3D CFD Simulation Using a Parallel Hybrid Approach of the Block Cimmino Iterative Method

Ronan Guivarch[2] Alexandre Silva Lopes[1] Daniel Ruiz[2]
Mohamed Zenadi[3]

(1) CEsA–FEUP, Porto, Portugal

(2) IRIT–INP–ENSEEIHT, Université de Toulouse, France

(3) IRIT–UPS, Université de Toulouse, France

Outline

Introduction

- Block Cimmino method: stationary iterative method to solve linear systems based on a row projection technique
- Linear system divided into subsystems
- At every iteration, one minimum norm solution is computed for each subsystem and these are used to construct an approximation to the solution of the full linear system

Introduction

- Natural parallelization methodologies: distribute one or several subsystems per processor (block iterative),
- Our experimental strategy: let a direct solver handle the data distribution and in particular nested levels of parallelism (hybrid)

Introduction

- Natural parallelization methodologies: distribute one or several subsystems per processor (block iterative),
- Our experimental strategy: let a direct solver handle the data distribution and in particular nested levels of parallelism (hybrid)

Block Cimmino Algorithm

The system

The linear system of equations

$$\mathbf{A}x = b$$

where $\mathbf{A}$ is a $m \times n$ matrix, is partitioned into l subsystems, with $l \leq m$, such that:

$$\begin{pmatrix} \mathbf{A}_1 \\ \mathbf{A}_2 \\ \vdots \\ \mathbf{A}_l \end{pmatrix} x = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_l \end{pmatrix}$$

Block Cimmino Algorithm, continued

The principle

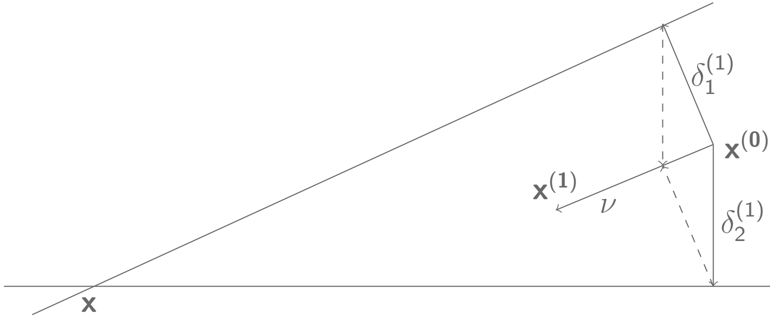
- Block Cimmino iteration is formulated as:

$$\begin{aligned}\delta_i^{(k)} &= \mathbf{A}_i^{+} b_i - \mathbf{P}_{\mathcal{R}(A_i^T)} x^{(k)} \\ &= \mathbf{A}_i^{+}\left(b_i - \mathbf{A}_i x^{(k)}\right) \\ x^{(k+1)} &= x^{(k)} + \nu \sum_{i=1}^{l} \delta_i^{(k)}\end{aligned}$$

  where the matrix $\mathbf{A}_i^{+}$ refers to the Moore-Penrose pseudo-inverse of $\mathbf{A}_i$ defined as $\mathbf{A}_i^{+} = \mathbf{A}_i^T\left(\mathbf{A}_i\mathbf{A}_i^T\right)^{-1}$ and $\mathbf{P}_{\mathcal{R}(A_i^T)}$ is an orthogonal projector onto the range of $\mathbf{A}_i^T$
- However, the Block Cimmino method will converge for any other pseudo-inverse of $\mathbf{A}_i$ such as the generalized pseudo-inverse $\mathbf{A}_{i\ \mathbf{G}^{-1}}^{-} = \mathbf{G}^{-1}\mathbf{A}_i^T\left(\mathbf{A}_i\mathbf{G}^{-1}\mathbf{A}_i^T\right)^{-1}$, were $\mathbf{G}$ is an ellipsoidal norm matrix

Block Cimmino Algorithm, continued

The geometric representation



Optimal asymptotic rate of convergence is $\nu = 2/(\mu_{max} + \mu_{min})$

Block Cimmino Algorithm, continued

How to solve it?

- Normal equations: high condition number $\kappa(AA^T) = (\kappa(A))^2$ and the risk of cancellation
- Augmented systems approach

$$\begin{bmatrix} \mathbf{I} & \mathbf{A}_i^T \\ \mathbf{A}_i & 0 \end{bmatrix} \begin{bmatrix} u_i \\ v_i \end{bmatrix} = \begin{bmatrix} 0 \\ b_i - \mathbf{A}_i x \end{bmatrix}$$

with solution

$$\begin{aligned} v_i &= -\left(\mathbf{A}_i \mathbf{A}_i T\right)^{-1} r_i \\ u_i &= \mathbf{A}_i^{+}(b_i - \mathbf{A}_i x) \\ &= \delta_i \end{aligned}$$

Block Cimmino Algorithm, continued

How to solve it?

- Normal equations: high condition number $\kappa(AA^T) = (\kappa(A))^2$ and the risk of cancellation
- Augmented systems approach

$$\begin{bmatrix} \mathbf{I} & \mathbf{A}_i^T \\ \mathbf{A}_i & 0 \end{bmatrix} \begin{bmatrix} u_i \\ v_i \end{bmatrix} = \begin{bmatrix} 0 \\ b_i - \mathbf{A}_i x \end{bmatrix}$$

with solution

$$\begin{aligned} v_i &= -\left(\mathbf{A}_i \mathbf{A}_i T\right)^{-1} r_i \\ u_i &= \mathbf{A}_i^{+}(b_i - \mathbf{A}_i x) \\ &= \delta_i \end{aligned}$$

Block Cimmino Algorithm, continued

The iteration matrix

- Using Elfving notation

$$\begin{aligned} E_{RJ}x &= h_{RJ} \\ E_{RJ} &= \sum_{i=1}^{l} A_i^T(A_iA_i^T)^{-1}A_i \\ h_{RJ} &= \sum_{i=1}^{l} A_i^T(A_iA_i^T)^{-1}b_i \end{aligned}$$

- Iteration matrix is symmetric positive definite (SPD) (provided **G** is SPD too)
- The convergence rate can be improved with the use of Block-CG

Block Cimmino Algorithm, continued

Acceleration using Block-CG

```
$X^{(0)} \leftarrow$ *arbitrary*
$\tilde{R}^{(0)} \leftarrow H_{RJ} - E_{RJ}X^{(0)}$
$\tilde{P}^{(0)} \leftarrow \tilde{R}^{(0)}$
$k \leftarrow 0$
**loop**
    $\Omega^{(k)} \leftarrow E_{RJ}P^{(k)}$
    $\beta^{(k)} \leftarrow (\tilde{P}^{(k)T}\Omega^{(k)})^{-1}(\tilde{R}^{(k)T}\tilde{R}^{(k)})$
    $X^{(k+1)} \leftarrow X^{(k)} + \beta^{(k)}\tilde{P}^{(k)}$
    **if** Converged **then**
        *exit loop*
    **end if**
    $\tilde{R}^{(k+1)} \leftarrow \tilde{R}^{(k)} - \beta^{(k)}\Omega^{(k)}$
    $\alpha^{(k)} \leftarrow (\tilde{R}^{(k)T}\tilde{R}^{(k)})^{-1}(\tilde{R}^{(k+1)T}\tilde{R}^{(k+1)})$
    $\tilde{P}^{(k+1)} \leftarrow \tilde{R}^{(k+1)} + \alpha^{(k+1)}\tilde{R}^{(k)}$
    $k \leftarrow k + 1$
**end loop**
```

$$E_{RJ}X = H_{RJ}$$

- System to be solved at each iteration
- Best way: Direct solver!

Block Cimmino Algorithm, continued

Acceleration using Block-CG

```
$X^{(0)} \leftarrow \textit{arbitrary}$
$\tilde{R}^{(0)} \leftarrow H_{RJ} - E_{RJ}X^{(0)}$
$\tilde{P}^{(0)} \leftarrow \tilde{R}^{(0)}$
$k \leftarrow 0$
loop
    $\Omega^{(k)} \leftarrow E_{RJ}P^{(k)}$
    $\beta^{(k)} \leftarrow (\tilde{P}^{(k)T}\Omega^{(k)})^{-1}(\tilde{R}^{(k)T}\tilde{R}^{(k)})$
    $X^{(k+1)} \leftarrow X^{(k)} + \beta^{(k)}\tilde{P}^{(k)}$
    if Converged then
        exit loop
    end if
    $\tilde{R}^{(k+1)} \leftarrow \tilde{R}^{(k)} - \beta^{(k)}\Omega^{(k)}$
    $\alpha^{(k)} \leftarrow (\tilde{R}^{(k)T}\tilde{R}^{(k)})^{-1}(\tilde{R}^{(k+1)T}\tilde{R}^{(k+1)})$
    $\tilde{P}^{(k+1)} \leftarrow \tilde{R}^{(k+1)} + \alpha^{(k+1)}\tilde{R}^{(k)}$
    $k \leftarrow k + 1$
end loop
```

$$E_{RJ}X = H_{RJ}$$

- System to be solved at each iteration
- Best way: Direct solver!

Traditional Parallelization Approach

- Common way to parallelize a Block Algorithm
  - distribute augmented systems on available processors
  - easy distribution: one augmented system per processor
- This natural way becomes not so easy when we try to find a distribution which leads to a kind of load-balancing: high number of processors $\Rightarrow$ create the same number of blocks
  - convergence of block-iterative algorithms often decreases when the number of blocks increases
  - blocks with small granularity $\Rightarrow$ the cost of communications can overcome the cost of computations

Original Parallelization Approach

- For the parallelization, we try to follow an alternative approach:
    - no manual augmented systems distribution
    - we let the direct solver handle the data distribution and the main parallelization tasks

Original Parallelization Approach

- For the parallelization, we try to follow an alternative approach:
  - no manual augmented systems distribution
  - we let the direct solver handle the data distribution and the main parallelization tasks

Original Parallelization Approach

- For the parallelization, we try to follow an alternative approach:
  - no manual augmented systems distribution
  - we let the direct solver handle the data distribution and the main parallelization tasks

MUMPS

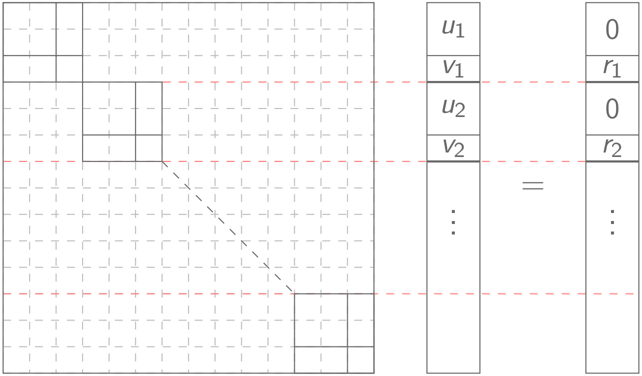
MUltifrontal Massively Parallel Sparse direct Solver

- collaboration of IRIT with INRIA and CERFACS
  http:/mumps.enseeiht.fr or http://graal.ens-lyon.fr/MUMPS
- `MUMPS` solves $\mathbf{A}x = b$, where $\mathbf{A}$ is a large sparse matrix, with direct factorization of $\mathbf{A}$ into $\mathbf{A} = \mathbf{LU}$ or $\mathbf{LDL^T}$.
- 3 main steps (plus initialization and termination):

JOB=1 Analysis,
JOB=2 Factorization,
JOB=3 Solution.

How MUMPS handles our tasks

- Data from Block Cimmino subsystems are gathered in a larger sparse matrix



How MUMPS handles our tasks

- This block diagonal matrix is given to MUMPS
- A single instance of MUMPS distributes the matrix after analysing its structural properties
  - factorization handling directly 3 levels of parallelism
  - efficient embedded load-balancing strategy
- MUMPS does the analysis (sequential) and the factorization (parallel)
- At each iteration MUMPS solves the subsystems (trees) seen as an unique system (forest)

How MUMPS handles our tasks

- This block diagonal matrix is given to MUMPS
- A single instance of MUMPS distributes the matrix after analysing its structural properties
  - factorization handling directly 3 levels of parallelism
  - efficient embedded load-balancing strategy
- MUMPS does the analysis (sequential) and the factorization (parallel)
- At each iteration MUMPS solves the subsystems (trees) seen as an unique system (forest)

How MUMPS handles our tasks

- This block diagonal matrix is given to MUMPS
- A single instance of MUMPS distributes the matrix after analysing its structural properties
  - factorization handling directly 3 levels of parallelism
  - efficient embedded load-balancing strategy
- MUMPS does the analysis (sequential) and the factorization (parallel)
- At each iteration MUMPS solves the subsystems (trees) seen as an unique system (forest)

How MUMPS handles our tasks

- This block diagonal matrix is given to MUMPS
- A single instance of MUMPS distributes the matrix after analysing its structural properties
  - factorization handling directly 3 levels of parallelism
  - efficient embedded load-balancing strategy
- MUMPS does the analysis (sequential) and the factorization (parallel)
- At each iteration MUMPS solves the subsystems (trees) seen as an unique system (forest)

Some advantages in this approach

- Sparse linear solver handles other levels of parallelism on top of the block partitioning
  - the one coming from the sparsity structure
  - BLAS3 kernels
- More degrees of freedom when partitioning the initial matrix
  - less blocks than processors but larger ones
  - may help to increase the speed of convergence of the iterative method, while still maintaining enough degree of parallelism
  - independent from the machine hierarchy

Some advantages in this approach

- Sparse linear solver handles other levels of parallelism on top of the block partitioning
    - the one coming from the sparsity structure
    - BLAS3 kernels
- More degrees of freedom when partitioning the initial matrix
    - less blocks than processors but larger ones
    - may help to increase the speed of convergence of the iterative method, while still maintaining enough degree of parallelism
    - independent from the machine hierarchy

Context
The hardware

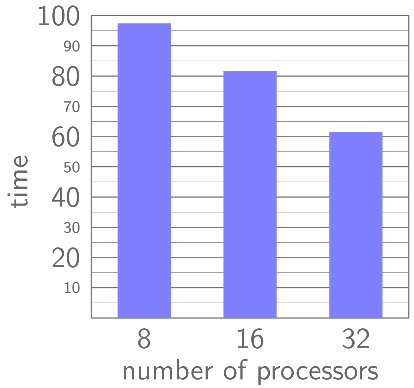
Name	Hyperion [Altix ICE 8200]
Where	**C**entre **I**nteruniversitaire de **C**alcul de **T**oulouse
Power	■ 352 nodes (bi-Intel "Nehalem" EP quad-core) ■ 4.5GB per core ■ 33TFlops ■ 334th in Top500 (June 2010)

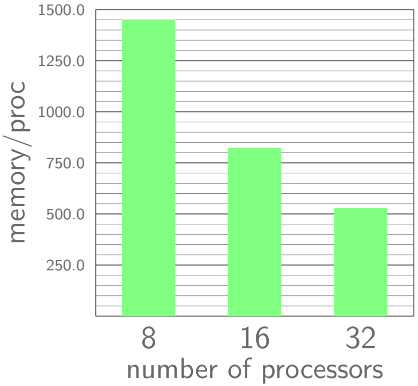
Context
The Problem

Problem	Wind energy (CEsA–FEUP, Porto)
Matrix	Symmetric, indefinite and cyclic band diagonal
Where	Discretization of the pressure equation
Size	$N_i \times N_j \times N_k = 96 \times 128 \times 128$ nodes 1,572,864

Numerical experiments: Factorization (1)

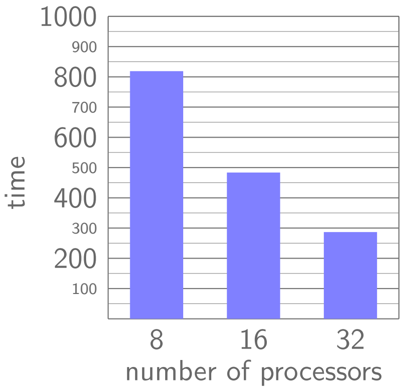
- 16 equal sized partitions
- $order(S) = 3,932,145$

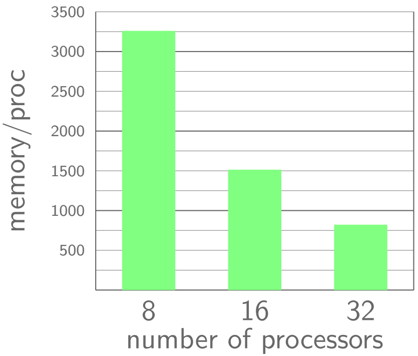




Numerical experiments: Factorization (2)

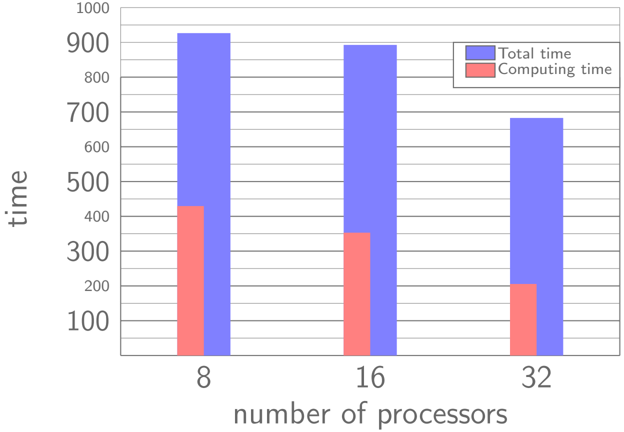
- 8 equal sized partitions
- $order(S) = 3,538,937$





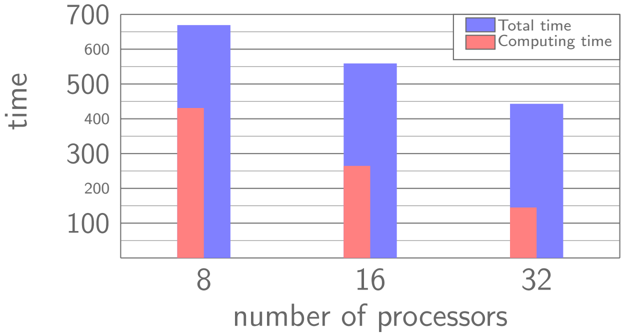
Numerical experiments: Solution (1)

- 16 equal sized partitions
- 156 iterations



Numerical experiments: Solution (2)

- 8 equal sized partitions
- 98 iterations



Conclusion

Conclusion:

- The three levels of parallelism well handled
- Decorrelation between the number of processors and the number of partitions

Prospects:

- Parallelize the block Cimmino acceleration
- Reduce the communication
- etc.

Conclusion

Conclusion:

- The three levels of parallelism well handled
- Decorrelation between the number of processors and the number of partitions

Prospects:

- Parallelize the block Cimmino acceleration
- Reduce the communication
- etc.

Thank You!