FIGHTING AGAINST MASSIVELY PARALLEL ACCELERATORS OF VARIOUS ARCHITECTURES FOR THE EFFICIENCY OF FINITE-VOLUME PARALLEL CFD CODES

A. V. Gorobets[1,2], F. X. Trias[1], A. Oliva[1]

1 Centre Tecnològic de Transferència de Calor
UNIVERSITAT POLITÈCNICA DE CATALUNYA
Heat and Mass Transfer Technological Center
Technical University of Catalonia, Barcelona, Spain

2 **CAA LAB of KIAM RAS,** Moscow, Russia

Zoo of architectures and frameworks

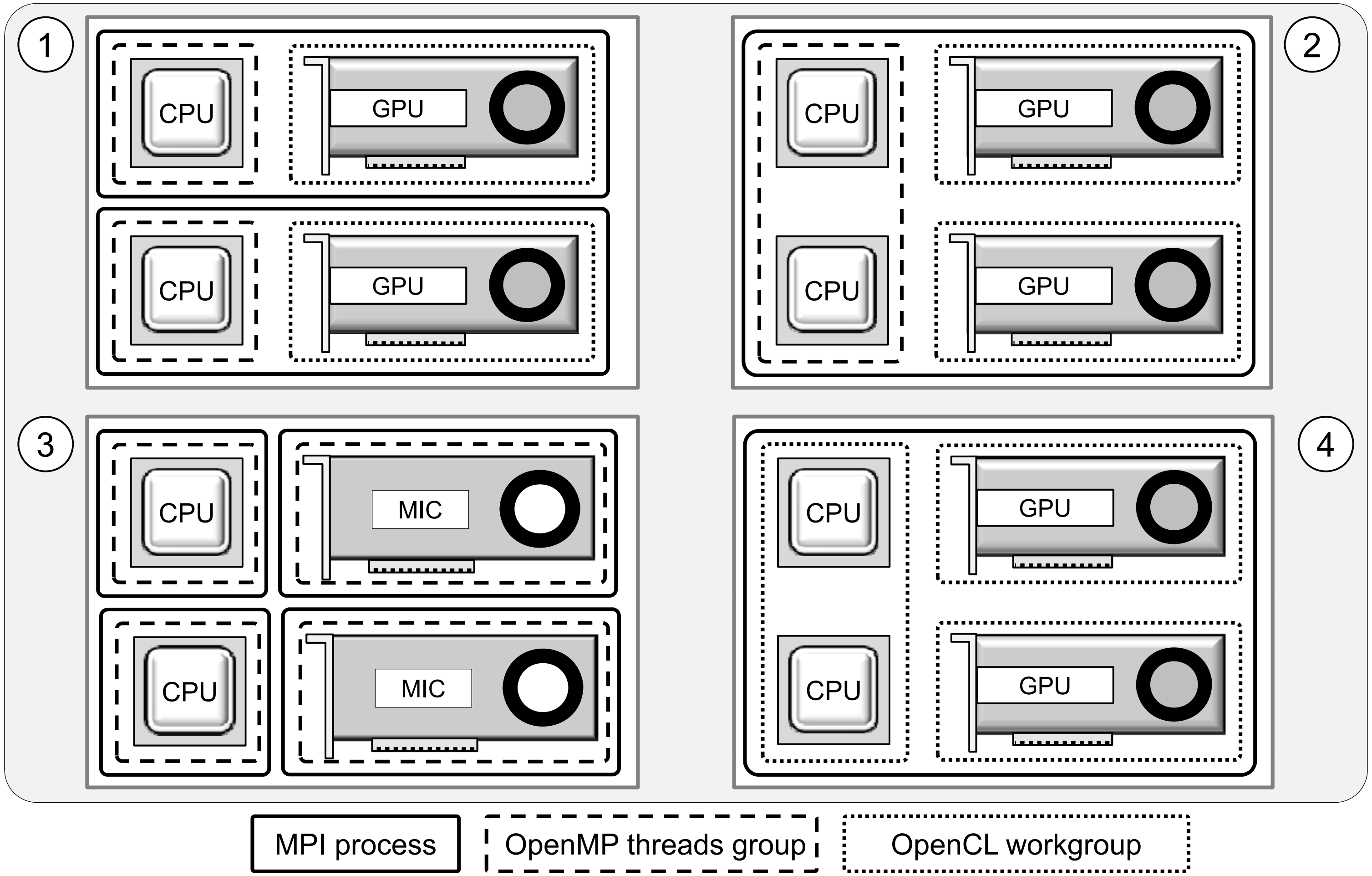
- Too many different architectures are competing:
  CPU, Intel Xeon Phi, AMD GPU, NVIDIA GPU, ARM, …

- Complexity of multi-level parallel model
- Variety of proprietary frameworks that are nonsense to rely upon
- Dramatic insufficiency of memory bandwidth for the available computing power
- SIMD, stream processing - limited forms of parallel processing that deny many existing fast and computationally efficient algorithms
- Increasingly more difficult to use supercomputers efficiently

Parallel programming approaches

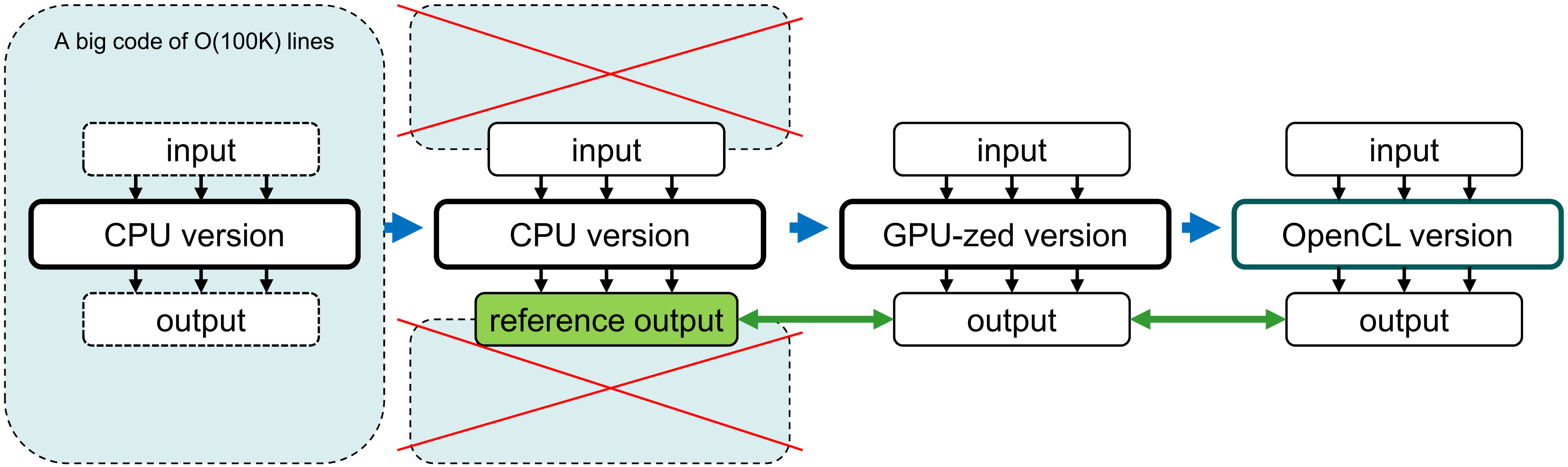
	Cluster	CPU	NVIDIA GPU	AMD GPU	Intel Xeon Phi	ARM
MPI	+	+	-	-	±	-
OpenMP	-	+	-	-	+	-
OpenCL	-	+	+	+	+	+
CUDA	-	-	+	-	-	-
Distributed memory MIMD	+	+	-	-	-	-
Shared memory MIMD	-	+	-	-	+	-
Stream processing	+	+	+	+	+	+

Multi-level parallelization



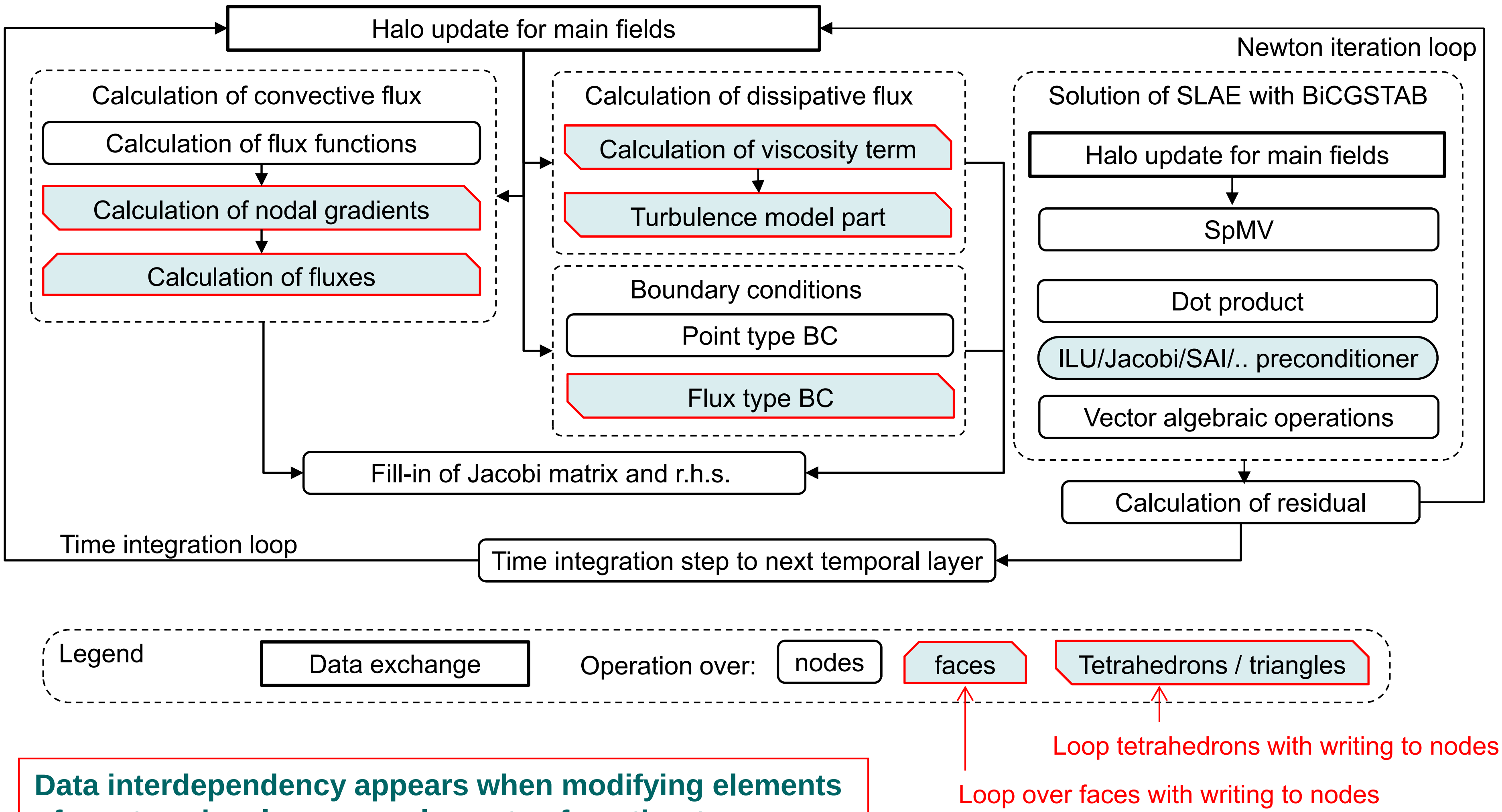
Tactics: divide and conquer

- Decomposition of algorithm into basic operations – as few as possible
- For each operation create standalone test with wrapping that includes profiling, inputs and reference outputs in files
- For each operation create “GPU-zed” version for CPU:
  - adapt an operation to stream processing at lower parallelization level
  - make alternative data structures that fit better accelerator architectures
  - mimic execution on accelerator by external parallel loop that iterates ranks of a workgroup
- For each operation on a base of GPUzed version create an OpenCL kernel and ensure correctness
- Optimize OpenCL version ensuring correctness every small step



Fit to stream processing or die

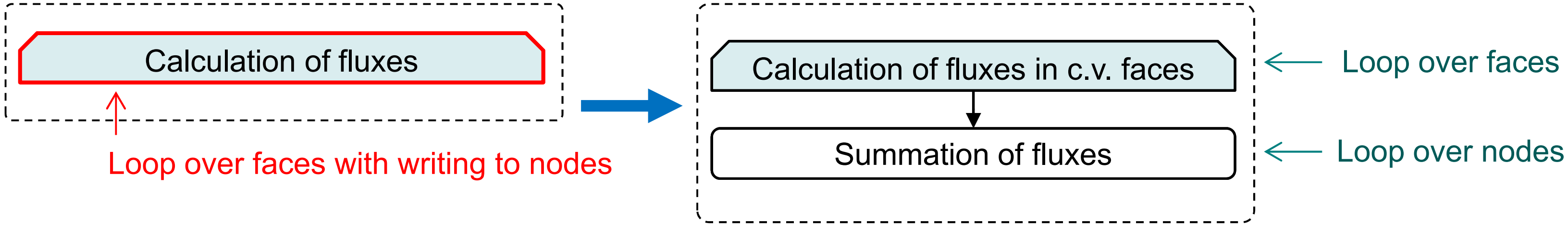
Outline of an edge-based vertex-centered algorithm



Data interdependency appears when modifying elements of one type in a loop over elements of another type

Fit to stream processing or die

1) **Decomposition of operation into two operations over elements of different type:**



- **Additional array over faces stores intermediate result**
- **Summation is in the loop over nodes using CSR-like inverse topology**

- Needs additional memory and can't be used if intermediate storage is too big

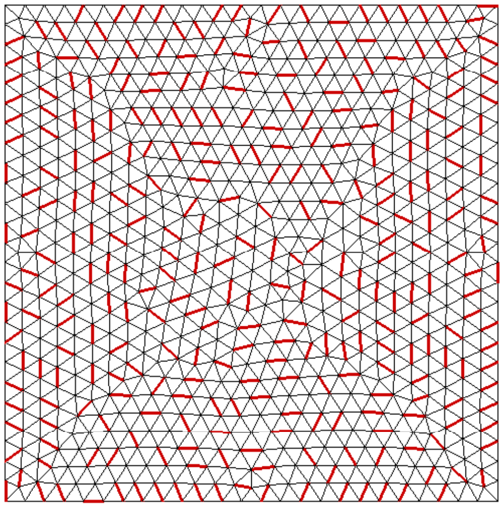
2) **Decomposition of operation into multiple operations over subsets:**

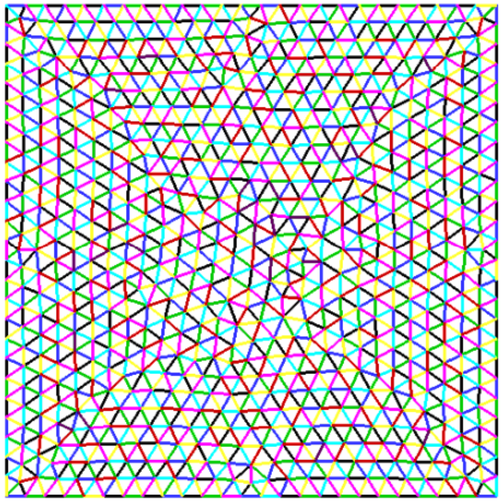
- **Edges of the connectivity graph (dual mesh graph etc.) are colored in a way that no edges of one color share same node**

- Needs multiple kernel executions

- Inefficient memory access: no way for coalescing

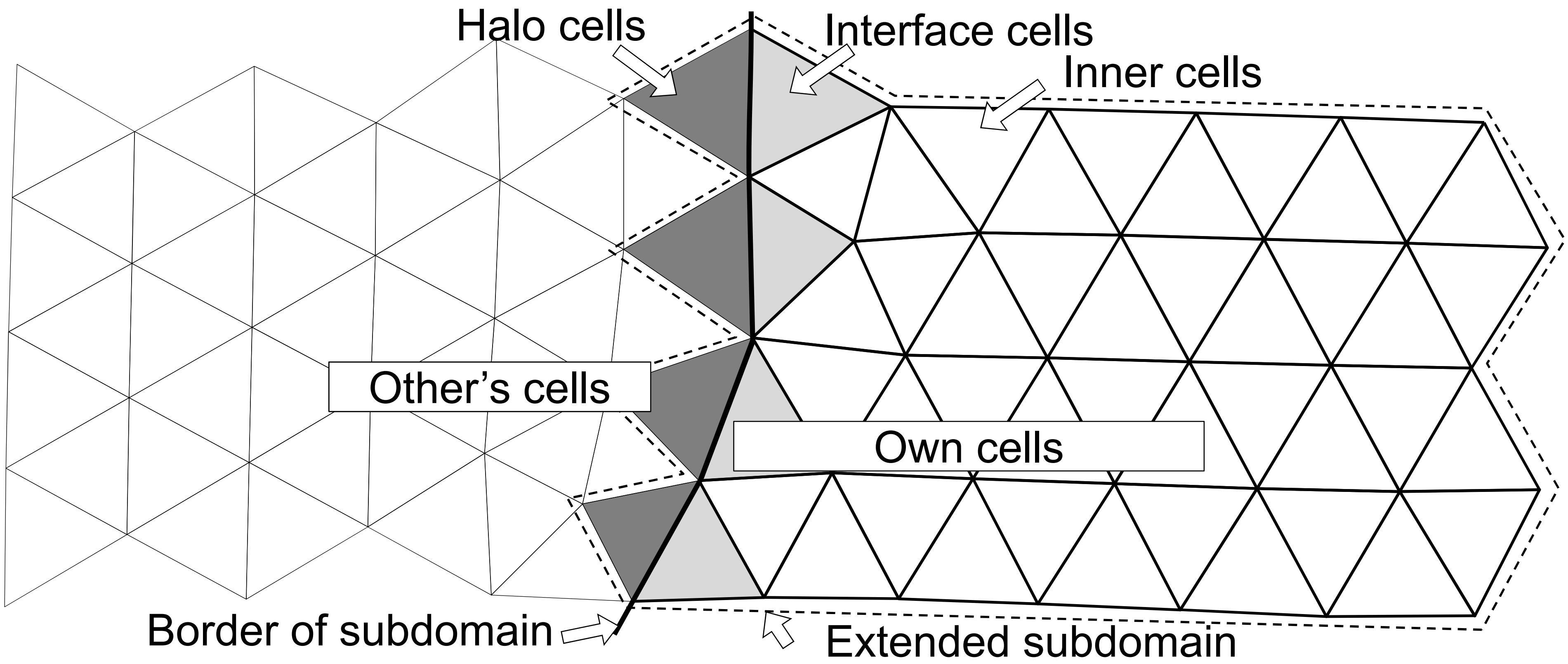
- around 10-20% slower if 1) applicable



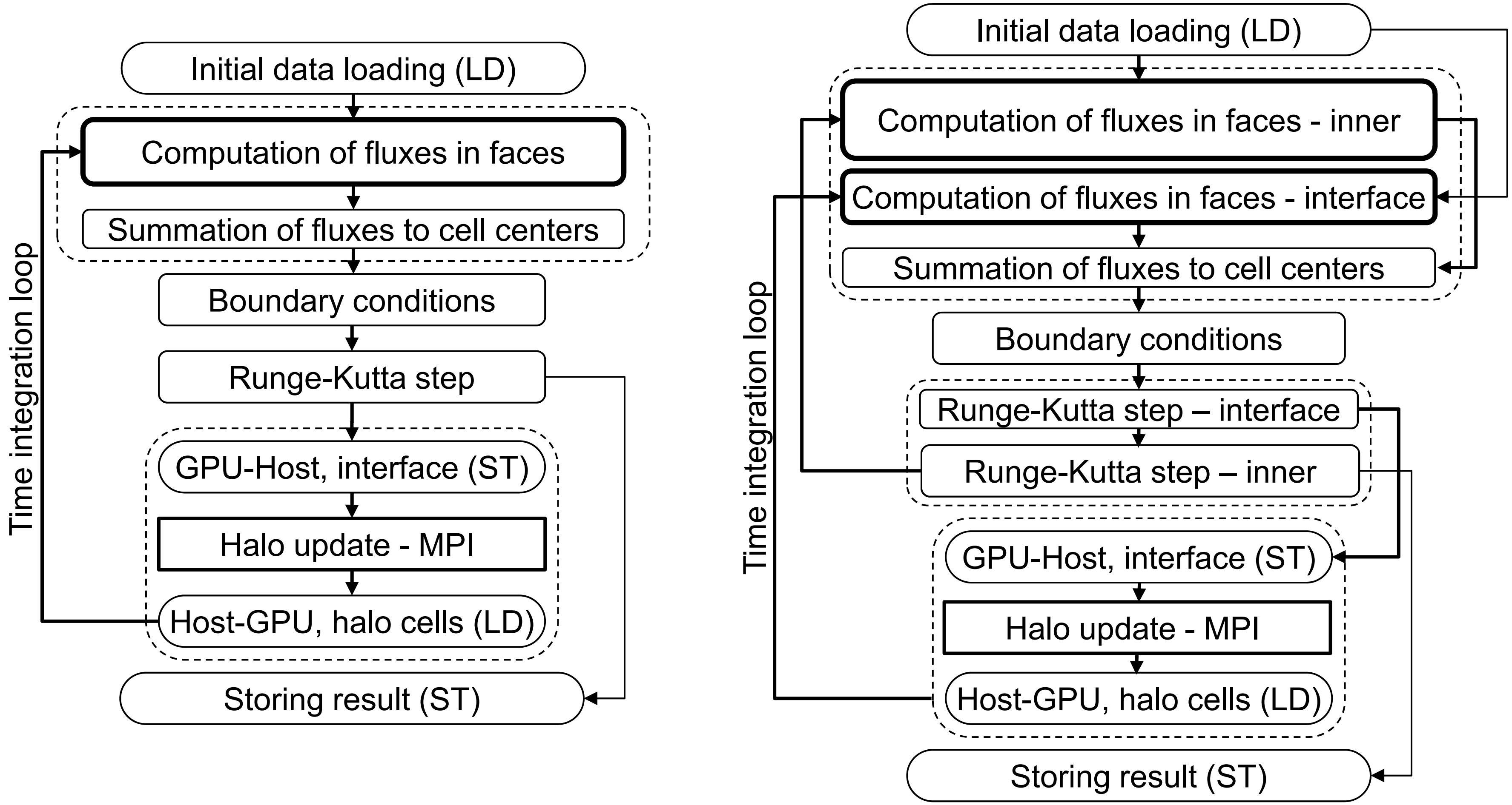


Sets of mesh elements in a domain decomposition

- A cell (or a mesh element) that belongs to a subdomain is its own cell
- An own cell that is coupled with a cell from another subdomain is an interface cell
- An own cell that is coupled only with own cells is an inner cell
- A cell from another subdomain that is coupled with an own cell is a halo cell

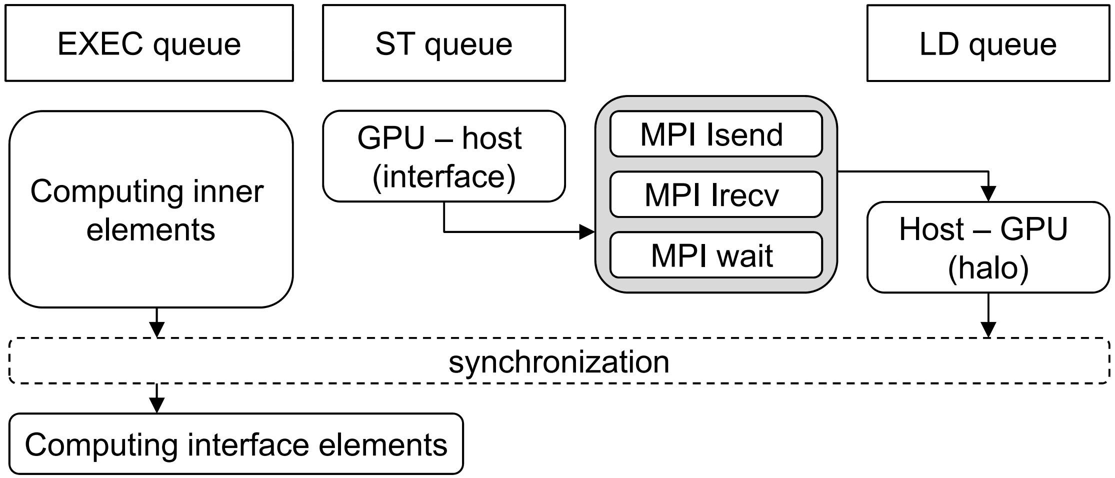


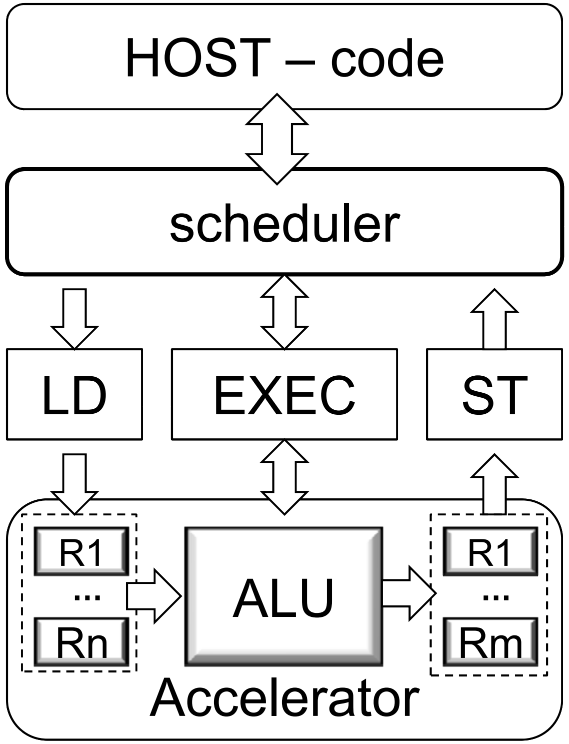
Graph of a finite-volume algorithm



Scheduling infrastructure

- Register is a region in device global memory, instruction is an OpenCL kernel for the device
- Scheduler has with 3 queues: LD, ST, EXEC
  - LD command: load from host to device
  - ST command: from device to host
  - EXEC command launches a kernel on a device
- Independent commands can run simultaneously

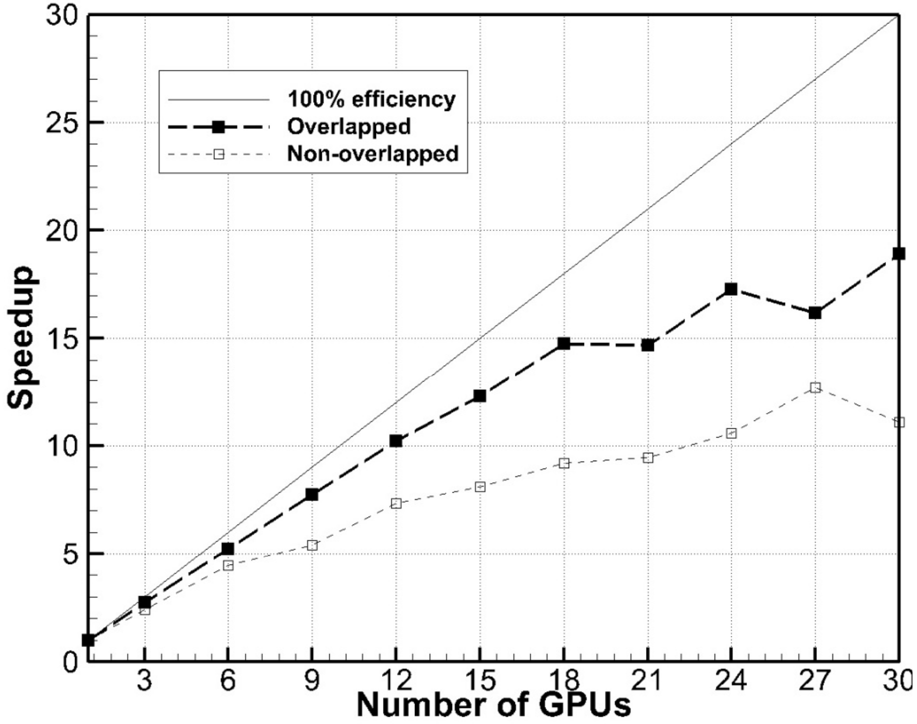


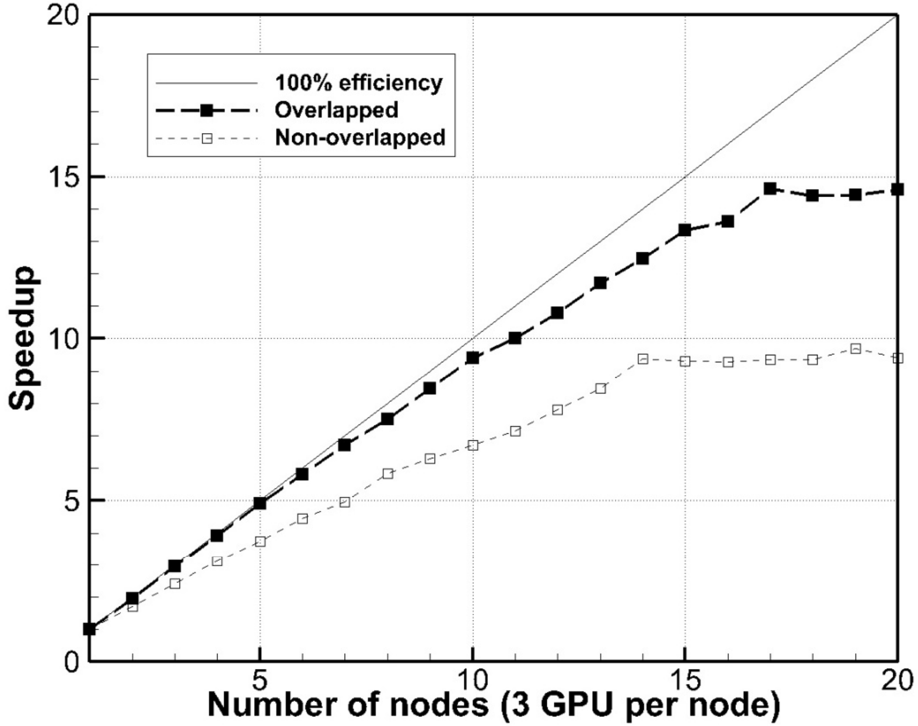


Bogdanov P. B., Efremov A. A. from Scientific Research Institute of System Development of RAS
Programming infrastructure of heterogeneous computing based on OpenCL and its applications
GPU Technology Conference GTC-2013, March 18-21, San Jose, California, USA.

Speedup on a hybrid supercomputer

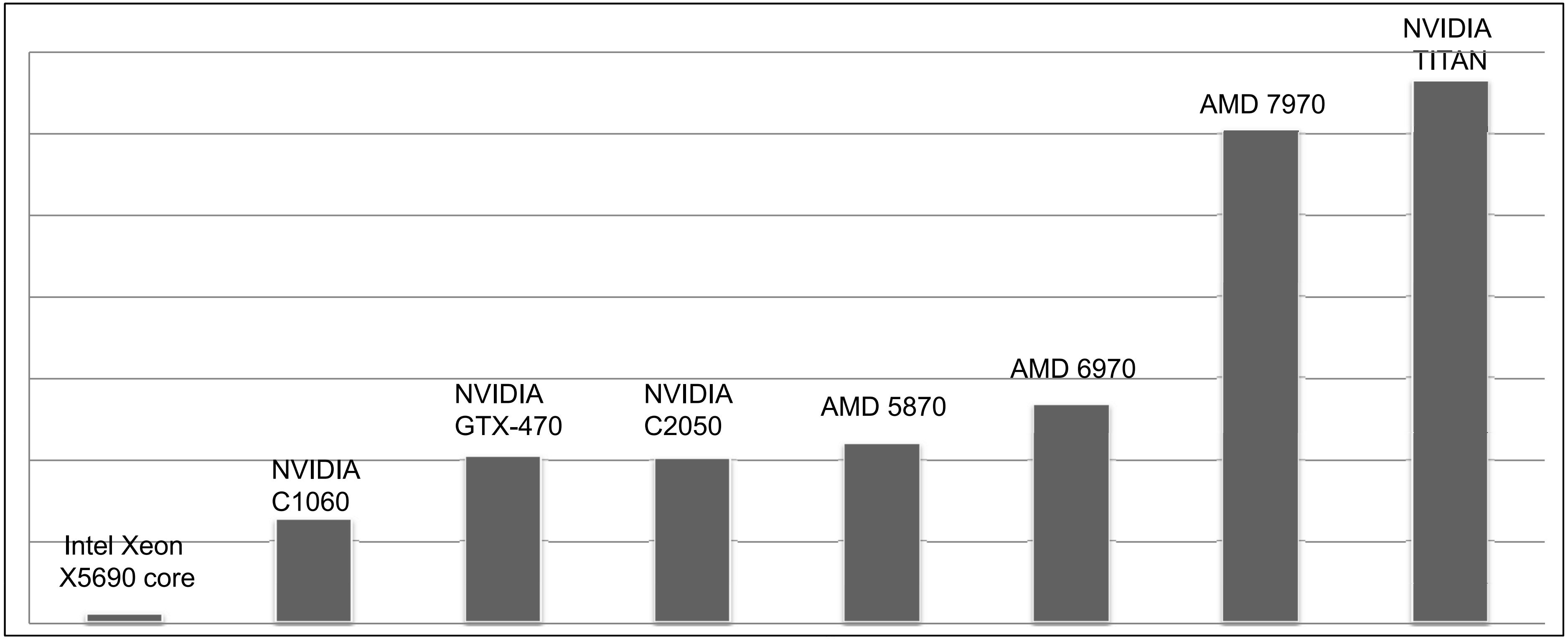
Speedups on K100 supercomputer for a mesh with 4 millions of cells (left) starting from one GPU and for a mesh with 16 millions of cells (right) starting from one computing node.





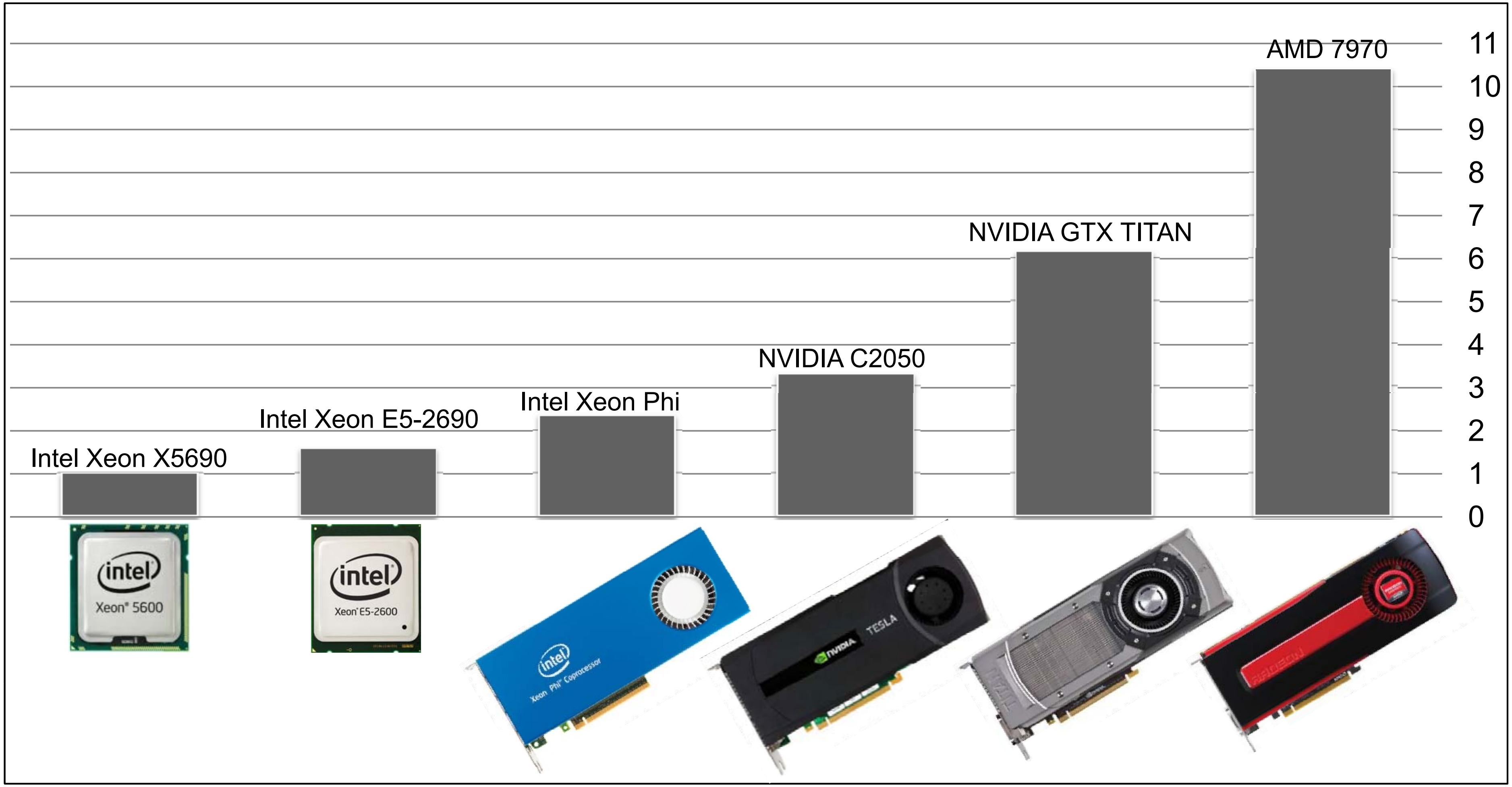
Comparison of performance

Comparison of computing devices on calculation of overall time step – 1st order scheme



Comparison of performance

Comparison of computing devices on calculation of overall time step – 2nd order scheme



Algorithm for incompressible flows

- **Navier-Stokes system to solve:**

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

$$\frac{\partial\mathbf{u}}{\partial t}+(\mathbf{u}\cdot\nabla)\mathbf{u}=\frac{\mathrm{Pr}}{\sqrt{\mathrm{Ra}}}\nabla^2\mathbf{u}-\nabla p+\mathbf{f},$$

$$\frac{\partial T}{\partial t}+(\mathbf{u}\cdot\nabla)T=\frac{1}{\sqrt{\mathrm{Ra}}}\nabla^2 T.$$

- **Discrete system for pressure-velocity coupling:**

$$\frac{\mathbf{u}^{n+1}-\mathbf{u}^n}{\Delta t}=\frac{3}{2}\mathbf{R}^n-\frac{1}{2}\mathbf{R}^{n-1}-Gp^{n+1},$$

$$M\mathbf{u}^{n+1}=0,$$

where $\mathbf{R}(\mathbf{u})=-C(\mathbf{u})\mathbf{u}-D\mathbf{u}+f$

- **Fractional step projection method:**

Predictor velocity: $\mathbf{u}^p=\mathbf{u}^n+\Delta t\left(\frac{3}{2}\mathbf{R}^n-\frac{1}{2}\mathbf{R}^{n-1}\right)$

Unknown velocity: $\mathbf{u}^{n+1}=\mathbf{u}^p-G\tilde{p},$ where $\tilde{p}=\Delta t p^{n+1}$

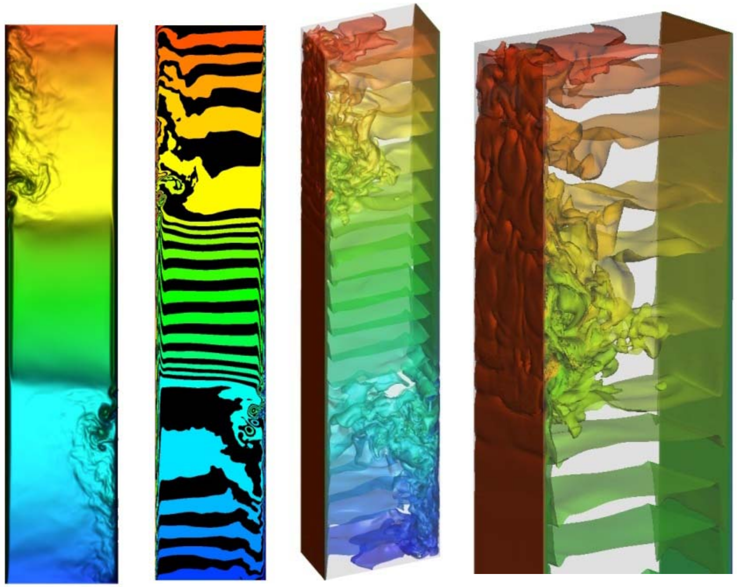
Mass conservation equation: $M\mathbf{u}^{n+1}=M\mathbf{u}^p-GM\tilde{p}=0$

$$M\mathbf{u}^{n+1}=M\mathbf{u}^p-GM\tilde{p}=-M\Omega M^*\tilde{p}=\boxed{L\tilde{p}=M\mathbf{u}^p}$$

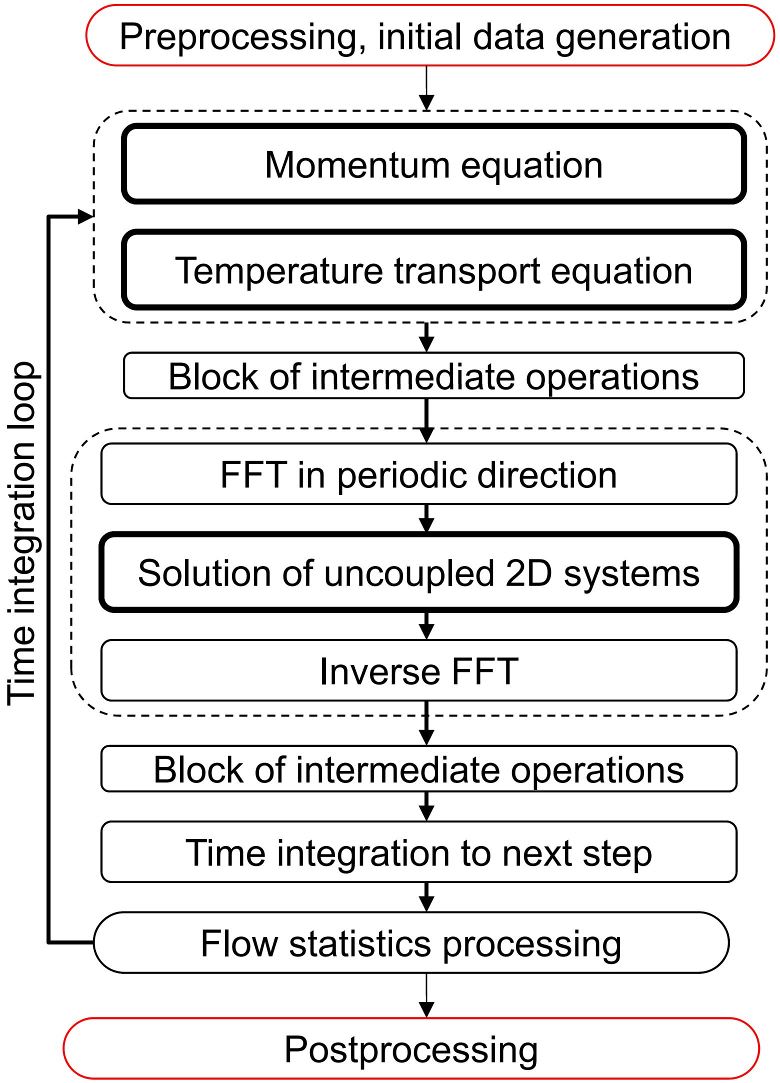
The Poisson equation

The algorithm of the time step

1. Predictor velocity field, $\mathbf{u}^p$, is obtained explicitly
2. Temperature transport equation is solved explicitly
3. Correction, $\tilde{p}$, is obtained from the Poisson equation
4. Resulting velocity field, $\mathbf{u}^{n+1}$, is obtained



Outline of the algorithm



Basic operations

T1. **Linear operator**

T2. **Nonlinear operator**

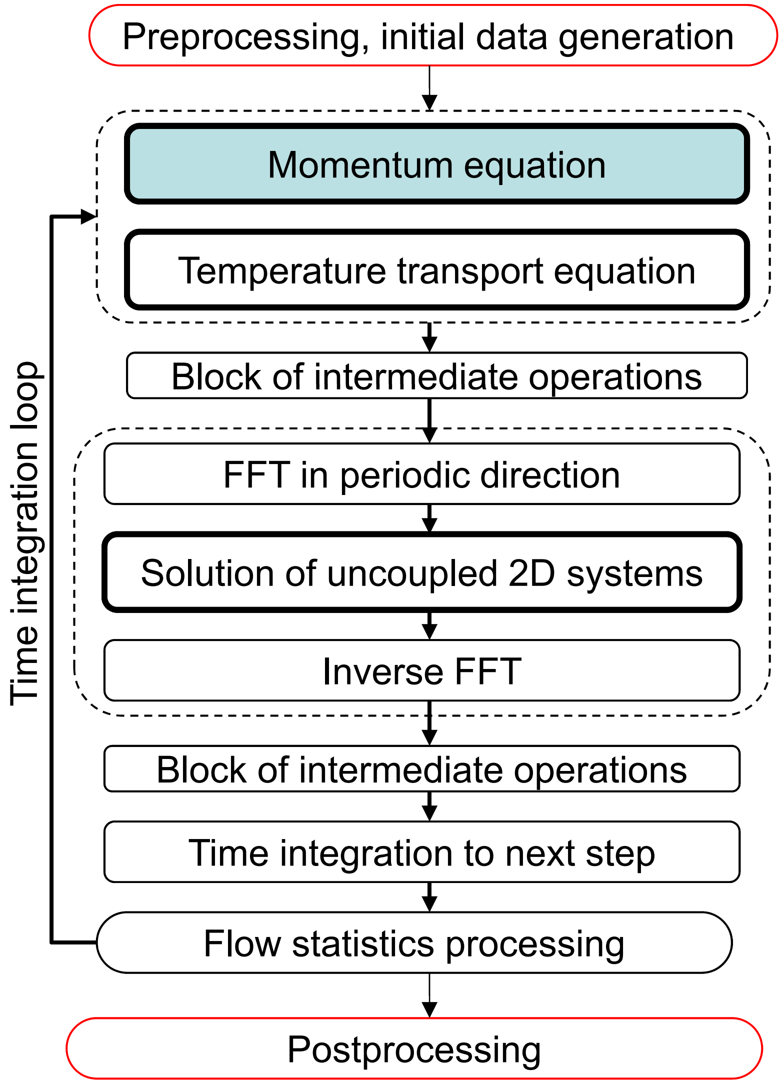
T3. $\mathbf{y} = \sum_{i=1}^{n} a_i \mathbf{x}_i + c$

T4. **FFT, inverse IFFT**

T5. **multi-SpMV**

T6. **reduction – norm, dot product**

Outline of the algorithm



Basic operations

T1. **Linear operator**

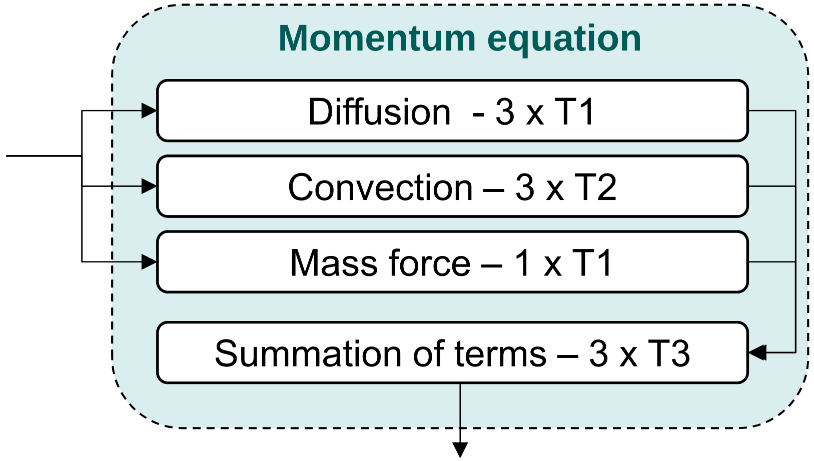
T2. **Nonlinear operator**

T3. $\mathbf{y} = \sum_{i=1}^{n} a_i \mathbf{x}_i + c$

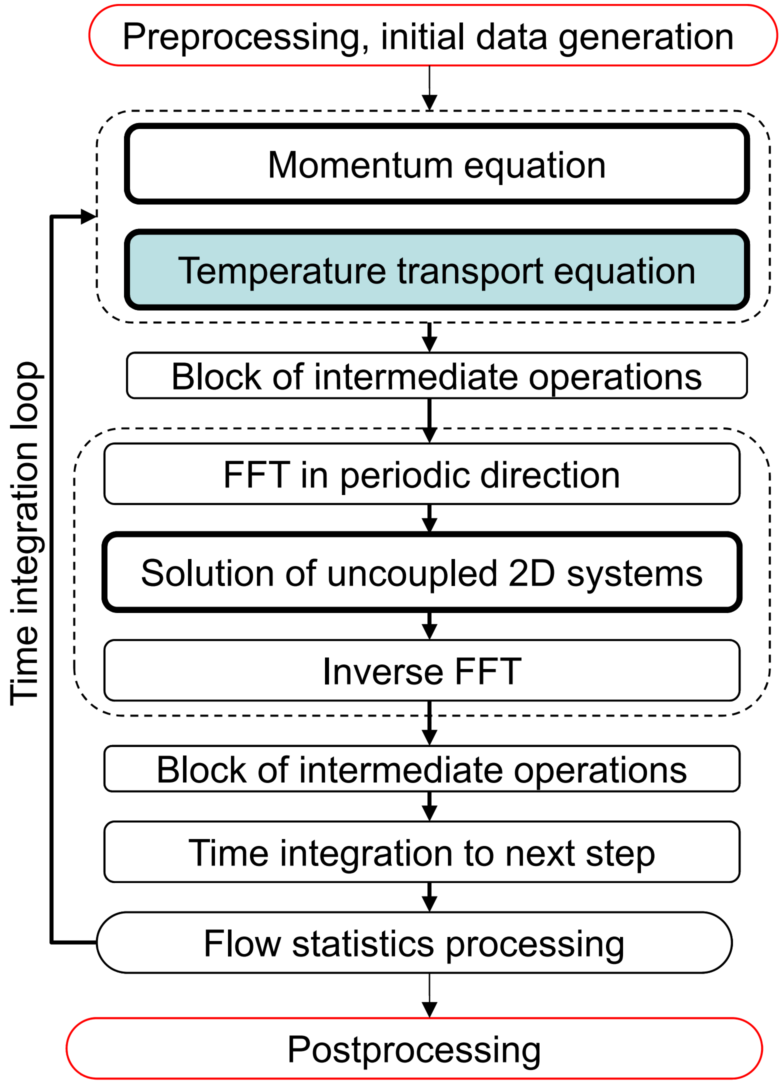
T4. **FFT, inverse IFFT**

T5. **multi-SpMV**

T6. **reduction – norm, dot product**

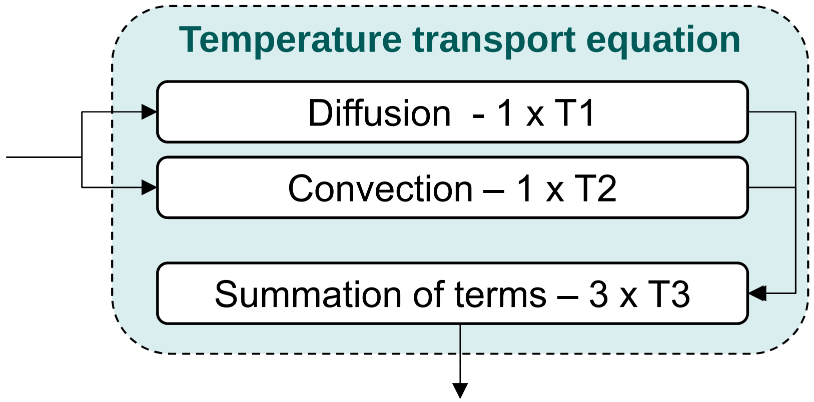


Outline of the algorithm

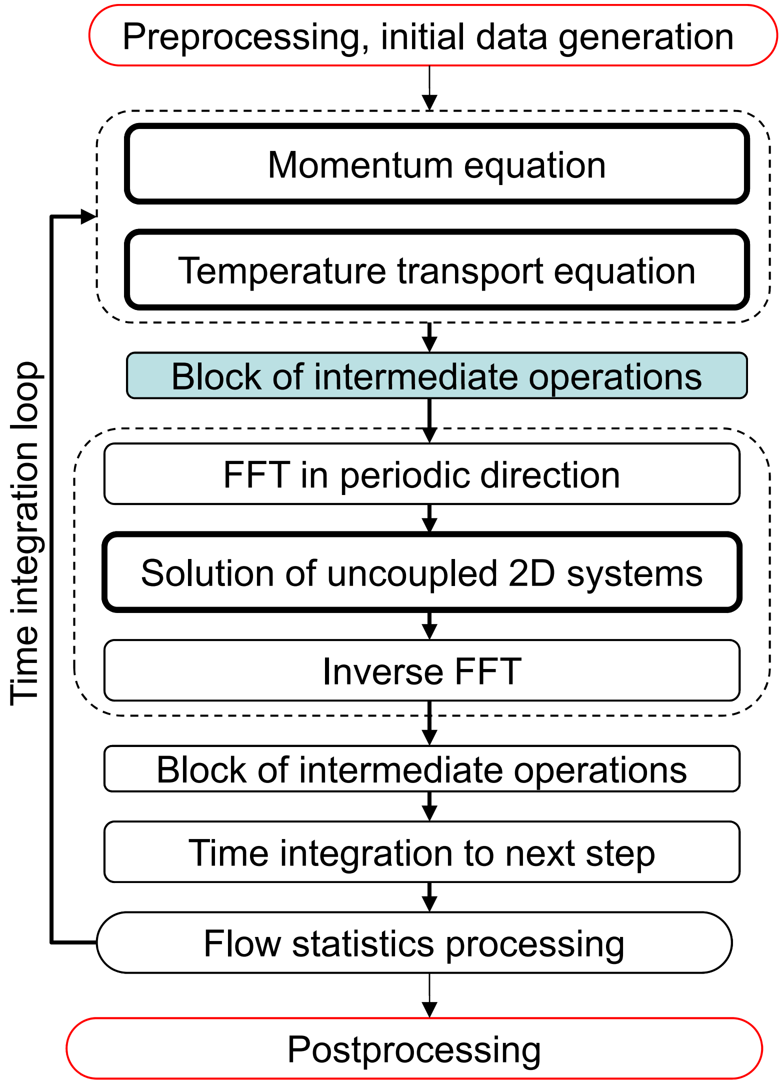


Basic operations

- **T1.** **Linear operator**
- **T2.** **Nonlinear operator**
- **T3.** $\mathbf{y} = \sum_{i=1}^{n} a_i \mathbf{x}_i + c$
- **T4.** **FFT, inverse IFFT**
- **T5.** **multi-SpMV**
- **T6.** **reduction – norm, dot product**

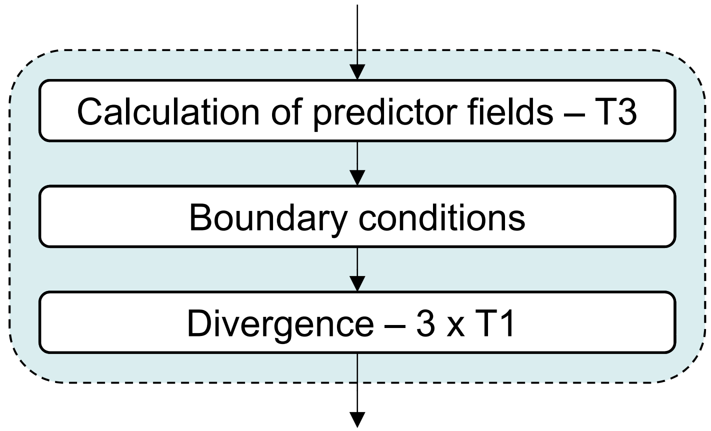


Outline of the algorithm

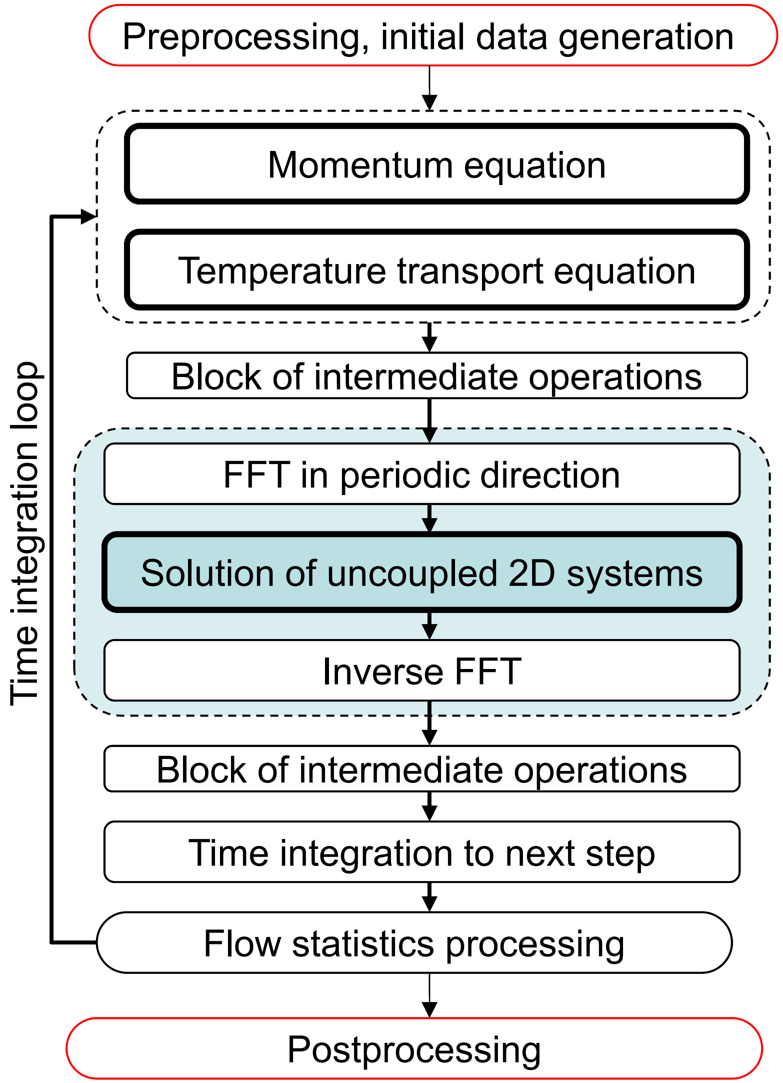


Basic operations

- **T1.** Linear operator
- **T2.** Nonlinear operator
- **T3.** $\mathbf{y} = \sum_{i=1}^{n} a_i \mathbf{x}_i + c$
- **T4.** FFT, inverse IFFT
- **T5.** multi-SpMV
- **T6.** reduction – norm, dot product

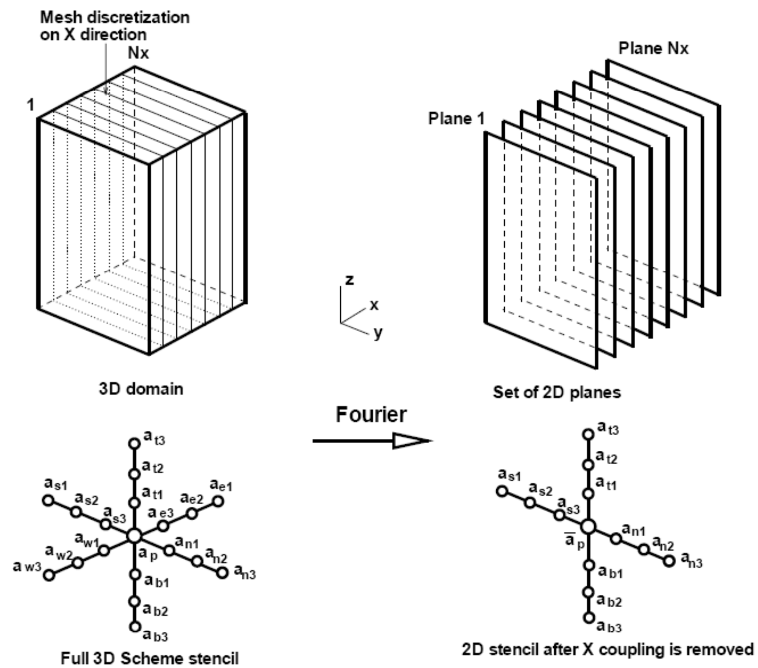


Outline of the algorithm



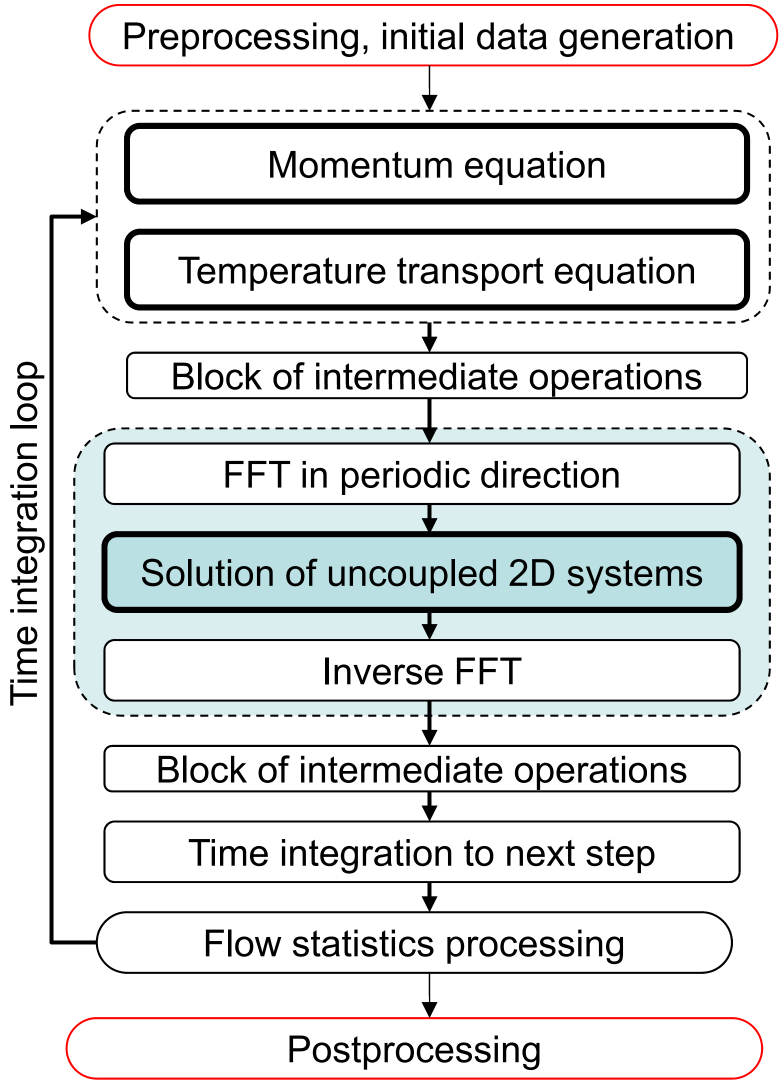
The algorithm of the solver

1. FFT diagonalization FFT uncouples 3D problem into set of independent 2D problems (planes)
2. The Schur complement based direct method is used to solve planes that correspond to lower Fourier frequencies
3. The preconditioned CG method is used to solve the remaining planes
4. Inverse FFT to restores solution of the 3D problem.



Uncoupling of a 3D domain using FFT

Outline of the algorithm



Basic operations

T1. **Linear operator**

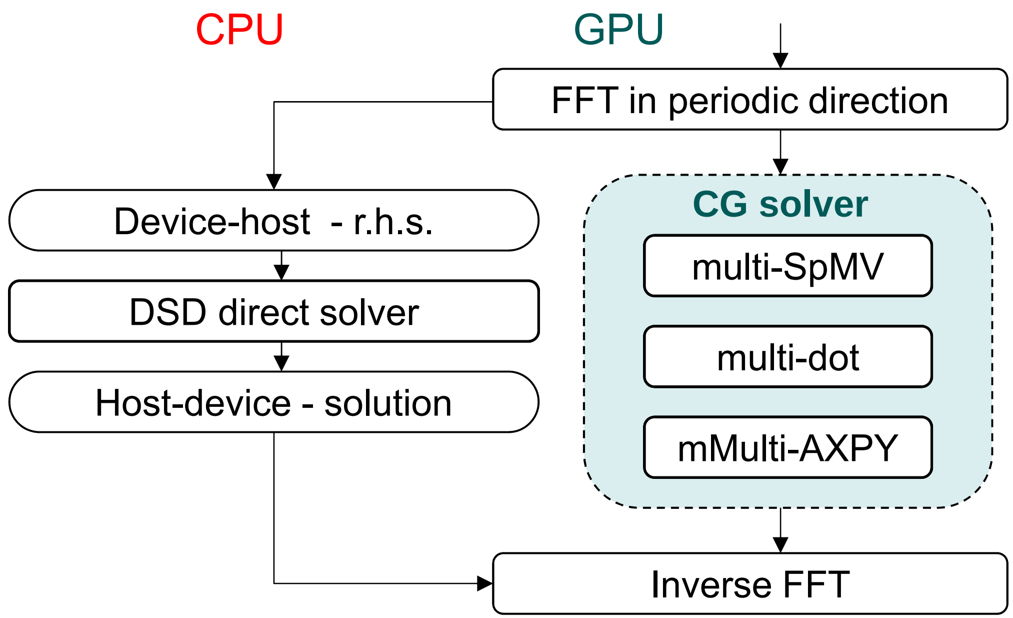
T2. **Nonlinear operator**

T3. $\mathbf{y} = \sum_{i=1}^{n} a_i \mathbf{x}_i + c$

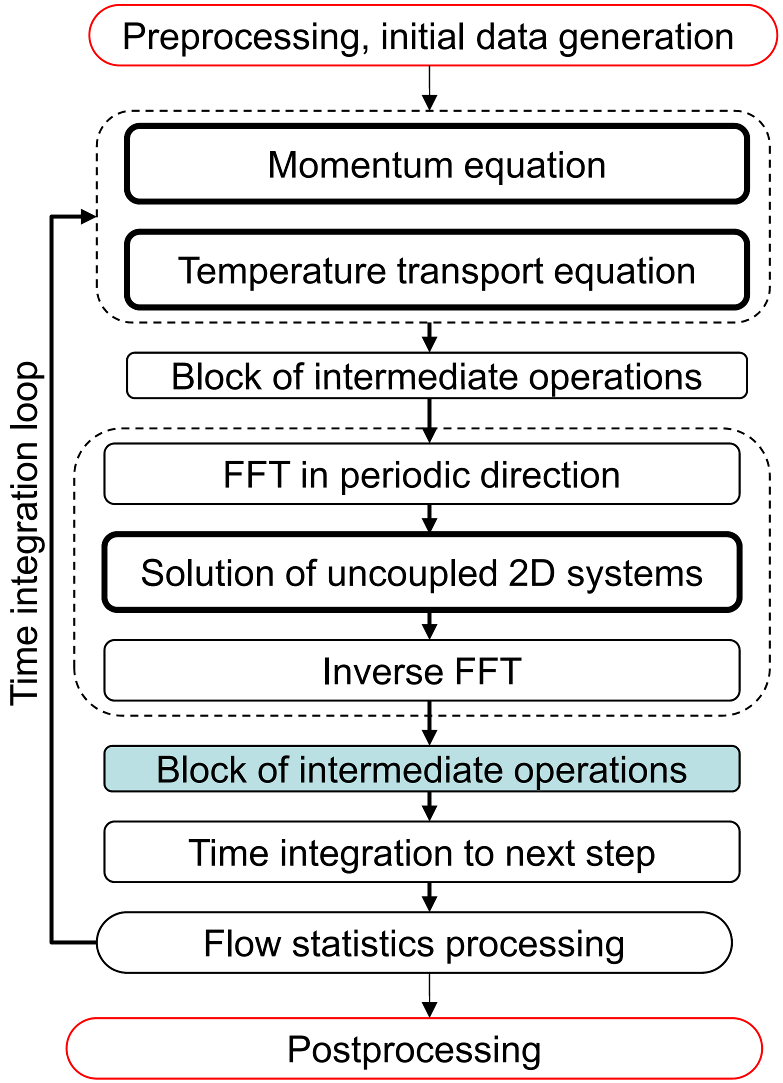
T4. **FFT, inverse IFFT**

T5. **multi-SpMV**

T6. **reduction – norm, dot product**



Outline of the algorithm



Basic operations

T1. **Linear operator**

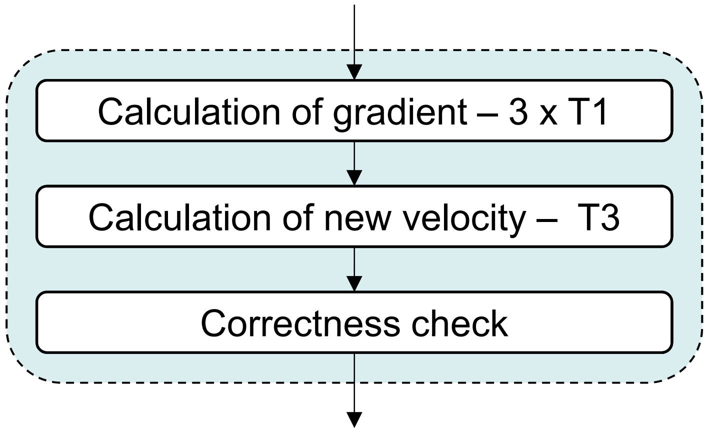
T2. **Nonlinear operator**

T3. $\mathbf{y} = \sum_{i=1}^{n} a_i \mathbf{x}_i + c$

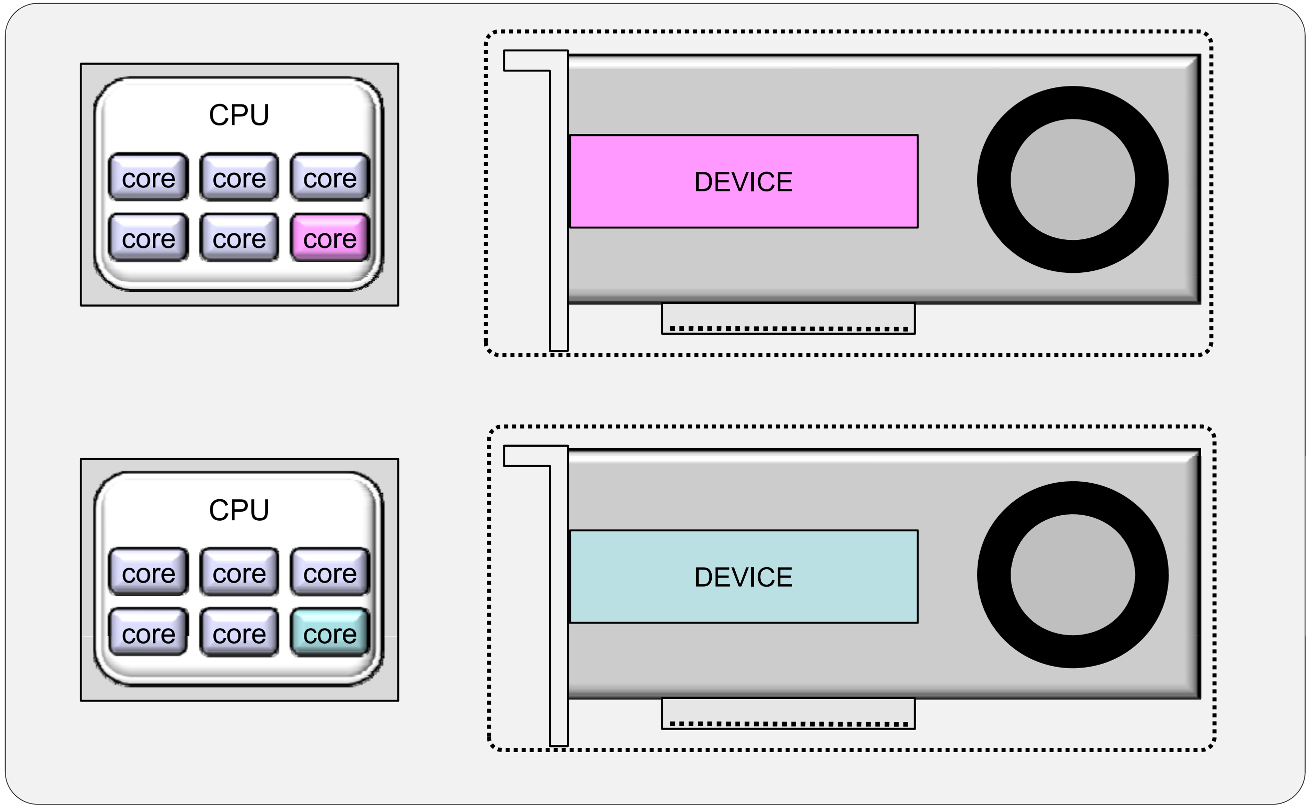
T4. **FFT, inverse IFFT**

T5. **multi-SpMV**

T6. **reduction – norm, dot product**



Loading heterogeneous node

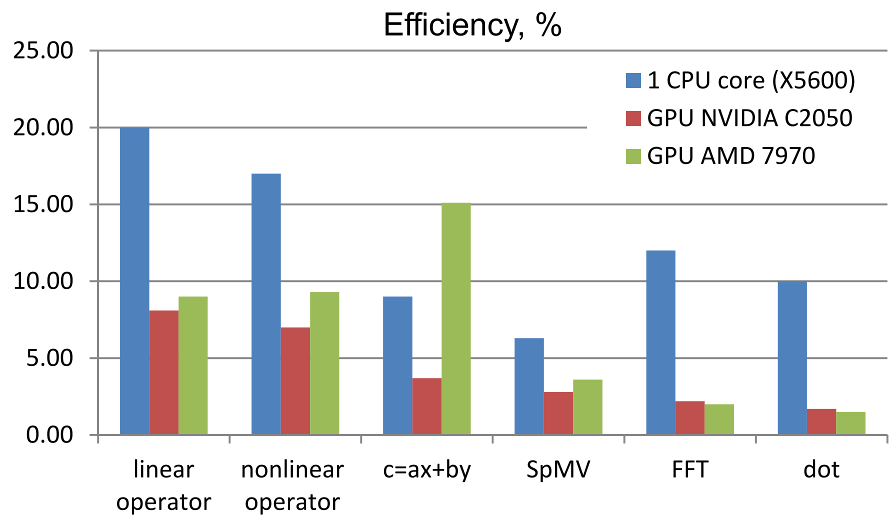


MPI process 0

MPI process 1

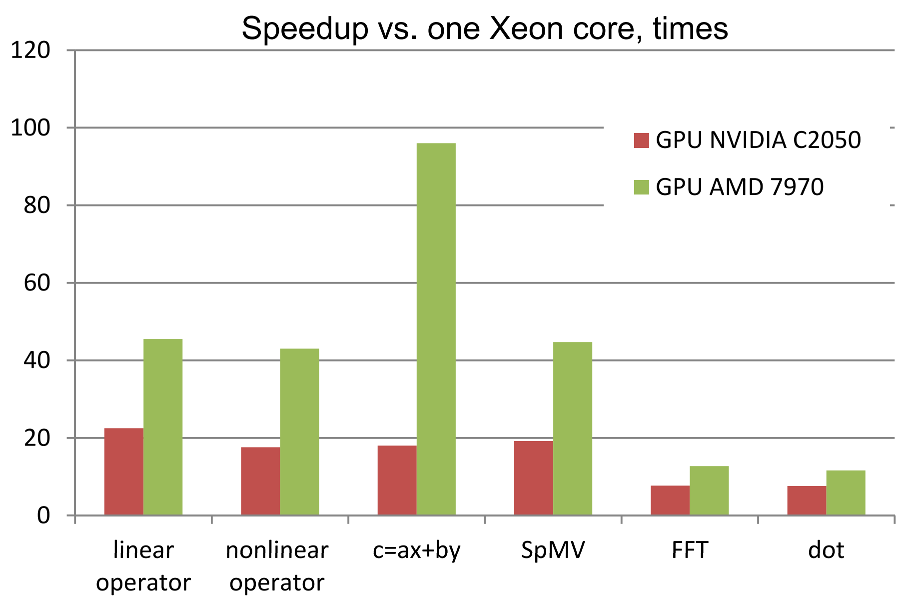
MPI process 2

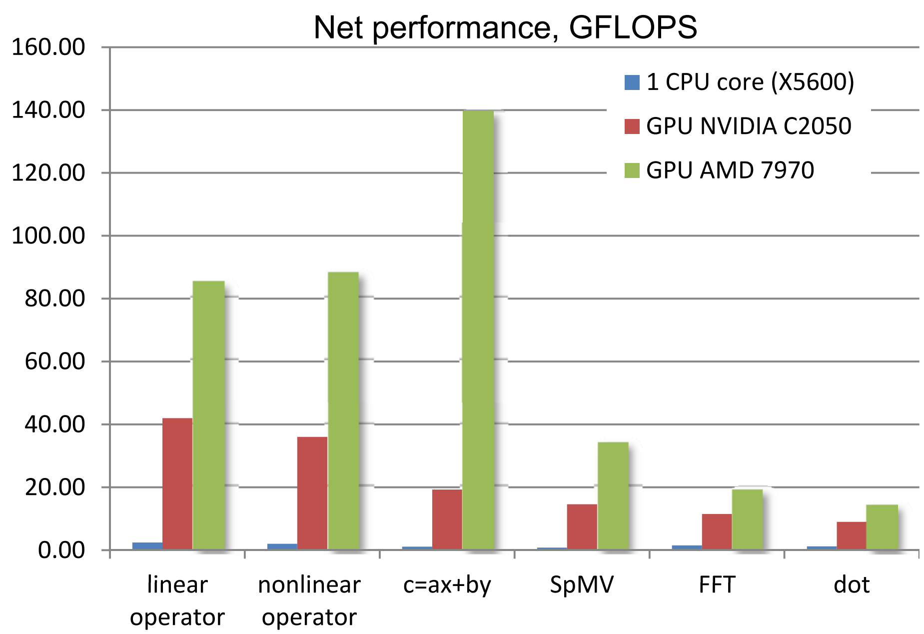
Performance of basic operations on GPU



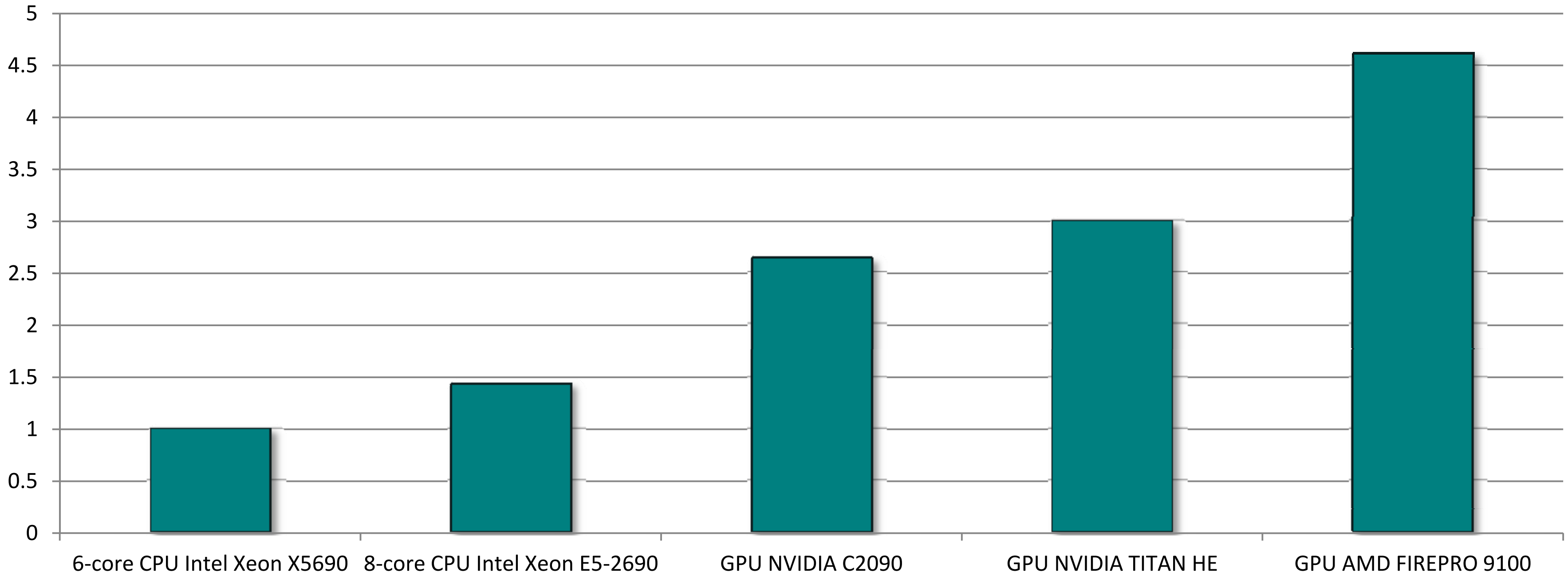
The test case

- DHC typical case is used
- 4-th order scheme
- Mesh size 1.2M nodes
- Imitates a typical computing load
- Reference: Intel Xeon X5690 core

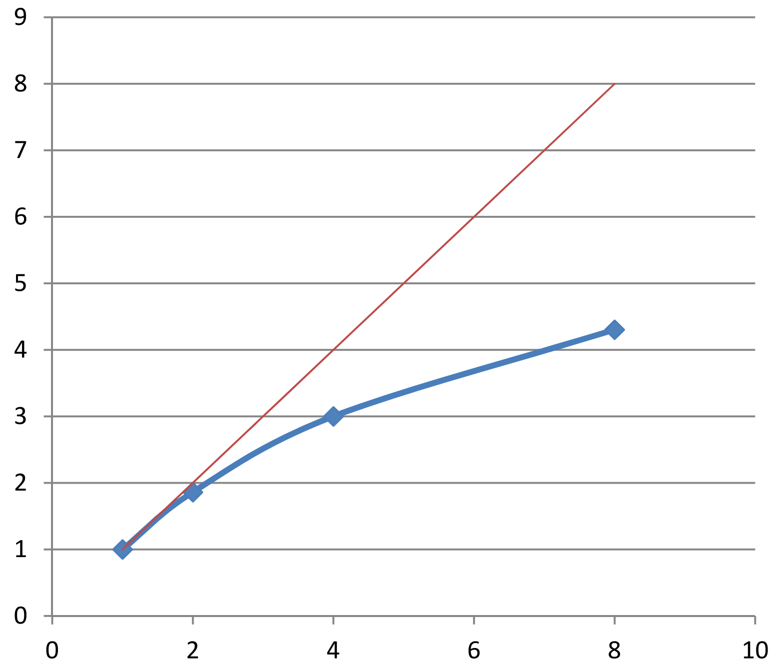


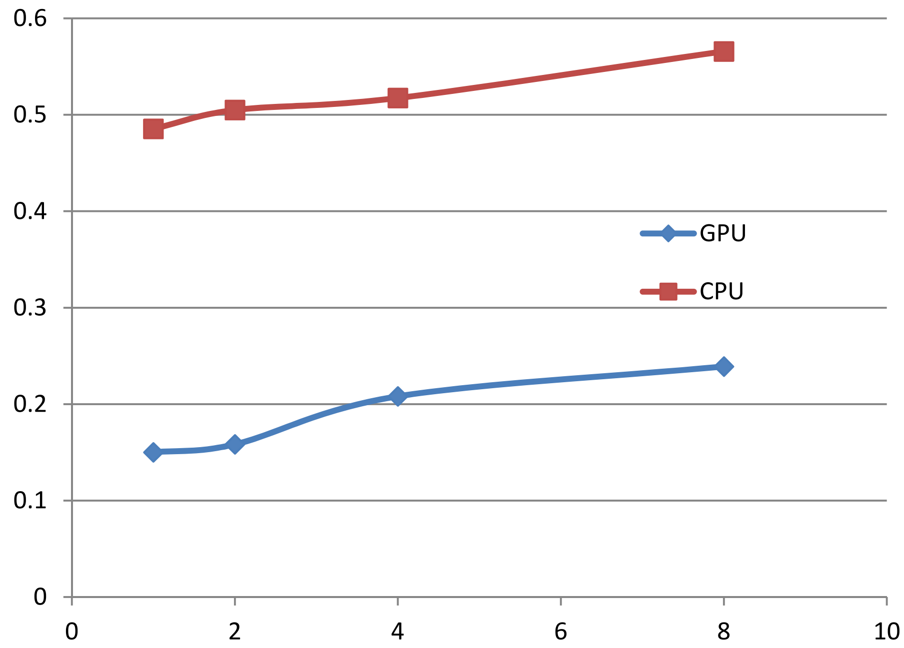


Overlap performance on GPU



Miserable (yet) speedups





Test case 2M nodes, 2-nd order scheme

Conclusions

- It is possible to obtain reasonable efficiency on modern hybrid systems with finite-volume CFD codes – compressible and incompressible, structured and unstructured
- …but it is such a torture!
- It is really a nightmare to develop, debug, maintain, modify a hybrid code
- CPU net efficiency 10-20%
  - GPU net efficiency 5-10%
    - Intel Xeon Phi net efficiency 2-3%

…

Finally will appear ExaFLOP or WhateverFLOP supercomputer
absolutely powerful
that can compute absolutely nothing!

LINPACK is the future of supercomputing...