An Efficient Direct Algorithm for the Solution of the Fourth-Order Poisson Equation on Loosely Coupled Parallel Computers

F.X. Trias, M. Soria, F. García, O. Lehmkuhl

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

- Introduction
- Loosely coupled parallel computers
- Direct Schur - Fourier Decomposition (DSFD) overview
- Fourier diagonalization
- Direct Schur
- DSFD Benchmarks
- Iterative Schur
- Benchmarks
- Conclusions
Introduction - Governing equations

Direct numerical simulation of turbulent natural convection flows.

Navier-Stokes coupled with energy transport equation.

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

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 - Poisson equation

- Most DNS-LES codes use explicit or semiexplicit time-integration methods and segregated methods to solve the pressure-velocity coupling.
- In all them, at least one Poisson equation has to be solved per time step (or iteration).
- The discrete Poisson equation must discretized consistently with divergence and gradient operators, so that $L = MG$.
- Spectro-consistent schemes impose an additional condition that links divergence $M$ and convection operator $C$: If $Mv = 0$, then $C$ must be skew-symmetric.
- Thus, if a fourth-order spectro consistent discretization is desired, at least one fourth-order equation must be solved for each time step.
- This is an important difference. The second order matrix has 7 diagonals, while the fourth-order equation has 19 diagonal and is not diagonal-dominant.
- However, as the Laplacian operator is linear, its matrix depends only on the mesh and it remains constant during all the simulation, $10^5$ to $10^7$ time steps.
Why is Poisson equation so difficult for parallel computers?

Why is Poisson equation so difficult? A physical argument

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

Incompressibility \( \rightarrow \frac{\partial \rho}{\partial p} = 0 \rightarrow c = \infty \rightarrow \) local changes affect instantaneously all the domain

This behaviour is inheritted 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 \( M \approx 10^{5\cdots7} \)
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
Penta-diagonal matrix-vector product on a cluster and a Cray T3E

Minimum number of nodes per processor for efficiency > 50%?

T_{communication} = T_{computation}

- Cray T3E (300 MHz) versus 900 MHz K7 (100 Mbit/s fast ethernet)
- \( \frac{N}{P} \): number of nodes assigned to each processor.
How will be the parallel computers in 2010?

This is, will DSDF algorithm be more or less useful in the future?

A parallel computer can be characterized by:

- $F$: Number of effective operations per second
- $L$: Latency time in seconds
- $B$: Network bandwidth
- $P$: Number of processors

Thus, $FL$ is the number of operations that one processor can do during the latency time.

Will $FL$ increase or decrease?

If we restrict our attention to low cost parallel computers, that must be assembled with standard components, $F$ will increase but it is my opinion that $L$ will remain constant.

There is at least one reason for $L$ to remain constant: there is (almost) no demand for very low latency networks.

And for true parallel computers?
Alternatives to solve the Poisson equation

The main challenge for DNS 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 degrade with $P$
- Fast-Poisson solvers and FFT-based methods are restricted to determined 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 3D equation to a family of independent 2D equations** (this imposes certain restrictions)
  - The 2D equations are solved with a **Direct Schur** decomposition 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 arbitrary order 3D Poisson equations to machine accuracy** 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
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.
Heptadiagonal system to be solved: \( A^{3d} x^{3d} = b^{3d} \)

\[
a^p_{i,j,k} x^p_{i,j,k} + \sum_{nb} a_{i,j,k}^{nb} x_{i,j,k}^{nb} = b_{i,j,k}
\]

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:

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

where:

\[
x_{j,k} = [x_{1,j,k}, x_{2,j,k}, \cdots, x_{N_x,j,k}]^t
\]
Fourier Diagonalization (2/8)

**Second-order discretization**

\[
\begin{bmatrix}
A_{1,1}^p & A_{1,1}^n & \ldots & A_{1,1}^t \\
A_{2,1}^s & A_{2,2}^p & A_{2,1}^n & \ldots & A_{2,1}^t \\
\vdots & & & \ddots & \vdots \\
A_{j,k}^b & \ldots & A_{j,k}^s & A_{j,k}^p & A_{j,k}^n & \ldots & A_{j,k}^t \\
\vdots & & & \ddots & \vdots & & \vdots \\
A_{N_y,N_z}^b & \ldots & A_{N_y,N_z}^s & A_{N_y,N_z}^p & A_{N_y,N_z}^n & \ldots & A_{N_y,N_z}^t
\end{bmatrix}
\begin{bmatrix}
x_{1,1} \\
x_{2,1} \\
\vdots \\
x_{j,k} \\
\vdots \\
x_{N_y,N_z}
\end{bmatrix}
=
\begin{bmatrix}
b_{1,1} \\
b_{2,1} \\
\vdots \\
b_{j,k} \\
\vdots \\
b_{N_y,N_z}
\end{bmatrix}
\]

\[A_{j,k}^b x_{j,k-1} + A_{j,k}^s x_{j-1,k} + A_{j,k}^p x_{j,k} + A_{j,k}^n x_{j+1,k} + A_{j,k}^t x_{j,k+1} = b_{j,k}\]

- \(A_{j,k}^n, A_{j,k}^s, A_{j,k}^t\) and \(A_{j,k}^b\) are \(N_x \times N_x\) diagonal matrices.
- \(A_{j,k}^p\) are \(N_x \times N_x\) circulant tridiagonal matrices. They contain the coefficients that link the unknowns of plane \(i\) with the unknowns of the neighbouring planes \(i-1\) and \(i+1\). If they could be diagonalized, each plane could be solved separately.

\[
A_{j,k}^p = \begin{bmatrix}
\beta & \alpha & \alpha & \ldots \\
\alpha & \beta & \alpha & \ldots \\
\alpha & \ldots & \alpha & \beta \\
\end{bmatrix}
\]

where \(\alpha = a_{j,k}^w = a_{j,k}^c\) and \(\beta = a_{j,k}^p\)
Fourier Diagonalization (3/8)

Second-order discretization

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:

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

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

$$\overline{x}_1 = \frac{2}{N_x} \sum_{i=1}^{N_x} x_i$$

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

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

$$\overline{x}_N = \sum_{i=1}^{N_x} x_i (-1)^i$$
Fourier Diagonalization (4/8)

Second-order discretization

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

$$Q^{-1} A_{j,k}^p Q = \lambda_{j,k}$$

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

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

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

$$
A_{j,k}^b \bar{x}_{j,k-1} + A_{j,k}^s \bar{x}_{j-1,k} + \lambda_{j,k} \bar{x}_{j,k} + A_{j,k}^n \bar{x}_{j+1,k} + A_{j,k}^t \bar{x}_{j,k+1} = \bar{b}_{j,k}
$$

As the non-diagonal entries of matrices $A_{j,k}^p$ have been eliminated, unknown $\bar{x}_{i,j,k}$ is only coupled with unknowns in the same plane $i$.
Fourier Diagonalization (5/8)
Second-order discretization

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

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

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
Fourier Diagonalization (6/8)

Fourth-order discretization

The system $A^{3d} x^{3d} = b^{3d}$ than arises from the fourth-order discretization has 19 diagonals.

$A_{p,j,k}^p$ are heptadiagonal $N_x \times N_x$ circulant matrices,

$$A_{p,j,k}^p = \begin{bmatrix}
\beta & \alpha_1 & \alpha_2 & \alpha_3 & \alpha_3 & \alpha_2 & \alpha_1 \\
\alpha_1 & \beta & \alpha_1 & \alpha_2 & \alpha_3 & \alpha_3 & \alpha_2 \\
\alpha_2 & \alpha_1 & \beta & \alpha_1 & \alpha_2 & \alpha_3 & \alpha_3 \\
\alpha_3 & \alpha_3 & \alpha_2 & \alpha_1 & \beta & \alpha_1 & \alpha_2 \\
\alpha_2 & \alpha_3 & \alpha_3 & \alpha_2 & \alpha_1 & \beta & \alpha_1 \\
\alpha_1 & \alpha_2 & \alpha_3 & \alpha_3 & \alpha_2 & \alpha_1 & \beta 
\end{bmatrix}$$

where

$$\begin{align*}
\alpha_1 &= a_{j,k}^w = a_{j,k}^e \\
\alpha_2 &= a_{j,k}^{w2} = a_{j,k}^{e2} \\
\alpha_3 &= a_{j,k}^{w3} = a_{j,k}^{e3} \\
\beta &= a_{j,k}^p
\end{align*}$$
Fourier Diagonalization (7/8)
Fourth-order discretization

However, as $A^p_{j,k}$ are also $N_x \times N_x$ \textbf{circulant} matrices, they have the same base of eigenvectors

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

Now, the \textbf{eigenvalues} will be given by

$$\lambda_j = \beta + 2 \sum_{i=1}^{3} \alpha_i \cos\left(\frac{2\pi i (j - 1)}{N_x}\right) \quad j = 1, \ldots, N_x$$

And, after diagonalization, we will obtain a \textbf{13-diagonal scalar equation system} in $\bar{x}$.

$$\sum_{d=1}^{3} \left( a_{j,k}^{bd} \bar{x}_{i,j,k-d} + a_{j,k}^{sd} \bar{x}_{i,j-d,k} + a_{j,k}^{nd} \bar{x}_{i,j+d,k} + a_{j,k}^{td} \bar{x}_{i,j,k+d} \right) + a_{i,j,k}^{p} \bar{x}_{i,j,k} = \bar{b}_{i,j,k}$$
In conclusion,

- **Only** the eigenvalues calculation needs to be **changed** respect to Fourier diagonalization of the system that arises from a second-order discretization.

- The **computational cost** of the Fourier diagonalization does **NOT**\(^1\) **depend** on the **order** of accuracy of the Poisson equation.

---

\(^1\)If we do not consider the slight differences in the eigenvalues calculation, of course.
Direct Schur Decomposition - Main ideas (1/7)

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

Partition and reordering of the unknowns for the second-order(left) and fourth-order(right) discrete Poisson equations.

**Only one all-to-all communication** episode is needed
No property of $A$ (i.e., symmetry or positive-definiteness) or the underlying mesh is required

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 $Ax = b$ than to evaluate the matrix-vector product $A^{-1}b$
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:

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

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

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

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

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) $[\tilde{A}_{s,s}^p]_c \leftarrow A_{s,p} t$

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) $\tilde{b}_s^p \leftarrow A_{s,p} t$
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 \text{ 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 \( x_p \)
The algorithm is not conceptually complex but its efficiency depends on certain details:

- In order to evaluate \( \tilde{A}_{s,s}^{-1} b_s \), 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 \( \tilde{A}_{s,s}^t \) is evaluated and then inverse and transpose operators are permuted.
- **Distributed storage** of \( \tilde{A}_{s,s}^{-1} \). Each processor only stores the rows of the inverse needed to perform its part of \( \tilde{A}_{s,s}^{-1} b_s \).
- 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 \( \tilde{A}_{s,s}^{-1} \) is treated as **dense** and **stored by rows** at the different processors.
Benchmark (1/9) - Speedup of DSD on a PC cluster

Totally direct, second order equation

900 MHz K7 processors; Switched 100 Mbits/s network

The cost of DSFD is roughly $N \times$ times the cost of one pentadiagonal equation. Irregular behaviour for a fixed $N$ is due to different bandwidths 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 (2/9) - DSD solution time on a PC cluster
Totally direct, second order equation
900 MHz K7 processors; Switched 100 Mbits/s network

For $N = 125 \times 10^3$, $P = 24$ Direct Shur Complement is $\approx 30$ times faster than sequential ACM multigrid with $\epsilon^* = 10^{-3}$
Benchmark (3/9) - DSFD solution time on a PC cluster

Totally direct, second order equation

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 (4/9) - Decomposition of total time with 36 processors
Direct Solution of Second-order Poisson Equation
Benchmark (5/9) - Decomposition of total time with 24 processors
Direct Solution of Second- and Fourth-order Poisson Equations

2600 MHz K7 processors; Switched 100 Mbits/s network
Benchmark (6/9) - DSFD solution time on a PC cluster
Direct Solution of Second- and Fourth-order Poisson Equations

2600 MHz K7 processors; Switched 100 Mbits/s network
Benchmark (7/9) - Speedup of DSFD on a PC cluster
Direct Solution of Second- and Fourth-order Poisson Equations

2600 MHz K7 processors; Switched 100 Mbits/s network
DSD - RAM Memory Requirements (1/2)

The total number of array positions that each processor needs is approximately

\[ N_t^{2d}(M, I, N_{sp}, P) \approx 3M \left( \frac{N_{sp}}{P} \right)^{\frac{3}{2}} + 8I^2 N_{sp} \frac{P^{1/2} - 1}{P^{1/2}} \]

where \( P \) is the number of processors, \( I \) is the interface size, \( M \) the scheme stencil size and \( N_{sp} \) the number of unknowns per plane.

As \( M \) and \( I \) are increasing functions of the order of the scheme\(^2\), RAM memory requirements **may become a serious obstacle** to use DSD for **high-order schemes**.

\(^2\)In our case:

\[ M = o - 1 \quad I = o - 1 \]
In particular,

- **Second-order:**

\[
N_t^{2d} \approx 3 \left( \frac{N_{sp}}{P} \right)^{3/2} + 8N_{sp} \frac{P^{1/2} - 1}{P^{1/2}}
\]

local band–LU problems + interface equation

- **Fourth-order:**

\[
N_t^{2d} \approx 9 \left( \frac{N_{sp}}{P} \right)^{3/2} + 72N_{sp} \frac{P^{1/2} - 1}{P^{1/2}}
\]

local band–LU problems + interface equation

Note that the **main difficulty** arises from the **interface equation** whose RAM memory requirement does NOT decrease with the number of processors.
The aim is to **reduce the RAM memory requirements** maintaining the results accurate enough with **reasonable computing times**.

To do so:

- The Fourier diagonalization is carried out as in the fully direct methods
- To solve each 2D equation, the Direct Schur part is replaced by a stationary iterative method

Let each 2D equation be:

\[ Ax = b \]

Matrix \( A \) is separated into a \( A_d \) (to be solved with DSD) and \( A_i \)

\[ A = A_d + A_i \]

The **iteration** takes the form

\[ A_d x^{k+1} = b - A_i x^k \]

Now the question is how to choose \( A_d \) and \( A_i \).
Stationary Iterative Approach for Schur Problems (3/3)

Option 1. Save memory in the local band-LU problems and in the interface equation $A^{1,1}_d$: Reduce the stencil to one node in all the domain

Option 2. Save memory only in the interface equation $A^{3,1}_d$: it reduces the stencil size to one node but only for the nodes near the interface
Benchmark (8/9) - DSFD Iterative approach (4th order eq.)

CPU time versus RAM memory

2600 MHz K7 processors; Switched 100 Mbits/s network; \( N \approx 3.1 \times 10^6 \); \( \epsilon = 10^{-6} \)

With the same RAM memory requirement **slightly smaller** computing times are obtained when both iterative approaches \( A_{d}^{3,1} \) and \( A_{d}^{1,1} \) are properly combined.
Benchmark (9/9) - DSFD Iterative approach (4rth order eq.)
number of iterations needed for each plane

24 2600 MHz K7 processors; Switched 100 Mbits/s network; \( N = 64 \times 156 \times 312 \); \( \epsilon = 10^{-6} \)

The number of iterations needed for high-frequency planes is smaller because the systems are better conditioned.
Conclusions

- After a pre-processing stage, DSFD methods allows to solve second and fourth order Poisson equations to machine accuracy with just one message.
- This allows the algorithm to run efficiently even on loosely coupled parallel computers, where it is significantly faster than Krylov subspace solvers.
- Even on PC clusters with 100 Mbits/s networks, the algorithm scales well up to 36-48 processors.
- For the situations where the RAM memory and not the network performance is the limiting factor, DSFD can be used with a stationary iterative algorithm.
- DSFD algorithm has allowed us to run DNS simulations on a low cost PC cluster.
- The DSD part can be extended to non-structured meshes.
Please let me finish with a quote from the paper by Verstappen and Veldman (J.Comp.Physics, 2003): 

“More than one and a half century ago Claude Navier (1822) and George Stokes (1845) derived an excellent mathematical model for turbulent flow”