

# Accelerating incompressible fluid flow simulations using SIMD or GPU computing

Yushan Wang<sup>1</sup>, Marc Baboulin<sup>1,2</sup>,  
Yann Fraigneau<sup>1,3</sup>, Olivier Le Maître<sup>1,3</sup>, Karl Rupp<sup>4</sup>

<sup>1</sup> Université Paris-Sud, France

<sup>2</sup> INRIA, France

<sup>3</sup> CNRS, France

<sup>4</sup> Argonne National Laboratory, USA

# Outline

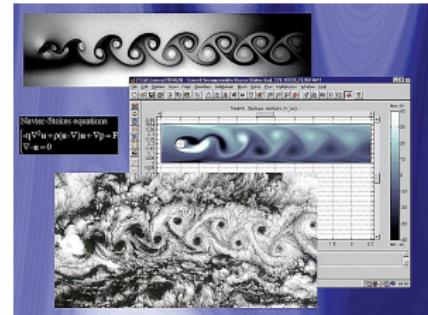
- Solving Navier-Stokes equations via a prediction-projection method
  - Helmholtz-like equation
  - Poisson equation
- Performance on a multicore architecture
- Accelerating tridiagonal systems solutions using SIMD
- GPU implementation
- Conclusion and future work

# Navier-Stokes equations

The Navier-Stokes equations describe mainly the motion of a viscous flow at all scales.

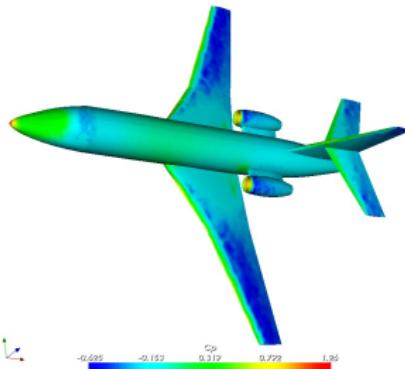


A Millennium Prize Problem of Navier-Stokes Equations.  
[http://www.claymath.org/millennium/  
Navier-Stokes\\_Equations/](http://www.claymath.org/millennium/Navier-Stokes_Equations/)



Global Climate Models and the Navier-Stokes Equations.

[http://climateaudit.org/2005/12/22/  
gcm's-and-the-navier-stokes-equations/](http://climateaudit.org/2005/12/22/gcms-and-the-navier-stokes-equations/)



Yushan Wang, LRI

Navier-Stokes simulation for the flow field around the Falcon business jet.  
[http://mfquant.net/gallery\\_cfd.html](http://mfquant.net/gallery_cfd.html)

Navier-Stokes Solver

# Navier-Stokes equations

$$\begin{cases} \frac{\partial \mathbf{V}}{\partial t} + \nabla \cdot (\mathbf{V} \otimes \mathbf{V}^T) = -\nabla P + \frac{1}{\text{Re}} \Delta \mathbf{V} \\ \nabla \cdot \mathbf{V} = 0 \end{cases}$$

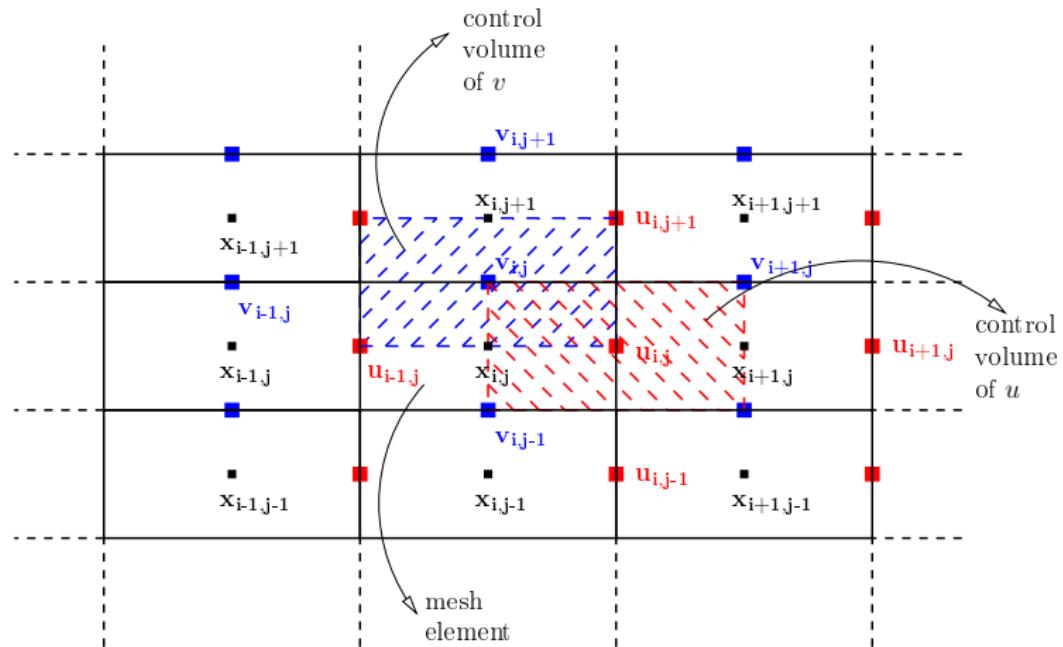
- $\mathbf{V}$  : velocity vector
- Re : Reynolds number
- $P$  : pressure

## Remark

- Density is neglected because the problem is supposed to be with constant coefficient.
- Reynolds number ( $\text{Re} = \rho UL/\mu$ ) indicates the fluid state. Larger Re demands finer mesh discretization.
- Convection term  $\mathcal{C}\mathcal{T} \equiv \nabla \cdot (\mathbf{V} \otimes \mathbf{V}^T)$  can be simplified as  $(\mathbf{V} \cdot \nabla)\mathbf{V}$  for incompressible fluid flow.

# Numerical method

Finite difference method with staggered mesh. ( $\mathbf{V} = (u, v)^T$ )



## Prediction-projection method

- $\frac{3\mathbf{V}^{n+1} - 4\mathbf{V}^n + \mathbf{V}^{n-1}}{2\Delta t} + \widetilde{\mathcal{CT}}^{n+1} = -\nabla P^{n+1} + \frac{1}{Re}\Delta\mathbf{V}^{n+1}$  (0)

- Hodge-Helmholtz decomposition:  $\mathbf{V}^* = \mathbf{V}^{n+1} + \nabla\psi$

- Prediction:  $\frac{3\mathbf{V}^* - 4\mathbf{V}^n + \mathbf{V}^{n-1}}{2\Delta t} + \widetilde{\mathcal{CT}}^{n+1} = -\nabla P^n + \frac{1}{Re}\Delta\mathbf{V}^*$  (1)

- Correction:  $\frac{3\mathbf{V}^{n+1} - 3\mathbf{V}^*}{2\Delta t} = -\nabla(P^{n+1} - P^n) + \frac{1}{Re}\Delta(\mathbf{V}^{n+1} - \mathbf{V}^*)$  (2)

- (1) +  $\frac{1}{Re}\Delta\mathbf{V}^n - \frac{1}{Re}\Delta\mathbf{V}^n \implies$  incremental **Helmholtz-like equation**:  $(I - \frac{2\Delta t}{3} \frac{1}{Re}\Delta)(\mathbf{V}^* - \mathbf{V}^n) = \mathbf{S}$  (3)

where  $\mathbf{S} = \frac{2\Delta t}{3}(\frac{1}{Re}\Delta\mathbf{V}^n - \widetilde{\mathcal{CT}}^{n+1} - \nabla P^n) + \frac{\mathbf{V}^n - \mathbf{V}^{n-1}}{3}$

- $\nabla \cdot (2) \implies$  **Poisson equation**:  $\Delta\phi = \frac{3}{2\Delta t}\nabla \cdot \mathbf{V}^*$  (4)

where  $\phi = P^{n+1} - P^n + \frac{1}{Re}\nabla \cdot \mathbf{V}^*$

## Prediction-projection method

- $\frac{3\mathbf{V}^{n+1} - 4\mathbf{V}^n + \mathbf{V}^{n-1}}{2\Delta t} + \widetilde{\mathcal{CT}}^{n+1} = -\nabla P^{n+1} + \frac{1}{Re}\Delta\mathbf{V}^{n+1}$  (0)

- Hodge-Helmholtz decomposition:  $\mathbf{V}^* = \mathbf{V}^{n+1} + \nabla\psi$

- Prediction:  $\frac{3\mathbf{V}^* - 4\mathbf{V}^n + \mathbf{V}^{n-1}}{2\Delta t} + \widetilde{\mathcal{CT}}^{n+1} = -\nabla P^n + \frac{1}{Re}\Delta\mathbf{V}^*$  (1)

- Correction:  $\frac{3\mathbf{V}^{n+1} - 3\mathbf{V}^*}{2\Delta t} = -\nabla(P^{n+1} - P^n) + \frac{1}{Re}\Delta(\mathbf{V}^{n+1} - \mathbf{V}^*)$  (2)

- (1) +  $\frac{1}{Re}\Delta\mathbf{V}^n - \frac{1}{Re}\Delta\mathbf{V}^n \Rightarrow$  incremental **Helmholtz-like equation**:  $(I - \frac{2\Delta t}{3} \frac{1}{Re}\Delta)(\mathbf{V}^* - \mathbf{V}^n) = \mathbf{S}$  (3)

where  $\mathbf{S} = \frac{2\Delta t}{3}(\frac{1}{Re}\Delta\mathbf{V}^n - \widetilde{\mathcal{CT}}^{n+1} - \nabla P^n) + \frac{\mathbf{V}^n - \mathbf{V}^{n-1}}{3}$

- $\nabla \cdot (2) \Rightarrow$  **Poisson equation**:  $\Delta\phi = \frac{3}{2\Delta t}\nabla \cdot \mathbf{V}^*$  (4)

where  $\phi = P^{n+1} - P^n + \frac{1}{Re}\nabla \cdot \mathbf{V}^*$

## Prediction-projection method

- $\frac{3\mathbf{V}^{n+1} - 4\mathbf{V}^n + \mathbf{V}^{n-1}}{2\Delta t} + \widetilde{\mathcal{CT}}^{n+1} = -\nabla P^{n+1} + \frac{1}{Re}\Delta\mathbf{V}^{n+1}$  (0)

- Hodge-Helmholtz decomposition:  $\mathbf{V}^* = \mathbf{V}^{n+1} + \nabla\psi$

- Prediction:  $\frac{3\mathbf{V}^* - 4\mathbf{V}^n + \mathbf{V}^{n-1}}{2\Delta t} + \widetilde{\mathcal{CT}}^{n+1} = -\nabla P^n + \frac{1}{Re}\Delta\mathbf{V}^*$  (1)

- Correction:  $\frac{3\mathbf{V}^{n+1} - 3\mathbf{V}^*}{2\Delta t} = -\nabla(P^{n+1} - P^n) + \frac{1}{Re}\Delta(\mathbf{V}^{n+1} - \mathbf{V}^*)$  (2)

- (1) +  $\frac{1}{Re}\Delta\mathbf{V}^n - \frac{1}{Re}\Delta\mathbf{V}^n \Rightarrow$  incremental Helmholtz-like equation:  $(I - \frac{2\Delta t}{3}\frac{1}{Re}\Delta)(\mathbf{V}^* - \mathbf{V}^n) = \mathbf{S}$  (3)

where  $\mathbf{S} = \frac{2\Delta t}{3}(\frac{1}{Re}\Delta\mathbf{V}^n - \widetilde{\mathcal{CT}}^{n+1} - \nabla P^n) + \frac{\mathbf{V}^n - \mathbf{V}^{n-1}}{3}$

- $\nabla \cdot (2) \Rightarrow$  Poisson equation:  $\Delta\phi = \frac{3}{2\Delta t}\nabla \cdot \mathbf{V}^*$  (4)

where  $\phi = P^{n+1} - P^n + \frac{1}{Re}\nabla \cdot \mathbf{V}^*$

## Prediction-projection method

- $\frac{3\mathbf{V}^{n+1} - 4\mathbf{V}^n + \mathbf{V}^{n-1}}{2\Delta t} + \widetilde{\mathcal{CT}}^{n+1} = -\nabla P^{n+1} + \frac{1}{Re}\Delta\mathbf{V}^{n+1}$  (0)

- Hodge-Helmholtz decomposition:  $\mathbf{V}^* = \mathbf{V}^{n+1} + \nabla\psi$

- Prediction:  $\frac{3\mathbf{V}^* - 4\mathbf{V}^n + \mathbf{V}^{n-1}}{2\Delta t} + \widetilde{\mathcal{CT}}^{n+1} = -\nabla P^n + \frac{1}{Re}\Delta\mathbf{V}^*$  (1)

- Correction:  $\frac{3\mathbf{V}^{n+1} - 3\mathbf{V}^*}{2\Delta t} = -\nabla(P^{n+1} - P^n) + \frac{1}{Re}\Delta(\mathbf{V}^{n+1} - \mathbf{V}^*)$  (2)

- (1) +  $\frac{1}{Re}\Delta\mathbf{V}^n - \frac{1}{Re}\Delta\mathbf{V}^n \implies$  incremental Helmholtz-like equation:  $(I - \frac{2\Delta t}{3}\frac{1}{Re}\Delta)(\mathbf{V}^* - \mathbf{V}^n) = \mathbf{S}$  (3)

where  $\mathbf{S} = \frac{2\Delta t}{3}(\frac{1}{Re}\Delta\mathbf{V}^n - \widetilde{\mathcal{CT}}^{n+1} - \nabla P^n) + \frac{\mathbf{V}^n - \mathbf{V}^{n-1}}{3}$

- $\nabla \cdot (2) \implies$  Poisson equation:  $\Delta\phi = \frac{3}{2\Delta t}\nabla \cdot \mathbf{V}^*$  (4)

where  $\phi = P^{n+1} - P^n + \frac{1}{Re}\nabla \cdot \mathbf{V}^*$

## Prediction-projection method

- $\frac{3\mathbf{V}^{n+1} - 4\mathbf{V}^n + \mathbf{V}^{n-1}}{2\Delta t} + \widetilde{\mathcal{CT}}^{n+1} = -\nabla P^{n+1} + \frac{1}{Re}\Delta\mathbf{V}^{n+1}$  (0)

- Hodge-Helmholtz decomposition:  $\mathbf{V}^* = \mathbf{V}^{n+1} + \nabla\psi$

- Prediction:  $\frac{3\mathbf{V}^* - 4\mathbf{V}^n + \mathbf{V}^{n-1}}{2\Delta t} + \widetilde{\mathcal{CT}}^{n+1} = -\nabla P^n + \frac{1}{Re}\Delta\mathbf{V}^*$  (1)

- Correction:  $\frac{3\mathbf{V}^{n+1} - 3\mathbf{V}^*}{2\Delta t} = -\nabla(P^{n+1} - P^n) + \frac{1}{Re}\Delta(\mathbf{V}^{n+1} - \mathbf{V}^*)$  (2)

- (1) +  $\frac{1}{Re}\Delta\mathbf{V}^n - \frac{1}{Re}\Delta\mathbf{V}^n \Rightarrow$  incremental **Helmholtz-like equation:**  $(\mathbf{I} - \frac{2\Delta t}{3}\frac{1}{Re}\Delta)(\mathbf{V}^* - \mathbf{V}^n) = \mathbf{S}$  (3)

where  $\mathbf{S} = \frac{2\Delta t}{3}(\frac{1}{Re}\Delta\mathbf{V}^n - \widetilde{\mathcal{CT}}^{n+1} - \nabla P^n) + \frac{\mathbf{V}^n - \mathbf{V}^{n-1}}{3}$

- $\nabla \cdot (2) \Rightarrow$  Poisson equation:  $\Delta\phi = \frac{3}{2\Delta t}\nabla \cdot \mathbf{V}^*$  (4)

where  $\phi = P^{n+1} - P^n + \frac{1}{Re}\nabla \cdot \mathbf{V}^*$

## Prediction-projection method

- $\frac{3\mathbf{V}^{n+1} - 4\mathbf{V}^n + \mathbf{V}^{n-1}}{2\Delta t} + \widetilde{\mathcal{CT}}^{n+1} = -\nabla P^{n+1} + \frac{1}{Re}\Delta\mathbf{V}^{n+1}$  (0)

- Hodge-Helmholtz decomposition:  $\mathbf{V}^* = \mathbf{V}^{n+1} + \nabla\psi$

- Prediction:  $\frac{3\mathbf{V}^* - 4\mathbf{V}^n + \mathbf{V}^{n-1}}{2\Delta t} + \widetilde{\mathcal{CT}}^{n+1} = -\nabla P^n + \frac{1}{Re}\Delta\mathbf{V}^*$  (1)

- Correction:  $\frac{3\mathbf{V}^{n+1} - 3\mathbf{V}^*}{2\Delta t} = -\nabla(P^{n+1} - P^n) + \frac{1}{Re}\Delta(\mathbf{V}^{n+1} - \mathbf{V}^*)$  (2)

- (1) +  $\frac{1}{Re}\Delta\mathbf{V}^n - \frac{1}{Re}\Delta\mathbf{V}^n \Rightarrow$  incremental **Helmholtz-like equation:**  $(\mathbf{I} - \frac{2\Delta t}{3}\frac{1}{Re}\Delta)(\mathbf{V}^* - \mathbf{V}^n) = \mathbf{S}$  (3)

where  $\mathbf{S} = \frac{2\Delta t}{3}(\frac{1}{Re}\Delta\mathbf{V}^n - \widetilde{\mathcal{CT}}^{n+1} - \nabla P^n) + \frac{\mathbf{V}^n - \mathbf{V}^{n-1}}{3}$

- $\nabla \cdot (2) \Rightarrow$  **Poisson equation:**  $\Delta\phi = \frac{3}{2\Delta t}\nabla \cdot \mathbf{V}^*$  (4)

where  $\phi = P^{n+1} - P^n + \frac{1}{Re}\nabla \cdot \mathbf{V}^*$

# Prediction-projection method

Time increment on  $\mathbf{V}$  and  $P$ :

$$\begin{cases} P^{n+1} = P^n + \phi - \frac{1}{\text{Re}} \nabla \cdot \mathbf{v}^* \\ \mathbf{v}^{n+1} = \mathbf{v}^* - \frac{2\Delta t}{3} \nabla \phi \end{cases}$$

Time iterations:

$$\left. \begin{array}{c} P^n \\ \mathbf{v}^n \end{array} \right\} \xrightarrow{\text{Helmholtz-like eq.}} \mathbf{v}^* \xrightarrow{\text{Poisson eq.}} \phi \xrightarrow{\text{Increments}} \left. \begin{array}{c} P^{n+1} \\ \mathbf{v}^{n+1} \end{array} \right\}$$

## Solving Helmholtz-like equation with ADI method

$$(\mathbf{I} - \frac{2\Delta t}{3} \frac{1}{\text{Re}} \Delta) (\mathbf{V}_i^* - \mathbf{V}_i^n) = \mathbf{S}_i \quad i \in \{x, y, z\}$$

### Alternating Direction Implicit method

The 3D operator  $(\mathbf{I} - \epsilon \Delta)$  is approximated as a product of three 1D operators:

$$\mathbf{I} - \epsilon \Delta = (\mathbf{I} - \epsilon \Delta_x)(\mathbf{I} - \epsilon \Delta_y)(\mathbf{I} - \epsilon \Delta_z) + O(\epsilon^2)$$

$$\left\{ \begin{array}{lcl} (\mathbf{I} - \frac{2\Delta t}{3} \frac{1}{\text{Re}} \Delta_x) \mathbf{T}' & = & \mathbf{S} \\ (\mathbf{I} - \frac{2\Delta t}{3} \frac{1}{\text{Re}} \Delta_y) \mathbf{T}'' & = & \mathbf{T}' \\ (\mathbf{I} - \frac{2\Delta t}{3} \frac{1}{\text{Re}} \Delta_z) (\mathbf{V}_i^* - \mathbf{V}_i^n) & = & \mathbf{T}'' \end{array} \right. \quad i \in \{x, y, z\}$$

## Solving Helmholtz-like equation with ADI method

$$(\mathbf{I} - \frac{2\Delta t}{3} \frac{1}{\text{Re}} \Delta) (\mathbf{V}_i^* - \mathbf{V}_i^n) = \mathbf{S}_i \quad i \in \{x, y, z\}$$

### Alternating Direction Implicit method

The 3D operator  $(\mathbf{I} - \epsilon \Delta)$  is approximated as a product of three 1D operators:

$$\mathbf{I} - \epsilon \Delta = (\mathbf{I} - \epsilon \Delta_x)(\mathbf{I} - \epsilon \Delta_y)(\mathbf{I} - \epsilon \Delta_z) + \mathcal{O}(\epsilon^2)$$

$$\left\{ \begin{array}{lcl} (\mathbf{I} - \frac{2\Delta t}{3} \frac{1}{\text{Re}} \Delta_x) \mathbf{T}' & = & \mathbf{S} \\ (\mathbf{I} - \frac{2\Delta t}{3} \frac{1}{\text{Re}} \Delta_y) \mathbf{T}'' & = & \mathbf{T}' \\ (\mathbf{I} - \frac{2\Delta t}{3} \frac{1}{\text{Re}} \Delta_z) (\mathbf{V}_i^* - \mathbf{V}_i^n) & = & \mathbf{T}'' \end{array} \right. \quad i \in \{x, y, z\}$$

## Solving Helmholtz-like equation with ADI method

$$(\mathbf{I} - \frac{2\Delta t}{3} \frac{1}{\text{Re}} \Delta) (\mathbf{V}_i^* - \mathbf{V}_i^n) = \mathbf{S}_i \quad i \in \{x, y, z\}$$

### Alternating Direction Implicit method

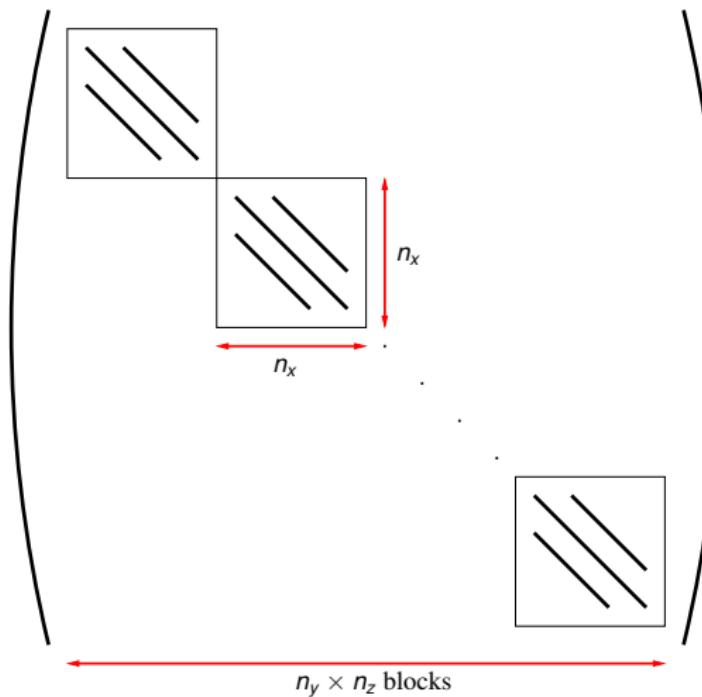
The 3D operator  $(\mathbf{I} - \epsilon \Delta)$  is approximated as a product of three 1D operators:

$$\mathbf{I} - \epsilon \Delta = (\mathbf{I} - \epsilon \Delta_x)(\mathbf{I} - \epsilon \Delta_y)(\mathbf{I} - \epsilon \Delta_z) + O(\epsilon^2)$$

$$\left\{ \begin{array}{lcl} (\mathbf{I} - \frac{2\Delta t}{3} \frac{1}{\text{Re}} \Delta_x) \mathbf{T}' & = & \mathbf{S} \\ (\mathbf{I} - \frac{2\Delta t}{3} \frac{1}{\text{Re}} \Delta_y) \mathbf{T}'' & = & \mathbf{T}' \\ (\mathbf{I} - \frac{2\Delta t}{3} \frac{1}{\text{Re}} \Delta_z) (\mathbf{V}_i^* - \mathbf{V}_i^n) & = & \mathbf{T}'' \end{array} \right. \quad i \in \{x, y, z\}$$

## Block tridiagonal matrix

Matrix structure of  $(\mathbf{I} - \frac{2\Delta t}{3} \frac{1}{\text{Re}} \Delta_x)$



# Solving Poisson equation using partial diagonalization

$$\Delta\phi = \frac{3\nabla \cdot \mathbf{V}^*}{2\Delta t} \Leftrightarrow L\phi = S \Leftrightarrow (L_x + L_y + L_z)\phi = S.$$

$$\left. \begin{array}{rcl} L_x & = & Q_x \Lambda_x Q_x^{-1} \\ L_y & = & Q_y \Lambda_y Q_y^{-1} \\ S' & = & Q_x^{-1} Q_y^{-1} S \\ \phi' & = & Q_x^{-1} Q_y^{-1} \phi \end{array} \right\} \Rightarrow (\Lambda_x + \Lambda_y + L_z)\phi' = S'$$

- Projection of source term:  $S' = Q_x^{-1} Q_y^{-1} S$
- Solution of tridiagonal system:  $(\Lambda_x + \Lambda_y + L_z)\phi' = S'$
- Final solution is:  $\phi = Q_y Q_x \phi'$
- Remark: this method is only available for **separable problems**.

# Solving Poisson equation using partial diagonalization

$$\Delta\phi = \frac{3\nabla \cdot \mathbf{V}^*}{2\Delta t} \Leftrightarrow L\phi = S \Leftrightarrow (L_x + L_y + L_z)\phi = S.$$

$$\left. \begin{array}{lcl} L_x & = & Q_x \Lambda_x Q_x^{-1} \\ L_y & = & Q_y \Lambda_y Q_y^{-1} \\ S' & = & Q_x^{-1} Q_y^{-1} S \\ \phi' & = & Q_x^{-1} Q_y^{-1} \phi \end{array} \right\} \Rightarrow (\Lambda_x + \Lambda_y + L_z)\phi' = S'$$

- Projection of source term:  $S' = Q_x^{-1} Q_y^{-1} S$
- Solution of tridiagonal system:  $(\Lambda_x + \Lambda_y + L_z)\phi' = S'$
- Final solution is:  $\phi = Q_y Q_x \phi'$
- Remark: this method is only available for **separable problems**.

# Solving Poisson equation using partial diagonalization

$$\Delta\phi = \frac{3\nabla \cdot \mathbf{V}^*}{2\Delta t} \Leftrightarrow L\phi = S \Leftrightarrow (L_x + L_y + L_z)\phi = S.$$

$$\left. \begin{array}{lcl} L_x & = & Q_x \Lambda_x Q_x^{-1} \\ L_y & = & Q_y \Lambda_y Q_y^{-1} \\ S' & = & Q_x^{-1} Q_y^{-1} S \\ \phi' & = & Q_x^{-1} Q_y^{-1} \phi \end{array} \right\} \Rightarrow (\Lambda_x + \Lambda_y + L_z)\phi' = S'$$

- Projection of source term:  $S' = Q_x^{-1} Q_y^{-1} S$
- Solution of tridiagonal system:  $(\Lambda_x + \Lambda_y + L_z)\phi' = S'$
- Final solution is:  $\phi = Q_y Q_x \phi'$
- Remark: this method is only available for **separable problems**.

# Solving Poisson equation using partial diagonalization

$$\Delta\phi = \frac{3\nabla \cdot \mathbf{V}^*}{2\Delta t} \Leftrightarrow L\phi = S \Leftrightarrow (L_x + L_y + L_z)\phi = S.$$

$$\left. \begin{array}{rcl} L_x & = & Q_x \Lambda_x Q_x^{-1} \\ L_y & = & Q_y \Lambda_y Q_y^{-1} \\ S' & = & Q_x^{-1} Q_y^{-1} S \\ \phi' & = & Q_x^{-1} Q_y^{-1} \phi \end{array} \right\} \Rightarrow (\Lambda_x + \Lambda_y + L_z)\phi' = S'$$

- Projection of source term:  $S' = Q_x^{-1} Q_y^{-1} S$
- Solution of tridiagonal system:  $(\Lambda_x + \Lambda_y + L_z)\phi' = S'$
- Final solution is:  $\phi = Q_y Q_x \phi'$
- Remark: this method is only available for **separable problems**.

# Solving Poisson equation using partial diagonalization

$$\Delta\phi = \frac{3\nabla \cdot \mathbf{V}^*}{2\Delta t} \Leftrightarrow L\phi = S \Leftrightarrow (L_x + L_y + L_z)\phi = S.$$

$$\left. \begin{array}{rcl} L_x & = & Q_x \Lambda_x Q_x^{-1} \\ L_y & = & Q_y \Lambda_y Q_y^{-1} \\ S' & = & Q_x^{-1} Q_y^{-1} S \\ \phi' & = & Q_x^{-1} Q_y^{-1} \phi \end{array} \right\} \Rightarrow (\Lambda_x + \Lambda_y + L_z)\phi' = S'$$

- Projection of source term:  $S' = Q_x^{-1} Q_y^{-1} S$
- Solution of tridiagonal system:  $(\Lambda_x + \Lambda_y + L_z)\phi' = S'$
- Final solution is:  $\phi = Q_y Q_x \phi'$
- Remark: this method is only available for **separable problems**.

# Solving Poisson equation using partial diagonalization

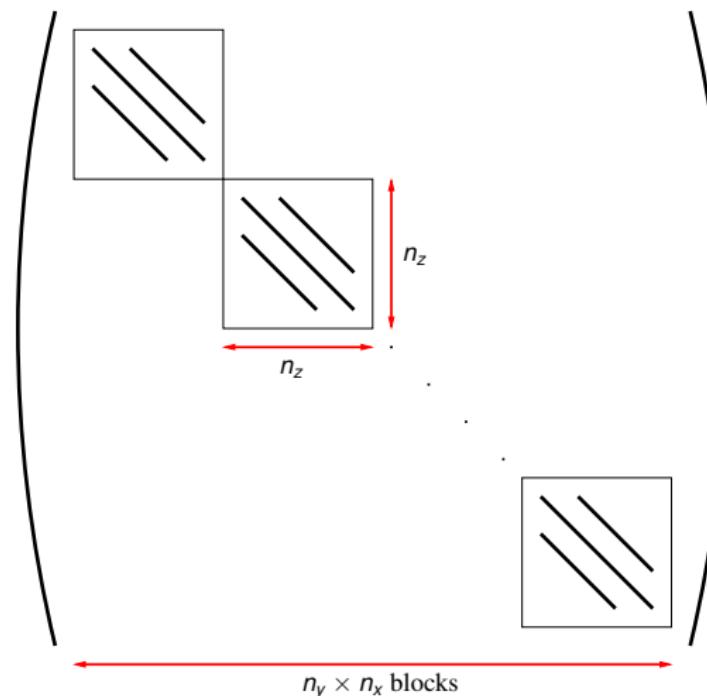
$$\Delta\phi = \frac{3\nabla \cdot \mathbf{V}^*}{2\Delta t} \Leftrightarrow L\phi = S \Leftrightarrow (L_x + L_y + L_z)\phi = S.$$

$$\left. \begin{array}{lcl} L_x & = & Q_x \Lambda_x Q_x^{-1} \\ L_y & = & Q_y \Lambda_y Q_y^{-1} \\ S' & = & Q_x^{-1} Q_y^{-1} S \\ \phi' & = & Q_x^{-1} Q_y^{-1} \phi \end{array} \right\} \Rightarrow (\Lambda_x + \Lambda_y + L_z)\phi' = S'$$

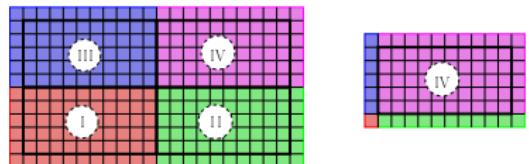
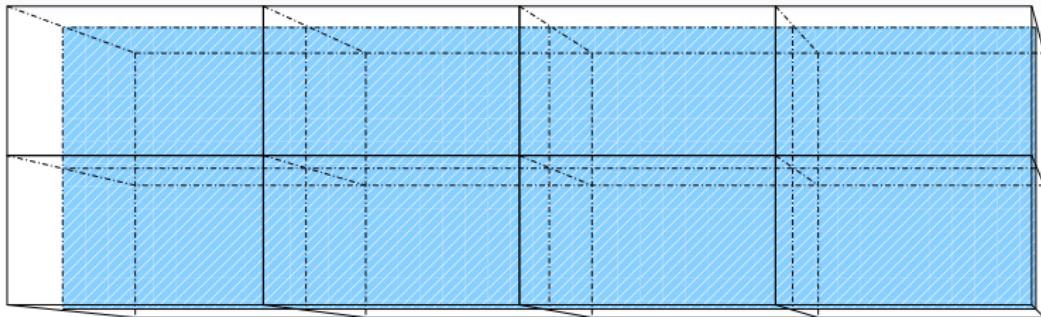
- Projection of source term:  $S' = Q_x^{-1} Q_y^{-1} S$
- Solution of tridiagonal system:  $(\Lambda_x + \Lambda_y + L_z)\phi' = S'$
- Final solution is:  $\phi = Q_y Q_x \phi'$
- Remark: this method is only available for **separable problems**.

# Block tridiagonal matrix

Matrix structure of  $(\Lambda_x + \Lambda_y + L_z)$



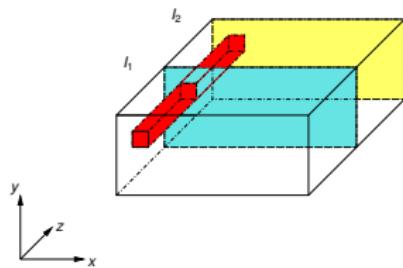
# Parallel implementation



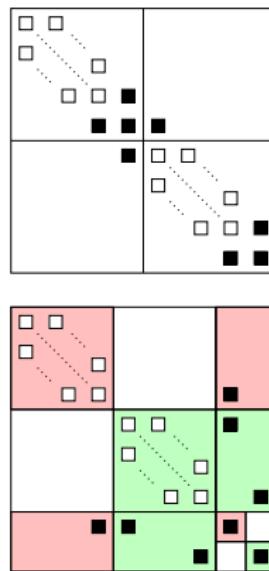
- Domain decomposition via Schur complement method.
- Interface exchanges via MPI.
- One subdomain corresponds to one process.
- Kernels from ScaLAPACK and MKL libraries.

# Schur complement method

The Schur complement method is applied when there are multiple subdomains along the solving direction.



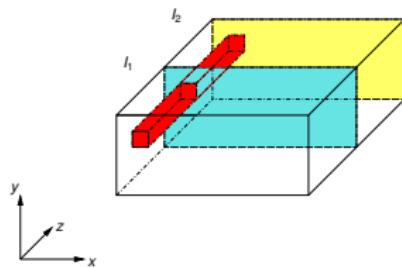
Example for solving a tridiagonal system  
along z-direction.



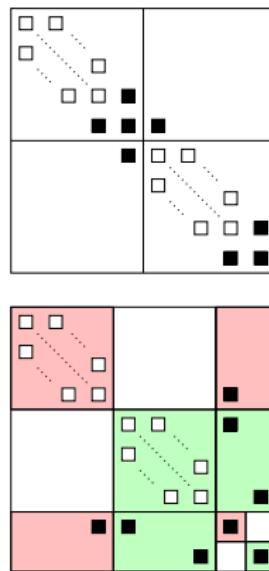
This method results in information exchanges!

# Schur complement method

The Schur complement method is applied when there are multiple subdomains along the solving direction.

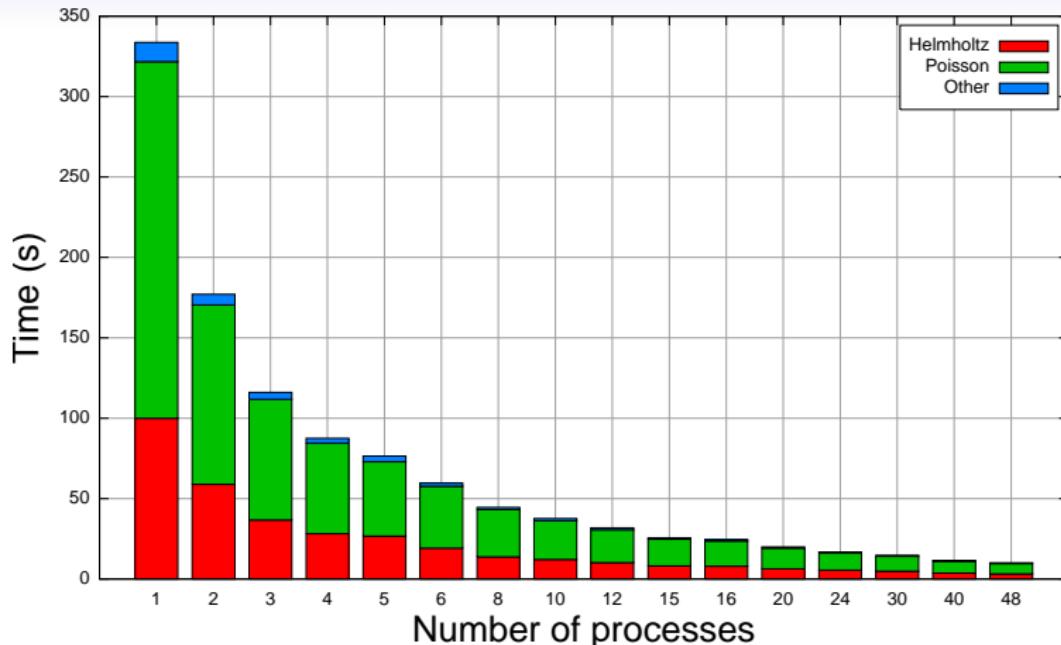


Example for solving a tridiagonal system  
along z-direction.



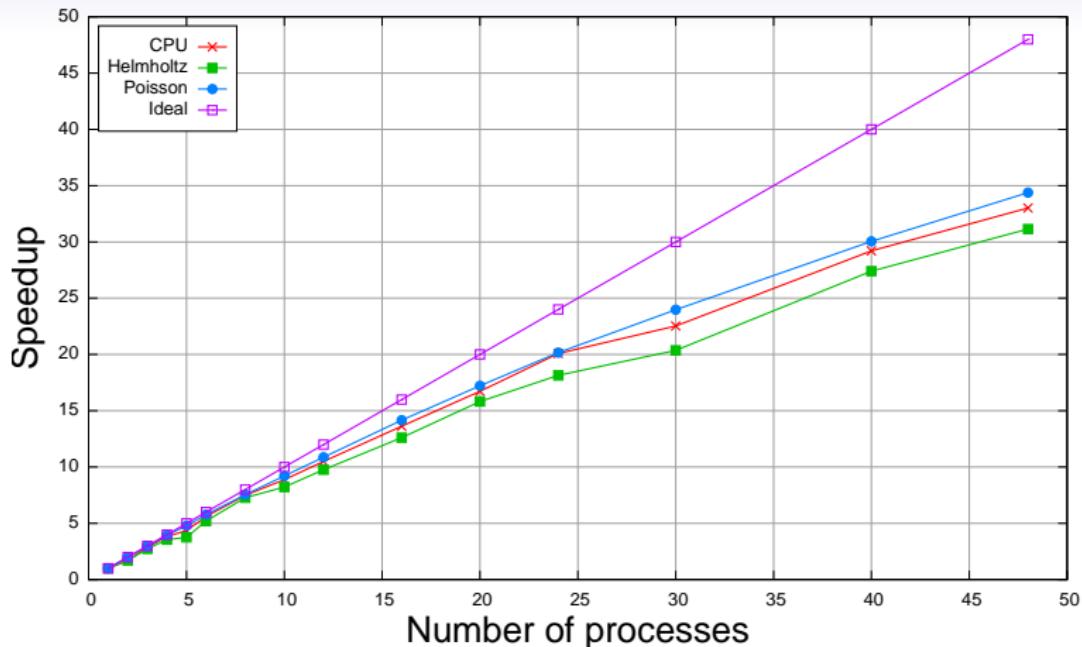
This method results in information exchanges!

# Performance results: Time breakdown



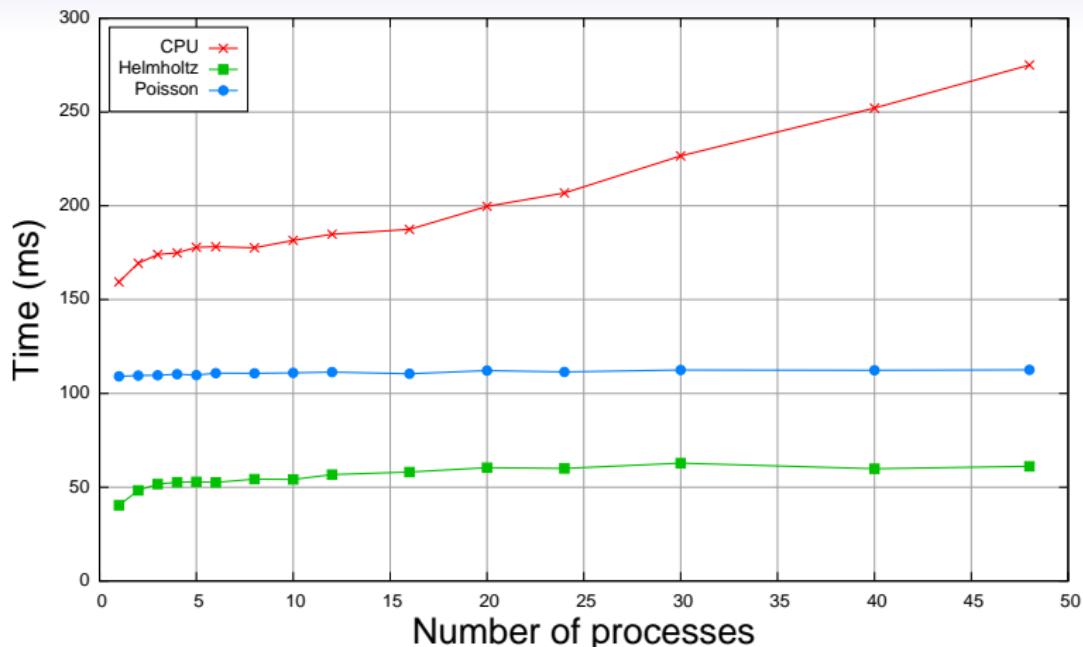
Problem size :  $240^3$   
double precision  
MagnyCours-48 system from University of Tennessee  
 $4 \times 12$  AMD Opteron Processor 6172

# Performance results: Strong scalability



Problem size :  $240^3$   
double precision  
MagnyCours-48 system from University of Tennessee  
 $4 \times 12$  AMD Opteron Processor 6172

# Performance results: Weak scalability



Problem size per process :  $240 \times 240 \times 10$   
double precision

MagnyCours-48 system from University of Tennessee  
 $4 \times 12$  AMD Opteron Processor 6172

# Tridiagonal Solver

At each time step, 10 tridiagonal systems to solve.

- **Helmholtz-like equation:**

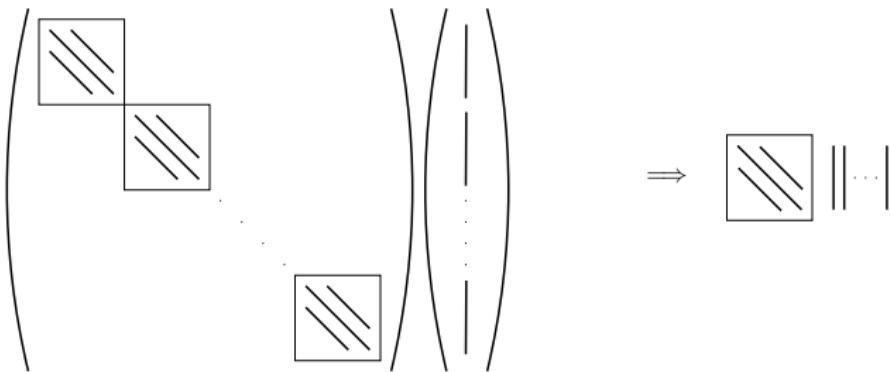
$$\begin{cases} \left(\mathbf{I} - \frac{2\Delta t}{3} \frac{1}{\text{Re}} \Delta_x\right) \mathbf{T}' &= \mathbf{S}_i \\ \left(\mathbf{I} - \frac{2\Delta t}{3} \frac{1}{\text{Re}} \Delta_y\right) \mathbf{T}'' &= \mathbf{T}' \quad i \in \{x, y, z\} \\ \left(\mathbf{I} - \frac{2\Delta t}{3} \frac{1}{\text{Re}} \Delta_z\right) (\mathbf{V}_i^* - \mathbf{V}_i^n) &= \mathbf{T}'' \end{cases}$$

- **Poisson equation:**

$$(\Lambda_x + \Lambda_y + L_z) \phi' = S'$$

# Tridiagonal Solver

The tridiagonal systems have the same **block tridiagonal** structure.



**Helmholtz-like equation:**

The tridiagonal blocks are identical  $\Rightarrow$  a smaller tridiagonal system with multiple RHS.

Second order central difference scheme  $\Rightarrow$  **diagonally dominant** matrix.

# Thomas Algorithm

$$\begin{pmatrix} b_1 & c_1 & & & \\ a_2 & b_2 & c_2 & & \\ \ddots & \ddots & \ddots & & \\ & a_i & b_i & c_i & \\ & \ddots & \ddots & \ddots & \\ & a_{n-1} & b_{n-1} & c_{n-1} & \\ & a_n & b_n & & \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_i \\ \vdots \\ x_{n-1} \\ x_n \end{pmatrix} = \begin{pmatrix} s_1 \\ s_2 \\ \vdots \\ s_i \\ \vdots \\ s_{n-1} \\ s_n \end{pmatrix}.$$

Forward elimination

**for**  $i = 2$  **to**  $n$ , **do**

$$b_i = b_i - \frac{c_{i-1} \times a_i}{b_{i-1}};$$

$$s_i = s_i - \frac{s_{i-1} \times a_i}{b_{i-1}};$$

**end**

Backward substitution:

$$x_n = \frac{s_n}{b_n};$$

**for**  $i = n - 1$  **to**  $1$ , **do**

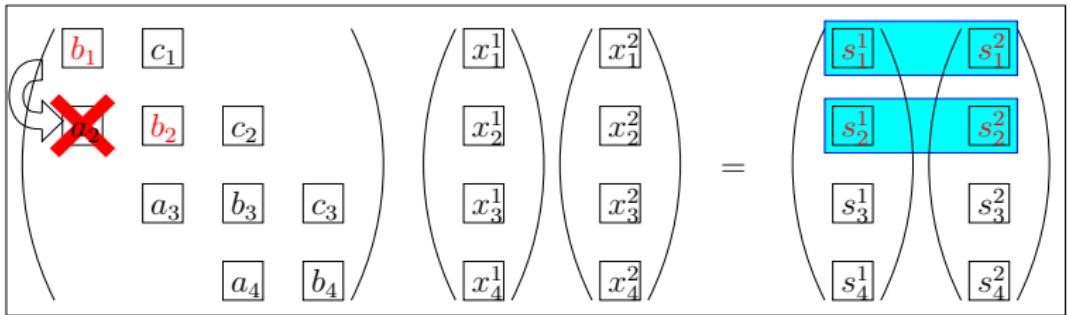
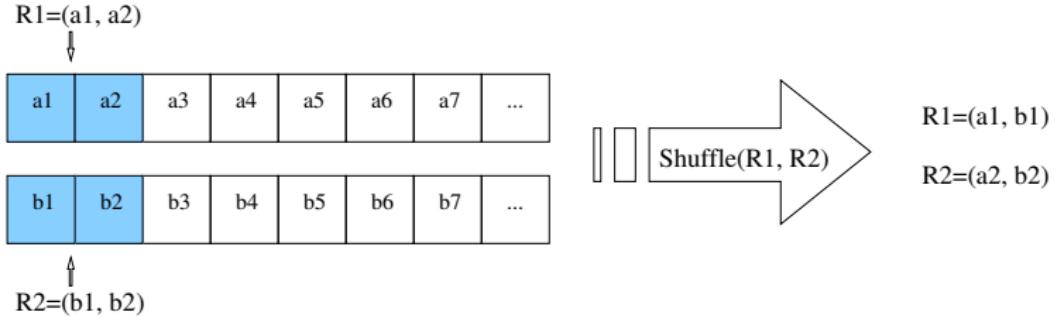
$$x_i = \frac{s_i - c_i \times x_{i+1}}{b_i};$$

**end**

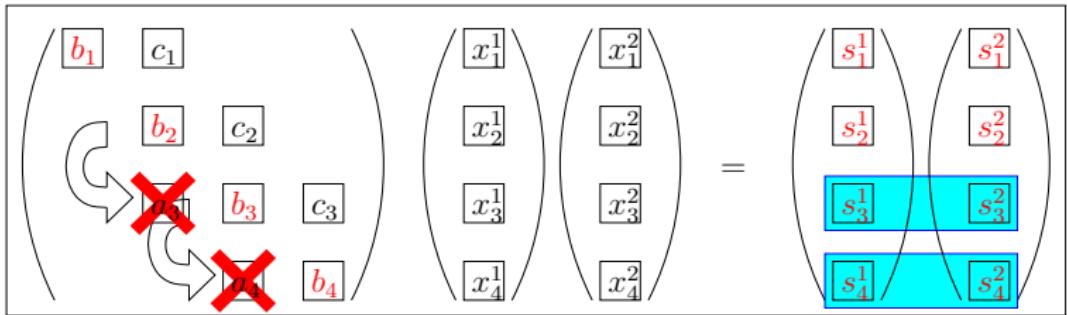
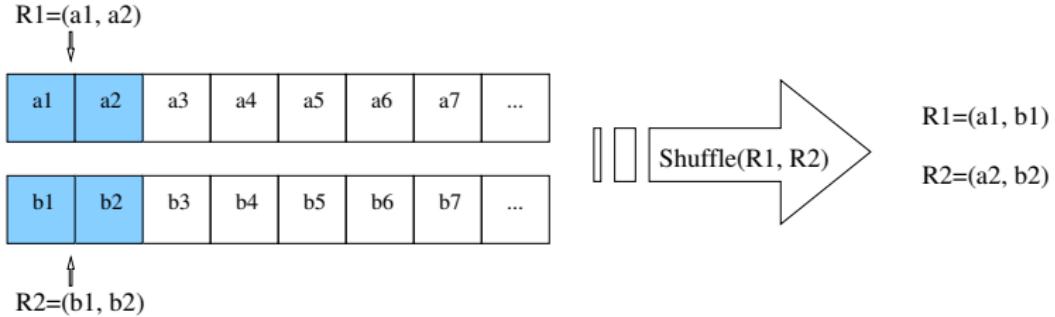
# Vectorization

- Implemented using a generic SIMD abstraction library (BOOST.SIMD) for all SSE variants and AVX.
- Boost.SIMD, a C++ template library that simplifies the exploitation of SIMD hardware within a standard C++ programming model.
- Scalable system that takes care of increasing wide of SIMD systems (128 bits today, 512 bits in Intel Xeon Phi coprocessors).
- See [ Estérie et al., Boost.SIMD: Generic Programming for portable SIMDization ] .

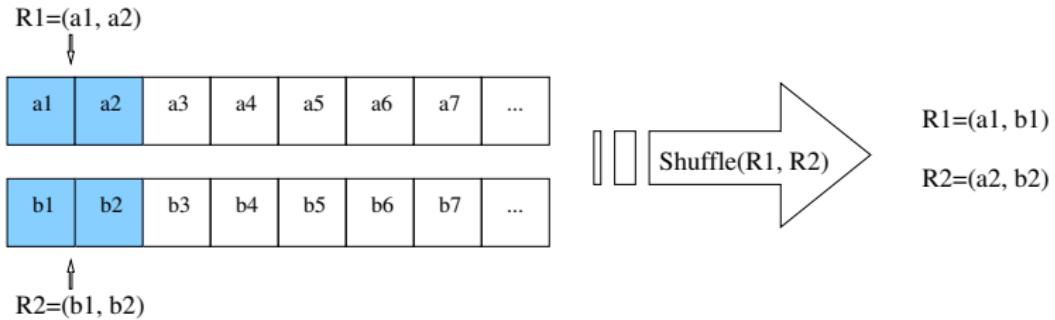
# Tridiagonal solver with vectorization



# Tridiagonal solver with vectorization

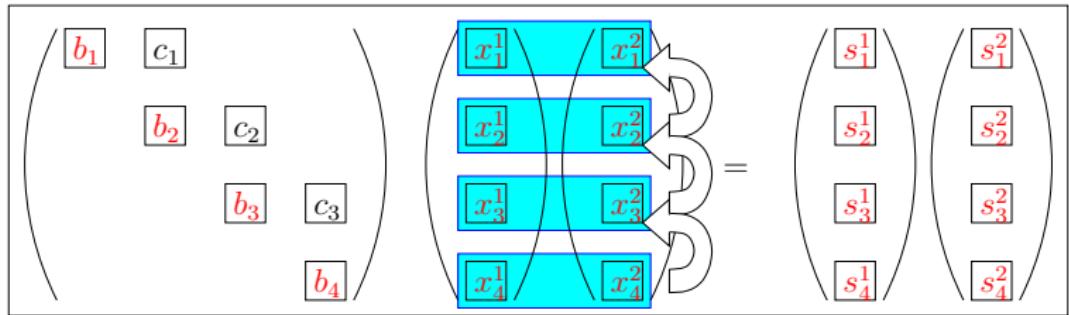
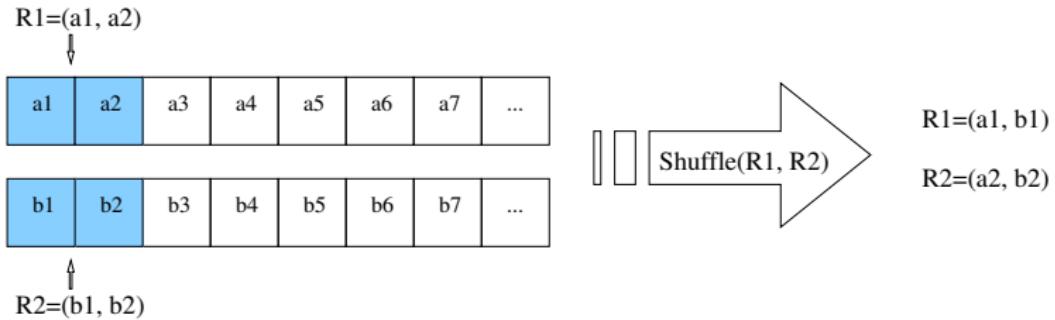


# Tridiagonal solver with vectorization

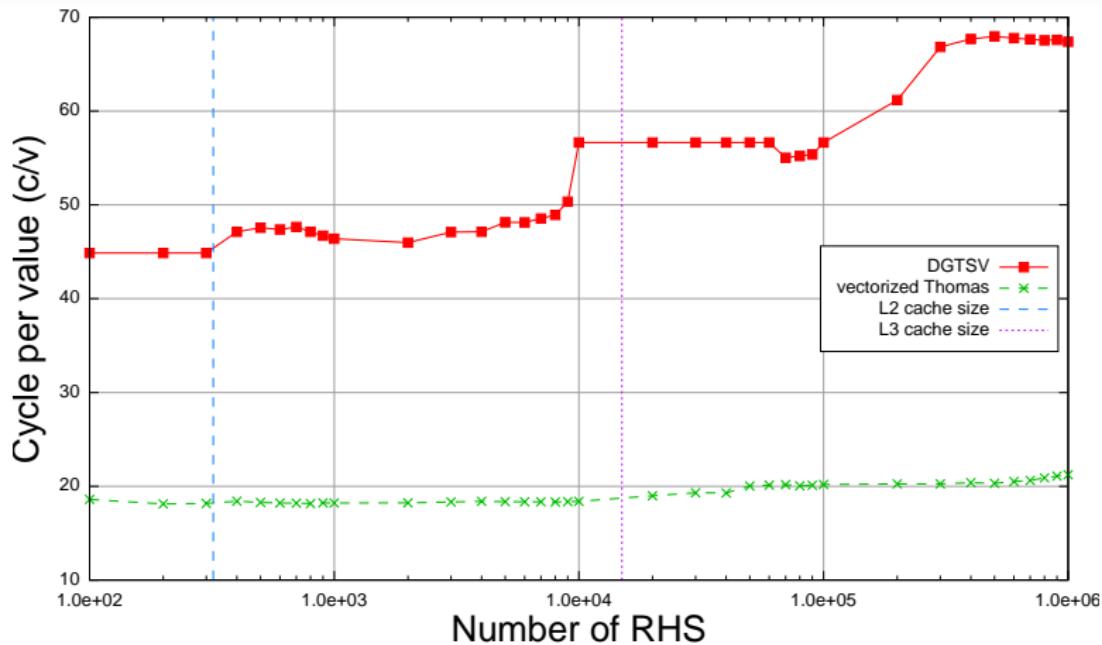


$$\begin{pmatrix} b_1 & c_1 \\ b_2 & c_2 \\ b_3 & c_3 \\ b_4 & \end{pmatrix} \begin{pmatrix} x_1^1 \\ x_2^1 \\ x_3^1 \\ x_4^1 \end{pmatrix} \begin{pmatrix} x_1^2 \\ x_2^2 \\ x_3^2 \\ x_4^2 \end{pmatrix} = \begin{pmatrix} s_1^1 \\ s_2^1 \\ s_3^1 \\ s_4^1 \end{pmatrix} \begin{pmatrix} s_1^2 \\ s_2^2 \\ s_3^2 \\ s_4^2 \end{pmatrix}$$

# Tridiagonal solver with vectorization



# Performance: Cycle per value



Intel(R) Xeon(R) CPU E5645 @ 2.40GHz  
double precision

## Steps of NS solver

Domain initialization

Computation of eigen values and vectors

For each time iteration:

- Solve Helmholtz equation
- Solve Poisson equation
- Variables increments
- Record current numerical solution

## Steps of NS solver

Domain initialization

Computation of eigen values and vectors

For each time iteration:

- Solve Helmholtz equation
- Solve Poisson equation
- Variables increments
- Record current numerical solution

## Helmholtz-like equation

- Tridiagonal block structure with identical blocks.
- One GPU thread deals with one RHS value.
- Data reordering after each solving step.

## Poisson equation

- Tridiagonal block structure with different blocks.
- One GPU thread deals with one tridiagonal block.
- Matrix-matrix multiplication.
- Data reordering after each multiplication and solving step.

## Preliminary results

	Helmholtz	Poisson
Transfers CPU → GPU	0.416s	0.126s
Matrix multiplication	-	0.024s
Solution reordering	0.014s	0.014s
Tridiagonal system solve	0.169s(9)	0.169s(1)
Total/iteration GPU solver	<b>1.569s</b>	<b>0.333s</b>
Total/iteration CPU solver (48 cores)	<b>3.21s</b>	<b>6.45s</b>

- Tesla C2075 (448 CUDA cores), mini-titan@lri
- Matrix multiplication by DGEMM of MAGMA library
- Transfers not included (needed only at the begining of the time iteration)

## Conclusion

- Scalable algorithm and CPU implementation of a 3D Navier-Stokes equations.
- Tridiagonal solver acceleration using vectorization.
- For discontinuous domains, we use an iterative method to solve the Poisson equation. (SOR+multigrid)
- GPU Helmholtz and Poisson solver.

## Ongoing work

MultiGPU solver for Navier-Stokes equations using partial diagonalisation and ADI method. Collaboration with Argonne National Laboratory (Karl Rupp).

Thank You!