Parallel CFD 2003
May 13-15, 2003 Moscow, RUSSIA

Direct Numerical Simulation of Natural Convection Flows Using PC Clusters

M.Soria, F.X.Trias, A.Oliva, C.D. Pérez-Segarra

Centre Tecnològic de Transferència de Calor (CTTC)
Lab. de Termotècnia i Energètica, Universitat Politècnica de Catalunya (UPC)
ETSEIT, c/ Colom 11, 08222 Terrassa, Spain
e-mail: manel@labtie.mmt.upc.es, web page: http://www.upc.es/lte

Presentation outline

- **1-Introduction**
  - Context
  - Governing equations
  - Pressure-velocity coupling. Poisson equation. Difficulties of incompressible flows
  - Loosely-coupled parallel computers
  - Alternatives to solve the Poisson equation
- **2-DSFD method to solve efficiently Poisson equations on PC clusters**
  - Fourier diagonalization
  - Direct Schur Decomposition
  - Benchmarks
- **3-Application example: DNS of turbulent natural convection in a cavity**
  - Problem definition
  - Verification of the code and simulations
  - Instantaneous flows
  - First-order statistics of the flow
  - Kinetic energy balances and spectroconsistent discretization
  - Second-order statistics of the flow
  - Comparision of 2D and 3D profiles
- **4-Conclusions**

Introduction - Context

- Navier-Stokes equations
- Turbulent, Incompressible flows
- Time-accurate integration
- Using primitive variables (i.e., p-$\mathbf{u}$)
- Pressure-velocity coupling solved with segregated approaches
- Structured and staggered meshes
- Finite-control volume formulation
- DNS and LES applications
- Symmetry-preserving (spectroconsistent) discretization
- The main problem from a parallel computing point of view is to **solve the incompressibility restriction**, expressed in terms of a **Poisson equation**

Introduction - Governing equations

Direct numerical simulation of turbulent natural convection flows.

Navier-Stokes coupled with energy transport equation.

$$
\begin{aligned}
\nabla \cdot \mathbf{u} &= 0 \\
\frac{\partial \mathbf{u}}{\partial t} &= -\mathbf{u} \cdot \nabla \mathbf{u} + \nu \nabla^2 \mathbf{u} - \frac{1}{\rho} \nabla p + \mathbf{f} \\
\frac{\partial T}{\partial t} &= -\mathbf{u} \cdot \nabla T + \alpha \nabla^2 T
\end{aligned}
$$

Where $f_z = g\beta T$

Assuming that the fluid is incompressible, Newtonian, of constant physical properties, using Boussinesq approximation to account for the density variations and neglecting thermal radiation.

Pressure velocity coupling (1/3)

Momentum equation is expressed as:

$$\frac{\partial \mathbf{u}}{\partial t} = \mathbf{R}\left(\mathbf{u}\right) - \frac{1}{\rho}\nabla p$$

Where $\mathbf{R} = -\mathbf{u}\cdot\nabla\mathbf{u} + +\nu\nabla^2\mathbf{u} + \mathbf{f}$

- **Spatial discretization:** Second order spectro-consistent discretization (fourth-order scheme is being implemented)
- **Time discretization:**
  - Central difference is used for the time derivative term
  - Fully explicit second order Adams-Bashforth scheme for $\mathbf{R}$
  - Implicit first-order Euler scheme for pressure-gradient term and mass-conservation equation

Pressure velocity coupling (2/3)

Time-discrete system to be solved:

$$\begin{aligned} \frac{\mathbf{u}^{n+1} - \mathbf{u}^n}{\Delta t} &= \frac{3}{2}\mathbf{R}^n - \frac{1}{2}\mathbf{R}^{n-1} - \frac{1}{\rho}\nabla p^{n+1} \\ \nabla \cdot \mathbf{u}^{n+1} &= 0 \end{aligned}$$

Predictor velocity is defined as $\mathbf{u}^p = \mathbf{u}^n + \Delta t \left[\frac{3}{2}\mathbf{R}^n - \frac{1}{2}\mathbf{R}^{n-1}\right]$

$\mathbf{u}^p$ is calculated with data available from previous time steps

Then, the unkown velocity is $\mathbf{u}^{n+1} = \mathbf{u}^p - \nabla\tilde{p}$

To evaluate $\tilde{p} = \frac{\Delta t}{\rho}p^{n+1}$, mass conservation equation is imposed:

$$\nabla \cdot \mathbf{u}^{n+1} = \nabla \cdot \mathbf{u}^p - \nabla \cdot (\nabla\tilde{p}) = 0$$

This leads to a **Poisson equation** $\nabla^2\tilde{p} = \nabla \cdot \mathbf{u}^p$ that must be solved to evaluate $\tilde{p}$ and then $\mathbf{u}^p$

This approach is similar in all the segregated formulations for incompressible flows

Pressure velocity coupling (3/3)

Why is Poisson equation so difficult ? A physical argument

Sound velocity is $c = \sqrt{\left(\frac{\partial p}{\partial \rho}\right)_s}$

Incompressibility $\longrightarrow \frac{\partial \rho}{\partial p} = 0 \longrightarrow c = \infty \longrightarrow$ local changes affect instantaneously all the domain

This behaviour is inhereited by the Poisson equation, without time derivatives

$$\nabla^2 \tilde{p} = \nabla \cdot \mathbf{u}^p$$

It has to be solved implicitly even if the rest of the formulation is explicit

At least one large linear equation system, coupling distant nodes, has to be solved per time step/iteration:

$$Ax = b$$

However, under certain conditions matrix A remains constant during all the problem

Our problem is actually to solve: $Ax_i = b_i \quad i = 1 \cdots M$ where $M \approx 10^{5 \cdots 7}$

Loosely coupled parallel computers

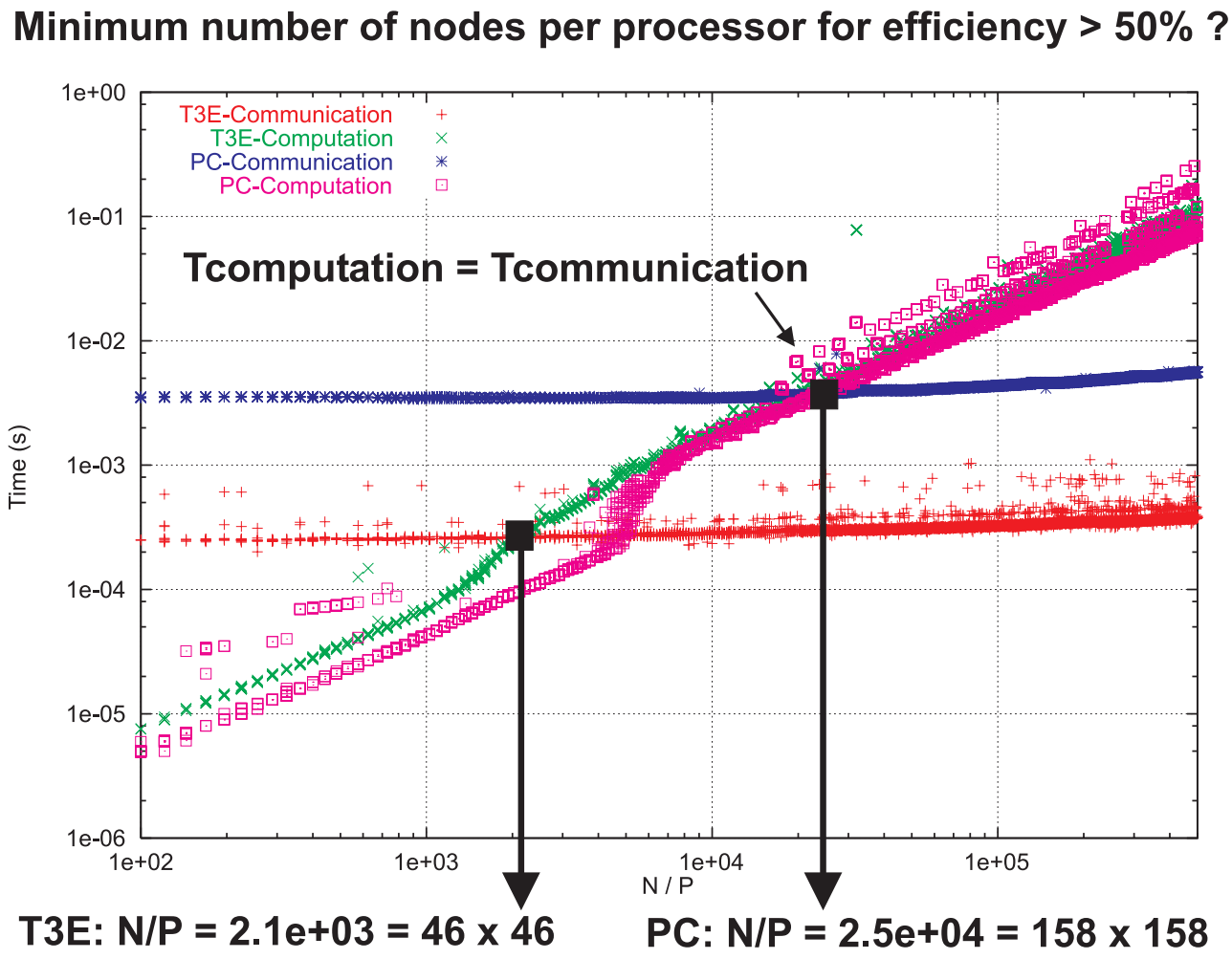
Low cost PC clusters are loosely coupled parallel computers:

- Good floating point power per-processor (excellent ratio **CPU power / cost**)
- Comparatively slow network (low bandwidth - **high latency**)

Parallel algorithms must be tolerant to slow networks to run efficiently on a PC cluster

For PCFD, unlikely other applications, latency is often the most critical problem

Matrix-vector product on a cluster and a Cray T3E



- Cray T3E (300 MHz) versus 900 MHz K7 (100 Mbit/s fast ethernet)
- $\frac{N}{P}$: number of nodes assigned to each processor.

Alternatives to solve the Poisson equation

The main challenge on a loosely-coupled system is the efficient solution of the Poisson equation

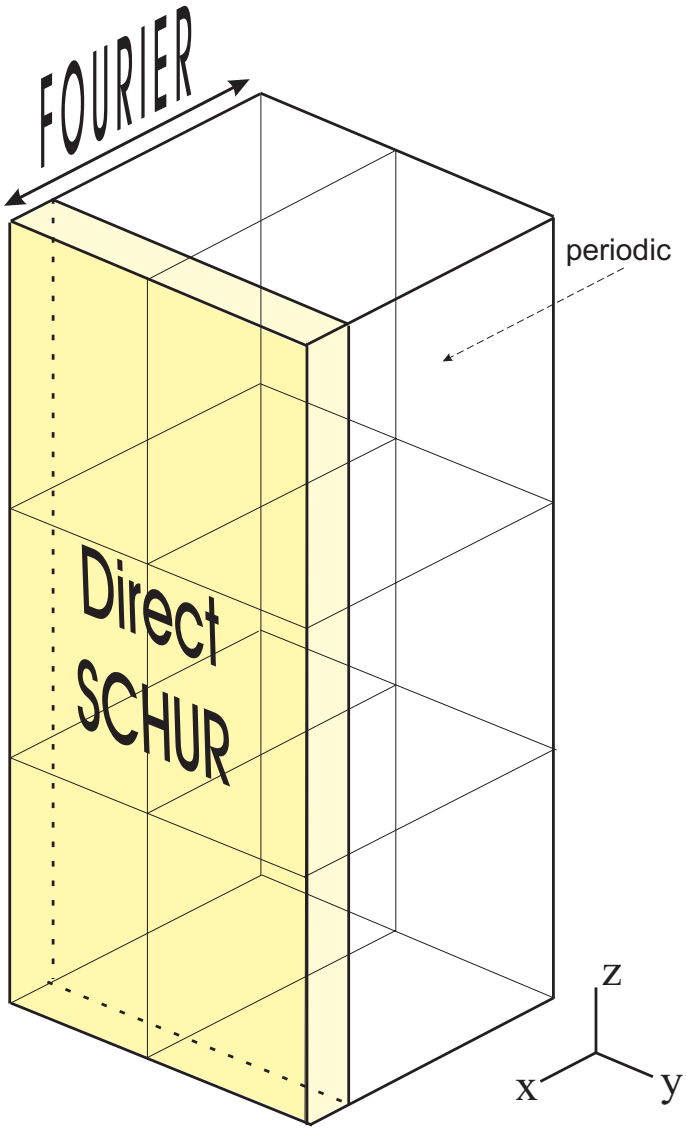
- MG algorithms
  - Very efficient on sequential systems
  - Need very low latency parallel computers; on high latency systems they must be combined with direct parallel solvers
- Krylov subspace algorithms
  - Parallelize well
  - Depend on good preconditioners to be efficient
  - Preconditioners tend to degradate with P
- Fast-Poisson solvers and **FFT-based methods** are restricted to determinated classes of problems
- **Schur Complement** methods
  - Iterative
  - Direct - **Use the fact that matrix A is constant**

DSFD - Overview (1/2)

- DSFD is a combination of a **FFT-based** method and a **Direct Schur** method:
  - **Fourier** diagonalization is applied to **reduce the heptadiagonal equation to a family of independent pentadiagonal equations** (this imposes certain restrictions)
  - The **pentadiagonal equations** are solved with a **Direct Schur** decompositiont method
- It is an **algebraic** approach: only the discrete equation system is used
- **Direct** method, based on a preprocessing stage where only matrix A is used
- After pre-processing, DSFD allows the solution of heptadiagonal equations using parallel computers **with just one message**
- DSFD is an interesting option to consider if:
  - A loosely coupled parallel computer is to be used
  - The matrix A has to be used many times with different right-hand-sides
  - The problem to be solved is periodic in one direction

E.g., for LES/DNS with Beowulf clusters

DSFD - Overview (1/2)



- The mesh must be uniform in x direction (usually periodic).
- Domain is decomposed only in directions y, z to avoid doing parallel FFT, that is very inefficient (in our context) on loosely coupled computers

Fourier Diagonalization (1/5)

Heptadiagonal system to be solved: $A^{3d}x^{3d} = b^{3d}$

$$a^{p}_{i,j,k}\, x^{p}_{i,j,k} + \Sigma_{nb} a^{nb}_{i,j,k}\, x^{nb}_{i,j,k} = b_{i,j,k}$$

To express it with **block matrices**, vectors x^{3d} and b^{3d} are divided into $N_y N_z$ **subvectors** with N_x components each:

$$x^{3d} = \left[x_{1,1}, x_{2,1}, \cdots x_{j,k}, \cdots, x_{N_y,N_z}\right]^t$$

where:

$$x_{j,k} = \left[x_{1,j,k}, x_{2,j,k}, \cdots, x_{N_x,j,k}\right]^t$$

Fourier Diagonalization (2/5)

$$\begin{bmatrix} A^p_{1,1} & A^n_{1,1} & \cdots & A^t_{1,1} & & & & \\ A^s_{2,1} & A^p_{2,2} & A^n_{2,1} & \cdots & A^t_{2,1} & & & \\ & & \ddots & & & & & \\ & A^b_{j,k} & \cdots & A^s_{j,k} & A^p_{j,k} & A^n_{j,k} & \cdots & A^t_{j,k} \\ & & & & & \ddots & & \\ & & & & A^b_{N_y,N_z} & \cdots & A^s_{N_y,N_z} & A^p_{N_y,N_z} \end{bmatrix} \begin{bmatrix} x_{1,1} \\ x_{2,1} \\ \vdots \\ x_{j,k} \\ \vdots \\ x_{N_y,N_z} \end{bmatrix} = \begin{bmatrix} b_{1,1} \\ b_{2,1} \\ \vdots \\ b_{j,k} \\ \vdots \\ b_{N_y,N_z} \end{bmatrix}$$

$$A^b_{j,k}\, x_{j,k-1} + A^s_{j,k}\, x_{j-1,k} + A^p_{j,k}\, x_{j,k} + A^n_{j,k}\, x_{j+1,k} + A^t_{j,k}\, x_{j,k+1} = b_{j,k}$$

- $A^n_{j,k}$, $A^s_{j,k}$, $A^t_{j,k}$ and $A^b_{j,k}$ are $N_x \times N_x$ diagonal matrices
- $A^p_{j,k}$ are $N_x \times N_x$ **circulant** tridiagonal matrices:

$$A^p_{j,k} = \begin{bmatrix} \beta & \alpha & & & \alpha \\ \alpha & \beta & \alpha & & \\ & & \ddots & & \\ \alpha & & & \alpha & \beta \end{bmatrix}$$

where $\alpha = a^w_{j,k} = a^e_{j,k}$ and $\beta = a^p_{j,k}$

Fourier Diagonalization (3/5)

All the circulant matrices of order N_x **have the same base of eigenvectors**

Let Q be the matrix whose columns are the eigenvectors of **all** the $A^p_{j,k}$

The product $x = Q\overline{x}$ **inverse Fourier transform**:

$$x_i = \frac{1}{2}\overline{x}_1 + \sum_{\nu=1}^{\frac{N_x}{2}-1}\left(\overline{x}_{2\nu}\cos\left(\nu i\frac{2\pi}{N_x}\right) + \overline{x}_{2\nu+1}\sin\left(\nu i\frac{2\pi}{N_x}\right)\right) + \frac{1}{2}\overline{x}_{N_x}(-1)^i \qquad i = 1\cdots N_x$$

and the product $\overline{x} = Q^{-1}x$ is a **direct Fourier transform**:

$$\begin{aligned}
\overline{x}_1 &= \frac{2}{N_x}\sum_{i=1}^{N_x} x_i \\
\overline{x}_{2\nu} &= \frac{2}{N_x}\sum_{i=1}^{N_x} x_i\cos\left(\nu i\frac{2\pi}{N_x}\right) \qquad \nu = 1\cdots\frac{N_x}{2}-1 \\
\overline{x}_{2\nu+1} &= \frac{2}{N_x}\sum_{i=1}^{N_x} x_i\sin\left(\nu i\frac{2\pi}{N_x}\right) \qquad \nu = 1\cdots\frac{N_x}{2}-1 \\
\overline{x}_{N_x} &= \sum_{i=1}^{N_x} x_i(-1)^i
\end{aligned}$$

Fourier Diagonalization (4/5)

All the matrices $A^p_{j,k}$ have tridiagonal form in the same base:

$$Q^{-1} A^p_{j,k} Q = \lambda_{j,k}$$

Where $\lambda_{j,k}$ is a **diagonal matrix** whose elements are:

$$\begin{aligned} \lambda_1 &= \beta + 2\alpha \\ \lambda_{2\nu} &= \lambda_{2\nu+1} = -4\alpha \sin^2\left(\frac{\nu\pi}{N_x}\right) + \beta + 2\alpha \qquad \nu = 1 \cdots \frac{N_x}{2} - 1 \\ \lambda_{N_x} &= \beta - 2\alpha \end{aligned}$$

Thus, expressing $x_{j,k}$ as $Q\overline{x}_{j,k}$ and premultiplying by Q^{-1}, the block equation becomes:

$$A^b_{j,k}\,\overline{x}_{j,k-1} + A^s_{j,k}\,\overline{x}_{j-1,k} + \lambda_{j,k}\,\overline{x}_{j,k} + A^n_{j,k}\,\overline{x}_{j+1,k} + A^t_{j,k}\,\overline{x}_{j,k+1} = \overline{b}_{j,k}$$

As the non-diagonal entries of matrices $A^p_{j,k}$ have been eliminated, **unknown $\overline{x}_{i,j,k}$ is only coupled with unknowns in the same plane i**

Fourier Diagonalization (5/5)

Selecting of the i component of each of the $N_y N_z$ block equations, we obtain a **penta-diagonal scalar equation system** in $\overline{x}$

$$a^b_{j,k}\,\overline{x}_{i,j,k-1} + a^s_{j,k}\,\overline{x}_{i,j-1,k} + \overline{a}^p_{i,j,k}\,\overline{x}_{i,j,k} + a^n_{j,k}\,\overline{x}_{i,j+1,k} + a^t_{j,k}\,\overline{x}_{i,j,k+1} = \overline{b}_{i,j,k}$$

The operations to be performed to solve the heptadiagonal equation system are:

1. Direct FFT. Calculate the $N_y N_z$ transformed right-hand-side sub-vectors, $\overline{b}_{j,k} = Q^{-1} b$
2. Solve the N_x decoupled pentadiagonal equation systems $\overline{A}_i \overline{x}_i = \overline{b}_i$.
3. Inverse FFT. Carry out the antitransformation of the $N_y N_z$ solution sub-vectors $\overline{x}_{j,k} = Q^{-1} x_{j,k}$

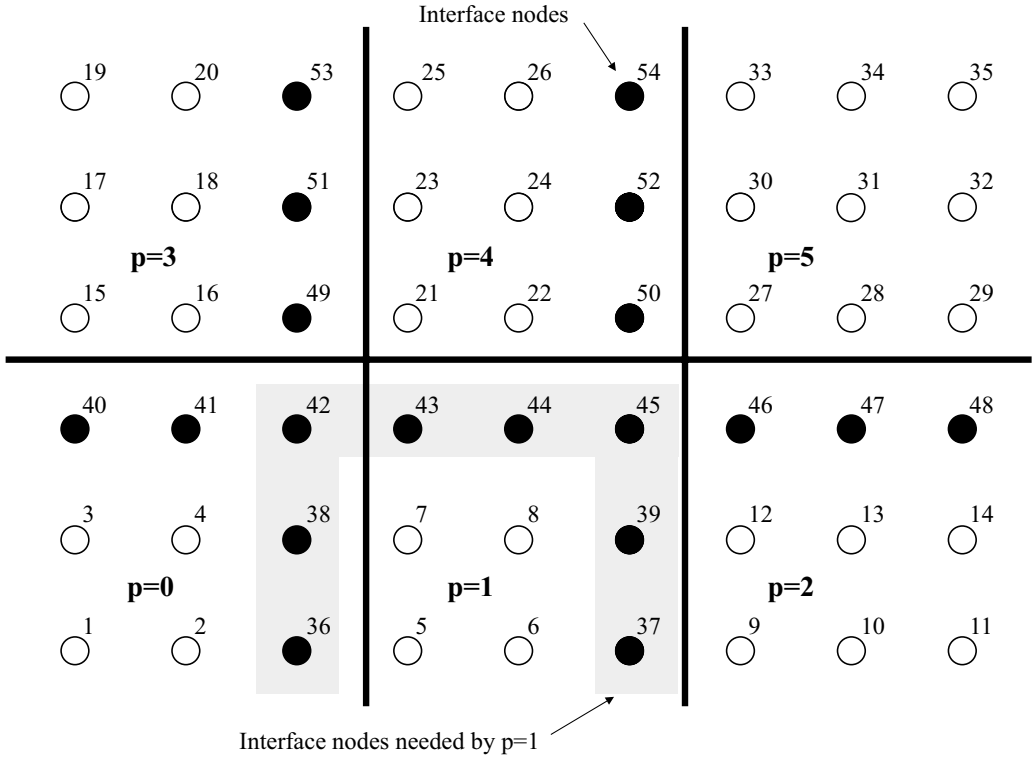
FFT are very cheap if done sequentially, but in our conditions they can not be carried out efficiently with loosely coupled parallel computers (to the knowledge of the authors)

This is why the domain is only decomposed in directions y, z

Solution of the pentadiagonal systems is done with a **Direct Schur Decomposition** using just one message for all the pentadiagonal equation systems

Direct Schur Decomposition - Main ideas (1/7)

Based on **non-overlapping** subdomains with **implicit treatment** of the interface



Only one all-to-all communication episode is needed

Direct Schur Decomposition - Overview (2/7)

No property of A (i.e., symmetry or positive-definiteness) or the underlying mesh is requiered

Each processor has to solve **twice** its own subdomain and **cooperate** to solve an interface equation to obtain **the exact solution** of the problem

After a pre-processing step **only one all-to-all communication** episode is needed.

It can be applied to 2D and 3D problems. Its **main limitation** for 3D problems is the **memory**. This is why the Fourier decomposition is carried out first

Compared with **matrix inversion**:

- $DSD(A)$ needs **far less storage memory** than A^{-1}
- Its **faster to preprocess** $DSD(A)$ than A^{-1}
- When $DSD(A)$ is available, it is **faster to solve** than the matrix-vector product $A^{-1}b$
- Direct Schur decomposition is **more appropiated** to parallel computers

Direct Schur Decomposition (3/7)

Each of the pentadiagonal systems is denoted by: $Ax = b$

Decomposition (a subvector is assigned to each processor): $x = [x_0, x_1, \cdots, x_{P-1}, x_s]^t$

After **reordering**, and **using block matrices**, the system is:

$$\begin{bmatrix} A_{0,0} & 0 & \cdots & 0 & A_{0,s} \\ 0 & A_{1,1} & \cdots & 0 & A_{1,s} \\ \vdots & & & \vdots & \\ 0 & 0 & \cdots & A_{P-1,P-1} & A_{P-1,s} \\ A_{s,0} & A_{s,1} & \cdots & A_{s,P-1} & A_{s,s} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_{P-1} \\ x_s \end{bmatrix} = \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_{P-1} \\ b_s \end{bmatrix}$$

Assuming non-singular local matrices, **Block Gaussian elimination** allows to express it as:

$$\begin{bmatrix} A_{0,0} & 0 & \cdots & 0 & A_{0,s} \\ 0 & A_{1,1} & \cdots & 0 & A_{1,s} \\ \vdots & & & \vdots & \\ 0 & 0 & \cdots & A_{P-1,P-1} & A_{P-1,s} \\ 0 & 0 & \cdots & 0 & \tilde{A}_{s,s} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_{P-1} \\ x_s \end{bmatrix} = \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_{P-1} \\ \tilde{b}_s \end{bmatrix}$$

Direct Schur Decomposition (4/7)

The **interface equation** $\tilde{A}_{s,s}x_s = \tilde{b}_s$ **allows to solve the interface nodes before the rest !**

$$\tilde{A}_{s,s} = A_{s,s} - \sum_{p=0}^{P-1} A_{s,p}A_{p,p}^{-1}A_{p,s}$$

$$\tilde{b}_s = b_s - \sum_{p=0}^{P-1} A_{s,p}A_{p,p}^{-1}b_p$$

After solving the interface equation, each local subproblem can be easily solved by a processor without additional communications:

$$A_{p,p}x_p = b_p - A_{p,s}x_s$$

Direct Schur Decomposition (5/7)

Evaluation of **interface equation** $\tilde{A}_{s,s}x_s = \tilde{b}_s$

$$\tilde{A}_{s,s} = A_{s,s} - \sum_{p=0}^{P-1} A_{s,p}A_{p,p}^{-1}A_{p,s} \qquad \tilde{b}_s = b_s - \sum_{p=0}^{P-1} A_{s,p}A_{p,p}^{-1}b_p$$

To avoid evaluation of $A_{p,p}^{-1}$:

1. Evaluation of interface matrix $\tilde{A}_{s,s} = A_{s,s} - \sum_{p=0}^{P-1} \tilde{A}_{s,s}^{p}$
   Contribution from processor p: $\tilde{A}_{s,s}^{p} = A_{s,p}A_{p,p}^{-1}A_{p,s}$
   Column c of $\tilde{A}_{s,s}^{p}$ is evaluated as:
   a) Solve t from $A_{p,p}t = [A_{p,s}]_c$
   b) $\left[\tilde{A}_{s,s}^{p}\right]_c \leftarrow A_{s,p}t$
2. Evaluation of right-hand-side vector $\tilde{b}_s = b_s - \sum_{p=0}^{P-1} \tilde{b}_s^p$
   Contribution from processor p: $\tilde{b}_s^p = A_{s,p}A_{p,p}^{-1}b_p$
   a) Solve t from $A_{p,p}t = b_p$
   b) $\tilde{b}_s^p \leftarrow A_{s,p}t$

Direct Schur Decomposition (6/7)

The algorithm has two parts:

1. **Preprocessing stage**
   Used only once
   As $\tilde{A}_{s,s}$ only depends on A (and not b), it is evaluated and **inverted** for each plane
2. **Solution stage**
   Called for each b to be solved ($10^{5\cdots7}$ times!)
   a) $\tilde{b}_s$ is evaluated. Each processor has to solve a local equation system to do so. **Here, one all-to-all message is needed.** The information for all the pentadiagonal equations is packed in the same all-to-all message
   b) Each processor performs the part of the matrix vector product $\left[\tilde{A}_{s,s}\right]^{-1} b_s$ needed to evaluate the nodes of x_s adjacent to its own subdomain
   c) Each processor solves a local equation to determinate its subvector x_p

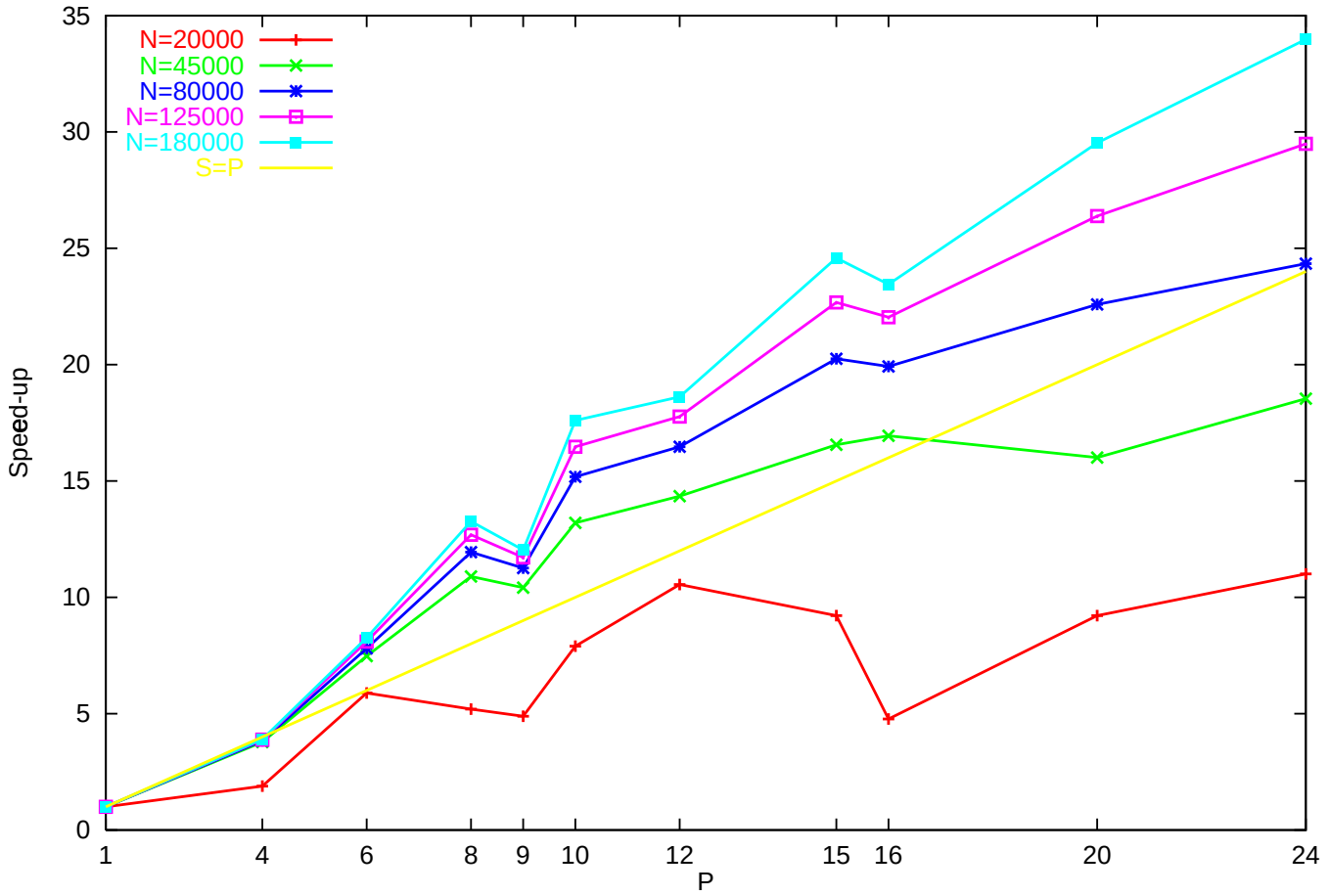
Direct Schur Decomposition (7/7)

The algorithm is not conceptually complex but its efficiency depends on certain details:

- In order to evaluate $\left[\tilde{A}_{s,s}\right]^{-1} b_s$, a **parallel block-LU algorithm** is used to improve the efficiency on loosely coupled systems
- After the block-LU decomposition, the **inverse is calculated row by row**. The block LU of $\tilde{A}^t_{s,s}$ is evaluated and then inverse and transpose operators are permuted
- **Distributed storage** of $\left[\tilde{A}_{s,s}\right]^{-1}$. Each processor only stores the rows of the inverse needed to perform its part of $\left[\tilde{A}_{s,s}\right]^{-1} b_s$.
- The local equation systems are solved using a **band-LU algorithm**
- Different **matrix data structures** are needed:
  - Matrices $A_{s,p}$, $A_{p,s}$, $A_{s,s}$ are treated as **sparse**
  - Matrices $A_{p,p}$ are **banded**
  - Matrix $\tilde{A}_{s,s}$ is treated as a **block matrix** and **distributed by rows** at the different processors
  - Matrix $\left[\tilde{A}_{s,s}\right]^{-1}$ is treated as **dense** and **stored by rows** at the different processors

Benchmark (1/4) - Speedup of DSD on a PC cluster

900 MHz K7 processors; Switched 100 Mbits/s network

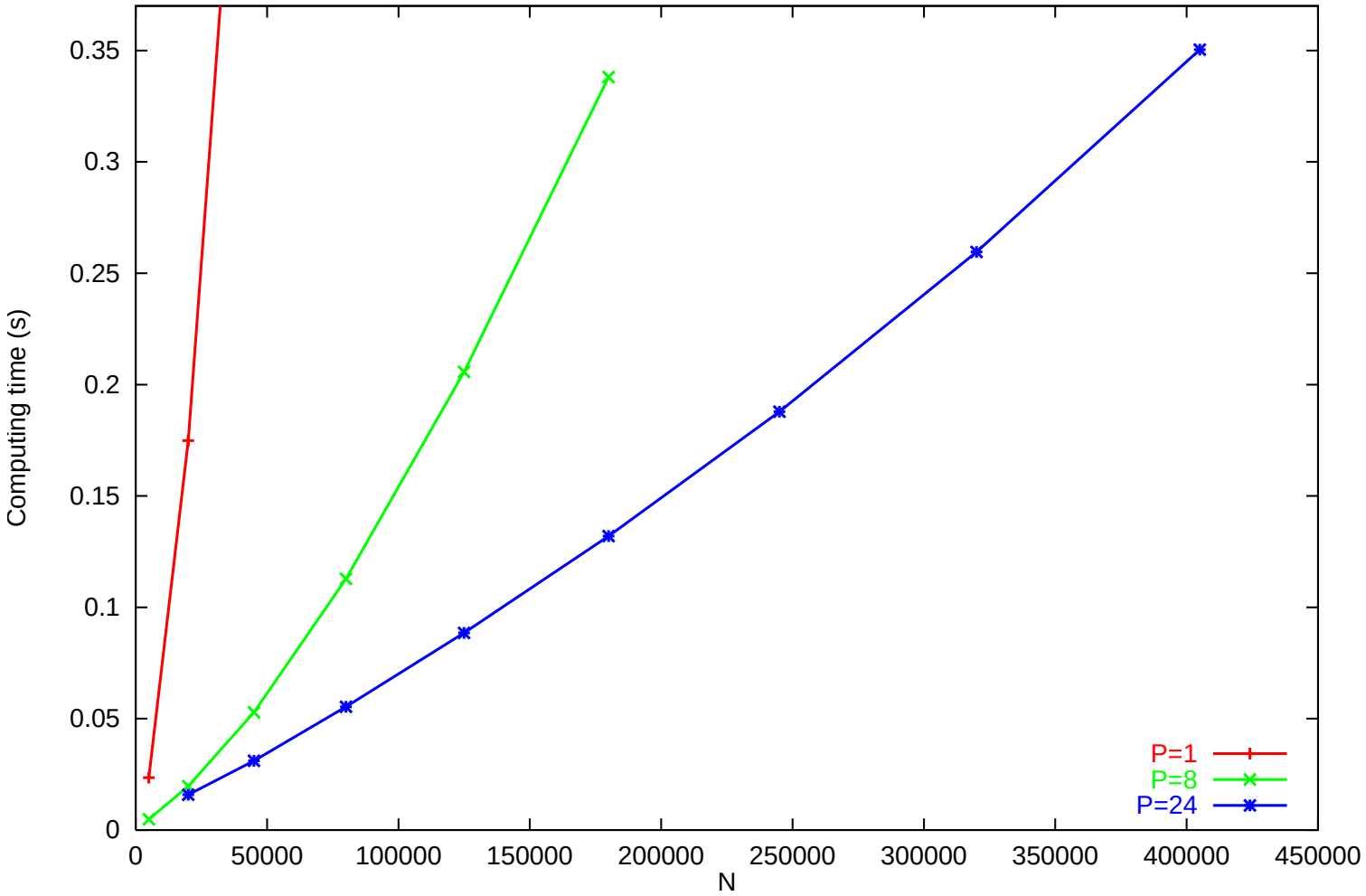


The cost of DSFD is roughly N_x times the cost of one pentadiagonal equation Irregular behaviour for a fixed N is due to different bandwiths of local problems depending on P.

Super-linear S is due to the non-linear cost of local band-LU solver and low cost of interface solution and communications.

Benchmark (2/4) - DSD solution time on a PC cluster

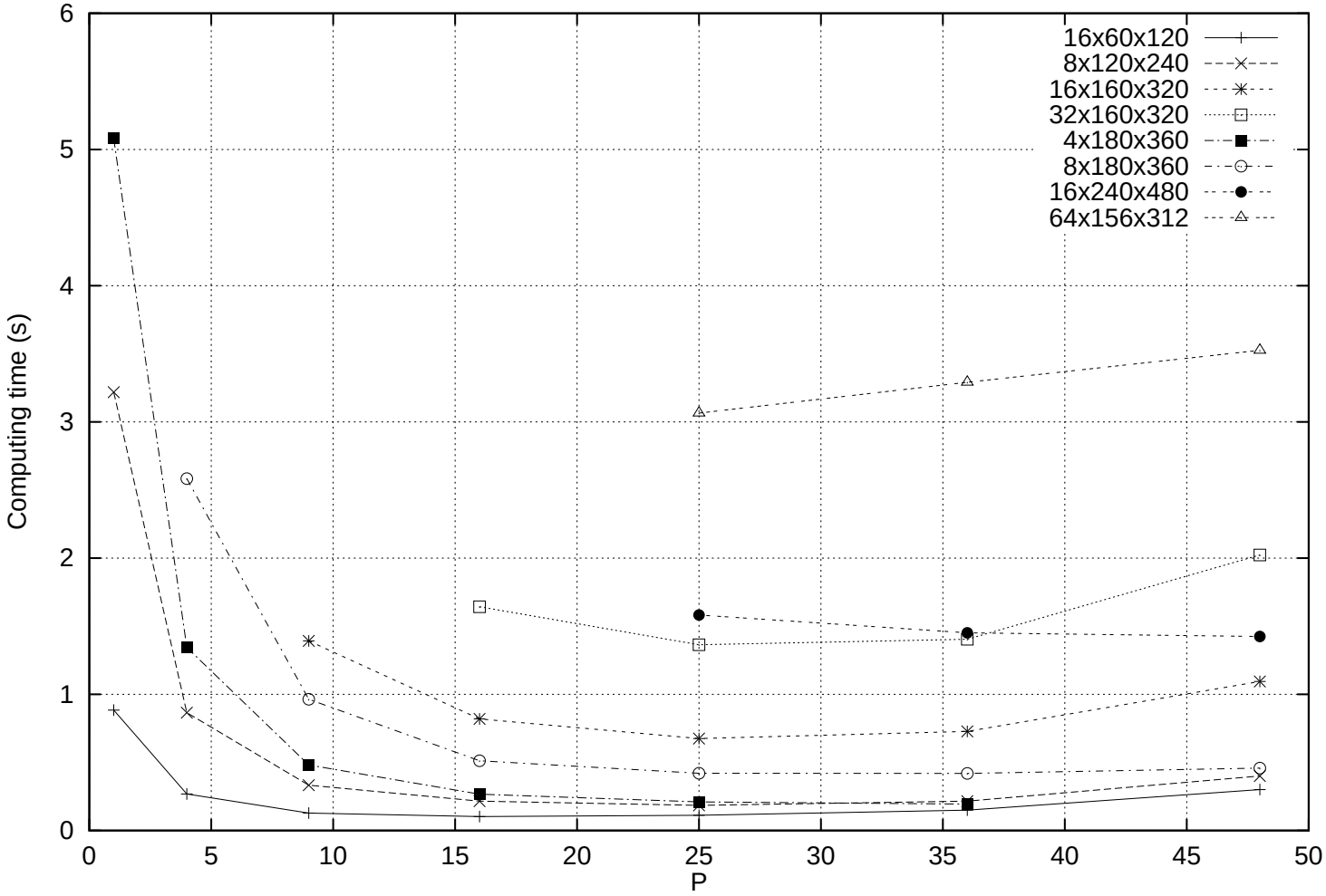
900 MHz K7 processors; Switched 100 Mbits/s network



For $N = 125 \times 10^3$, $P = 24$ Direct Shur Complement is ≈ 30 times faster than sequential ACM multigrid with $\epsilon^* = 10^{-3}$

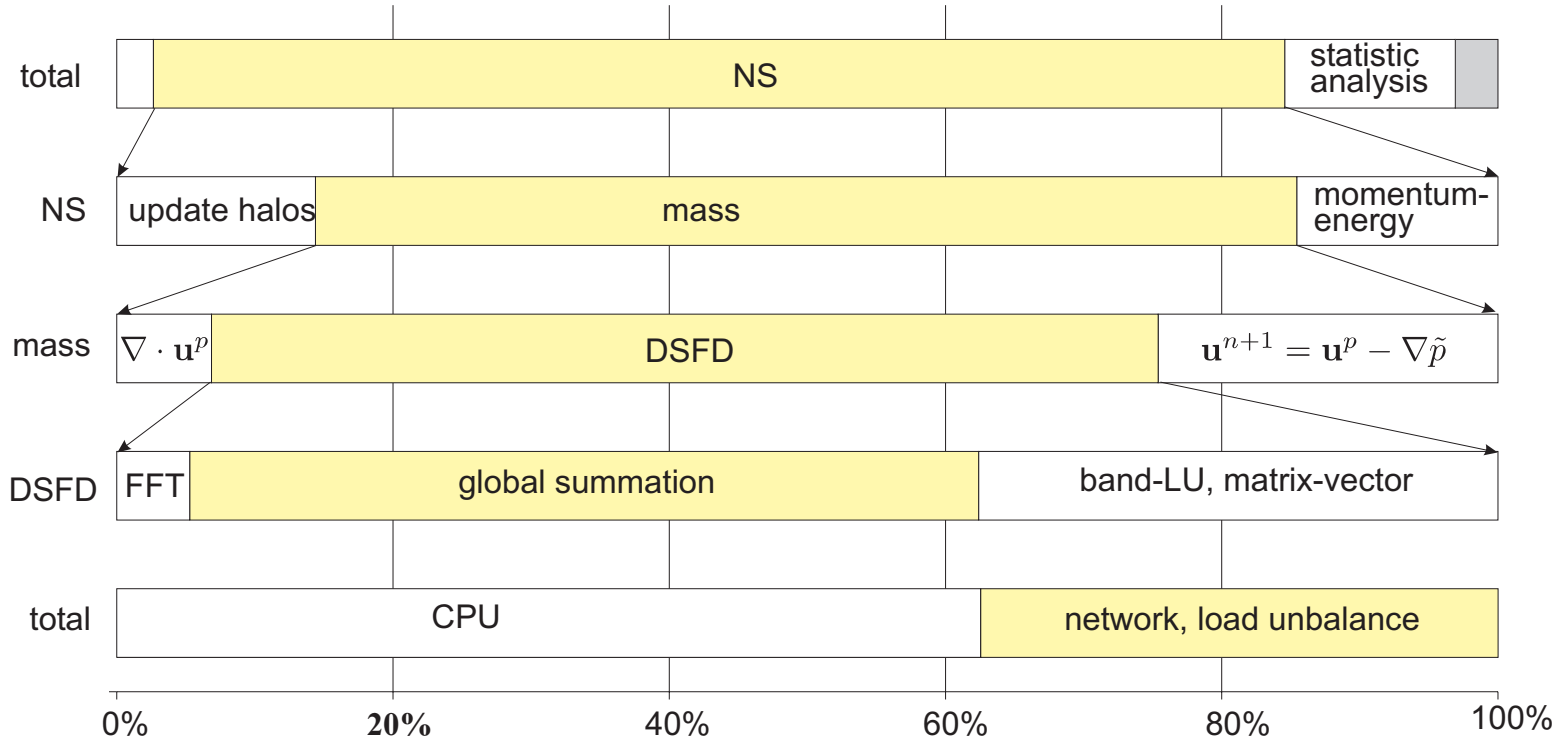
Benchmark (3/4) - DSFD solution time on a PC cluster

900 MHz K7 processors; Switched 100 Mbits/s network



For $N = 3{,}1 \times 10^6$, each equation can be solved almost to machine accuracy in about three seconds

Benchmark (4/4) - Decomposition of total time with 36 processors



Natural Convection Flows in Cavities

The **first aplication** of the new DNS code based on DSFD and spectro-consistent discretizations has been a **natural convection** problem

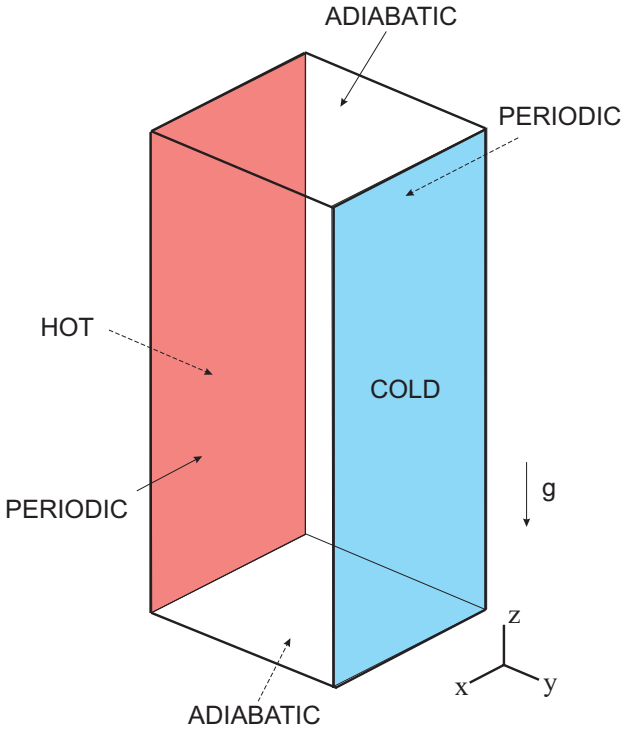
The majority of the natural convection problems in closed cavities can be classified in three groups:

- Cavities where the flow is due to **internal heat generation**
- Cavities **heated from below** (Rayleigh-Bénard configuration)
- **Differentially heated cavities** (DHC)
  - Laminar flows: 2D - 3D
  - Turbulent flows:
    - RANS models
    - LES models
    - **DNS**: in **all previous DNS studies** a 2D behavior has been assumed

Our goal: clarify the effects of the assumption of 2D over the statistics of a turbulent DHC flow

Differentially Heated Cavity (DHC) Problem (1/2)

General schema



Boundary conditions:

- **Isothermal vertical walls**
- **Adiabatic horizontal walls**
- **Periodic** boundary conditions in the x direction, orthogonal to the main flow

Dimensionless governing numbers:

- $Ra_z = \frac{\beta \Delta T L_z^3 g}{\alpha \nu}$
- $Pr = \frac{\nu}{\alpha}$
- Height aspect ratio $A_z = \frac{L_z}{L_y}$
- Depth aspect ratio $A_x = \frac{L_x}{L_y}$

Differentially Heated Cavity (DHC) Problem (2/2)

Definition of our problem

Dimensionless governing number values:

$$Ra_z = 6{,}4 \times 10^8$$
$$Pr = 0{,}71$$
$$A_z = 4$$
$$A_x = 1$$

Basic motivations:

- This case is an extension to 3D of one of the 2D problems **studied in detail in the literature**
- Periodic boundary conditions allow to study the **3D effects due to instability** of the main flow and not to the boundary conditions
- This variant is **computationally more convenient** since the resulting flow has no boundary layers in the x direction:
  - The mesh can be coarser and uniform in x direction
  - As the mesh is uniform Fourier-based methods, such as **DSFD** can be used

Code and Simulation Verifications

Two basic verifications are necessary in order to ensure the numerical results are reasonably close to the analytic solution

1. Verification of the code
   - Show that the code solves the governing equations with the expected order of accuracy
   - A method (MMS) based on the systematic discretization convergence tests using analytic solutions has been used
2. Verification of the simulation
   - In order to select the mesh size, integration period, depth lenhgth, etc, **a compromise between accuracy and time computing must be made**
   - It is important to **evaluate if the numerical results obtained are reasonably close to the asymptotic solution**
   - In our case, as the flow is chaotic, we are actually checking if the **statistics of the flow** are close to the asymptotic solution

Code verification using the MMS method (1/4)

Assume that the functions for the integration of mass and momentum equations are to be verified

MMS is carried out in three basic steps:

1. Generation of the **analytic velocity and source terms** fields:
   - An arbitrary analytic non-trivial function $\mathbf{u}_a\left(\mathbf{x}, t\right)$ which accomplishes selected boundary conditions and incompressibility constraint ($\nabla \cdot \mathbf{u}_a = 0$) is chosen
   - The source term $\mathbf{f}_a\left(\mathbf{x}, t\right)$ that would match with the arbitrary solution assuming that pressure gradient is null, is calculated analytically as:

$$\mathbf{f}_a = \frac{\partial \mathbf{u}_a}{\partial t} + \mathbf{u}_a \cdot \nabla \mathbf{u}_a - Pr \nabla^2 \mathbf{u}_a$$

2. Obtention of the **numerical solution**:
   - The analytic source term $\mathbf{f}_a\left(\mathbf{x}, t\right)$ is evaluated at the discretization nodes and then used as input data for the numerical code.

Code verification using the MMS method (2/4)

3. Evaluation of the **numerical errors** and **order of accuracy** schemes verification:
   - The discrete numerical solution $\mathbf{u}_n(\mathbf{x}, t)$ is compared with $\mathbf{u}_a(\mathbf{x}, t)$:

$$\|\mathbf{e}\|_\infty = \|\mathbf{u_a} - \mathbf{u_n}\|_\infty = Max \mid \mathbf{u_a}(x_i, y_j, z_k, t_n) - \mathbf{u_n}(x_i, y_j, z_k, t_n) \mid$$

   - This measure is repeated for systematically refined grids.
     As $\|\mathbf{e}\|_\infty$ must tend to zero with an expected order of accuracy

$$\|\mathbf{e}\|_\infty = C_t \Delta t^{p_t} + C_h \Delta h^{p_h} + H.O.T$$

     numerical results of p_t and p_h can be evaluated separately and compared with the theoretical values

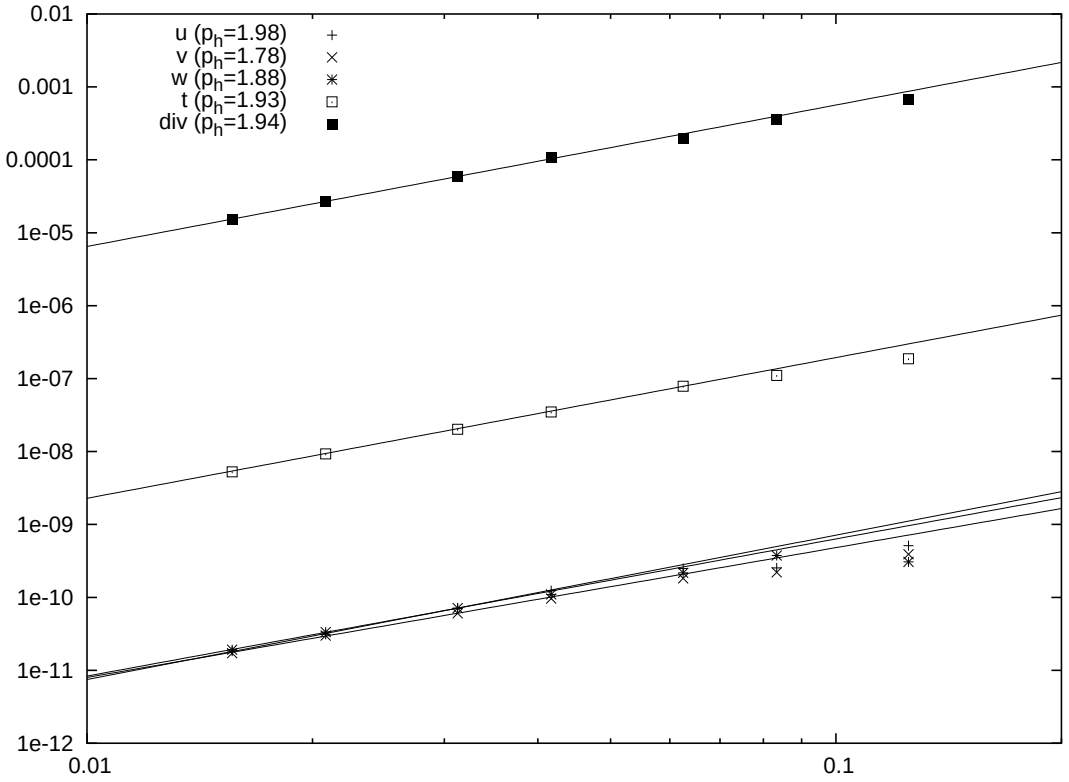
For the case of the mass-momentum system, since we assumed a null ∇p field, an additional verification is necessary

For each $\mathbf{u}^p$ there is **only one** p scalar field (except a constant) such that $\mathbf{u} = \mathbf{u}^p - \nabla p$ is divergence-free ($\nabla \cdot \mathbf{u}$),

Thus, if the velocity fields evaluated by the code are divergence free, the field ∇p must be correct

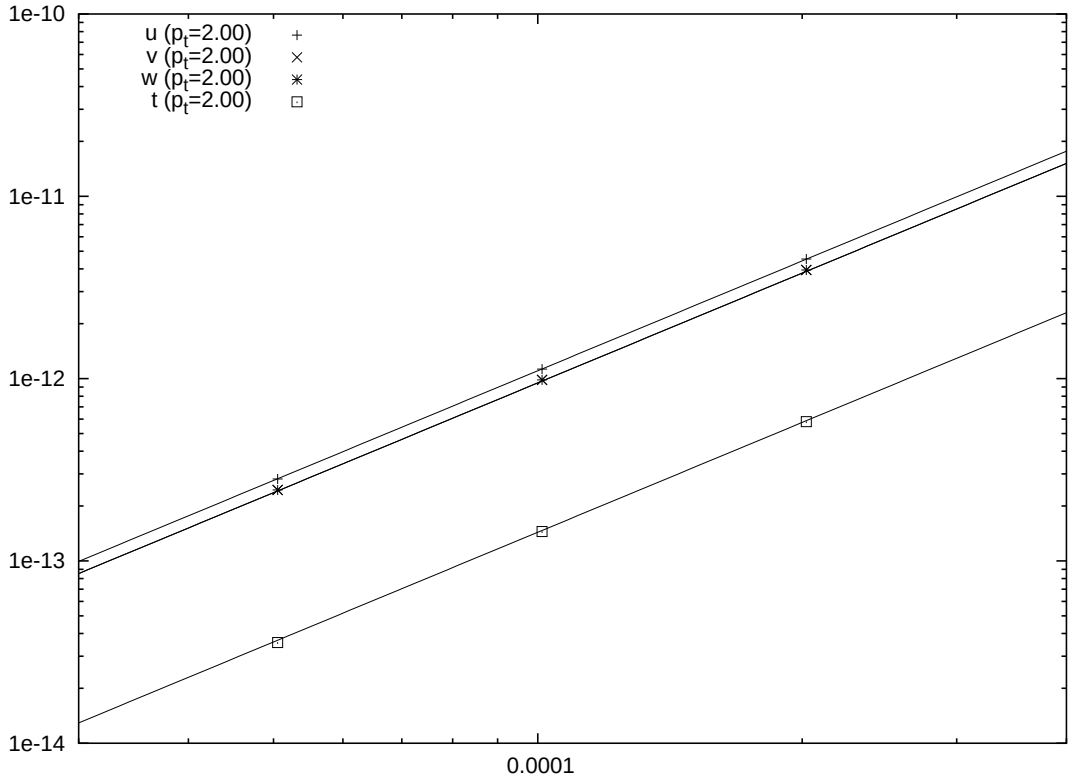
However, it must be verified that the operator $\nabla \cdot$ is correct (i.e., with the expected order of accuracy)

Code verification using the MMS method (3/4)



Error versus mesh size $\Delta h = L_x/N_x$ for meshes concentrated in axis y and z with $N_x = N_y = N_z$. In parentheses the temporal order of accuracy (p_h).

Code verification using the MMS method (4/4)



Errors versus time step Δt. In parentheses the temporal order of accuracy (p_t).

Verification of the simulation (1/5)

After the absence of errors in the code has been ensured, it is necessary to determinate if **the simulation parameters used allow to obtain numerical results reasonably close to the asymptotic solution**

The parameters are:

- Mesh size
- Mesh concentration
- Time step
- Domain length in the direction orthogonal to the main flow (L_x)
- Beginning of the averaging period (t_0)
- Integration time to evaluate the flow statictics (Δt_a)

For all them, a compromise between **accuracy** and **computing time** must be accepted

Verification of the simulation (2/5)

A total of **eleven** 2D and 3D simulations have been carried out and compared in order to estimate the accuracy of the results

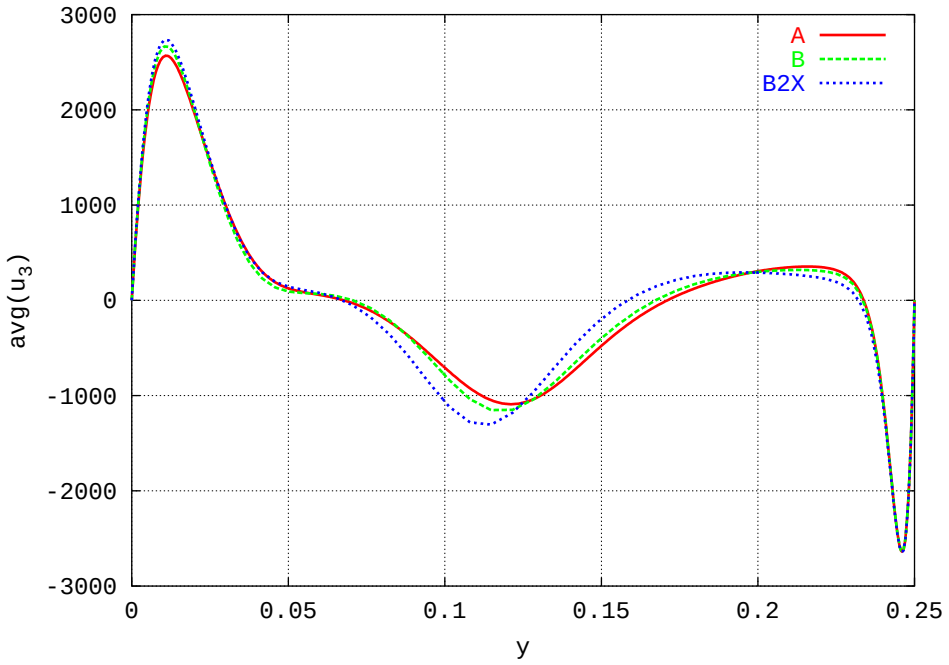
Meshes between $4{,}8 \times 10^4$ to $3{,}1 \times 10^6$ nodes have been used

	N_x	N_y	N_z	L_x	γ_y	γ_z	Δy_{min}	Δy_{max}	Δz_{min}	Δz_{max}	$\overline{\Delta t}$
A	64	156	312	1	1.5	1.5	$4{,}95 \times 10^{-4}$	$2{,}70 \times 10^{-3}$	$9{,}75 \times 10^{-4}$	$5{,}35 \times 10^{-3}$	$4{,}73 \times 10^{-8}$
B	32	78	156	1	2.0	2.0	$5{,}08 \times 10^{-4}$	$6{,}83 \times 10^{-3}$	$9{,}75 \times 10^{-4}$	$1{,}35 \times 10^{-2}$	$5{,}14 \times 10^{-8}$
B2X	64	78	156	2	2.0	2.0	$5{,}08 \times 10^{-4}$	$6{,}83 \times 10^{-3}$	$9{,}75 \times 10^{-4}$	$1{,}35 \times 10^{-2}$	$5{,}15 \times 10^{-8}$
C	16	39	78	1	2.0	2.0	$1{,}10 \times 10^{-3}$	$1{,}40 \times 10^{-2}$	$2{,}03 \times 10^{-3}$	$2{,}73 \times 10^{-2}$	$6{,}25 \times 10^{-8}$
A2D	-	156	312	-	1.5	1.5	$4{,}95 \times 10^{-4}$	$2{,}70 \times 10^{-3}$	$9{,}75 \times 10^{-4}$	$5{,}35 \times 10^{-3}$	$4{,}89 \times 10^{-8}$
B2D	-	78	156	-	2.0	2.0	$5{,}08 \times 10^{-4}$	$6{,}83 \times 10^{-3}$	$9{,}75 \times 10^{-4}$	$1{,}35 \times 10^{-2}$	$3{,}17 \times 10^{-8}$
C2D	-	39	78	-	2.0	2.0	$1{,}10 \times 10^{-3}$	$1{,}40 \times 10^{-2}$	$2{,}03 \times 10^{-3}$	$2{,}73 \times 10^{-2}$	$6{,}25 \times 10^{-8}$
AB2D	-	156	312	-	2.0	2.0	$2{,}44 \times 10^{-4}$	$3{,}38 \times 10^{-3}$	$4{,}78 \times 10^{-4}$	$6{,}70 \times 10^{-3}$	$1{,}19 \times 10^{-8}$
AA2D	-	218	438	-	1.5	1.0	$3{,}48 \times 10^{-4}$	$1{,}90 \times 10^{-3}$	$1{,}26 \times 10^{-3}$	$3{,}00 \times 10^{-3}$	$2{,}42 \times 10^{-8}$

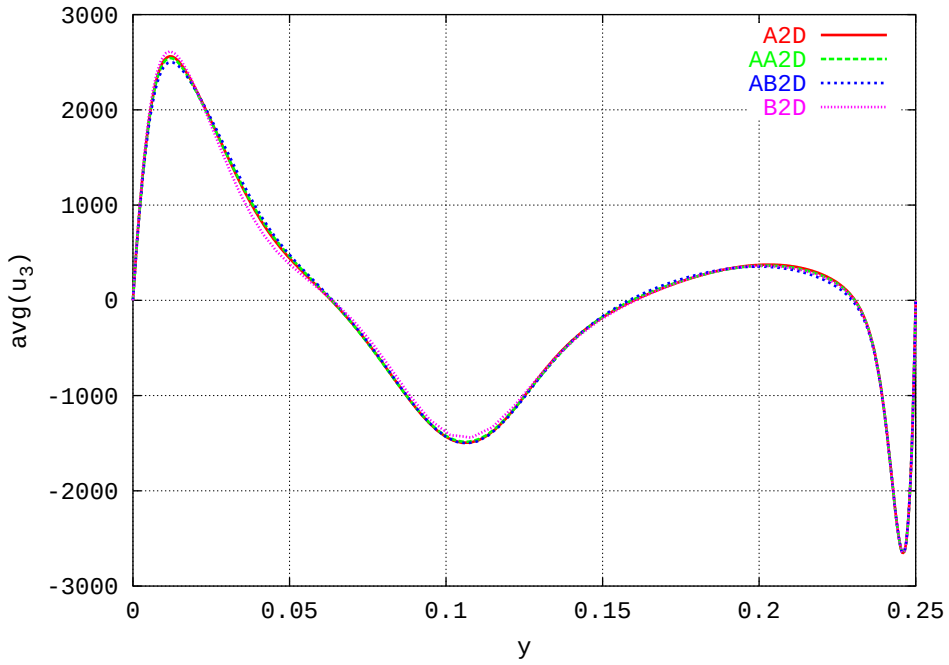
Verification of the simulation (3/5)

First-order statistics

3D



2D

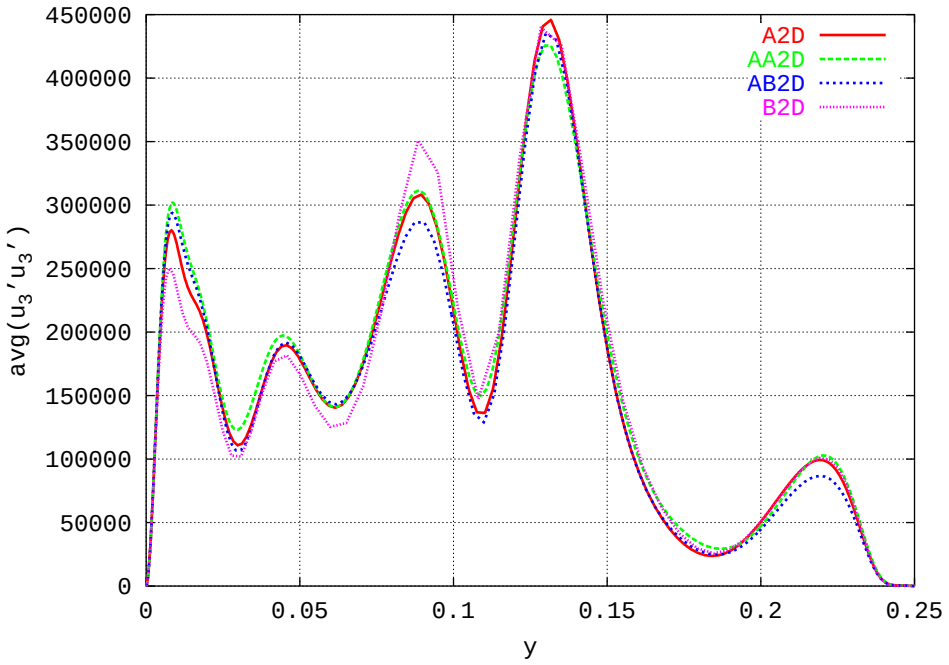


Comparison of $\overline{u_3}$ in the section $z = \frac{15}{16} L_z$

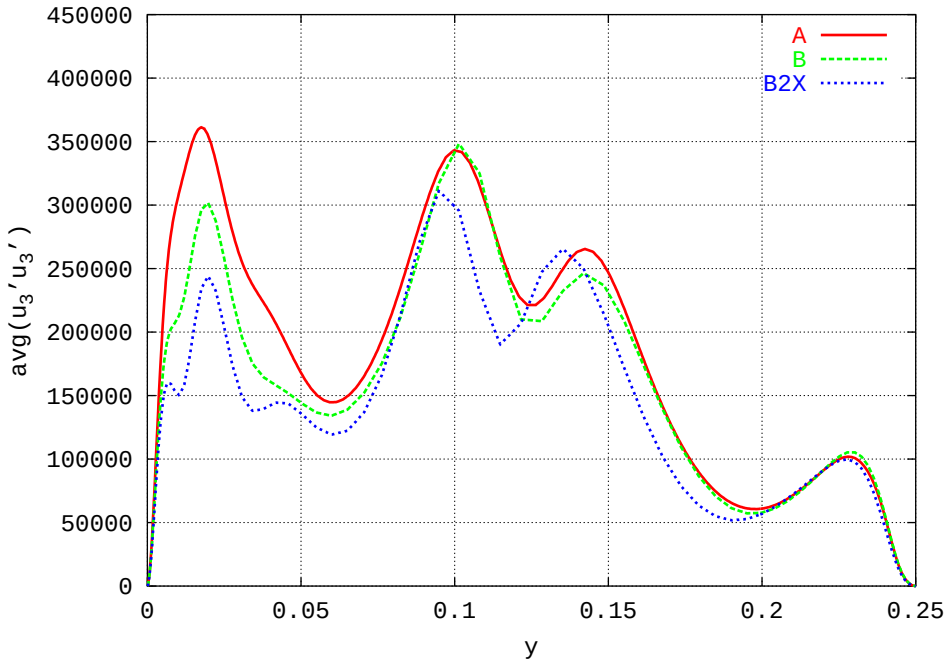
Verification of the simulation (4/5)

Second-order statistics

3D



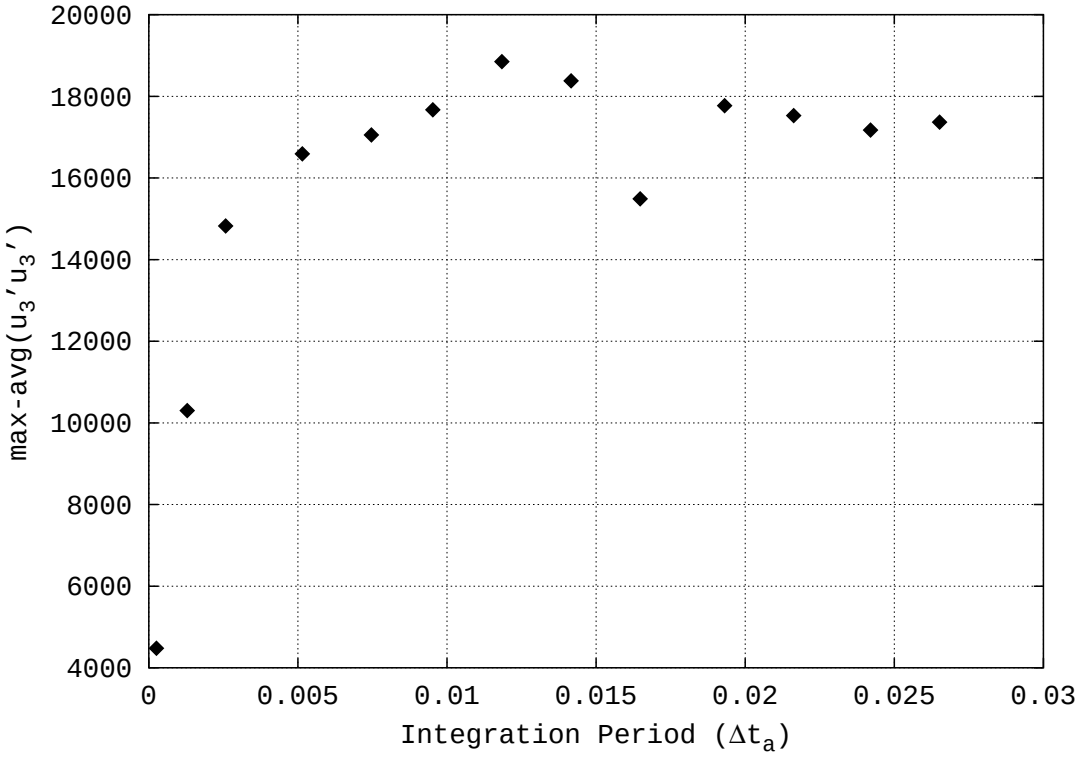
2D



Comparison of $\overline{u_3' u_3'}$ in the section $z = \frac{15}{16} L_z$ (where the discrepancies are largest).

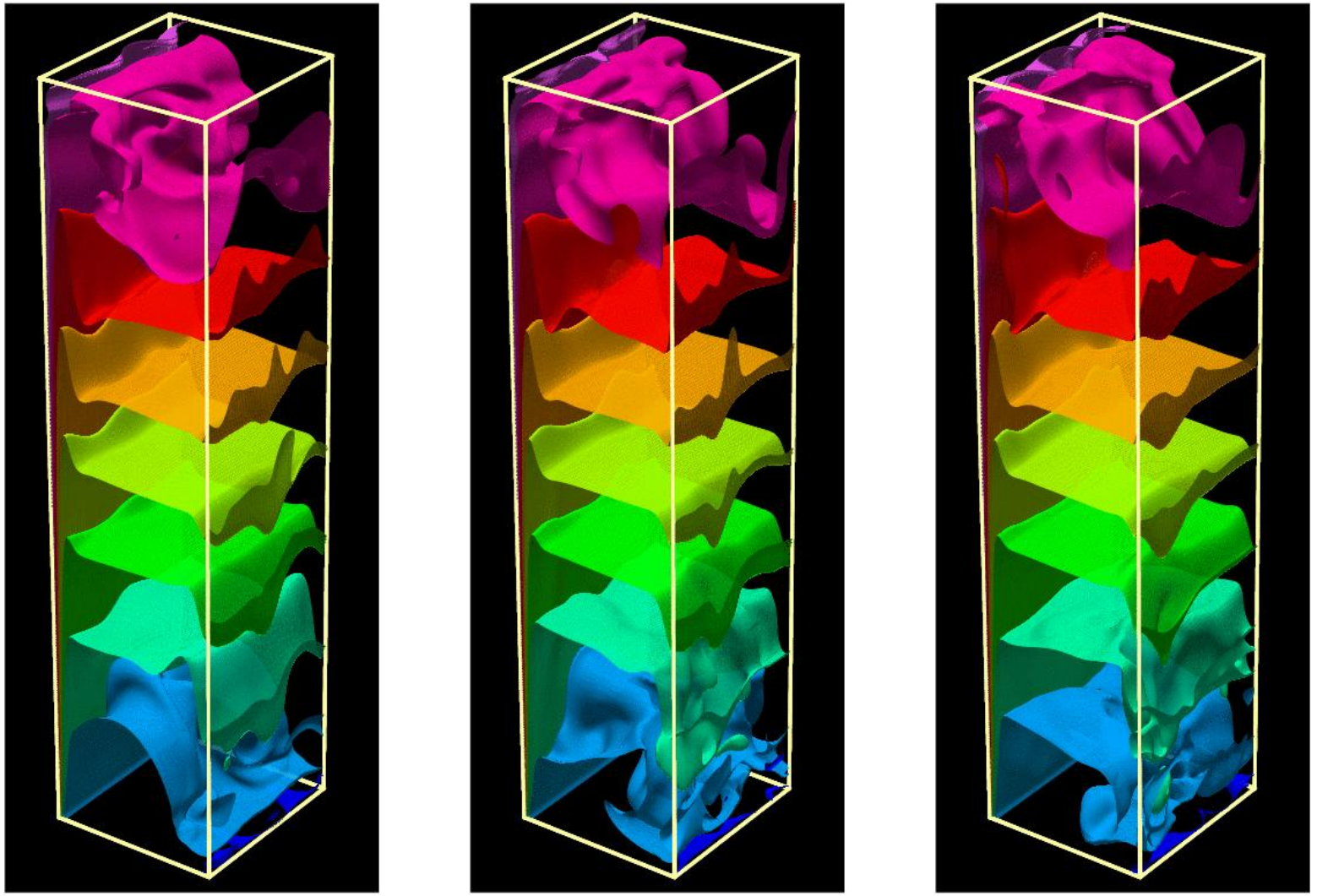
Verification of the simulation (5/5)

Estimation of the integration period Δt_a needed to evaluate first- and second-order statistics.



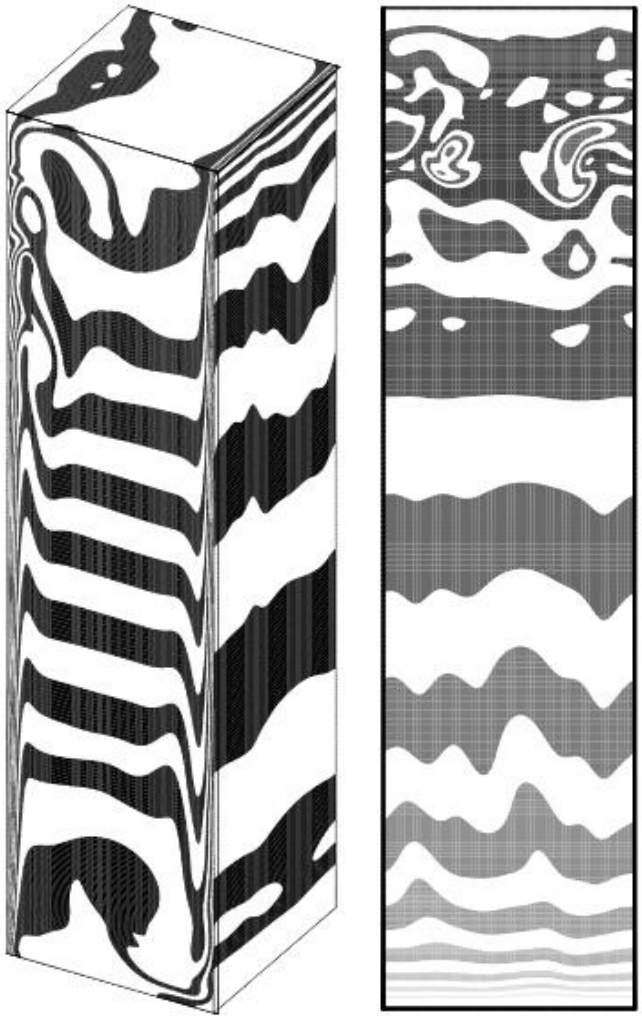
Maximum $\overline{u_3' u_3'}$ at the central z plane versus Δt_a

Instantaneous Isotherms

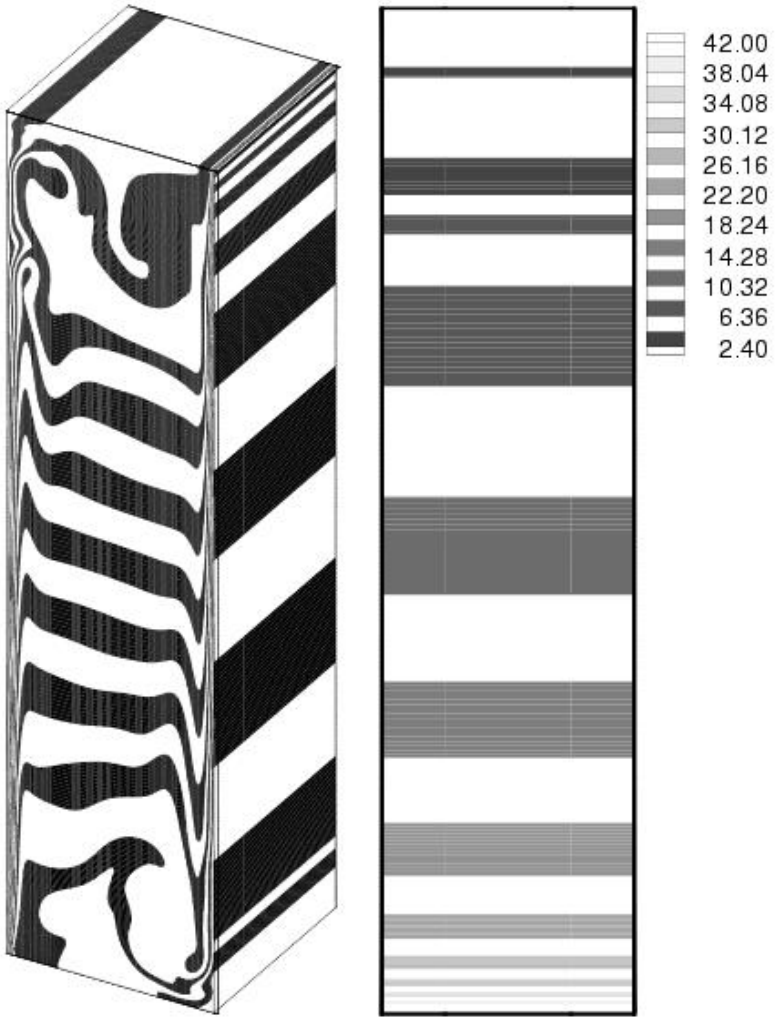


Temperature and Nusselt Instantaneous Fields

3D

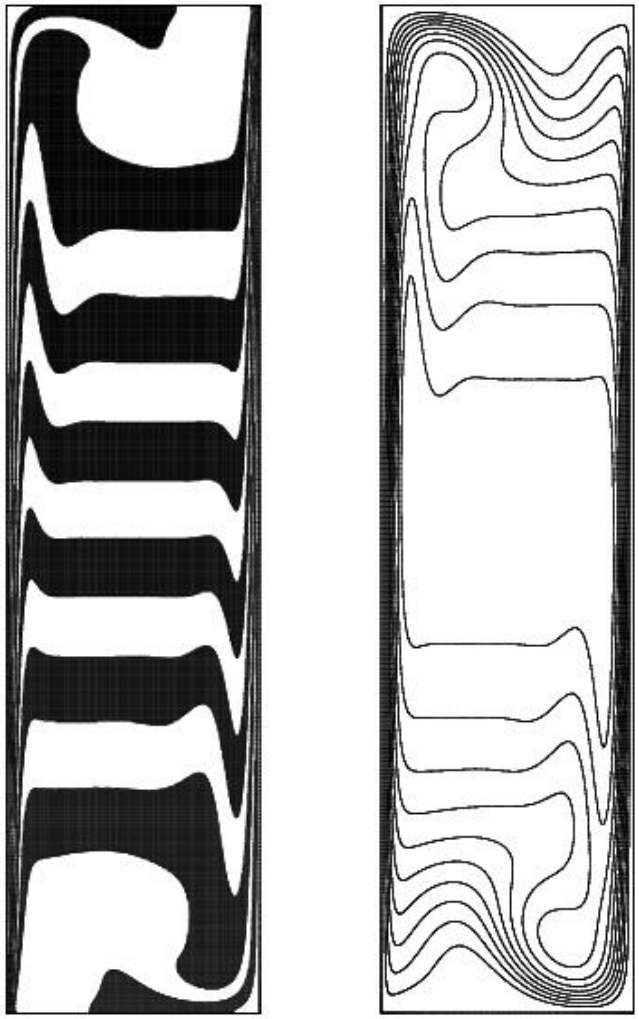


2D

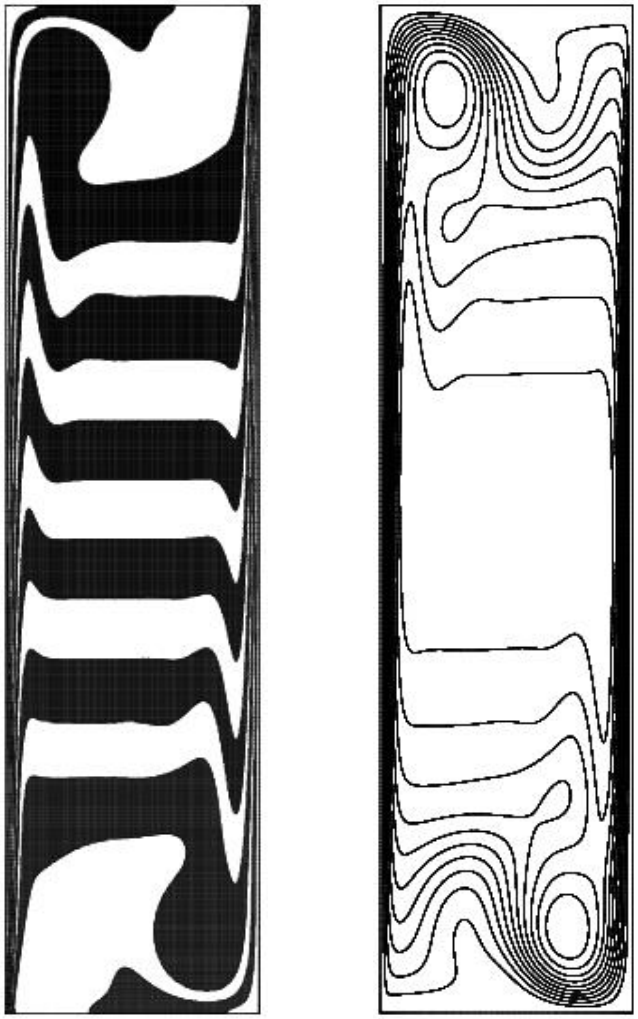


Averaged Temperature Field and Streamlines

3D



2D



Kinetic Energy Balances (1/3)

The **transport equation for kinetic energy**, $e = \frac{1}{2}\mathbf{u}\cdot\mathbf{u}$ is obtained from the scalar product of velocity vector and momentum equation,

$$\frac{\partial e}{\partial t} = -\nabla\cdot(e\mathbf{u}) + Pr\,\nabla\cdot\left[\mathbf{u}\cdot\left(\nabla\mathbf{u}+\nabla\mathbf{u}^t\right)\right] - Pr\,\phi - \nabla\cdot(p\mathbf{u}) + \mathbf{u}\cdot\mathbf{f}$$

Where $\phi(\mathbf{u}) = \left(\nabla\mathbf{u}+\nabla\mathbf{u}^t\right) : \nabla\mathbf{u}$. The term $-Pr\,\phi$ is the *kinetic energy dissipation ratio* that arises from the viscous forces term $Pr\,\nabla^2\mathbf{u}$.

Integration of previous expression in the domain Ω yields:

$$\frac{d}{dt}E = \int_{\partial\Omega}\left[-e\mathbf{u} + Pr\,\mathbf{u}\cdot\left(\nabla\mathbf{u}+\nabla\mathbf{u}^t\right) + p\mathbf{u}\right]\cdot d\mathbf{S} + \int_{\Omega}\left[\mathbf{u}\cdot\mathbf{f} - Pr\,\phi\right]d\Omega$$

Where $E = \int_\Omega e d\Omega$ is the total kinetic energy.

Kinetic Energy Balances (2/3)

For our **boundary conditions**, the surface integral is null:

$$\frac{d}{dt}E = \int_\Omega \left[\mathbf{u}\cdot\mathbf{f} - Pr\,\phi\right] d\Omega = \int_\Omega \left[RaPrTu_3 - Pr\phi\right] d\Omega$$

The only terms that contribute to the evolution of the total kinetic energy arise from:

- The **viscous term**: $-Pr\phi$, that necessarily **dissipates kinetic energy** to thermal energy (as $\phi \geq 0$).
- The **body force term**: $\mathbf{u}\cdot\mathbf{f}$ that can either **generate or dissipate kinetic energy**.

Kinetic Energy Balances (3/3)

Averaging for a long enough Δt_a, a global kinetic energy balance is obtained, that (per volume unit) is:

$$\underbrace{\frac{Ra}{V}\int_\Omega \left[\overline{u_3}\,\overline{T} + \overline{u_3' T'}\right] d\Omega}_{\overline{E_g}} = \underbrace{\frac{1}{V}\int_\Omega \left[\phi\left(\overline{\mathbf{u}}\right) + \overline{\phi\left(\mathbf{u}'\right)}\right] d\Omega}_{\overline{E_d}}$$

That is, for a **statistically stationary flow**:

- $\overline{E_g}$, the *averaged kinetic energy generation rate* (only due to the bouyancy forces in our case)
- $\overline{E_d}$, the *averaged kinetic energy dissipation rate* due to viscous forces.

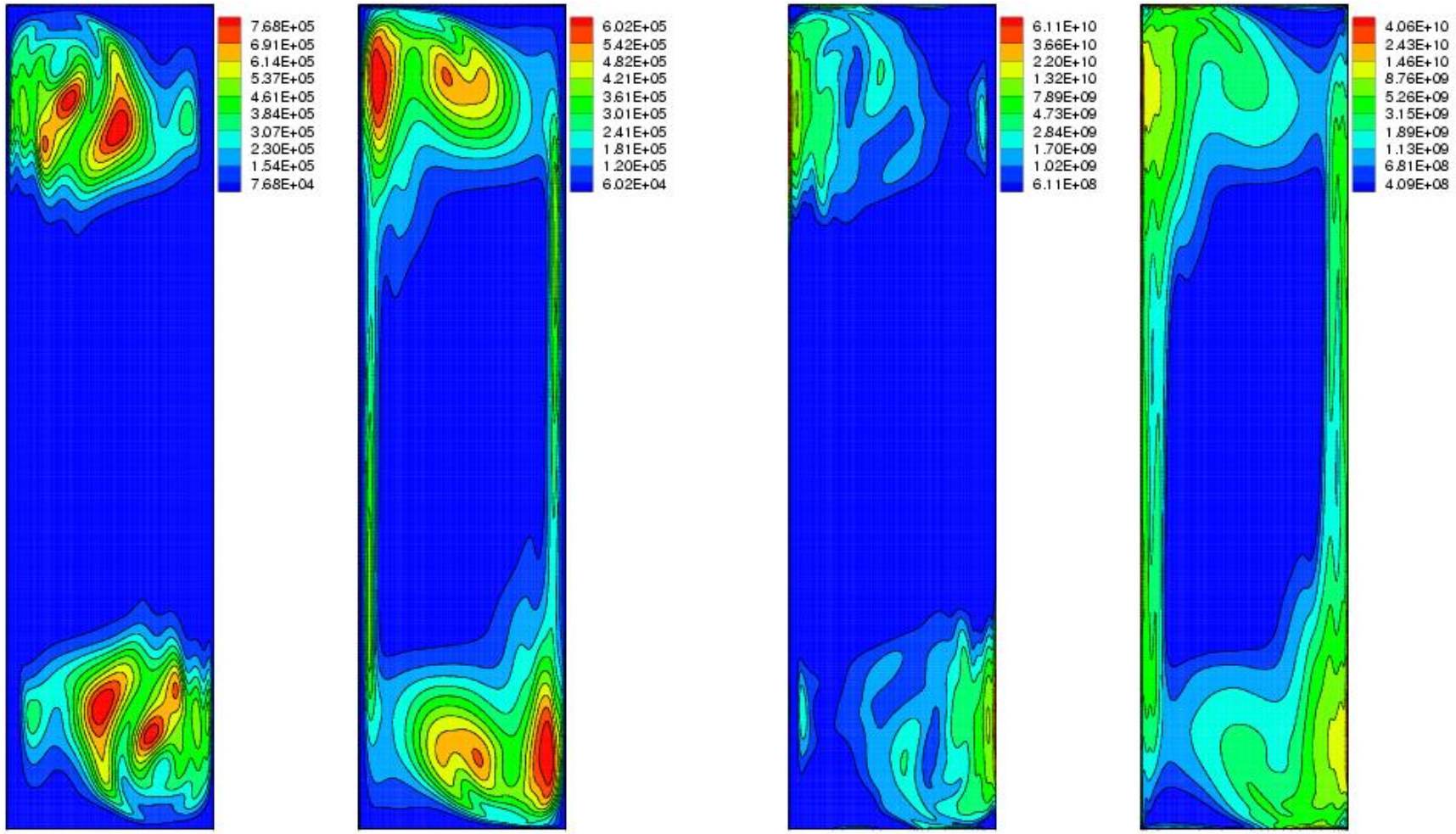
The **spectro-consistent** numerical scheme used for the simulations allows the discrete velocity fields to **verify exactly the global kinetic energy balance** even for coarse meshes

This has been tested as an additional verification of the code.

Second-order Statistics (1/4)

Turbulent kinetic energy and turbulent kinetic energy dissipation rate

2D 3D 2D 3D

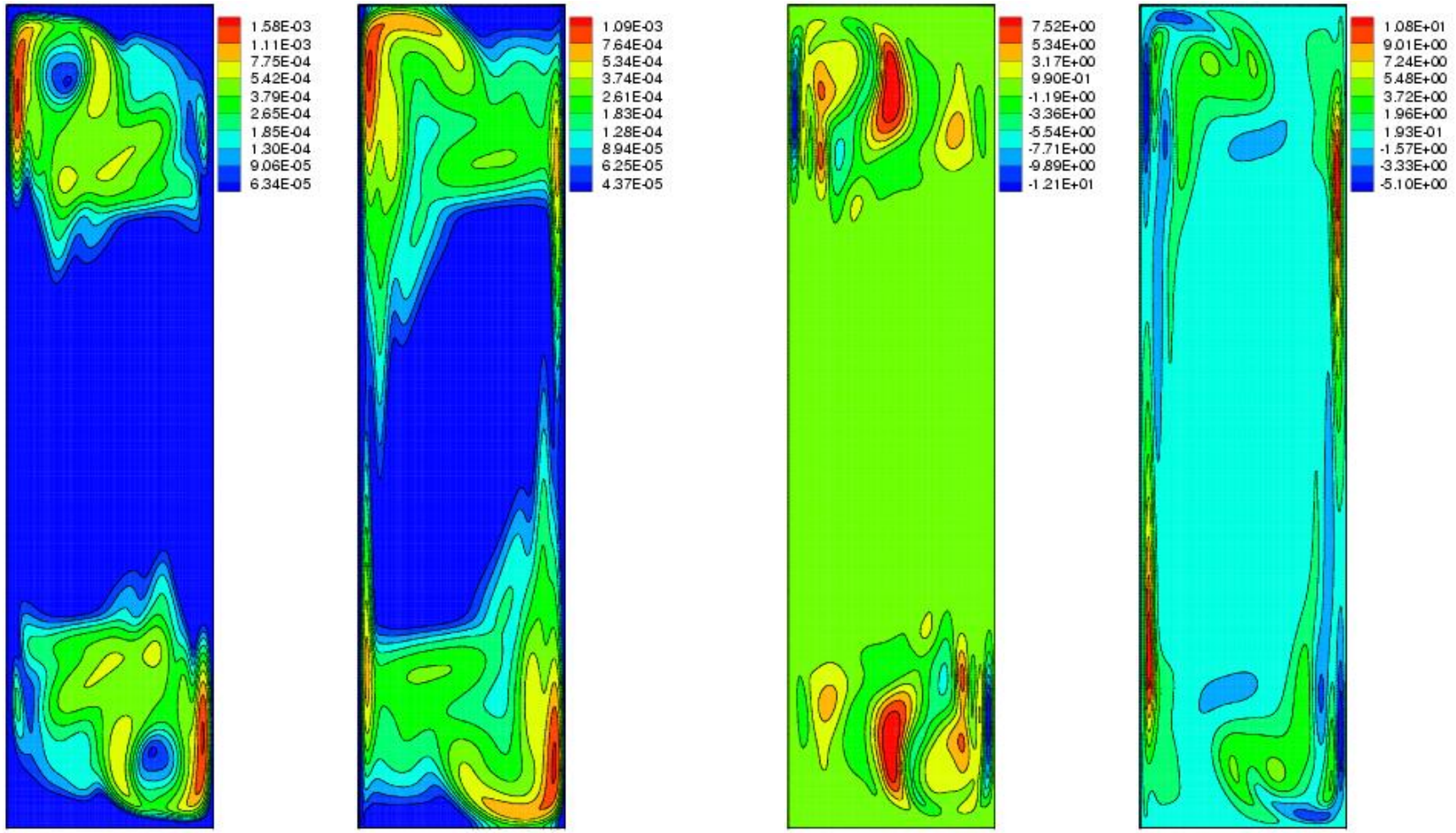


$\overline{u_i' u_i'}$ $\overline{\phi\left(\mathbf{u}'\right)}$

Second-order Statistics (2/4)

Variance of temperature and ratio of turbulent kinetic energy generation

2D 3D 2D 3D

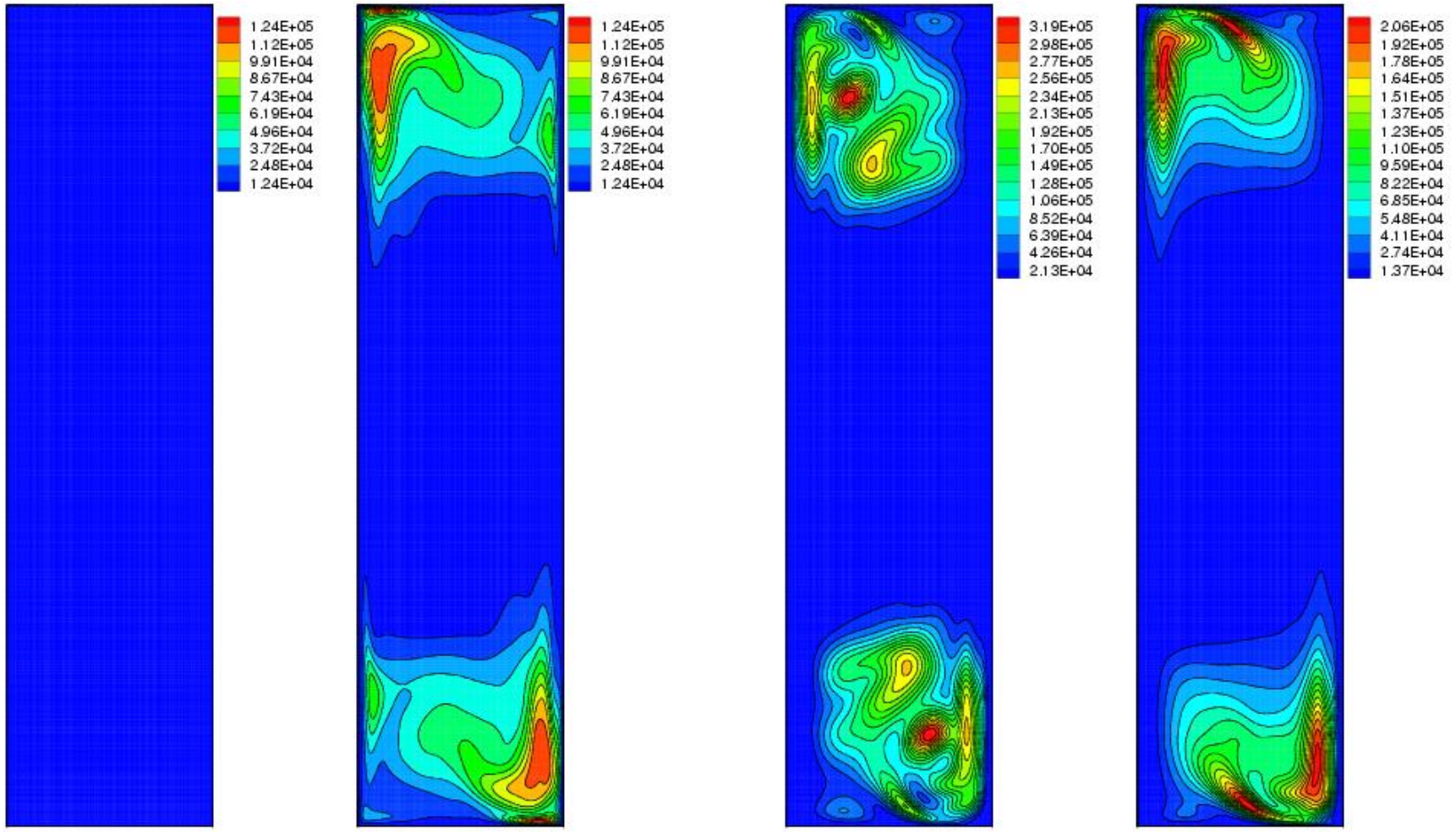


$\overline{T'T'}$ $\overline{u_3'T'}$

Second-order Statistics (3/4)

Components of Reynolds stress tensor

2D 3D 2D 3D

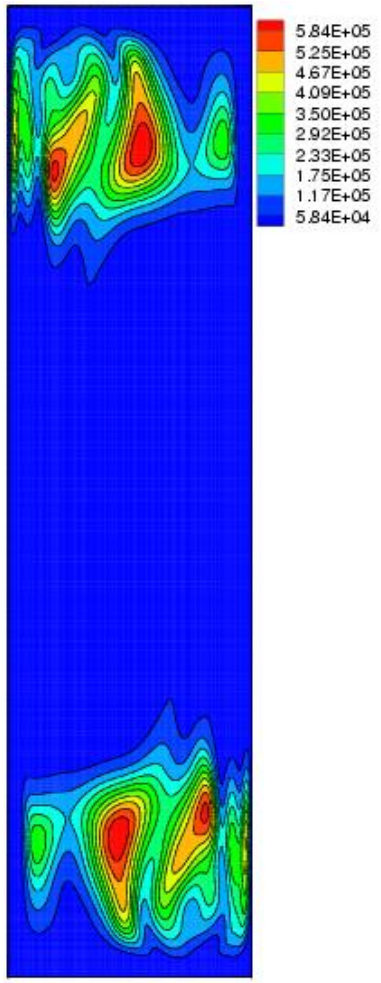


$\overline{u_1' u_1'}$ $\overline{u_2' u_2'}$

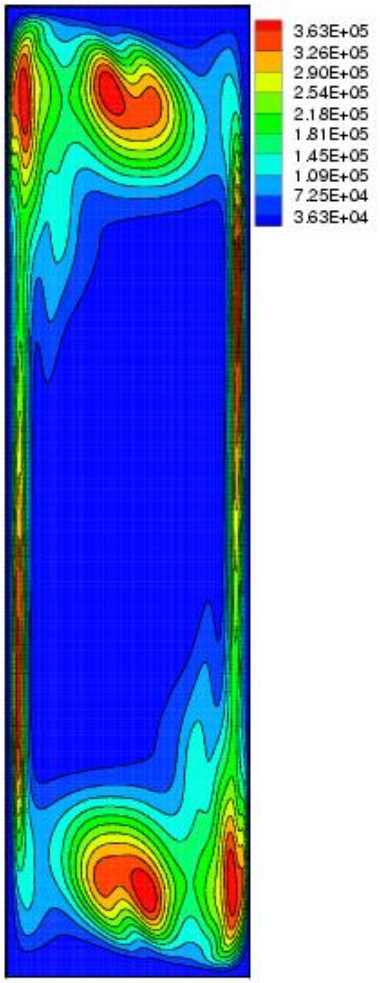
Second-order Statistics (4/4)

Components of Reynolds stress tensor

2D

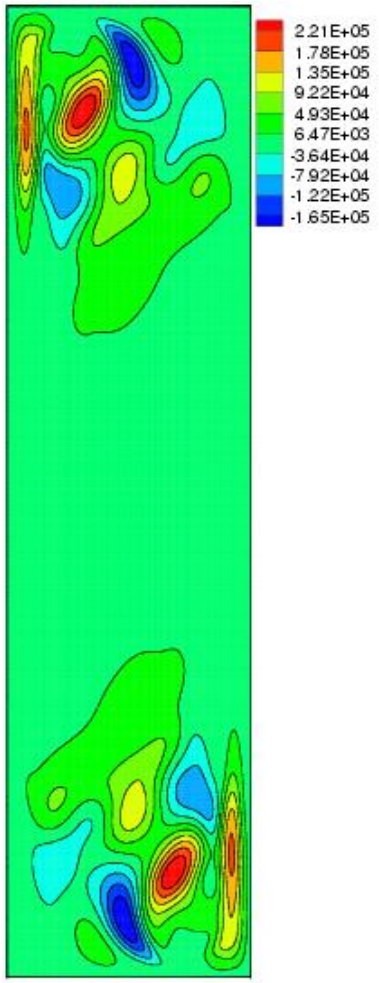


3D

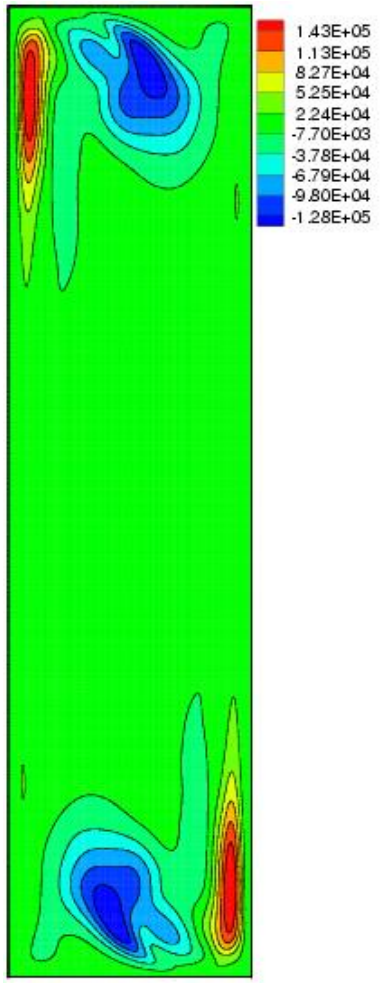


$\overline{u_3' u_3'}$

2D

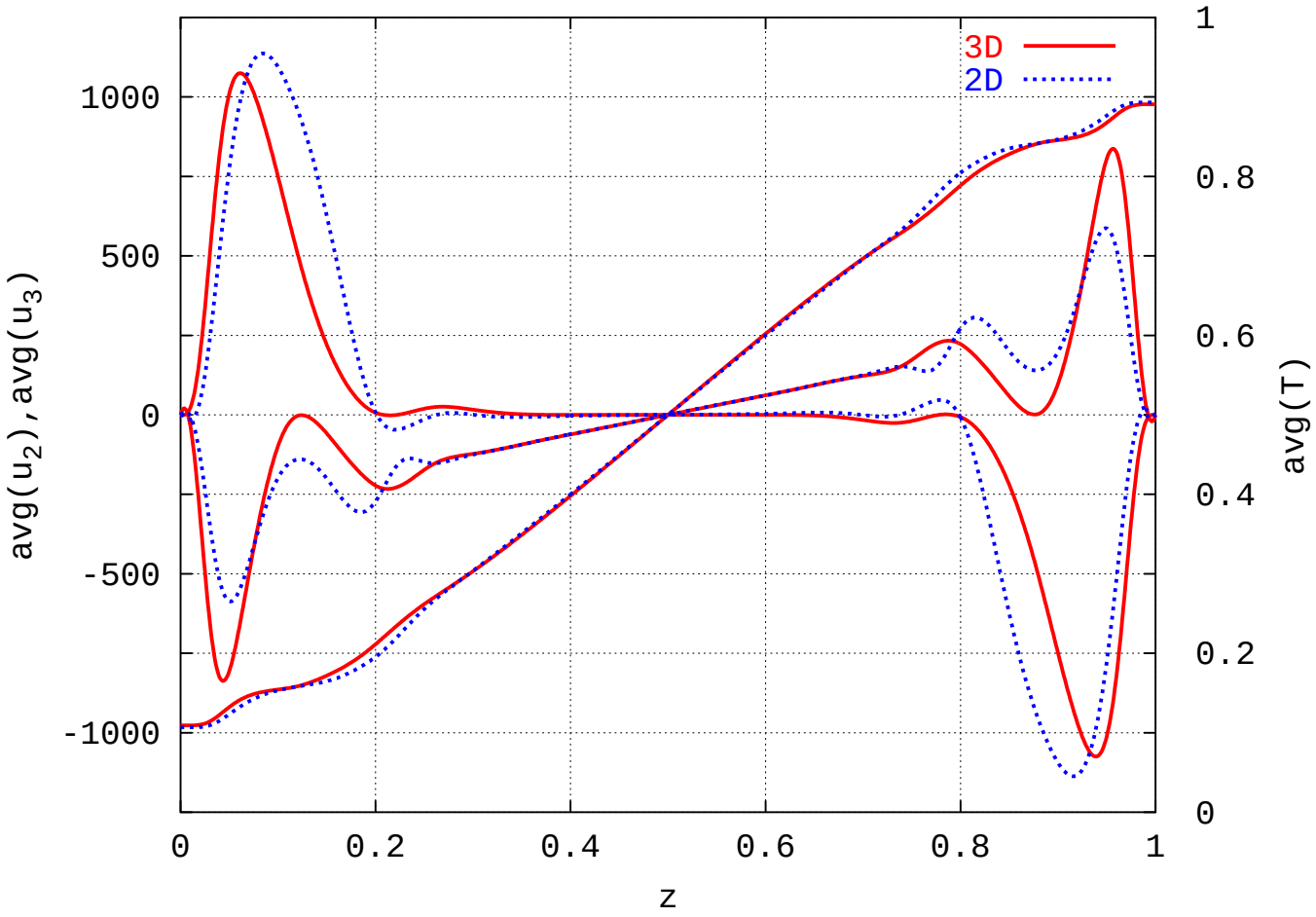


3D



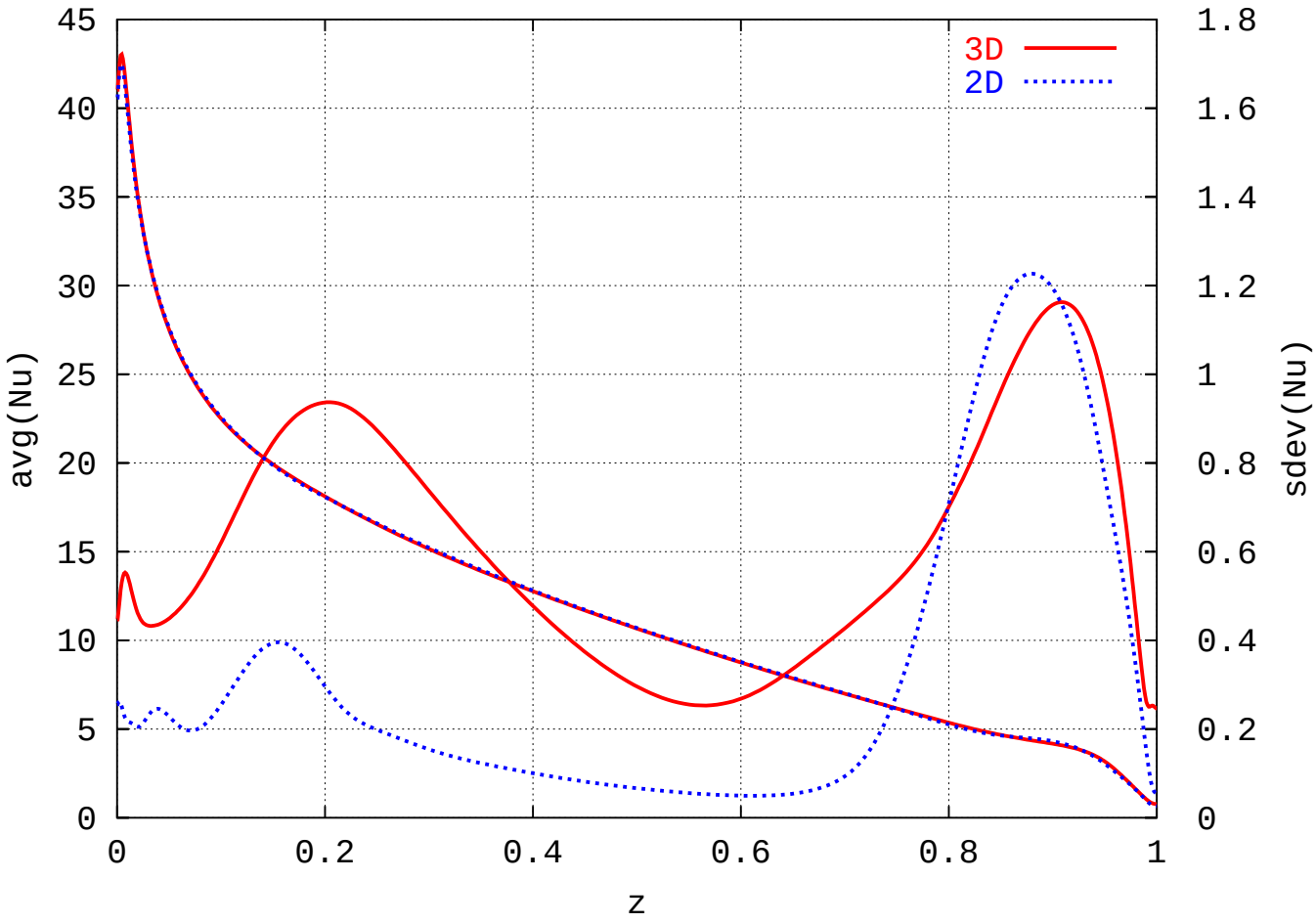
$\overline{u_2' u_3'}$

2D vs 3D profiles (1/2)



Vertical profiles at $y = 0{,}5L_y$ of $\overline{u_2}$, $\overline{u_3}$ and $\overline{T}$.

2D vs 3D profiles (2/2)



Averaged local Nusselt numbers and standard deviation of local Nusselt numbers.

Conclusions (1/2)

Respect to the application of DSFD to incompressible DNS with PC clusters

- **Accuracy** is close to machine precission, more than enough for CFD applications
- **Efficiency.** It allows to solve Poisson equations with $N = 3{,}1 \times 10^6$ in about three seconds, using 36 processors at 900 MHz with 512 MBytes RAM
- **Scalability.** Despite the low network performance, DSFD scales well up to 36-48 processors
- **Compared with sequential Additive Correction Multigrid** it is about 30 times faster and orders of magnitude more accurate

Therefore, within its range of application, it is an interesting alternative

Conclusions (2/2)

Respect to the **comparison** of the DNS 2D and 3D results:

- First-order statistics are similar, but second-order statistics are substantially different
- The main differences are at the vertical boundary layers, **where the 2D simulations incorrectly predict very low values for all the second-order statistics**

The **validity** of 2D flow assumption for relatively low-Rayleigh number **depends on** DNS applications:

- Development or enhancement of RANS turbulence models $\Longrightarrow$ NO VALID
- Main features of natural convection flows, such as the local and overall Nu numbers $\Longrightarrow$ VALID

To confirm these conclusions, the present results will be extended to higher Ra numbers

As a general conclusion, **the DSFD algorithm allows to use very low cost PC clusters for DNS simulations** with meshes of about 3×10^6 control volumes and 10^6 time steps.