A COMPARATIVE STUDY OF LOCAL AND GLOBAL PRECONDITIONERS FOR FINITE ELEMENT ANALYSIS

Laisa Karoline Muller

laisakmuller@gmail.com

Leonardo Muniz de Lima

lmuniz@ifes.edu.br

Lucia Catabriga

luciac@inf.ufes.br

Universidade Federal do Espírito Santo

Avenida Fernando Ferrari, 514, Goiabeiras, 29075-910, Vitória, Espírito Santo, Brasil

Abstract. *Finite element method (FEM) invariably leads to solving large and sparse linear systems of equations. Excellent options include nonstationary iterative Krylov methods such as Generalized Minimal Residual (GMRES). Krylov methods demand matrix-vector products as one of the main operation, which efficiency depends on matrix storage. In addition, non-symmetric linear systems from convective dominant problems have slow convergence requiring suitable preconditioners. An efficient storage format is important in order to obtain low memory usage and high computational performance. The resulting matrices from finite element analysis can be stored locally using element by element (EBE) data strategy or globally using Compressed Sparse Row (CSR) format. Global storage allows us to use effective preconditioners like incomplete LU factorization (ILU) combined with matrix reordering schemes while local storage schemes provide simple and efficient implementations of diagonal block based preconditioners. In this work we present a relevant trade-off analysis between local and global preconditioners for finite element implementations performing two examples considering advective-diffusive and inviscid compressible flows.*

Keywords: *Finite Element Method, GMRES Method, Global and Local Preconditioner*

1 INTRODUCTION

The finite element method (FEM) was introduced in the late 1950s as a method to solve partial differential equations (PDEs). The main idea is to divide the domain of the solution into subdomains in which, using variational concepts, the solution is calculated over each subdomain (Becker et al., 1981). Generally, this process results in a system of equations to be solved for each nonlinear and/or time step iteration. Excellent options to solve these equations include non stationary iterative Krylov methods such as Generalized Minimal Residual (GMRES) (Saad & Schultz, 1986). Krylov methods demand matrix-vector products as one of the main operations, which efficiency depends on the type of data structure used to store it.

Applying some kind of preconditioning may increase the performance of Krylov methods. Preconditioning is a transformation of the original system into another which should be easier to be solved. The preconditioner efficiency also depends on the type of data structure used. Although the preconditioner may have desired qualities, the result is not identifiable a priori to the application. In this way, there are numerous attempts to produce efficient preconditioners (Benzi, 2002; Wathen, 2015).

Saad (1994) presents different forms of storing sparse matrices to reduce memory requirements. Global storage allows the use of effective preconditioners such as incomplete LU factorization (ILU), while local storage schemes provides simple and efficient implementations of diagonal block based preconditioners. Our focus will be on the Compressed Sparse Row (CSR) structure for global matrices and on the element-by-element (EBE) structure for local matrices (Ribeiro & Coutinho, 2005).

The computational time and resources needed for the successful application of GMRES may be greatly diminished (becoming even trivial) when optimal preconditioners is used. As the ideal preconditioner cannot be found before its application, the analysis of preconditioning solutions for each type of problem is needed. Thus, our purpose is to present a comparison of preconditioners applied to the EBE and CSR structures on the time dependent advective-diffusive and inviscid compressible flows equations. The analysis of these results allow us to find a closer to ideal preconditioner considering the computational performance efficiency for each problem studied in this work.

The remaining of this paper is organized as follows. In section 2, we describe the governing equations for time dependent advective-diffusive and inviscid compressible flows problems and the stabilized finite element formulation including the time advancing algorithm. In section 3, we give a brief note about preconditioning, discussing the EBE and CSR data structures with an emphasis on its matrix-vector products and also on its algorithms. We describe the preconditioners for each type of data structure used. Section 4 present numerical experiments considering preconditioners performance applied to the EBE and CSR structures. Section 5 ends the article with emphasis on the main conclusions.

2 GOVERNING EQUATIONS AND FINITE ELEMENT FORMULATION

Let us consider two class of problems characterized by the transport and Euler equations. Transport equations describe quantities to be transported, for example, temperature and con-

centration. Euler equations represent a system of conservative laws governed by inviscid compressible flows.

2.1 Transport Equation

Given the functions $g(\mathbf{x}, t)$ and $u_0(\mathbf{x}) = u(\mathbf{x}, 0)$ as the boundary and initial conditions, respectively, the transport equation may be expressed by

$$\begin{aligned} \frac{\partial u}{\partial t} - \nabla \cdot (\boldsymbol{\kappa} \nabla u) + \boldsymbol{\beta} \cdot \nabla u + \sigma u &= f, \ \text{ in } \Omega \times (0, t_f), \\ u &= g, \ \text{ on } \Gamma \times (0, t_f), \\ u(\mathbf{x}, 0) &= u_0(\mathbf{x}), \ \text{ in } \Omega. \end{aligned} \tag{1}$$

The variables are described as follows: u is the quantity being transported; $\Omega \subset \mathbb{R}^2$ is the domain of the problem and $\Gamma = \partial\Omega$ is its boundary; $t_f > 0$ is the final time; $\boldsymbol{\kappa}$ represents the diffusivity tensor; $\boldsymbol{\beta}$ is the velocity field, given by $\boldsymbol{\beta} = (\beta_x, \beta_y)$; σ is the reaction coefficient; and f represents the source term.

To obtain the finite element discretization, consider $\mathcal{T}_h$ a set of triangular partitions of the domain Ω into n_{el} elements such that

$$\mathcal{T}_h = \{\Omega_e \,|\, \Omega = \cup_{e=1}^{n_{el}} \Omega_e,\ \Omega_i \cap \Omega_j = \emptyset,\ i \neq j,\ i, j = 1, \ldots, n_{el}\}. \tag{2}$$

Also, consider the two finite spaces given by

$$\begin{aligned} \mathcal{V}_h &= \{v_h \in H_0^1 \,|\, v|_{\Omega_e} \in \mathbb{P}_1(\Omega_e),\ \forall \Omega_e \in \mathcal{T}_h \text{ and } v|_{\Gamma_D} = 0\}, \\ \mathcal{S}_h &= \{u_h = u_h(\cdot, t) \in H^1 \,|\, t \in [0, t_f],\ u_h(\mathbf{x}, t) = g, \forall \mathbf{x} \in \Gamma\}, \end{aligned}$$

where $\mathbb{P}_1(\Omega_e)$ are the set of first orders polynomial functions defined in Ω_e and Γ_D is the Dirichlet boundary. The space $\mathcal{S}_h$ is called trial solutions and $\mathcal{V}_h$ is called weighting functions. Thereby, the finite element discretization of Eq. (1) considering a stabilized formulation reads: Given g and f, find $u \in \mathcal{S}_h$ such that, for all $v_h \in \mathcal{V}_h$,

$$\int_\Omega \left(v_h \frac{\partial u_h}{\partial t} + \boldsymbol{\kappa} \nabla u_h \nabla v_h + (\boldsymbol{\beta} \cdot \nabla u_h)\, v_h + \sigma u_h v_h \right) d\Omega + \sum_{e=1}^{n_{el}} \int_{\Omega_e} R(u_h)\, \tau\, \boldsymbol{\beta} \cdot \nabla v_h \, d\Omega_e$$

$$+ \sum_{e=1}^{n_{el}} \int_{\Omega_e} \nabla v_h \, \boldsymbol{\delta} \, \nabla u_h \, d\Omega_e = \int_\Omega f v_h \, d\Omega + \int_\Gamma g v_h \, ds.$$

where $R(u_h)$ is the residual of Eq. (1), defined by

$$R(u_h) = \frac{\partial u_h}{\partial t} + \boldsymbol{\kappa} \nabla u_h + (\boldsymbol{\beta} \cdot \nabla u_h) + \sigma u_h - f,$$

τ is the stabilization parameter for SUPG (Brooks & Hughes, 1982) and $\boldsymbol{\delta}$ is the discontinuity-capturing operator YZβ (Tezduyar, 2004). The SUPG parameter is given as follow

$$\tau = \frac{h\xi}{2||\,\boldsymbol{\beta}\,||_\infty}, \ \text{ with } \ \xi = \max\left\{0, 1 - \frac{1}{Pe}\right\} \ \text{ and } \ Pe = \frac{||\,\boldsymbol{\beta}\,||_\infty h}{2\boldsymbol{\kappa}} \ \text{ (Peclet number).}$$

The discontinuity-capturing operator YZβ is defined as $\boldsymbol{\delta} = \delta \boldsymbol{I}$, where

$$\delta = |Y^{-1} R| \left(\sum_{i=1}^{d} \left| Y^{-1} \frac{\partial u_h}{\partial x_i} \right|^2 \right)^{\frac{\beta}{2} - 1} \left(\frac{\tilde{h}}{2} \right)^{\beta},$$

with

$$\tilde{h} = 2\left(\sum_{a=1}^{n_{el}} |\mathbf{j}\cdot\nabla \boldsymbol{N_a}|\right)^{-1} \quad \text{and} \quad \mathbf{j} = \frac{\nabla u_h}{||\,\nabla u_h\,||},$$

and $Y = u_{ref}$ is the reference value of the scalar field u_h (Bazilevs et al., 2007).

2.2 Euler Equations

The system of conservation laws governing inviscid compressible flows are described by the Euler equations. For two-dimensional domains, these equations may be written in terms of conservation variables $\mathbf{U} = (\rho, \rho v_x, \rho v_y, \rho e)$, without source terms, as

$$\frac{\partial \mathbf{U}}{\partial t} + \frac{\partial \mathbf{F}_x}{\partial x} + \frac{\partial \mathbf{F}_y}{\partial y} = \mathbf{0}, \text{ on } \Omega \times [0, t_f], \tag{3}$$

where ρ is the fluid density, $\mathbf{v} = (v_x, v_y)^T$ is the velocity field, e represents the total energy per unit mass, $\mathbf{U}$ is the vector of solution, $\mathbf{F}_x$ and $\mathbf{F}_y$ represents the Euler fluxes. Another way to describe Euler equations is to write Eq. (3) in terms of $\mathbf{A}_x = \frac{\partial \mathbf{F}_x}{\partial \mathbf{U}}$ and $\mathbf{A}_x = \frac{\partial \mathbf{F}_y}{\partial \mathbf{U}}$. Thus,

$$\frac{\partial \mathbf{U}}{\partial t} + \mathbf{A}_x\frac{\partial \mathbf{U}}{\partial x} + \mathbf{A}_y\frac{\partial \mathbf{U}}{\partial x} = \mathbf{0}. \tag{4}$$

For a general boundary condition operator $\mathbf{B}$, $\mathbf{G} = (g_1(t), g_2(t), g_3(t), g_4(t))^T$ and $\mathbf{U}_0$ a given initial solution, the boundary conditions associated with Eq. (4) are given as follows

$$\begin{aligned} \mathbf{BU} &= \mathbf{G}, \text{ on } \Gamma \times [0, t_f], \\ \mathbf{U}(\mathbf{x}, 0) &= \mathbf{U}_0, \text{ on } \Omega. \end{aligned}$$

The finite element discretization defines $\mathcal{T}_h$ a set of triangular partitions of the domain Ω described in Eq. (2) and consider the following two new finite dimensional spaces $\mathcal{S}^h$ and $\mathcal{V}^h$ given by

$$\begin{aligned} \mathcal{S}^h &= \left\{\mathbf{U}^h \,|\, \mathbf{U}^h \in [\mathbf{H}^{1h}(\Omega)]^4,\, \mathbf{U}^h|_{\Omega_e} \in [\mathbb{P}_1(\Omega_e)]^4,\, \mathbf{U}^h \cdot \mathbf{e}_k = g_k(t) \text{ on } \Gamma_{g_k}\right\}, \\ \mathcal{V}^h &= \left\{\mathbf{W}^h \,|\, \mathbf{W}^h \in [\mathbf{H}^{1h}(\Omega)]^4,\, \mathbf{W}^h|_{\Omega_e} \in [\mathbb{P}_1(\Omega_e)]^4,\, \mathbf{W}^h \cdot \mathbf{e}_k = 0 \text{ on } \Gamma_{g_k}\right\}, \end{aligned}$$

where $k = 1, \ldots, 4$ and $\mathbf{H}^{1h}(\Omega) \subset \mathbf{C}^0(\Omega)$ is the finite dimensional function space over Ω. There by, the finite element discretization for Eq. (4) considering stabilized formulation reads: Find $\mathbf{U}^h \in \mathcal{S}^h$ such that, for all $\mathbf{W}^h \in \mathcal{V}^h$

$$\begin{aligned} &\int_\Omega \mathbf{W}^h \cdot \left(\frac{\partial \mathbf{U}^h}{\partial t} + \mathbf{A}_x^h\frac{\partial \mathbf{U}^h}{\partial x} + \mathbf{A}_y^h\frac{\partial \mathbf{U}^h}{\partial y}\right) d\Omega + \\ &\sum_{e=1}^{n_{el}} \int_{\Omega_e} \boldsymbol{\tau}\mathbf{A}_i^h \left(\frac{\partial \mathbf{W}^h}{\partial x_i}\right) \cdot R(\mathbf{U}^h)\, d\Omega + \\ &\sum_{e=1}^{n_{el}} \int_{\Omega_e} \boldsymbol{\delta} \left(\frac{\partial \mathbf{W}^h}{\partial x}\frac{\partial \mathbf{U}^h}{\partial x} + \frac{\partial \mathbf{W}^h}{\partial y}\frac{\partial \mathbf{U}^h}{\partial y}\right) d\Omega = \mathbf{0},\ i = 1, 2, \end{aligned} \tag{5}$$

where $R(\mathbf{U}^h)$ is the residual vector

$$R(\mathbf{U}^h) = \frac{\partial \mathbf{U}^h}{\partial t} + \mathbf{A}_x^h\frac{\partial \mathbf{U}^h}{\partial x} + \mathbf{A}_y^h\frac{\partial \mathbf{U}^h}{\partial y},$$

The term $\boldsymbol{\tau}$ is associated to the SUPG stabilized parameter and is given by $\boldsymbol{\tau} = \tau\mathbf{I}$ such that

$$\tau = \max\left\{0, \tau_t + \zeta(\tau_a - \tau_\delta)\right\},$$

where

$$\zeta = \frac{2\alpha CFL}{1+2\alpha CFL}, \quad \tau_t = \frac{2}{3(1+2\alpha CFL)}\tau_a, \quad \tau_a = \frac{h}{2(c+|\,\mathbf{v}\boldsymbol{\beta}\,|)}, \quad \tau_\delta = \frac{\delta}{(c+|\,\mathbf{v}\boldsymbol{\beta}\,|)^2}. \tag{6}$$

The parameter CFL correspond to the Courant-Friedrichs-Levy number and is given by

$$CFL = \frac{(c+|\,\mathbf{v}\boldsymbol{\beta}\,|)\Delta t}{h}, \quad \text{with } \boldsymbol{\beta} = \frac{\nabla||\,\mathbf{U}\,||_2^2}{||\,\nabla||\,\mathbf{U}\,||_2^2\,||_2}.$$

In Eq. (6), c represents the acoustic speed, h is the mesh parameter, α controls stability and accuracy of the time-marching scheme and δ is the discontinuity-capturing parameter.

Returning to Eq. (5), the parameter $\boldsymbol{\delta}$ is related to the discontinuity-capturing operator YZβ and is given by $\boldsymbol{\delta} = \delta\mathbf{I}$ such that

$$\delta = ||\,\mathbf{Y}^{-1}R(\mathbf{U}^h)\,|| \left(\sum_{i=1}^{d}\left|\left|\mathbf{Y}^{-1}\frac{\partial\mathbf{U}^h}{\partial x_i}\right|\right|^2\right)^{\frac{\beta}{2}-1} \left|\left|\mathbf{Y}^{-1}\mathbf{U}^h\right|\right|^{1-\beta}\left(\frac{\tilde{h}}{2}\right),$$

where $\mathbf{Y}$ is the following matrix

$$\mathbf{Y} = \begin{bmatrix} U_1|_{ref} & 0 & 0 & 0 \\ 0 & U_2|_{ref} & 0 & 0 \\ 0 & 0 & U_3|_{ref} & 0 \\ 0 & 0 & 0 & U_4|_{ref} \end{bmatrix},$$

with $U_i|_{ref}$ the reference values of the components of $\mathbf{U}$ (Tezduyar & Senga, 2006).

2.3 Time Advancing Algorithm

Applying the standard finite element approximation on Eq. (5), for Euler equation, or on Eq. (1), for transport equation, we arrive at a local system of ordinary differential equations:

$$M\dot{U} + KU = F$$

where U is the nodal values of the unknowns ($\mathbf{U}^h$ in Euler equation or u_h in transport equation), whereas $\dot{U}$ is its time derivative and M and K are built from elements contributions. The numerical solution is advanced in time by the implicit predictor-multi-corrector algorithm given in Hughes & Tezduyar (1984). Algorithm 1 shows the implicit predictor-multi-corrector steps, where Δt is the time-step; subscripts $n+1$ and n meaning, respectively, the solution on the time-step $n+1$ and n; $\alpha = 0.5$ is the time advancing parameter; i is the multi-corrector counter. The resulting linear system

$$M^*\Delta U^{n+1,i+1} = R^{n+1,i} \tag{7}$$

is solved several times considering time advancing and multi-corrections. The linear system solutions are performed by preconditioned nonstationary iterative method storing the matrix in element-by-element (EBE) or Compressed Sparse Row (CSR) formats.

Step 1. $i = 0$

Step 2. Predictor phase:

$U^{n+1,0} = U^n + (1-\alpha)\Delta t \dot{U}_h^n,$
$\dot{U}_h^{n+1,0} = 0$

Step 3. Multi-corrector phase:

Residual Force:

$R^{n+1,i} = F^{n+1} - M\dot{U}^{n+1,i} - KU^{n+1,i}$

Solve Eq. (7) using preconditined GMRES with

$M^* = M + \alpha\Delta t K$

Corrector:

$U^{n+1,i+1} = U^{n,i} + \alpha\Delta t \Delta\dot{U}^{n+1,i+1},$
$\dot{U}^{n+1,i+1} = \dot{U}^{n+1,i} + \Delta\dot{U}^{n+1,i+1}$

Algorithm 1: Predictor Multi-corrector Algorithm

3 A BRIEF NOTE ABOUT PRECONDITIONING

As reported by Wathen (2015), to solve a resulting linear system using GMRES may be a good option only if it converges in few iterations. This might be the case if a good preconditioner is applied. The concept of preconditioning is quite simple. Given a linear system $Ax = b$, the main idea behind a good preconditioner is to obtain an appropriate matrix M such as the product $M^{-1}A$ be better conditioned than A. Theoretically, the best choice to M is the own A, but obviously to find A^{-1} is harder than to solve the initial system $Ax = b$. We are facing a trade-off problem as long as we need to improve the condition number of $M^{-1}A$ matrix so that the linear system $M^{-1}Ax = M^{-1}b$ is easier to be solved than the $Ax = b$.

There are three major ways to apply preconditioning: left, right and split. Let M and N preconditioner matrices and y an auxiliary vector for a linear system $Ax = b$. Thus, left, right and split preconditioning are defined as

$$\begin{aligned}
&\text{(Left)} && M^{-1}Ax &&= M^{-1}b, \\
&\text{(Right)} && AM^{-1}y &&= b,\ x = M^{-1}y, \\
&\text{(Split)} && M^{-1}AN^{-1}y &&= M^{-1}b,\ x = N^{-1}y.
\end{aligned}$$

As stated in Saad (2003), a Krylov method (e.g. GMRES) solves a linear system $Ax = b$ by repeatedly performing matrix-vector products involving A. Preconditioning does not facilitate matrix vector product. But when preconditioning is efficient, the number of iterations decreases dramatically. In addition, for finite element context, the kind of preconditioner depends on how to store the sparse matrix. In our case, the local element-by-element (EBE) or the global Compressed Sparse Row (CSR).

3.1 Data Structures for Preconditioning

In this article we consider two class of preconditioners: local and global. This difference arises from data structure schemes for matrices derived from finite element discretization. When matrices are calculated at element level, we use the element-by-element (EBE) scheme to store the local matrices (Hughes, 1987). Here we store element data in an element matrix A_e. Each matrix A_e has dimension $nnoel \cdot ndof \times nnoel \cdot ndof$, where $nnoel$ is the number of nodes per element and $ndof$ is the number of degrees of freedom per node. Matrices A_e have dimensions 3×3 ($ndof = 1$) for transport equation and 12×12 ($ndof = 4$) the Euler equations case, once we are considering linear triangular elements ($nnoel = 3$).

For global data structure, finite element data are stored using Compressed Sparse Row (CSR) scheme (Gustavson, 1972). It means that given a sparse nonsymmetric matrix A with order n and nnz nonzero coefficients, three vectors are created: AA stores all nnz coefficients of A; JA stores the column indexes of the nonzero coefficients of A; IA, with size $n+1$, stores the locations in the vector AA that starts and ends each row of A.

The inner loop of GMRES method demands three main operations: (i) a matrix-vector product; (ii) dot products; (iii) scalar-vector products. The matrix-vector product is the main operation and the unique that considers the matrix coefficients. Thus, a given matrix-vector product in EBE format is performed according to Algorithm 2 and matrix-vector product in CSR format as described in Algorithm 3.

for *e = 1, nel* **do**
  Recover the global numbering of the local degrees of freedom.
  Gathering operation: $p_e \leftarrow p$.
  Perform product: $v_e = A_e p_e$.
  Scattering and accumulation operations: $v \leftarrow v + v_e$.
end

Algorithm 2: EBE matrix-vector product

for $i = 1, n$ **do**
  $v(i) \leftarrow 0$
  for $k = IA(i), IA(i+1) - 1$ **do**
    $v(i) \leftarrow v(i) + AA(k) * p(JA(k))$
  end
end

Algorithm 3: CSR matrix-vector product

REMARK 1: As far as finite element method is concerned, there is no straightforward way to assemble global matrix A in CSR format. It happens because the sparsity of A is unstructured in general. So we adopted to map each element matrix A_e in global structures AA, JA, and IA. It somewhat increases memory consumption but accelerate global assembly and aids matrix reordering operations.

REMARK 2: Reordering and scaling are techniques which improve preconditioning significantly (Benzi, 2002). In our implementations, reordering presented direct effect just over global preconditioners. Scaling demonstrated reasonable improvement of the convergence exclusively when is applied in local preconditioning.

REMARK 3: Algorithm 4 illustrates details related to step 3 of the Algorithm 1. Consider-

ations about sequence of operations like building finite element matrices, reordering, scaling, setup of the preconditioner, and preconditioning are adressed.

Preprocessing:

1. *Read nodes and elements*
2. *Choose EBE or CSR format*
3. *Apply reordering*

Inside step 3 of Algorithm 1:

4. *Building of Matrix $M^* = M + \alpha\Delta t K$ and vector R*
5. *Scaling of M^* and R*
6. *Setup of the preconditioner*

Inside solver GMRES:

$\vdots$

7. *Matrix vector product*
8. *Preconditioning*

$\vdots$

Algorithm 4: Details about step 3 of Algorithm

3.2 Local Preconditioners

A set of element level preconditioners is considered: diagonal (**Diag**), block diagonal (**BlockDiag**), Gauss Seidel (**SGSe**), block Gauss Seidel (**BlockSGSe**), LU factorization (**LUe**) and, block LU factorization (**BlockLUe**). The prefix **Block** means that the respective preconditioner acts in situations where the number of degrees of freedom per node is greater than one (Euler equations). The **e** suffix determines preconditioning directly over local matrices A_e.

Our local preconditioners are based on the diagonal (or block diagonal for Euler equations) of local matrices A_e. The scaling is used as a pre-preconditioner that the objective is to normalize the coefficients of A by its diagonal coefficients (or block diagonal).

This process, in transport equation, define a new system to be solved in conforming to split scaling as

$$\tilde{A}\tilde{x} = \tilde{b}, \text{ where } \tilde{A} = D^{-1/2}AD^{-1/2},\ \tilde{x} = D^{1/2}x \text{ , and } \tilde{b} = D^{-1/2}b \tag{8}$$

with D being the diagonal matrix of A. At the element level, the split scaling is defined as

$$D^{-1/2}AD^{-1/2} = \mathop{\mathbf{A}}_{e=1}^{nel} D_e^{-1/2}A_eD_e^{-1/2} \tag{9}$$

where A_e is a elementary matrix with dimension 3×3 and $D_e^{-1/2}$ is an abstraction of global diagonal represented in each element e. The final result of the element-by-element scaling can be defined as

$$D_e^{-1/2}A_eD_e^{-1/2} = \begin{bmatrix} \frac{a_{11}^e}{a_{ii}} & \frac{a_{12}^e}{\sqrt{a_{ii}a_{jj}}} & \frac{a_{13}^e}{\sqrt{a_{ii}a_{kk}}} \\ \frac{a_{21}^e}{\sqrt{a_{jj}a_{ii}}} & \frac{a_{22}^e}{a_{jj}} & \frac{a_{23}^e}{\sqrt{a_{ii}a_{kk}}} \\ \frac{a_{31}^e}{\sqrt{a_{kk}a_{ii}}} & \frac{a_{32}^e}{\sqrt{a_{kk}a_{jj}}} & \frac{a_{33}^e}{a_{kk}} \end{bmatrix}$$

with a^e_{11}, a^e_{12}, a^e_{13}, a^e_{21}, a^e_{22}, a^e_{23}, a^e_{31}, a^e_{32}, and a^e_{33} coefficients of the element matrix A_e. The terms a_{ii}, a_{jj}, and a_{kk} are coefficients associated with global unknowns i, j, and k.

For the Euler equations, the new system is defined considering the block scaling, as

$$\tilde{A}x = \tilde{b}, \text{ where } \tilde{A} = W^{-1}A, \text{ and } \tilde{b} = W^{-1}b, \tag{10}$$

with W being the block diagonal of A which depends of the four degrees of freedom per node. At the element level, this block scaling is defined as

$$W^{-1}A = \mathop{\mathbf{A}}_{e=1}^{nel} W_e^{-1}A_e \tag{11}$$

where A_e is the local matrix with dimension 12×12 and W_e^{-1}, in turn, is an abstraction of global block diagonal represented in each element e. Similarly, result of the block scaling element-by-element can be defined as

$$W_e^{-1}A_e = \begin{bmatrix} A_{ii}^{-1}A_{11}^e & A_{ii}^{-1}A_{12}^e & A_{ii}^{-1}A_{13}^e \\ A_{jj}^{-1}A_{21}^e & A_{jj}^{-1}A_{22}^e & A_{jj}^{-1}A_{23}^e \\ A_{kk}^{-1}A_{31}^e & A_{kk}^{-1}A_{32}^e & A_{kk}^{-1}A_{33}^e \end{bmatrix} \tag{12}$$

with A^e_{11}, A^e_{12}, A^e_{13}, A^e_{21}, A^e_{22}, A^e_{31}, A^e_{32}, and A^e_{33} blocks of dimension 4×4 for each element matrix A_e. A_{ii}, A_{jj}, and A_{kk} also have dimension 4×4 but represent the global block matrices according to the number of degrees of freedom of the global nodes i, j, and k, respectively. We calculate the inverse of each global block matrix A_{ii}, with $1 \leq ii \leq nnodes$, where $nnodes$ defines the number of nodes in the mesh.

REMARK 4: When the number of degrees of freedom is greater than 1, block scaling does not occurs according to number of unknowns but in relation to the number of nodes. Thus, whenever there is a Dirichlet boundary condition, prior to scaling, the respective row and column of the element matrix A_e must be modified conform to identity matrix of dimension $ndof \cdot nnoel \times ndof \cdot nnoel$ (12×12 for Euler equations). That means if a row i of A_e is associated with Dirichlet, zeros are set in row and column i of A_e, an exception to coefficient a^e_{ii} that receive 1.

Diag and BlockDiag Preconditioners

The numerical results of the **Diag** and **BlockDiag** preconditioners are equivalent to the scaling processes, as described by Eqs. (8) and (10). Although preconditioners needs to be applied every iteration when a matrix vector product is solicited, the runtime of the **Diag** or **BlockDiag** is smaller when compared to split and Block Scalings.

The **Diag** and **BlockDiag** preconditioners operate conforming to statements 6 and 8 of the Algorithm 4. They do not execute statement 5. In the setup of the preconditioner **Diag** the global diagonal inverse is calculated. The setup of the **BlockDiag** depends on obtaining inverses of diagonal blocks per node (see Eq.(12)). The effective preconditioning (statement 8) is a simple multiplication of the diagonal (or block diagonal) over the vector resulting from matrix-vector product.

SGSe and BlockSGSe Preconditioners

The **SGSe** and **BlockSGSe** preconditioners are obtained by the trivial Gauss-Seidel decomposition at the element level, that is

$$A_e - diag(A_e) + I_e = \begin{bmatrix} I & A_{12} & A_{13} \\ A_{21} & I & A_{23} \\ A_{31} & A_{32} & I \end{bmatrix}$$

and

$$L_e = \begin{bmatrix} I & O & O \\ A_{21} & I & O \\ A_{31} & A_{32} & I \end{bmatrix}, \; U_e = \begin{bmatrix} I & A_{12} & A_{13} \\ O & I & A_{23} \\ O & O & I \end{bmatrix}$$

where A_{ij}, with $1 \leq i, j \leq 3$, are matrices blocks whose dimension depends on the number of degrees of freedom per node ($ndof = 1$ for transport equation or $ndof = 4$ for Euler equations) and I represents identity matrix with same dimension.

The preconditioner matrix is defined by

$$M = \mathop{\mathbf{A}}_{e=1}^{n_{el}} L_e U_e.$$

The first stage is the scaling based in statements 5 and 6 of Algorithm 4 as defined in Eqs. (9) and (11). The **SGSe** and **BlockSGSe** preconditioners are performed after each matrix-vector product in the GMRES iterations by

$$v = M^{-1}Ap, \;\; \text{where} \;\; z = Ap \;\; \text{(with Algorithm 2), then}$$
$$v = M^{-1}z \Longrightarrow Mv = z \Longrightarrow LUv = z, \;\; \text{with} \;\; Ly = z \;\; \text{and} \;\; Uv = y, \;\; \text{where}$$
$$L = \mathop{\mathbf{A}}_{e=1}^{n_{el}} L_e \;\; \text{and} \;\; U = \mathop{\mathbf{A}}_{e=1}^{n_{el}} U_e.$$

LUe and BlockLUe Preconditioners

The preconditioners **LUe** and **BlockLUe** are a type of element level LU decomposition where each element matrix is factored. Thus, factors L_e and U_e are obtained from $A_e - diag(A_e) + I_e$, considering the traditional LU factorization.

Once again, the scaling is performed in the begining of the process (statement 5 and 6 of Algorithm 4). The **LUe** and **BlockLUe** are performed after each matrix-vector product using the same idea of the **SGSe** class of preconditioners considering the local LU factorization.

3.3 Global Preconditioners

For CSR scheme, the preconditioner matrix may be obtained by incomplete LU decomposition (ILU) (Meijerink & van der Vorst, 1977). This type of preconditioner is a modification of

Gaussian elimination in which fill-ins are allowed at a certain level p, named ILU(p). Thus, a preconditioner matrix is defined as $M = \tilde{L}\tilde{U}$, where $\tilde{L}$ and $\tilde{U}$ are the incomplete (approximate) LU factors considering p fill-in levels and are less sparse than A.

As already stated in Remark 2, reordering is an excellent option to improve the preconditioning. Although some cautions should be taken. That because runtime to assemble CSR structures and to reorder them could be a bottleneck. So we used an auxiliary data structure to map the assembly and reordering of CSR structures at the same time.

As we can note in statement number 3 of Algorithm 4, reordering is applied once, independently of time advance and multi-corrections. Thanks to auxiliary structures, AA of CSR (structure that store the finite element coefficients) can be assembled every time without effort because JA and IA were already reordered initially.

4 NUMERICAL EXPERIMENTS

We compare global and local preconditioners applied to transport and Euler equations considering cpu time (seconds) and number of iterations. ILU(p) global preconditioners are used taking into account the reverse Cuthill-McKee (RCM) reordering scheme (George & Liu, 1981). Preconditioned GMRES method with 30 vectors on Krylov basis, 1000 maximum number of iterations and tolerance of 10^{-6} are also considered. Problem domains are discretized into unstructured triangular meshes. The ILU(p) and RCM codes were based on the library ITSOL (Li et al., 2016) and Octave (Eaton et al., 2016), respectively. All numerical experiments were performed on a machine with processor Intel®Core™i7-4770 CPU @ 3.40GHz, 15GiB RAM and Ubuntu 14.04 LTS as operational system.

4.1 Transport Equation

First, we consider the well known rotating cone problem (Brooks & Hughes, 1982). Following Eq. (1), the diffusivity tensor is given by $\boldsymbol{\kappa} = \kappa\mathbf{I}$, where $\kappa = 10^{-8}$; the rotational flow field is given by $\boldsymbol{\beta} = (-y, x)^T$; $\sigma = 0.0$ and $f = 0.0$. Final time is set to complete a single period rotation as $t_f = 6.28$ and the time step is $\Delta t = 10^{-2}$. Boundary conditions are $u = 0$ on Γ and initial solution is given by

$$u_0(\mathbf{x}) = \begin{cases} 7.5\left(1.0 + \cos\left(\pi\sqrt{(x+2.5)^2 + y^2}\right)\right), \text{ if } \sqrt{(x+2.5)^2 + y^2} \leq 1.0, \\ 0, \text{ otherwise.} \end{cases}$$

Problem domain is a square defined in $\Omega = [-5.0, -5.0] \times [5.0, 5.0]$. Three unstructured triangular mesh sizes were considered. Mesh m_{3421_6640} with 3421 nodes and 6640 elements; mesh m_{37243_73816} with 37243 nodes and 73816 elements; and mesh m_{83214_165426} with 83214 nodes and 165426 elements.

Table 1 shows the total number of iterations (iter) and cpu-time (cputime) in seconds for local and global preconditioners. For the EBE case, $\mathbf{LUe}$ preconditioner has better behavior, in terms of cpu-time and number of iterations. Regarding CSR format, ILU(p) preconditioners with RCM reordering demonstrate better computational performance. The number of iterations decreases as fill-in level increases. In meantime, the cost to construct matrices with higher fill-in affects the final cpu-time. On other hand, as the mesh grows, increasing the fill-in level of ILU preconditioners makes the preconditioning more efficient.

Table 1: Cone computational performance.

			m_{3421_6640}		m_{37243_73816}		m_{83214_165426}	
			iter	cputime	iter	cputime	iter	cputime
EBE	No precond		194,589	19.06	371,675	700.08	10,552,927	52422.46
	Diag		126,012	13.73	103,564	244.77	498,515	2634.34
	SGSe		63,215	13.29	52,844	237.50	377,775	3571.03
	LUe		51,723	**11.79**	43,886	**209.27**	151,626	**1320.18**
CSR	No precond		191,613	18.19	450,956	493.28	10,742,456	37032.86
	ILU(0)	not	49,713	14.97	38,878	219.54	125,905	1197.47
		RCM	33,369	**14.04**	28,540	**170.07**	83,352	697.19
	ILU(1)	not	29,352	18.12	21,924	284.63	66,715	1190.03
		RCM	24,325	15.06	19,484	191.51	49,539	**604.18**
	ILU(2)	not	23,835	22.34	19,805	410.92	47,940	1403.43
		RCM	18,841	17.37	14,810	225.99	35,353	643.06

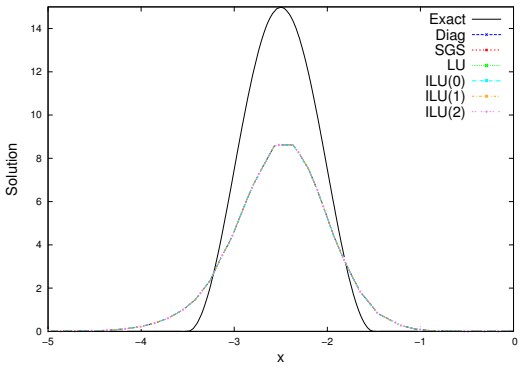
Figure 1 shows a comparison between exact cone solution at $u = (x, 0)$ and the approximated ones with all preconditioners defined in this work. As we can see, preconditioned solutions are practically identical. For mesh m_{3421_6640}, exact cone solution could not be achieved due to fewer number of elements to represent the solution with good accuracy (see Fig. 1a). For meshes m_{37243_73816} and m_{83214_165426}, represented in Figs. 1b and 1c, mesh refinement produced a more accurated solution. Additionally, in Fig. 2, we represent the contours of cone solution at $t = 0.0$, $t = 1.57$, $t = 3.14$, $t = 4.71$ and $t = 6.28$, representing one single period rotation of cone problem.

4.2 Euler Equations

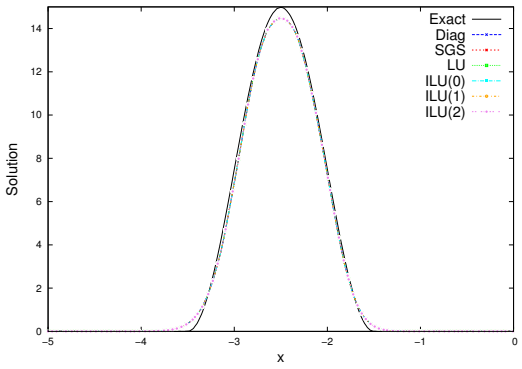
For Euler equations, the problem studied is called wind tunnel with a step (Woodward & Colella, 1984) or, to be short, tunnel problem. Tunnel domain is given by $0 \leq x \leq 3$ and $0 \leq y \leq 1$ with a step at $x = 0.6$ and height of $y = 0.2$. As in transport equation, we consider three different unstructured triangular mesh sizes: mesh m_{1425_2688} with 1425 nodes and 2688 elements; mesh m_{3807_7344} with 3807 nodes and 7344 elements; and mesh m_{15112_29687} with 15112 nodes and 29687 elements. The final time is $t_f = 2.0$ and the time step is $\Delta t = 10^{-3}$.

Table 2 shows the analysis of global and local preconditioners. The number of iterations (iter) decreases whenever the mesh is refined. The **BlockSGSe** and **BlockLUe** preconditioners are exceptions to this rule (for mesh m_{15112_29687} the number of iterations increases). Despite **BlockLUe** presents the smaller number of iterations among the local preconditioners, **BlockDiag** is the fastest (the smaller cpu-time in seconds).

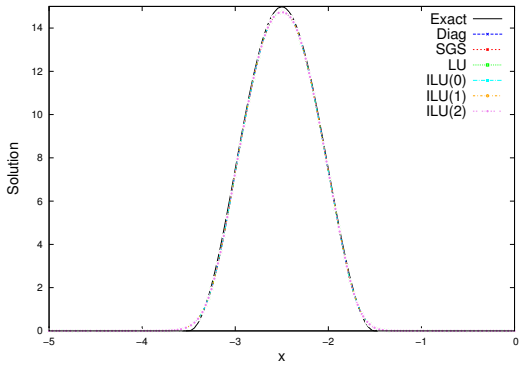
For CSR case, the ILU preconditioners with RCM reordering produces the best global



(a) m_{3421_6640}.

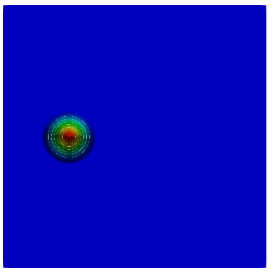


(b) m_{37243_73816}.

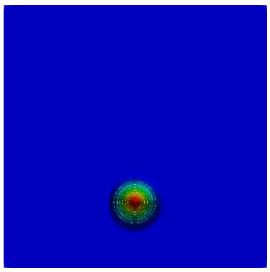


(c) m_{83214_165426}.

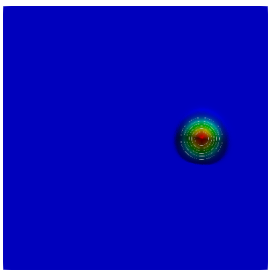
Figure 1: Cone solution at $u = (x, 0)$.



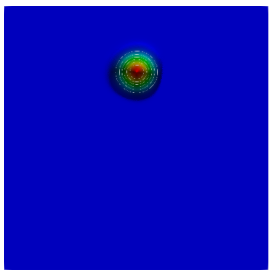
(a) $t = 0.0$.



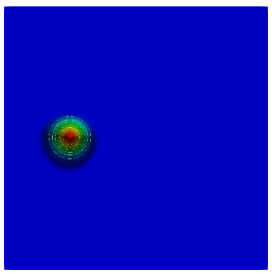
(b) $t = 1.57$.



(c) $t = 3.14$.



(d) $t = 4.71$.



(e) $t = 6.28$.

Figure 2: Cone solution for $t = 0.0$, $t = 1.57$, $t = 3.14$, $t = 4.71$ and $t = 6.28$.

Table 2: Tunnel computational performance.

			m_{1425_2688}		m_{3807_7344}		m_{15112_29687}	
			iter	cputime	iter	cputime	iter	cputime
EBE	No precond		844,050	219.51	834,861	899.29	825,259	5085.89
	BlockDiag		608,467	**180.76**	556,447	**696.25**	531,158	**3528.08**
	BlockSGSe		404,719	217.57	355,826	754.28	711,525	8202.64
	BlockLUe		353,312	319.00	326,568	1060.67	340,201	4974.27
CSR	No precond		843,682	346.90	856,419	1014.44	821,943	4795.41
	ILU(0)	not	193,376	282.81	175,739	821.66	146,821	3762.36
		RCM	127,409	**221.47**	116,917	**644.83**	105,888	**2798.83**
	ILU(1)	not	108,462	406.79	100,358	1203.18	82,749	5529.60
		RCM	82,120	253.00	79,959	764.31	69,781	3203.15
	ILU(2)	not	83,566	573.83	78,989	1775.42	61,828	8094.46
		RCM	62,220	334.13	60,799	1014.13	59,756	4359.38

preconditioning again. For mesh m_{3807_7344}, ILU(0) with RCM is faster than **BlockDiag**. ILU(1) with RCM as well when mesh is m_{15112_29687}. Considering a preconditioner or not, the CSR scheme becomes a better option as we increase the size of the mesh. In addition, as the mesh is refined, the ILU preconditioners combined with RCM become the best options when compared with local preconditioners. Fig. 3 represents the tunnel solutions for ILU(0) with RCM.

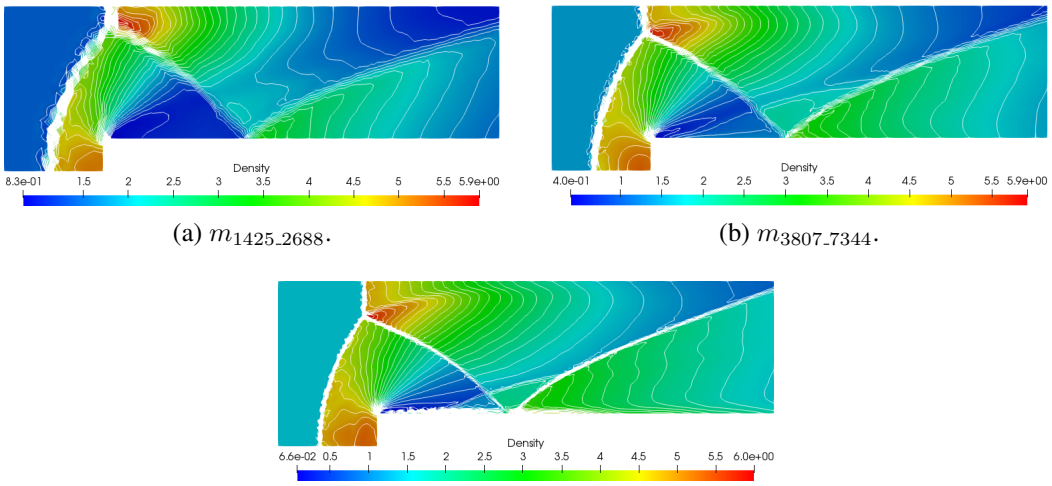
4.3 Preconditioners Trade-off Analysis

Previous experiments demonstrated that preconditioners efficiency depends on several features. Degrees of freedom per node, mesh size, reordering and scaling are examples of items which may influence the choice of a suitable preconditioner. In this subsection, we analyzed the main operations involved in the solution process and particularly in the preconditioning.

There are five operations that deserve our attention (see statements 4, 5, 6, 7 and 8 of Algorithm 4), named: `Build_Matrices`, `Scaling`, `Preconditioner_Setup`, and `Matrix_Vector_Product`. They demand the major part of computation time and can represent the behavior of the preconditioners.

We investigate the solution process and preconditioning of the cone and tunnel problems according to the three sizes of mesh. Each analysis set is divided into local and global preconditioners. Only the reordered versions of ILU preconditioner are considered. We use the Tuning and Analysis Utilities (TAU) (Shende & Malony, 2006), version 2.26, to obtain a profile of each preconditioner.

Figures 4 and 5 illustrate how the preconditioning acts for both problems, considering



(a) m_{1425_2688}. (b) m_{3807_7344}.

(c) m_{15112_29687}.

Figure 3: Tunnel density solution at $t_f = 2.0$ for ILU(0) preconditioner with RCM .

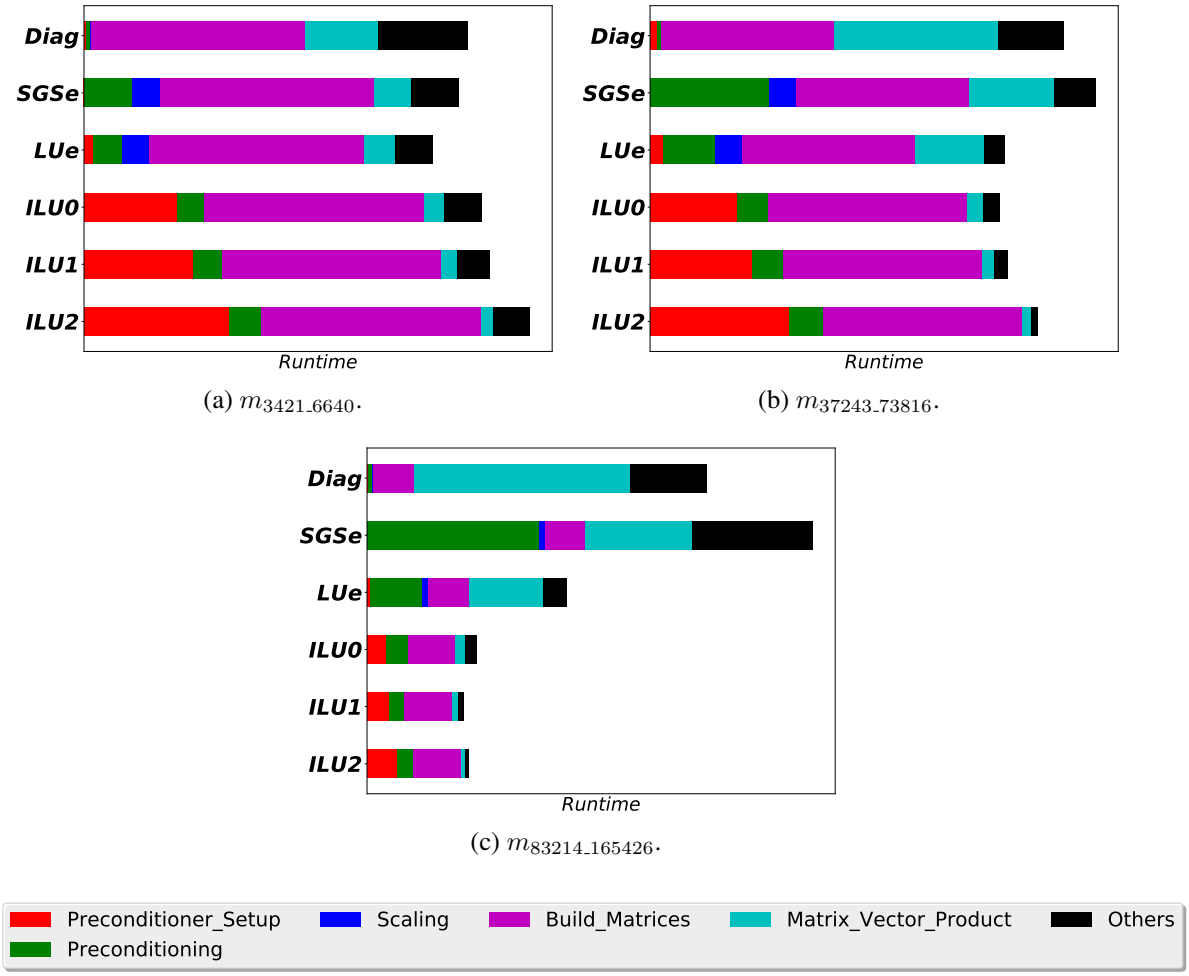
all mesh sizes tested. The absolute runtime are shown and there are not marks in horizontal axis because values are not in the same scale, but it is a good idea to follow the values presented in Tables. 1 and 2. As we can note, local preconditioners **Diag** (or **BlockDiag**), **SGSe** (or **BlockSGSe**), **LUe** (or **BlockLUe**) and global preconditioners ILU(0), ILU(1), ILU(2) were evaluated. `Scaling` was applied solely for **SGSe** (or **BlockSGSe**) and **LUe** (or **BlockLUe**). In addition, the remainder of execution time (excluding the 5 outstanding operations) was stated in the item called `Others`.

When analyzing Figs. 4a and 4b, we realize that the `Build_Matrices` operation as the prominent about execution time. For global preconditioners, the `Preconditioner_Setup` increases as we consider higher fill-in levels. Among local preconditioners, **LUe** was the most efficient. As the mesh grows, global preconditioners incontestably were responsible for the best performances (see Fig. 4c).

Figure 5 reveals local and global preconditioners behavior applied to the tunnel problem. The **BlockDiag** is notoriously the best option when we use local preconditioners. The **BlockSGSe** seemed to improve as the mesh increased, but this trend was not confirmed (see Fig. 5c). The ILU(0) tends to improve its performance when meshes are larger. However, this improvement was not so significant when compared with one degree of freedom problem (please, see Fig. 4c).

5 CONCLUSIONS

In this work we evaluated six preconditioners for transport and Euler equations. For transport equation, we applied the following preconditioners: **Diag**, **SGSe**, **LUe** and ILU(p), $p = 0, 1$ and 2. The ILU(p) preconditioners are presented with and without RCM reordering and the others preconditioners are applied with split scaling. The same is applied to Euler equations, but preconditioners for EBE case are named **BlockDiag**, **BlockSGSe**, and **BlockLUe**. We divided these preconditioners in two classes: local (**Diag**, **SGSe**, **LUe**, **BlockDiag**, **BlockSGSe**, and **BlockLUe**) and global (ILU(p)) preconditioners. For transport and Euler equations, with meshes refined sufficiently, global preconditioners shown to be

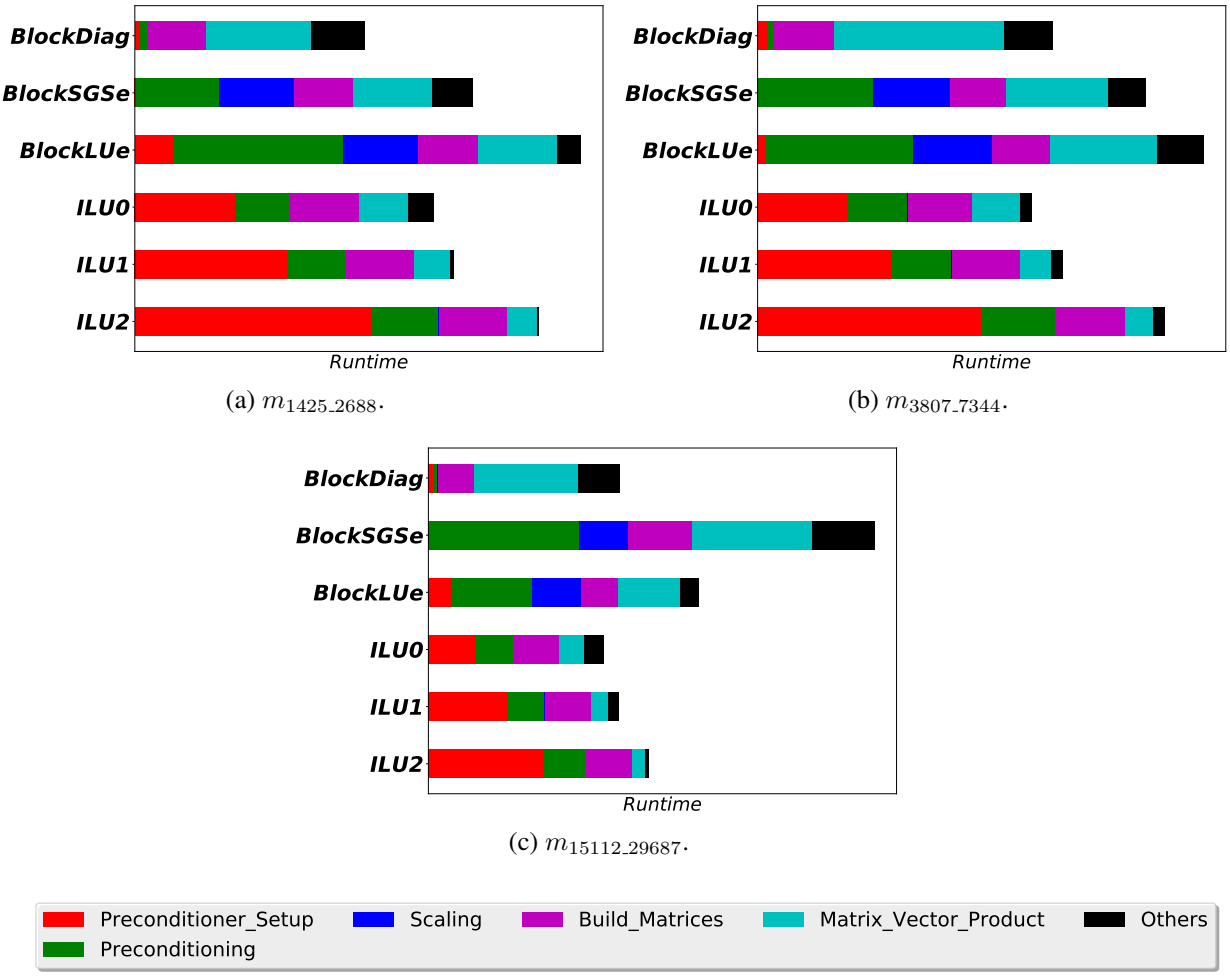


(a) m_{3421_6640}.

(b) m_{37243_73816}.

(c) m_{83214_165426}.

Figure 4: Preconditioners trade-off analysis: cone problem.



(a) m_{1425_2688}.

(b) m_{3807_7344}.

(c) m_{15112_29687}.

Figure 5: Preconditioners trade-off analysis: tunnel problem.

the best choice, specially when ILU(p) is combined with RCM reordering. When only local preconditioners are analyzed, for transport equations, the **LUe** achieves the best performance reducing the computational time about 97.5% when compared to the case without preconditioner. For Euler equation, the **BlockDiag** achieved a better performance for local preconditioners decreasing computational time about 31.0% when compared to the not preconditioned case. Focusing on CSR scheme, the ILU(0) with RCM compared to the case without preconditioner, decrease about 98.0% in computational time for the transport problem and 42.0% for the Euler problem. Further experiments need to be done to confirm the behavior of local and global preconditioners in other general configurations.

ACKNOWLEDGEMENTS

The first and second authors would like to thank CAPES by partial financial support.

REFERENCES

Bazilevs, Y., Calo, V. M., Tezduyar, T. E. & Hughes, T. J. R., 2007. YZβ discontinuity capturing for advection-dominated processes with application to arterial drug delivery, *International Journal for Numerical Methods in Fluids* 54(6-8), 593–608.

Becker, E. B., Carey, G. F. & Oden, J. T., 1981. *Finite Elements: An introduction*, Texas finite element series, Prentice-Hall.

Benzi, M., 2002. Preconditioning techniques for large linear systems: A survey, *J. Comput. Phys.* 182(2), 418–477.

Brooks, A. N. & Hughes, T. J. R., 1982. Streamline upwind/Petrov-Galerkin formulations for convection dominated flows with particular emphasis on the incompressible Navier-Stokes equations, *Computer Methods in Applied Mechanics and Engineering* 32(1-3), 199–259.

Eaton, J. W., Bateman, D., Hauberg, S. & Wehbring, R., 2016. *GNU Octave version 4.2.0 manual: a high-level interactive language for numerical computations*.
URL: *http://www.gnu.org/software/octave/doc/interpreter*

George, A. & Liu, J. W., 1981. *Computer Solution of Large Sparse Positive Definite*, Prentice Hall Professional Technical Reference.

Gustavson, F. G., 1972. *Some Basic Techniques for Solving Sparse Systems of Linear Equations*, Springer US, Boston, MA, pp. 41–52.

Hughes, T. J. R., 1987. *The finite element method: linear static and dynamic finite element analysis*, Prentice-Hall, New Jersey.

Hughes, T. J. R. & Tezduyar, T. E., 1984. Finite element methods for first-order hyperbolic systems with particular emphasis on the compressible Euler equations, *Computer methods in applied mechanics and engineering* 45(1), 217–284.

Li, N., Suchomel, B., Osei-Kuffuor, D., Li, R. & Saad, Y., 2016. 'Itsol: Iterative solvers package'.
URL: *https://www-users.cs.umn.edu/∼saad/software/ITSOL/*

Meijerink, J. A. & van der Vorst, H. A., 1977. An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix, *Mathematics of Computation* 31(137), 148–162.

Ribeiro, F. L. B. & Coutinho, A. L. G. A., 2005. Comparison between element, edge and compressed storage schemes for iterative solutions in finite element analyses, *International Journal for Numerical Methods in Engineering* 63(4), 569–588.

Saad, Y., 1994. 'Sparskit: a basic tool kit for sparse matrix computations - version 2'.

Saad, Y., 2003. *Iterative methods for sparse linear systems*, second edn, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA.

Saad, Y. & Schultz, M. H., 1986. Gmres: A generalized minimal residual algorithm for solving nonsymmetric linear systems, *SIAM J. Sci. Stat. Comput.* 7(3), 856–869.

Shende, S. S. & Malony, A. D., 2006. The tau parallel performance system, *Int. J. High Perform. Comput. Appl.* 20(2), 287–311.

Tezduyar, T. E., 2004. *Finite element methods for fluid dynamics with moving boundaries and interfaces*, John Wiley & Sons, Ltd, chapter 17.

Tezduyar, T. E. & Senga, M., 2006. Stabilization and shock-capturing parameters in SUPG formulation of compressible flows, *Computer Methods in Applied Mechanics and Engineering* 195(13), 1621–1632.

Wathen, A. J., 2015. Preconditioning, *Acta Numerica* 24, 329–376.

Woodward, P. & Colella, P., 1984. The numerical simulation of two-dimensional fluid flow with strong shocks, *Journal of Computational Physics* 54, 115–173.