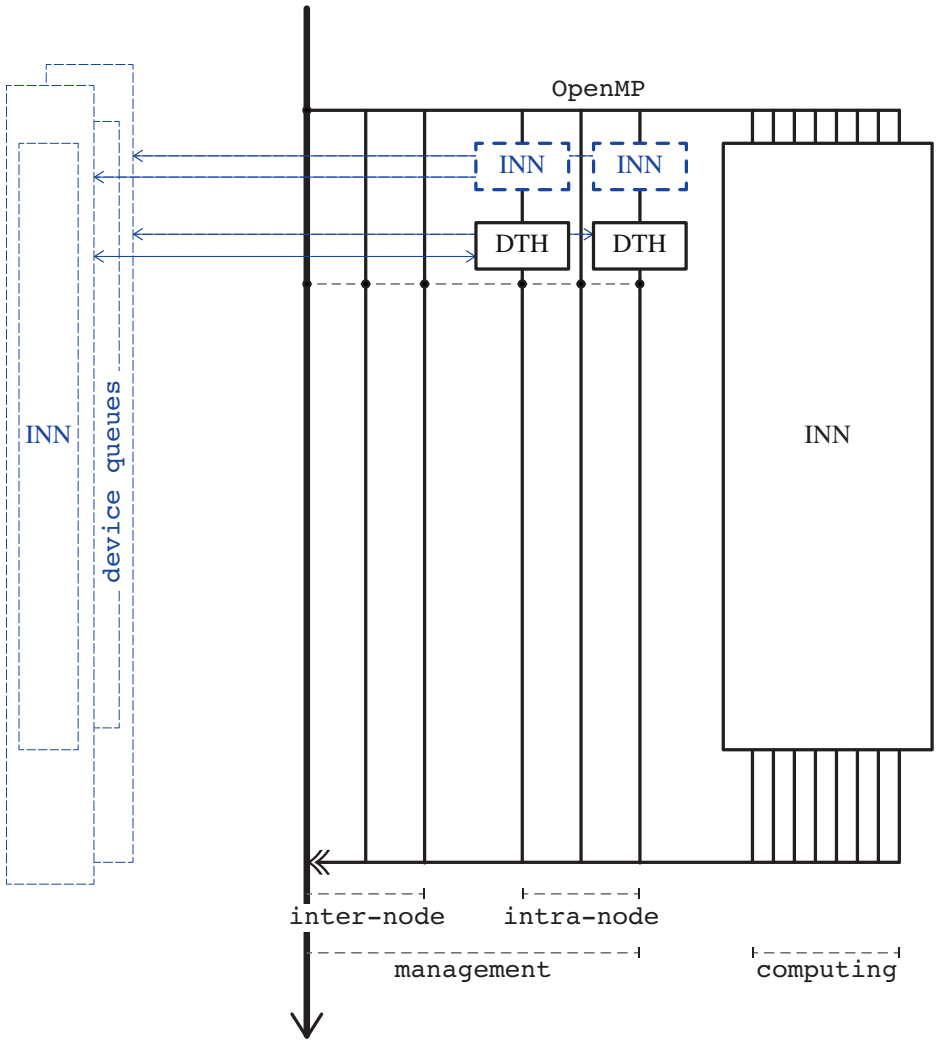
Flat multithreaded execution strategies

To minimise the overhead of the communications, efficient multithreaded execution strategies are required. Roughly, the idea is to overlap the communications with the computations.



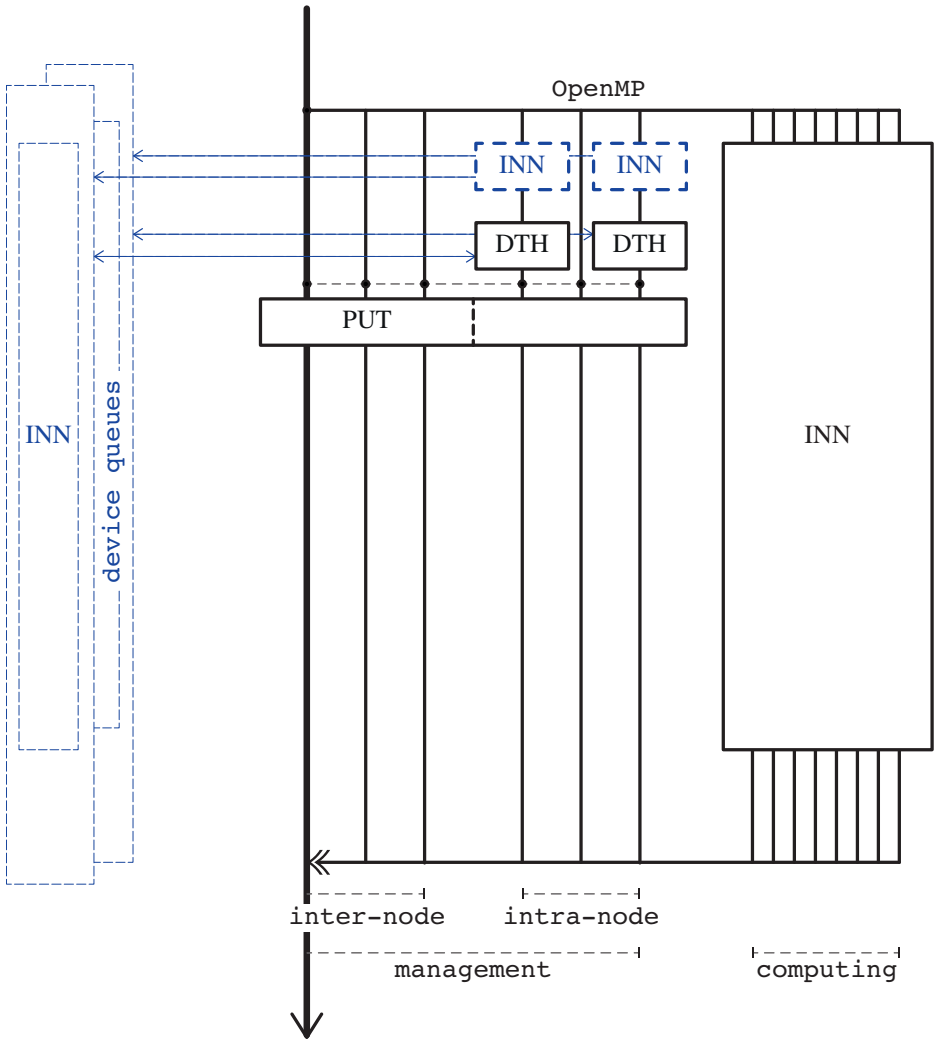
Flat multithreaded execution strategies

To minimise the overhead of the communications, efficient multithreaded execution strategies are required. Roughly, the idea is to overlap the communications with the computations.



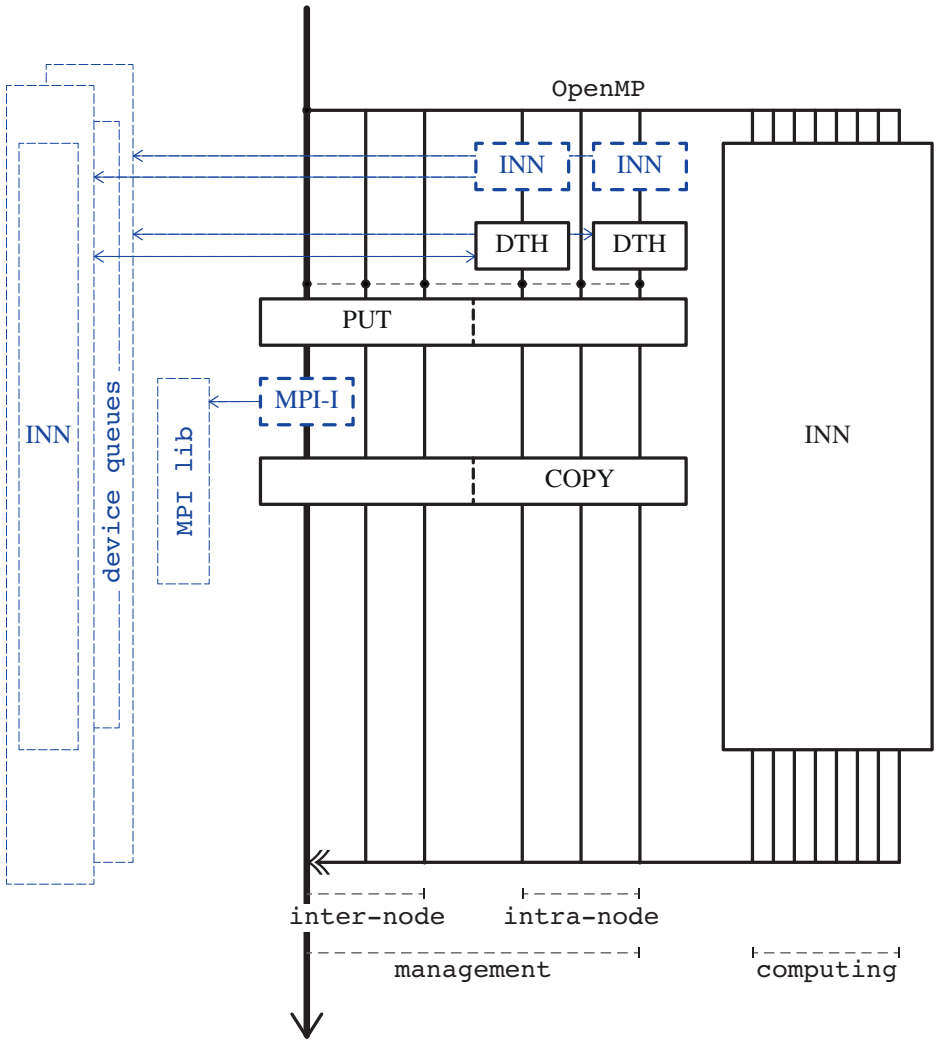
Flat multithreaded execution strategies

To minimise the overhead of the communications, efficient multithreaded execution strategies are required. Roughly, the idea is to overlap the communications with the computations.



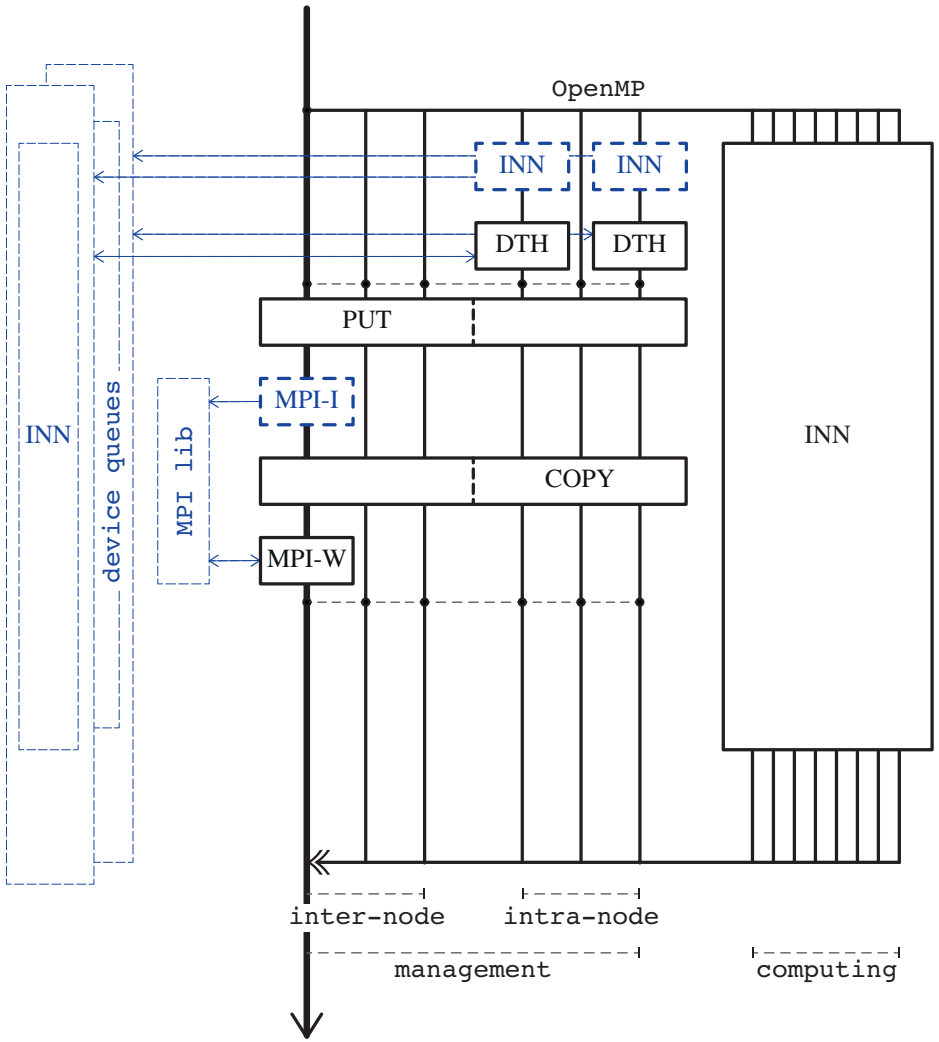
Flat multithreaded execution strategies

To minimise the overhead of the communications, efficient multithreaded execution strategies are required. Roughly, the idea is to overlap the communications with the computations.



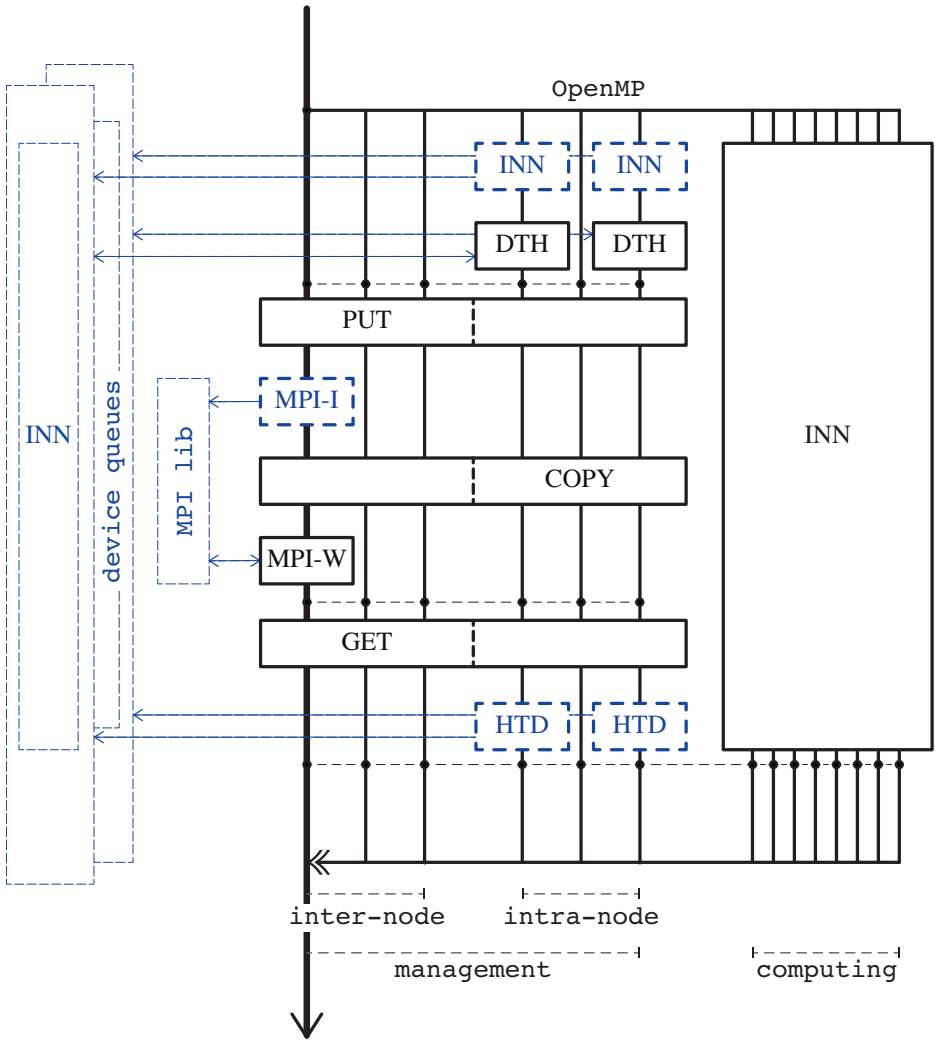
Flat multithreaded execution strategies

To minimise the overhead of the communications, efficient multithreaded execution strategies are required. Roughly, the idea is to overlap the communications with the computations.



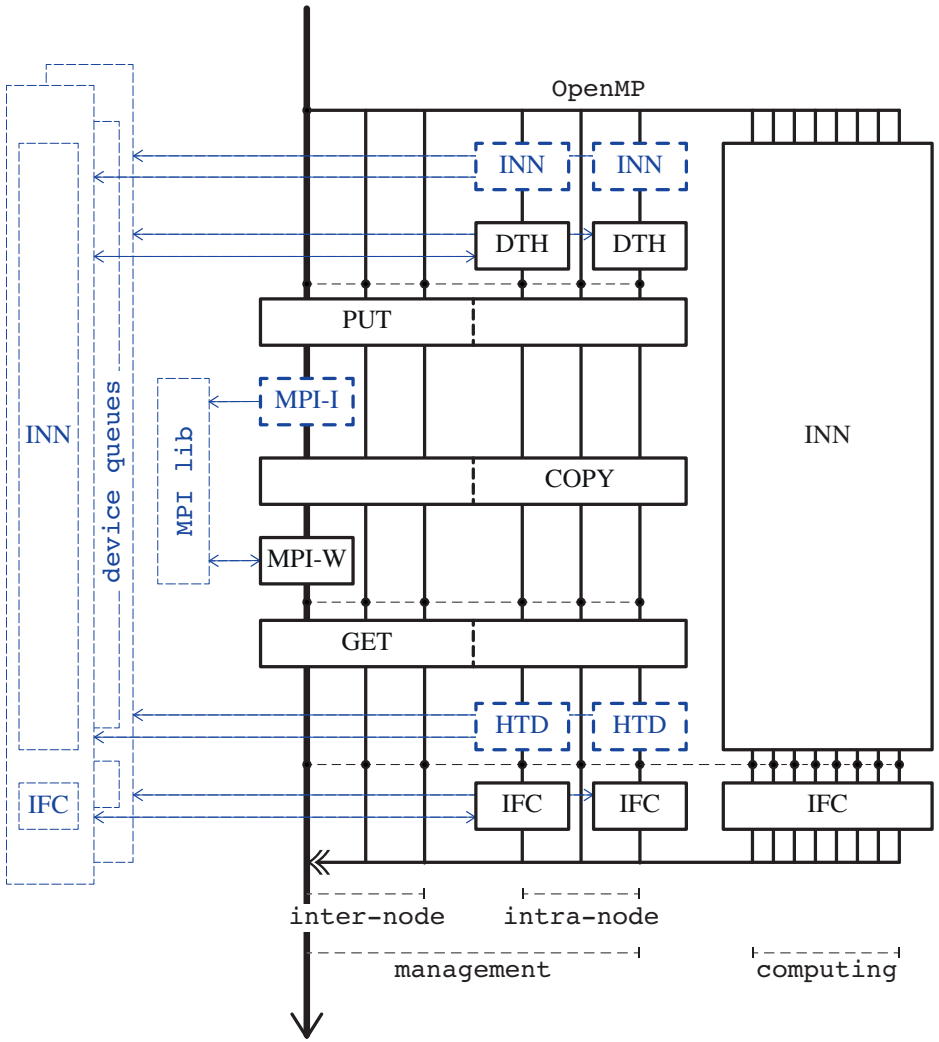
Flat multithreaded execution strategies

To minimise the overhead of the communications, efficient multithreaded execution strategies are required. Roughly, the idea is to overlap the communications with the computations.



Flat multithreaded execution strategies

To minimise the overhead of the communications, efficient multithreaded execution strategies are required. Roughly, the idea is to overlap the communications with the computations.



Performance analysis

System parametrization

Kernel parametrization

System parametrization

- π: peak performance.

Kernel parametrization

System parametrization

- π: peak performance.
- β: memory bandwidth.

Kernel parametrization

System parametrization

- π: peak performance.
- β: memory bandwidth.
- λ: memory latency.

Kernel parametrization

System parametrization

- π: peak performance.
- β: memory bandwidth.
- λ: memory latency.

Kernel parametrization

- W: number of operations.

System parametrization

- π: peak performance.
- β: memory bandwidth.
- λ: memory latency.

Kernel parametrization

- W: number of operations.
- Q: memory traffic.

System parametrization

- π: peak performance.
- β: memory bandwidth.
- λ: memory latency.

Kernel parametrization

- W: number of operations.
- Q: memory traffic.
- X: data exchange requirements.

System parametrization

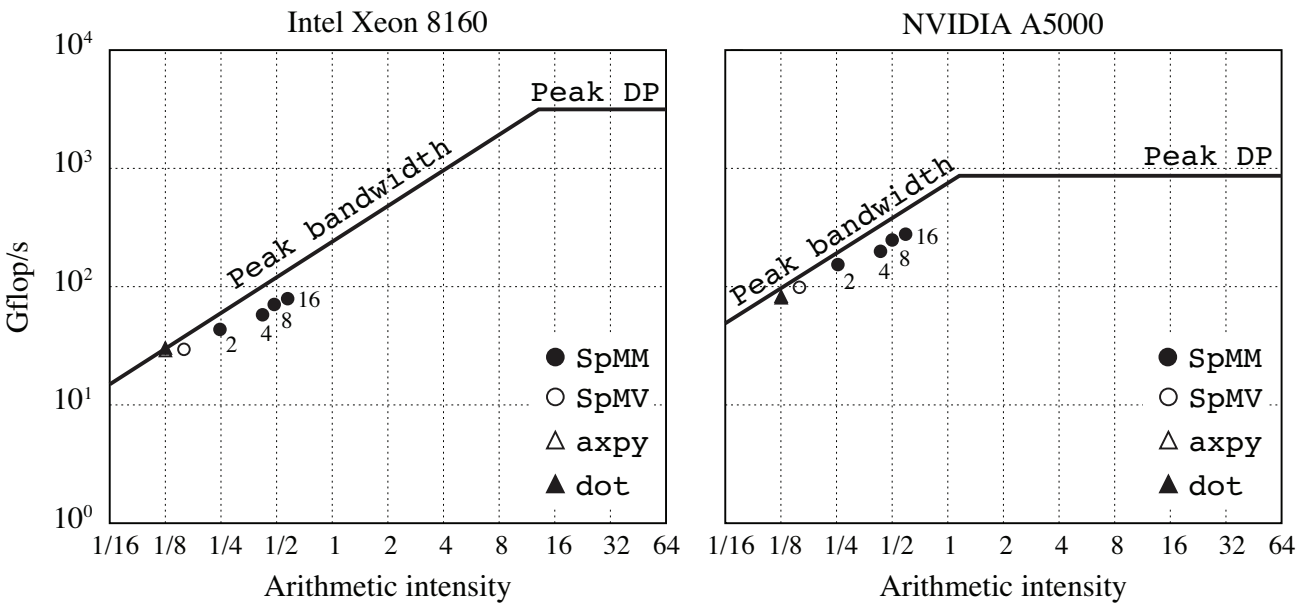
- π: peak performance.
- β: memory bandwidth.
- λ: memory latency.

Kernel parametrization

- W: number of operations.
- Q: memory traffic.
- X: data exchange requirements.

On single-devices, the performance is typically estimated with the Roofline model:

$$\pi_k = min(\pi_u, AI_k \beta_u),$$



System parametrization

- π: peak performance.
- β: memory bandwidth.
- λ: memory latency.

Kernel parametrization

- W: number of operations.
- Q: memory traffic.
- X: data exchange requirements.

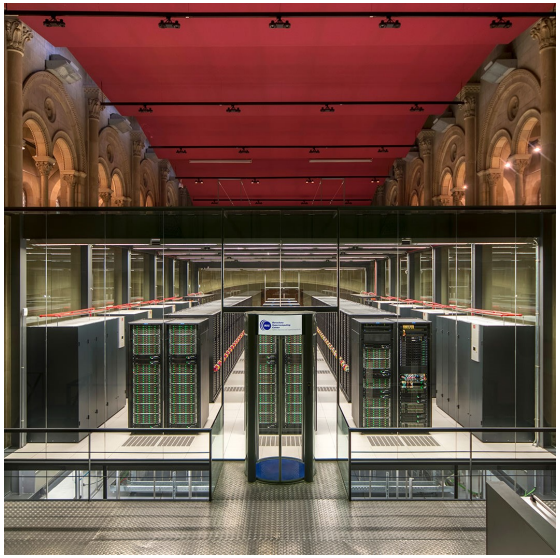
In distributed parallel processing, the parallel performance can be evaluated as follows:

$$\frac{t_k}{t_x} = \frac{Q_k}{X_n} \cdot \frac{\frac{1}{\beta_n}}{\frac{2}{\beta_l} + \frac{3 + 3\bar{\chi}u}{\beta_h} + \frac{\bar{\chi}}{\beta_x}} \geq 1.$$

With this formula we will estimate whether a specific distributed application, given the mesh size and decomposition, can scale or not.

Target HPC systems

MareNostrum 4



#rank82
3456 nodes with:

- 2× Intel Xeon 8160
- 1× Intel Omni-Path

TSUBAME3.0

#rank64
540 nodes with:

- 2× Intel Xeon E5-2680 v4
- 4× NVIDIA Tesla P100
- 4× Intel Omni-Path

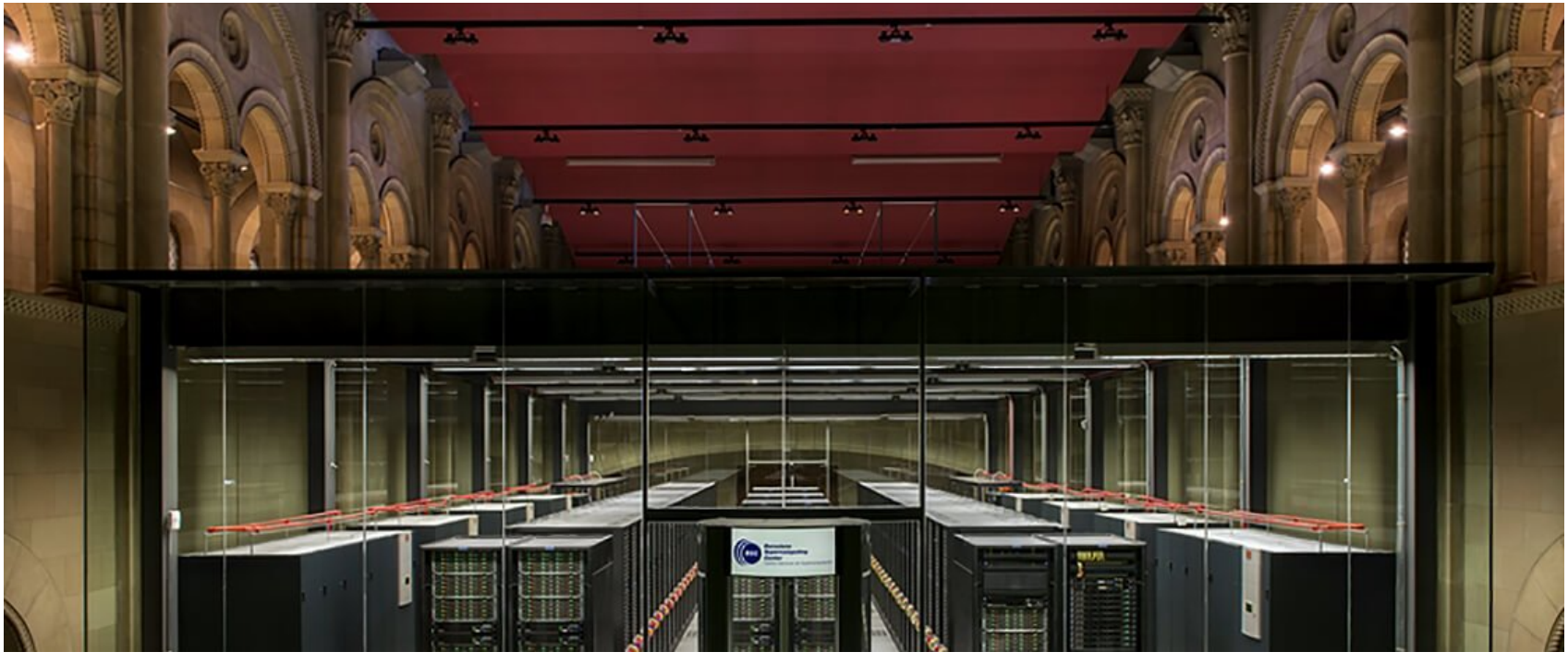
Lomonosov 2

#rank262
1696 nodes with:

- 1× Intel Xeon E5-2697 v3
- 1× NVIDIA Tesla K40M
- 1× InfiniBand FDR

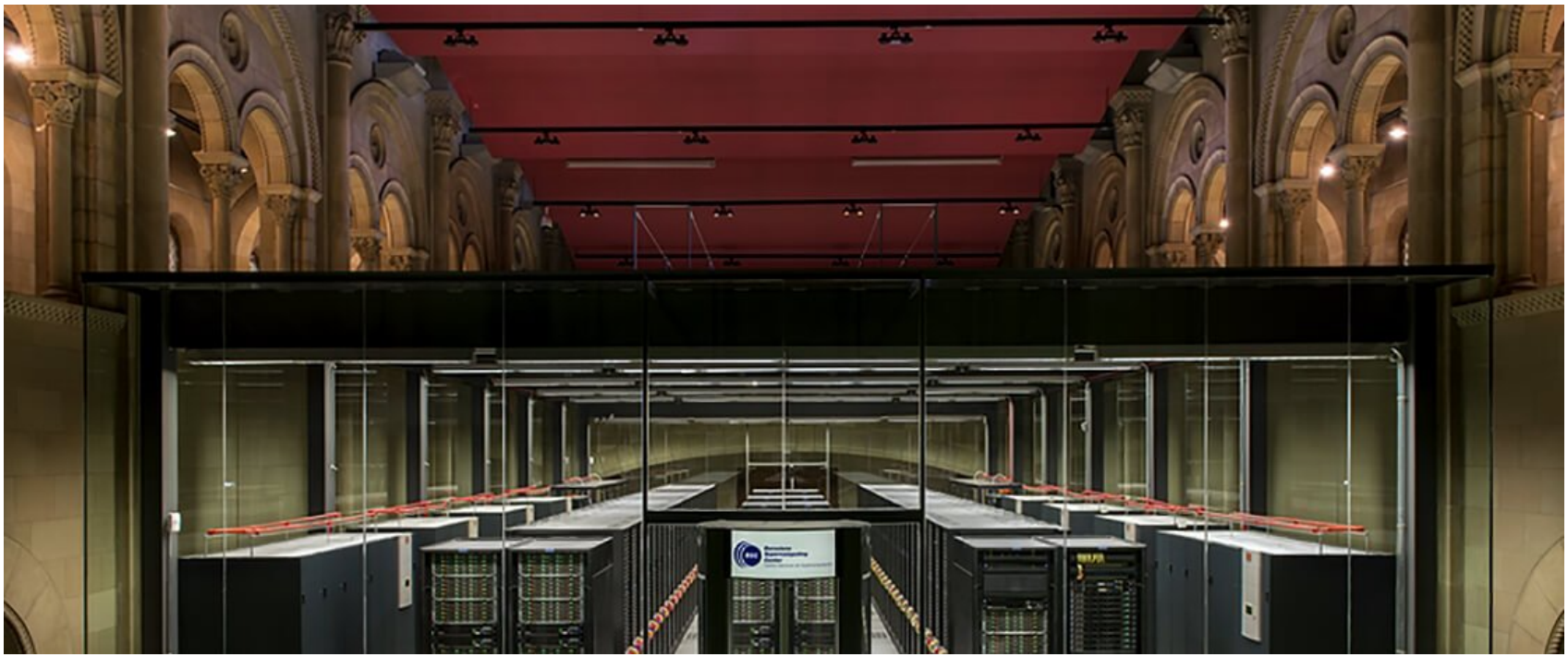
Other systems: JFF third- and fourth-generation, MareNostrum 3, MinoTauro, K60, Titan, Mira, Cori, Marconi100...

Testing conditions:



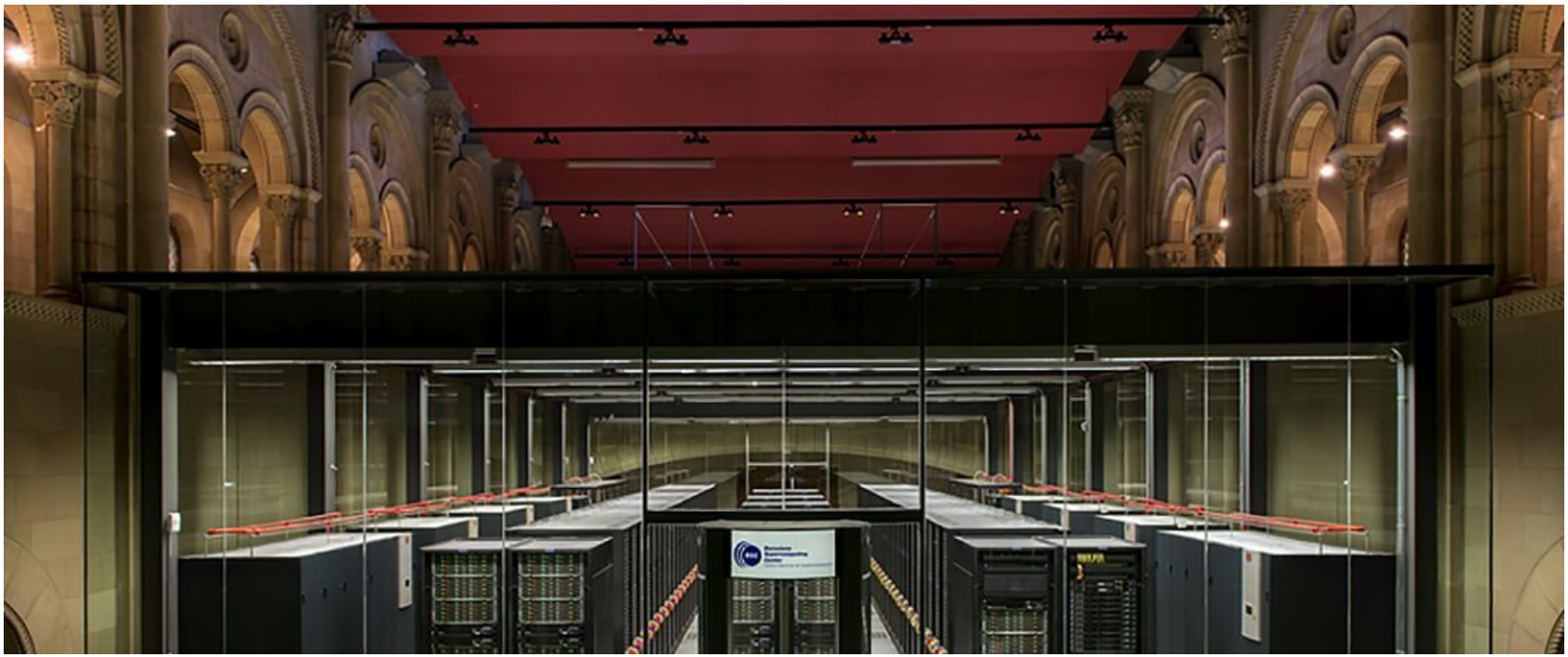
Testing conditions:

- Sparse matrices of symmetry-preserving discretization of Laplacian operator.



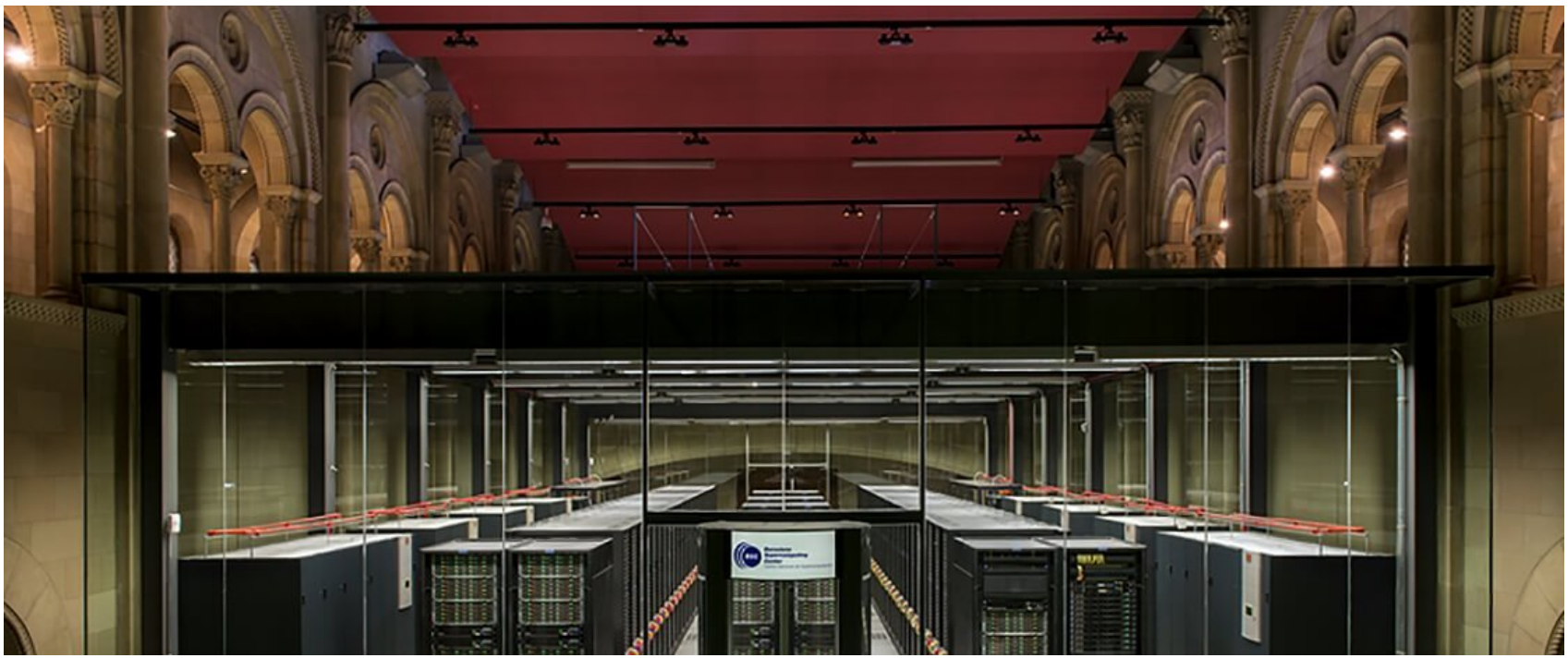
Testing conditions:

- Sparse matrices of symmetry-preserving discretization of Laplacian operator.
- Discretization on hex-dominant meshes.



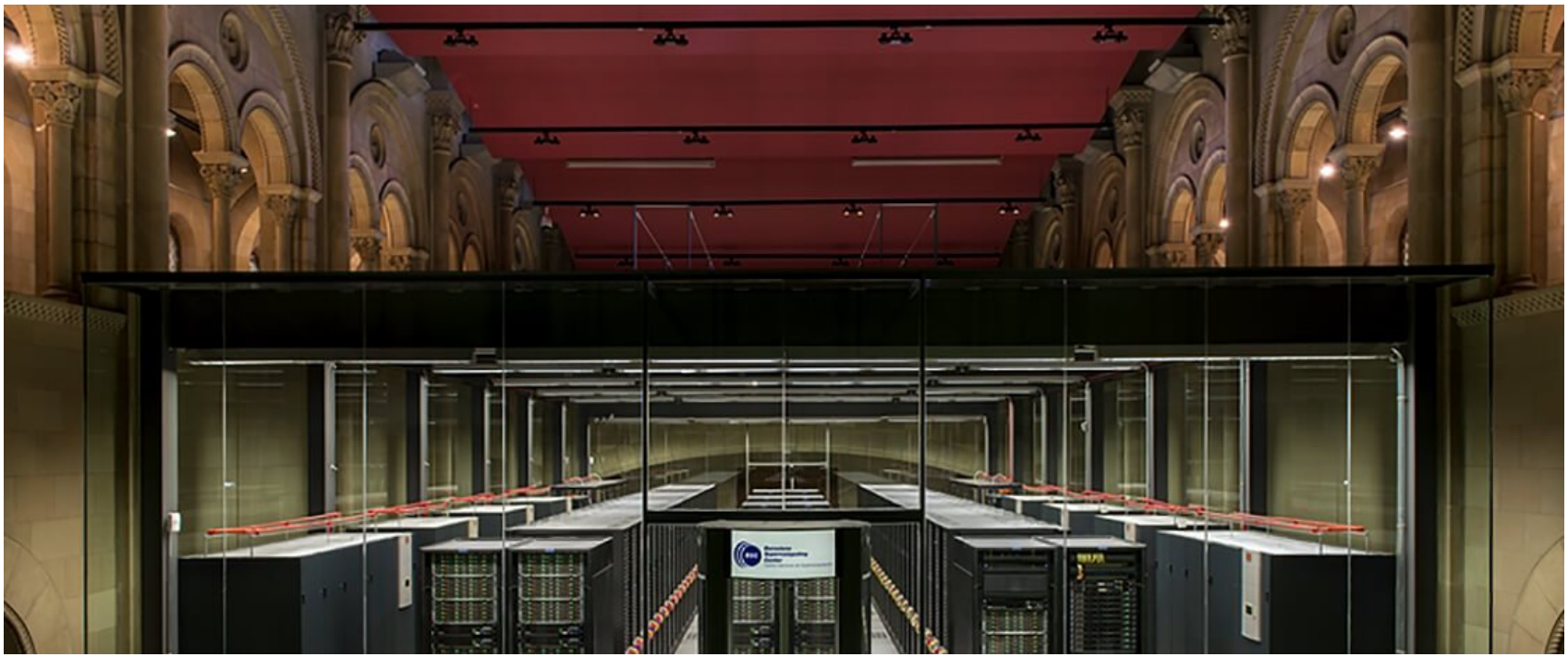
Testing conditions:

- Sparse matrices of symmetry-preserving discretization of Laplacian operator.
- Discretization on hex-dominant meshes.
- ELLPACK sparse matrix storage format is used.



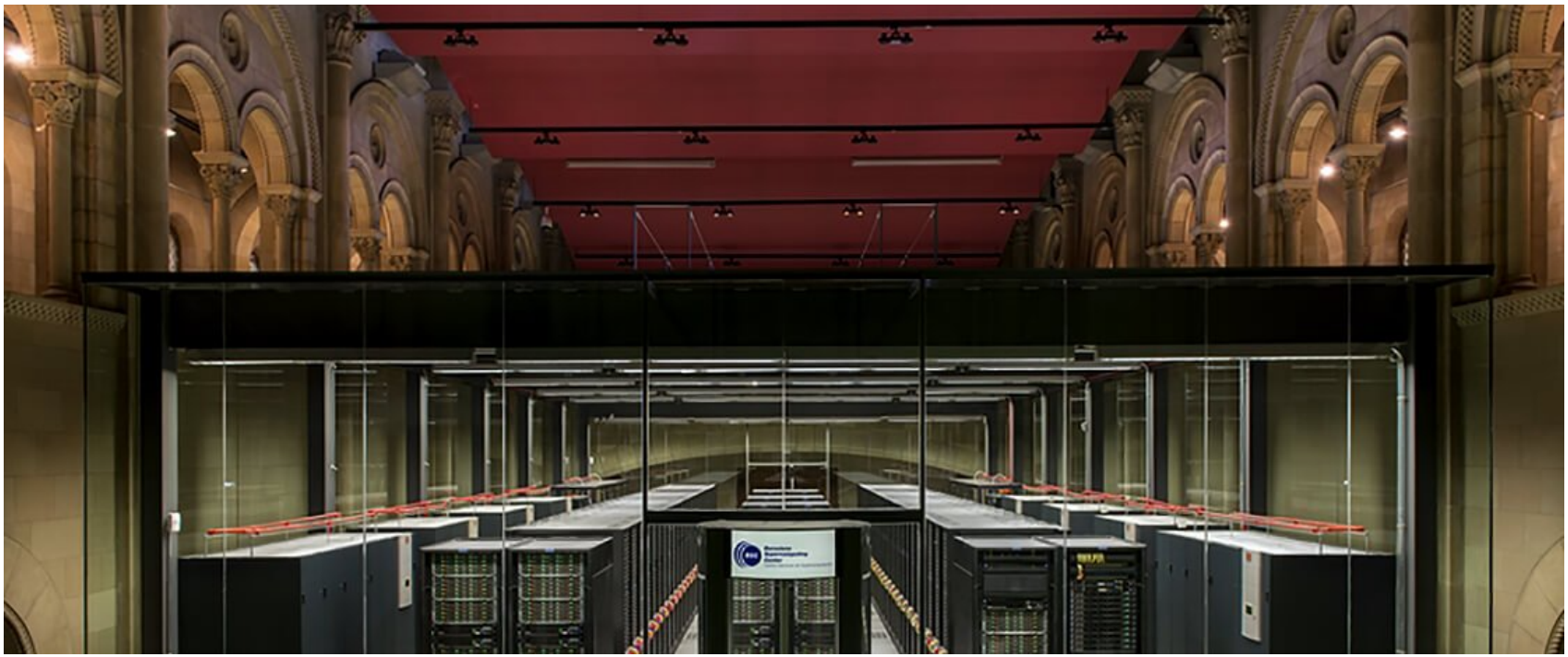
Testing conditions:

- Sparse matrices of symmetry-preserving discretization of Laplacian operator.
- Discretization on hex-dominant meshes.
- ELLPACK sparse matrix storage format is used.
- Problem is large enough to avoid cache reuse (CPU) and low occupancy (GPU).

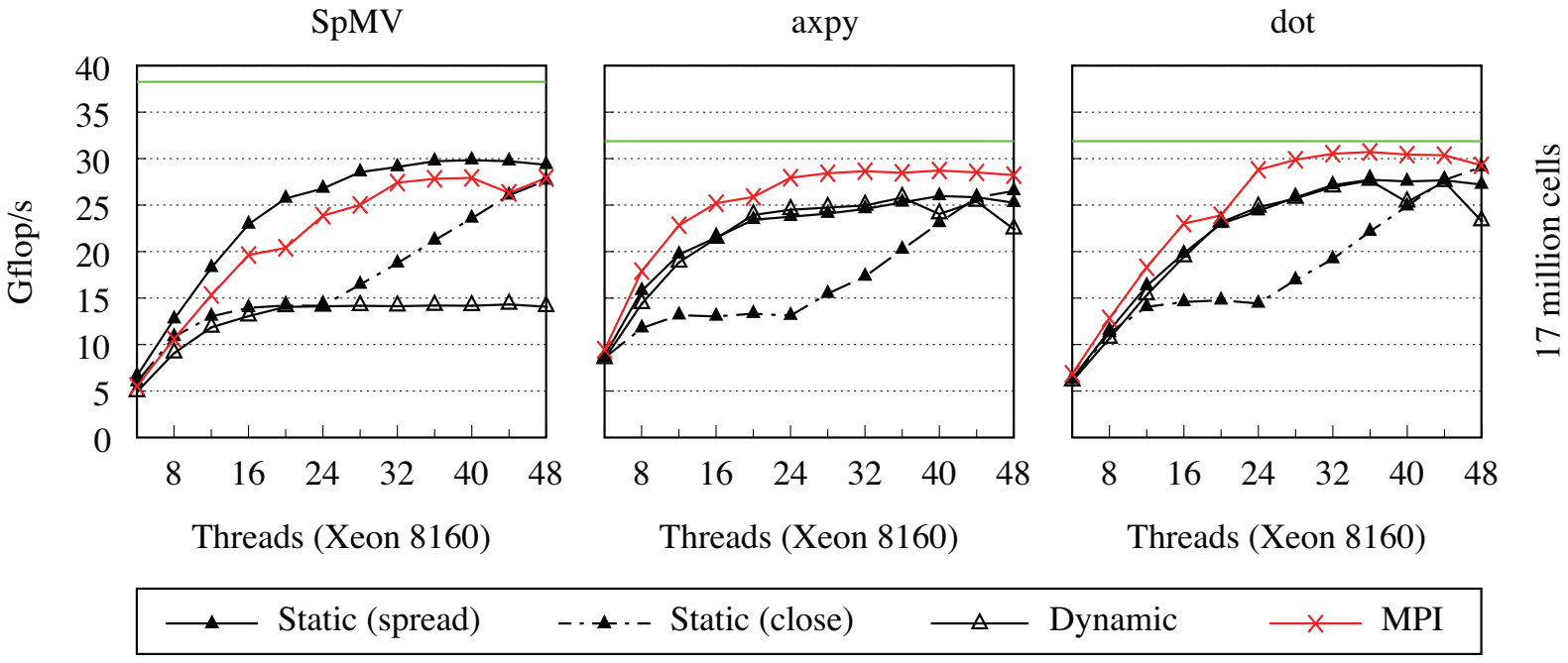


Testing conditions:

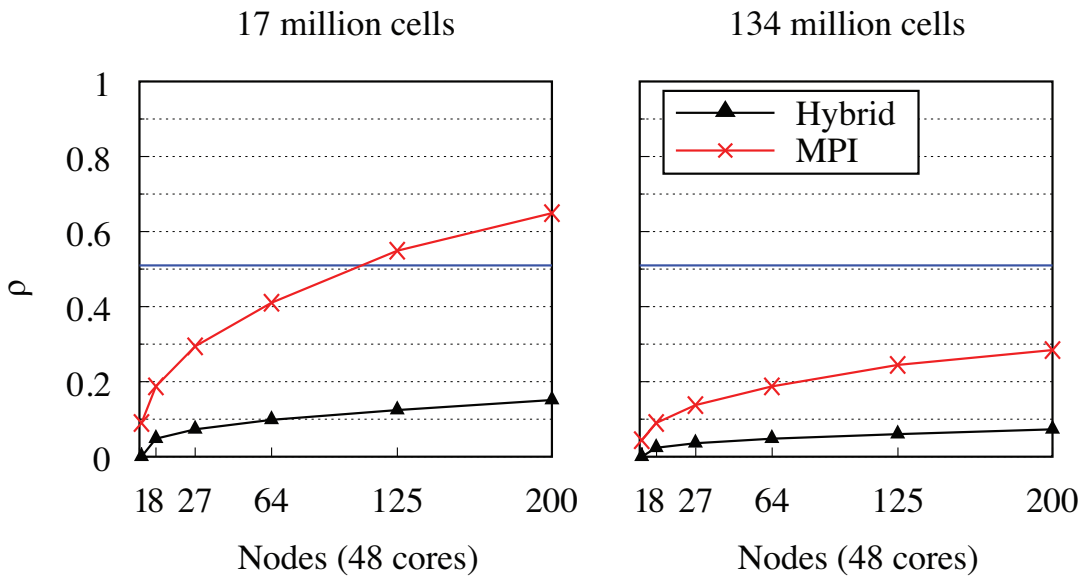
- Sparse matrices of symmetry-preserving discretization of Laplacian operator.
- Discretization on hex-dominant meshes.
- ELLPACK sparse matrix storage format is used.
- Problem is large enough to avoid cache reuse (CPU) and low occupancy (GPU).
- Tests are repeated 1,000 times and results are averaged.



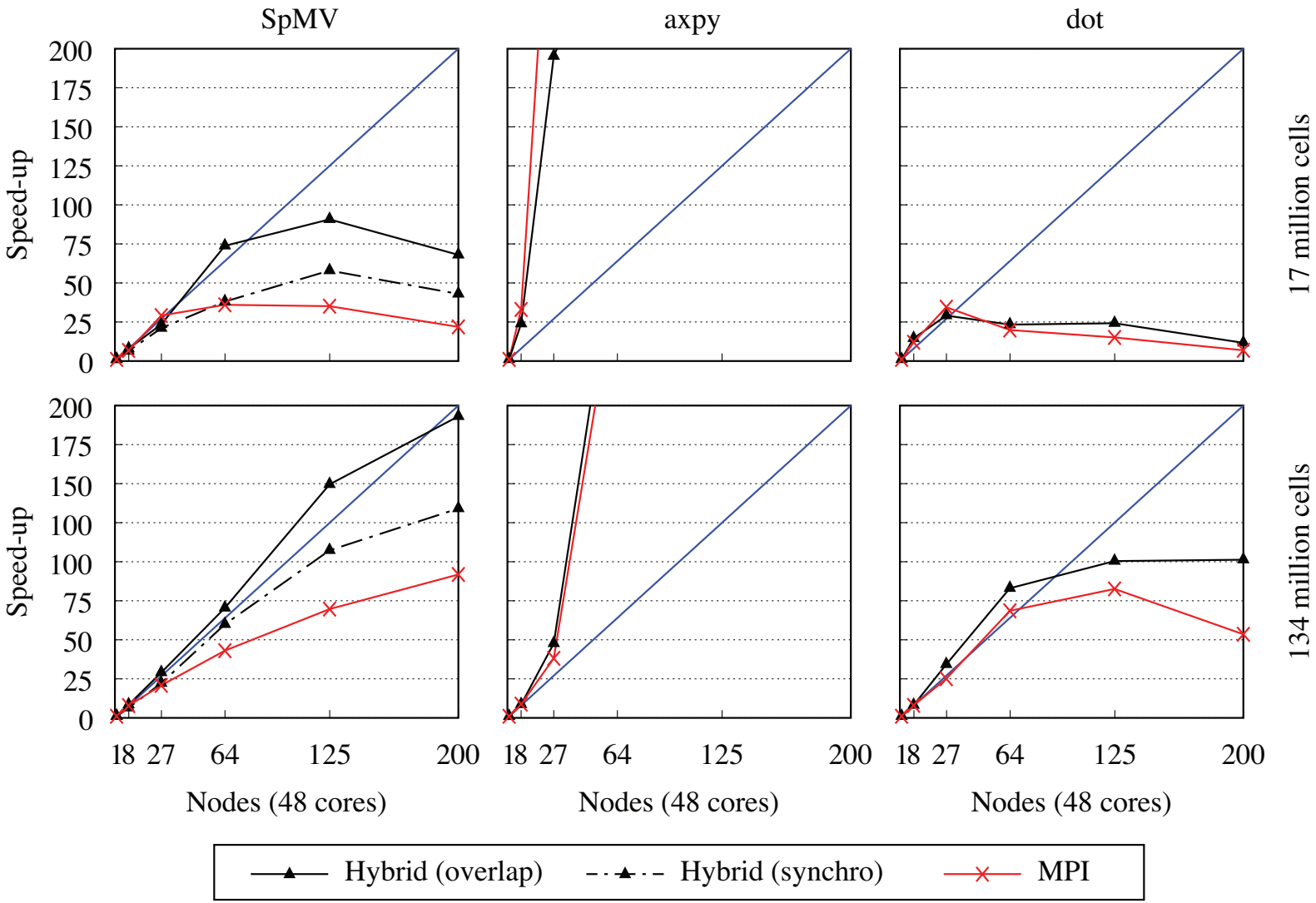
Single-node


SpMV
axpy
dot
Gflop/s
40
35
30
25
20
15
10
5
0
8
16
24
32
40
48
Threads (Xeon 8160)
17 million cells
Static (spread)
Static (close)
Dynamic
MPI

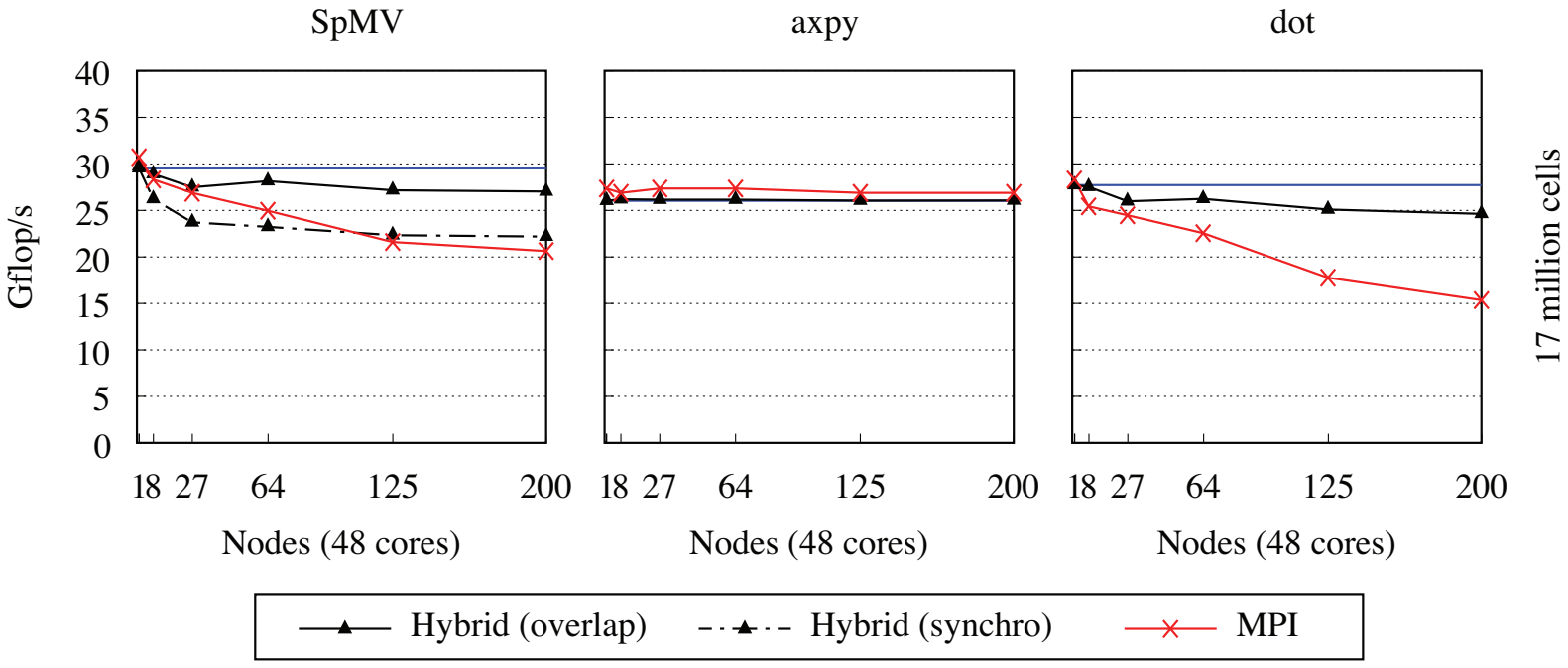
Parallel "roofline"



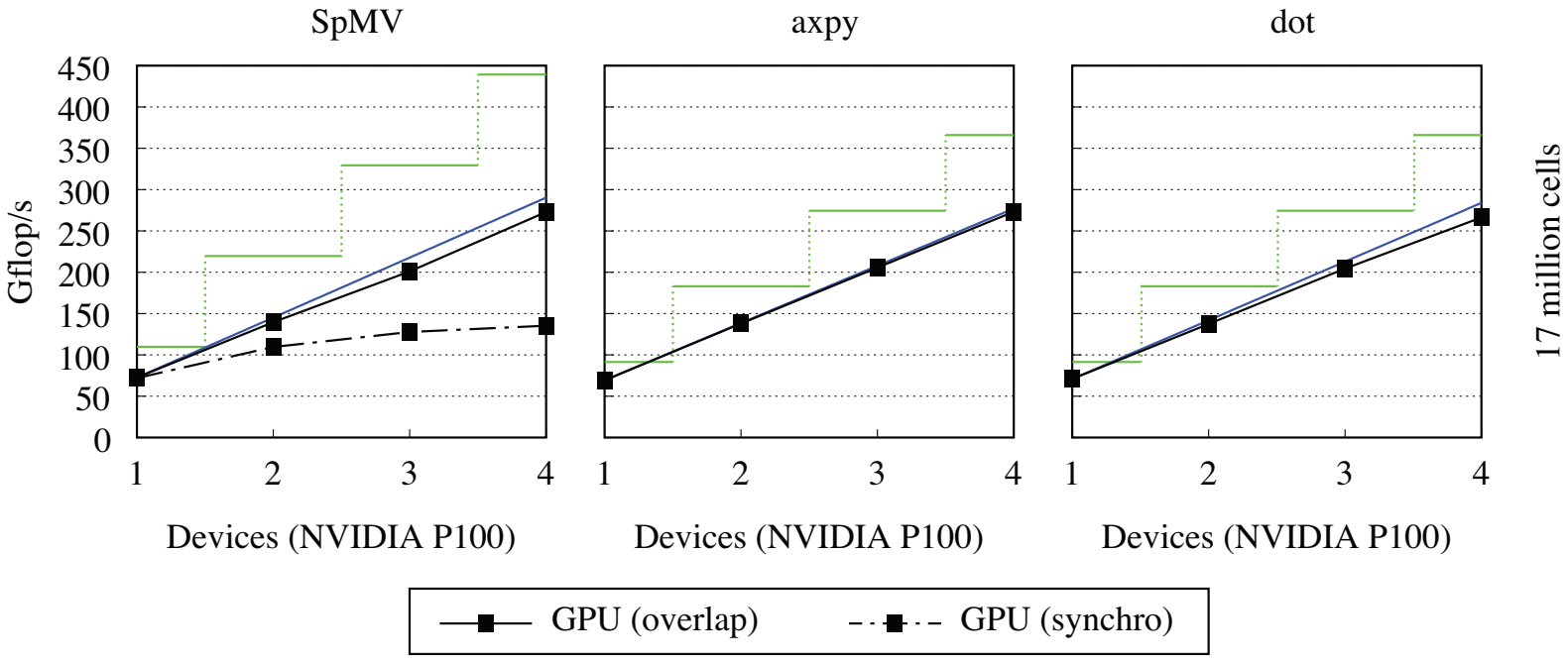
Strong scaling



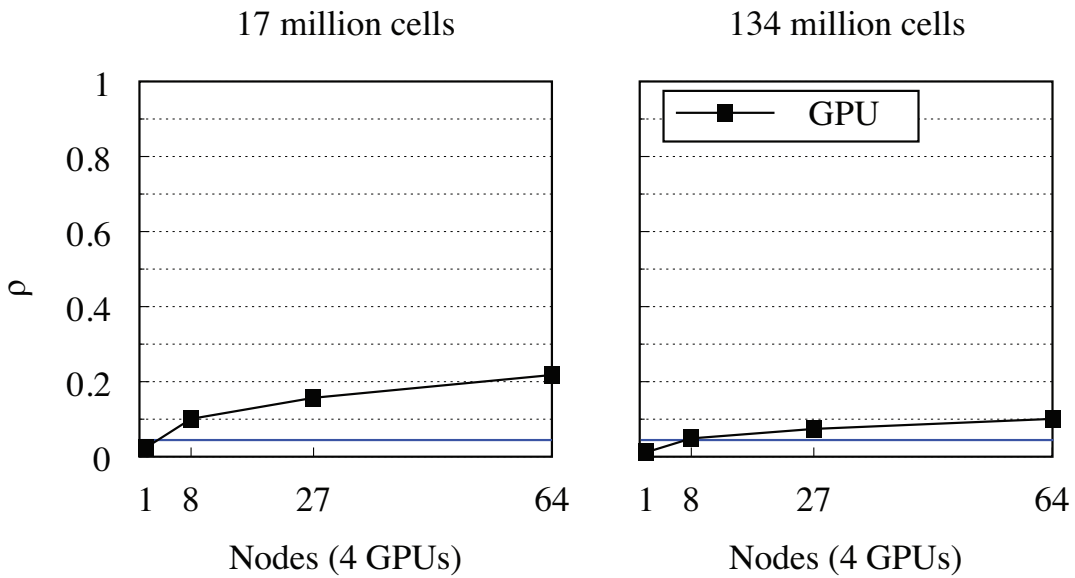
Weak scaling



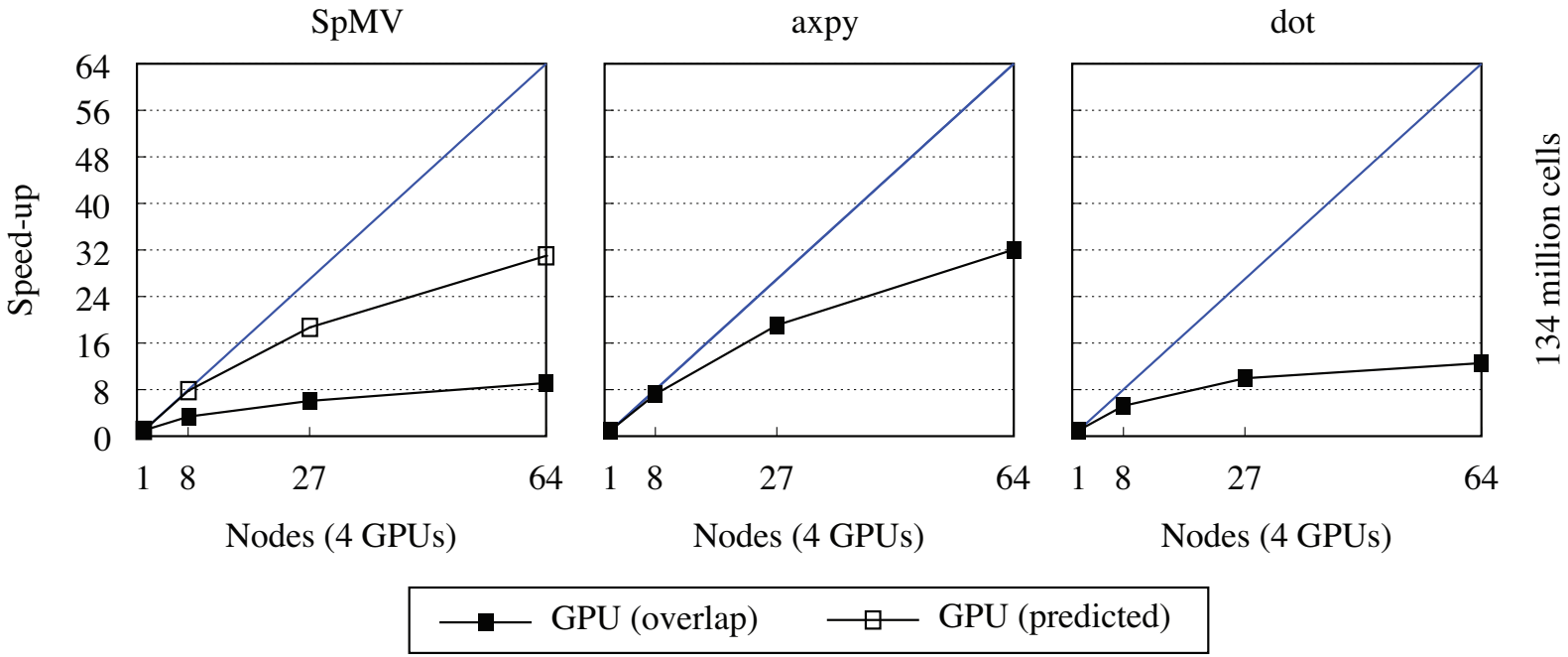
Single-node


SpMV
axpy
dot
Gflop/s
450
400
350
300
250
200
150
100
50
0
1
2
3
4
Devices (NVIDIA P100)
17 million cells
GPU (overlap)
GPU (synchro)

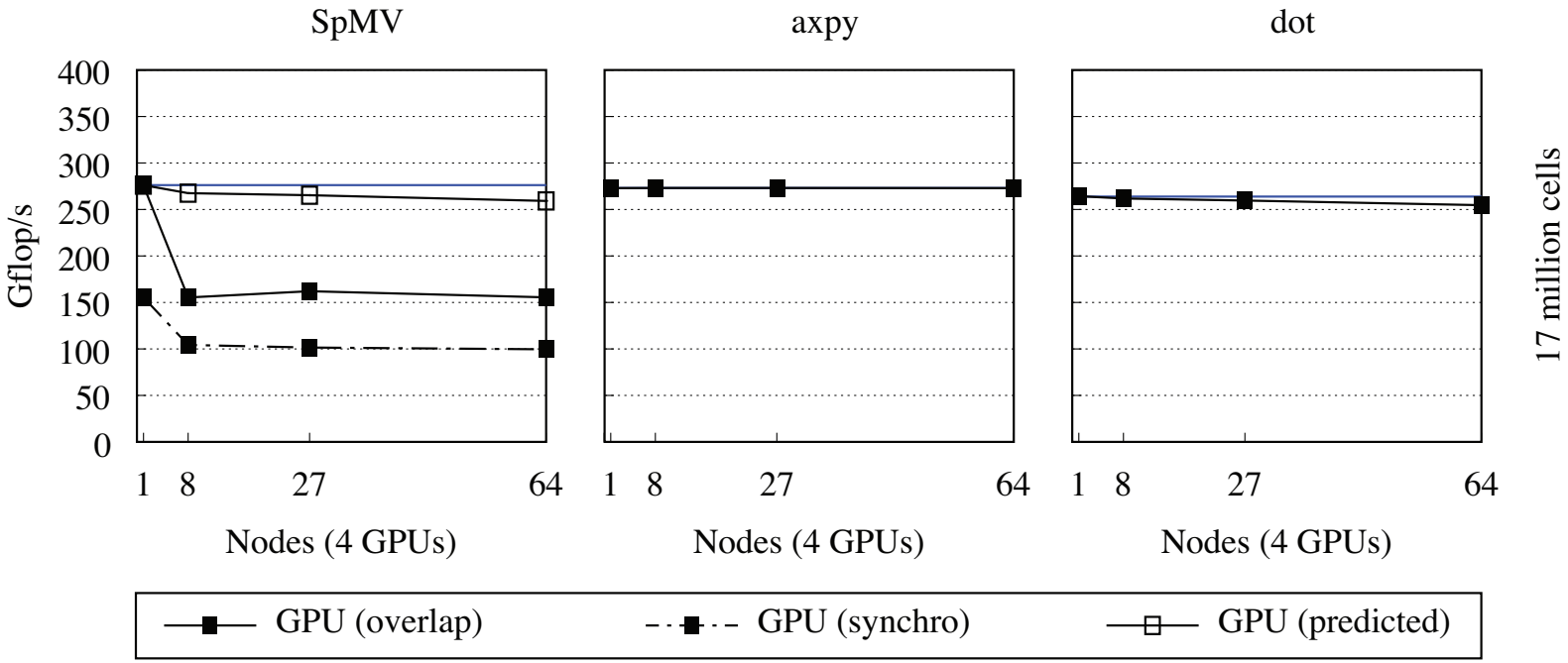
Parallel ”roofline”



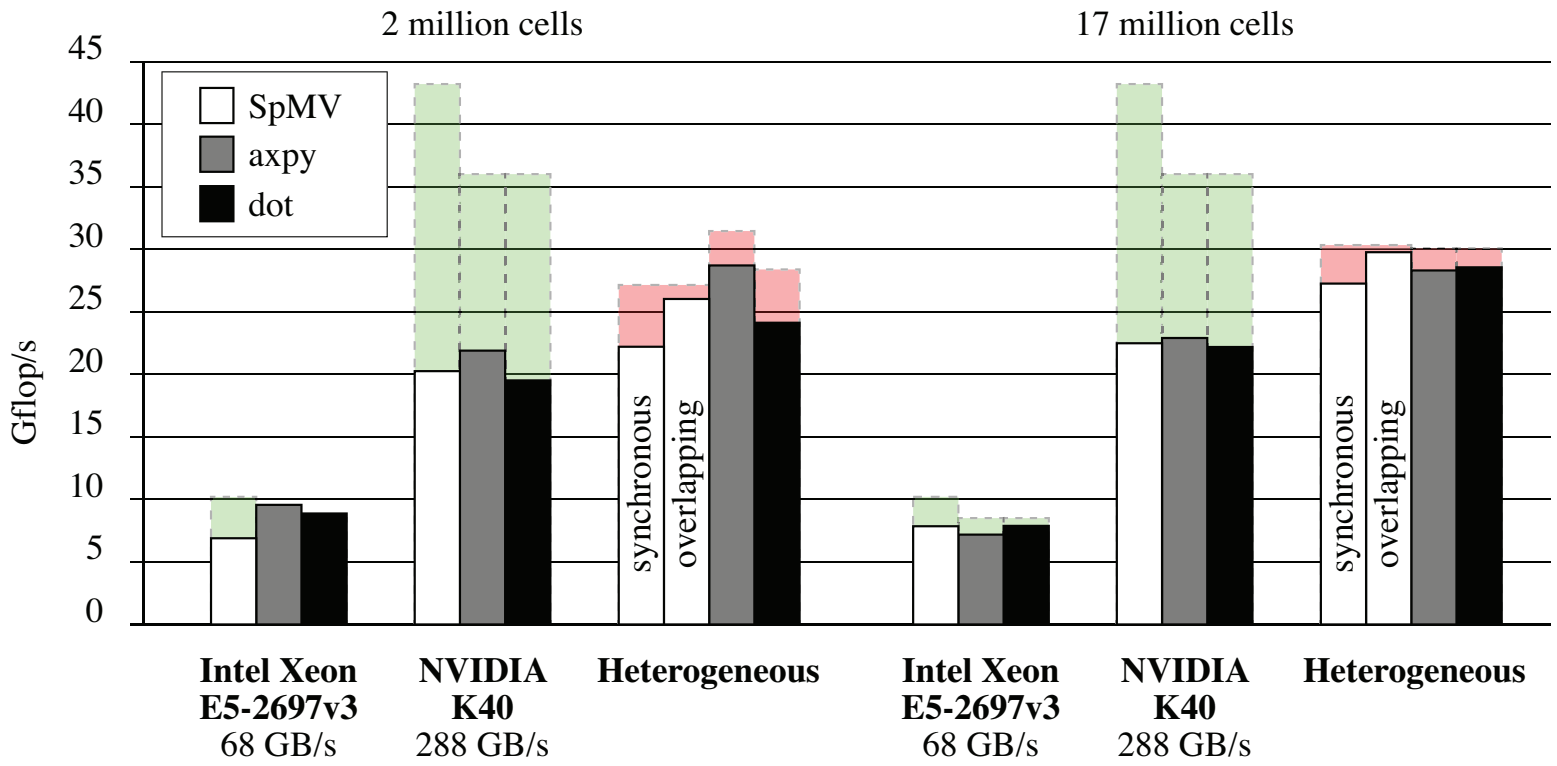
Strong scaling



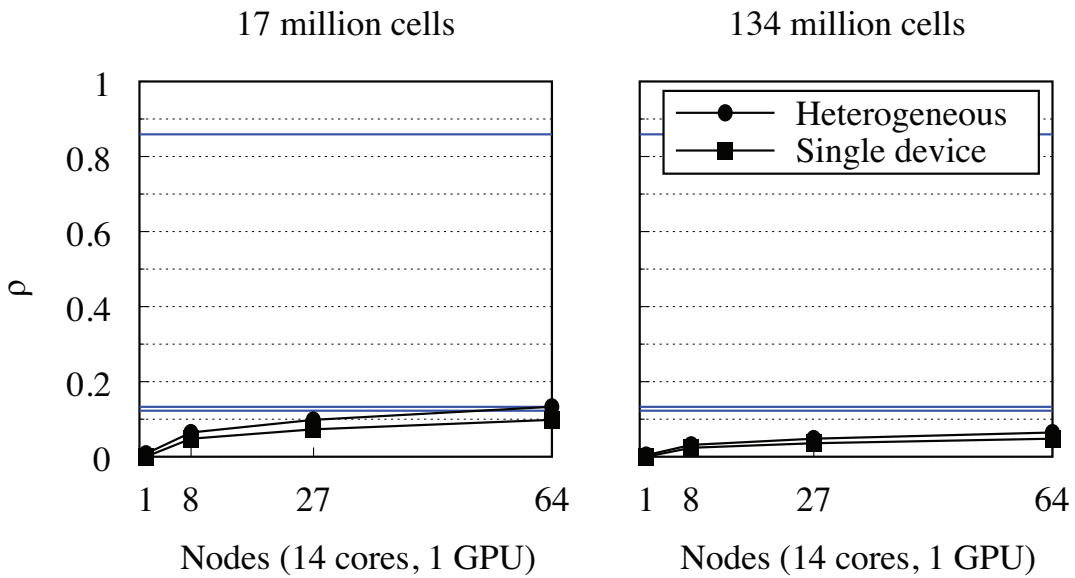
Weak scaling



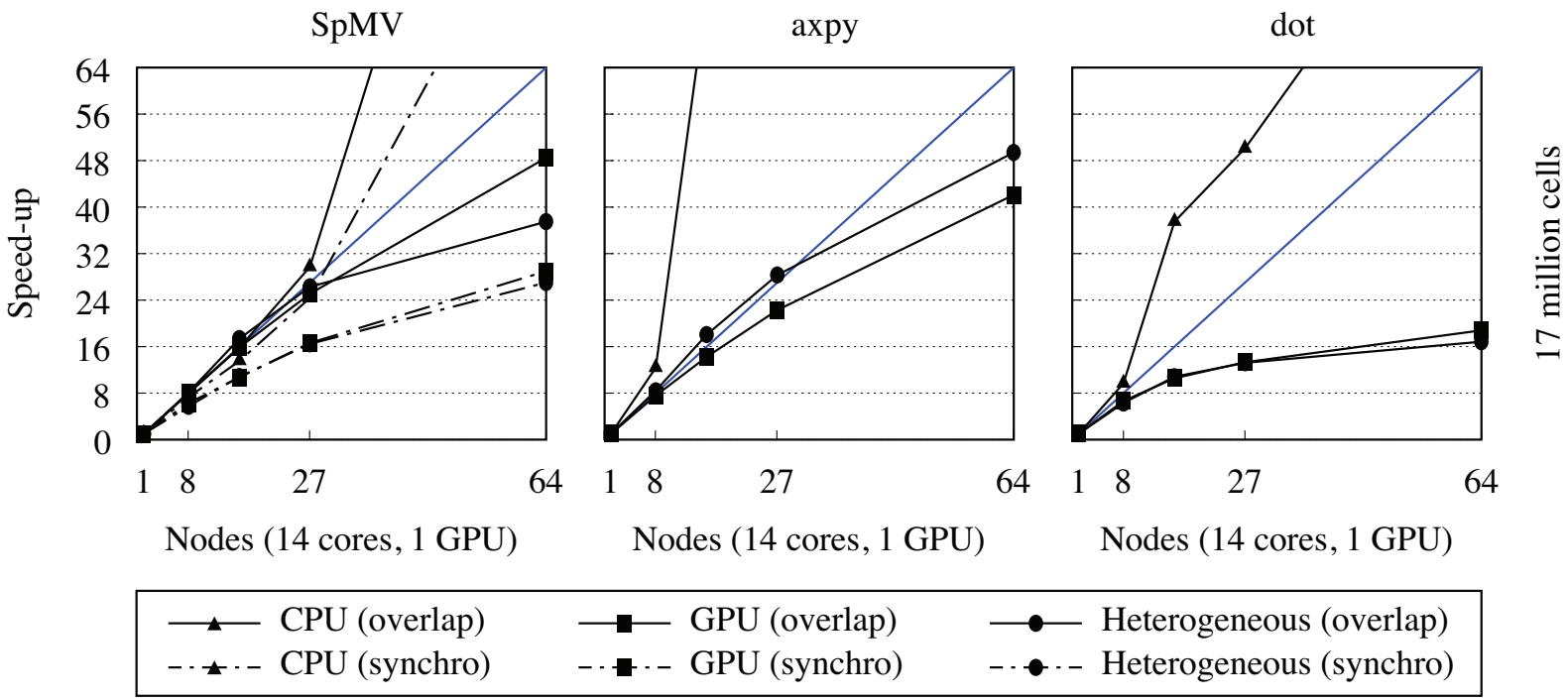
Single-node


2 million cells
17 million cells
45
40
35
30
25
20
15
10
5
0
Gflop/s
SpMV
axpy
dot
synchronous
overlapping
synchronous
overlapping
Intel Xeon E5-2697v3 68 GB/s
NVIDIA K40 288 GB/s
Heterogeneous
Intel Xeon E5-2697v3 68 GB/s
NVIDIA K40 288 GB/s
Heterogeneous

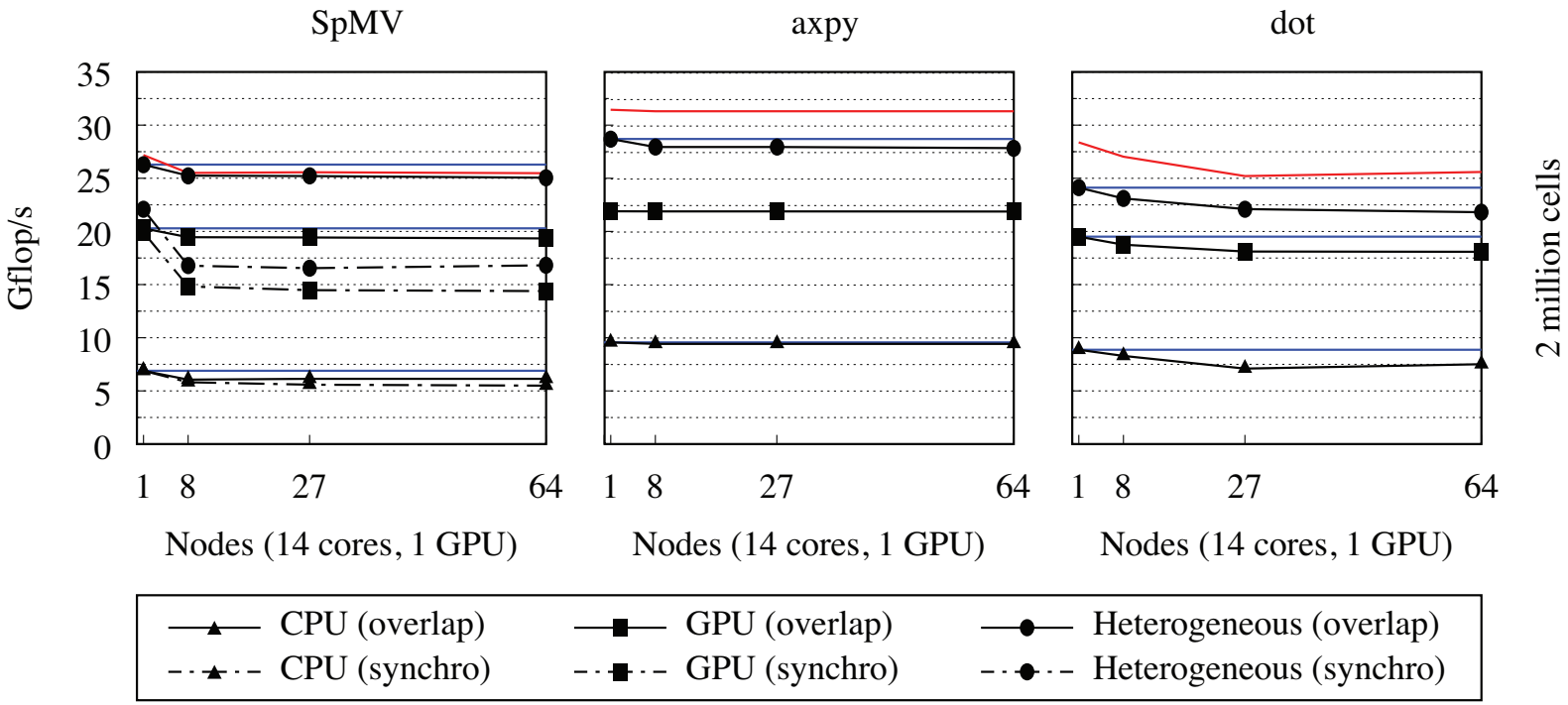
Parallel ”roofline”



Strong scaling

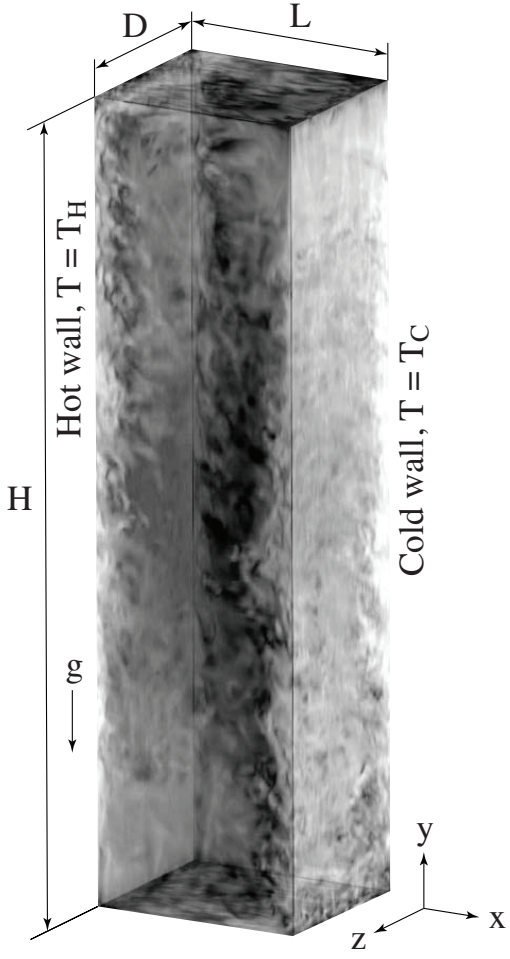


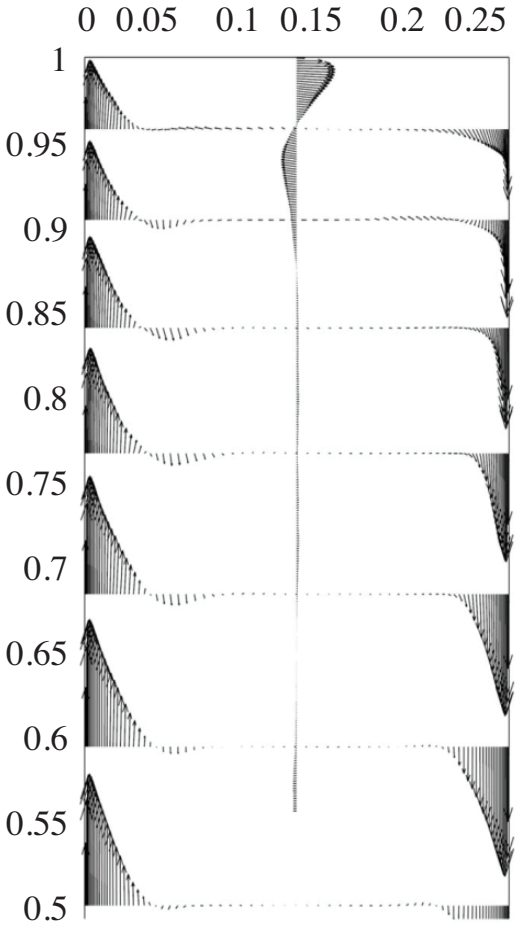
Weak scaling



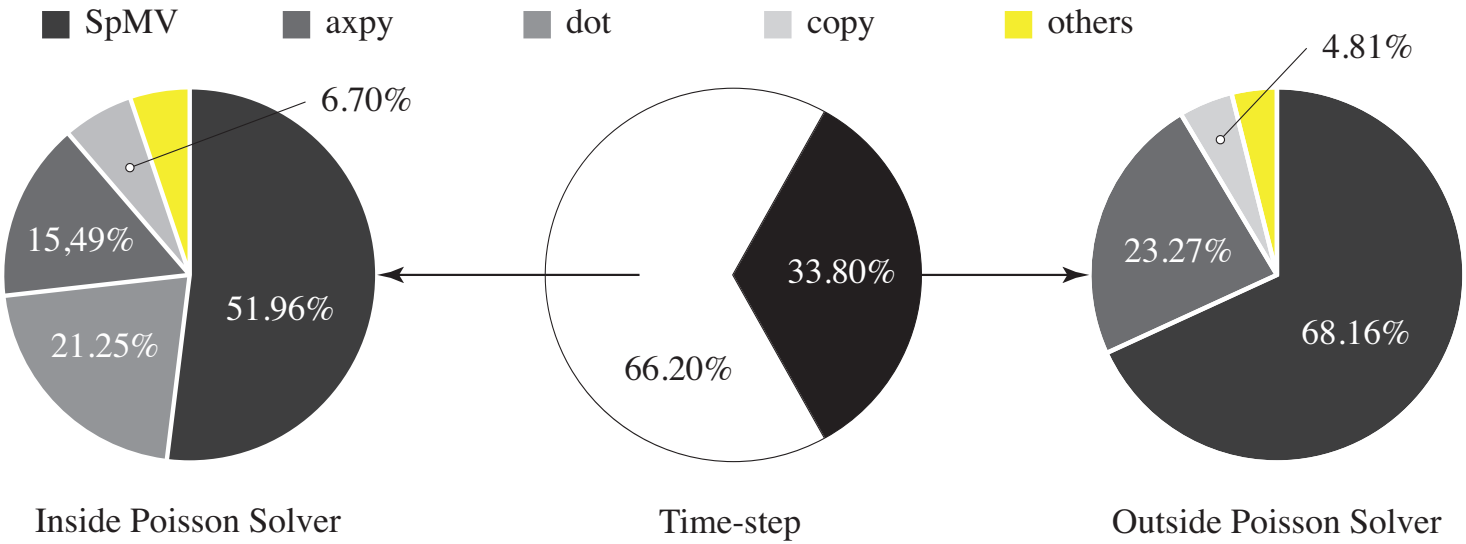
Applications

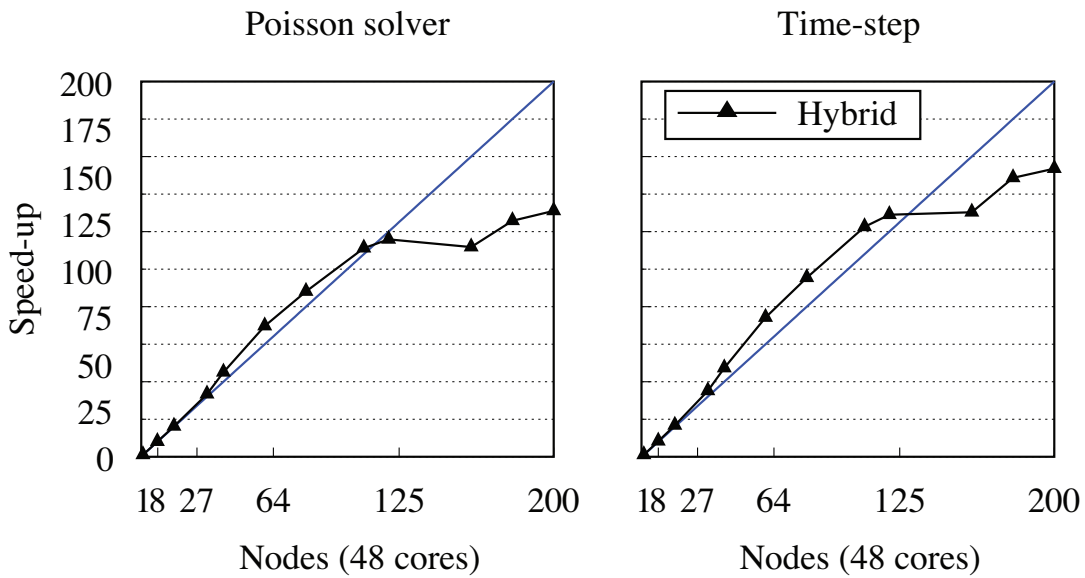
DNS of a Differentially Heated Cavity



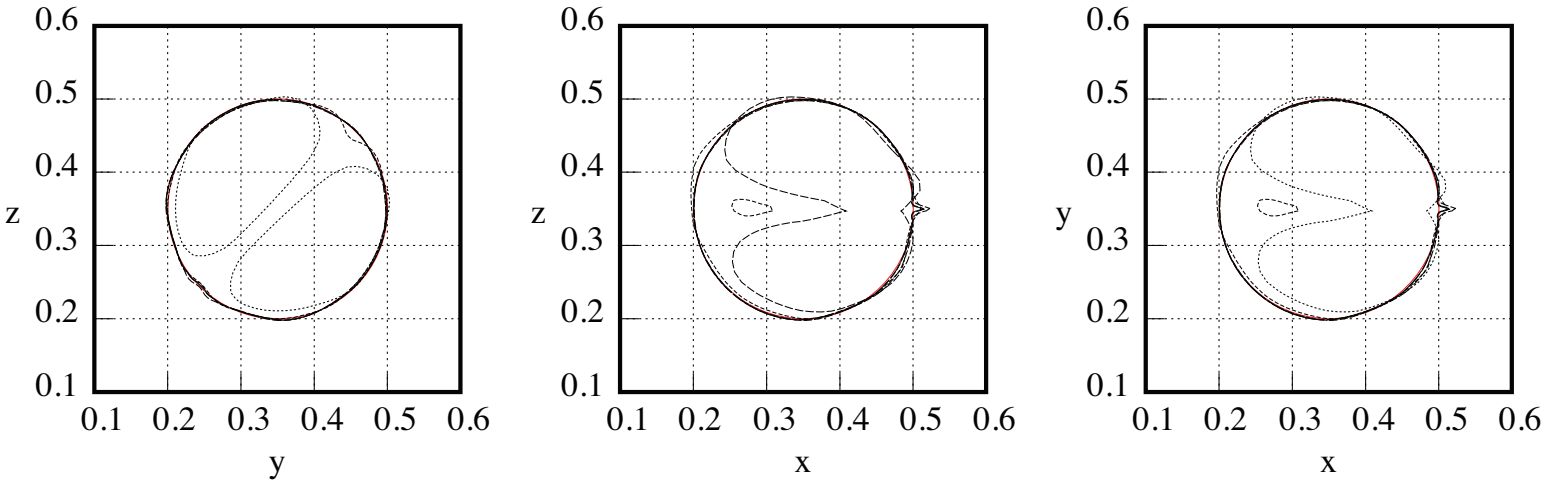


DNS of a Differentially Heated Cavity





Advection of a Level Set function



Conclusions and Future Work

Conclusions

Ongoing and Future Work

Conclusions

- An algebra-based framework has been presented as a naturally portable strategy for implementing numerical simulation codes.

Ongoing and Future Work

Conclusions

- An algebra-based framework has been presented as a naturally portable strategy for implementing numerical simulation codes.
- The hierarchical parallel implementation of our framework has been detailed, and its performance evaluated on various HPC system.

Ongoing and Future Work

Conclusions

- An algebra-based framework has been presented as a naturally portable strategy for implementing numerical simulation codes.
- The hierarchical parallel implementation of our framework has been detailed, and its performance evaluated on various HPC system.
- The SpMM kernel has proved to easily increase arithmetic intensity and performance of algebraic implementations.

Ongoing and Future Work

Conclusions

- An algebra-based framework has been presented as a naturally portable strategy for implementing numerical simulation codes.
- The hierarchical parallel implementation of our framework has been detailed, and its performance evaluated on various HPC system.
- The SpMM kernel has proved to easily increase arithmetic intensity and performance of algebraic implementations.
- The application of HPC^2 to CFD has been demonstrated on incompressible single- and multiphase large-scale simulations.

Ongoing and Future Work

Conclusions

- An algebra-based framework has been presented as a naturally portable strategy for implementing numerical simulation codes.
- The hierarchical parallel implementation of our framework has been detailed, and its performance evaluated on various HPC system.
- The SpMM kernel has proved to easily increase arithmetic intensity and performance of algebraic implementations.
- The application of HPC^2 to CFD has been demonstrated on incompressible single- and multiphase large-scale simulations.

Ongoing and Future Work

- To design a new update mechanism to accelerate the data exchanges, for instance, taking into account NUIOA factor in inter- and intra-node exchanges.

Conclusions

- An algebra-based framework has been presented as a naturally portable strategy for implementing numerical simulation codes.
- The hierarchical parallel implementation of our framework has been detailed, and its performance evaluated on various HPC system.
- The SpMM kernel has proved to easily increase arithmetic intensity and performance of algebraic implementations.
- The application of HPC^2 to CFD has been demonstrated on incompressible single- and multiphase large-scale simulations.

Ongoing and Future Work

- To design a new update mechanism to accelerate the data exchanges, for instance, taking into account NUIOA factor in inter- and intra-node exchanges.
- To fully integrate the SpMM into the framework to exploit multiple parameters simulations, mesh symmetries, parallel-in-time executions, multiple transport equations, among others.

Conclusions

- An algebra-based framework has been presented as a naturally portable strategy for implementing numerical simulation codes.
- The hierarchical parallel implementation of our framework has been detailed, and its performance evaluated on various HPC system.
- The SpMM kernel has proved to easily increase arithmetic intensity and performance of algebraic implementations.
- The application of HPC^2 to CFD has been demonstrated on incompressible single- and multiphase large-scale simulations.

Ongoing and Future Work

- To design a new update mechanism to accelerate the data exchanges, for instance, taking into account NUIOA factor in inter- and intra-node exchanges.
- To fully integrate the SpMM into the framework to exploit multiple parameters simulations, mesh symmetries, parallel-in-time executions, multiple transport equations, among others.
- To fully couple the HPC^2 framework with Termofluids.

Thank you for your attention