<http://www.diva-portal.org>

Postprint

This is the accepted version of a paper published in . This paper has been peer-reviewed but does not include the final publisher proof-corrections or journal pagination.

Citation for the original published paper (version of record):

Otero, E., Eliasson, P. [Year unknown!]

Acceleration on Stretched Meshes with Line-Implicit LU-SGS in Parallel Implementation.

Access to the published version may require subscription.

N.B. When citing this work, cite the original published paper.

Permanent link to this version:

<http://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-156412>

Acceleration on Stretched Meshes with Line-Implicit LU-SGS in Parallel Implementation

Evelyn Otero^{a*} and Peter Eliasson^b

^a*Royal Institute of Technology (KTH), Department of Aeronautical and Vehicle Engineering, SE-100 44 Stockholm, Sweden;* ^b*Swedish Defense Research Agency (FOI), Department of Aeronautics and Autonomous Systems, SE-164 90 Stockholm, Sweden*

The implicit Lower-Upper Symmetric Gauss-Seidel solver is combined with the line-implicit technique to improve convergence on the very anisotropic grids necessary for resolving the boundary layers. The computational fluid dynamics code used is Edge, a Navier-Stokes flow solver for unstructured grids based on a dual grid and edge-based formulation. Multigrid acceleration is applied with the intention to accelerate the convergence to steady state. Lower-Upper Symmetric Gauss-Seidel works in parallel and gives better linear scaling with respect to the number of processors, than the explicit scheme. The ordering techniques investigated have shown that node numbering does influence the convergence and that the orderings from Delaunay and advancing front generation were among the best tested. 2D Reynolds-Averaged Navier-Stokes computations have clearly shown the strong efficiency of our novel approach line-implicit Lower-Upper Symmetric Gauss-Seidel which is four times faster than implicit Lower-Upper Symmetric Gauss-Seidel and line-implicit Runge-Kutta. Implicit Lower-Upper Symmetric Gauss-Seidel for Euler and line-implicit Lower-Upper Symmetric Gauss-Seidel for Reynolds-Averaged Navier-Stokes are at least twice faster than explicit and line-implicit Runge-Kutta, respectively, for 2D and 3D cases. For 3D Reynolds-Averaged Navier-Stokes, multigrid did not accelerate the convergence and therefore may not be needed.

Keywords: convergence acceleration; LU-SGS; line-implicit; parallelization; ordering techniques; implicit time-stepping; line-implicit LU-SGS

1. Introduction

Computational fluid dynamics (CFD) has become a significant tool routinely used in the aerodynamic design and optimization of aircraft. CFD tools are used from the conception phase to the production as preliminary tests of a specific aircraft design. In some areas of design, CFD simulations have largely replaced experiments to provide aerodynamic data. However, computing time can become very costly, thus convergence acceleration techniques are needed to speed up CFD solvers.

Edge (Eliasson 2001, 2002; Eliasson and Weinerfelt 2007) is a flow solver for unstructured grids based on a dual grid and edge-based formulation on node-centered finite-volume discretization. Edge uses an explicit multistage Runge-Kutta time integration solver (Eliasson 2001). Agglomeration multigrid (MG) acceleration speeds up the convergence rate based on the Full Approximation Scheme/Storage (FAS) (Brandt 1977) for nonlinear problems. This is combined with the line-implicit method (Venkatakrishnan 1998; Turkel et al. 1999; Mavriplis 1999; Eliasson et al. 2009; Langer 2013) for Reynolds-Averaged Navier-Stokes (RANS) on stretched meshes. The line-implicit or semi-implicit Runge-Kutta time stepping approach integrates implicitly in time along lines in regions where the computational grid is highly stretched. The explicit Runge-Kutta time integration

*Corresponding author. Email: otero@kth.se

is retained everywhere else.

Implicit schemes which mitigate the CFL time-step limit of explicit schemes are becoming more and more widespread in the CFD field, for reducing simulation time. The use of an explicit or implicit scheme is considered when dealing with problems solved by time- (or pseudo-time-)stepping. The time-step will have an influence either on the accuracy of the solution for time-dependent problems or on the convergence to steady-state for non-linear problems. This paper considers computation of steady states and focuses on the speed of convergence to steady-state.

An implicit Lower-Upper Symmetric Gauss-Seidel (LU-SGS) solver has been implemented in the code Edge (Eliasson 2001). The LU-SGS approximate factorization method is attractive because of its good stability properties and limited memory requirement. First developed for structured grids by Jameson and Yoon (1987), then successfully extended to hybrid structured-unstructured grids (Soetrismo et al. 1996; Sharov and Nakahashi 1998), LU-SGS has been improved through different methods. The definition of a proper implicit operator (Chen and Wang 2000; Kim and Kwon 2005; Dwight 2006) and the combination with the GMRES solver as a preconditioning method (Sharov et al. 2000; Nejat and Ollivier-Gooch 2005) are some of them. The method considers a linearization which is inexactly solved by a few steps of a symmetric Gauss-Seidel method through a forward and backward sweep. A block matrix implicit operator with upwind dissipation has been defined (Otero and Eliasson 2011) and controls, by parameters in the artificial dissipation model, contributions to the diagonal blocks of the system matrix to make them “heavier” (Dwight 2006).

The lack of efficiency of LU-SGS for RANS computations on high aspect ratio grids (Wright et al. 1995) and the success of the line-implicit method on highly stretched meshes (Eliasson et al. 2009), has motivated our most novel implementation, the line-implicit LU-SGS solver, combining the LU-SGS solver with the line-implicit method, for the first time, to our knowledge. The idea is to provide LU-SGS with a supplementary acceleration technique only when dealing with stretched meshes. For unstretched grids, LU-SGS works as in its original algorithm. Other techniques are investigated in this paper to improve the efficiency of LU-SGS by Message Passing Interface (MPI) (Barney 2014) parallelization (Wright et al. 1995; Wissink et al. 1997; Sharov et al. 2000; Luo et al. 2003) and node ordering techniques (George and Liu 1981; Borne 2000). First results have been presented (Otero and Eliasson 2013) and are improved in the present paper following a parameter study previously carried out (Otero and Eliasson 2014).

The paper starts with the theoretical presentation of the governing equations, and the spatial and temporal discretization applied. The line-implicit Runge-Kutta and the LU-SGS solvers are then described. Finally the different acceleration techniques applied are presented, supported by numerical results. Comparisons are carried out with explicit Runge-Kutta on Euler meshes and line-implicit Runge-Kutta on RANS meshes and the role of multigrid is analyzed.

2. Problem formulation

2.1. Governing equations

The RANS equations for compressible flow are solved by Edge (Eliasson 2001) in conservation form with an integral formulation for correct capturing of discontinuities such as shock waves. Moreover the system relies on a turbulence model. The conservation-law form of the Navier-Stokes equations concerns the mass, momentum and energy and can be formulated as in Eq. (1).

$$\frac{\partial \mathbf{u}}{\partial t} + \nabla \cdot \mathbf{F} = \mathbf{q} \quad (1)$$

where $\mathbf{u}$ is the vector of the conserved variables $\begin{pmatrix} \rho \\ \rho u \\ \rho v \\ \rho w \\ \rho E \end{pmatrix}$ with E the total energy, ρ the density,

and u, v, w the velocities. The pressure p is needed and computed from the conserved variables. $\mathbf{q}$ is the vector of source terms which comes from the turbulence model.

Moreover $\mathbf{F} = \mathbf{F_I} - \mathbf{F_V}$, where $\mathbf{F_I}$ is the matrix of the inviscid flux and $\mathbf{F_V}$ is the matrix of the viscous flux which resolved into Cartesian components are

$$\mathbf{F_I} = (\mathbf{F_{Ix}} \ \mathbf{F_{Iy}} \ \mathbf{F_{Iz}}) = \begin{pmatrix} \rho u & \rho v & \rho w \\ \rho u^2 + p & \rho v u & \rho w u \\ \rho u v & \rho v^2 + p & \rho w v \\ \rho u w & \rho v w & \rho w^2 + p \\ u(\rho E + p) & v(\rho E + p) & w(\rho E + p) \end{pmatrix}$$

$$\mathbf{F_V} = (\mathbf{F_{Vx}} \ \mathbf{F_{Vy}} \ \mathbf{F_{Vz}}) = \begin{pmatrix} 0 & 0 & 0 \\ \tau_{11} & \tau_{21} & \tau_{31} \\ \tau_{12} & \tau_{22} & \tau_{32} \\ \tau_{13} & \tau_{23} & \tau_{33} \\ (\mathbf{TV})_1 + h_1 & (\mathbf{TV})_2 + h_2 & (\mathbf{TV})_3 + h_3 \end{pmatrix}$$

with $\mathbf{V} = \begin{pmatrix} u \\ v \\ w \end{pmatrix}$ the primitive variables, $\tau_{ij} = \mu \left(\frac{\partial V_i}{\partial x_j} + \frac{\partial V_j}{\partial x_i} \right) - \frac{2}{3} \mu (\nabla \cdot \mathbf{V}) \delta_{ij}$ the stress tensor elements of $\mathbf{T}$, and $\mathbf{h} = (h_1, h_2, h_3)$ the conductive heat flux according to the Fourier's law.

The integral form of the Navier-Stokes equations, Eq. (1), over a volume Ω contained in the surface S reads as Eq. (2)

$$\int_{\Omega} \frac{\partial \mathbf{u}}{\partial t} d\Omega + \int_S (\mathbf{F_I} - \mathbf{F_V}) \mathbf{n} dS = \int_{\Omega} \mathbf{q} d\Omega \quad (2)$$

where $\mathbf{n}$ is the outward normalized normal vector to S .

2.2. Spatial discretization

To the integral form, Eq. (2), is applied a finite-volume discretization, Eq. (3), to the control volume V_i surrounding the node v_i . The control volume is defined by the so called dual grid and surrounds each node of the generated grid, connected by edges. A surface $\mathbf{n}S$ is given for each edge.

$$\frac{d}{dt}(\mathbf{u}_i V_i) + \sum_{j \in N(i)} \hat{\mathbf{F}}_{Iij} \mathbf{n}_{ij} S_{ij} - \sum_{j \in N(i)} \hat{\mathbf{F}}_{Vij} \mathbf{n}_{ij} S_{ij} = \mathbf{q}_i V_i \quad (3)$$

with $N(i)$ the set of neighboring nodes to node i , $\hat{\mathbf{F}}_{Iij}$ and $\hat{\mathbf{F}}_{Vij}$, the discretized inviscid and viscous flux matrices between nodes i and j , $\mathbf{n}_{ij}$ the normal vector to the surface between node i and node j , and S_{ij} the surface area between nodes i and j .

The inviscid flux has been discretized using a Jameson central scheme (Eliasson 2001) and for

the viscous flux, only the normal derivatives are kept leading to a thin-layer discretization

$$(\mathbf{T}_{ij}\mathbf{n}_{ij})_{tl} = (\mu + \mu_t) \left(\frac{\partial \mathbf{V}_{ij}}{\partial \mathbf{n}} + \frac{1}{3} \left(\frac{\partial \mathbf{V}_{ij}}{\partial \mathbf{n}} \cdot \mathbf{n}_{ij} \right) \mathbf{n}_{ij} \right) \quad (4)$$

with μ and μ_t the dynamic and turbulent viscosity coefficients respectively. One may also add the remaining tangential derivatives $(\mathbf{T}_{ij}\mathbf{n}_{ij})_{tan}$ from nodal Green-Gauss integration (Eliasson 2001) to obtain a full viscous operator, $\mathbf{T}_{ij}\mathbf{n}_{ij} = (\mathbf{T}_{ij}\mathbf{n}_{ij})_{tl} + (\mathbf{T}_{ij}\mathbf{n}_{ij})_{tan}$.

Concerning the turbulence model, the Spalart-Allmaras one equation turbulence model (Spalart and Allmaras 1992) is used. From the “standard” model described in (Rumsey 2014), a simplified formulation is implemented and reads in compressible form

$$\frac{\partial \rho \tilde{\nu}}{\partial t} + \underbrace{\mathbf{V} \cdot \nabla \rho \tilde{\nu}}_{convection} = \underbrace{c_{b1} \tilde{S} \rho \tilde{\nu}}_{production} - \underbrace{c_{w1} f_w \rho \left(\frac{\tilde{\nu}}{d} \right)^2}_{destruction} + \underbrace{\frac{\rho}{\sigma} \nabla \cdot ((\nu + \tilde{\nu}) \nabla \tilde{\nu})}_{dissipation} + \underbrace{\frac{c_{b2} \rho}{\sigma} [\nabla \cdot (\tilde{\nu} \nabla \tilde{\nu}) - \tilde{\nu} \Delta \tilde{\nu}]}_{diffusion} \quad (5)$$

The turbulent eddy viscosity is computed from $\mu_t = \rho \tilde{\nu} f_{v1}$ where

$$f_{v1} = \frac{\chi^3}{\chi^3 + c_{v1}^3} \quad , \quad f_{v2} = 1 - \frac{\chi}{1 + \chi f_{v1}} \quad \text{with} \quad \chi = \frac{\tilde{\nu}}{\nu}, \quad \text{and} \quad \nu = \frac{\mu}{\rho} \quad \text{the kinematic viscosity.}$$

Moreover we define the modified vorticity,

$$\tilde{S} = \Omega + \frac{\tilde{\nu}}{\kappa^2 d^2} f_{v2}$$

where $\Omega = \|\nabla \times \mathbf{V}\|$ is the magnitude of the vorticity, d the distance from the field point to the closest wall, and the function f_w reads

$$f_w = g \left[\frac{1 + c_{w3}^6}{g^6 + c_{w3}^6} \right]^{\frac{1}{6}} \quad \text{with} \quad g = r + c_{w2}(r^6 - r) \quad \text{and} \quad r = \min \left[\frac{\tilde{\nu}}{\tilde{S} \kappa^2 d^2}, 10 \right]$$

The constants are $c_{b1} = 0.1355$, $\sigma = 2/3$, $c_{b2} = 0.622$, $\kappa = 0.41$, $c_{w1} = \frac{c_{b1}}{\kappa^2} + \frac{1+c_{b2}}{\sigma}$, $c_{w2} = 0.3$, $c_{w3} = 2$, $c_{v1} = 7.1$, and $c_{t3} = 1.2$, $c_{t4} = 0.5$.

Finally we have also boundary fluxes for nodes on the boundaries and we use weak boundary conditions in all boundaries (Eliasson and Nordström 2013).

2.3. Implicit time-stepping

The spatial discretization of the RANS equations, Eq. (3), is expressed by $\mathbf{r}$, the nonlinear residual vector containing parts pertaining to the convective processes, the viscous forces, the artificial dissipation, and the turbulence model. We solve the nonlinear steady state problem, $\mathbf{r}(\mathbf{u}) = 0$ of Eq. (3), by a time marching method solving time in-accurately the RANS equations. The resulting equations in Eq. (6) are discretized in time by the implicit (or Backward) Euler method to produce the final discretized system, Eq. (7).

$$\mathbf{V} \frac{d\mathbf{u}}{dt} + \mathbf{r}(\mathbf{u}) = 0 \quad (6)$$

$$\frac{1}{\Delta t} \mathbf{V} \cdot (\mathbf{u}^{n+1} - \mathbf{u}^n) + \mathbf{r}(\mathbf{u}^{n+1}) = 0 \quad (7)$$

The linearization of the residual $\mathbf{r}(\mathbf{u}^{n+1})$ as $\mathbf{r}(\mathbf{u}^{n+1}) \approx \mathbf{r}(\mathbf{u}^n) + \frac{\partial \mathbf{r}(\mathbf{u}^n)}{\partial \mathbf{u}} \Delta \mathbf{u}^n$ leads to a linear system of the form $\mathbf{M}\mathbf{x} = \mathbf{b}$, Eq. (8), to be solved approximately by an iterative linear solver.

$$\underbrace{\left(\frac{1}{\Delta t} \mathbf{V} + \frac{\partial \tilde{\mathbf{r}}(\mathbf{u}^n)}{\partial \mathbf{u}} \right)}_{\mathbf{M}} \underbrace{\Delta \mathbf{u}^n}_{\mathbf{x}^n} = \underbrace{-\mathbf{r}(\mathbf{u}^n)}_{\mathbf{b}} \quad (8)$$

with

$$\Delta \mathbf{u}^n = \mathbf{u}^{n+1} - \mathbf{u}^n \quad (9)$$

$\tilde{\mathbf{r}}$ is an approximation of the RHS spatial discretization, $\mathbf{r}$, and $\mathbf{M}$ and $\mathbf{b}$ are defined as the Left Hand Side (LHS) and Right Hand Side (RHS) respectively, $\mathbf{LHS} \Delta \mathbf{u}^n = \mathbf{RHS}$.

This implicit formulation approaches the Newton method as $\frac{1}{\Delta t} \rightarrow 0$. In this sense one can make assumptions about the convergence when approaching a zero value for the residual $\mathbf{r}$ where Newton's method has a quadratic convergence.

The implicit operator $\mathbf{M}$ can be constructed for an optimized convergence rate without affecting the solution limit $\mathbf{u}^n, n \rightarrow \infty$. If $\mathbf{A} = \frac{\partial \mathbf{r}(\mathbf{u})}{\partial \mathbf{u}}$ is the exact Jacobian, time-stepping becomes efficient because the spectrum of the linearized system will be significantly compressed compared to the original dynamic system. However this would require the storage and inversion of a big matrix. Since the steady solution is independent of $\mathbf{M}$, we try to find a compromise between computational cost of producing $\mathbf{M}$ and solving the system Eq. (8), and the loss of accuracy by approximation and inexact solution, (Otero and Eliasson 2011).

Approximate methods for the construction of the Jacobian for different space discretizations are discussed by e.g. Rossow (2006); Swanson et al. (2007); Langer and Li (2011); Langer (2013) etc. We have chosen the LU-SGS iterative method to solve the system. A special effort was previously made to define a proper LHS operator independently of the RHS for the best convergence (Otero and Eliasson 2011). An upwind dissipation type operator shown to be the most optimized method offering parameters of freedom to increase the diagonal dominance of the system matrix and hence the linear convergence rate (Otero and Eliasson 2011). The following work is based on this first investigation and the final objective is to improve the convergence compared to the original solver in Edge for RANS computations, the line-implicit Runge-Kutta method.

2.4. Line-implicit Runge-Kutta

The scheme for the semi-implicit 3-stage Runge-Kutta algorithm (Eliasson et al. 2009) is presented in Eq. 10, which is only solved along the lines implicitly and explicitly everywhere else.

$$\begin{aligned} \mathbf{u}^0 &= \mathbf{u}^n \\ \left(I + \gamma_k \frac{\Delta t}{V} \frac{\partial \tilde{\mathbf{r}}}{\partial \mathbf{u}} \right) (\mathbf{u}^{*,k} - \mathbf{u}^{*,k-1}) &= \mathbf{u}^0 - \mathbf{u}^{*,k-1} - \alpha_k \frac{\Delta t}{V} \mathbf{r}(\mathbf{u}^{*,k-1}); \quad k = 1, 2, 3 \\ \mathbf{u}^{n+1} &= \mathbf{u}^{*,3}. \end{aligned} \quad (10)$$

The explicit Runge-Kutta coefficients are $(\alpha_1, \alpha_2, \alpha_3) = (2/3, 2/3, 1)$ and the implicit coefficients are $(\gamma_1, \gamma_2, \gamma_3) = (1, 1, 1)$. The explicit Runge-Kutta scheme is recovered when these last values are zero.

3. LU-SGS solver

3.1. The method

LU-SGS is an iterative solver, almost matrix-free in the sense that one does not need to store the whole matrix but just access it by matrix-vector products and only a sequence of small $k \times k$ linear systems with k the number of unknowns per node needs to be solved by Gaussian elimination.

The LU-SGS scheme is related to the Symmetric Gauss-Seidel (SGS) relaxation method. First the block system matrix $\mathbf{M}$ is decomposed into a block lower triangular part $\mathbf{L}$, a block diagonal $\mathbf{D}$, and a block upper triangle $\mathbf{U}$ as Eq. (11). A block is the set of unknowns per node.

$$\mathbf{M}\mathbf{x} = (\mathbf{L} + \mathbf{D} + \mathbf{U})\mathbf{x} = \mathbf{b} \quad (11)$$

The SGS iteration is composed by sweeps, backward and forward,

$$\begin{aligned} (\mathbf{D} + \mathbf{L}) \mathbf{x}^* &= \mathbf{b} - \mathbf{U}\mathbf{x}^n \\ (\mathbf{D} + \mathbf{U}) \mathbf{x}^{n+1} &= \mathbf{b} - \mathbf{L}\mathbf{x}^* \end{aligned} \quad (12)$$

If one restricts the number of forward and backward sweeps to one, the system simplifies to Eq. (13) with an initial solution $\mathbf{x}^0 = 0$, giving then the LU-SGS iteration.

$$\begin{aligned} (\mathbf{D} + \mathbf{L}) \mathbf{x}^* &= \mathbf{b}, \\ (\mathbf{D} + \mathbf{U}) \mathbf{x}^1 &= \mathbf{b} - \mathbf{L}\mathbf{x}^* \end{aligned} \quad (13)$$

Since the SGS iterations are done inside each time-step, these are called inner or linear iterations. The number of linear iterations is retained as a tunable parameter and the implementation will focus on Eq. (12). The LU-SGS denomination will be used here independently of the number of sweeps.

The following section concerns the definition of the flux Jacobian approximations and we focus only on the inviscid flux Jacobian. In fact the viscous flux Jacobians are the exact Jacobians of the thin-layer discretization of the RHS, Eq. 4, and the turbulence Jacobians are also exact derivatives of the RHS terms, Eq. 5, for both assuming constant dynamic (and turbulent) viscosity and heat conductivity.

Note that for each time step, all the Jacobians are computed once and stored prior to the solver computations. The objective is to take advantage, also in time, of the convergence rate improvement by increasing the number of sweeps.

3.2. Inviscid flux Jacobian approximation

The LHS Jacobians of the inviscid flux are approximated by differentiating a first order upwind scheme instead of the Jameson scheme used in the RHS. The discretized flux to differentiate, Eq. 14, contains a central part computed as the average of the variables (as in the RHS) and an additional

dissipation $\mathbf{d}_{ij}$ which will define the approximation.

$$\hat{\mathbf{f}}_{Iij} = \mathbf{f}_I \left(\frac{\mathbf{u}_i + \mathbf{u}_j}{2} \right) - \mathbf{d}_{ij} \quad (14)$$

with $\hat{\mathbf{f}}_{Iij} = \hat{\mathbf{F}}_{Iij} \mathbf{n}_{ij} S_{ij}$ and $\mathbf{f}_I = \mathbf{F}_I \mathbf{n} S$. The computation of $\mathbf{d}_{ij}$ is based on a first order upwind scheme to ensure diagonal dominance and is computed as

$$\mathbf{d}_{ij} = \frac{1}{2} |\mathbf{A}''|_{ij} (\mathbf{u}_j - \mathbf{u}_i) \quad \text{with} \quad |\mathbf{A}''|_{ij} = (\mathbf{R} |\mathbf{\Lambda}''| \mathbf{R}^{-1})_{ij} \quad (15)$$

The matrix $\mathbf{A}$ is the arithmetic average which can replace the Roe average matrix, not necessary with a central scheme in the RHS. Note that a control on this matrix is provided by the integration of a parameter which adds a fraction ϵ of the spectral radius λ_{maxij} of the flux Jacobian matrix $\mathbf{A}_{ij}$, to the eigenvalues λ_k in $\mathbf{\Lambda}''$,

$$|\lambda_k''|_{ij} = |\lambda_k|_{ij} + \epsilon \lambda_{maxij} \quad (16)$$

where $\lambda_{maxij} = \max_k |\lambda_k|_{ij}$. Its influence is studied in our paper (Otero and Eliasson 2014).

The Jacobians are computed in Eq. 17 assuming a constant matrix $\mathbf{A}$ and using a flux splitting technique (Nejat and Ollivier-Gooch 2005) which found its origin in Jameson and Turkel (1981), Yoon and Kwak (1992).

$$\begin{aligned} \mathbf{A}_{ij}^+ &= \frac{\partial \hat{\mathbf{f}}_{Iij}}{\partial \mathbf{u}_i} = \frac{\partial \mathbf{f}_{Iij}}{\partial \mathbf{u}_i} + \frac{1}{2} |\mathbf{A}''|_{ij} \\ \mathbf{A}_{ij}^- &= \frac{\partial \hat{\mathbf{f}}_{Iij}}{\partial \mathbf{u}_j} = \frac{\partial \mathbf{f}_{Iij}}{\partial \mathbf{u}_j} - \frac{1}{2} |\mathbf{A}''|_{ij} \end{aligned} \quad (17)$$

3.3. Modified system and algorithm

The final linear system to be solved using the definitions described previously can be written

$$\underbrace{\left[\text{diag} \left(\frac{V_i}{\Delta t_i} \right) + \sum_{j \in N(i)} \left(\mathbf{A}_{ij}^+ + \frac{\partial \mathbf{f}_{Vij}}{\partial \mathbf{u}_i} \right) \right]}_{\mathbf{D}_i} \Delta \mathbf{u}_i = -\mathbf{r}_i(\mathbf{u}) - \sum_{j \in N(i)} \underbrace{\left(\mathbf{A}_{ij}^- - \frac{\partial \mathbf{f}_{Vij}}{\partial \mathbf{u}_j} \right)}_{\mathbf{M}_{ij}} \Delta \mathbf{u}_j \quad (18)$$

This system clearly differs from the original LU-SGS by the diagonal matrix $\mathbf{D}$ of block type and in general, by the few approximations applied to the Jacobian computations. In fact approximation of the Jacobian-vector products by residual vector differences is commonly used. A clear comparison is provided by Chen and Wang (2000).

The system is solved with LU-SGS as

For $iter = 1 : N_{swp}$

Forward sweep

For $i = 1, 2, \dots, N$

$$\mathbf{D}_i \Delta \mathbf{u}_i^* = -\mathbf{r}_i(\mathbf{u}) - \sum_{j \in L(i)} \mathbf{M}_{ij} \Delta \mathbf{u}_j^* - \sum_{j \in U(i)} \mathbf{M}_{ij} \Delta \mathbf{u}_j^n$$

Backward sweep**For** $i = N, N - 1, \dots, 1$

$$\mathbf{D}_i \Delta \mathbf{u}_i^{n+1} = -\mathbf{r}_i(\mathbf{u}) - \sum_{j \in U(i)} \mathbf{M}_{ij} \Delta \mathbf{u}_j^{n+1} - \sum_{j \in L(i)} \mathbf{M}_{ij} \Delta \mathbf{u}_j^*$$

 $n = n + 1;$ **End** *iter*;**4. Line-implicit LU-SGS****4.1. Definition and line construction**

Another version of LU-SGS consists in combining the original LU-SGS algorithm with a line-implicit method. By this combination we expect to improve the convergence in the prismatic stretched grids needed for resolution of boundary layers and wakes in RANS computations. The line-implicit method (Eliasson et al. 2009) consists in integrating the flow equations implicitly in time along structured lines in stretched regions of the grid, usually in the boundary layer. Figure 1 shows the implicit lines (emphasized) in the structured stretched area surrounded by an unstructured grid. The lines are the edges transversal to the prism layers and are not broken by the partitioning.

Figure 1. Implicit lines (emphasized) on stretched grid.

How the lines are computed is an important factor for the efficiency of the method. The algorithm used is based on a graph algorithm and a complete description can be found in Eliasson et al. (2009). The idea is to group together neighboring stretched edges through a set of steps which can be summarized as follows. The algorithm starts by constructing a list of edges and aspect ratios (AR) for each node. All the nodes are initially free which means that they do not belong to any line. The nodes are then sorted with respect to the aspect ratios and the node having the largest $AR > AR_{max}$ is selected as starting node. The algorithm follows and repeats with the connected edge and node with smallest $AR > AR_{min}$. The construction of a line is interrupted when there are no more free nodes and edges which satisfy $AR > AR_{min}$. This procedure is executed for each new line and the line construction algorithm finishes when there are no free nodes with $AR > AR_{max}$. Important conditions are that a line should have minimum 10 nodes and the parameters are $AR_{max} = 100$ and $AR_{min} = 2.5$. These are just choices made, but not optimized values. Moreover the lines do not overlap and the edges and nodes belong only to one line.

Several aspects need to be considered to use our approach with lines. First our method works if the grid is stretched in one direction only. This means that we have only lines in 2D and 3D, i.e. never planes. Then it works for corners if the grid is stretched in one direction where there will be

a line on one of the walls. The approach requires that the grid is structured in the direction of the stretching which is typically the case for RANS grids with prismatic and/or hexahedral elements in 3D (and quads in 2D). The negative impact from the lines are some added computational effort and minor effects on the parallelization and domain decomposition since we require a line to be contained within a single partition.

The implementation of the combined schemes, LU-SGS with a line-implicit technique, results in splitting the linear system of Eq. (8) into two parts. The first one corresponds to the anisotropic part where the lines are defined and the line-implicit method is applied. The remaining part computes the isotropic part by the “original” LU-SGS. By “original” LU-SGS we mean, the original algorithm with our modified system.

4.2. Algorithm

The matrix $\mathbf{M}$ is split as $\mathbf{M} = \mathbf{L} + \mathbf{B} + \mathbf{U}$ and is ordered such that $\mathbf{B}$ contains the block diagonal and (possibly) blocksub- and blocksuper diagonals of the “implicit lines”. Outside the lines, a single block $\mathbf{B}_i = \mathbf{D}_i$, Eq. 18. Nodes are always strongly connected to nearest neighbors but also, depending on the spatial discretization scheme, to next nearest neighbors. This happens e.g. for the Jameson fourth order artificial dissipation terms (Jameson 1983) or the Roe scheme (Roe 1981) extended to a High-Resolution scheme. Figure 2 defines the block matrices $\mathbf{L}$, $\mathbf{B}$ and $\mathbf{U}$ whose dimensions vary with the size of the line to be solved.

Figure 2. Linear system matrix definition.

The line-implicit LU-SGS algorithm works such that when $\mathbf{B}$ is only the block-diagonal it solves by nodes with the “original” LU-SGS and when $\mathbf{B}$ includes the blocksub- and blocksuper-diagonals, LU-SGS is combined with the line-implicit method solving by lines.

The algorithm is line based in the sense that it works through lines computations. However inside each line it computes by nodes as the “original” LU-SGS, taking advantage of the sparsity of the matrices. A line l can have a size $n_l > 1$ (line-implicit) or be a single node $n_l = 1$ (LU-SGS). Initially a specific node ordering is created (Otero 2012) which consists in storing all the lines followed by the remaining single nodes of the domain. For each line to be solved, the corresponding n_l by z linear system block matrices are constructed, with $z \in [1, N_{tot}]$ and N_{tot} the total number of nodes in the domain. Each $k \times k$ block in the matrix can be, for instance, of size $k = 5$ for 3D laminar problems or $k = 6$ for 3D RANS problems with Spalart-Allmaras one-equation model (Spalart and Allmaras 1992). This system can be expressed as a subset of the whole linear system

(Eq. (8)) by the line indices l as shown in Eqs. (19) and (20) and the Figure 2. This linear system is solved directly by LU decomposition.

The line-implicit LU-SGS algorithm is presented in Eqs. (19) and (20). The computations of Eqs. (19) and (20) apply to each line l of size n_l or node if $n_l = 1$. The total number of lines is defined by N_l . It can be noted that a computation by lines of one node each, implies $N_l = N_{tot}$. $\mathbf{x}^*$, $\mathbf{x}^n$ and $\mathbf{x}^{n+1}$ are initialized to zero. The algorithm is described inside an inner loop of N_{swp} sweeps.

For $iter = 1 : N_{swp}$

Forward sweep

For $l = 1, 2, \dots, N_l$

$$\mathbf{B}_l \mathbf{x}_l^* = \mathbf{b}_l - \mathbf{L}_l \mathbf{x}^* - \mathbf{U}_l \mathbf{x}^n \quad (19)$$

Backward sweep

For $l = N_l, N_l - 1, \dots, 1$

$$\mathbf{B}_l \mathbf{x}_l^{n+1} = \mathbf{b}_l - \mathbf{U}_l \mathbf{x}^{n+1} - \mathbf{L}_l \mathbf{x}^* \quad (20)$$

$n = n + 1$;

End $iter$;

$\mathbf{L}_l$ and $\mathbf{U}_l$ are composed of blocks $\mathbf{L}_{ij}$ and $\mathbf{U}_{ij}$, respectively (see Figure 2) defined in Eq. 18.

5. MPI parallelization

Before considering the parallelization using MPI we have to understand the data dependencies of the solution algorithms. The data dependencies are linked to the ordering of the unknowns and equations. For Edge, the ordering is associated with the graph of the grid so the ordering of unknowns and equations follows from the node numbering. Edge is parallelized by domain decomposition which means that the domain of computation is partitioned by subdomains corresponding to the number of processors as shows Figure 3. Inside each subdomain, Edge reorders the node numbers by a process to reduce the matrix bandwidth and thus improve cache memory coherence. Because of the strong data dependency compared to a Jacobi method, the

Figure 3. Domain decomposition for parallelization, 8 processors. NACA0012 airfoil.

original LU-SGS algorithm does not parallelize in a straightforward way by domain decomposition.

Clearly we wish to retain the LU-SGS convergence speed also after domain decomposition. The idea is to remove the data dependency at inter-domain boundaries, by applying a Jacobi iterative

method. Sharov et al. (2000) and Luo et al. (2003) refer to this method as a hybrid approach, the so called hybrid LU-SGS or HLU-SGS. The idea was also applied for structured grids by Wissink et al. (1997). Moreover Sharov et al. (2000) have shown that interprocessor data exchange is necessary for better performance on high number of processors. Since here we are strongly interested by a high number of processors, the MPI data exchange will always be applied in our implementation as following.

The parallel LU-SGS version has been implemented as in Figure 4. The synchronized MPI data communication is executed after each forward sweep and for multiple iterations, also at the end of each iteration after the backward sweep. The Jacobi iterative algorithm inside LU-SGS results in a dynamic reordering of the outer connected nodes after each MPI data exchange. In the forward sweep, these nodes are located in the upper matrix, and in the backward sweep, in the lower matrix. By this way, the values of the outer connected nodes used by processor 1 are always from the previous sweep computed by processor 2 in the connected domain.

Figure 4. LU-SGS MPI parallelization.

The implementation has been done such that the serial LU-SGS code, which is the original version, switches into its parallel version by detection of a number of partitions bigger than 1. Thus the LU-SGS denomination refers to its serial (or original) and parallel version, the last being mainly used for all the simulations. This technique of parallelization applies also to its combination with the line-implicit method, the line-implicit LU-SGS, section 4. The degree of convergence deceleration suffered by the proposed scheme compared to a sequential LU-SGS is analyzed in section 7.3.

6. Ordering techniques

From the strong data dependency of LU-SGS follows that the ordering of the nodes will influence the LU-SGS convergence (Otero 2012). It is natural to look for the optimal ordering, but it is very hard to find for unstructured grids. The idea has been then to limit the search to *numberings which make geometric neighbor nodes have small number differences*. We start with variants of methods developed for low fill-in in Gaussian elimination on sparse matrices (Otero 2012). In our setting, which never creates the “big” matrix for elimination, such orderings would provide spatial and temporal data locality which means to use data closely stored in the memory (spatial locality) or being reused as much as possible (temporal locality).

The Cuthill-McKee (CM) algorithm (George and Liu 1981) groups by levels the nodes in the mesh. The nodes are ordered by increasing order of degree corresponding to the number of neigh-

boring nodes. Reversing the final CM ordering produces the Reverse CM which may be the most used profile reduction ordering method (George and Liu 1981). The choice of starting node is a degree of freedom, and we add a physical approach (Otero 2012) to CM by first numbering the nodes along the boundaries, namely the airfoil wall and the outer boundary. The levels created will follow the desired starting boundary as shown in Figure 5 to make numbering in layers.

Figure 5. Level structure from strong boundary.

7. Numerical Results

The convergence rate (residual decrease per time-step) and residual decrease vs. wall clock time are used for comparison with the Edge standard scheme, the line-implicit and explicit Runge-Kutta methods. Multigrid acceleration and local time-steps are used in both, LU-SGS and Runge-Kutta. Similar comparisons but for a high-order spectral volume method can be found in Parsani et al. (2010). The iteration number is the number of time-steps, each including a MG W-cycle usually through three grid levels. The CFL number used with LU-SGS, is for most of the cases, 1000 and with line-implicit Runge-Kutta, 10000 (Eliasson et al. 2009). More details of parameter configuration can be found in our previous work (Otero and Eliasson 2014). The different cases run are shown as following with their corresponding meshes and flow conditions.

7.1. Test cases

2D computations are run on NACA0012 airfoil unstretched (Euler) grid, Figure 6, and stretched (RANS) grid, Figure 7, for inviscid and viscous flow respectively. The RANS mesh has a maximum aspect ratio of the stretching of 15×10^3 and a normal distance from the solid surface to the first interior node of 10^{-6} chord length. The lines contain about 40 nodes. The number of grid points is of 16K and 54K for the unstretched and stretched meshes respectively. On the Euler mesh, the flow is characterized by a free stream Mach number of $M_\infty = 0.8$ and an angle of attack, $\alpha = 1.25^\circ$. The flow conditions on the RANS mesh are a Reynolds number of $Re = 6.2 \times 10^6$ with a free stream Mach number of $M_\infty = 0.754$ and an angle of attack, $\alpha = 2.5^\circ$.

2D inviscid computations have been also run over a RAE2822 airfoil on an unstructured unstretched mesh, Figure 8. The triangular grid contains 31K cells and 16K nodes. The flow conditions are the same than for the NACA0012 airfoil inviscid mesh, Figure 6.

3D Euler computations are run over the ONERA M6 wing, Figure 9, an unstructured unstretched grid with 226K grid nodes and 1.4×10^6 tetrahedral cells. The flow conditions are $M_\infty = 0.84$ and an angle of attack, $\alpha = 3.06^\circ$ angle of attack.

For 3D RANS, the ONERA M6 wing unstructured stretched grid is shown in Figure 10. The grid has 917K grid nodes, 30 prismatic layers, 1.2×10^6 prismatic cells and 1.8×10^6 tetrahedral cells. The maximum aspect ratio of the cells on the surface is 12×10^3 . The wall normal distance to the first interior node is about 1.5×10^{-6} chords. The implicit lines contain 24-30 nodes. The

Figure 6. NACA0012 airfoil structured unstretched mesh, 16K grid nodes.

Figure 7. NACA0012 airfoil unstructured stretched mesh, 54K grid nodes.

Figure 8. RAE2822 airfoil unstructured unstretched mesh, 16K grid nodes.

Figure 9. M6 wing surface mesh. Unstructured unstretched grid with 226K grid nodes.

flow is characterized by a Reynolds number of $Re = 11.72 \times 10^6$ at $\alpha = 3.06^\circ$ angle of attack with a free stream Mach number of $M_\infty = 0.84$.

CRM wing-body is another mesh used for 3D RANS. The model comes from DLR and has been used in the fourth Drag Prediction Workshop (DPW4). The mesh has been generated with Solar and is an unstructured stretched grid with 34M grid points, Figure 11. It contains 38.6×10^6 tetrahedral elements, 72.7×10^3 prismatic elements and has a maximum aspect ratio of 15×10^3 . The flow conditions are $Re = 5 \times 10^6$ with a free stream Mach number of $M_\infty = 0.85$ and an angle of attack, $\alpha = 2.5^\circ$.

Figure 10. M6 wing surface mesh. Unstructured stretched grid with 917K grid nodes.

Figure 11. DPW4 CRM fine surface mesh. Unstructured stretched grid with 34M grid nodes.

7.2. Ordering techniques

The ordering techniques are investigated as following on structured and unstructured 2D meshes for inviscid transonic flow computations.

Structured meshes

The ordering from the generation of the structured single 2D C-type mesh of the NACA0012 airfoil, Figure 6, is lexicographic. Figure 12, where the z -axis counts node number, shows the ordering resulting from starting along the wake and boundary airfoil wall and ending at the outer layer of the C-type mesh.

Figure 12. C-type mesh. NACA0012 airfoil.

Figure 13 shows the convergence of the different ordering techniques implemented based on the CM algorithm compared to the lexicographic ordering. The CM methods differ by the initial vectors which contain different sequences of node numbers and can have some sections reversed

(Otero 2012). These initial vectors preserve the pattern from the lexicographic ordering method, following the airfoil and wake (Otero 2012). These similar patterns give however slower convergences than the lexicographic ordering (Figure 13). It turns out that many variations on this theme give non-convergent iterations which are sensitive to the starting vector of the CM algorithm. Note that the lexicographic ordering though successful on Euler meshes would be disastrous for Navier-Stokes meshes.

Figure 13. Ordering techniques on C-type mesh. Euler computations over NACA0012 airfoil.

Unstructured meshes

Unstructured grids are usually generated by advancing front (AF) or Delaunay methods, and we compare such orderings with the CM variants. The study has been carried out over the RAE2822 airfoil, Figure 8. The original ordering comes from an advancing front technique, Figure 14, with the airfoil in the center. The fronts start at airfoil and outer boundary and advance, numbering the points in the process, until collision as can be observed by the free nodes remaining.

Figure 14. Advancing front ordering shape. RAE2822 airfoil.

Figure 15 illustrates the CM ordering path, starting at the airfoil (cf. zoom) and at the outer boundary. As we can observe, CM follows exactly the pattern creating level structures (Figure 5).

We can notice in Figure 16a, that the shapes from CM and advancing front of Figure 14 look almost identical (Otero 2012). The difference is that the nodes which are created when the advancing fronts collide and seem “randomly” numbered in AF, have been brought into the pattern by CM. This cleaner technique, *CM outer-wall*, leads to similar convergence acceleration than AF as shown in Figure 16d. Other patterns as *increasing x* coordinate have given slower convergence.

Figure 15. CM ordering starting at the strong boundaries. RAE2822 airfoil.

Figure 16. Ordering techniques on RAE2822 airfoil. Shapes: CM outer-wall (a), CM wall (b), increasing x (c). Residual convergence (d).

7.3. MPI parallelization

The results shown in Figure 17 are RANS parallel computations with the line-implicit LU-SGS solver, one sweep, run over the NACA0012 airfoil, Figure 7. The following analysis applies also to the implicit LU-SGS (without lines), to Euler computations (Otero and Eliasson 2013), and to several sweeps. The extremely low degree of convergence deceleration suffered by the proposed scheme compared to a sequential LU-SGS, is shown in Figure 17 (left). This means that the LU-SGS solver can be parallelized with negligible effects on convergence rate. Moreover Figure 17 (right) shows that LU-SGS scales linearly with number of processors. Another observation concerns the effect of the number of sweeps on the parallelization performance. Whereas Wissink et al. (1997) indicates that 2 sweeps are necessary for good parallel performance and almost identical convergence than the original LU-SGS, our method has shown to be as efficient as in Figure 17 independently of the number of sweeps applied. This property gives us flexibility in the parameter variation for

convergence acceleration (Otero and Eliasson 2014).

Figure 17. Density residual convergence in iterations (left) and time on single processor (right) for different number of processors. RANS computations on NACA0012 airfoil stretched mesh, 54K grid nodes.

The speed-ups obtained can be found in Figure 18. The computations have been run with implicit LU-SGS for a 3D case of 1.7M grid points in RANS simulations. Linear speed up has been noted up to 32 processors. LU-SGS shows better scalability than Runge-Kutta because of the increased processor-local work required by the linear algebra work.

Figure 18. Speed-up for RANS computations on a 3D case (LANN wing), 1.7M grid points. 1-312 processors.

7.4. *Line-implicit LU-SGS*

The combined line-implicit LU-SGS has been run for 2D RANS computations with 4 processors on the NACA0012 airfoil of Figure 7. The lines were computed on the prismatic layer defined on Figure 19. Other line definitions, than the one presented in section 4.1, by extending the stretched lines outside the boundary layer or by creating new lines outside the prismatic layer were implemented but did not improve the convergence. The creation of lines in an isotropic mesh does not seem to make sense.

Therefore, because of the semicoarsening of the MG agglomeration process, applied where the grid is stretched, and decreasing strongly the aspect ratio, line-implicit LU-SGS is only used in the finest grid level of multigrid (Eliasson et al. 2009).

Figure 19. Stretched lines in the prismatic layer. NACA0012 airfoil stretched mesh.

The efficiency of the combined solver is illustrated in Figure 20. The convergence in iterations is 14 times faster than the line-implicit Runge-Kutta solver, Figure 20 (left), and 4 times faster in time, Figure 20 (right). By increasing the number of inner iterations or sweeps N_{swp} to 2, the convergence rate increases by a factor almost of 2 compared to one sweep, decreasing the computing time, Figure 20 (right). Our previous parameter analysis carried out (Otero and Eliasson 2014) showed the existence of an optimized number of sweeps for best convergence, decreasing with coarser meshes. This explains the small optimal number of sweeps close to one, for 2D meshes, where $N_{swp} = 3$ causes deceleration.

The improvement with respect to the “original” LU-SGS is relevant being approximately 4 times faster in iterations, Figure 20 (left), and time, Figure 20 (right). As expected for RANS meshes, explicit Runge-Kutta is clearly the most inefficient solver and is strongly accelerated combined with a line-implicit method (Eliasson et al. 2009).

Figure 20. Residual convergence in iterations (left) and time on single processor (right). 2D RANS with 4 processors on NACA0012 airfoil stretched mesh, 54K grid nodes. Comparison with the line-implicit method.

7.5. Euler and RANS computations

2D and 3D Euler, RANS computations are shown in Figures 21 and 22 below. The results show the convergence acceleration achieved in iterations and time of the LU-SGS solver with lines (line-implicit LU-SGS) and without lines (implicit LU-SGS) compared to explicit and line-implicit Runge-Kutta. For the RANS computations the acceleration comparison measured is done between the line-implicit solvers unless specified.

The accelerations in time coming from LU-SGS are about twice faster as in Figure 22d, up to about 3 and 4 times faster, Figures 21d,b and Figure 22b respectively. This factor of acceleration can vary with the magnitude of residual reduction where 6 orders of magnitude is generally enough for a converged solution (Otero and Eliasson 2014). Moreover LU-SGS can converge to higher orders of residual reduction magnitude compared to Runge-Kutta as in Figures 22c and d among many other cases tested. The success of the line-implicit method presented in section 4 is again

illustrated, this time in a much bigger mesh of 34M grid points, Figures 22c and d. At around 5 orders of magnitude of residual reduction, line-implicit LU-SGS is twice faster than line-implicit Runge-Kutta and about three times faster than implicit LU-SGS and explicit Runge-Kutta.

The impact in convergence rate with LU-SGS is even stronger going from a factor of acceleration of the order of 7 and 9, Figure 21a and Figure 22c, up to 14, Figure 22a, and 21, Figure 21c. In fact big CFL numbers used with the LU-SGS solver decrease the total number of iterations necessary for convergence.

By observing the convergence in iterations and time, the trade-off is to obtain the best compromise between number of iterations and computing time per iteration. In this sense, implicit LU-SGS and explicit Runge-Kutta are complementary. Whereas LU-SGS has more work per iteration than Runge-Kutta as shows Table 1, but needs less iterations for convergence, Runge-Kutta will require a bigger number of iterations of less computing time each iteration, Table 1.

Figure 21. Euler parallel computations with 4 processors on unstretched grid for NACA0012 airfoil (a, b), 16K grid nodes, and M6 wing (c, d), 226K grid nodes. Density residual convergence in iterations (left) and time on single processor (right).

Figure 22. RANS parallel computations on stretched grid for NACA0012 airfoil, 54K grid nodes, with 4 processors (a, b), and CRM fine wing-body, 34M grid nodes, with 384 processors (c, d). Density residual convergence in iterations (left) and time on single processor(right)

Computing times per step for inviscid, laminar viscous, and RANS with Spalart-Allmaras turbulence model are shown in Table 1 on M6 wing unstretched and stretched meshes, Figures 9 and 10 respectively. The comparisons are done between implicit LU-SGS and explicit Runge-Kutta, and, line-implicit LU-SGS and line-implicit Runge-Kutta. The floating point work for LU-SGS as well as the Runge-Kutta scheme is proportional to the number of edges in the mesh graph and one would expect these ratios to be independent of spatial dimension, grid size, etc. But they do depend on the hardware since the compute/memory traffic ratio is different for the linear algebra of LU-SGS and the flux computations that dominate the RK scheme. As expected, the amount of extra work added by the line-implicit method reduces the ratio between LU-SGS and Runge-Kutta for laminar and turbulence model which is approximately halved when computing both with the line-implicit method.

Table 1. Time in seconds per iteration; M6 wing unstretched (Euler) and stretched meshes (Laminar and RANS).

	Euler	Laminar	RANS
Imp. LU-SGS	10.8	26	35.5
Exp. RK	3	7	8.5
RATIO LU-SGS/RK	3.6	3.7	4.2
Line-imp. LU-SGS	-	30.5	40.2
Line-imp. RK	-	14.4	22
RATIO LU-SGS/RK	-	2.1	1.8

Table 2 summarizes the LU-SGS convergence rates in decades of residual reduction per 1000 iterations. The cases considered are Euler and RANS simulations for 2D and 3D problems from the Figures 21 and 22. LU-SGS solver shows to be mostly efficient for Euler computations with a factor up to 8.5 in decades of residual reduction per 1000 iterations compared to the other cases.

For RANS cases, the best performance has been obtained with the combined line-implicit LU-SGS method. Its performance becomes maximum for 2D case, slightly over the value for the 3D Euler computation and about 5 times bigger than the decades obtained without the line-implicit method.

Table 2. Residual reduction in decades per 1000 iterations with LU-SGS for 2D and 3D, Euler and RANS Computations.

Simulation	Decades /1000 It.	
	2D	3D
Euler (unstretched mesh)	26.6	13.3
RANS (stretched mesh)		
<i>Implicit LU-SGS</i>	3.12	3.2
<i>Line-implicit LU-SGS</i>	14.1	3.9

7.6. Multigrid effect

An analysis has been carried out on the role of multigrid in the convergence acceleration procedure. The motivation to use a multigrid strategy comes from its strong convergence acceleration on Euler grids. However it is also known that multigrid is inefficient on RANS meshes (Mavriplis 1998) which led us to investigate the real contribution of multigrid on the results obtained. So the question is, do we need multigrid?

From the tests run, multigrid has generally shown a relevant impact on the convergence acceleration. However for 3D RANS computations, the results have been different. Figure 23 illustrates the last statement, where a comparison is done between line-implicit LU-SGS, with and without multigrid. The configuration used was, three MG levels, the LU-SGS smoother only on the finest grid with optimal parameters (cf. (Otero and Eliasson 2014) for details) as CFL numbers of 100 and 1000, and Runge-kutta smoothing on the coarser grid levels. The cases were run on the M6 wing, the CRM wing-body, both of them described in section 7.1, and the NASA “Trap Wing” model with flap and slat with 110M grid nodes, used in the first AIAA CFD High Lift Prediction Workshop (HiLiftPW-1) (Rumsey 2010). These examples show that indeed multigrid does not affect the convergence rate and the conclusion was that multigrid is not needed when using LU-SGS with optimal parameters.

Figure 23. Multigrid (MG) effect for 3D RANS parallel computations. Density residual convergence in iterations.

8. Conclusion

The overall conclusion is that LU-SGS does provide faster computation and scales better on distributed memory machines than Runge-Kutta, accelerated with the line-implicit method on

stretched grids. The LU-SGS iteration as multigrid smoother does increase the convergence rate to steady state for all the flow models tested: transonic Euler, laminar Navier-Stokes, and RANS with turbulence model. The runtime scales linearly with number of processors and the Jacobi iterative method used at inter-domain boundaries to approximate the Gauss-Seidel update, seems to have no ill effects. Node numbering does influence the convergence. The native orderings from structured as well as unstructured 2D meshes generated by advancing fronts or Delaunay methods were, fortunately, among the best tested. There are several parameters to tune, but numerical experiments have led to default settings which seem robust for convergence acceleration.

For 2D and 3D, Euler and RANS computations, the wall clock time is at least halved to reach sufficient residual reduction compared to the explicit and line-implicit Runge-Kutta smoothers. For RANS computations with very thin cells on solid boundaries, further acceleration can be obtained by accurately solving the strongest couplings in the linear system, i.e., in the prismatic mesh layers along the edges transversal to solid boundaries and wake mean surfaces. A combination of the line-implicit technique with LU-SGS was implemented. For high orders of magnitude of residual reduction, this accelerated the convergence up to a factor of 14 in iterations and 4 in time, compared to the line-implicit Runge-Kutta solver. Finally, in the cases tested, the classical multigrid acceleration was found ineffective and not needed for 3D RANS simulations, in combination with the LU-SGS solver using optimal parameters.

References

- Barney, B. (2014). Message Passing Interface (MPI). <https://computing.llnl.gov/tutorials/mpi/>.
- Borne, S. L. (2000). Ordering Techniques for Two- and Three-Dimensional Convection-Dominated Elliptic Boundary Value Problems. Technical report, Lehrstuhl Praktische Mathematik, Universität Kiel, Germany.
- Brandt, A. (1977). Multi-Level Adaptive Solutions to Boundary-Value Problems. *Mathematics of Computation* 31(138), 333–390.
- Chen, R. F. and Z. J. Wang (2000). Fast, Block Lower-Upper Symmetric Gauss-Seidel Scheme for Arbitrary Grids. *AIAA Journal* 38(12), 2238–2245.
- Dwight, R. P. (2006). *Efficiency Improvements of RANS-Based Analysis and Optimization using Implicit and Adjoint Methods on Unstructured Grids*. Ph. D. thesis, School of Mathematics, University of Manchester.
- Eliasson, P. (2001). EDGE, A Navier-Stokes Solver for Unstructured Grids. Internal report FOI-R-0298-SE.
- Eliasson, P. (2002, June). Edge, a Navier-Stokes Solver for Unstructured Grids. In *Finite Volumes for Complex Applications III*, pp. 527–534.
- Eliasson, P. and J. Nordström (2013). The Influence of Viscous Operator and Wall Boundary Conditions on the Accuracy of the Navier-Stokes Equations. AIAA Paper 2013–2956.
- Eliasson, P. and P. Weinerfelt (2007). Recent Applications of the Flow Solver Edge. In *Proceedings to 7th Asian CFD Conference*, Bangalore, India.
- Eliasson, P., P. Weinerfelt, and J. Nordström (2009, January). Application of a Line-Implicit Scheme on Stretched Unstructured Grids. AIAA Paper 2009–163.
- George, A. and J. W.-H. Liu (1981). *Computer Solution of Large Sparse Positive Definite Systems*. Englewood Cliffs, New Jersey 07632: Prentice-Hall.
- Jameson, A. (1983). Transonic Flow Calculations. Technical Report MAE Report 1651, Princeton University.
- Jameson, A. and E. Turkel (1981). Implicit Schemes and LU Decompositions. *Mathematics of Computation* 37, 385–397.
- Jameson, A. and S. Yoon (1987). Lower-Upper Implicit Schemes with Multiple Grids for the Euler Equations. *AIAA Journal* 25(7), 929–935.
- Kim, J. S. and O. J. Kwon (2005). An Efficient Implementation of Implicit Operator for Block LU-SGS Method. *Computational Fluid Dynamics* 13(2), 154–159.
- Langer, S. (2013). Application of a Line Implicit Method to Fully Coupled System of Equations for Turbulent Flow Problems. *International Journal of Computational Fluid Dynamics* 27(3), 131–150.
- Langer, S. and D. Li (2011). Application of Point Implicit Runge-Kutta Methods to Inviscid and Laminar Flow Problems using AUSM and AUSM⁺ Upwinding. *International Journal of Computational Fluid*

- Dynamics* 25(5), 255–269.
- Luo, H., D. Sharov, and J. D. Baum (2003). Parallel Unstructured Grid GMRES+LU-SGS Method For Turbulent Flows. AIAA Paper 2003–0273.
- Mavriplis, D. (1998). On Convergence Acceleration Techniques for Unstructured Meshes. Technical Report ICASE No. 98-44 and NASA/CR-1998-208732, Institute for Computer Applications in Science and Engineering, NASA Langley Research Center.
- Mavriplis, D. J. (1999). Directional Agglomeration Multigrid Techniques for High-Reynolds-Number Viscous Flows. *AIAA Journal* 37(10), 1222–1230.
- Nejat, A. and C. Ollivier-Gooch (2005, June). A High-Order Accurate Unstructured GMRES Algorithm for Inviscid Compressible Flows. AIAA Paper 2005–5341.
- Otero, E. (2012, June). Acceleration of Compressible Flow Simulations with Edge using Implicit Time Stepping. Licentiate thesis, University of KTH, Stockholm.
- Otero, E. and P. Eliasson (2011). Convergence Acceleration of the CFD code Edge by LU-SGS. In *3rd CEAS Air&Space Conference*, Venice, Italy, pp. 606–613.
- Otero, E. and P. Eliasson (2013). Improving the Performance of the CFD code Edge using LU-SGS and Line-Implicit Methods. In *Proceedings of the 4:th CEAS Conference in Linköping, 2013*, Linköping, Sweden, pp. 124–134. Linköping University Electronic Press.
- Otero, E. and P. Eliasson (2014, June). Parameter Investigation with Line-Implicit Lower-Upper Symmetric Gauss-Seidel on 3D Stretched Grids. AIAA Paper 2014–2094.
- Parsani, M., K. V. den Abeele, C. Lacor, and E. Turkel (2010). Implicit LU-SGS Algorithm for High-Order Methods on Unstructured Grid with P-Multigrid Strategy for Solving the Steady Navier-Stokes Equations. *Journal of Computational Physics* 229, 828–850.
- Roe, P. L. (1981). Approximate Riemann Solvers, Parameter Vectors, and Difference Schemes. *Journal of Computational Physics* 43, 357–372.
- Rossow, C. C. (2006). Convergence Acceleration for Solving the Compressible Navier-Stokes Equations. *AIAA Journal* 44(2), 345–352.
- Rumsey, C. (2010). The 1st AIAA CFD High Lift Prediction Workshop (HiLiftPW-1). <http://hilitpw.larc.nasa.gov/index-workshop1.html>.
- Rumsey, C. (2014). Turbulence Modeling Resource. <http://turbmodels.larc.nasa.gov>.
- Sharov, D., H. Luo, J. D. Baum, and R. Löhner (2000). Implementation of Unstructured Grid GMRES+LU-SGS Method on Shared-Memory, Cache-based Parallel Computers. AIAA Paper 2000–0927.
- Sharov, D. and K. Nakahashi (1998). Reordering of Hybrid Unstructured Grids for Lower-Upper Symmetric Gauss-Seidel Computations. *AIAA Journal* 36(3), 484–486.
- Soetrisno, M., S. Imlay, D. Roberts, and D. Taffin (1996, January). Development of a 3-D Zonal Implicit Procedure for Hybrid Structured-Unstructured Grids. AIAA Paper 96–0167.
- Spalart, P. R. and S. R. Allmaras (1992). A One-Equation Turbulence Model for Aerodynamic Flows. AIAA Paper 1992–0439.
- Swanson, R. C., E. Turkel, and C. C. Rossow (2007). Convergence Acceleration of Runge-Kutta Schemes for Solving the Navier-Stokes Equations. *Journal of Computational Physics* 224(1), 365–388.
- Turkel, E., V. Vatsa, and V. Venkatakrishnan (1999). Uni-Directional Implicit Acceleration Techniques for Compressible Navier-Stokes Solvers. AIAA Paper 1999–3265.
- Venkatakrishnan, V. (1998). Improved Multigrid Performance of Compressible Navier-Stokes Solvers. AIAA Paper 1998–2967.
- Wissink, A., A. Lyrintzis, and A. Chronopoulos (1997). A Parallel Newton-Krylov Method for Rotorcraft Flowfield Calculations. AIAA Paper 1997–2049.
- Wright, M., G. Candler, and M. Prampolini (1995). A Data-Parallel LU Relaxation Method for the Navier-Stokes Equations. AIAA Paper 1995–1750.
- Yoon, S. and D. Kwak (1992). Implicit Navier-Stokes Solver for Three Dimensional Compressible Flows. *AIAA Journal*, 2653–2659.