Numerical Methods for High Performance Computing

Lorenzo Schiavone

August 14, 2025

Contents

1 Problem Statement

Consider the Transient Convection-Diffusion Equation

$$\begin{aligned} \frac{\partial}{\partial t}u - K\Delta u + \operatorname{div}(\beta u) = 0 \quad & \text{in } \Omega, \\ u(\mathbf{x}) = 1 \quad & \text{for } \mathbf{x} = \Gamma \\ \nabla u \cdot \nu = 0 \quad & \text{for } \mathbf{x} \in \partial\Omega \setminus \Gamma \end{aligned} \tag{D}$$

for $\Omega = [0,1] \times [0,1] \times [0,1]$, $\Gamma = \{0\} \times \{0\} \times [0,.3]$, where $u = u(x,y,z)$ represents the scalar concentration of a solute dissolved in a fluid moving with velocity β constant and homogenous and $K = \texttt{diag}(K_1, K_2, K_3)$ is the diffusion matrix assumed constant as well.

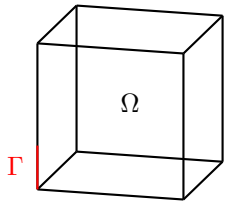

Figure 1: Domain of study.

1.1 Weak Galerkin Formulation

The Weak Formulation of (D) is: find $u \in H^1(\Omega)$ such that boundary conditions hold and

$$\int_\Omega \frac{\partial u}{\partial t} v \, d\Omega + \int_\Omega K\nabla u \cdot \nabla v \, d\Omega + \int_\Omega \operatorname{div}(\beta u) v \, d\Omega = 0 \tag{W}$$

for every $v \in H_0^1(\Omega)$, where we have integrated by part the first term and the boundary term vanishes because $v \in H_0^1(\Omega)$.

Now, considering the discretization $\mathcal{T}_h$ of Ω with nodes $\mathcal{N}_h = \{\mathbf{x}_1, \ldots, \mathbf{x}_{n_{\text{nodes}}}\}$ and tetrahedral finite elements $\mathcal{E}_h$, we can approximate the function space $H^1(\Omega)$ with

$$\mathcal{V}_h = \operatorname{span}\{\phi_1, \ldots, \phi_{n_{\text{nodes}}}\},$$

where the ϕ_i are the piecewise linear Lagrangian basis function. Then, we can write the approximate solution $u_h(t) = \sum_{j=0}^{n_{\text{nodes}}} u_j(t)\phi_j$ and, as $\mathcal{V}_h$ is a finite dimensional vector space, is enough to verify Equation (W) for the basis functions ϕ_i. It yields the Weak Galerkin Formulation: find $u_h(t) \in \mathcal{V}_h$ such that boundary conditions hold and

$$\int_\Omega \sum_{j=1}^{n_{\text{nodes}}} u_j'(t)\phi_j\phi_i \, d\Omega + \int_\Omega \sum_{j=1}^{n_{\text{nodes}}} u_j(t) K\nabla\phi_j \cdot \nabla\phi_i \, d\Omega + \int_\Omega \sum_{j=1}^{n_{\text{nodes}}} u_j(t) \operatorname{div}(\beta\phi_j)\phi_i \, d\Omega = 0 \tag{W_h}$$

or

$$\sum_{j=1}^{n_{\text{nodes}}} u_j'(t) \left(\int_\Omega \phi_j\phi_i \, d\Omega\right) + \sum_{j=1}^{n_{\text{nodes}}} u_j(t) \left(\int_\Omega K\nabla\phi_j \cdot \nabla\phi_i \, d\Omega\right) + \sum_{j=1}^{n_{\text{nodes}}} u_j(t) \left(\int_\Omega \operatorname{div}(\beta\phi_j)\phi_i \, d\Omega\right) = 0 \tag{W_h}$$

that in matrix form reads

$$P\mathbf{u}' + H\mathbf{u} + B\mathbf{u} = 0$$

where $\mathbf{u} = (u_1, \ldots, u_{n_{\text{nodes}}})^T$, and the mass matrix P, the diffusion stiffness matrix H and the non-symmetric convective matrix B have components, respectively,

$$P_{ij} = \int_\Omega \phi_j\phi_i \, d\Omega, \qquad H_{ij} = \int_\Omega K\nabla\phi_j \cdot \nabla\phi_i \, d\Omega \quad \text{and} \quad B_{ij} = \int_\Omega \operatorname{div}(\beta\phi_j)\phi_i \, d\Omega.$$

The derivative $\mathbf{u}'$ is approximated using implicit Euler scheme i.e.,

$$\mathbf{u}' \approx \frac{\mathbf{u} - \mathbf{u}^{\text{old}}}{\Delta t}$$

with a discrete time Δt so that the final linar system is

$$\frac{P}{\Delta t}\mathbf{u} + H\mathbf{u} + B\mathbf{u} = \frac{P}{\Delta t}\mathbf{u}^{\text{old}}.$$

1.2 Finite Elements Matrices

Inside each tetrahedral element $e \in \mathcal{E}$ with vertices $\mathbf{x}_i, \mathbf{x}_j, \mathbf{x}_k, \mathbf{x}_m$ only the corresponding basis function $\phi_i, \phi_j, \phi_k, \phi_m$ are non zero. Their local expression is

$$\phi_l^{(e)}(x, y, z) = \frac{a_l + b_l x + c_l y + d_l z}{6V^{(e)}} \quad \text{for } l \in \{i, j, k, m\},$$

where $V^{(e)}$ is the surface measure of the element,

$$V^{(e)} = \frac{1}{6} \det \left(\begin{bmatrix} 1 & x_i & y_i & z_i \\ 1 & x_j & y_j & z_j \\ 1 & x_k & y_k & z_k \\ 1 & x_m & y_m & z_m \end{bmatrix} \right),$$

and the coefficients $\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}$ are computed so that $\phi_l(\mathbf{x_r}) = \delta_{lr}$, i.e.,

$$a_i = \det \begin{bmatrix} x_j & y_j & z_j \\ x_k & y_k & z_k \\ x_m & y_m & z_m \end{bmatrix} \quad b_i = -\det \begin{bmatrix} 1 & y_j & z_j \\ 1 & y_k & z_k \\ 1 & y_m & z_m \end{bmatrix} \quad a_j = -\det \begin{bmatrix} x_i & y_i & z_i \\ x_k & y_k & z_k \\ x_m & y_m & z_m \end{bmatrix} \quad b_j = \det \begin{bmatrix} 1 & y_i & z_i \\ 1 & y_k & z_k \\ 1 & y_m & z_m \end{bmatrix}$$

$$c_i = \det \begin{bmatrix} 1 & x_j & z_j \\ 1 & x_k & z_k \\ 1 & x_m & z_m \end{bmatrix} \quad d_i = -\det \begin{bmatrix} 1 & x_j & y_j \\ 1 & x_k & y_k \\ 1 & x_m & y_m \end{bmatrix} \quad c_j = -\det \begin{bmatrix} 1 & x_i & z_i \\ 1 & x_k & z_k \\ 1 & x_m & z_m \end{bmatrix} \quad d_j = \det \begin{bmatrix} 1 & x_i & y_i \\ 1 & x_k & y_k \\ 1 & x_m & y_m \end{bmatrix}$$

$$a_k = \det \begin{bmatrix} x_i & y_i & z_i \\ x_j & y_j & z_j \\ x_m & y_m & z_m \end{bmatrix} \quad b_k = -\det \begin{bmatrix} 1 & y_i & z_i \\ 1 & y_j & z_j \\ 1 & y_m & z_m \end{bmatrix} \quad a_m = -\det \begin{bmatrix} x_i & y_i & z_i \\ x_j & y_j & z_j \\ x_k & y_k & z_k \end{bmatrix} \quad b_m = \det \begin{bmatrix} 1 & y_i & z_i \\ 1 & y_j & z_j \\ 1 & y_k & z_k \end{bmatrix}$$

$$c_k = \det \begin{bmatrix} 1 & x_i & z_i \\ 1 & x_j & z_j \\ 1 & x_m & z_m \end{bmatrix} \quad d_k = -\det \begin{bmatrix} 1 & x_i & y_i \\ 1 & x_j & y_j \\ 1 & x_m & y_m \end{bmatrix} \quad c_m = -\det \begin{bmatrix} 1 & x_i & z_i \\ 1 & x_j & z_j \\ 1 & x_k & z_k \end{bmatrix} \quad d_m = \det \begin{bmatrix} 1 & x_i & y_i \\ 1 & x_j & y_j \\ 1 & x_k & y_k \end{bmatrix}$$

Then,

$$\nabla \phi_i^{(e)} = \frac{1}{6V^{(e)}} \begin{bmatrix} b_i \\ c_i \\ d_i \end{bmatrix}$$

is constant and the local diffusion stiffness matrix is

$$\begin{aligned} H_{ij}^{(e)} &= \int_e K \frac{1}{6V^{(e)}} (b_i, c_i, d_i)^T \cdot \frac{1}{6V^{(e)}} (b_j, c_j, d_j)^T \, d\Omega \\ &= \frac{|V^{(e)}|}{36{V^{(e)}}^2} (K_1 b_i b_j + K_2 c_i c_j + K_3 d_i d_j) = \frac{1}{36|V^{(e)}|} (K_1 b_i b_j + K_2 c_i c_j + K_3 d_i d_j). \end{aligned}$$

Compactly, if we collect $\mathbf{b} = \begin{bmatrix} b_i & b_j & b_k & b_m \end{bmatrix}$ and, likewise $\mathbf{c} = \begin{bmatrix} c_i & c_j & c_k & c_m \end{bmatrix}$ and $\mathbf{d} = \begin{bmatrix} d_i & d_j & d_k & d_m \end{bmatrix}$, we can write

$$H^{(e)} = \frac{1}{36|V^{(e)}|} (K_1 \mathbf{b}^T \mathbf{b} + K_2 \mathbf{c}^T \mathbf{c} + K_3 \mathbf{d}^T \mathbf{d}).$$

Convective Matrix Moreover, since

$$\operatorname{div}(\beta\phi_j) = \beta \cdot \nabla\phi_j = \frac{1}{6V^{(e)}} \left(\beta \cdot \begin{bmatrix} \mathbf{b} \\ \mathbf{c} \\ \mathbf{d} \end{bmatrix} \right)_j =: \frac{1}{6V^{(e)}} \sigma_j,$$

where σ is the row vector $\beta \cdot \begin{bmatrix} \mathbf{b} \\ \mathbf{c} \\ \mathbf{d} \end{bmatrix}$ for brevity, and

$$\int_e \phi_i^{(e)}\, d\Omega = \frac{1}{4}|V^{(e)}| \quad \text{as} \quad \phi_i^{(e)} + \phi_j^{(e)} + \phi_k^{(e)} + \phi_m^{(e)} = 1,$$

we have that the local convective matrix is

$$B_{ij}^{(e)} = \int_e \operatorname{div}(\beta\phi_j^{(e)})\phi_i^{(e)}\, d\Omega = \frac{1}{6V^{(e)}}\sigma_j \int_e \phi_i^{(e)}\, d\Omega = \frac{1}{6V^{(e)}}\sigma_j \frac{1}{4} V^{(e)} = \frac{\operatorname{sgn} V^{(e)}}{24}\sigma_j.$$

In matrix form,

$$B^{(e)} = \frac{\operatorname{sgn} V^{(e)}}{24} \begin{bmatrix} \sigma \\ \sigma \\ \sigma \\ \sigma \end{bmatrix},$$

where the rows are equal as the coefficients are independent of the row index.

Mass Matrix The mass matrix coefficients are

$$P_{ij}^{(e)} = \int_\Omega \phi_i^{(e)}\phi_j^{(e)}\, d\Omega = \begin{cases} |V^{(e)}|/10 & \text{if } i = j \\ |V^{(e)}|/20 & \text{if } i \neq j \end{cases}$$

that in matrix form is

$$P^{(e)} = \frac{|V^{(e)}|}{20} \begin{bmatrix} 2 & 1 & 1 & 1 \\ 1 & 2 & 1 & 1 \\ 1 & 1 & 2 & 1 \\ 1 & 1 & 1 & 2 \end{bmatrix}.$$

2 Matrix CSR format

Matrices are stored in Compressed Sparse Row (CSR) format for memory efficiency. Each matrix $N \times N$ is described by:

- a vector of double `coef` with the non zero coefficients,
- a vector of int `ja` with the same length of `coef` where each component is the column index of the corresponding non zero coefficient,
- a vector of int `iat` of dimension $N + 1$ where `iat`(i) has the index where the coefficients of the row i begin in `coef`.

Using this structure, the matrix vector product $Av = b$ is computed as in the following snippet.

```
for(int i = 0; i < N; i++){
    double sum = 0.0;
    for(int k = iat[i]; k < iat[i+1]; k++)
        sum += coef[k] * v[ja[k]];
    b[i] = sum;
}
```

3 Parallel Assembly

Given the tetrahedral discretization of the unit cube, e.g., "Cubo_4820.coor" for the node coordinates and "Cubo_4820.tetra" for the mesh connectivities, the topology of the sparse matrices P, H, and B, i.e., the vectors `iat` and `ja`, is computed using a routine in `topol.cpp`, a file already provided. For parallelization, we used OpenMP directives in C, experimenting with two methods.

Methods In the first method, we open a parallel region within the main loop over the elements. When adding local contributions to the global matrices, we avoid data races using the *omp atomic* directive, which ensures safe access to specific memory locations.

In the second method, we divide the nodes among threads, and each thread is responsible for assembling only the rows corresponding to the nodes assigned to it. To accomplish this, we open a parallel region before the main loop, and each thread iterates through all elements. At the beginning of each loop iteration, we check whether any node of the current element lies within the index range handled by the thread. If none do, the thread skips the element; otherwise, it proceeds with the assembly. This method is preferable because the *omp atomic* clause may become a bottleneck on high thread counts.

RCM Ordering The RCM Ordering minimize the maximum distance from the diagonal of the connectivity matrix as we can see in Figure 2. This ordering is cache efficient and well suited for the second method, so that most of the elements have all the nodes dealt by the same thread and very few are shared between threads avoid repeating the computation of local matrices different times.

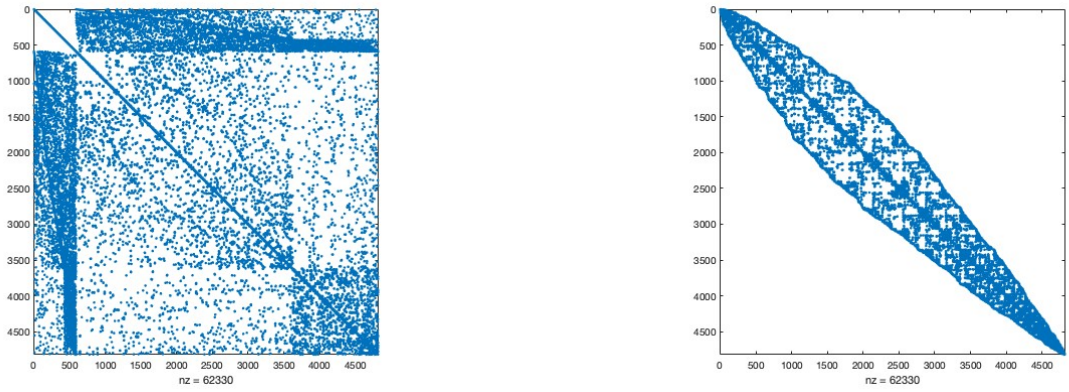

Figure 2: Comparison between the Sparsity Pattern of the Native Order, in the left, and RCM Order, in the right, for Cubo_4820.

4 GMRES

The Generalized Minimal Residual method (GMRES) is an iterative Krylov subspace method used to solve nonsymmetric and possibly indefinite systems of linear equations of the form

$$Ax = b,$$

where $A \in \mathbb{R}^{n \times n}$ is a general (not necessarily symmetric) matrix, and $b \in \mathbb{R}^n$ is the right-hand side vector. GMRES constructs an approximate solution x_k in the Krylov subspace

$$\mathcal{K}_k(A, r_0) = \text{span}\{r_0, Ar_0, A^2 r_0, \ldots, A^{k-1} r_0\},$$

where $r_0 = b - Ax_0$ is the initial residual, such that the residual norm $\|b - Ax_k\|_2$ is minimized at each iteration k.

The method builds an orthonormal basis of $\mathcal{K}_k(A, r_0)$ using the Arnoldi process, resulting in a Hessenberg matrix H_k. The approximate solution is then computed by solving a least-squares problem involving H_k. To solve the least squares problem, the QR factorization of the Hessenberg matrix H_j is computed. Moreover, we have that $\|b - Ax_j\|_2 = \min \|\beta e_1 - H_j y\|_2 \ = |(Q^T e_1)_{j+1}|$ and there is no need to solve explicitely the least square problem, that can be done once for all at the end.

Numerical Implementation of QR At first, we implement the full QR with the Modified Gram Schmidt method. Later, we exploit the incremental nature of the problem to update the QR factorization through Givens rotations. It allows to be more efficient both in memory and in computing resources.
At the step j the QR factorization of the $(j+1) \times j$ upper Hessenberg matrix $\widetilde{H}_j$ is

$$\widetilde{Q}_j^T \widetilde{H}_j = \widetilde{R}_j,$$

where $\widetilde{Q}_j^T \in \mathbb{R}^{(j+1)\times(j+1)}$ is orthogonal and $\widetilde{R}_j \in \mathbb{R}^{(j+1)\times j}$ is upper triangular. Because $\widetilde{R}_j$ has one more row than columns, it can be written as

$$\widetilde{R}_j = \begin{bmatrix} R_j \\ 0 \end{bmatrix},$$

where $R_j \in \mathbb{R}^{j\times j}$ is upper triangular. At the next step, to update the QR decomposition, as the Hessenberg matrix is augmented only by one row and one column

$$\widetilde{H}_{j+1} = \begin{bmatrix} \widetilde{H}_j & h_{j+1} \\ 0 & h_{j+2,j+1} \end{bmatrix},$$

where $h_{j+1} \in \mathbb{R}^{j+1}$ is the new column, we first extend Q_j^T by embedding it into a $(j+2) \times (j+2)$ identity matrix

$$\begin{bmatrix} Q_j^T & 0 \\ 0 & 1 \end{bmatrix} \widetilde{H}_{j+1} = \begin{bmatrix} R_j & r_{j+1} \\ 0 & \rho \\ 0 & \sigma \end{bmatrix}.$$

Since this matrix is almost upper triangular except for the entry σ, to eliminate σ, we apply a Givens rotation

$$G_j = \begin{bmatrix} I_j & 0 & 0 \\ 0 & c_j & s_j \\ 0 & -s_j & c_j \end{bmatrix},$$

where the cosine and sine coefficients are defined as

$$c_j = \frac{\rho}{\sqrt{\rho^2 + \sigma^2}}, \qquad s_j = \frac{\sigma}{\sqrt{\rho^2 + \sigma^2}}.$$

The new orthogonal matrix becomes

$$Q_{j+1}^T = G_j \begin{bmatrix} Q_j^T & 0 \\ 0 & 1 \end{bmatrix},$$

and the updated Hessenberg factorization satisfies

$$Q_{j+1}^T \widetilde{H}_{j+1} = \begin{bmatrix} R_j & r_{j+1} \\ 0 & r_{j+1,j+1} \\ 0 & 0 \end{bmatrix}, \quad \text{with} \quad r_{j+1,j+1} = \sqrt{\rho^2 + \sigma^2}.$$

In addition, the new residual vector $\widetilde{Q}_{j+1}^T e_1$ is easily updated as well to be

$$s_{\text{new}} = G_j s_{\text{prev}},$$

from the previous residual vector s_{prev}.

4.1 Parallelization and Restarts

We employ parallel regions for the most computationally expensive operations in the GMRES algorithm, i.e. the matrix vector products and the computation of scalar products involving large vectors. As the number of iterations increases, the cost of orthonormalizing the new vector Av_n also grows, due to the expanding Krylov subspace. A common strategy to mitigate this cost is to restart the algorithm after a fixed number of iterations. Specifically, once the Krylov basis reaches a predetermined maximum size, the algorithm is restarted using the current approximate solution as the new initial guess. This restart mechanism limits the size of the Krylov subspace and, consequently, the cost of orthonormalization and QR factorization. As a result, optimizing the QR decomposition becomes less critical, since the matrices involved remain relatively small.

4.2 Numerical Solution

We set $K_1 = 0.4$, $K_2 = 0.1$, $K_3 = 0.1$ and $\beta = \begin{bmatrix} 1 & 1 & 2 \end{bmatrix}$, so that we should see more diffusion in the x direction and convection in z direction. For the GMRES we use the Jacobi preconditioner as it is well suited for parallel implementation. Figure 3 displays the numerical solution for the topology *Cubo_246389*.

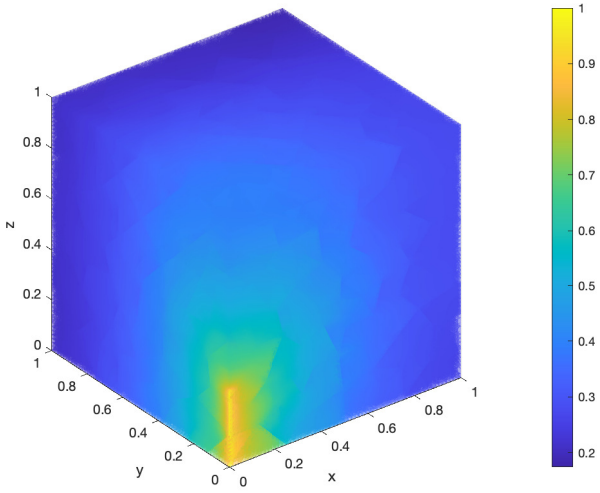

Figure 3: Numerical Solution for *Cubo_246389*.

5 Parallelization improvements

The experiment is performed on a MacBook Air 2017 with a 1.8 GHz Intel Core i5 dual-core (max 4 threads). As it is a domestic and not a computing specific device the speedups and parallelization improvements are modest and limited. We record the wall-clock time using the `omp_get_wtime` function in OpenMP. The Speedup is computed with $S_p = T_1/T_p$ and the Efficiency as $E_p = S_p/p$.

5.1 Assembly

For the mesh Cube_1772481 we record the total wall-clock time for the assembly using the two different methods and orders in Table 1 and speedup and efficiency in Table 2. We can notice that the second method reduces the wall-clock time by completely avoiding the atomic clause. This improvement is evident for both the Native and RCM orderings, the latter yielding the best performance overall.

	Native Order		RCM Order	
method n° threads	1	2	1	2
1	8.219900	5.204082	8.249664	5.618524
2	4.630122	3.205535	4.689079	3.162474
3	4.060645	2.878401	4.205213	2.737441
4	3.634686	2.751820	3.705338	2.398958

Table 1: Wall-clock time (sec) for the assembly of Cube_1772481 per number of threads using the different methods.

	Native Order		RCM Order	
method n° threads	1	2	1	2
1	1	1	1	1
2	1.77	1.62	1.76	1.78
3	2.03	1.81	1.96	2.05
4	2.26	1.89	2.23	2.34

	Native Order		RCM Order	
method n° threads	1	2	1	2
1	1	1	1	1
2	0.885	0.81	0.88	0.89
3	0.677	0.603	0.65	0.683
4	0.565	0.472	0.5575	0.585

Table 2: Speedup (left) and Efficiency (right) for the assembly of Cube_1772481 per number of threads.

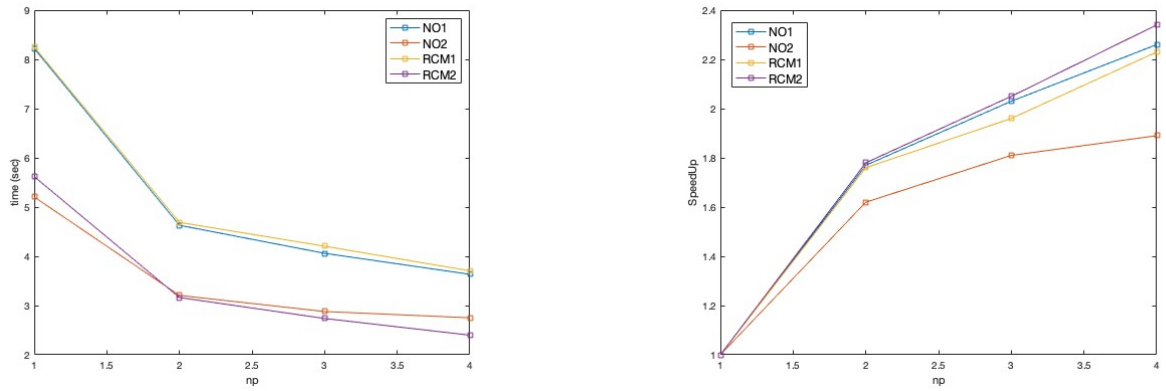

Figure 4: Wall-clock time and speedup for the parallel assembly of Cube_1772481.

5.2 GMRES

We record in Table 3 the wall-clock time for the GMRES algorithm for the topology Cubo_246389, with full QR or with Givens rotations, with different number of threads without `restart` and with `restart` = 20 and in Table 4 the speedup and the efficiency. With `restart` = 20 the runtime is reduced as expected and the gap between fullQR and updateQR is slimmed. On the other hand, without restart 692 iterations are needed to converge and the difference between methods is evident. Due to the intrinsical sequentiality of the algorithm, e.g., scalar products in the orthogonalization have to be performed necessarely one after the other, the speedups are sensibly worse compared to the assembly, which is well fit to be parallelized.

	no `restart`		`restart`=20	
method n° threads	fullQR	Givens	fullQR	Givens
1	180.900720	141.801716	27.755223	28.141819
2	148.415983	105.693213	20.456823	19.665255
3	151.490037	103.072335	19.955573	19.751338
4	153.800005	97.531304	18.681080	18.420930

Table 3: Wall-clock time (sec) for the GMRES on Cube_246389 per number of threads using the different methods.

	no `restart`		`restart`=20	
method n° threads	fullQR	Givens	fullQR	Givens
1	1	1	1	1
2	1.22	1.34	1.35	1.43
3	1.19	1.37	1.39	1.42
4	1.18	1.45	1.48	1.52

	no `restart`		`restart`=20	
method n° threads	fullQR	Givens	fullQR	Givens
1	1	1	1	1
2	0.61	0.67	0.675	0.715
3	0.396	0.456	0.463	0.473
4	0.285	0.3625	0.37	0.38

Table 4: Speedup (left) and Efficiency (right) for the GMRES on Cube_246389 per number of threads.

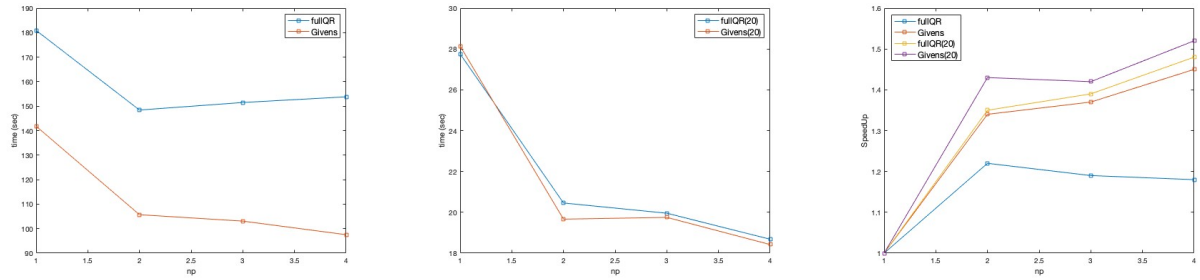

Figure 5: From left to right wall-clock time (respectively without restart and with `restart` = 20) and speedup for the GMRES algorithm with Cube_1772481.

6 Conclusive Remarks

We successfully parallelized, using OpenMP, the computation of a numerical solution of a transient convection-diffusion equation with the Finite Element Method. Even on a standard personal computer with a maximum of four available threads, we achieved a reduction in wall-clock time for both the assembly phase and the GMRES solver. The speedup was more pronounced in the assembly phase, as GMRES contains inherently sequential components that limit its parallel performance.