Enabling larger and faster simulations from mesh symmetries

Xavier Álvarez-Farré[1], Àdel Alsalti-Baldellou[1,2], Andrey Gorobets[3], Assensi Oliva[1], F. Xavier Trias[1]

In the 2nd High-Fidelity Industrial LES/DNS Symposium, September 22th–24th, 2021, Virtual Venue

[1]Heat and Mass Transfer Technological Center, **Technical University of Catalonia** (BarcelonaTech)
[2]**TermoFluids S.L.**
[3]Keldysh Institute of Applied Mathematics, **Russian Academy of Science**

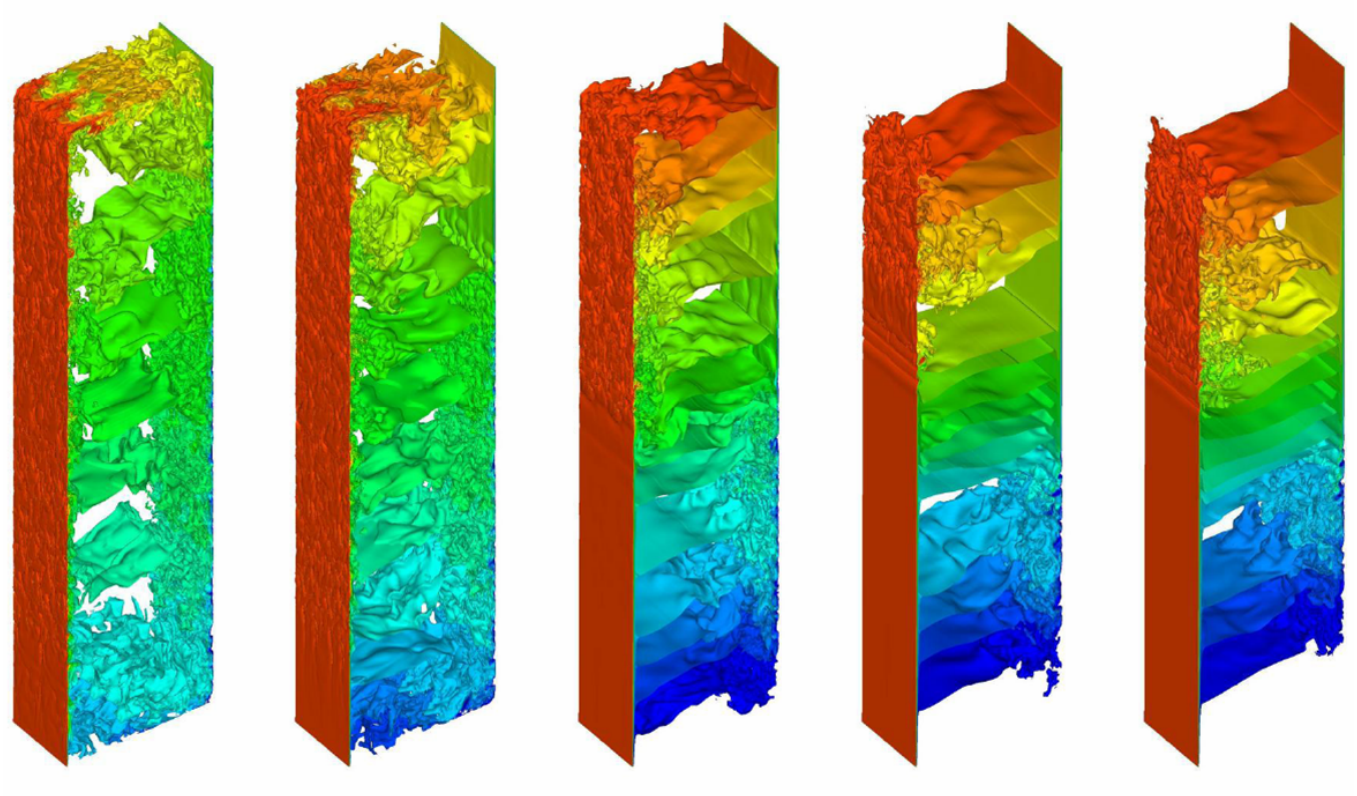
Centre Tecnològic de Transferència de Calor
UNIVERSITAT POLITÈCNICA DE CATALUNYA

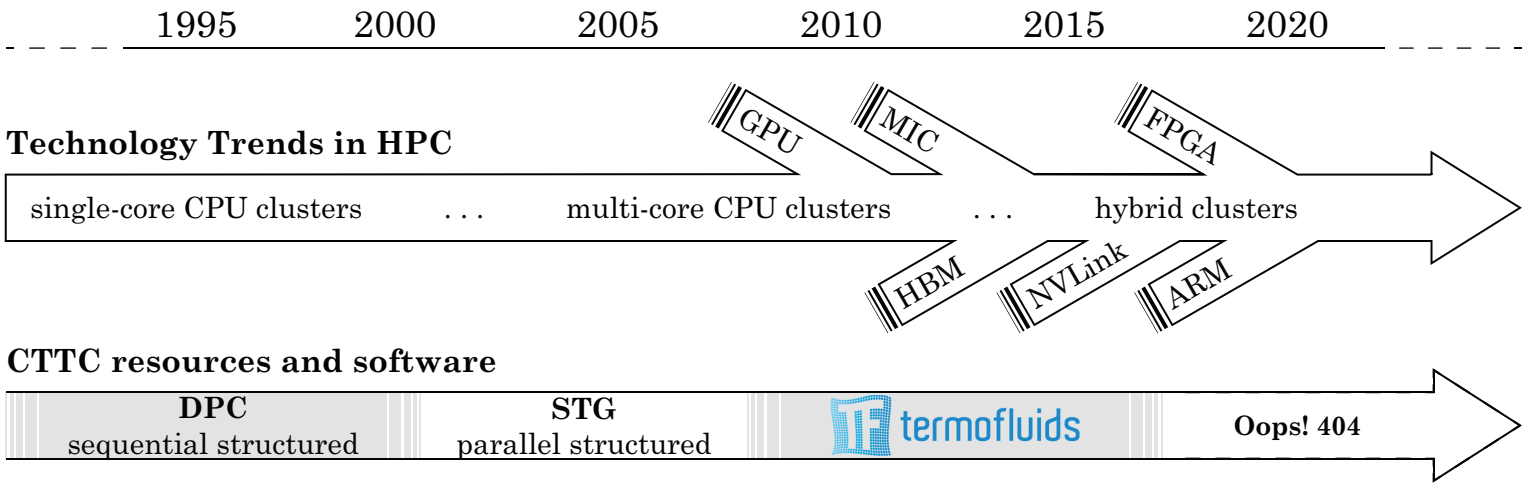
HiFiLeD SYMPOSIUM

Background

Research at CTTC Laboratory

The **Heat and Mass Transfer Technological Center** (CTTC) is a research group of the Technical University of Catalonia highly concerned about the environmental sustainability. Specifically, researchers at the CTTC have been enrolled in both fundamental and applied research, studying several phenomena: natural and forced convection, multi-phase flow, aerodynamics, among many others.



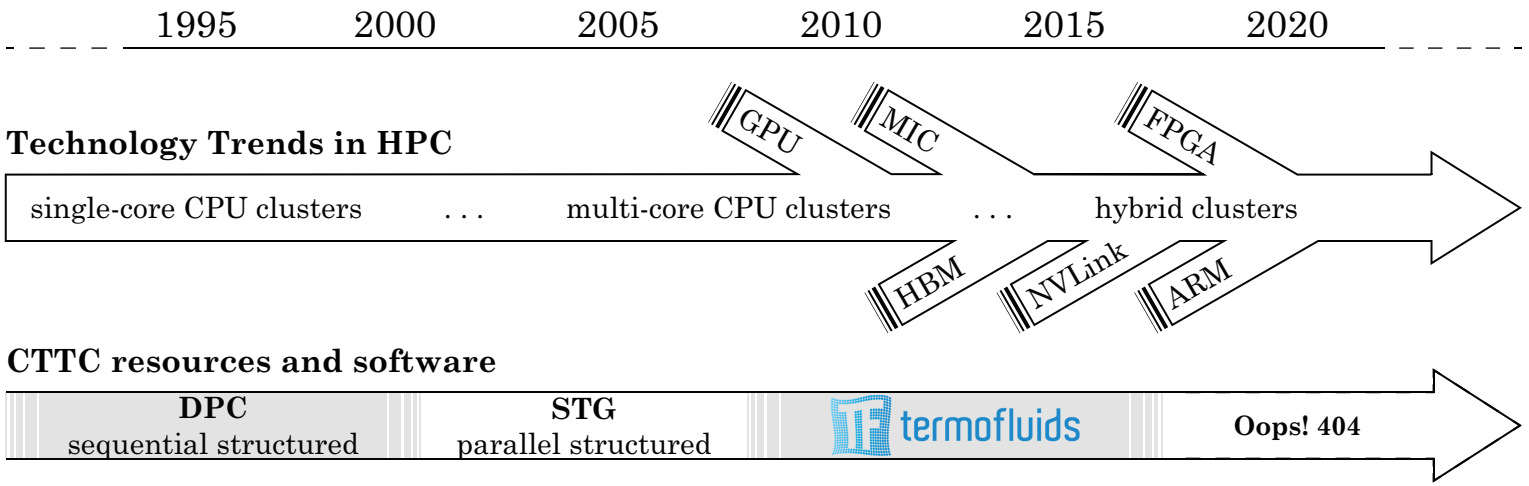


The evolution in hardware technologies

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.

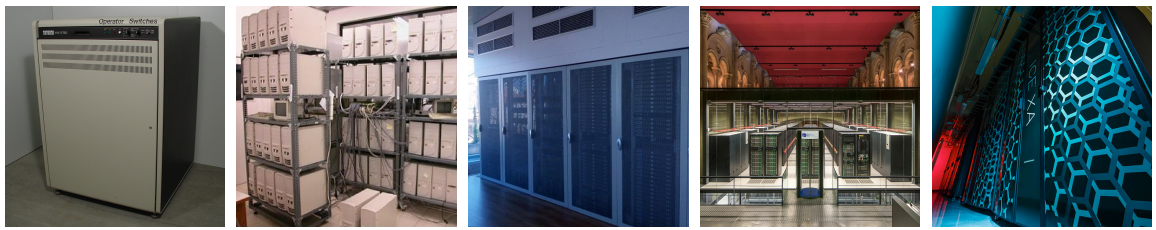


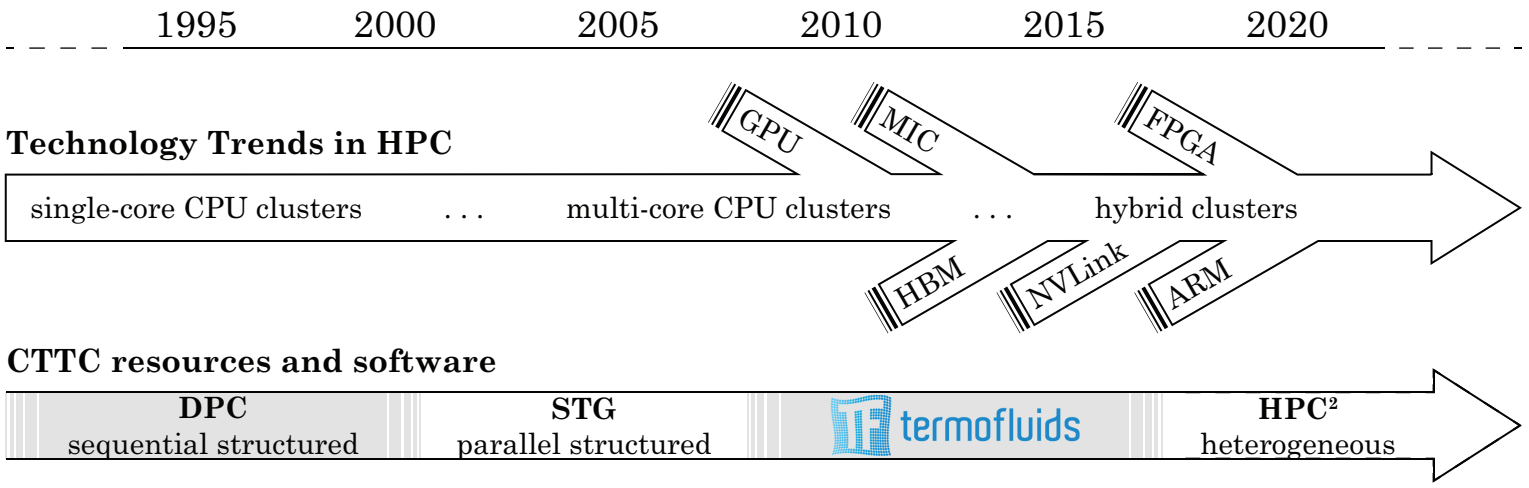




Since the beginning,

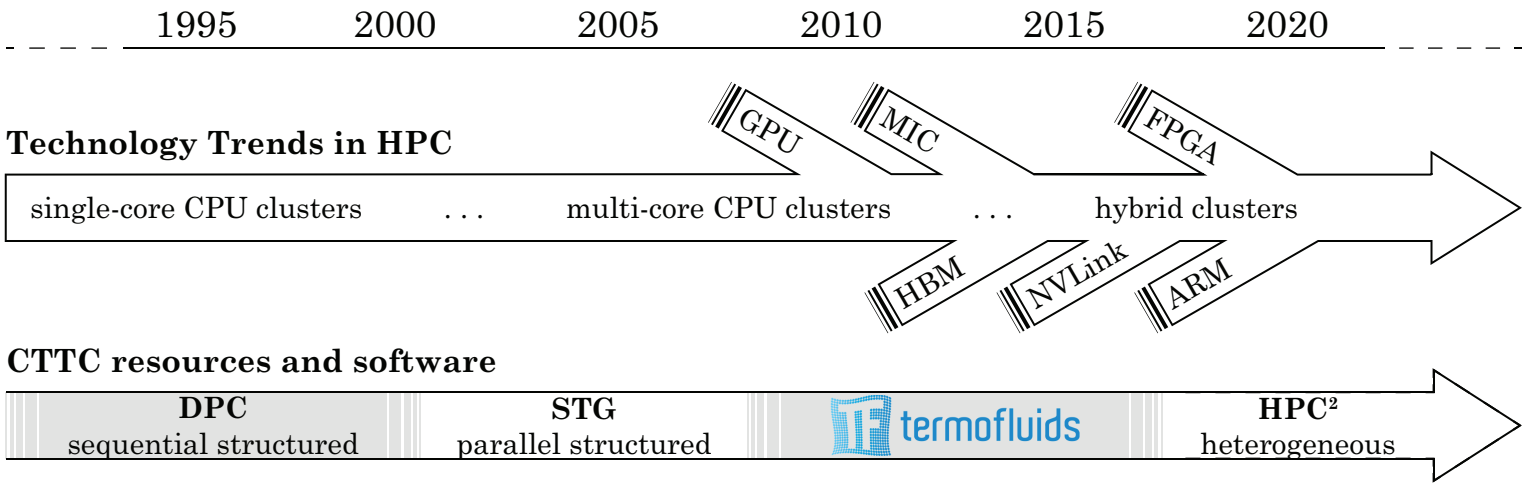
researchers of CTTC is devoted to develop and adapt CFD codes for the state-of-the art computer resources, from sequential structured to parallel unstructured applications.





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.

Is it necessary to use the new hardware architectures?

- In our opinion, yes. New hardware is designed to increase energy efficiency, an imperative to overcome the power constraints in the context of the exascale challenge.

Is it necessary to use the new hardware architectures?

- In our opinion, yes. New hardware is designed to increase energy efficiency, an imperative to overcome the power constraints in the context of the exascale challenge.

Do the traditional implementation models facilitate code portability?

- In our opinion, no. Legacy codes were not designed portable simply because it was not necessary before; these codes usually contain a large number of complex kernels and data structures mostly suitable for CPU architectures.

The future of scientific computing codes

Is it necessary to use the new hardware architectures?

- In our opinion, yes. New hardware is designed to increase energy efficiency, an imperative to overcome the power constraints in the context of the exascale challenge.

Do the traditional implementation models facilitate code portability?

- In our opinion, no. Legacy codes were not designed portable simply because it was not necessary before; these codes usually contain a large number of complex kernels and data structures mostly suitable for CPU architectures.

Do we need to change the way we look at scientific computing in general?

- In our opinion, yes. There is a large variety of hardware architectures and it is difficult to determine which are going to prevail. Therefore, sustainability and portability should become the center of scientific computing software design.

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:

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

In a matrix–vector notation, the finite-volume discretization of the NS equations on an arbitrary collocated mesh can be written[1] by:

$$\mathbf{M}\mathbf{u}_s = \mathbf{0}_c, \qquad \mathbf{\Omega} d_t \mathbf{u}_c + \mathbf{M}\mathbf{U}_s \mathbf{u}_c + \mathbf{D}\mathbf{u}_c - \mathbf{M}^\mathsf{T} p_c = \mathbf{0}_c.$$

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

[2] Álvarez-Farré et al., HPC^2–A fully-portable, algebra-based framework for heterogeneous computing. Application to CFD, Computers and Fluids, 173, 285–292, 2018.

Continuous, dimensionless Navier–Stokes equations read:

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

In a matrix–vector notation, the finite-volume discretization of the NS equations on an arbitrary collocated mesh can be written[1] by:

$$\mathbf{M}\mathbf{u}_s = \mathbf{0}_c, \qquad \mathbf{\Omega} d_t \mathbf{u}_c + \mathbf{M}\mathbf{U}_s\mathbf{u}_c + \mathbf{D}\mathbf{u}_c - \mathbf{M}^{\mathsf{T}} p_c = \mathbf{0}_c.$$

- The discrete variables are stored in vectors and the discrete operators in sparse matrices.

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

[2] Álvarez-Farré et al., HPC^2–A fully-portable, algebra-based framework for heterogeneous computing. Application to CFD, Computers and Fluids, 173, 285–292, 2018.

Continuous, dimensionless Navier–Stokes equations read:

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

In a matrix–vector notation, the finite-volume discretization of the NS equations on an arbitrary collocated mesh can be written[1] by:

$$\mathbf{M}\mathbf{u}_s = \mathbf{0}_c, \qquad \mathbf{\Omega} d_t \mathbf{u}_c + \mathbf{M}\mathbf{U}_s \mathbf{u}_c + \mathbf{D}\mathbf{u}_c - \mathbf{M}^\mathsf{T} p_c = \mathbf{0}_c.$$

- The discrete variables are stored in vectors and the discrete operators in sparse matrices.
- The numerical method results fully integrated into the data structures somehow so that generic algebraic kernels can be used[2].

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

[2] Álvarez-Farré et al., HPC^2–A fully-portable, algebra-based framework for heterogeneous computing. Application to CFD, Computers and Fluids, 173, 285–292, 2018.

Continuous, dimensionless Navier–Stokes equations read:

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

In a matrix–vector notation, the finite-volume discretization of the NS equations on an arbitrary collocated mesh can be written[1] by:

$$\mathbf{M}\mathbf{u}_s = \mathbf{0}_c, \qquad \mathbf{\Omega} d_t \mathbf{u}_c + \mathbf{M}\mathbf{U}_s\mathbf{u}_c + \mathbf{D}\mathbf{u}_c - \mathbf{M}^\mathsf{T} p_c = \mathbf{0}_c.$$

- The discrete variables are stored in vectors and the discrete operators in sparse matrices.
- The numerical method results fully integrated into the data structures somehow so that generic algebraic kernels can be used[2].
- The discrete operators can be built directly from the inherent incidence matrices that define the mesh mimicking the properties of the continuum operators.

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

[2] Álvarez-Farré et al., HPC^2–A fully-portable, algebra-based framework for heterogeneous computing. Application to CFD, Computers and Fluids, 173, 285–292, 2018.

An example of an algebra-based formulation of the algorithm for the time-integration phase relies on a reduced set of **only three linear algebra kernels**: the sparse matrix-vector product (SpMV), the linear combination of vectors (axpy) and the dot product (dot).

Algorithm 1 Time-integration step.

1: $\mathsf{R}\left(\boldsymbol{u}_s^n, \boldsymbol{u}_c^n, \boldsymbol{\theta}_c^n\right) \leftarrow -\mathsf{C}_c^{3d}\left(\boldsymbol{u}_s^n\right)\boldsymbol{u}_c^n - \mathsf{D}_c^{3d}\boldsymbol{u}_c^n + \boldsymbol{f}_c(\boldsymbol{\theta}_c^n)$ ▷ momentum's right-hand side

2: $\boldsymbol{u}_c^p = \boldsymbol{u}_c^n + \Delta t\left\{\frac{3}{2}\mathsf{R}\left(\boldsymbol{u}_s^n, \boldsymbol{u}_c^n\right) - \frac{1}{2}\mathsf{R}\left(\boldsymbol{u}_s^{n-1}, \boldsymbol{u}_c^{n-1}\right)\right\}$ ▷ predictor velocity

3: $\mathsf{L}\tilde{\boldsymbol{p}}_c^{n+1} = \mathsf{M}\boldsymbol{u}_s^p$, where $\boldsymbol{u}_s^p = \Gamma_{c\to s}\boldsymbol{u}_c^p$ ▷ solve Poisson equation

4: $\boldsymbol{u}_s^{n+1} = \boldsymbol{u}_s^p - \mathsf{G}\tilde{\boldsymbol{p}}_c^{n+1}$, where $\mathsf{G} = -\Omega_s^{-1}\mathsf{M}^T$ ▷ correct the staggered velocity field

5: $\boldsymbol{u}_c^{n+1} = \boldsymbol{u}_c^p - \mathsf{G}_c\tilde{\boldsymbol{p}}_c^{n+1}$, where $\mathsf{G}_c = -\Gamma_{s\to c}\Omega_s^{-1}\mathsf{M}^T$ ▷ correct the collocated velocity field

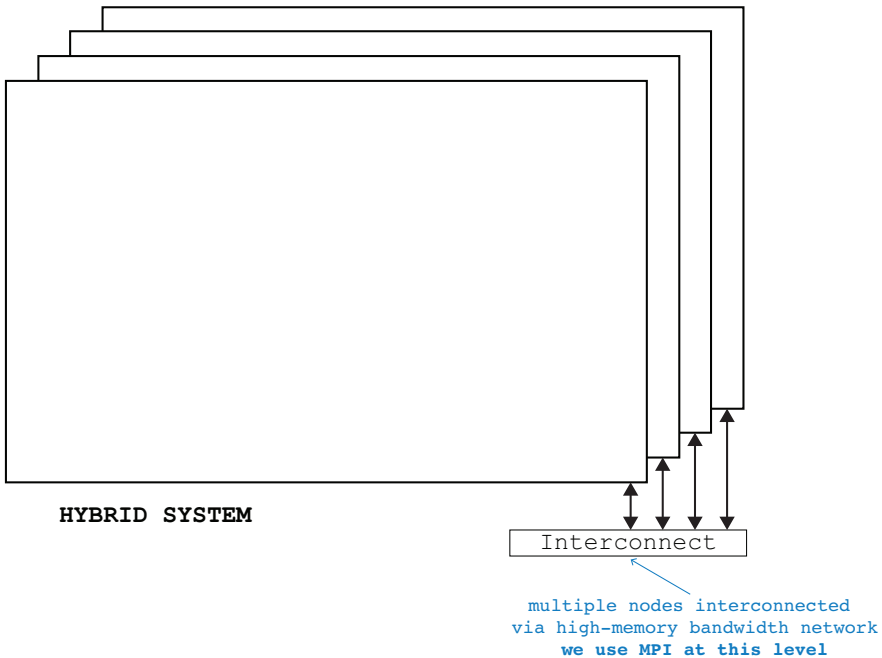
6: $\mathsf{R}_\theta\left(\boldsymbol{u}_s^n, \boldsymbol{\theta}_c^n\right) \equiv -\mathsf{C}_c\left(\boldsymbol{u}_s^n\right)\boldsymbol{\theta}_c^n - Pr^{-1}\mathsf{D}_c\boldsymbol{\theta}_c^n$ ▷ energy's right-hand side

7: $\boldsymbol{\theta}_c^{n+1} = \boldsymbol{\theta}_c^n + \Delta t\left\{\frac{3}{2}\mathsf{R}_\theta\left(\boldsymbol{u}_s^n, \boldsymbol{\theta}_c^n\right) - \frac{1}{2}\mathsf{R}_\theta\left(\boldsymbol{u}_s^{n-1}, \boldsymbol{\theta}_c^{n-1}\right)\right\}$ ▷ integrate energy

A hierarchical parallel implementation

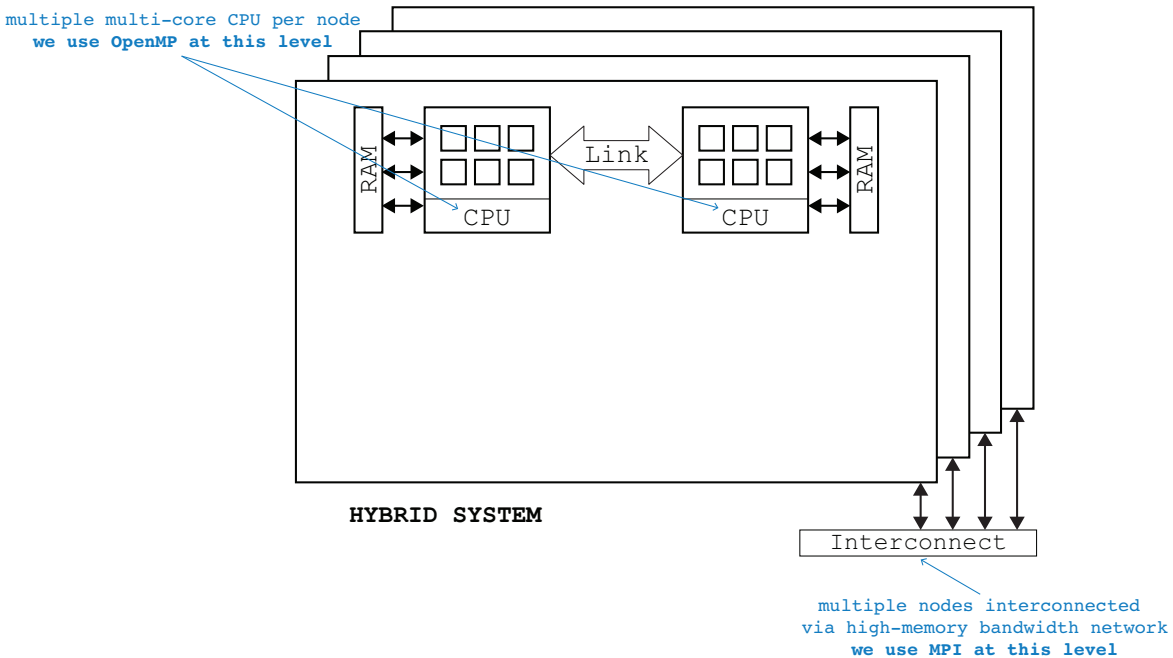
Hybrid HPC systems

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.



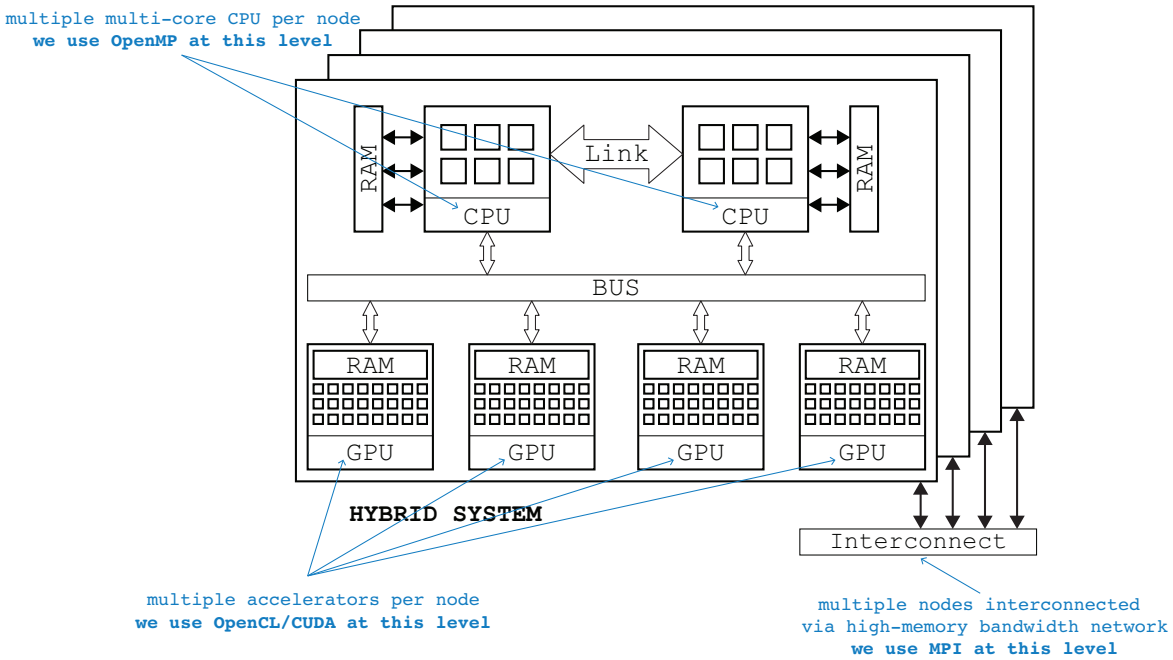
Hybrid HPC systems

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.



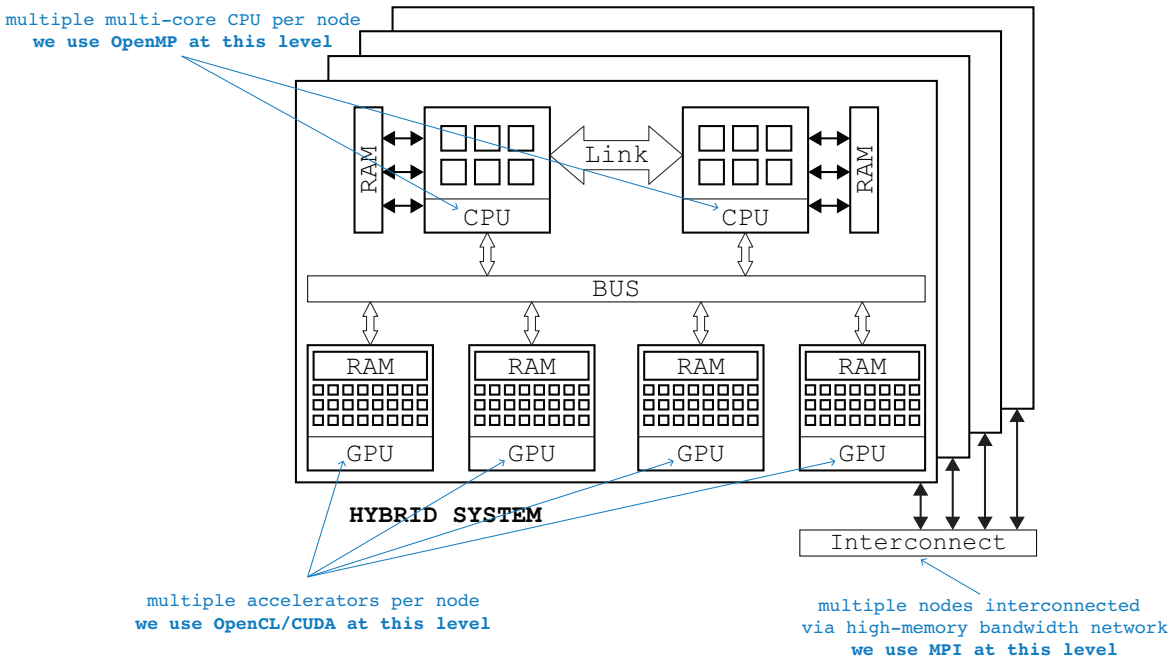
Hybrid HPC systems

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

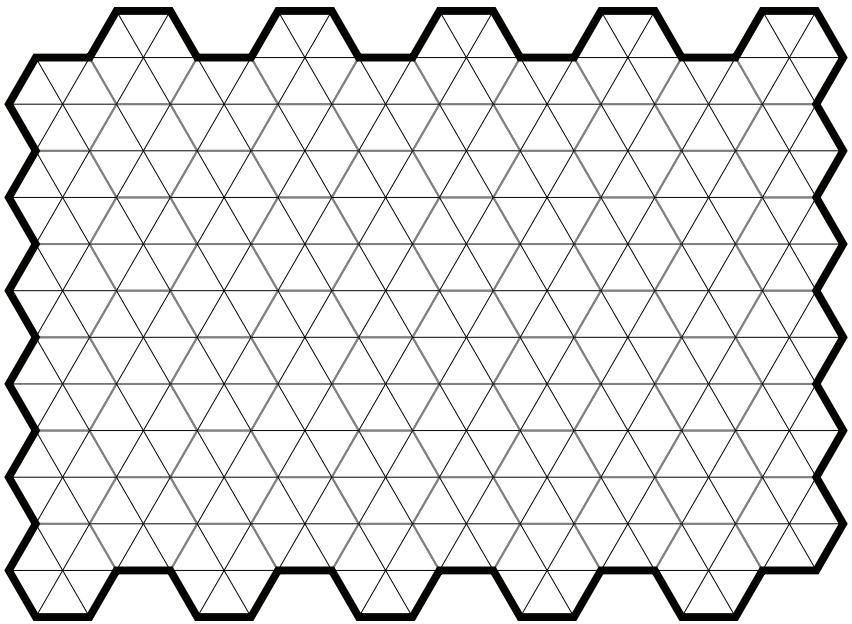
Modern HPC systems consist of multiple hybrid computing nodes interconnected via a communication infrastructure. The nodes are composed of many hardware devices of different architectures, such as central processing unit (CPU) or graphics processing unit (GPU), among others.



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).

Multilevel domain decomposition

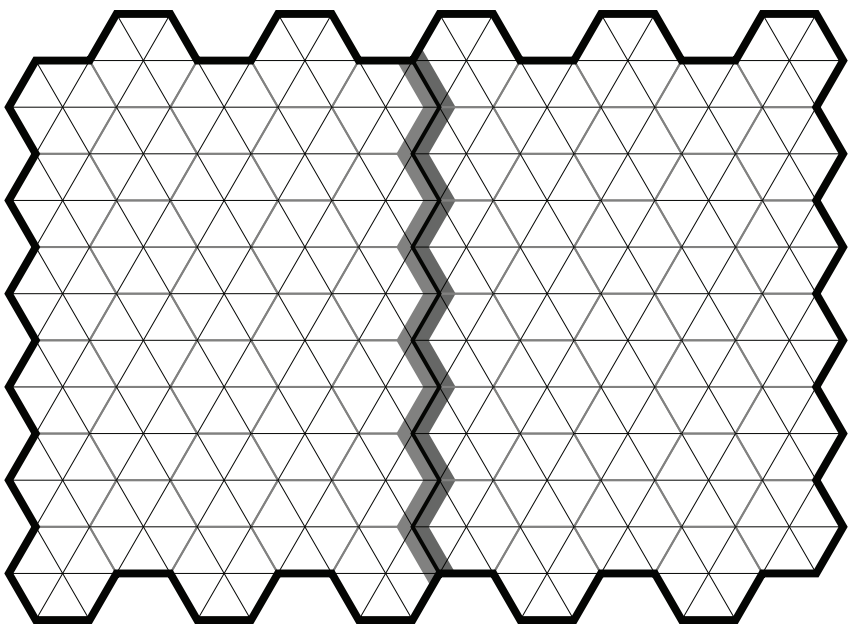
Multilevel workload distribution consists of dividing the computational domain (mesh) into subsets recursively to distribute it among the hardware of a computing system.



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.

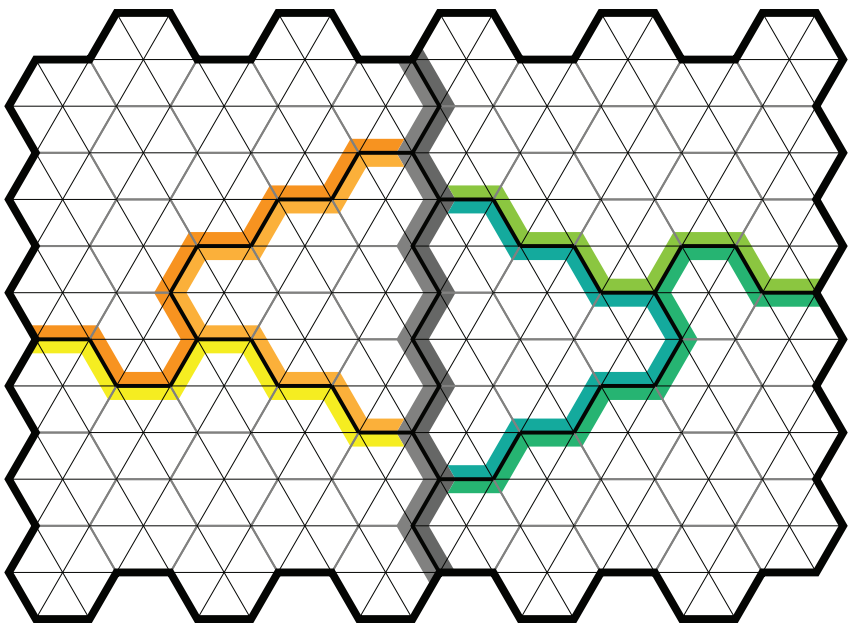
- First-level decomposition divides the workload among the computing nodes, that is, the MPI processes.



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.

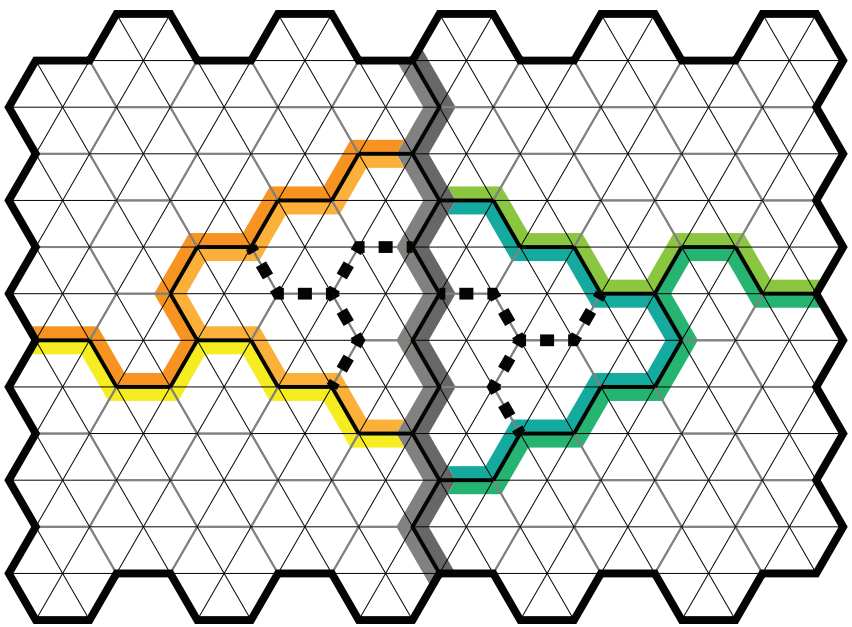
- First-level decomposition divides the workload among the computing nodes, that is, the MPI processes.
- Second-level decomposition divides the first-level partitions to share each MPI's workload among its available hardware, that is, the host and co-processors.



Multilevel domain decomposition

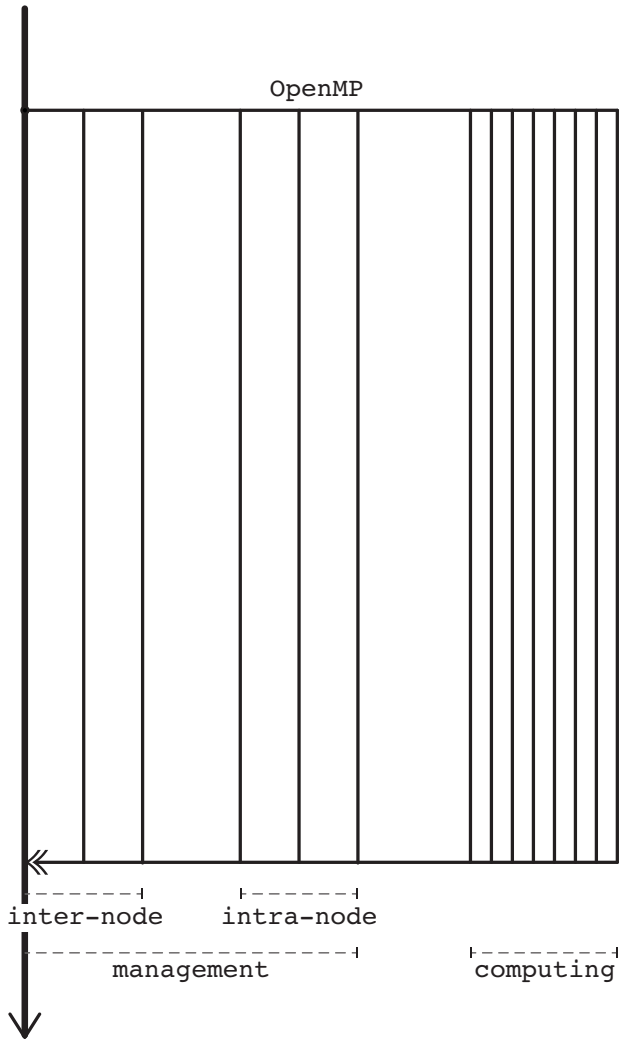
Multilevel workload distribution consists of dividing the computational domain (mesh) into subsets recursively to distribute it among the hardware of a computing system.

- First-level decomposition divides the workload among the computing nodes, that is, the MPI processes.
- Second-level decomposition divides the first-level partitions to share each MPI's workload among its available hardware, that is, the host and co-processors.
- Third-level decomposition divides the second-level partitions to distribute the workload of a device whose shared-memory space introduces a significant NUMA factor.



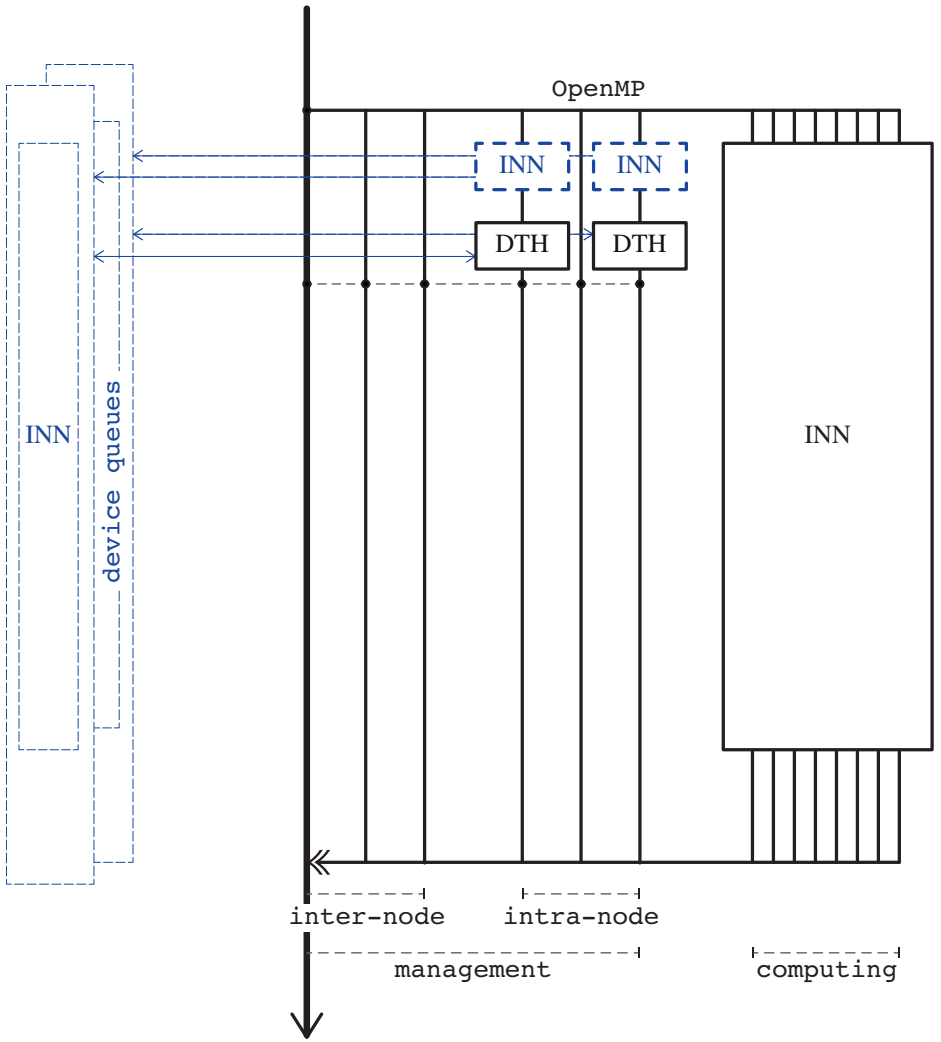
Flat multithreaded execution strategies

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



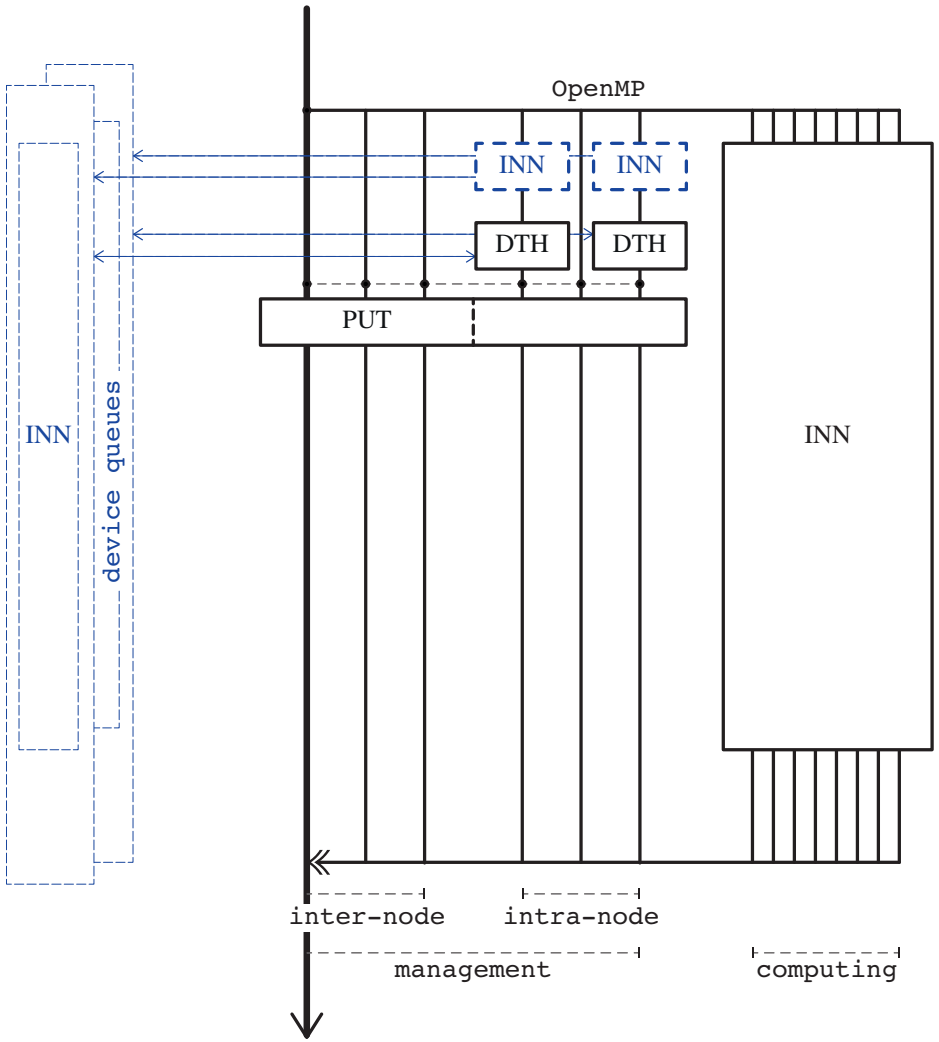
Flat multithreaded execution strategies

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



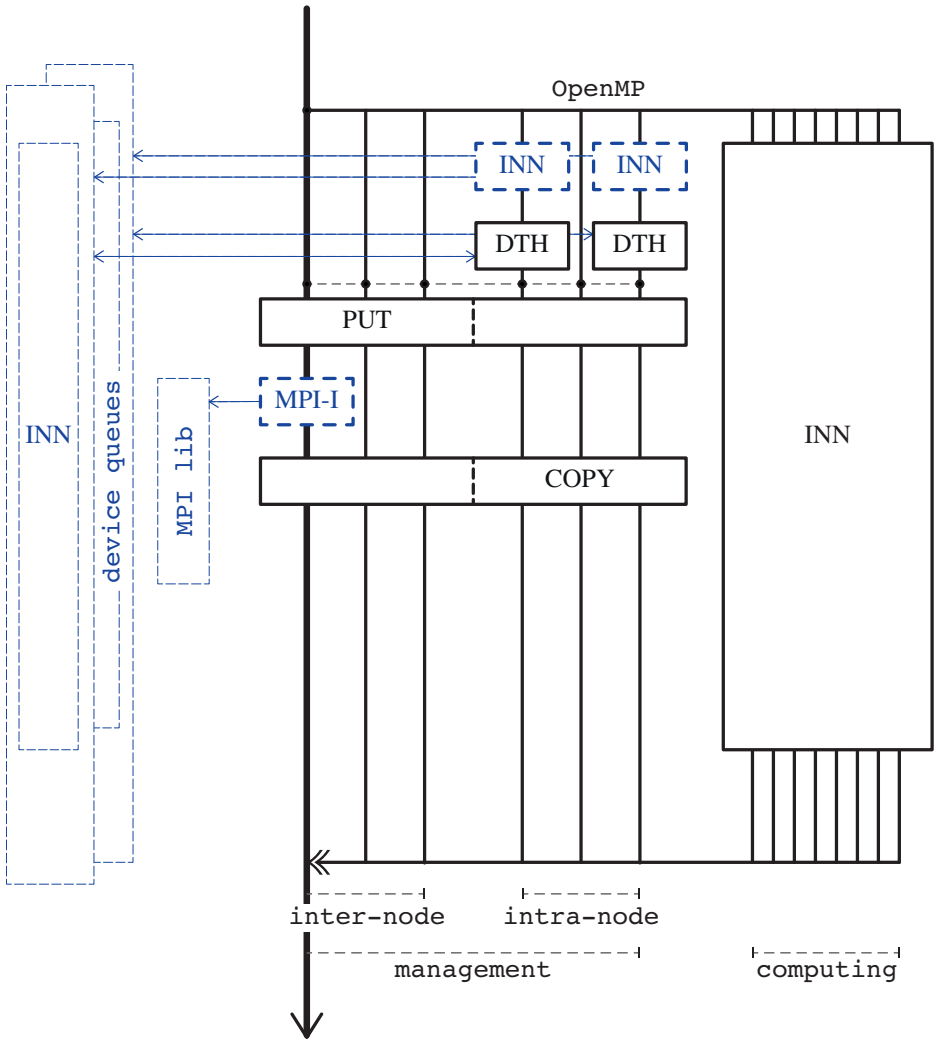
Flat multithreaded execution strategies

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



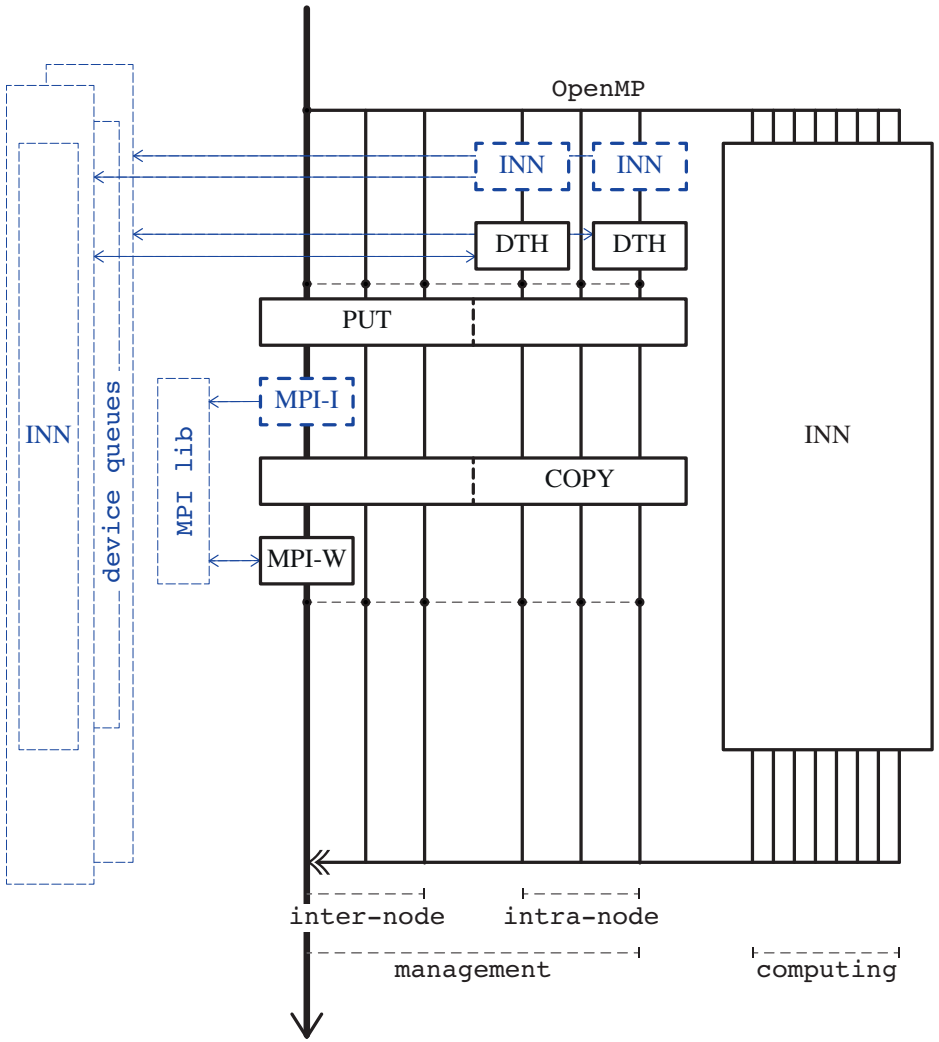
Flat multithreaded execution strategies

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



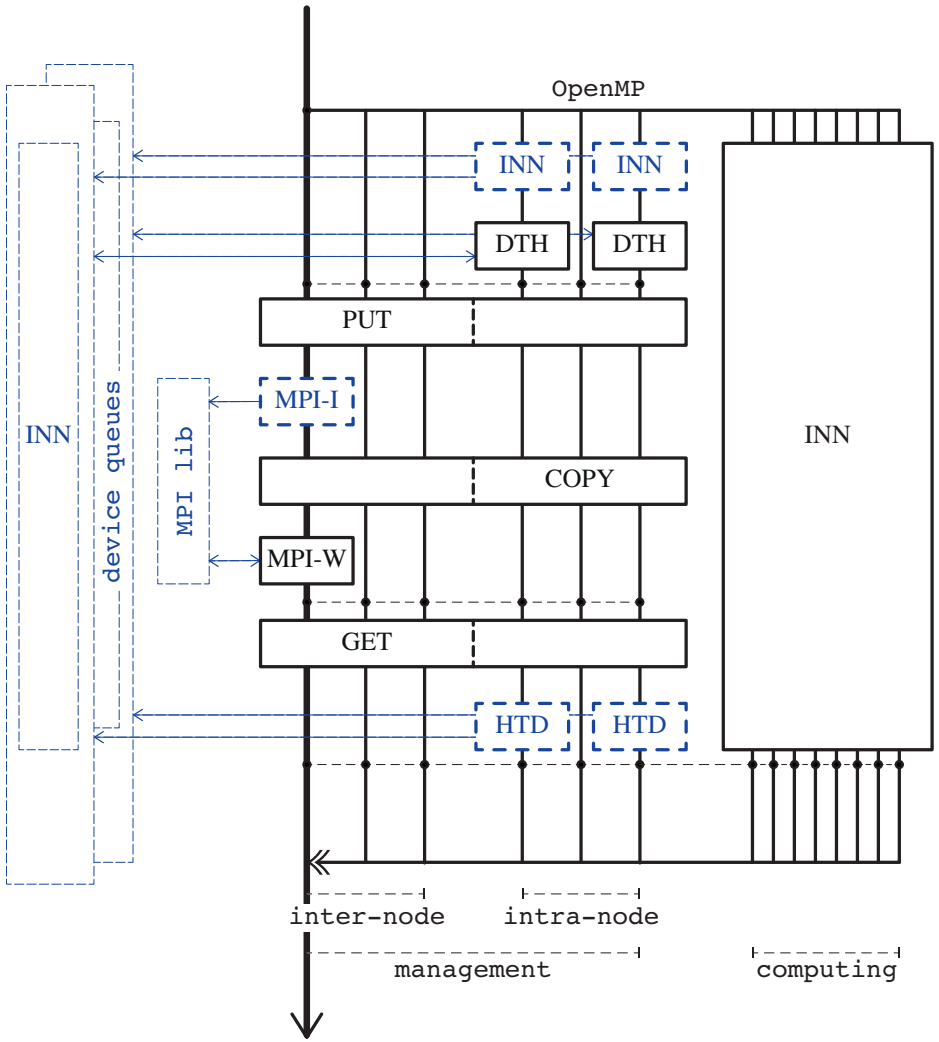
Flat multithreaded execution strategies

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



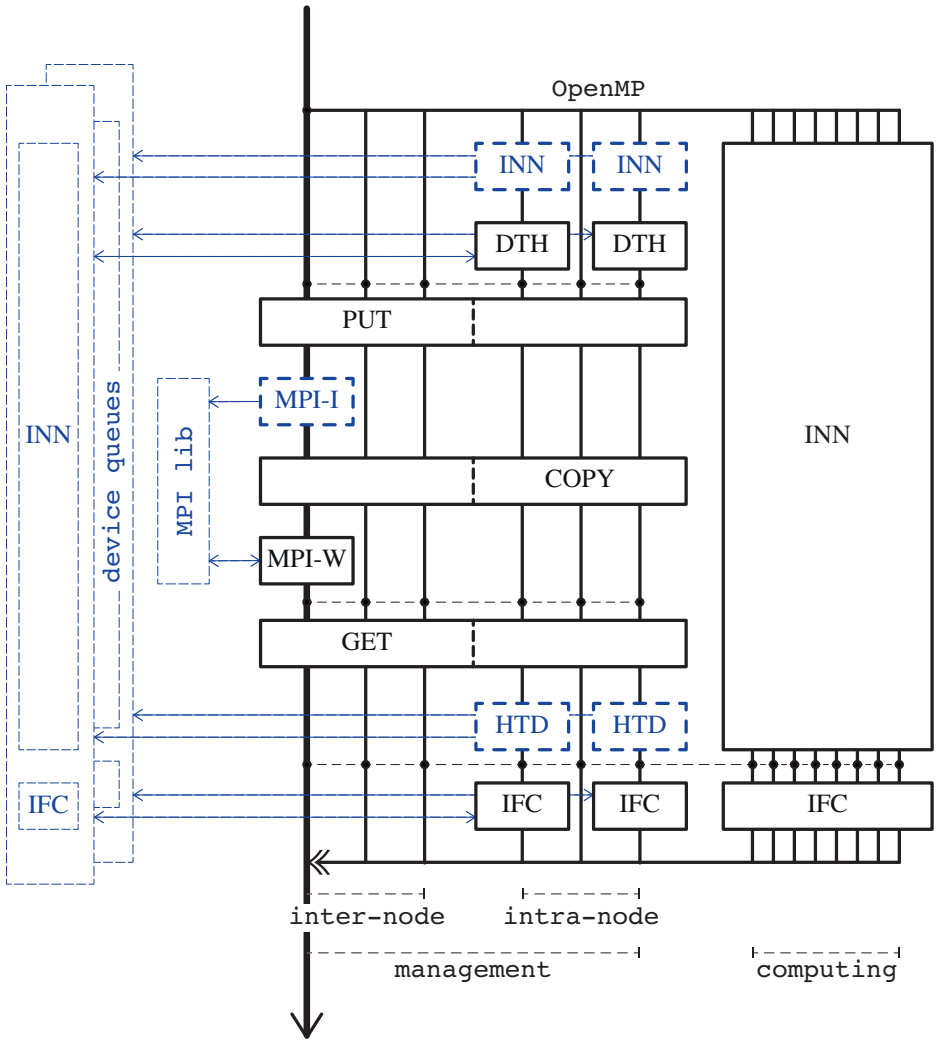
Flat multithreaded execution strategies

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

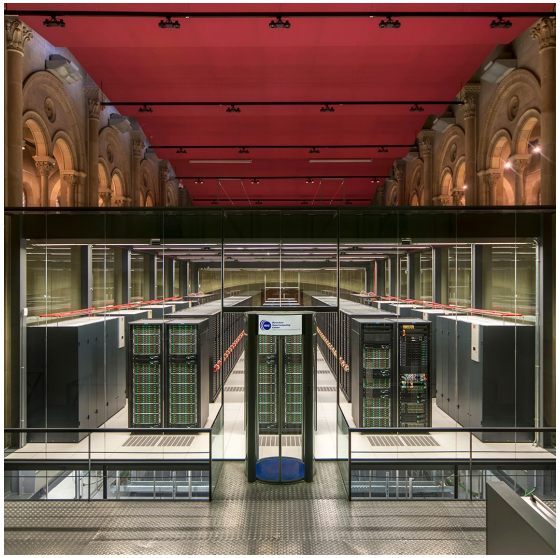
To minimise the overhead of the communications, efficient multithreaded execution strategies are required. Roughly, the idea is to overlap the communications with the computations.



Performance study

HPC systems

MareNostrum 4



#rank42
3456 nodes with:

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

Lomonosov-2

#rank156
1696 nodes with:

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

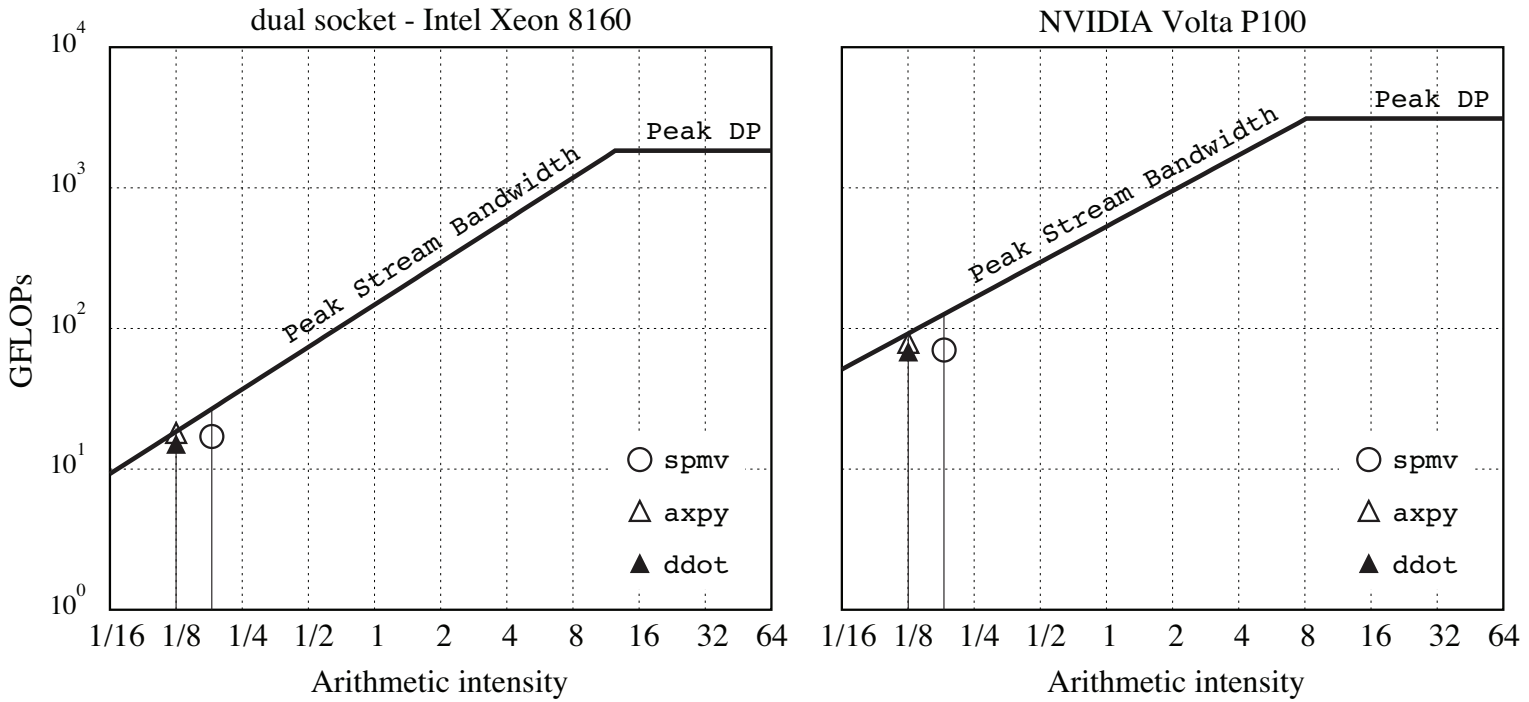
TSUBAME3.0

#rank31
540 nodes with:

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

Test case

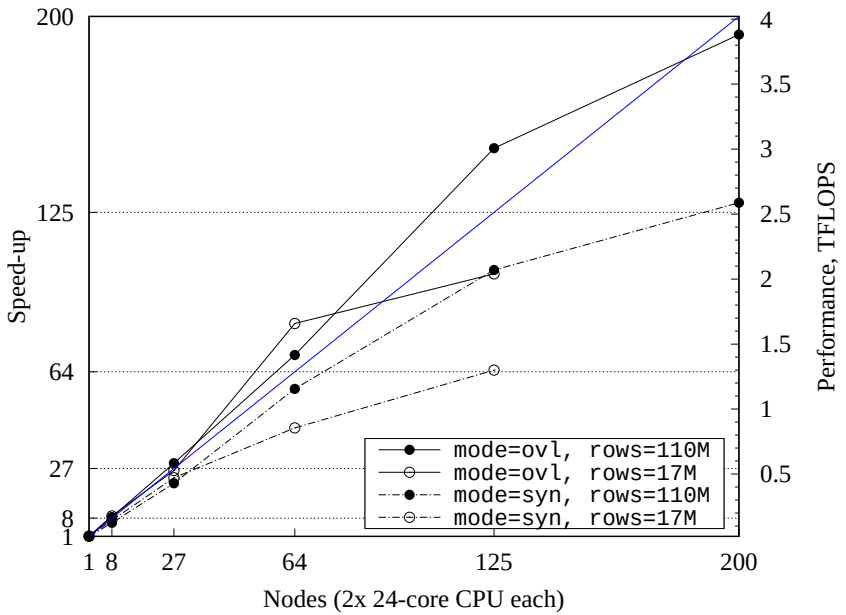
Single-node performance of SpMV, axpy and dot kernels shown in roofline model for two different architectures. The sparse matrix used arises from the symmetry-preserving discretization[3] of the Laplacian operator on unstructured hex-dominant mesh of 17 million cells. The sparse matrix storage format used is ELLPACK.

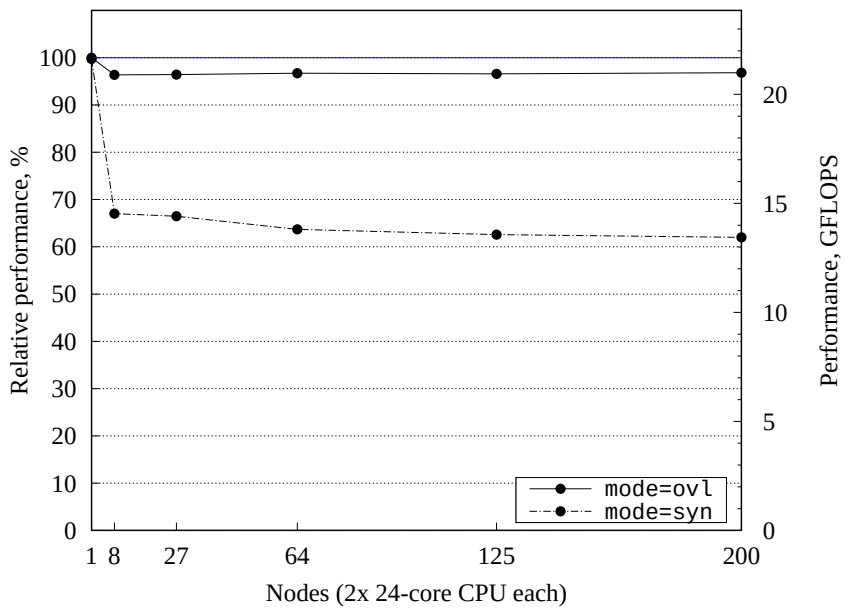


[3] Trias et al., Symmetry-preserving discretization of Navier-Stokes equations on collocated unstructured grids, J.Comp.Phys., 258, 246-267, 2014.

Test case

Multi-node strong (left) and weak (right) scaling of SpMV kernel on MareNostrum 4. The sparse matrix used arise from the symmetry-preserving discretization[4] of the Laplacian operator on unstructured hex-dominant mesh of 17 million cells (also 110 million in strong scaling). The sparse matrix storage format used is ELLPACK.

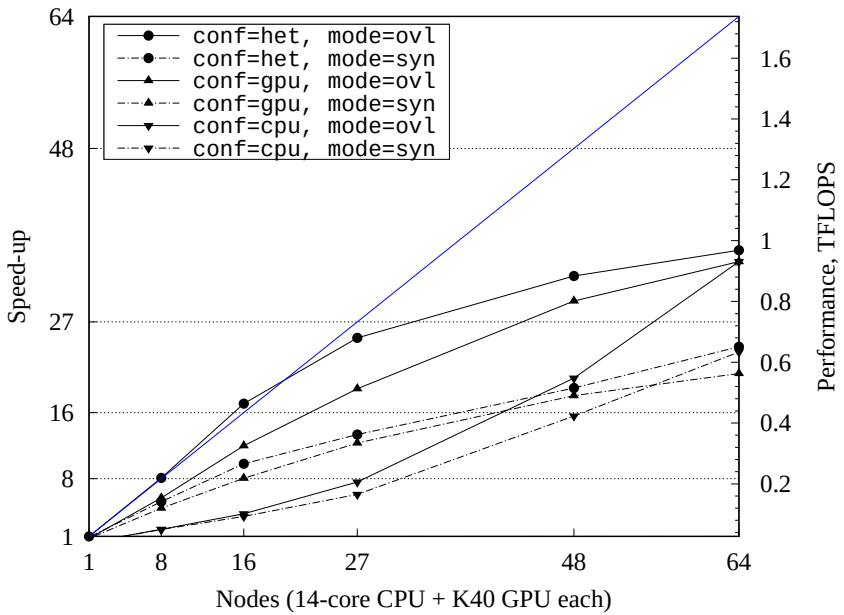


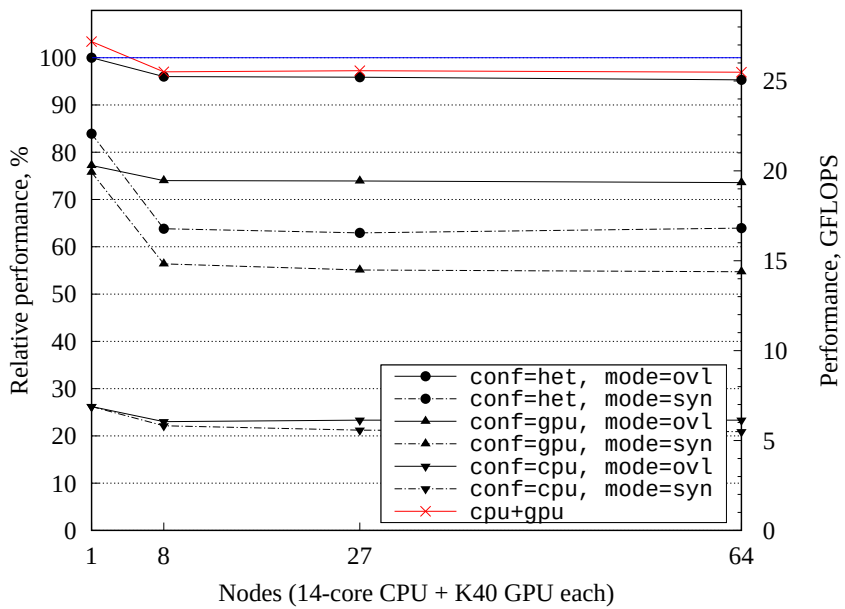


[4] Trias et al., Symmetry-preserving discretization of Navier-Stokes equations on collocated unstructured grids, J.Comp.Phys., 258, 246-267, 2014.

Test case

Multi-node strong (left) and weak (right) scaling of SpMV kernel on Lomonosov-2. The sparse matrix used arise from the symmetry-preserving discretization[5] of the Laplacian operator on unstructured hex-dominant mesh of 17 million cells. The sparse matrix storage format used is ELLPACK.

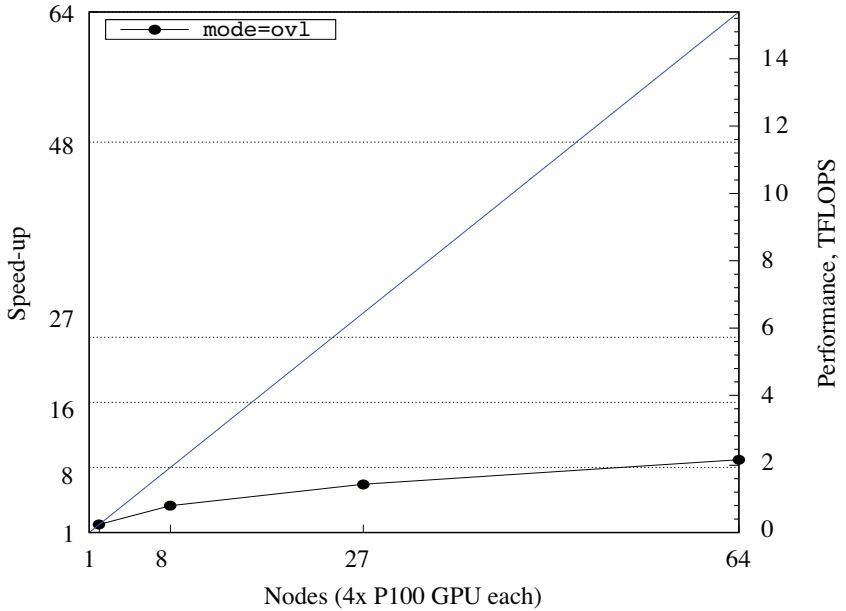


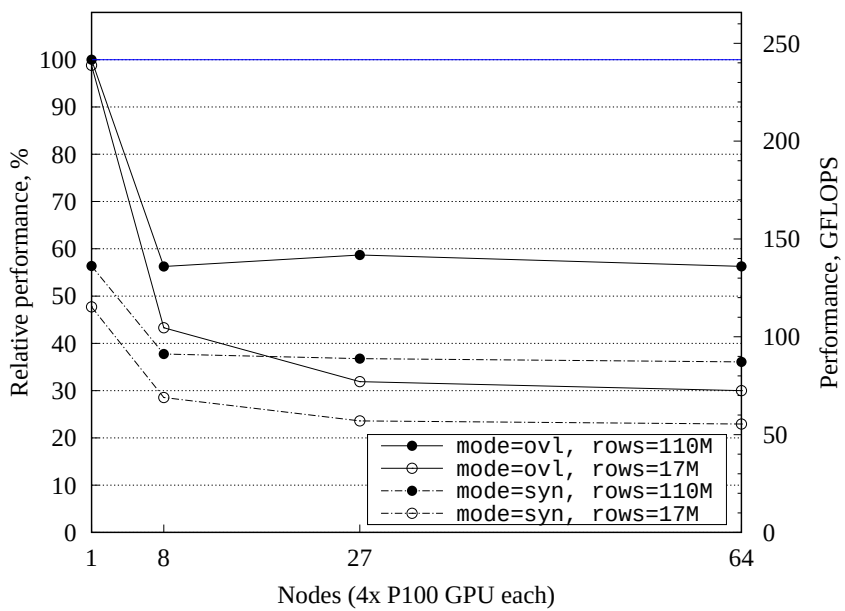


[5] Trias et al., Symmetry-preserving discretization of Navier-Stokes equations on collocated unstructured grids, J.Comp.Phys., 258, 246-267, 2014.

Test case

Multi-node strong (left) and weak (right) scaling of SpMV kernel on TSUBAME3.0. The sparse matrix used arise from the symmetry-preserving discretization[6] of the Laplacian operator on unstructured hex-dominant mesh of 17 million cells. The sparse matrix storage format used is ELLPACK.





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

Enabling larger and faster simulations

The SpMV is an essential operation in scientific computing, and therefore, it receives a great deal of attention. Given $\vec{x} \in \mathbb{R}^n$, $\vec{y} \in \mathbb{R}^m$, and $\mathsf{A} \in \mathbb{R}^{m \times n}$:

$$\vec{y} \leftarrow \mathsf{A}\vec{x} : AI_{spmv} = \frac{2nnz(\mathsf{A})}{12nnz(\mathsf{A}) + 4m + 8n + 8m} \approx 0.13.$$

Algorithm 2 SpMV implementation using the standard CSR matrix format.

Require: A, $\mathbf{x}$
Ensure: $\mathbf{y}$
1: **for** $i \leftarrow 1$ to m **do**
2: **for** $j \leftarrow \mathsf{A}.rptr[i]$ to $\mathsf{A}.rptr[i+1]$ **do**
3: $y[i] \leftarrow y[i] + \mathsf{A}.coef[j] \cdot \mathbf{x}[\mathsf{A}.cidx[j]]$
4: **end for**
5: **end for**

The SpMM represents the product of a sparse matrix by a dense matrix. It is very beneficial in terms of achievable performance to implement a specific SpMM that takes advantage of the reuse of the matrix coefficients. Given $\vec{x} \in \mathbb{R}^{kn}$, $\vec{y} \in \mathbb{R}^{km}$, and $\mathsf{A} \in \mathbb{R}^{m \times n}$:

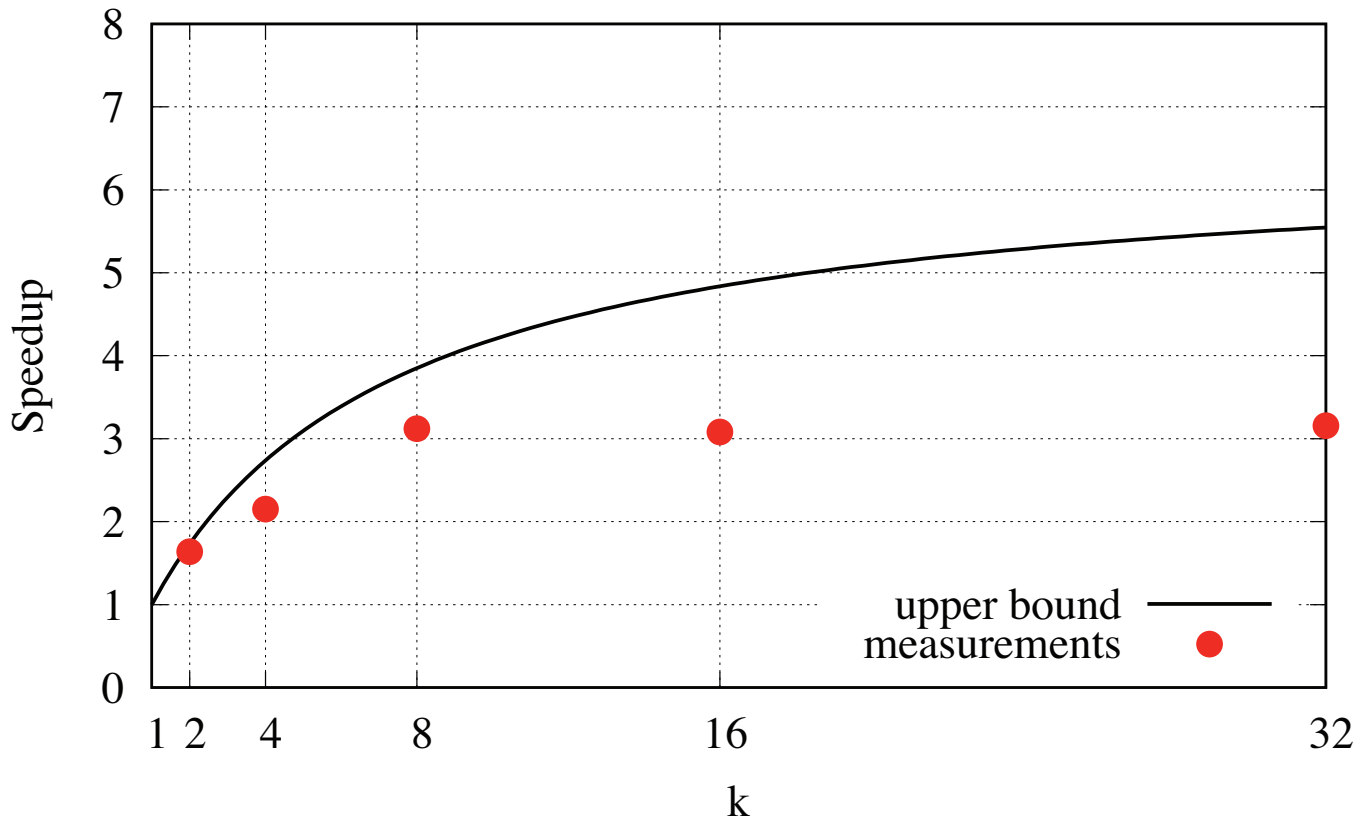
$$\begin{pmatrix} \mathbf{y}_1 \\ \vdots \\ \mathbf{y}_n \end{pmatrix} \leftarrow \begin{pmatrix} \mathsf{A} & & 0 \\ & \ddots & \\ 0 & & \mathsf{A} \end{pmatrix} \begin{pmatrix} \mathbf{x}_1 \\ \vdots \\ \mathbf{x}_n \end{pmatrix} : AI_{spmm} = \frac{2knnz(\mathsf{A})}{12nnz(\mathsf{A}) + 4m + 8kn + 8km}$$

Algorithm 3 SpMM implementation using the standard CSR matrix format.

Require: A, $\mathbf{x}$
Ensure: $\mathbf{y}$

1: **for** $i \leftarrow 1$ to m **do**
2: **for** $j \leftarrow \mathsf{A}.rptr[i]$ to $\mathsf{A}.rptr[i+1]$ **do**
3: **for** $k \leftarrow 1$ to K **do**
4: $y[i][k] \leftarrow y[i][k] + \mathsf{A}.coef[j] \cdot \mathbf{x}[\mathsf{A}.cidx[j]][k]$
5: **end for**
6: **end for**
7: **end for**

The gain of SpMM vs SpMV increases significantly with respect to k. In the plot, we show the theoretical values along with preliminary results, measured in our local JFF cluster with a 20-core Intel Xeon Gold 6230.



Applications of SpMM

First of all, remark that the SpMV is the most time-consuming kernel in a typical CFD simulation deployed in our framework, nearly 90%. Therefore, any SpMV optimization has a huge impact in performance.

- Multiple components of velocity in collocated formulation. Directly $k = 3$ for 3D simulations.

Applications of SpMM

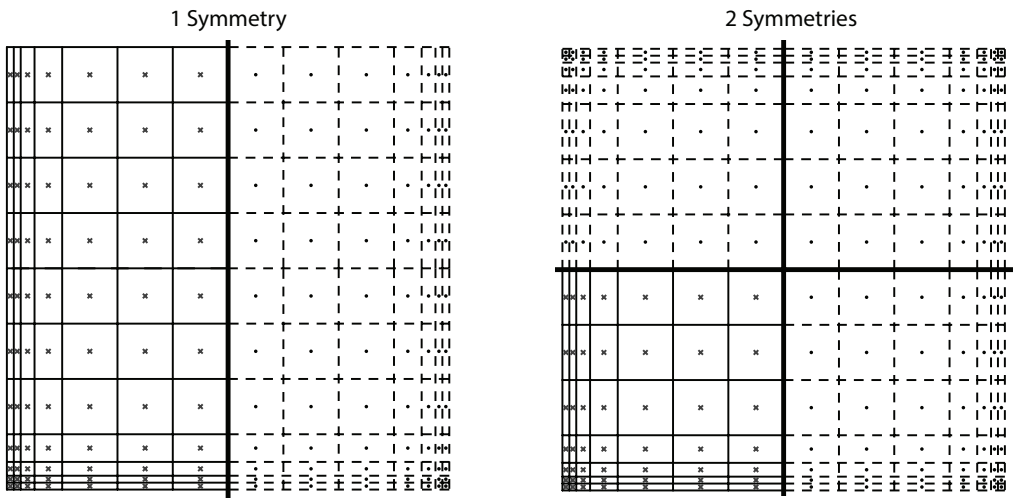
First of all, remark that the SpMV is the most time-consuming kernel in a typical CFD simulation deployed in our framework, nearly 90%. Therefore, any SpMV optimization has a huge impact in performance.

- Multiple components of velocity in collocated formulation. Directly $k = 3$ for 3D simulations.
- Multiple transport equations (*e.g.*, temperature, chemicals). Considering only temperature (Algorithm 1), increases to $k = 4$.

Applications of SpMM

First of all, remark that the SpMV is the most time-consuming kernel in a typical CFD simulation deployed in our framework, nearly 90%. Therefore, any SpMV optimization has a huge impact in performance.

- Multiple components of velocity in collocated formulation. Directly $k = 3$ for 3D simulations.
- Multiple transport equations (*e.g.*, temperature, chemicals). Considering only temperature (Algorithm 1), increases to $k = 4$.
- Simulations on a mesh with p symmetries. Increases k by a factor of p^2, and also reduces memory footprint of discrete operators.



A highly portable heterogeneous implementation of a Poisson solver for flows with mesh symmetries

Àdel Alsalti-Baldellou[1,2], F. Xavier Trias[1], Xavier Álvarez-Farré[1] and Assensi Oliva[1]

[1] Heat and Mass Transfer Technological Center, Technical University of Catalonia
C/ Colom 11, 08222 Terrassa (Barcelona), Spain; adel@cttc.upc.edu

[2] Termo Fluids SL, C/ Mag ́ı Colet 8, 08204 Sabadell (Barcelona), Spain

17-19 May 2021, ParCFD 2021

A highly portable heterogeneous implementation of a Poisson solver for flows with mesh symmetries

Àdel Alsalti-Baldellou[1,2], F. Xavier Trias[1], Xavier Álvarez-Farré[1] and Assensi Oliva[1]

[1] Heat and Mass Transfer Technological Center, Technical University of Catalonia
C/ Colom 11, 08222 Terrassa (Barcelona), Spain; adel@cttc.upc.edu

[2] Termo Fluids SL, C/ Mag ı́ Colet 8, 08204 Sabadell (Barcelona), Spain

17-19 May 2021, ParCFD 2021

The direct benefits of this approach are:

- Reduces the number of iterations and its computational cost (increased AI).

A highly portable heterogeneous implementation of a Poisson solver for flows with mesh symmetries

Àdel Alsalti-Baldellou[1,2], F. Xavier Trias[1], Xavier Álvarez-Farré[1] and Assensi Oliva[1]

[1] Heat and Mass Transfer Technological Center, Technical University of Catalonia
C/ Colom 11, 08222 Terrassa (Barcelona), Spain; adel@cttc.upc.edu

[2] Termo Fluids SL, C/ Mag ı́ Colet 8, 08204 Sabadell (Barcelona), Spain

17-19 May 2021, ParCFD 2021

The direct benefits of this approach are:

- Reduces the number of iterations and its computational cost (increased AI).
- Reduces substantially the memory footprint of the matrices.

A highly portable heterogeneous implementation of a Poisson solver for flows with mesh symmetries

Àdel Alsalti-Baldellou[1,2], F. Xavier Trias[1], Xavier Álvarez-Farré[1] and Assensi Oliva[1]

[1] Heat and Mass Transfer Technological Center, Technical University of Catalonia
C/ Colom 11, 08222 Terrassa (Barcelona), Spain; adel@cttc.upc.edu

[2] Termo Fluids SL, C/ Mag ı́ Colet 8, 08204 Sabadell (Barcelona), Spain

17-19 May 2021, ParCFD 2021

The direct benefits of this approach are:

- Reduces the number of iterations and its computational cost (increased AI).
- Reduces substantially the memory footprint of the matrices.
- Reduces substantially the cost of building complex preconditioners.

Conclusions and Future Work

Conclusions

Future Work

Conclusions

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

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.

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.
- HPC systems with extremely high ratios of memory bandwidth to network bandwidth (fat nodes) are harmful for ligth memory bound kernels such as SpMV; the calculations become too fast to hide the communications. In the case of TSUBAME3.0, this ratio was 2928:50.

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.
- HPC systems with extremely high ratios of memory bandwidth to network bandwidth (fat nodes) are harmful for ligth memory bound kernels such as SpMV; the calculations become too fast to hide the communications. In the case of TSUBAME3.0, this ratio was 2928:50.
- The application of SpMM within algebraic frameworks is demonstrated to be versatile and powerful. Particularly, in the presence of mesh symmetries the benefits are threefold: reduces number of iterations, computational cost and memory footprint.

Future Work

- To design a new update mechanism to accelerate the data exchanges, for instance, taking into account NUMA 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.
- HPC systems with extremely high ratios of memory bandwidth to network bandwidth (fat nodes) are harmful for ligth memory bound kernels such as SpMV; the calculations become too fast to hide the communications. In the case of TSUBAME3.0, this ratio was 2928:50.
- The application of SpMM within algebraic frameworks is demonstrated to be versatile and powerful. Particularly, in the presence of mesh symmetries the benefits are threefold: reduces number of iterations, computational cost and memory footprint.

Future Work

- To design a new update mechanism to accelerate the data exchanges, for instance, taking into account NUMA factor in inter- and intra-node exchanges.
- Applying our framework to multiple parameters simulations. Considering n different simulations, increases k by a factor of n, allowing for running multiple simulations faster while maintaining the memory footprint of discrete operators constant.

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.
- HPC systems with extremely high ratios of memory bandwidth to network bandwidth (fat nodes) are harmful for ligth memory bound kernels such as SpMV; the calculations become too fast to hide the communications. In the case of TSUBAME3.0, this ratio was 2928:50.
- The application of SpMM within algebraic frameworks is demonstrated to be versatile and powerful. Particularly, in the presence of mesh symmetries the benefits are threefold: reduces number of iterations, computational cost and memory footprint.

Future Work

- To design a new update mechanism to accelerate the data exchanges, for instance, taking into account NUMA factor in inter- and intra-node exchanges.
- Applying our framework to multiple parameters simulations. Considering n different simulations, increases k by a factor of n, allowing for running multiple simulations faster while maintaining the memory footprint of discrete operators constant.
- Applying our framework to parallel-in-time simulations. Considering t decompositions in time, increases k by a factor of t, allowing for solving multiple time-intervals faster while maintaining the memory footprint of discrete operators constant.

Thank you for your attention