USING HYBRID MPI+OPENMP PARALLELIZATION FOR LARGE-SCALE CFD/CAA SIMULATIONS

A. V. Gorobets, S. A. Soukov

Trilateral Russian-French-German Workshop
"Computational Experiment in Aeroacoustics"
Svetlogorsk, September 22-25, 2010

Outline

- **Modern supercomputers and specifics of CAA applications**
  - Specifics of CAA applications
  - Examples of modern supercomputers
  - Efficiency issues
- **Parallel CFD/CAA algorithm for compressible flows**
  - Overview of the code
  - Two-level MPI+OpenMP parallelization
  - Performance on Lomonosov supercomputer
- **Alternative architectures**
  - Hybrid computing model with GP GPU
  - Problems to solve
- **Illustrative applications**
  - Impinging jet and square cylinder DNS cases
  - Flow around a finite cylinder and round jet interaction with cylinder DNS cases

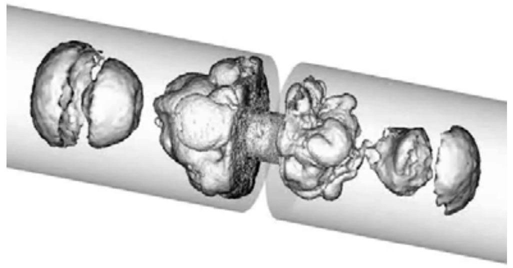
Specifics of CAA applications

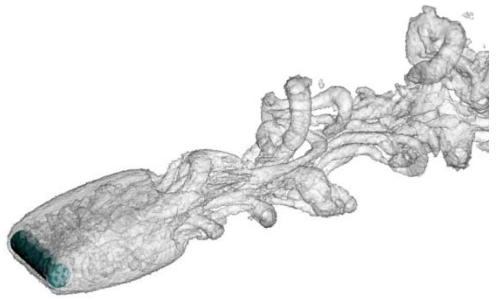
High computing cost is due to

- **The need in high-order schemes with extensive space stencil**
  to resolve well both complex turbulent flows and propagations of acoustic waves
- **Large computing domain**
  Big difference in scales, distant positions of probes, etc.
- **High resolution in space**
  both in the near field, where noise generation takes place, and more distant areas
- **High resolution in time**
  time step is limited by the high-frequency part of the spectrum
- **Long time integration period**
  to get well converged spectra and flow fields

High degree of parallelism

- **Applicability of explicit schemes in many cases of interest**
  due to limitations on time step implied by the physics of the processes to model
- **Scalable iterative solvers in case of implicit schemes**





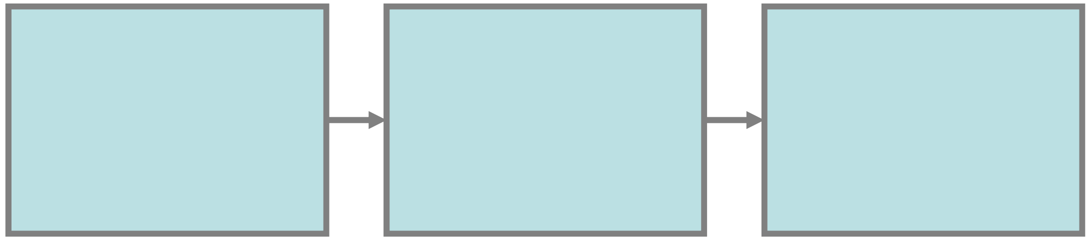
Specifics of CAA applications

A specific balance in computing cost from a parallel point of view

- **Computing price is high much rather due to a long time integration**
  than a mesh size. This leads to a bad computing pattern.
- **Large space stencil leads to wide halos around subdomains**
  and bigger size of communications between parallel processes

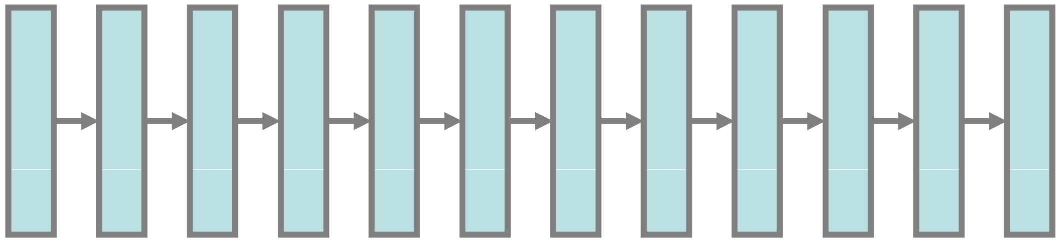
A bad pattern…

Small amount of “heavy” steps (in general) - good



here we can envy chemists :)

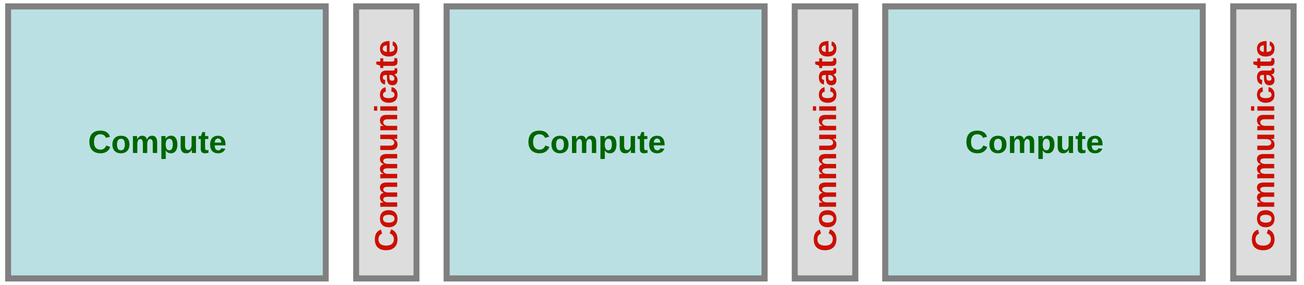
Big amount of “light” steps - bad



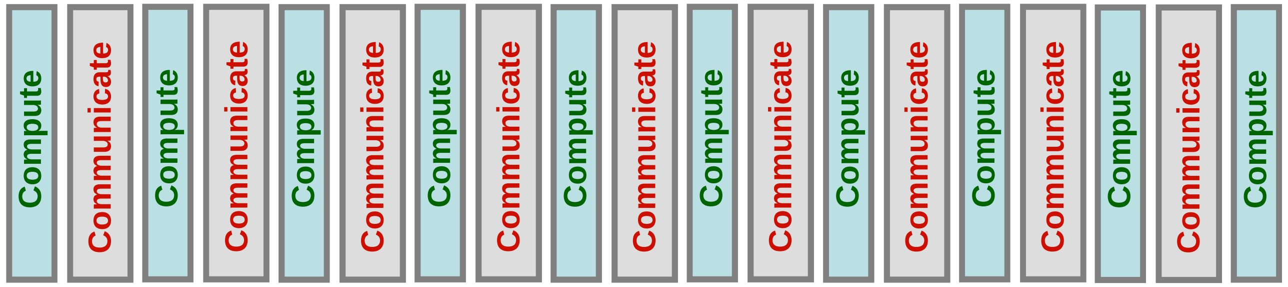
Specifics of CAA applications

Computational pattern and parallel efficiency

Lots of computations between exchange – **good pattern**



Frequent exchanges between a small computing load – **bad pattern**



- **For this reason CAA algorithms must speed up well even for coarse meshes**
  special attention must be paid on parallel efficiency and optimization of data exchange

Typical supercomputers

Lomonosov, MSU

Network	Infiniband
CPUs	Intel EM64T Xeon 55xx 2930 MHz
Number of cores	**35 776**
Nodes	**8 cores** (2 x 4-core CPUs), 12Gb of RAM
Rmax, Tflops	350
Location	Moscow, Russia

MVS-100000, JSC of RAS

Network	Infiniband
CPUs	Intel EM64T Xeon 53xx 3000 MHz
Number of cores	**11680**
Nodes	**8 cores** (2 x 4-core CPUs), 8Gb of RAM
Rmax, Tflops	107
Location	Moscow, Russia

- **Supercomputers consist of nodes coupled with a high-performance interconnection.**
  Each node has its own RAM memory address space, so it is a distributed memory parallel system.

- **Each node has multiple CPU cores**
  that share the same RAM memory. So the node itself is a shared memory parallel system

Multiple efficiency issues

Efficiency of computations

- Parallel efficiency
- Efficient use of CPUs
- Computational efficiency of the algorithm

Each of these indicators is meaningless if considered alone

Efficiency of performing a simulation

- Choice of the number of CPUs
- Efficient file system usage
- Reaching faster the statistically stationary state

Efficiency of the numerical method

Implicit or explicit time integration, which solver to use, etc.
The proper choice and configuration can depend on many factors like

- Geometry, Re/Ra number, Mach number, CFL, …
- Mesh size, concentration, ..
- Limitations on time step, specifics of the flow, etc…

Efficiency of the discretization

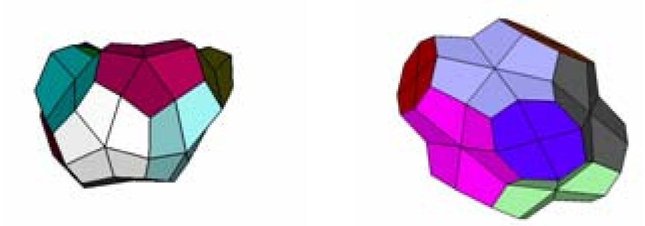
- Efficient geometry definition
- Optimization of the mesh concentrations, adaptive refinement
- Proper choice of the mesh elements

A CFD algorithm for compressible flows

The system to solve

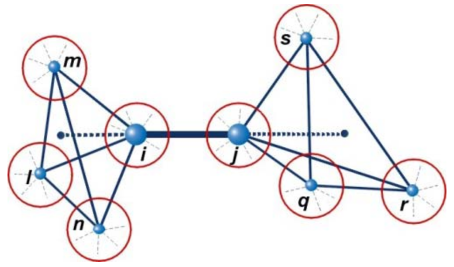
$$\frac{\partial \mathbf{Q}}{\partial t} + \frac{\partial \mathbf{F}(\mathbf{Q})}{\partial x} + \frac{\partial \mathbf{G}(\mathbf{Q})}{\partial y} + \frac{\partial \mathbf{H}(\mathbf{Q})}{\partial z} = \frac{1}{Re}\left(\frac{\partial \mathbf{F}_\nu(\mathbf{Q})}{\partial x} + \frac{\partial \mathbf{G}_\nu(\mathbf{Q})}{\partial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

where **Q** - is a vector of full or linearized conservative variables, **F, G, H** - vectors of full or linearized conservative fluxes, $\mathbf{F}_\nu$, $\mathbf{G}_\nu$, $\mathbf{H}_\nu$ - vectors of full or linearized dissipative fluxes, *Re* - Reynolds number.

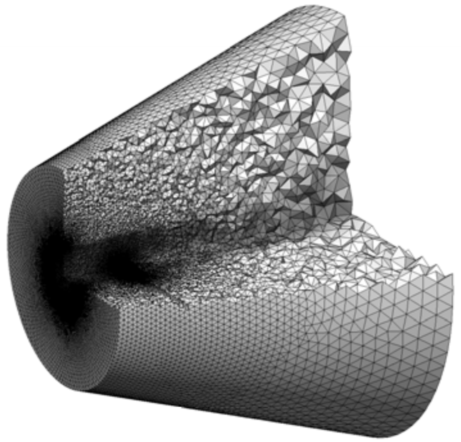

Examples of control volumes

The NOISETTE code for CFD/CAA

- **Euler based family of models**
  EE, NSE, NLDE, LEE, LNSE
- **Unstructured tetrahedral meshes**
- **High order numerical schemes***
  Multi-parameter **vertex-centered** scheme (up to 6th order):
  finite-volume approach for convective terms,
  finite-element approach for diffusive terms.
- **Implicit and explicit time integration**
  Explicit Runge-Kutta up to 4-th order in time
  Implicit up to 2-nd order in time



Extended high-order space stencil

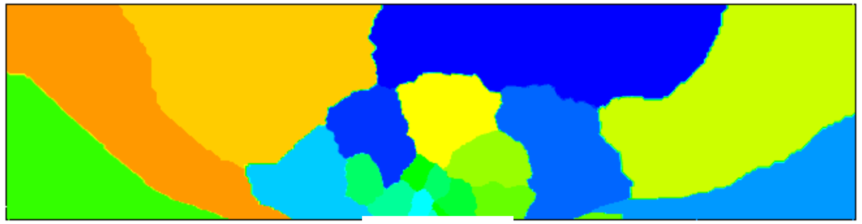

Example of a tetrahedral mesh

* Tatiana Kozubskaya Ilya Abalakin, Alain Dervieux, “High Accuracy Finite Volume Method for Solving Nonlinear Aeroacoustics Problems on Unstructured Meshes”, Chinese Journal of Aeroanautics, pages 97-104, 2006.

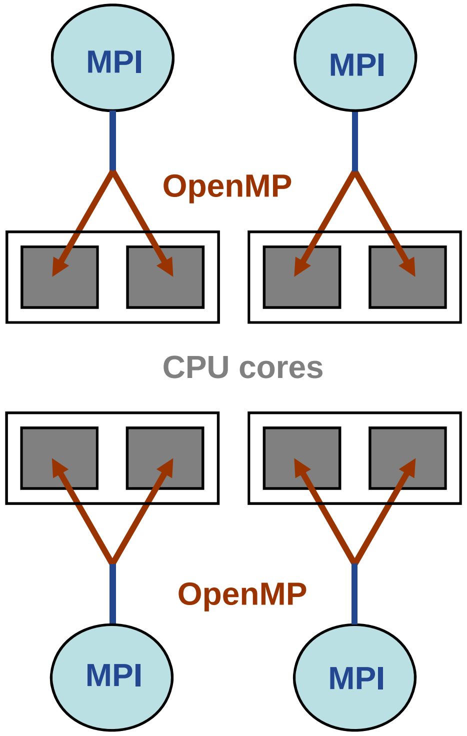
Two-level approach

Two-level hybrid MPI+OpenMP parallelization

- **MPI works on the first level**
  to couple a group of parallel processes running on different nodes of a supercomputer using a common geometric parallelism approach and the distributed memory model



- **OpenMP works on the second level inside nodes**
  it provides parallelism inside of multi-core nodes of a supercomputer within the shared memory model



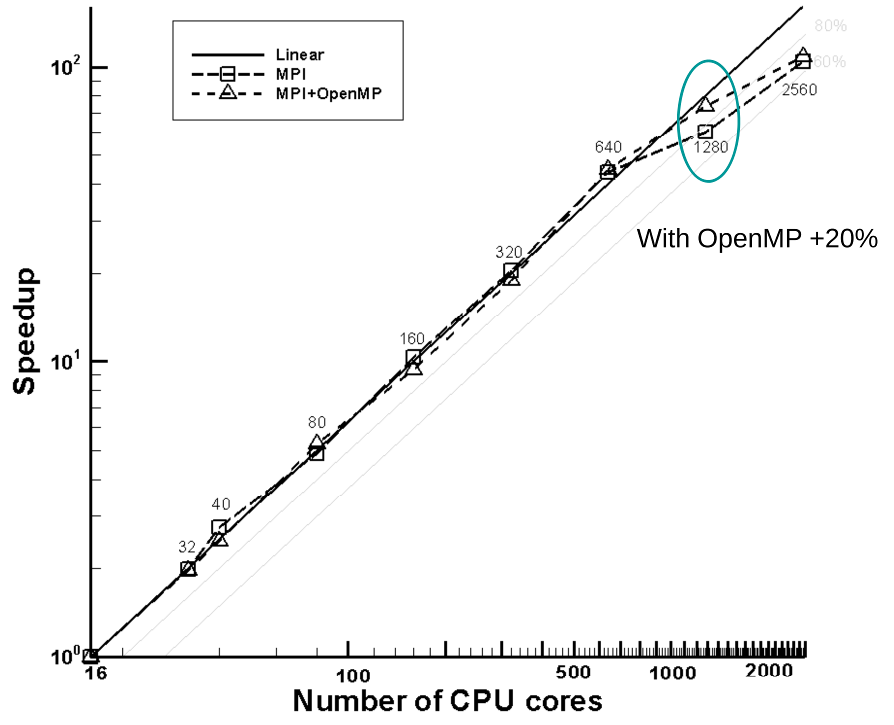
OpenMP gives following main advantages

- **The number of communicating processes reduced P_t times**
  where Pt is the number of OpenMP threads per MPI process.
- **The size of interfaces between subdomains reduced around Pt times**
  resulting in smaller amount of data exchange.
- **It improves the use of network hardware - communications are faster**
  no multiple processes in queue for a shared network hardware inside of a multi-core the node.
- **Easy to change the number of CPU cores**

Improving the MPI-only parallelization with OpenMP

Increasing efficiency with OpenMP

- **A small test with only 1.2M mesh was performed.**
- **Simplified OpenMP implementation with replication of data.**
  No reordering or second level decomposition.
  Just replication of fluxes and summation by the master thread.
- **The use of OpenMP with only 2 threads shown up to 20% increase in parallel efficiency.**



Speedup on MVS – 100000, explicit t. i.

Shared memory nightmare

Apart from advantages OpenMP has problems as well. Data races – that's not for kids…

- **Cache coherence problem**

  Shared variable a = 0
  Thread 1 writes to a: **a = 1**
  Then thread 2 reads from a: **b = a + 1**
  what will be in b? hint: (uncertain 1 or 2)

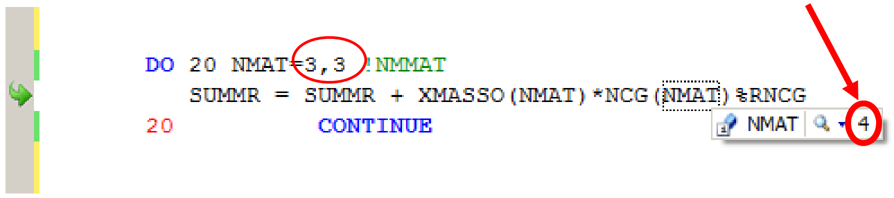
- **Intersection of memory write operations: inconsistency of data**

  Shared variable a = 0
  Thread 1 writes to a: a = a + 1 Thread 2 writes to a: a = a + 1
  what will be in a? hint: (uncertain 1 or 2)

  Shared variable a = 0
  Thread 1 sets a: a = 1 Thread 2 sets a: a = 2
  Thread 1 computes b1=f(a) Thread 2 computes b2=f(a)
  what will it result in? hint: (garbage)

- **As a consequence – strange bugs**
  that are highly dangerous for the mental health

THAT'S A KIND OF MAGIC!!!!11

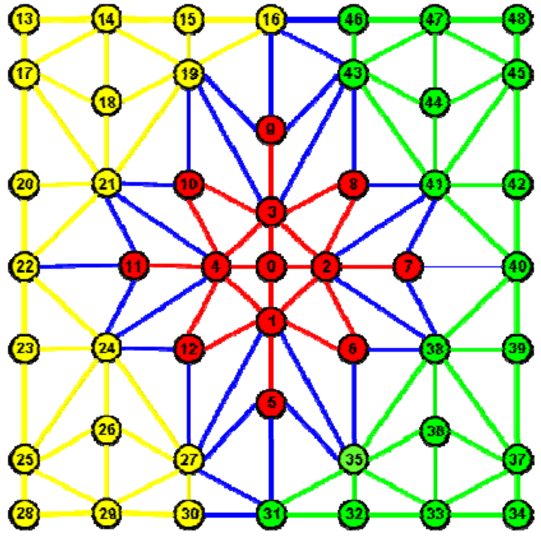


```
      DO 20 NMAT=3,3 !NMMAT
         SUMMR = SUMMR + XMASSO(NMAT)*NCG(NMAT)%RNCG
20             CONTINUE
```

A two-level MPI+OpenMP parallelization

MPI subdomains are decomposed further for OpenMP

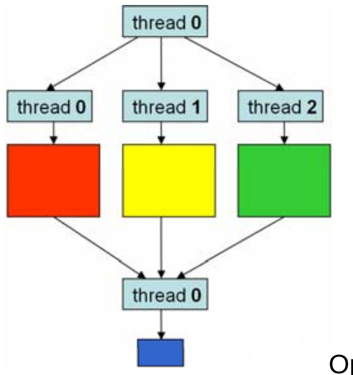
- **Nodes of each MPI subdomain are divided into P_t subsets**
- **Mesh elements that involve nodes from different subsets are the interface elements**



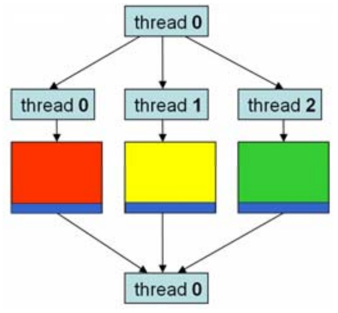
Second level decomposition

The following options of avoiding data intersection have been considered:

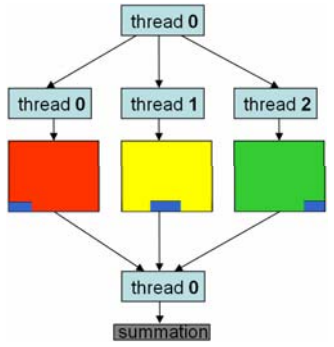
1. **Sequential processing of interface elements.**
   Each thread processes only its inner elements, then master thread processes the interface elements.
2. **Overlap.**
   Each thread processes its inner elements + the interface elements that have nodes belonging to the thread. Thread writes data only to positions of its nodes (in arrays, or rows of matrices).
3. **Replication of data.**
   Interface elements are also divided between threads, threads write data to its own arrays which then are summated by the master thread.



Option 1 (not used)



Option 2 – for gradients, coefficients of Jacobian, etc.



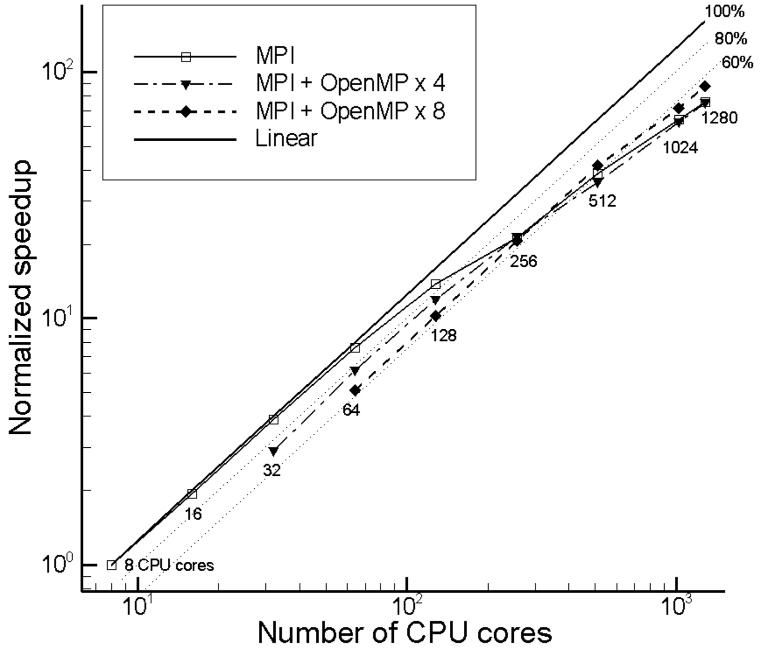
Option 3 – only for fluxes

Speedups with MPI vs. MPI+OpenMP for coarse meshes

Tests on Lomonosov supercomputer

- **Real cases with reduced meshes are used for tests**
  Mesh size is reduced in order to exhaust the parallelism with the available number of CPU cores.

- **OpenMP outperforms MPI for big number of CPUs when the number of nodes per core is smaller**
  Intersection point moves upwards with the growth of mesh size



Speedups with 780K nodes mesh

Speedups with MPI+OpenMP for full resolution meshes

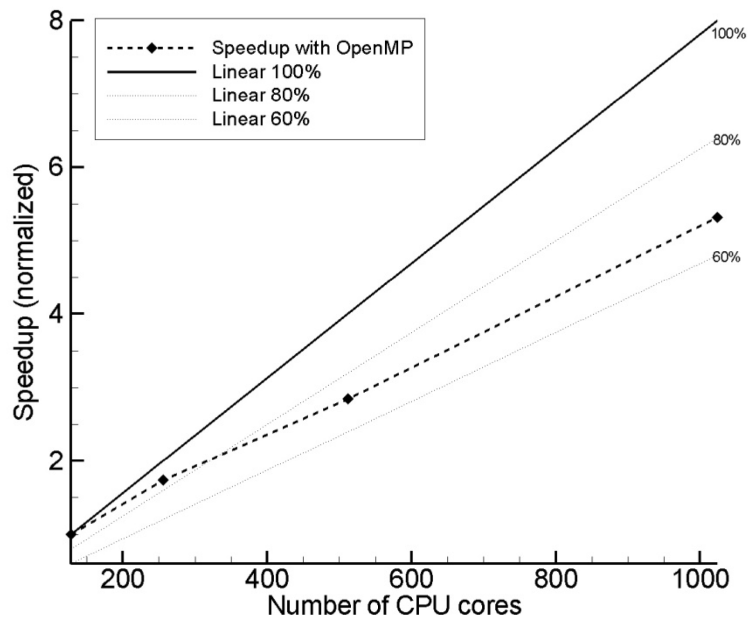
Tests on Lomonosov supercomputer

- **Round jet & cylinder DNS case**

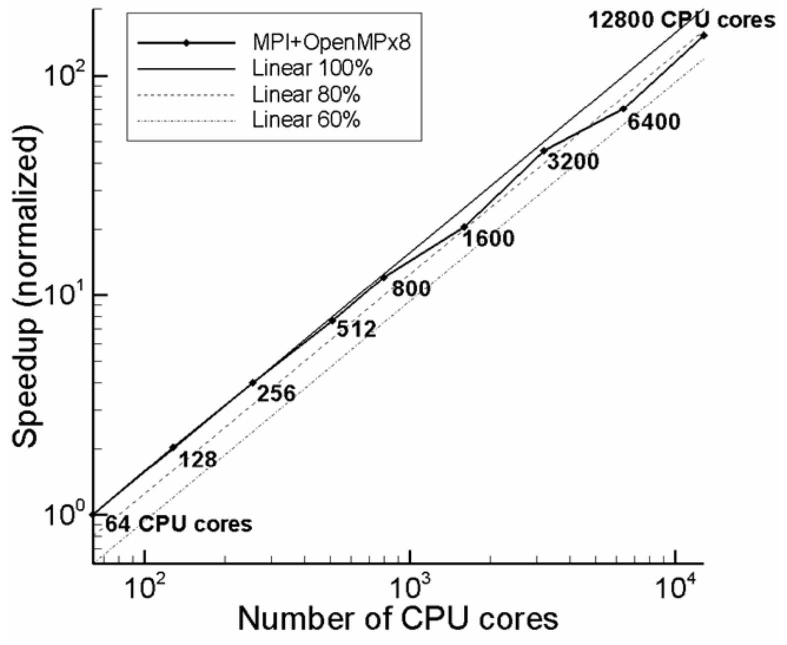
  Mesh size ~16M nodes, ~100M tetrahedrons, 4-th order 4-step Runge-Kutta explicit time integration

- **Reference times:**

  26.8 sec. per time step on 64 CPU cores and 0.38 sec. on 6400 cores, normalized speedup 70.4.



Speedup with OpenMP, MPI group 128 processes is fixed



Speedup with MPI, fixed 8 OpenMP threads (log scale)

Conclusions about MPI+OpenMP

- **This additional OpenMP parallelization significantly extends MPI scalability limitations**
  being at the same time rather easy to implement. If you aware what you fight with…
- **Analysis of throughout profiling results allows to estimate that the range of 100000 CPU cores**
  can be efficiently reached at least for relatively big meshes of 50-100M nodes and more.
- **Bigger meshes can be used.**
  Applicability of the algorithm was demonstrated earlier for a 200M mesh (>1G tetrahedrons) for MPI-only parallelization.
  Now with OpenMP more RAM memory available for MPI processes so we can go for bigger meshes.

Alternatives for the future: Hybrid MPI+OpenMP+SIMD

SIMD or vector parallelism

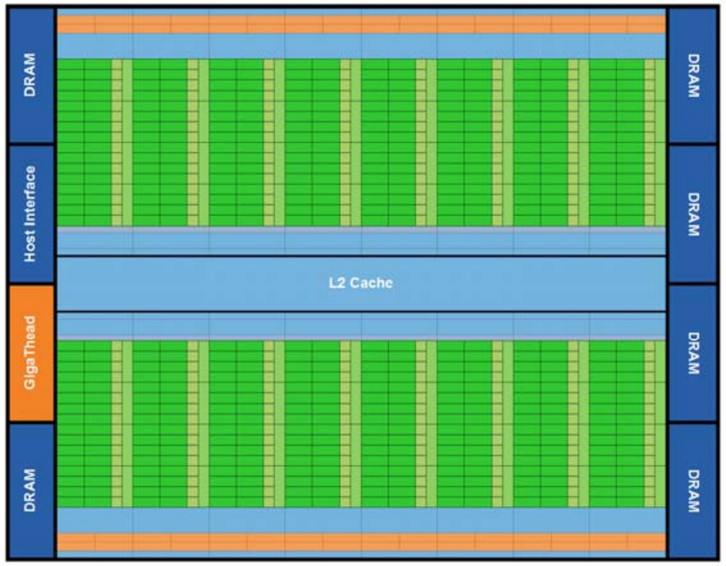
- **GP GPU computing is becoming more and more popular**
  offering brutal computing power
- **New hybrid architecture: Cluster of nodes with multi-core CPUs + one or more GPUs**
  combines different parallel models on a one single node.

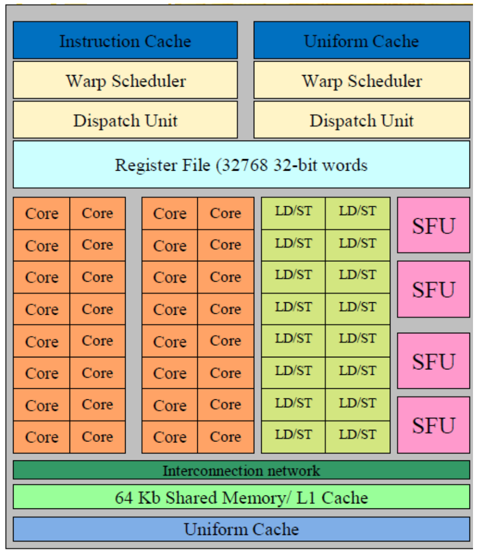
Advantages

- **Huge potential performance (which is rather hard to get)**
- **Overlap of computations and data exchange**
  GPU can compute while data is being transferred between RAM and GPU memory or between MPI processes

Problems and nightmares

- **Transferring from RAM to GPU memory is “very” slow.**
  Having half code on CPU and half code on GPU can hardly be efficient because of low computing price per memory unit.
- **SIMD parallelism is rather hard to use. Especially for unstructured meshes.**
  Firstly we are focusing on a sparse matrix-vector product for unstructured matrices.
  Then more problems will appear.
- **OpenCL is not yet available for Fortran and CUDA is not portable.**





Alternatives for the future: Hybrid MPI+OpenMP+SIMD

Unstructured sparse matrix and MVP

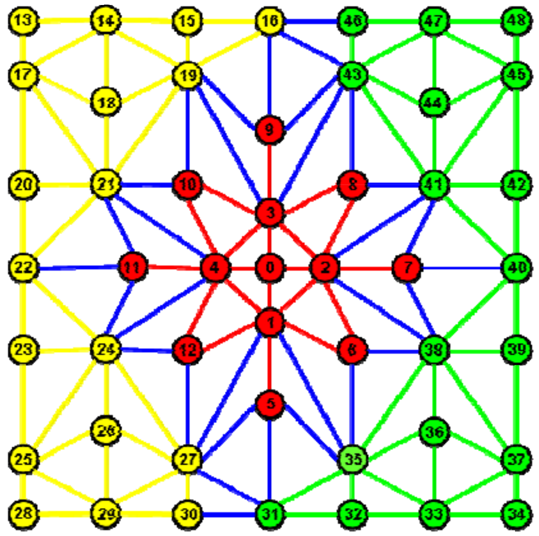
- **CSR representation of block sparse matrix**
- **How to compute coefficients efficiently and how to deal with this matrix well?**
  Mainly how to do MVP efficiently?

Problems

- **Vector parallelism implies that nodes have the same set of numerical operations**
  which is not the case for unstructured mesh: nodes have different numbers of neighbors hence different sets of operations.
- **Irregular positioning of data in memory and low computing cost per data unit**

Ways to survive

- **Performing MVP iterating rows**
  exploiting parallelism in block operations (each block is at least 25 elements) and trying to reorder nodes into groups with similar operation sets.
- **Iterating coupling between nodes**
  relying on atomic operations.



This unstructured kind of mesh topology hurts

Thank you for attention

Our calculations have been performed on the
Lomonosov supercomputer at the Moscow State University,
MVS-100000 supercomputer at the Joint Supercomputer Center of the RAS,