# Configuration of a linear solver for linearly implicit time integration and efficient data transfer in parallel thermo-hydraulic computations

Ravil Dorozhinskii

Computational Science and Engineering

March 21, 2019

# Outline I

# Overview of ATHLET and NuT software

# Mathematical Model

1. Liquid mass

$$\frac{\partial((1-\alpha)\rho_l)}{\partial t} + \nabla((1-\alpha)\rho_l \vec{w}_l) = -\psi \tag{1}$$

2. Vapor mass

$$\frac{\partial(\alpha\rho_v)}{\partial t} + \nabla(\alpha\rho_v \vec{w}_v) = \psi \tag{2}$$

3. Liquid momentum

$$\frac{\partial((1-\alpha)\rho_l \vec{w}_l)}{\partial t} + \nabla((1-\alpha)\rho_l \vec{w}_l \vec{w}_l) + \nabla((1-\alpha)p) = \vec{F}_l \tag{3}$$

4. Vapor momentum

$$\frac{\partial(\alpha\rho_v \vec{w}_v)}{\partial t} + \nabla(\alpha\rho_v \vec{w}_v \vec{w}_v) + \nabla(\alpha p) = \vec{F}_v \tag{4}$$

# Mathematical Model

5. Liquid energy

$$\frac{\partial\left[(1-\alpha)\rho_l(h_l+\frac{1}{2}\vec{w}_l\vec{w}_l-\frac{p}{\rho_l})\right]}{\partial t}+\nabla\left[(1-\alpha)\rho_l\vec{w}_l(h_l+\frac{1}{2}\vec{w}_l\vec{w}_l)\right]=-p\frac{\partial(1-\alpha)}{\partial t}+E_l \tag{5}$$

6. Vapor energy

$$\frac{\partial\left[\alpha\rho_v(h_v+\frac{1}{2}\vec{w}_v\vec{w}_v-\frac{p}{\rho_v})\right]}{\partial t}+\nabla\left[\alpha\rho_v\vec{w}_v(h_v+\frac{1}{2}\vec{w}_v\vec{w}_v)\right]=-p\frac{\partial\alpha}{\partial t}+E_v \tag{6}$$

7. Volume vapor fraction

$$\alpha=\frac{V_v}{V} \tag{7}$$

*According to [3]*

# Mathematical Model: from PDEs to ODEs

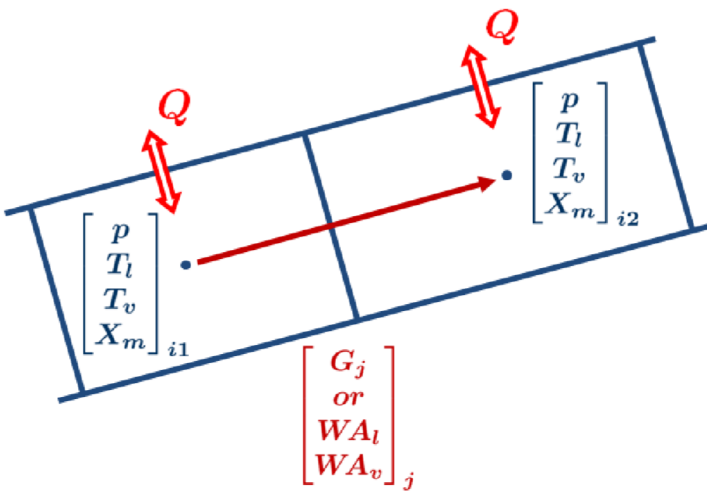



Figure 1: ATHLET: one dimensional **Finite Volume** formulation of the problem [13]

The system is transformed to a non-autonomous system of **ordinary differential equations** and expressed as an **initial value problem** after spatial finite-volume integration and some mathematical transformations [3].

$$\frac{dy}{dt} = f(t, y),\ t_0 \leq t \leq t_F\ y(t_0) = y_0 \tag{8}$$

where $y \in \mathbb{R}^N$ is a composite vector of variables, $f$ is a non-linear function such that $f : \mathbb{R} \times \mathbb{R}^N \supset \Omega \rightarrow \mathbb{R}^N$ .

# Numerical Integration: 6-stage W-method

$$(I - h_i J)\delta y_{ij} = h_i f(t_0 + j \cdot h_i, y_0 + \sum_{l=0}^{j-1} \delta y_{il}) + h^2 \frac{\partial f_0}{\partial t} \tag{9}$$

where $j = 0, \ldots, i-1$; $h_i = h/i$; $i = 1, 2, 3$

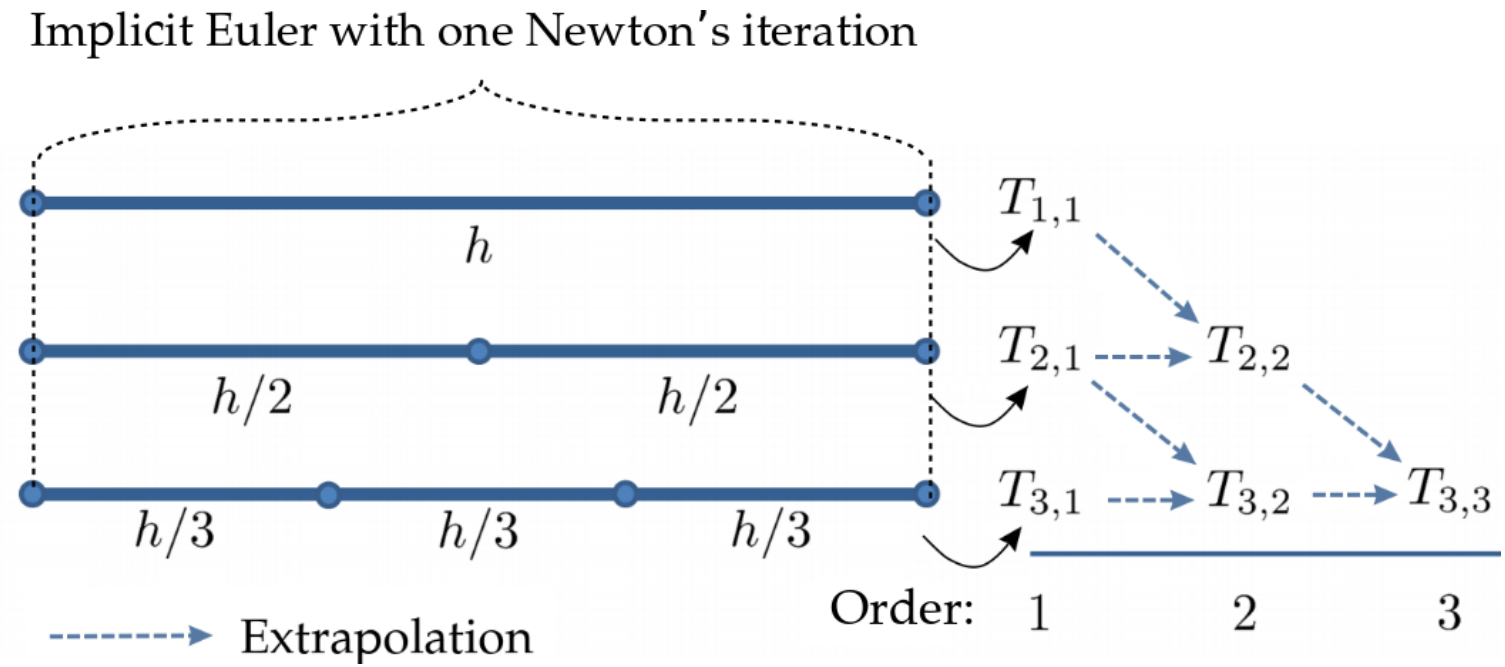



Figure 2: A general view on 6-stage W-method [13]

# Problem Statement

# Solving Sparse Linear Systems

Numerical **integration** of a system of ODEs by means of **W-methods**:

$$(I - h_i J)\delta y_{ij} = h_i f(t_0 + j \cdot h_i, y_0 + \sum_{l=0}^{j-1} \delta y_{il}) + h^2 \frac{\partial f_0}{\partial t} \tag{10}$$

can be considered as a solution of a **sequence of linear systems** from another point of view:

$$A_i \delta y_{ij} = b_{ij} \tag{11}$$

where $A = (I - h_i J)$ is a $\mathbb{R}^N \times \mathbb{R}^N$ non-singular **sparse matrix**; $\delta y_{ij}$ and $b_{ij}$ are $\mathbb{R}^N$ vectors; $i = 1, 2, 3$; $j = 0, \ldots, i - 1$

## NOTE

Computational **burden** of the W-method mainly lies in **solving of sparse linear systems** beside evaluation of non-linear function $f$ of Eq. 8

# Methodology and Matrix Sets

# Static Solver Configuration

- ATHLET is a CFD tool dedicated to **transient problems**
- Additionally, **topology** of hydrolic circuits can **be changed** during simulation-time
- Hence, the Jacobian matrix **structure** can **vary** significantly during numerical itegration

## Therefore:

1. Configuration of a linear solver in run-time is compute-intensive and time consuming
2. Reults of dynamic solver configuration can be ambiguous and difficult to interpret
3. A feasible and doable approach is **static solver configuration**

# GRS Matrix Set

GRS **matrix set** was generated by **running** the most **common GRS simulations** in ATHLET and **stopping** them **somewhere in the middle**. The corresponding shifted Jacobian matrices ware saved in the PETSc binary format.

| Name | n | nnz | nnz / n | Approximate Condition Number |
|---|---|---|---|---|
| pwr-3d | 6009 | 32537 | 5.4147 | 1.019e+07 |
| cube-5 | 9325 | 117897 | 12.6431 | 1.592e+09 |
| cube-64 | 100657 | 1388993 | 13.7993 | 7.406e+08 |
| cube-645 | 1000045 | 13906057 | 13.9054 | 6.474e+08 |
| k3-2 | 130101 | 787997 | 6.0568 | 1.965e+15 |
| k3-18 | 1155955 | 7204723 | 6.2327 | 1.947e+12 |

Table 1: GRS matrix set

# SuiteSparse Matrix Set

**SiteSparse** matrix set was generated by **downloading** a dozen of matrices from SuiteSparse Matrix Collection [5], [6], with the **aim** of **comparison** and **verification** of solver configurations

| Name | n | nnz | nnz / n | Approximate Condition Number |
|---|---|---|---|---|
| cant | 62451 | 4007383 | 64.1684 | 5.082e+05 |
| consph | 83334 | 6010480 | 72.1251 | 2.438e+05 |
| CurlCurl_3 | 1219574 | 13544618 | 11.1060 | 2.105e+05 |
| Geo_1438 | 1437960 | 63156690 | 43.9210 | 4.677e+05 |
| memchip | 2707524 | 13343948 | 4.9285 | 1.305e+07 |
| PFlow_742 | 742793 | 37138461 | 49.9984 | 5.553e+06 |
| pkustk10 | 80676 | 4308984 | 53.4110 | 5.589e+02 |
| torso3 | 259156 | 4429042 | 7.0903 | 2.456e+03 |
| x104 | 108384 | 8713602 | 80.3956 | 3.124e+05 |

Table 2: SuiteSparse matrix set

# Hardware

| | HW1 (GRS) | HW2 (LRZ Linux) |
|---|---|---|
| CPU(s) | 20 | 28 |
| On-line CPU(s) list | 0-19 | 0-27 |
| Thread(s) per core | 1 | 1 |
| Core(s) per socket | 10 | 14 |
| Socket(s) | 2 | 2 |
| NUMA node(s) | 2 | 4 |
| Model name | E5-2680 v2 | E5-2697 v3 |
| Stepping | 4 | 2 |
| CPU MHz | 1200.0 | 2036.707 |
| L1 d/i cache | 32K/32K | 32K/32K |
| L2 cache | 256K | 256K |
| L3 cache | 25600K | 17920K |
| NUMA node0 CPU(s) | 0-9 | 0-6 |
| NUMA node1 CPU(s) | 10-19 | 7-13 |
| NUMA node2 CPU(s) | - | 14-20 |
| NUMA node3 CPU(s) | - | 21-27 |

Table 3: Hardware specification

# Experimental setup

## Libraries and Compiler

1. PETSc version 3.10
2. OpenMPI version 3.1.1
3. Intel Compiler 18

# Overview of Solver Types

# Iterative methods

- Given an initial guess
- An iterative method generates a sequence of approximate solutions by means of a specific rule
- Depending on a method and the given problem,
- there may exist certain conditions such that the sequence eventually converges to the exact solution

- There exist two families of iterative methods: *stationary* and *Krylov-based* methods
- Nowadays, Krylov methods dominate in the field of scientific computing
  - because of their rather fast convergence
  - in case of solving well conditioned systems
  - or/and a ”*good*” initial guess

# Krylov-based methods

- *The key idea* is to construct an approximate solution
- as a linear combination of vectors $b$, $Ab$, $A^2b$, $A^3b, \ldots A^{n-1}b$
- known as Krylov subspace $\mathcal{K}_n$
- where, without of lost of generality, the initial guess $x_0$ is equal to zero
- At each iteration, the subspace is expanded by adding and evaluating the next vector in the sequence

- the methods define and expand another subspace $\mathcal{L}_n$
- such that $r_n = b - Ax_n \perp \mathcal{L}_n$
- which is known as the Petrov-Galerkin condition
- A construction of subspace $\mathcal{L}_n$ is defined by a method
- and based on matrix properties

# Aspects

## Parallelization Aspects

- Iterative methods usually make use of simple linear algebra kernels e.g. dot products, matrix-vector products, etc.
- Therefore, the methods efficiently make use of data-based parallelism

## Numerical Accuracy

- Convergence to the exact solution depends on a value of the condition number of a matrix
- Usually requires a linear transformation known as preconditioning
  - to reduce condition number and accelerate convergence

# Preconditioners

| Package name | Origin | Method | Tuning parameters | Comments |
|---|---|---|---|---|
| block Jacobi | PETSc | block Jacobi | -pc_bjacobi_blocks<br>-sub_pc_type | - |
| additive Schwarz | PETSc | additive Schwarz | -pc_asm_blocks<br>-pc_asm_overlap<br>-pc_asm_type<br>-pc_asm_local_type<br>-sub_pc_type | - |
| euclid | hypre | ILU(k) | -nlevel<br>-thresh<br>-filter | deprecated form PETSc |
| pilut | hypre | ILU(t) | -pc_hypre_pilut_tol<br>-pc_hypre_pilut_maxiter<br>-pc_hypre_pilut_factorrowsize | - |
| parasail | hypre | SPAI | -pc_hypre_parasails_nlevels<br>-pc_hypre_parasails_thresh<br>-pc_hypre_parasails_filter | - |
| SPAI | Grote, Barnard | SPAI | -pc_spai_epsilon<br>-pc_spai_nbstep<br>-pc_spai_max<br>-pc_spai_max_new<br>-pc_spai_block_size<br>-pc_spai_cache_size | - |
| BoomerAMG | hypre | algebraic multigrid | -pc_hypre_boomeramg_cycle_type<br>-pc_hypre_boomeramg_max_levels<br>-pc_hypre_boomeramg_max_iter<br>-pc_hypre_boomeramg_tol<br>etc. | 39 tuning parameters in total |

Table 4: Parallel preconditioning algorithms available in PETSC

# Direct Sparse Methods

Direct sparse methods combine the main advantages of direct and iterative methods

- numerical accuracy of the methods is comparable with the standard Gaussian Elimination process
- complexity is bounded by $O(n^2)$ due to efficient treatment of sparsity

A solution of a system of equations is computed:

- by means of forward and backward substitutions
- using $LU$ decomposition of the corresponding matrix.

# Direct Sparse Methods: Speech

- The multifrontal method is probably the most representative example of direct sparse solvers introduced by Duff and Reid
- The method is an improved version of the frontal method
- which can compute independent fronts in parallel.
- A front, also called a frontal matrix, can be considered as a small dense matrix
- resulting from a column elimination of the original system.
- There also exist left- and right-looking vatiants of the multifrontal method

Let's consider the multifrontal methods as an example

To keep the overview rather simple, we assume that matrix $A$ is **symmetric positive definite** and **sparse**

# Multifrontal Method: I

$$A = LDL^T \qquad \text{with} \quad (D)_{ii} > 0 \tag{12}$$

Given a rule and a sparsity pattern, one can build an elimination tree

$$p = min(i > j | l_{ij} \neq 0) \tag{13}$$

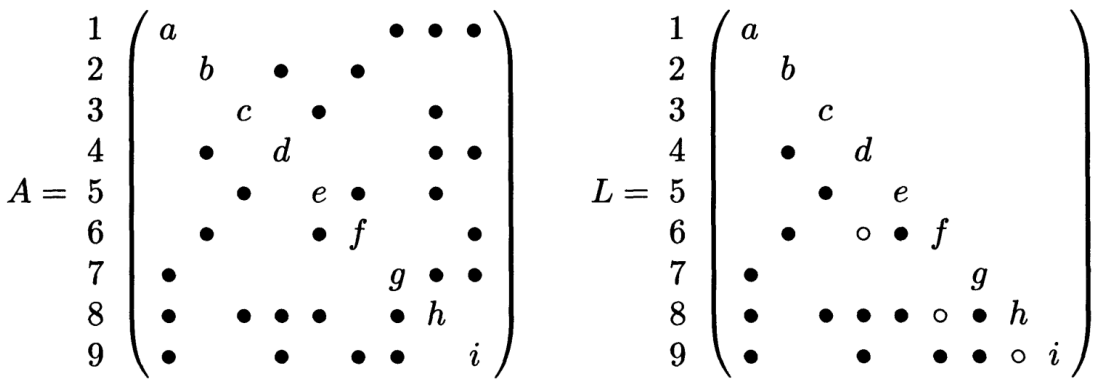



Figure 3: A sparse matrix and its Cholesky factor, [10]

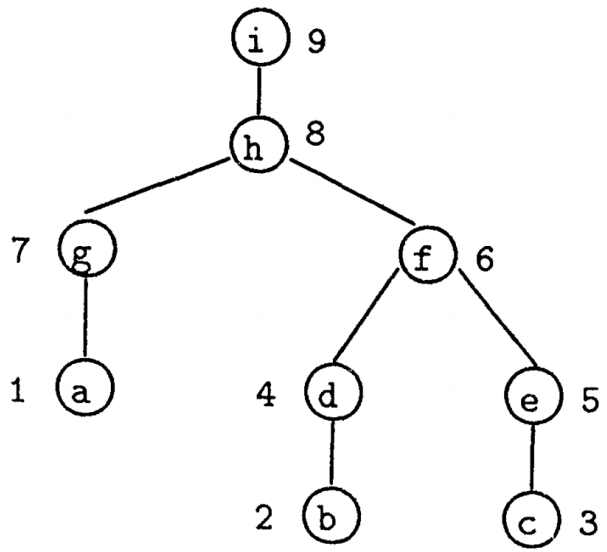



Figure 4: An elimination tree, [10]

# Multifrontal Method: II

The fundamental idea of the multifrontal method spins around frontal $F$ and update matrices $\hat{U}$

$$F_j = Fr_j + \hat{U}_j = \begin{bmatrix} a_{j,j} & a_{j,i_1} & a_{j,i_2} & \dots & a_{j,i_r} \\ a_{i_1,j} & & & & \\ a_{i_2,j} & & & & \\ \vdots & & & 0 & \\ a_{i_r,j} & & & & \end{bmatrix} + \hat{U}_j \tag{14}$$

where $i_0$, $i_1$, $i_2$, $\dots$, $i_r$ are row subscripts of non-zeros in $L_{*j}$ where $i_0 = j$; $r$ is the number of off-diagonal non-zero elements.

$$\hat{U}_j = - \sum_{k \in T[j]-j} \begin{bmatrix} l_{j,k} \\ l_{i_1,k} \\ \vdots \\ l_{i_r,k} \end{bmatrix} \begin{bmatrix} l_{j,k} & l_{i_1,k} & \dots & l_{i_r,k} \end{bmatrix} \tag{15}$$

where $\hat{U}_j$ can be treated as the second term of the Schur complement

# Multifrontal Method: III

Let's consider factorization of a 2-by-2 block matrix $A$

$$A = \begin{bmatrix} B & V^T \\ V & C \end{bmatrix} = \begin{bmatrix} L_B & 0 \\ VL_B^{-T} & I \end{bmatrix} \begin{bmatrix} I & 0 \\ 0 & Sr \end{bmatrix} \begin{bmatrix} L_B^T & L_B^{-1}V^T \\ 0 & I \end{bmatrix} \tag{16}$$

Assuming that block $B$ has already been factorized: $B = L_B L_B^T$

The Schur complement can be viewed as:

$$Sr = C - VB^{-1}V^T \tag{17}$$

*Dense Linear Algebra:* $-VB^{-1}V^T$ *Multifrontal method:* $\hat{U}_j$

$$-\sum_{k=1}^{j-1} \begin{bmatrix} l_{j,k} \\ \vdots \\ l_{n,k} \end{bmatrix} \begin{bmatrix} l_{j,k} & \dots & l_{n,k} \end{bmatrix} \tag{18}$$

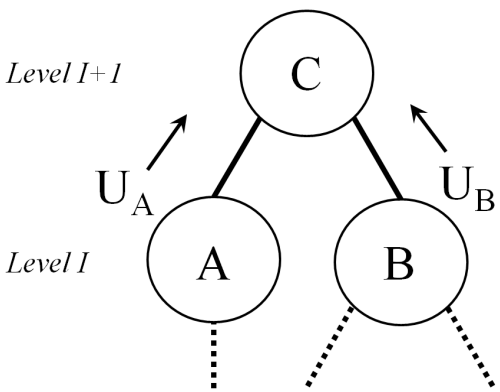
$$-\sum_{k \in T[j]-j} \begin{bmatrix} l_{i_0,k} \\ \vdots \\ l_{i_r,k} \end{bmatrix} \begin{bmatrix} l_{i_0,k} & \dots & l_{i_r,k} \end{bmatrix} \tag{19}$$

# Multifrontal Method: IV

After column $j$ factorization:

$$\hat{F}_j = \begin{bmatrix} l_{j,j} & \dots & 0 \\ \vdots & I & \\ l_{i_r,j} & & \end{bmatrix} \begin{bmatrix} 1 & \dots & 0 \\ \vdots & U_j & \\ 0 & & \end{bmatrix} \begin{bmatrix} l_{j,j} & \dots & l_{i_r,j} \\ \vdots & I & \\ 0 & & \end{bmatrix} \tag{20}$$

where sub-matrix $U_j$ represents the full update from all descendants of node $j$ and node $j$ itself




$$F_j = \begin{bmatrix} a_{j,j} & a_{j,i_1} & a_{j,i_2} & \dots & a_{j,i_r} \\ a_{i_1,j} & & & & \\ a_{i_1,j} & & & & \\ \vdots & & & 0 & \\ a_{i_r,j} & & & & \end{bmatrix} \boxplus U_{c_1} \boxplus \dots \boxplus U_{c_s} \tag{21}$$

Figure 5:
Information flow

# Supernodal Method

- In practice, an improved version of the multifrontal method is used
- A super-node is formed by a set of contiguous columns
- which have the same off-diagonal sparsity structure

$$\mathcal{F}_j = \begin{bmatrix} a_{j,j} & a_{j,j+1} & \dots & a_{j,j+t} & a_{j,i_1} & \dots & a_{j,i_r} \\ a_{j+1,j} & a_{j+1,j+1} & \dots & a_{j+1,j+t} & a_{j+1,i_1} & \dots & a_{j+1,i_r} \\ \vdots & \vdots & \dots & \vdots & & & \\ a_{j+t,j} & a_{j+t,j+1} & \dots & a_{j+t,j+t} & a_{j+t,i_1} & \dots & a_{j+t,i_r} \\ a_{i_1,j} & a_{i_1,j+1} & \dots & a_{i_1,j+t} & & & \\ \vdots & \vdots & \dots & \vdots & & 0 & \\ a_{i_r,j} & a_{i_r,j+1} & \dots & a_{i_r,j+t} & & & \end{bmatrix} \begin{matrix} \\ \\ \\ \\ \\ \\ \\ \end{matrix} \begin{matrix} \\ \\ \\ \\ \\ \\ \\ \end{matrix} \begin{matrix} \\ \\ \\ \\ \\ \\ \\ \end{matrix} \begin{matrix} \\ \\ \\ \\ \\ \\ \\ \end{matrix} \begin{matrix} \\ \\ \\ \\ \\ \\ \\ \end{matrix} \begin{matrix} \\ \\ \\ \\ \\ \\ \\ \end{matrix} \begin{matrix} \\ \\ \\ \\ \\ \\ \\ \end{matrix} \quad \begin{matrix} \\ \\ \\ \\ \\ \\ \\ \end{matrix} \begin{matrix} \\ \\ \\ \\ \\ \\ \\ \end{matrix} U_{c_1} \dots \begin{matrix} \\ \\ \\ \\ \\ \\ \\ \end{matrix} \begin{matrix} \\ \\ \\ \\ \\ \\ \\ \end{matrix} U_{c_s} \tag{22}$$

# Parallelization Aspects

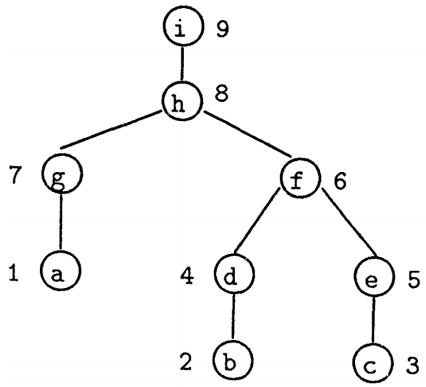



Figure 6: Original elimination tree

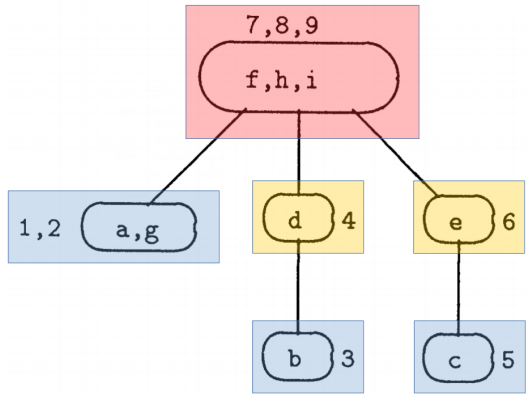



Figure 7: Supernodal/assembly tree

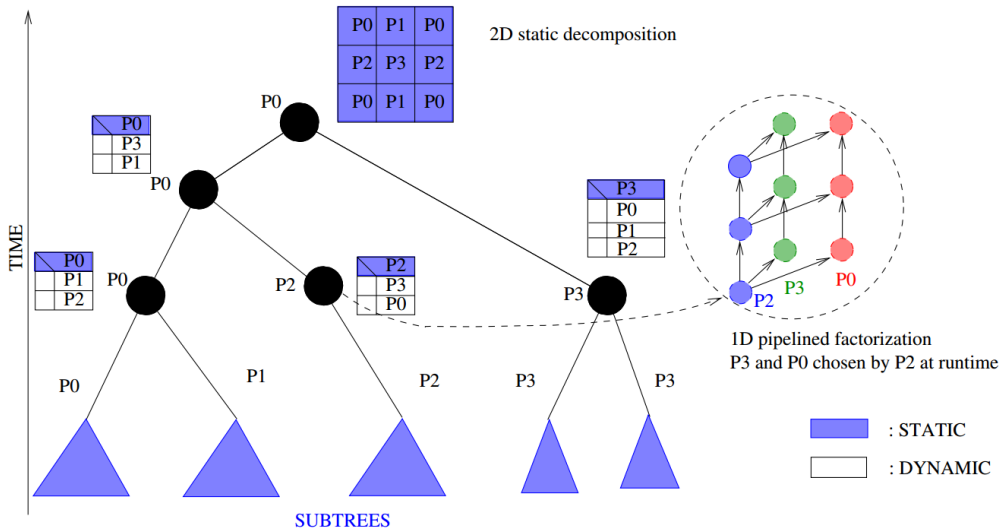



Figure 8: Static and dynamic scheduling in MUMPS, [8]

# Numerical Accuracy: I

- Pivoting is a big issue in direct sparse methods
  - analysis: absence of numerical information
  - numerical factorization: it may distort all prediction made during the analysis phases i.e.
    - fill-in prediction
    - load balancing

Therefore, threshold pivoting is commonly used for direct sparse methods

Threshold pivoting means that a pivot $|a_{i,i}|$ is accepted if it satisfies:

$$|a_{i,i}| \geq \alpha \times max_{k=i\ldots n}|a_{k,i}| \tag{23}$$

where $\alpha \in [0, 1]$ and $k = i \ldots n$ represents row indices of column $i$ within the fully summed block of a frontal matrix.

# Numerical Accuracy: II

- In case of small values of $\alpha$
- solutions can be numerically inaccurate
- may demand to perform solution refinements

As an example, solution accuracy can be improved using:

- iterative refinement method based a on solution residual
- resulting *LU* decomposition can be used as a preconditioner for a Krylov-based method e.g. GMRES

# Results and Conclusion

- As the first step, various preconditioning algorithms were tested
  - A coarse grid search was used
  - with maximum 3 values for each tuning parameter
  - starting from the default towards more accurate values
  - Testing showed there was no preconditioning algorithm
  - that could result in convergence for the entire set of matrices
- Even if we can find an algorithm and suitable parameter settings
  - which will result in convergence of the entire matrix set
  - there is no guarantee that it will work in all steps during a simulation
  - Iterative methods cannot fulfill *robustness* criterion
- Therefore, direct sparse methods is only one way to go
- In spite of limited tree-task parallelism

# Overview of available Direct Sparse Solver (DSS) libraries

# List of parallel DSSs

| Package | Method | Matrix Types | PETSc Interface | License |
|---|---|---|---|---|
| Clique | Multifrontal | Symmetric | Not Officially | Open |
| MF2 | Multifrontal | Symmetric pattern | No | - |
| DSCPACK | Multifrontal | SPD | No | Open |
| MUMPS | Multifrontal | General | Yes | Open |
| PaStiX | Left looking | General | Yes | Open |
| PSPASES | Multifrontal | SPD | No | Open |
| SPOOLES | Left-looking | Symmetric pattern | No | Open |
| SuperLU_DIST | Right-looking | General | Yes | Open |
| symPACK | Left-Right looking | SPD | No | Open |
| S+ | Right-lookin | General | No | - |
| PARDISO | Multifrontal | General | No | Commercial |
| WSMP | Multifrontal | General | No | Commercial |

Table 5: A list of direct sparse linear solvers adapted for distributed-memory computations, [9], [4]

# Comparisons of DSSs

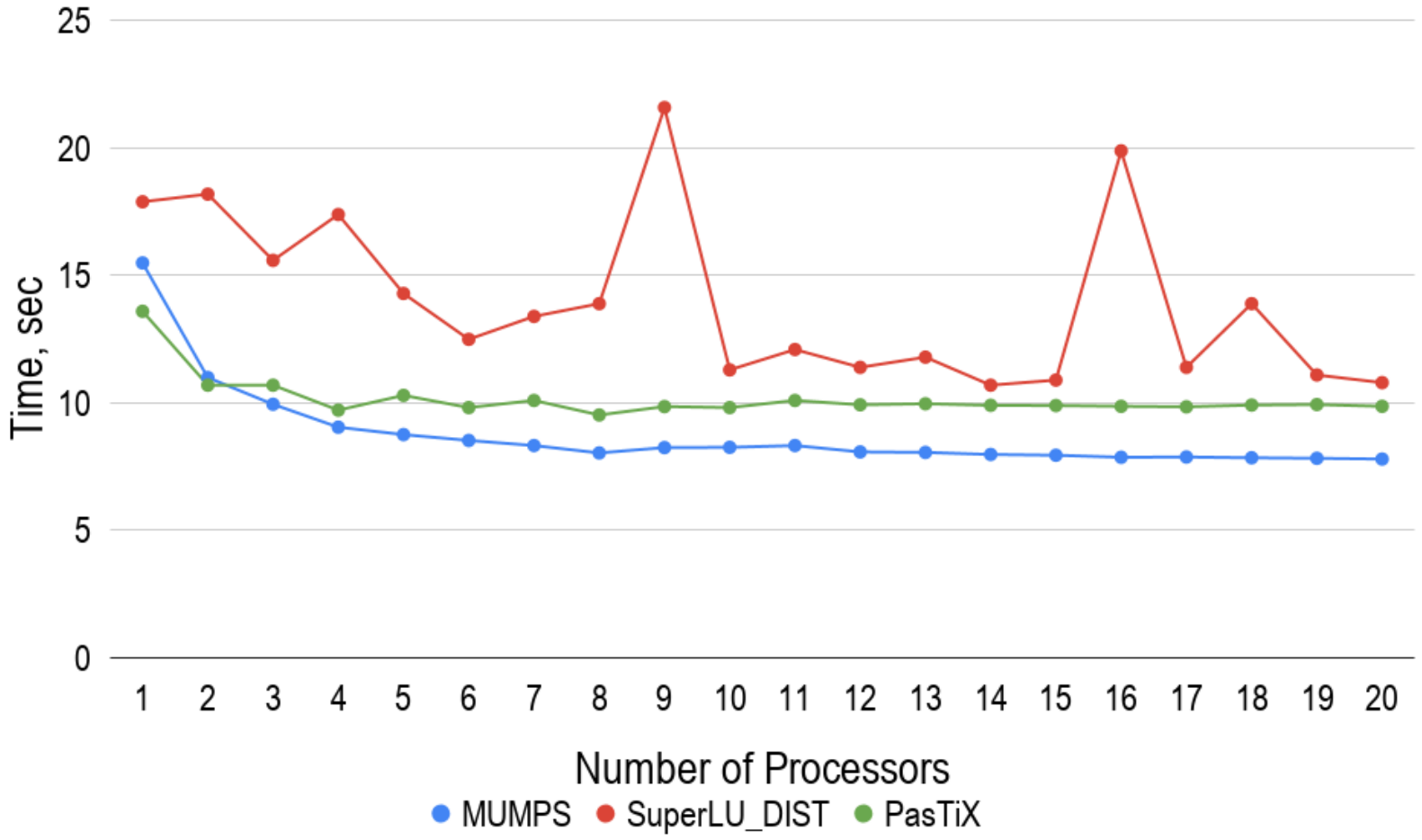



Figure 9: Comparisons of MUMPS, PasTiX and SuperLU_DIST libraries during 5 point-stencil Poisson matrix (1000000 equations) factorizations

# Configuration of MUMPS solver

# ABCD

# Fill Reducing Reorderings

Sequential:

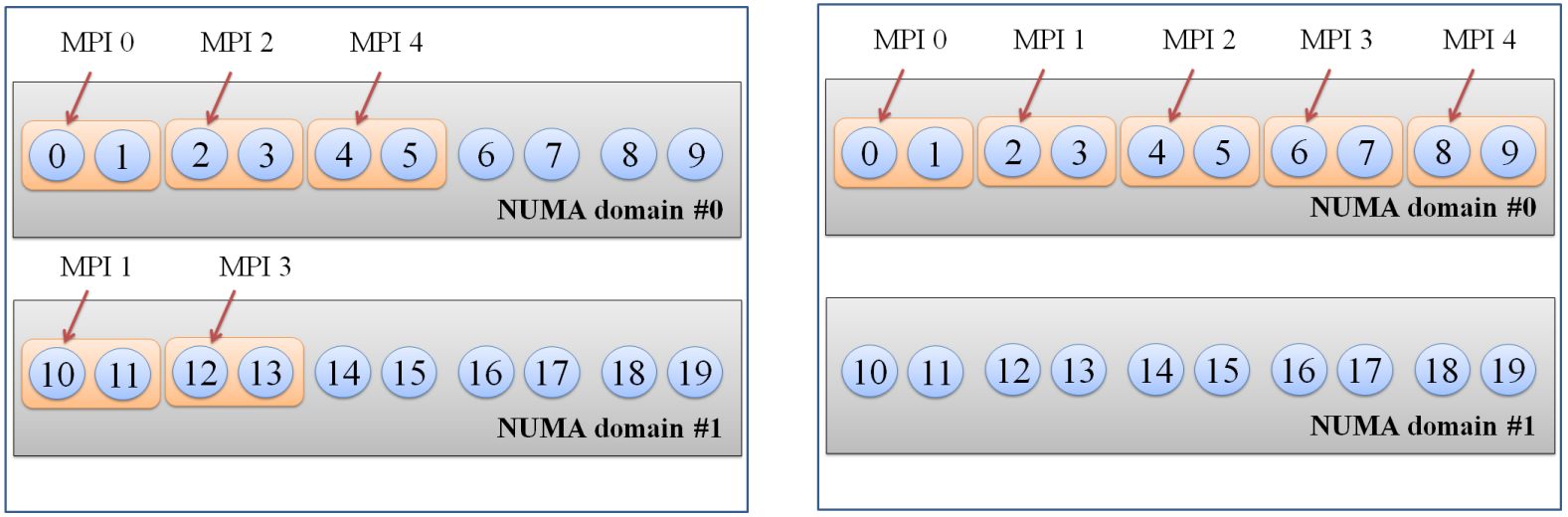
- Approximate Minimum Degree (AMD) [2]
- Approximate Minimum Fill (AMF)
- Approximate Minimum Degree with automatic quasi-dense row detection (QAMD) [1]
- Bottom-up and Top-down Sparse Reordering (PORD) [12]
- Nested Dissection coupled with AMD (Scotch) [11]
- Multilevel Nested Dissection coupled with Multiple Minimum Degree (METIS) [7]

Parallel:

- ParMETIS
- PT-Scotch

# Explicit MPI Process Pinnning

To make process pinning **deterministic**, a python script was developed to automatically generate **rankfiles** based on the number of **MPI processes**, **OpenMP threads** per MPI process, the maximum number of processing elements and the number of **NUMA domains**.




(a) Spread (b) Close

Figure 10: Process pinning of 2 OpenMP / 1 MPI on HW1 hardware

# ABCD

# ABCD

# ABCD

# ABCD

# Outline II

*Thanks for your attention!*


PR Amestoy. “Recent progress in parallel multifrontal solvers for unsymmetric sparse matrices”. In: *Proceedings of the 15th World Congress on Scientific Computation, Modelling and Applied Mathematics, IMACS*. Vol. 97. 1997.


Patrick R. Amestoy, Timothy A. Davis, and Iain S. Duff. “An Approximate Minimum Degree Ordering Algorithm”. In: *SIAM Journal on Matrix Analysis and Applications* (1996).


H. Austregesilo et al. *ATHLET Mod 3.1A – Models and Methods*. distributed with ATHLET. Mar. 2016.


Satish Balay et al. *PETSc Web page*. `http://www.mcs.anl.gov/petsc`. 2018. URL: `http://www.mcs.anl.gov/petsc`.


Timothy A. Davis and Yifan Hu. "The University of Florida Sparse Matrix Collection". In: *ACM Trans. Math. Softw.* 38.1 (Dec. 2011), 1:1–1:25. ISSN: 0098-3500. DOI: 10.1145/2049662.2049663. URL: http://doi.acm.org/10.1145/2049662.2049663.


I. S. Duff, Roger G. Grimes, and John G. Lewis. "Sparse Matrix Test Problems". In: *ACM Trans. Math. Softw.* 15.1 (Mar. 1989), pp. 1–14. ISSN: 0098-3500. DOI: 10.1145/62038.62043. URL: http://doi.acm.org/10.1145/62038.62043.


George Karypis and Vipin Kumar. *MeTis: Unstructured Graph Partitioning and Sparse Matrix Ordering System, Version 4.0*. http://www.cs.umn.edu/~metis. University of Minnesota, Minneapolis, MN, 2009.


Jean-Yves L'Excellent. "Multifrontal methods: parallelism, memory usage and numerical aspects". PhD thesis. Ecole normale supérieure de lyon-ENS LYON, 2012.


X. Li. *Direct Solvers for Sparse Matrices*. 2018. URL: http://crd-legacy.lbl.gov/~xiaoye/SuperLU/SparseDirectSurvey.pdf.


Joseph W. H. Liu. "The Multifrontal Method for Sparse Matrix Solution: Theory and Practice". In: *SIAM* (1992).


François Pellegrini. "Scotch and libScotch 5.1 user's guide". In: (2008).


Jürgen Schulze. "Towards a tighter coupling of bottom-up and top-down sparse matrix ordering methods". In: *BIT Numerical Mathematics* 41.4 (2001), pp. 800–841.


Tim Steinhoff. "Singly implicit FiterRK methods for thermal-hydraulic simulations". 2018. URL: http://somewhere.in.grs.com.