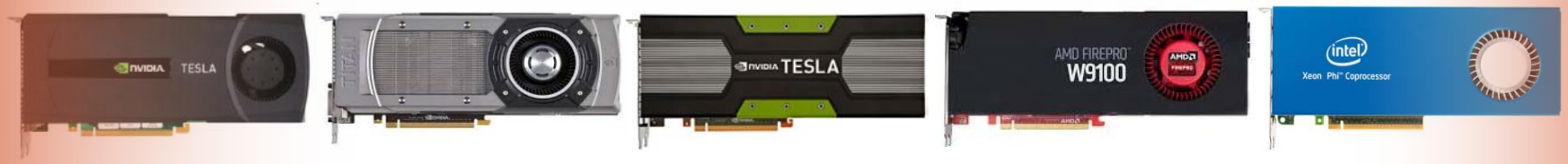
Highly portable CFD solutions for heterogeneous computing on unstructured meshes

A. V. Gorobets, S. A. Soukov, P. B. Bogdanov, X. Alvarez, F. X. Trias

Computational AeroAcoustics Laboratory
http://caa.imamod.ru
Keldysh Institute of Applied Mathematics RAS



Complicating factors

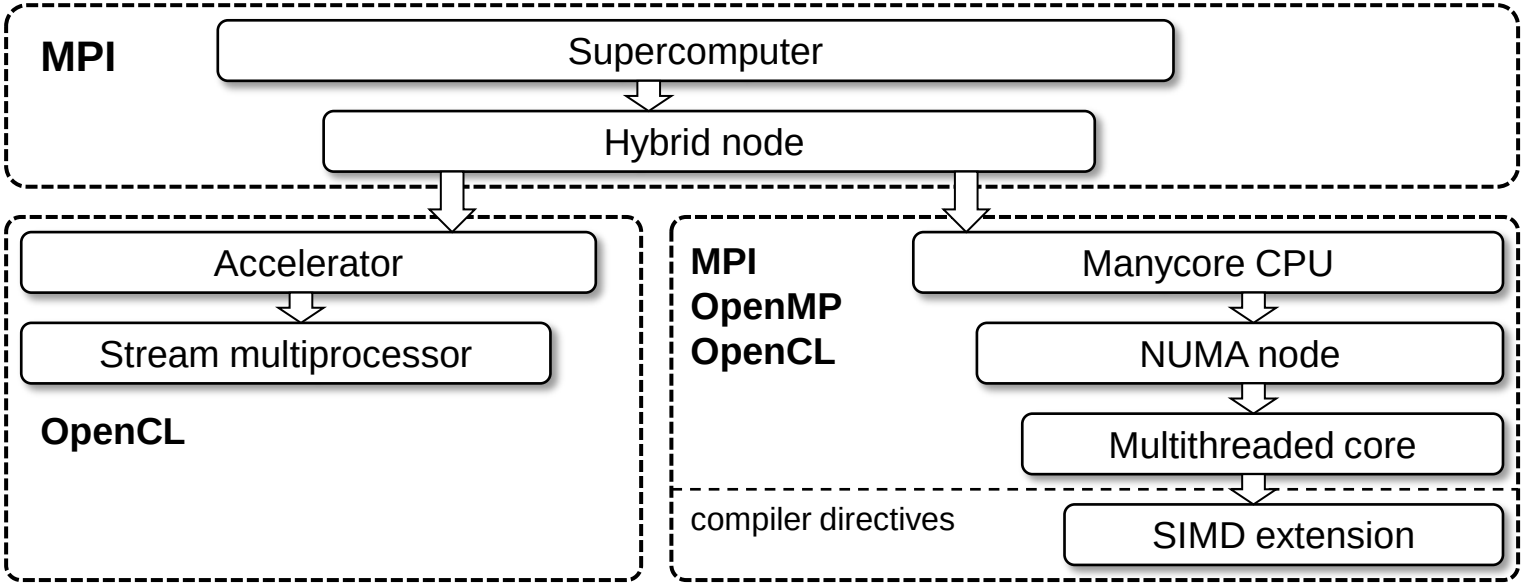
Trends in supercomputer architectures

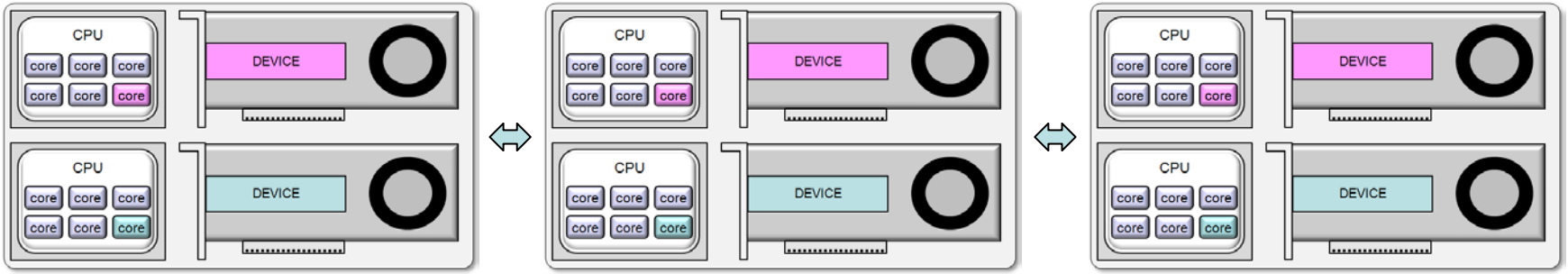
- FLOPS to memory bandwidth ratio grows fast (8180 – x2 GB/s vs x4 FLOPS)
- SIMD and stream processing domination
- Hybridization of computing devices and memory
- Architecture-specific frameworks
- Traditional increase in number of cores per device, devices per node, nodes per system

Our applications summary

- Unstructured meshes – irregular memory access
- Low computing density (0.25 ~ 2 FLOP per FP64 value)
- Non-uniform operations per cell in many cases
- 1-20KB per cell, 1.5-30 KFLOP per cell per time step
- Simulation cost 100K – 30M CPU hours

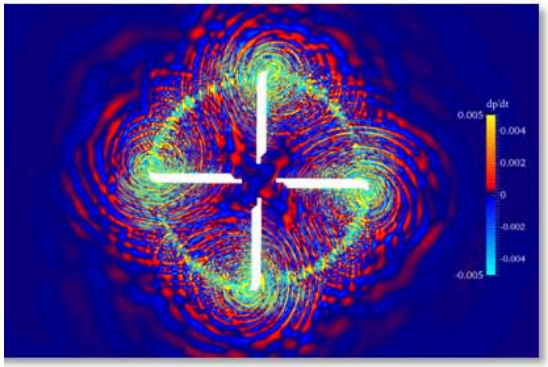
Multilevel parallelism

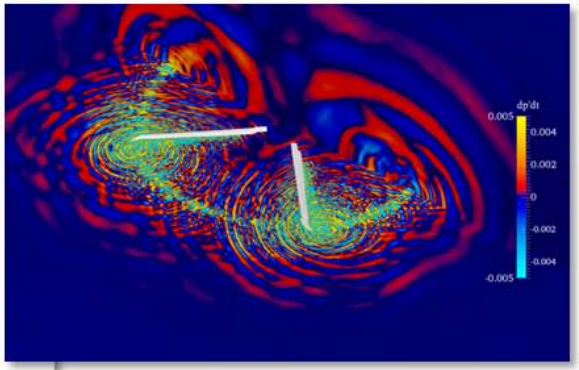


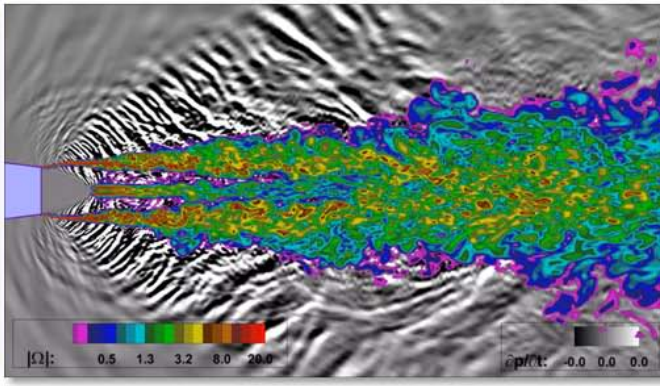


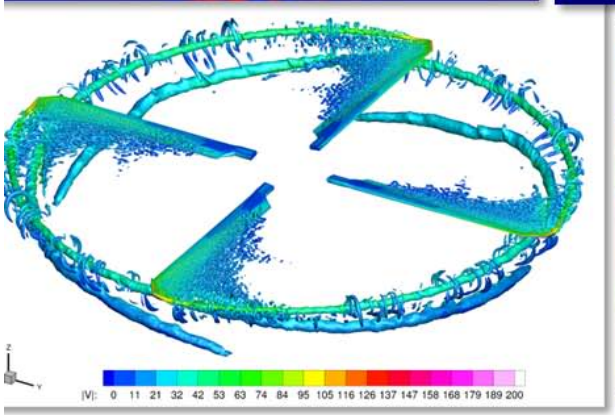
Devices: manycore CPU, GPU of AMD and NVIDIA, OpenCL-compatible FPGA, MIC

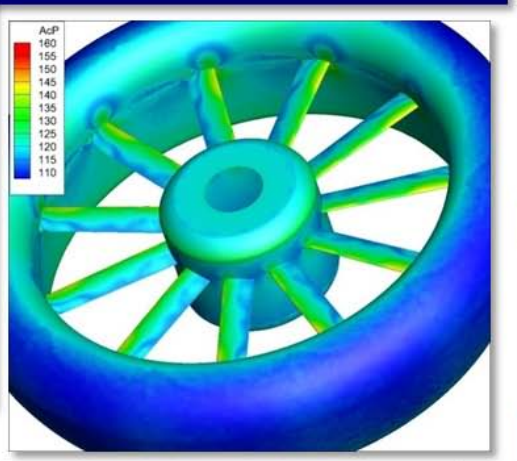
Our application area: CFD and CAA

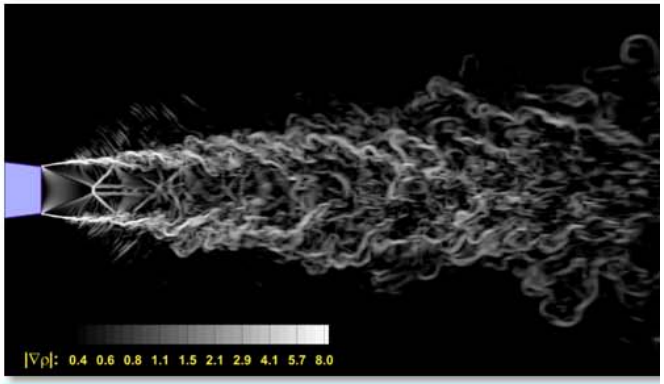


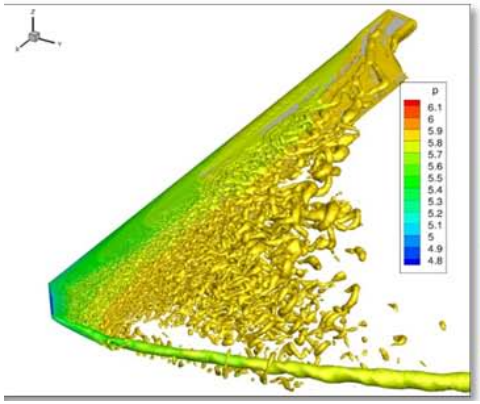


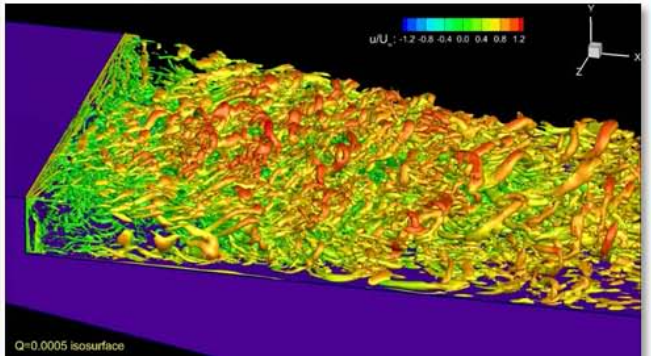


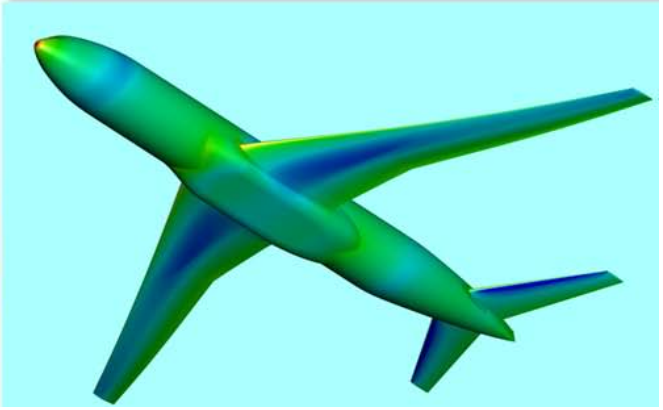








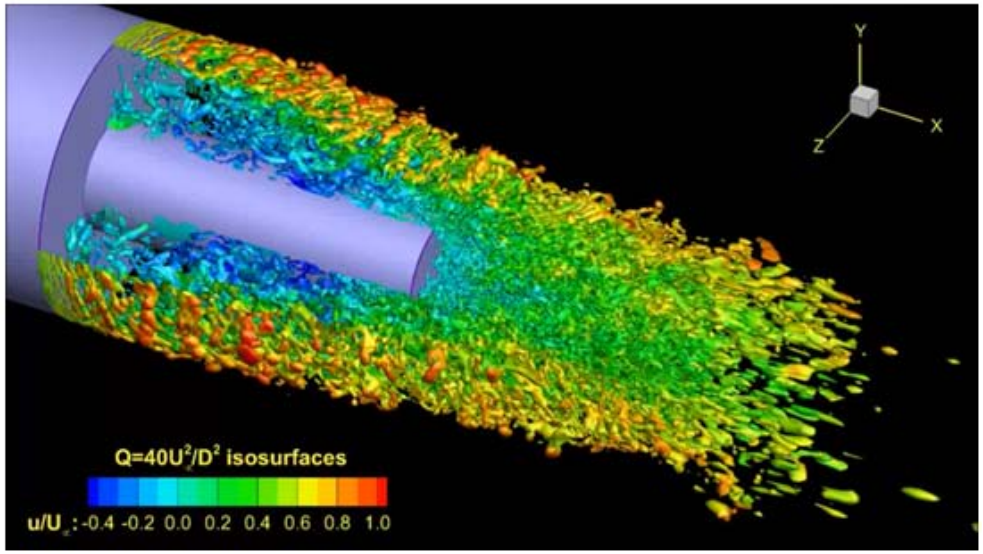


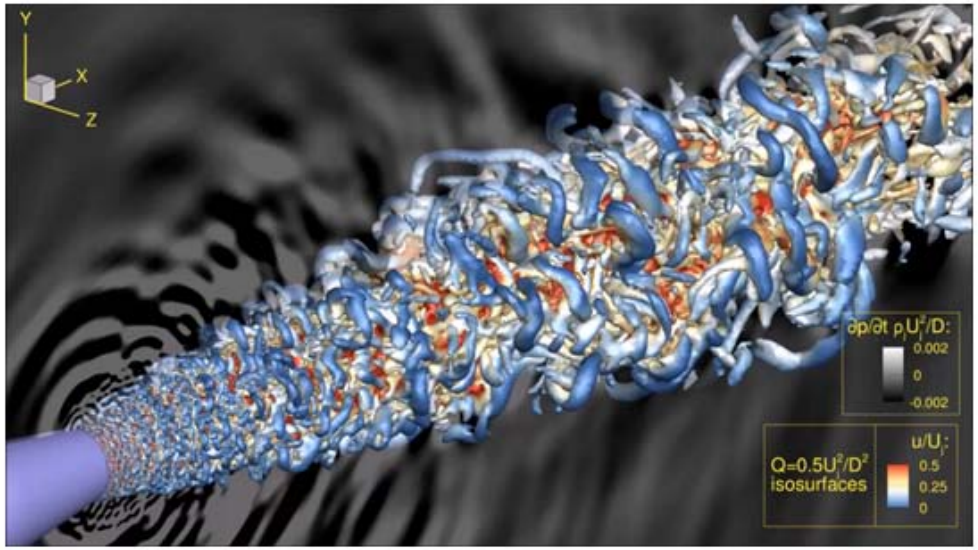


Requirements for portable supercomputing CFD solutions

Applications:
time-accurate simulations of turbulent flows

- **Portability across various architectures**
- **Full compatibility with SIMD and stream processing**
- **Scalability to thousands of devices**
- **Multilevel parallelism**





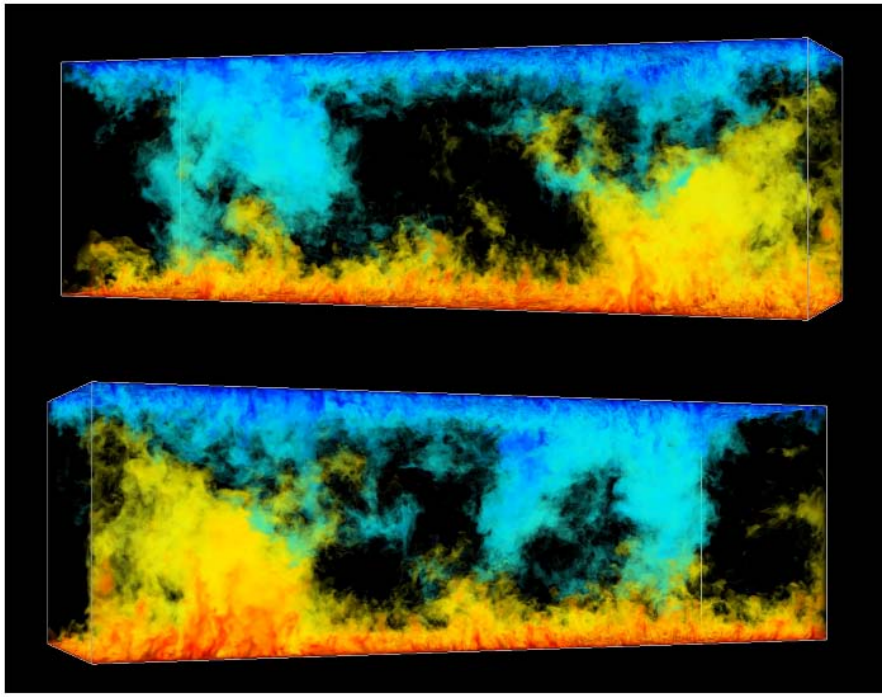
Time-accurate simulations of turbulent flows

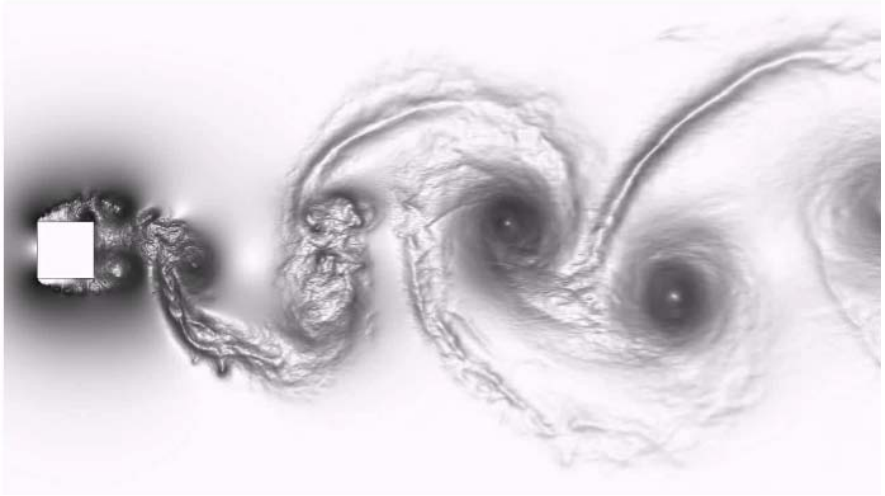
- **Complexity and variety of architectures need simple algorithms**

- **Fine resolution in space and time**
  ⇨ **high computing demands**

- **Small time step required**
  ⇨ **explicit schemes or**
  ⇨ **implicit schemes with simple solvers**
  stream processing compatible KSM, MG

Simple, portable and efficient algorithms for turbulent flows





Tapir code: CFD playground for heterogeneous computing

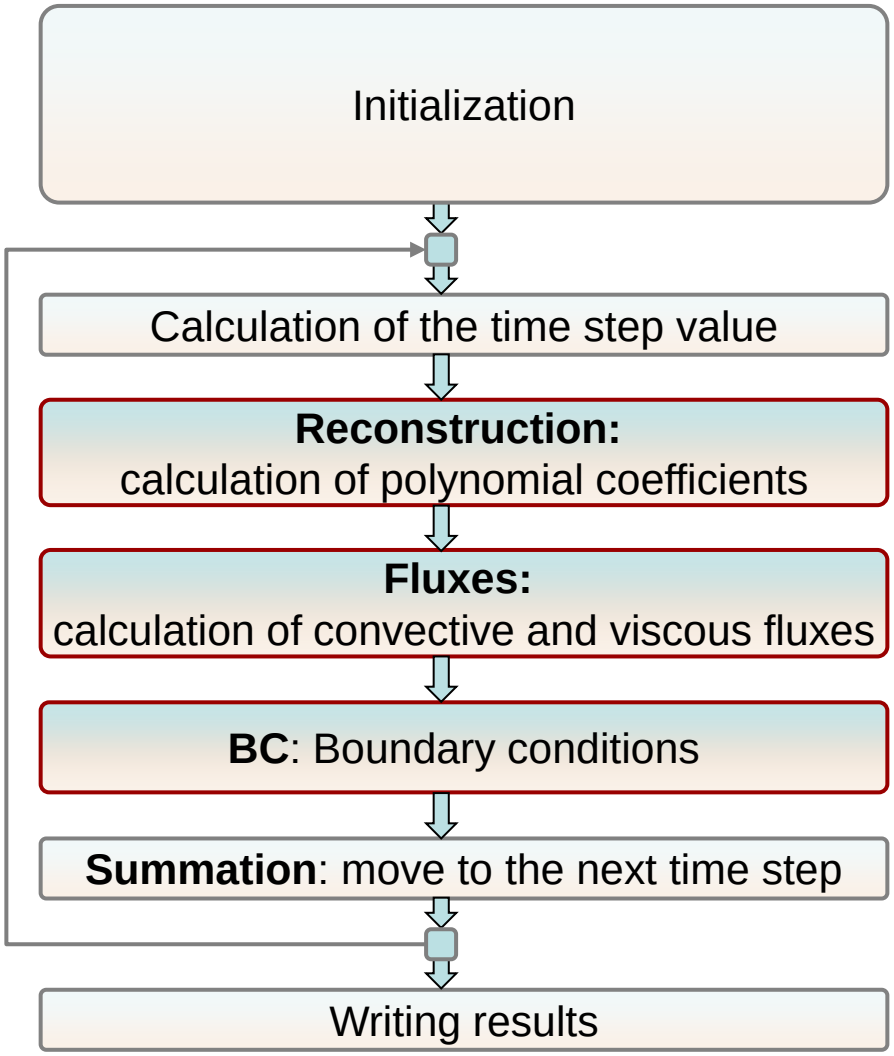
- **Compressible Navier-Stokes equations**

$$\frac{\partial \mathbf{Q}}{\partial t} + \frac{\partial \mathbf{F}(\mathbf{Q})}{\partial x} + \frac{\partial \mathbf{G}(\mathbf{Q})}{\partial y} + \frac{\partial \mathbf{H}(\mathbf{Q})}{\partial z} = \frac{1}{Re}\left(\frac{\partial \mathbf{F}_\nu(\mathbf{Q})}{\partial x} + \frac{\partial \mathbf{G}_\nu(\mathbf{Q})}{\partial y} + \frac{\partial \mathbf{H}_\nu(\mathbf{Q})}{\partial z}\right)$$

Q – vector of conservative variables;
F, G, H – vectors of conservative fluxes;
$\mathbf{F}_\nu$, $\mathbf{G}_\nu$, $\mathbf{H}_\nu$ – vectors of dissipative fluxes, *Re* – Reynolds number.

- **Cell-centered finite-volume method for unstructured hybrid**
- **High-order polynomial-based schemes (2 ~ 3 order)**
- **Various flow solvers depending on a problem (Roe, Rusanov, HLLE, HLLC, …)**
- **MPI + OpenMP + OpenCL parallelization**
  - total portability
  - communication hiding (overlap)
  - workgroup size automatic tuning
  - load balancing
  - RCM reordering

Algorithm outline



Parallel mesh partitioning
Geometric init
Data and kernels init

Loop over **elements**

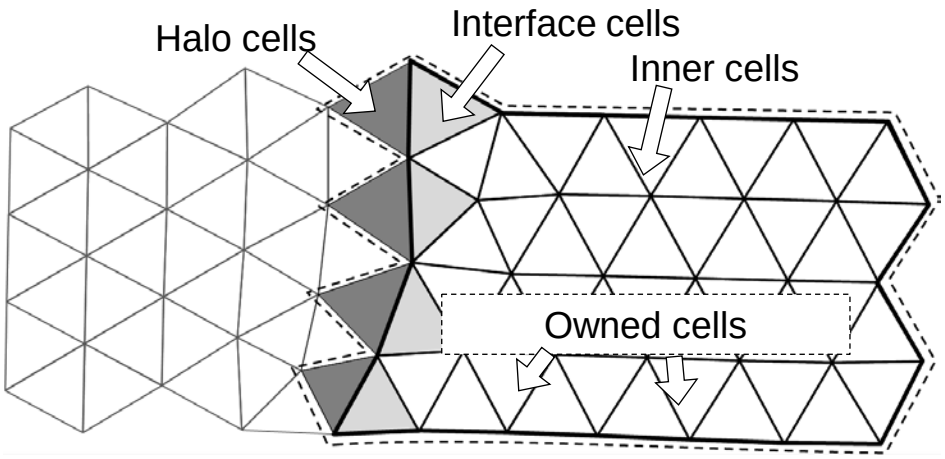
Loop over **elements**

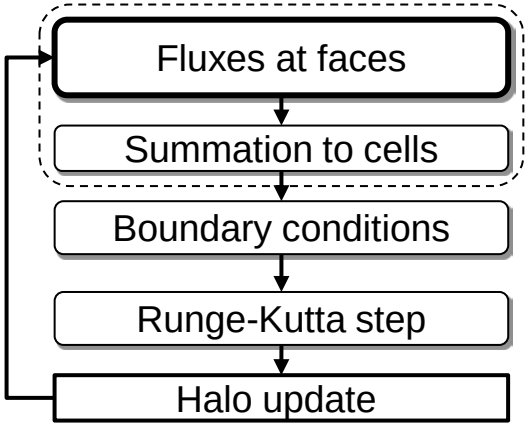
Loop over **faces**

Loop over **BC faces**

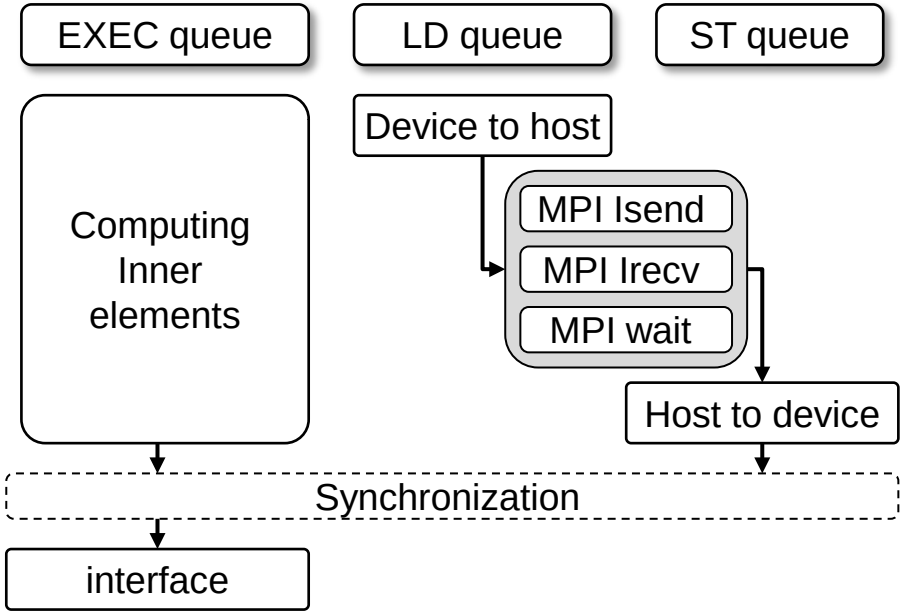
Loop over **elements**

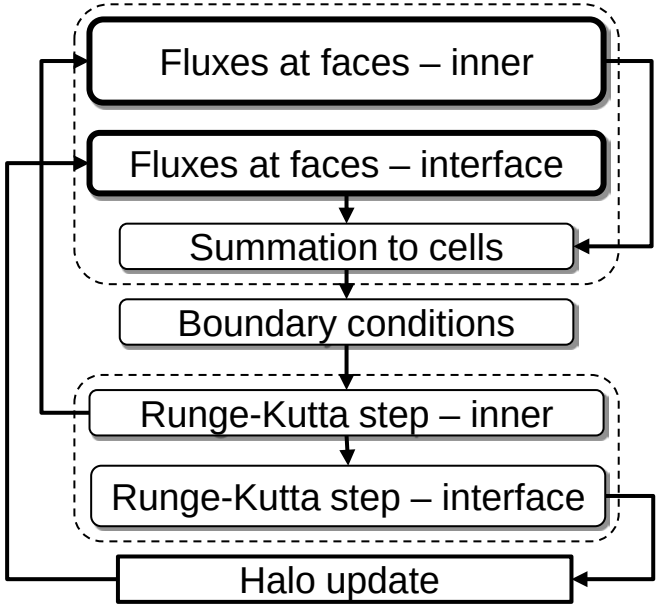
Overlap mode for communication hiding





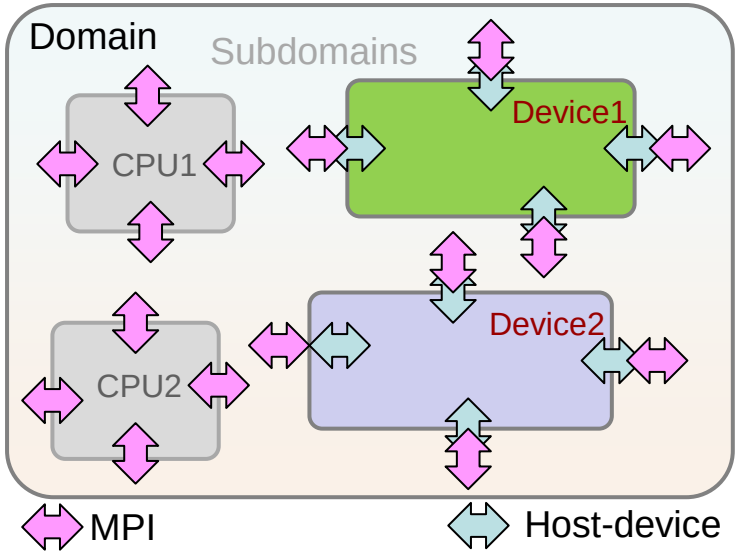
Overlapping computations and data transfer



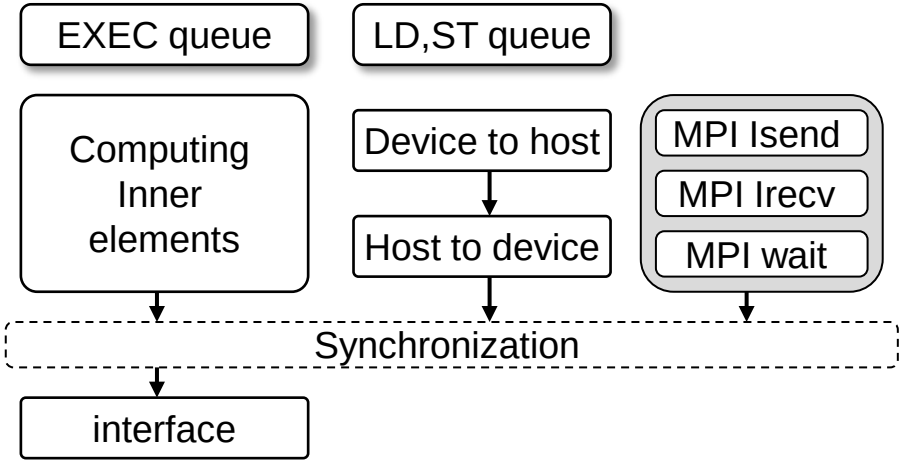


Two-level partitioning with communication hiding

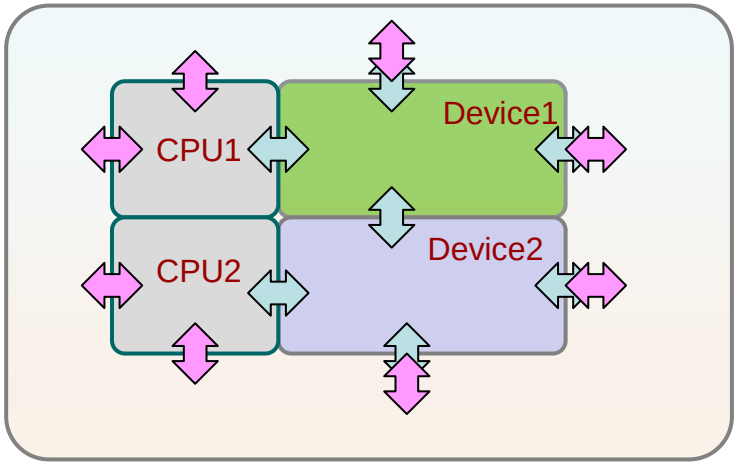
1-Level partition: MPI process per device



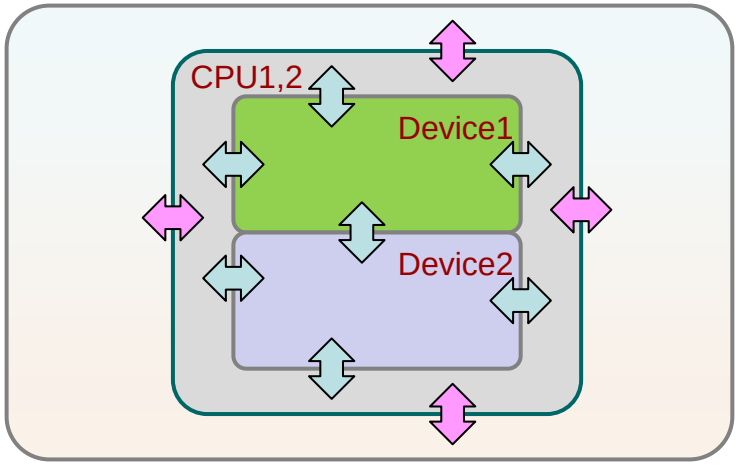
Double overlap: computing, host-device, MPI



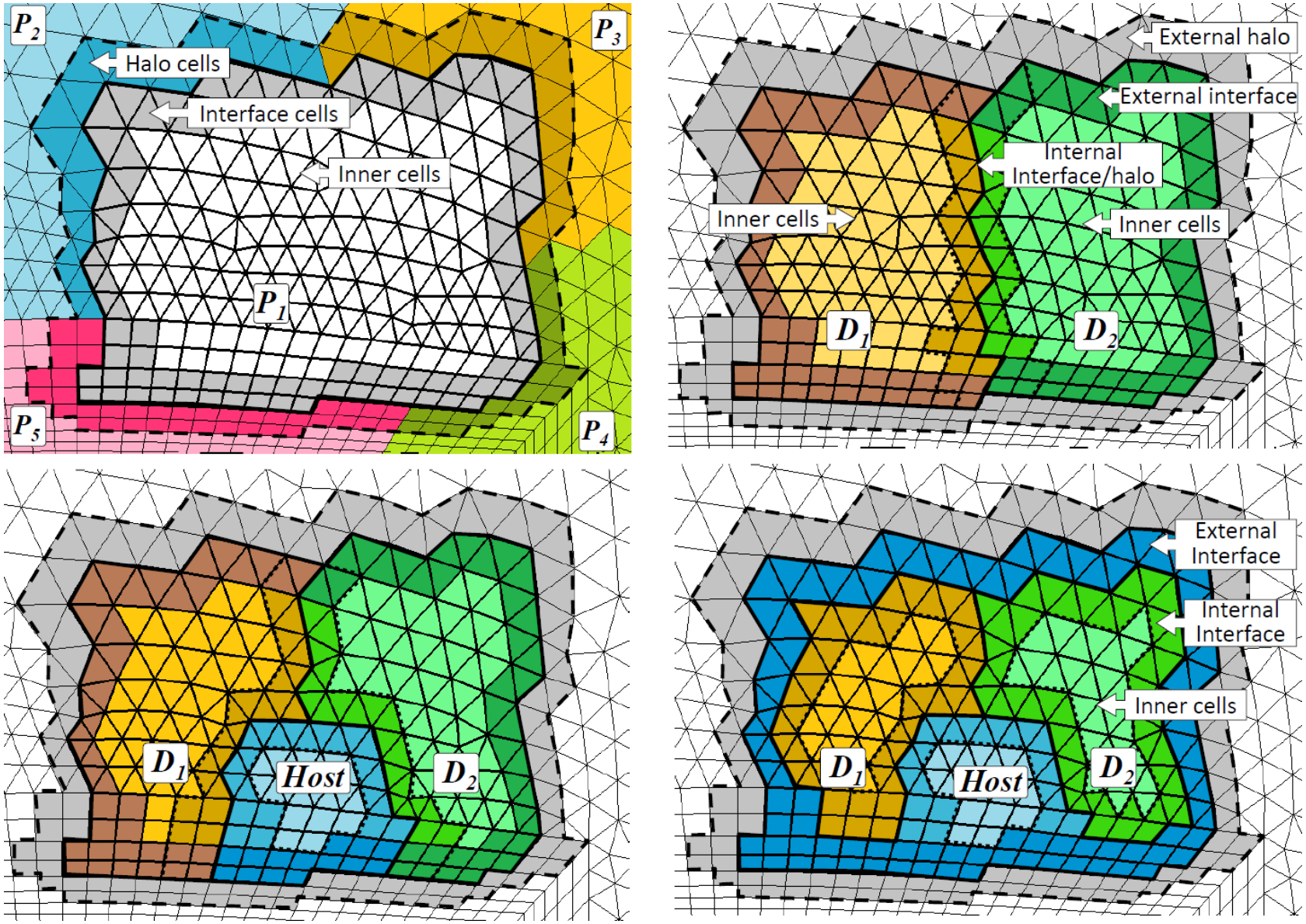
2-Level partition – naive



2-Level partition – **double overlap**



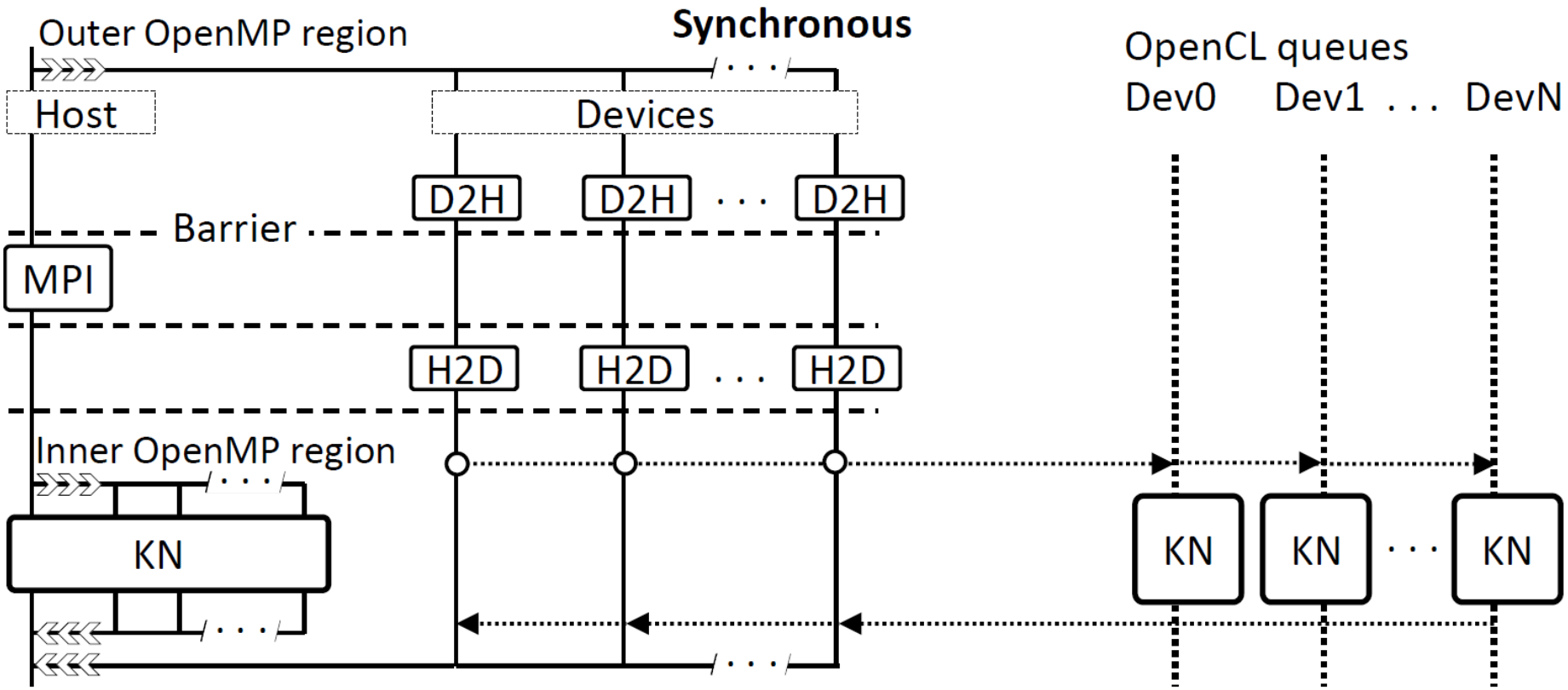
Two-level partitioning



Overlapped 2-level halo update

- **Device-to-host (D2H)**
  - data to be sent from interface cells are copied inside devices from a mesh functions array into an intermediate send buffer
  - send buffers from all devices are transferred to host
  - data from send buffers for inter-node transfer are packed into send messages
- **MPI exchanges**
  - the send messages are posted with non-blocking MPI_Isend calls
  - MPI_Irecv calls are posted to get incoming halo
  - MPI_Waitall ensures MPI exchanges have finished
- **Host-to-device (H2D)**
  - received messages are unpacked to device receive buffers together with internal halo
  - receive buffers are transferred to devices
  - inside devices receive buffers are copied to the mesh functions array

Heterogeneous execution modes

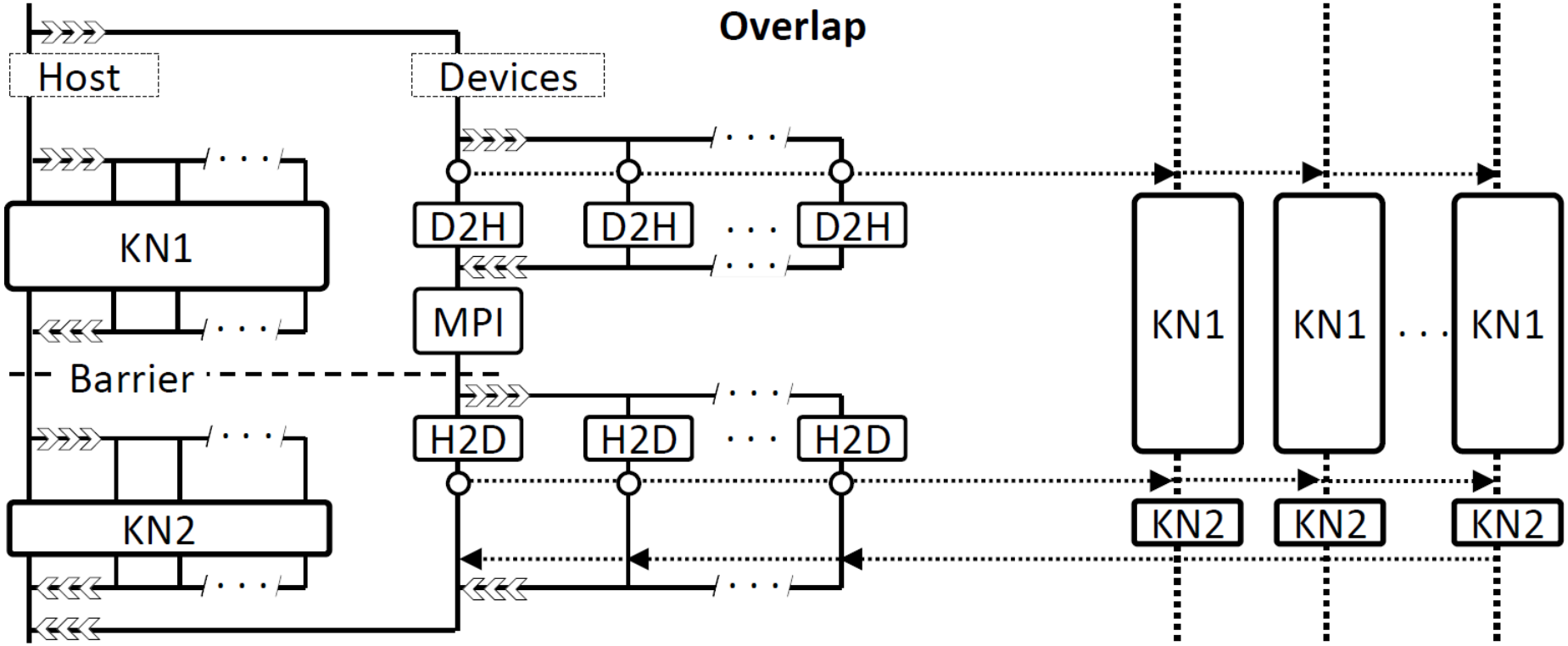


$$t_{sync} = t_{HD} + t_{MPI} + max(N^{d_k})C_D$$

$$t_{HD} = T_{HD}(max(N_I^{d_k}) + max(N_H^{d_k}))$$

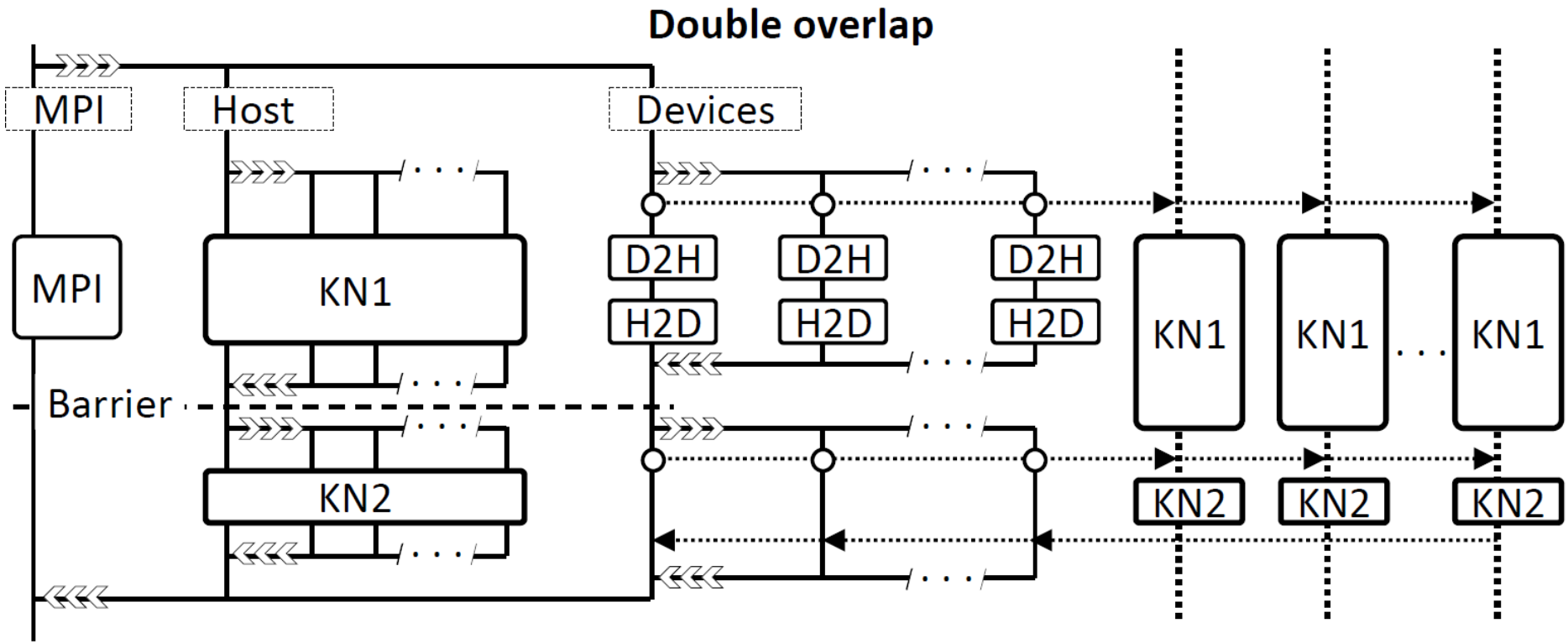
$$t_{MPI} = T_{MPI}(N_I + N_H)$$

Heterogeneous execution modes



$$t_{over} = max(t_{HD} + t_{MPI}, max(N_L^{d_k})C_D) + max(N_I^{d_k})C_D$$

Heterogeneous execution modes



$$t_{dover} = max(max(t_{HD}, t_{MPI}), max(N_L^{d_k})C_D) + (N_I^h)C_H$$

OpenMP implementation

- **OpenMP loop-based parallelization with dynamic schedule:**
  #pragma omp parallel for schedule (**dynamic**, 200)

- **On Intel CPUs performance is nearly identical to OpenCL**

- **Magic "#pragma ivdep" for vectorization**

Mesh 445K cells, 1.1M faces

Intel Xeon Device	N cores	Speedup	Parallel Efficiency
X5670 (Gulftown)	6	5.05	84%
E5-2690 (Sandy Bridge)	8	7.4	92%
E5-2697v3 (Haswell)	14	11.6	83%
Platinum 8160 (Skylake)	24	21	87%
Phi SE10X (Knights Corner)	60/240	99	
Phi 7290 (Knights Landing)	72/280	78	

OpenCL and CUDA implementation

- **Global group sizes**
  Reconstruction: 16 x NE, NE is the number of elements
  each thread computes 1 of 15 coefficients
  Fluxes: NF, the number of faces
  each thread computes flux through one face

- **Local workgroup sizes**
  automatic tuning at initialization, divisible by 16

- **Cuthill–McKee reordering**

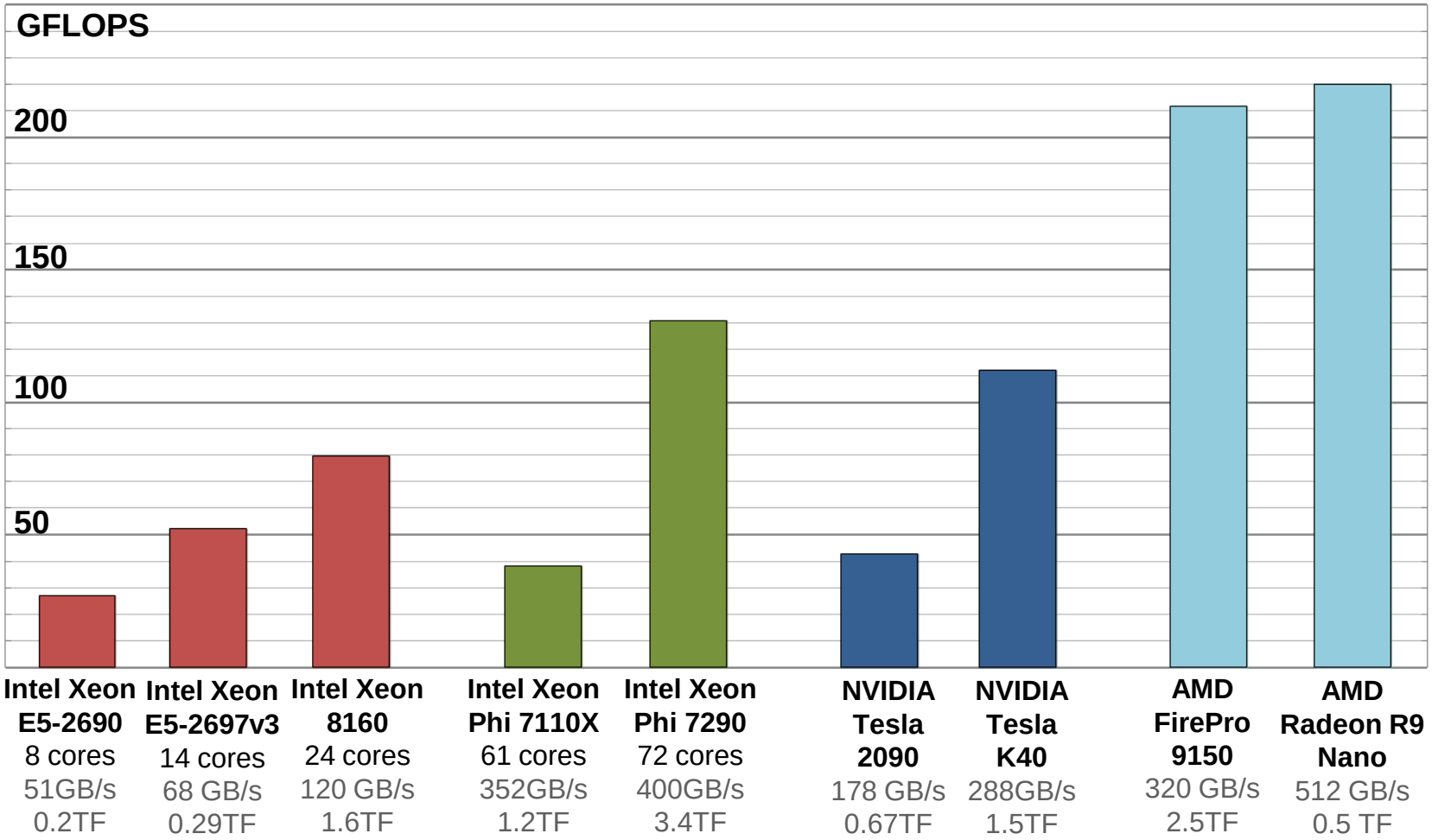
- **Data organization:**
  1) AOS - threads read their blocks (no coalescing)
  2) SOA - threads read their positions in 1D arrays (coalescing)
  3) AOS+LDS - transposed reading to local data storage with a local barrier; data as AOS, coalescing as SOA

SOA and AOS+LDS have equal performance and outperform AOS 20-40%

On NVIDIA 2050 the gain is 37%, on NVIDIA Titan it is 24%

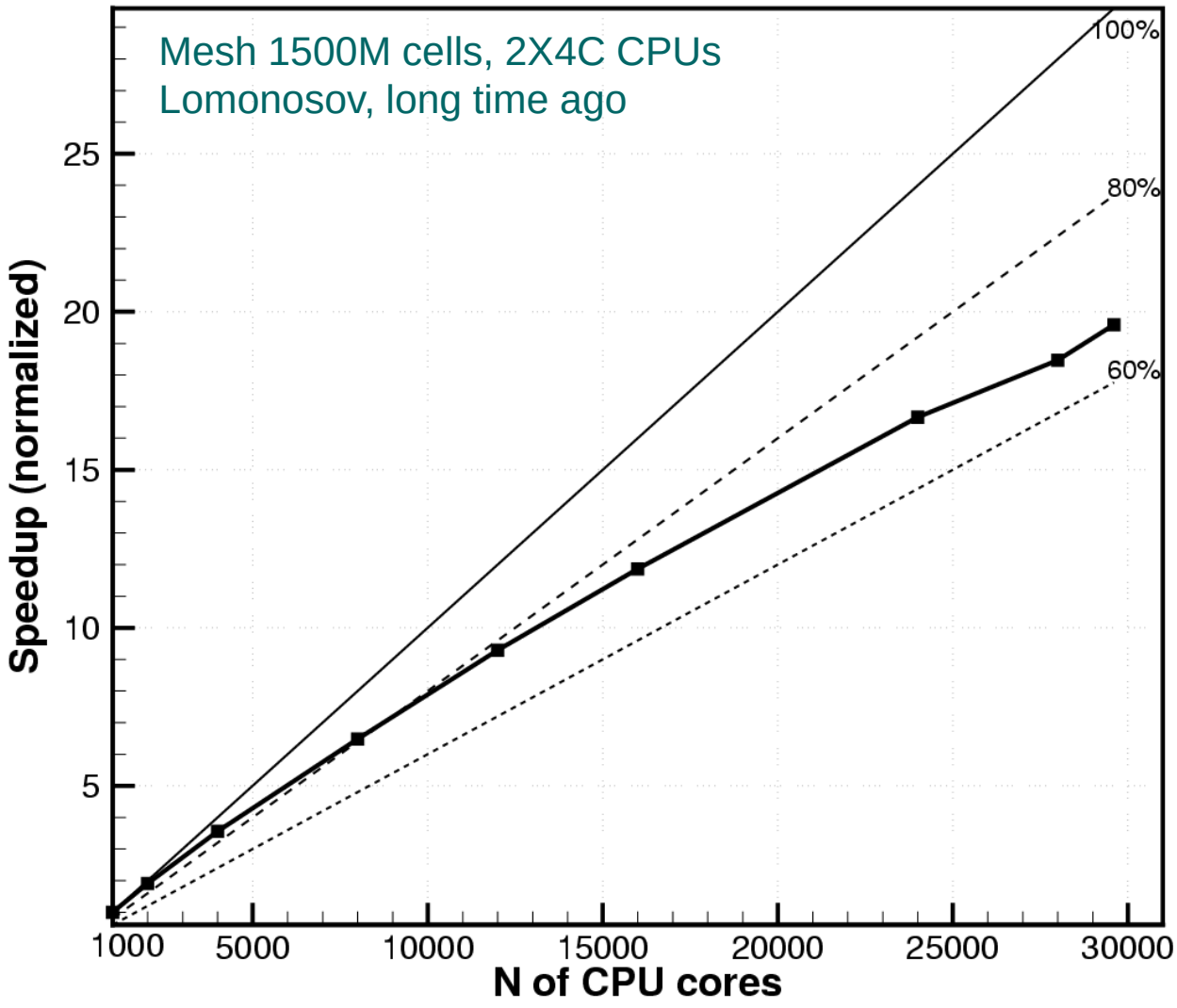
⇒ new models are more tolerant to naive memory access

Single device performance



Mesh 445K cells. 1 time step takes from 90 ms on X5670 to 5 ms on AMD Nano
computing density ~2 FLOP per byte, ~2.5 KFLOP per cell, ~1.1KB per cell

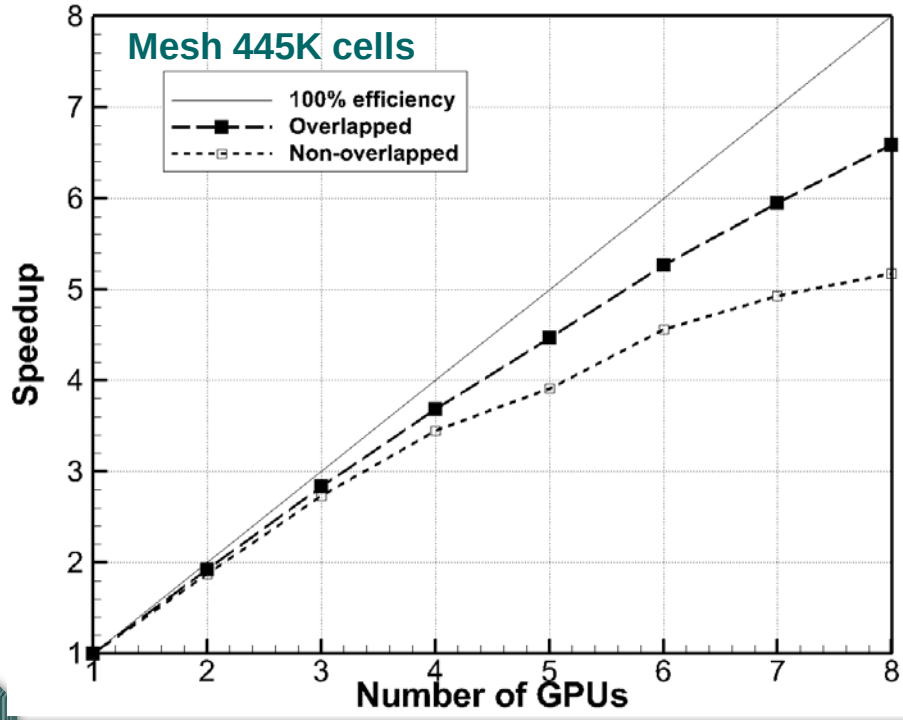
Baseline MPI+OpenMP CPU code

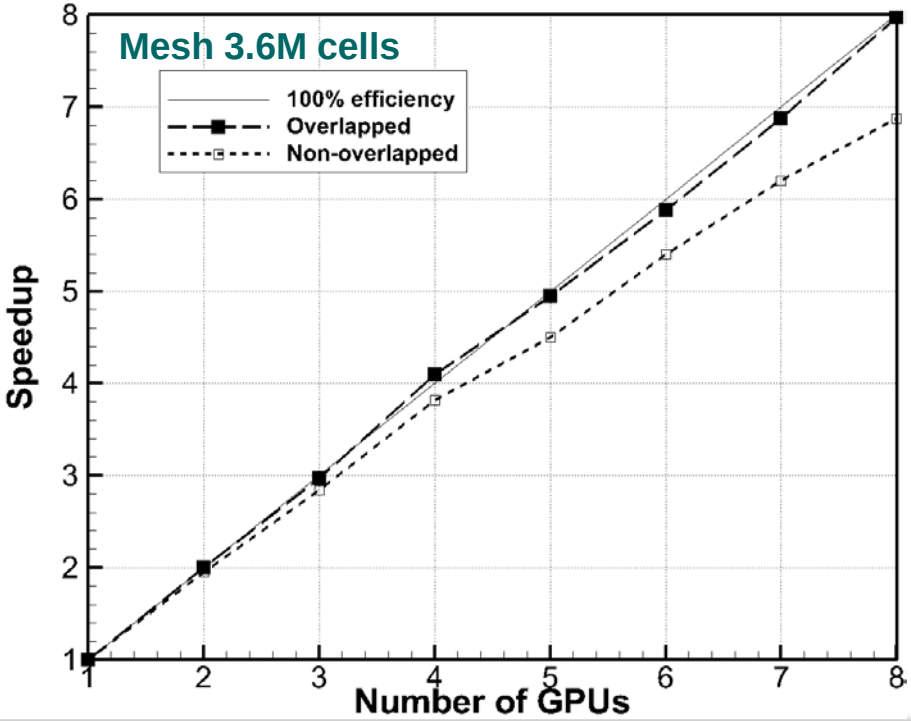


Multi-GPU single node

“Fat node” with 8 GPU

- **NVIDIA GTX Titan**
  288GB/s, 1.5TF



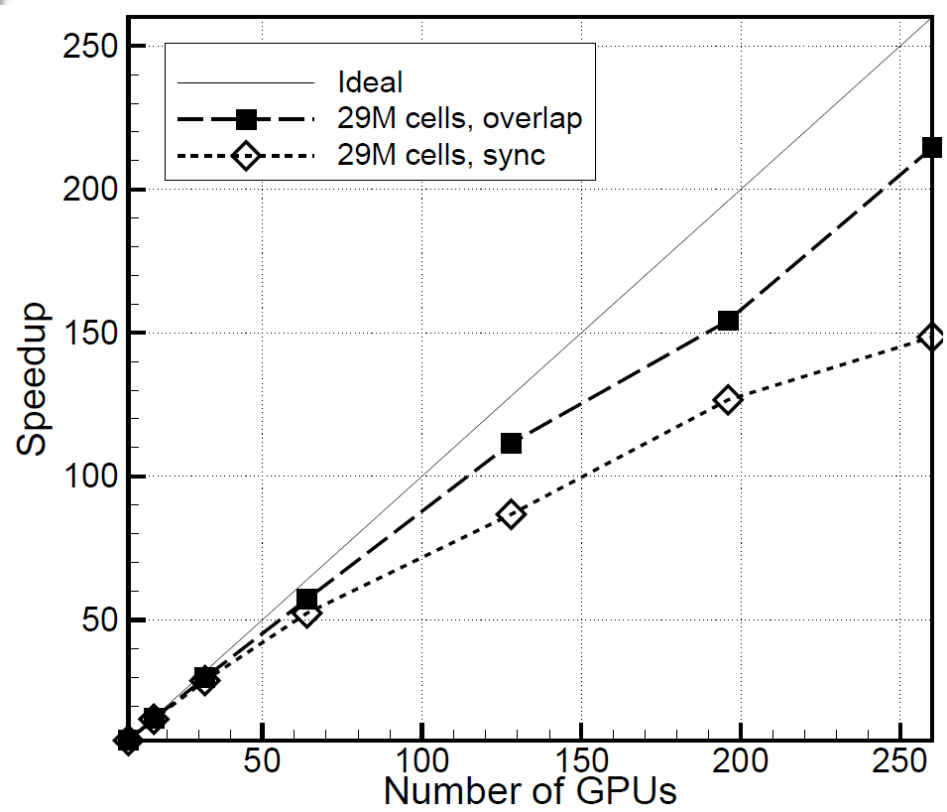


Multi-GPU hybrid supercomputer

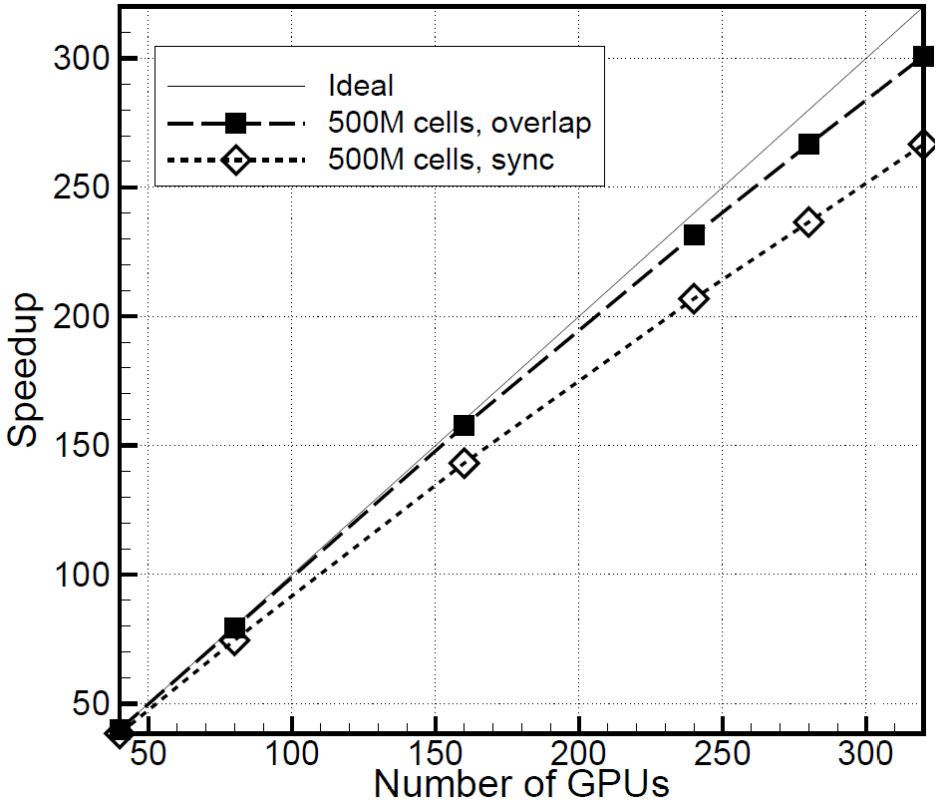
Lomonosov-2:
14C Xeon E5-2697v3
NVIDIA K40, IB FDR

HPC5: 2x 8C Xeon E5-2650v2
2x NVIDIA K80, IB FDR
Internal PE 98% on 4 devices
Sustained performance ~8%

Mesh 29M cells



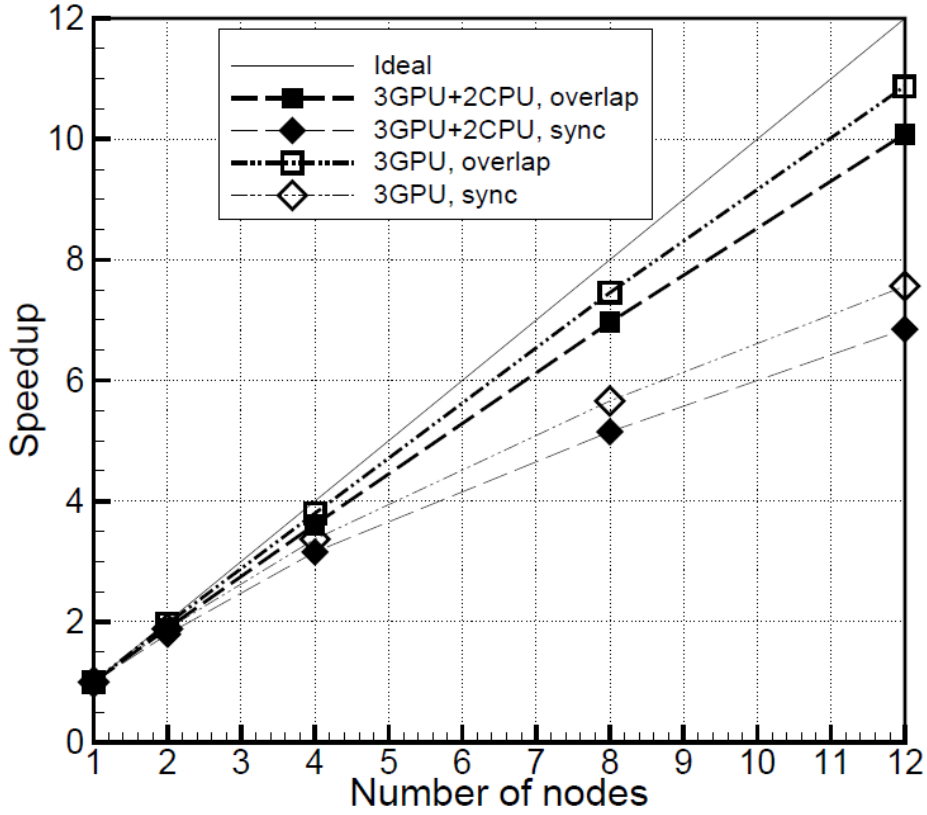
Mesh 500M cells

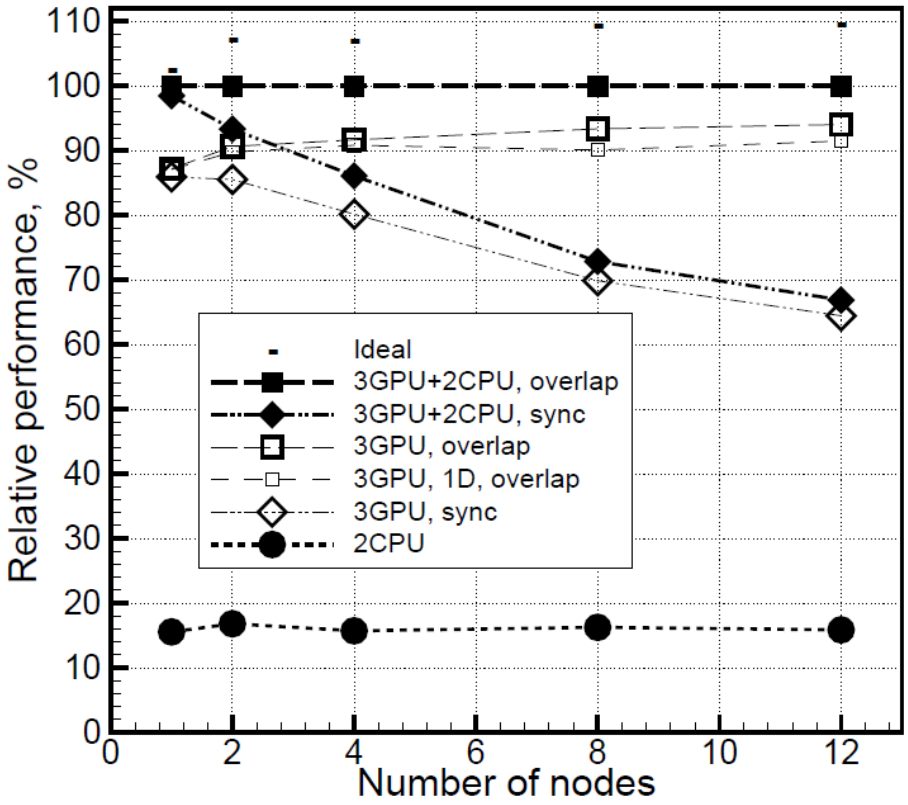


Heterogeneous execution CPU+GPU

K-10: 2 x 8C Xeon E5-2690 + 3 x NVIDIA 2090, IB QDR

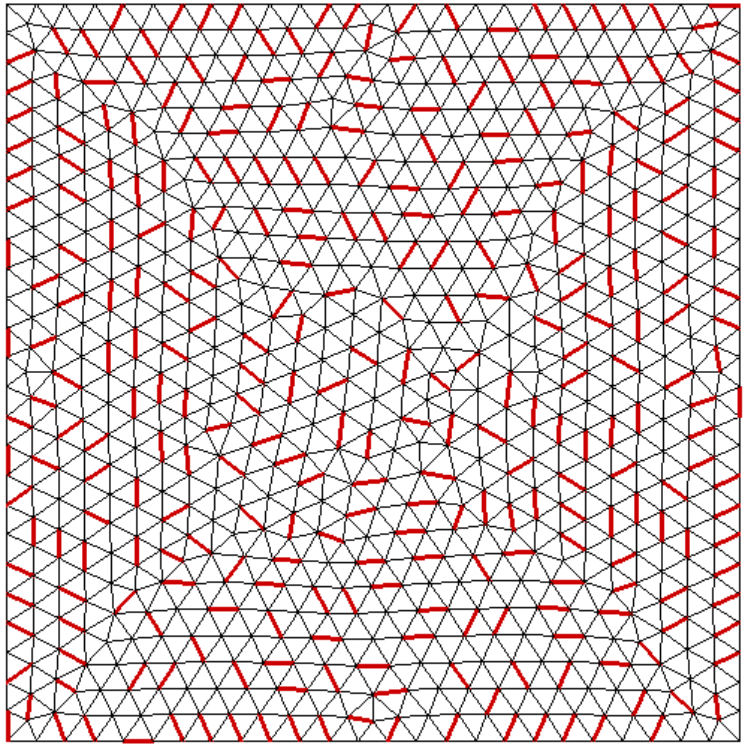
Mesh 12M cells

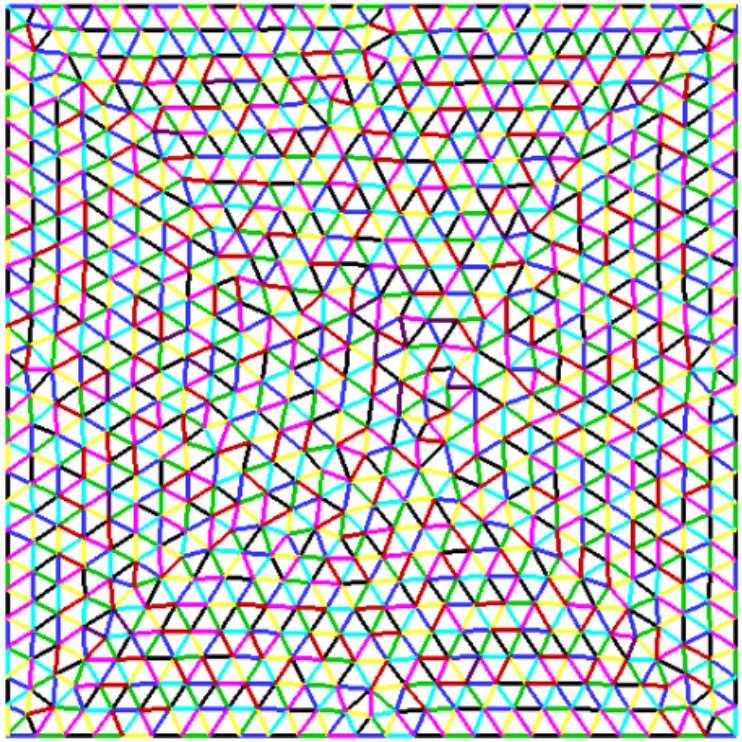




Extension to implicit schemes

- Newton-based implicit schemes
- Preconditioned BICGStab for solving Jacobi system: SpMV, axpy, dot
- Graph coloring for elimination of data interdependency at Jacobi matrix fill-in





HPC^2

High Performance Computing Hybrid Portable Code

A heterogeneous computing CFD software project
CTTC UPC, Barcelona, Spain and KIAM RAS

- **Incompressible turbulent flows and heat transfer**
- **Collocated cell-centered symmetry-preserving discretization, unstructured meshes**
- **MPI + OpenMP + OpenCL/CUDA parallelization**

External CFD code

termofluids

Discrete operators in algebraic form

→ HPC^2 →

Software implementation approach

- **Algorithm is formed of three basic linear algebra SP-compatible operations: SpMV, dot, axpy**

Preprocess: a big CPU code processes mesh data, builds geometry of cells, numerical scheme stencils, etc. and generates output matrices of discrete operators

Complex object-oriented code, user-oriented data structures, whatever implementation methods

Matrices and vectors

Time integration core:
HPC^2 "Algebraic" approach, simple performance-oriented data structures

- Choice of optimal storage formats for sparse matrices of various operators: COO, CSR, sliced block-transposed ELLPACK…
- Abstract implementation at the upper level
- Specification at the lower level for particular framework or combination of frameworks – OpenMP, CUDA, OpenCL

*G. Oyarzun, R. Borrell, A.Gorobets, A. Oliva. "Portable implementation model for CFD simulations. Application to hybrid CPU/GPU supercomputers". International Journal of Computational Fluid Dynamics.

Math model and method

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

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

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

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

- **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:**

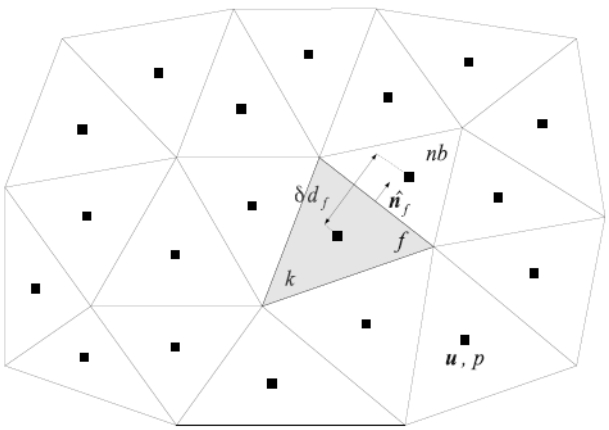
Predictor velocity: $\mathbf{u}^p = \mathbf{u}^n + \Delta t\left(\frac{3}{2}\mathbf{R}^n - \frac{1}{2}\mathbf{R}^{n-1}\right)$

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

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

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

Poisson equation



Symmetry-preserving 2nd order scheme*

The algorithm of the time step

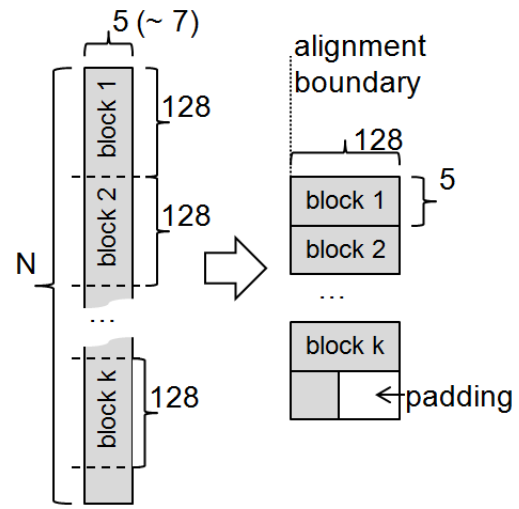
1. **Predictor velocity field, $\mathbf{u}^p$, is obtained explicitly**
2. **Correction, $\tilde{p}$, is obtained from the Poisson equation**
3. **Resulting velocity field, $\mathbf{u}^{n+1}$, is obtained**
4. **Energy equation is solved explicitly**

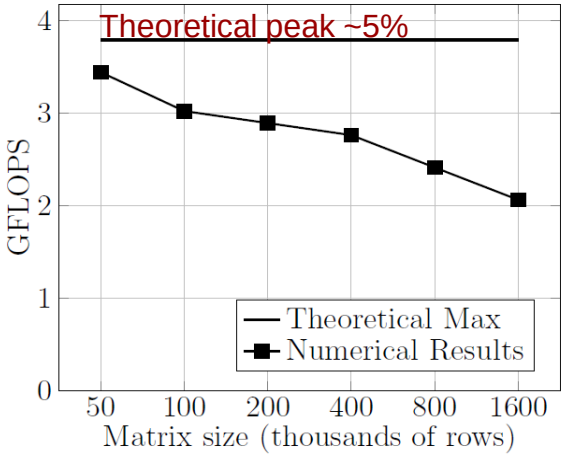
* F.X. Trias, O. Lehmkuhl, A. Oliva, C.D. Perez-Segarra, R.W.C.P. Verstappen,Symmetry-preserving discretization of Navier-Stokes equations on collocated unstructured grids, Journal of Computational Physics, Volume 258, 2014, Pages 246-267

SpMV-based code

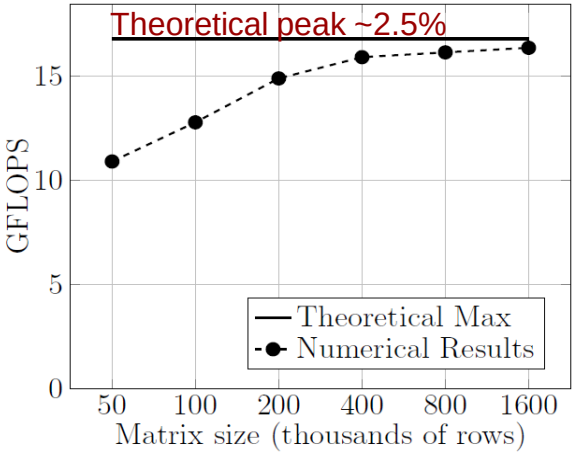
- **SpMV takes ~80% of computing time**
- **Convection operator is made by concatenation of 2 SpMVs: result of 1st SpMV is a matrix for 2nd**
- **Communication and computation overlap**
- **Elements are reordered by**
  - Inner/interface subsets
  - Number of nonzero elements per row
  - RCM band reduction

Device-oriented adaptation for ELLPACK

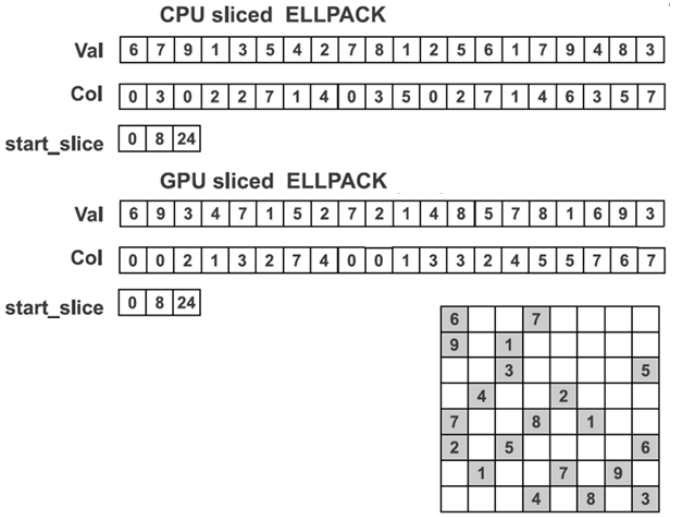




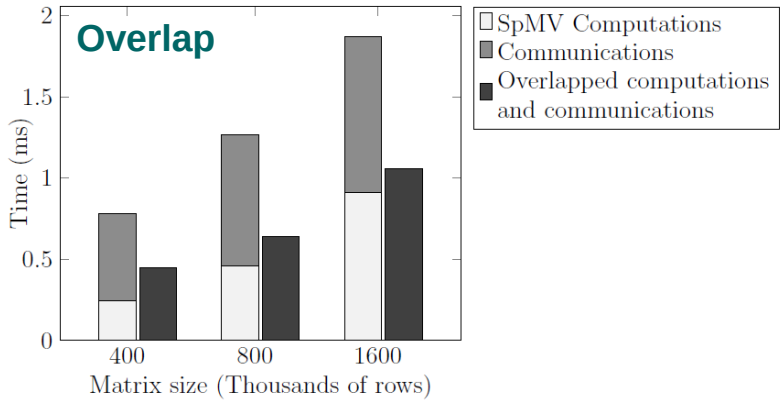
Xeon E5649 6C

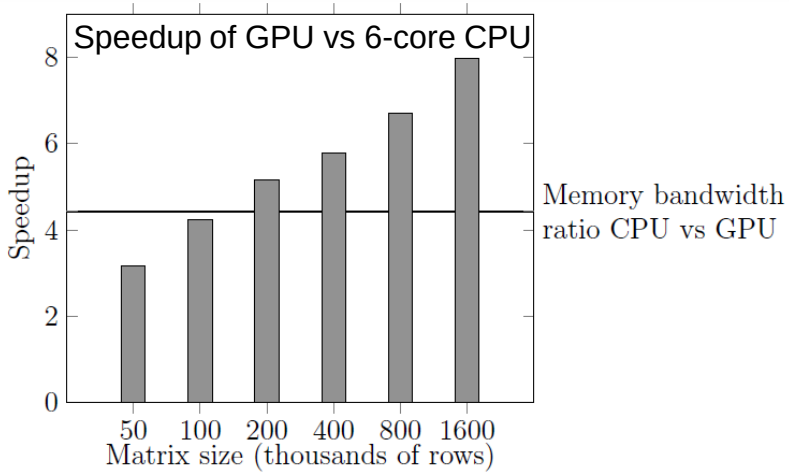


NVIDIA 2090

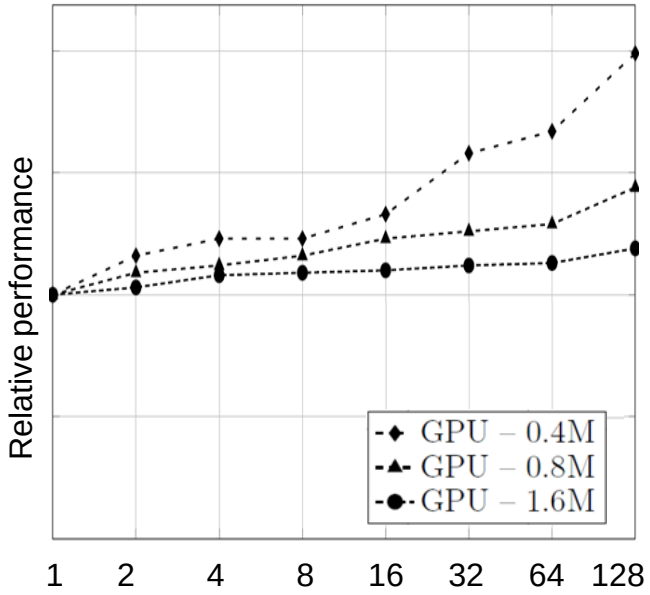


Parallel performance

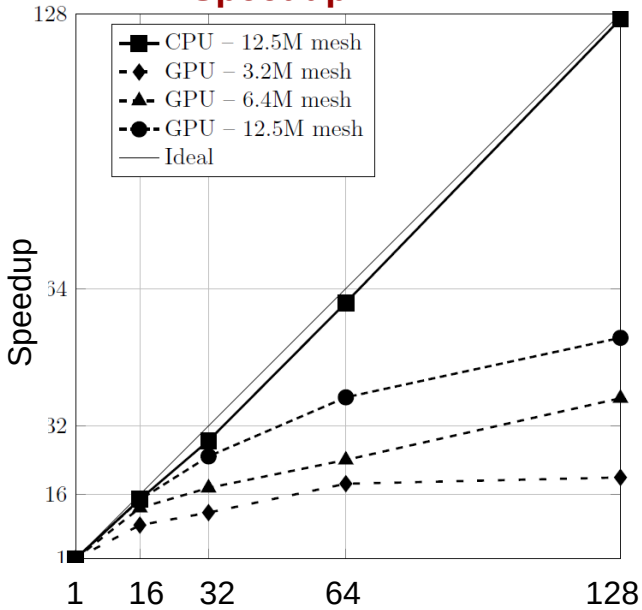




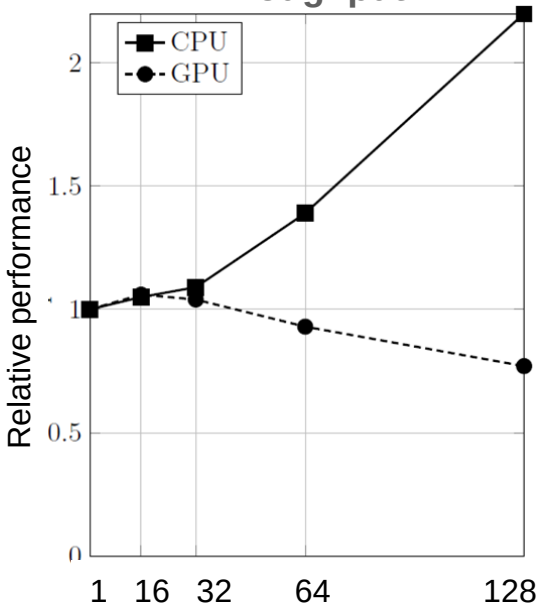
Scalability on GPUs



Speedup

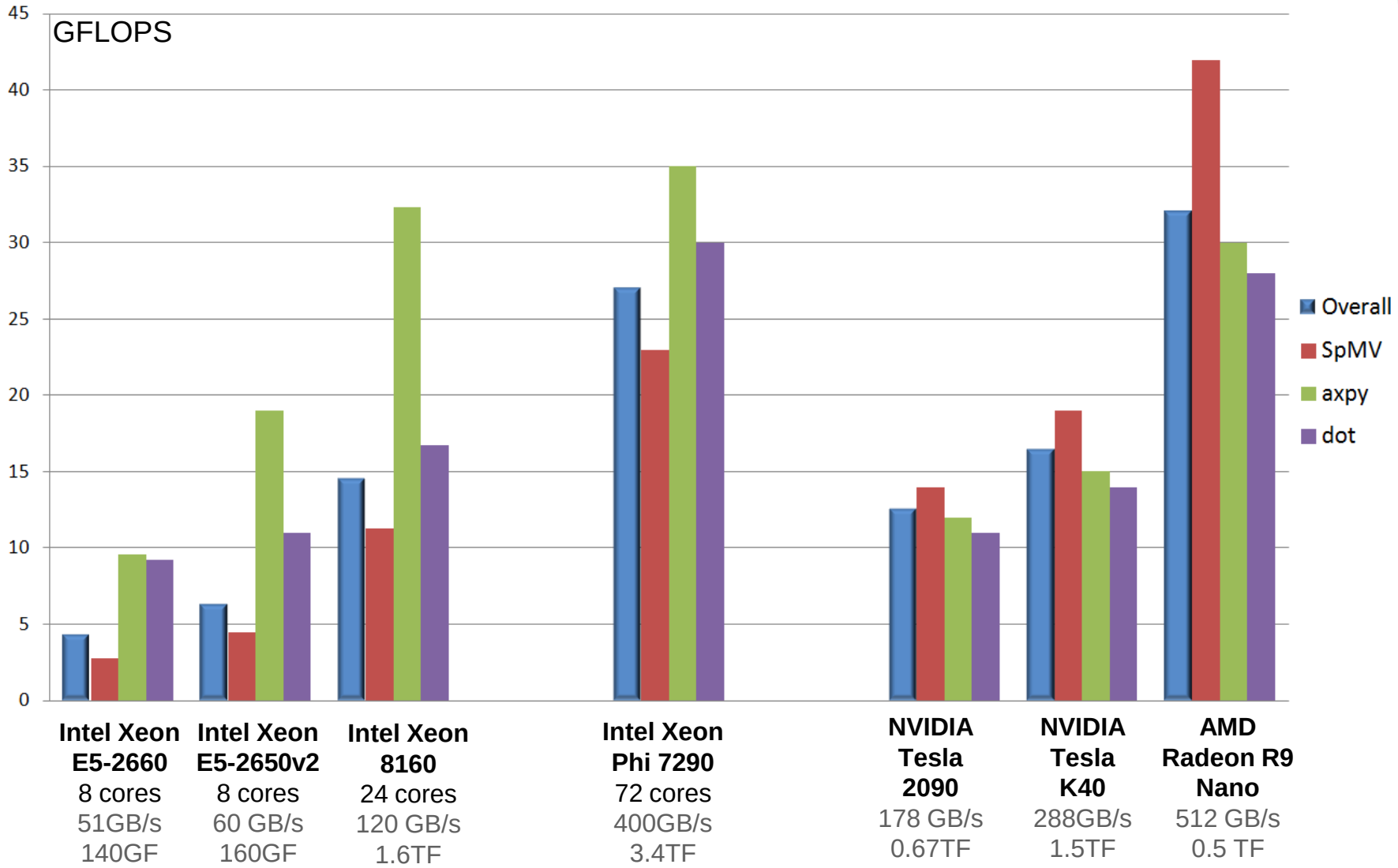


Throughput



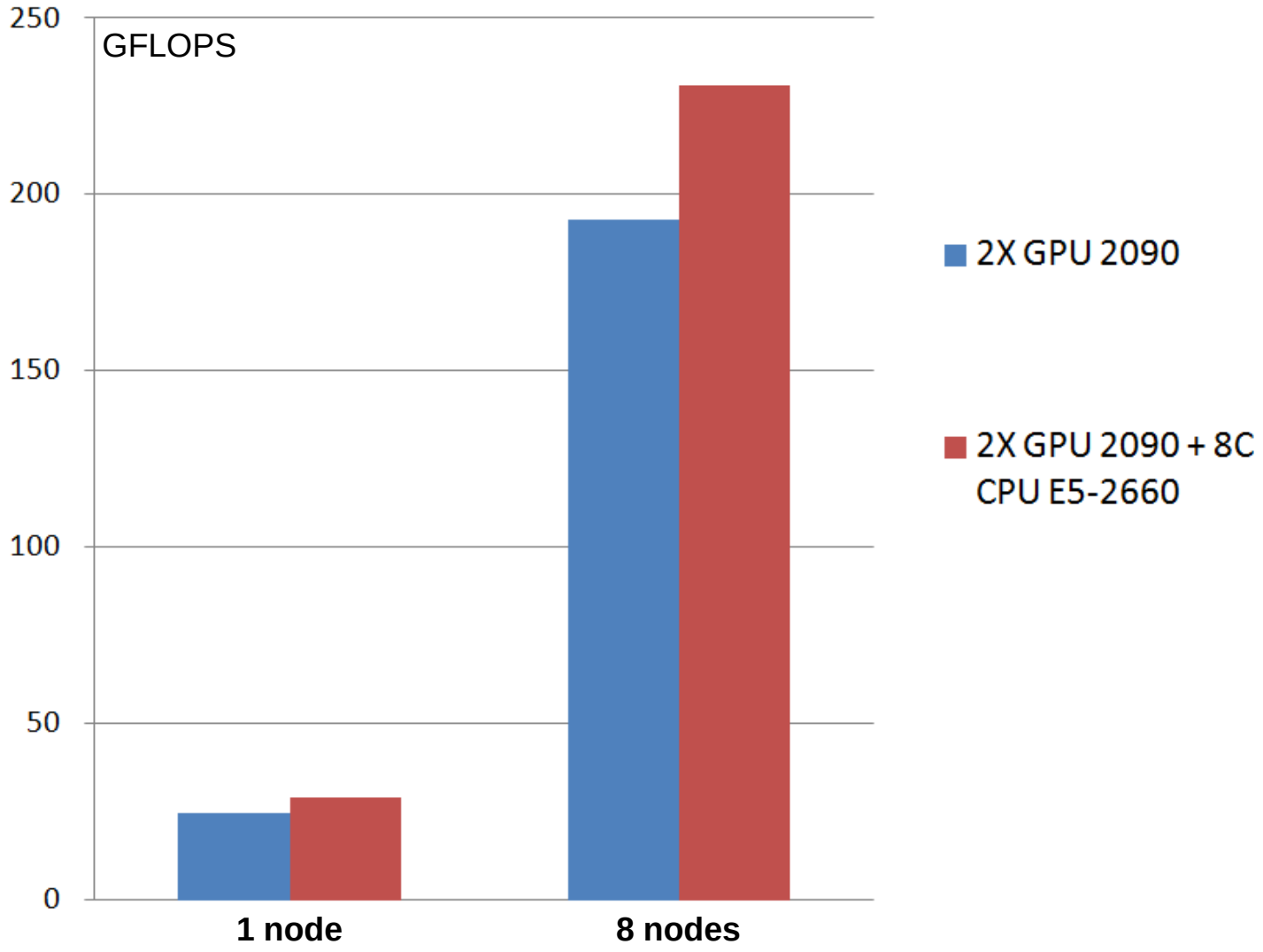
*G. Oyarzun, R. Borrell, A.Gorobets, A. Oliva. "Portable implementation model for CFD simulations. Application to hybrid CPU/GPU supercomputers". International Journal of Computational Fluid Dynamics.

Single device performance



Mesh 1M cells

Heterogeneous performance



Mesh 1M cells per node

Parallel performance on ARM machine

MONT-BLANC

- **Fused GPU+CPU ARM:**
  CPU Coretex A15
  2 cores, 1.7 Ghz, 12.8 GB/s, 6.8 GFLOPS

  GPU Mali T604
  4 cores, 533Mhz, 12.8GB/s, 21 GFLOPS

  Physically shared memory

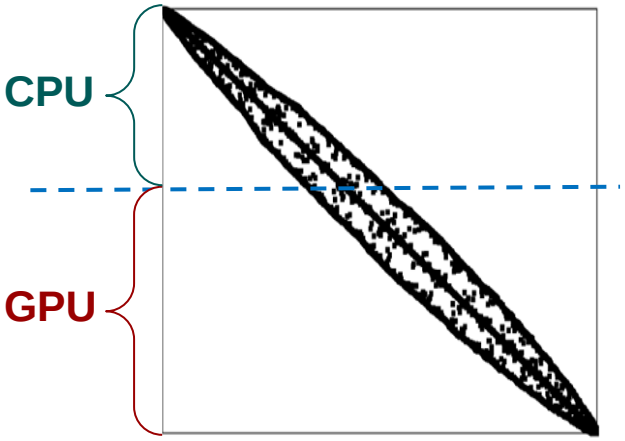
- **10Gb Ethernet network**

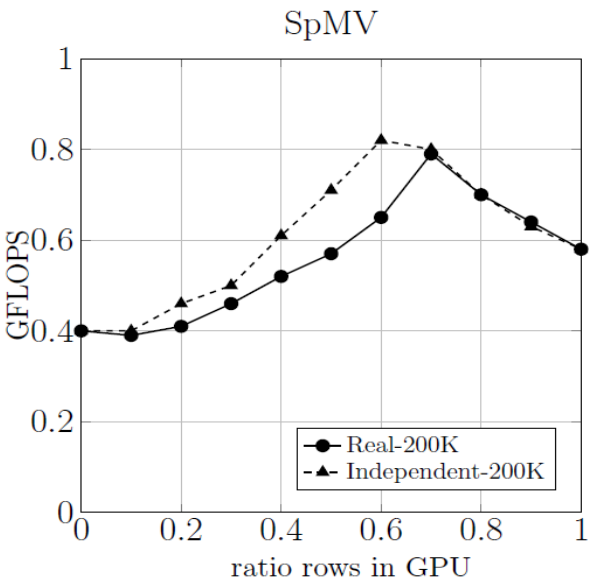
- **1080 compute cards, 2160 CPU cores and 1080 GPUs**

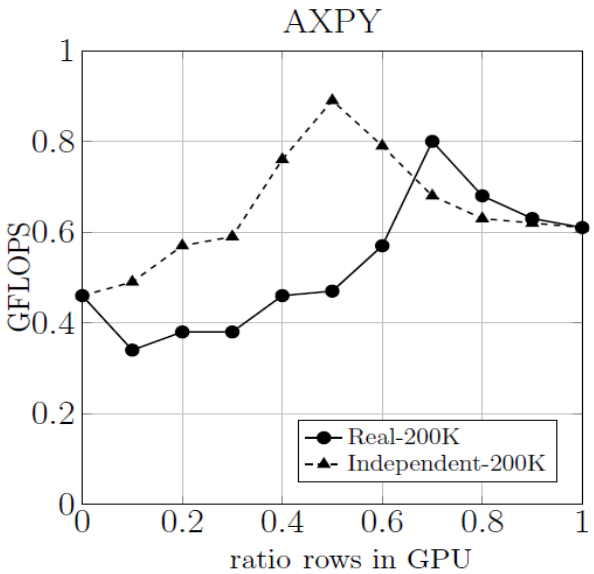
*G.Oyarzun, R. Borrell, A. Gorobets, F. Mantovani, A. Oliva. “Efficient CFD code implementation for the ARM-based Mont-Blanc architecture”. Future Generation Computer Systems. In press. https://doi.org/10.1016/j.future.2017.09.029

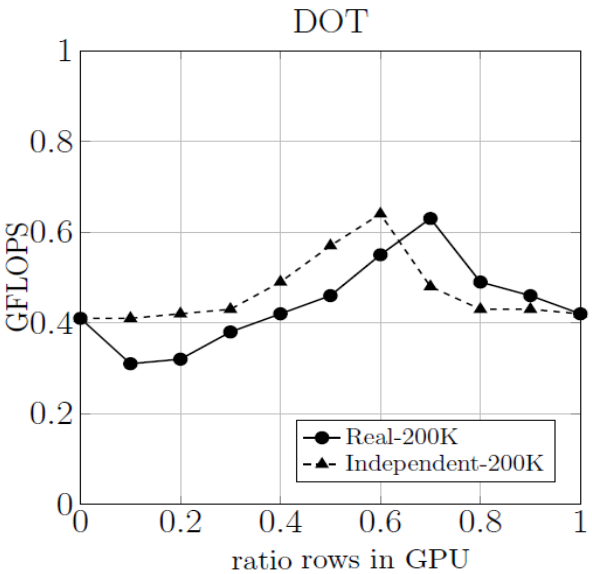
Parallel performance on ARM machine

- Automatic dynamic load balancing between CPU and GPU
- Balancing based on separate timing of CPU and GPU is far from optimal as devices compete for bandwidth
- Performance may vary runtime (overheat?)



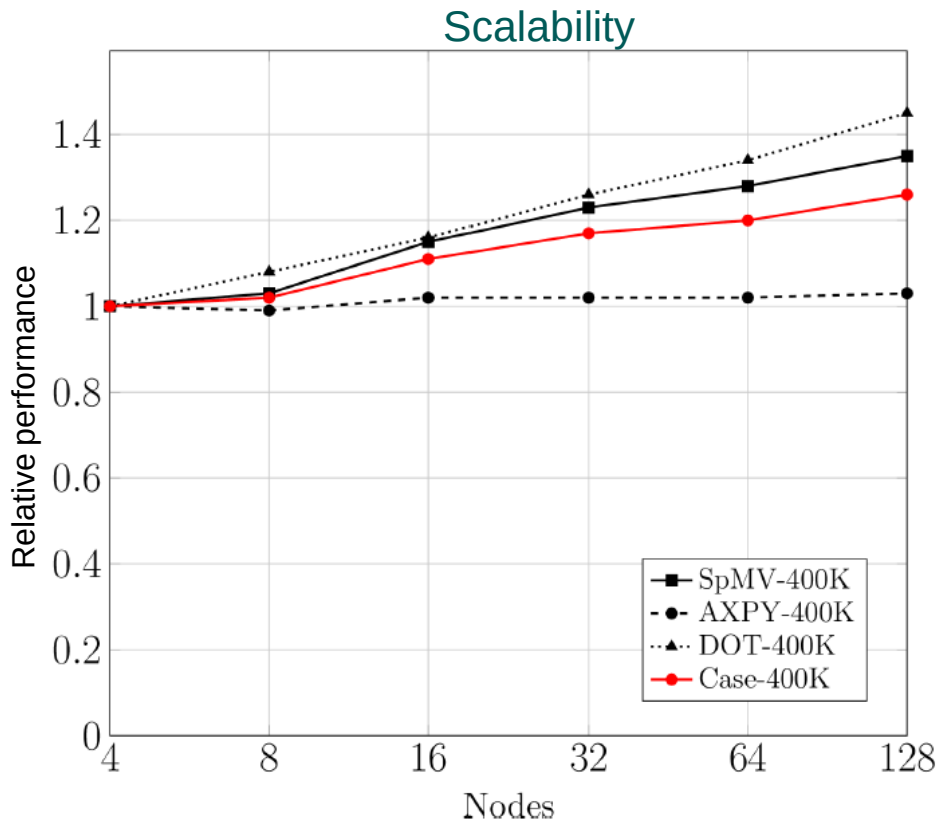


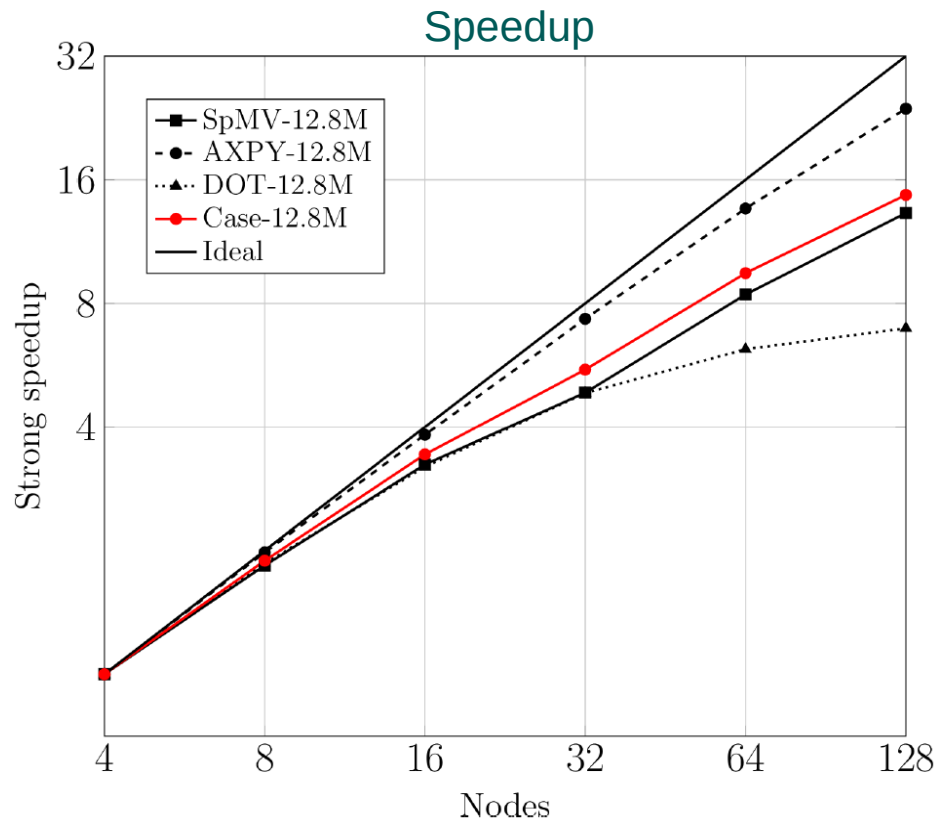




Parallel performance on ARM machine

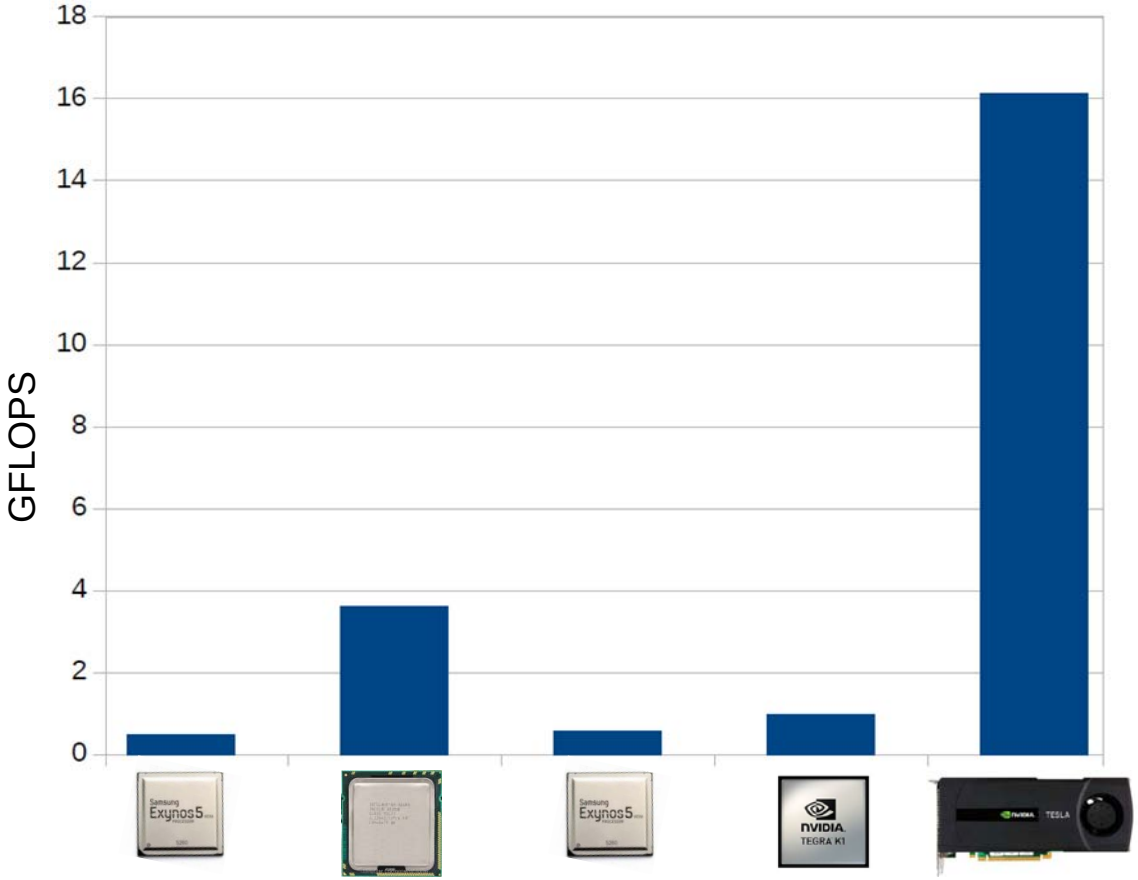
MPI-parallel execution on multiple nodes





*G.Oyarzun, R. Borrell, A. Gorobets, F. Mantovani, A. Oliva. “Efficient CFD code implementation for the ARM-based Mont-Blanc architecture”. Future Generation Computer Systems. In press. https://doi.org/10.1016/j.future.2017.09.029

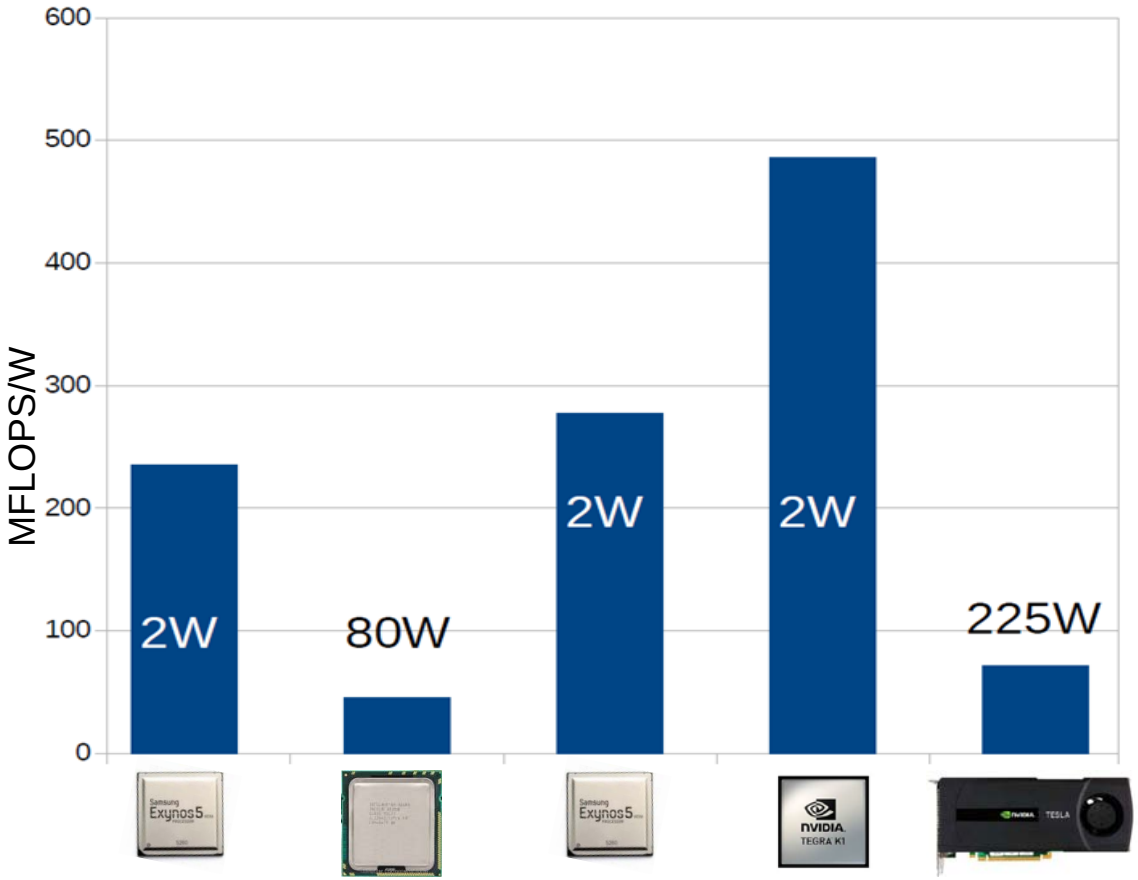
Comparison of performance on single devices



ARM Cortex-A15	Xeon E5649	Mali T604	NVIDIA Tegra K1	NVIDIA 2090
2 cores	6 cores	12.8GB/s	17.1GB/s	177GB/s
12.8GB/s	32GB/s	21GF	15GF	665GF
6.8GF	70GF			

*G.Oyarzun, R. Borrell, A. Gorobets, F. Mantovani, A. Oliva. “Efficient CFD code implementation for the ARM-based Mont-Blanc architecture”. Future Generation Computer Systems. In press. https://doi.org/10.1016/j.future.2017.09.029

Comparison of performance on single devices

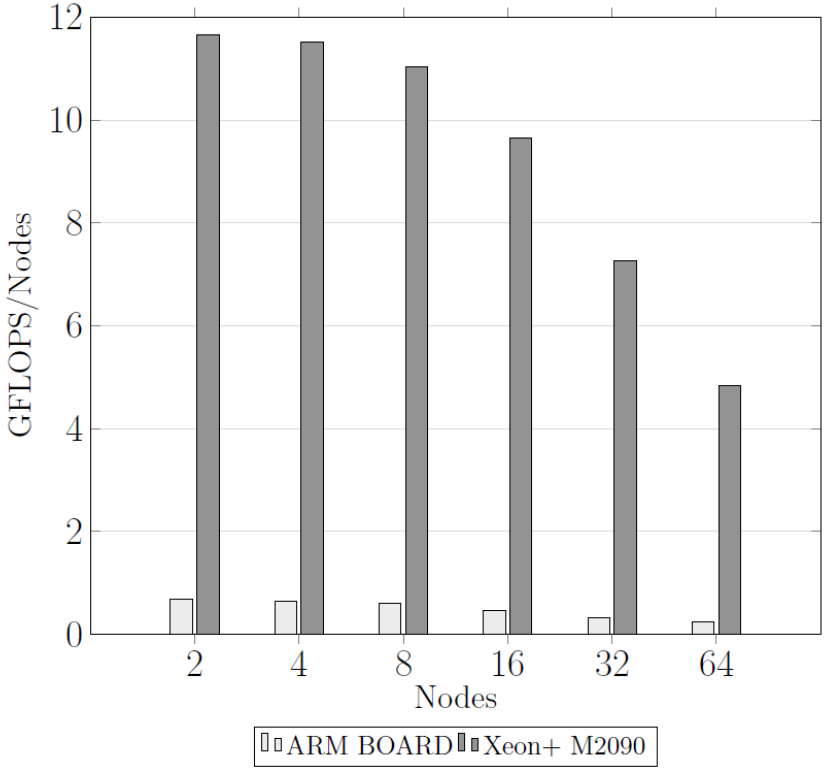


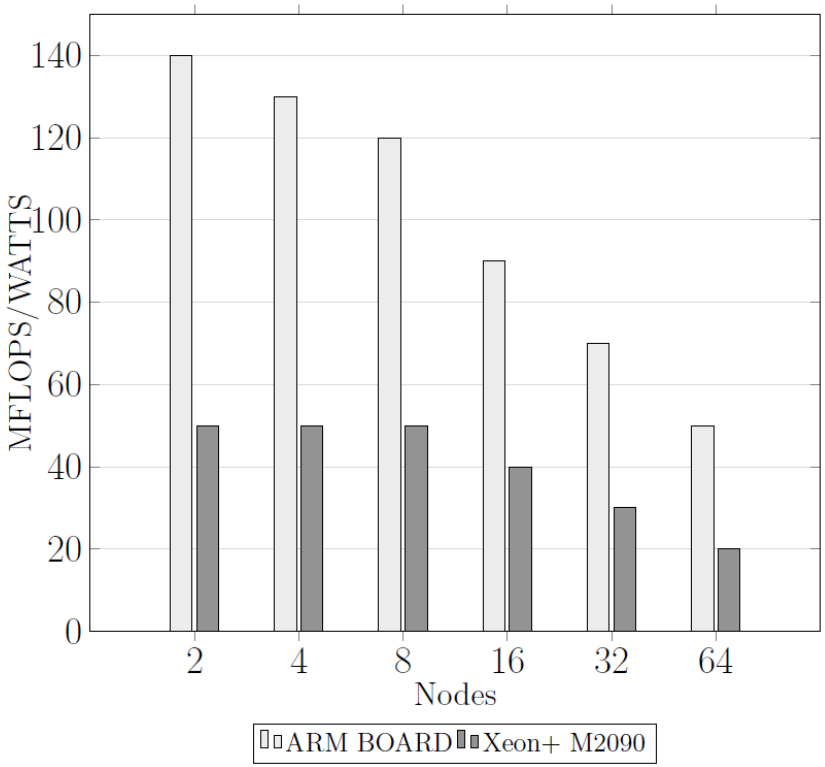
ARM Cortex-A15	Xeon E5649	Mali T604	NVIDIA Tegra K1	NVIDIA 2090
2 cores	6 cores	12.8GB/s	17.1GB/s	177GB/s
12.8GB/s	32GB/s	21GF	15GF	665GF
6.8GF	70GF			

*G.Oyarzun, R. Borrell, A. Gorobets, F. Mantovani, A. Oliva. “Efficient CFD code implementation for the ARM-based Mont-Blanc architecture”. Future Generation Computer Systems. In press. https://doi.org/10.1016/j.future.2017.09.029

Comparison of MontBlanc with a hybrid supercomputer

- **MontBanc ARM vs Minotauro**
  128 nodes with 2 CPU and 2 GPU
  Intel Xeon E5649 6C + NVIDIA M2090
  InfiniBand QDR network





Conclusions

- **Simplicity and stream processing compatibility**
- **Memory access optimization**
- **Overlap of computations and communications**
- **2-level partitioning and load balancing**
- **heterogeneous execution mode**
- **Resulting code can run on any hybrid architecture**
  CPU, GPU, MIC, ARM, SoC…
  to simulate compressible flows from low-Mach to supersonic
  or incompressible flows using unstructured hybrid meshes

Thank you for your attention

We used K-100 (KIAM RAS), MVS-10P (JSCC RAS), Lomonosov-2 (MSU) machines.
We are financially supported by RSF (№ 15-11-30039) and RFBR (№ 15-07-04213).
We thankfully acknowledge these institutions.