Proceedings of the XXVI Iberian Latin-American Congress on Computational Methods in Engineering – CILAMCE 2005
Brazilian Assoc. for Comp. Mechanics (ABMEC) & Latin American Assoc. of Comp. Methods in Engineering (AMC), Guarapari, Espírito Santo, Brazil, 19th – 21st October 2005

Paper CIL 12-0423

EDGE-BASED DATA STRUCTURES FOR THE PARALLEL SOLUTION OF 3D STEADY INCOMPRESSIBLE FLUID FLOW WITH THE SUPG/PSPG FINITE ELEMENT FORMULATION.

Renato N. Elias
Marcos A. D. Martins
Alvaro L. G. A. Coutinho
Rubens M. Sydenstricker
Center for Parallel Computations and Department of Civil Engineering
Federal University of Rio de Janeiro, P. O. Box 68506,
RJ 21945-970 – Rio de Janeiro, Brazil
{renato, marcos, alvaro, rubens}@nacad.ufrj.br

Abstract. The parallel edge-based SUPG/PSPG finite element formulation applied to 3D steady incompressible Navier-Stokes equations is presented. The highly coupled velocity-pressure nonlinear system of equations is solved with an inexact Newton-like method. The locally linear system of equations originated by the inexact nonlinear method is solved with a nodal block diagonal preconditioned GMRES solver. Matrix-vector computations within the GMRES solver are computed edge-by-edge, diminishing floating point operations, and indirect memory addressing and memory requirements. The parallel implementation comprising message passing parallelism is addressed and the results for the three dimensional lid driven cavity flows and the laminar flow through a race car body are discussed. Results have shown that edge-based data structure is more efficient than traditional element-based schemes and the parallel inexact non-linear solver has shown good performance and robustness.

Keywords. Edge-based data structure, Parallel finite elements, Inexact Newton methods, Navier-Stokes equations.

1 INTRODUCTION

We consider the simulation of steady incompressible fluid flow governed by Navier-Stokes equations using the stabilized finite element formulation as shown in Tezduyar (1991). This formulation allows that equal-order-interpolation velocity-pressure elements are employed by introducing two stabilization terms: the Streamline Upwind Petrov-Galerkin (SUPG) and the Pressure Stabilizing Petrov Galerkin stabilization (PSPG).

When discretized, the incompressible Navier-Stokes equations give rise to a fully coupled velocity-pressure system of nonlinear equations due the presence of convective terms in momentum equation. The inexact Newton method (Dembo et al., 1982) associated with a proper preconditioned iterative Krylov solver, such as GMRES, presents an appropriated framework to solve nonlinear systems, offering a trade-off between accuracy and the amount of computational effort spent per iteration.

Element-based data structures have been extensively used in the implementation of iterative solvers. A wide class of preconditioners specially designed for element-by-element

(EBE) techniques can be found in literature (Ferencz and Hughes, 1998). This type of data structure, besides fully exploiting the sparsity of the system, is suitable for parallelization and vectorization, as matrix-vector (matvec) products can be written in the form of a single loop structure (Coutinho et al., 1991). Another way of writing an efficient code for matvec products is by using the Compressed Storage Row (CSR) format (Saad, 1996), in which only global nonzero coefficients are stored. In the CSR format, the algorithm for matrix-vector products requires a two-loop structure: an outer loop over the equations and an inner loop over each nonzero contribution. In comparison with EBE implementations, this format has the advantage of storing only global coefficients, but with the overhead of an inner loop that can not be unrolled, specially if unstructured meshes are to be considered.

Edge-based data structures or EDE for short can be viewed as a blend of EBE and CSR formats: only global nonzero coefficients are stored and the single loop structure is maintained. In the context of finite elements, this type of data structure has been used since the early 1990s. Peraire et al. (1993) and Lyra et al. (1995) used edge-based data structures to compute the nodal balance of fluxes for compressible flow. Luo et al. (1994) used an edge-based approach in the development of an upwind finite element scheme for the solution of the Euler equations. Löhner (1994) showed different ways of grouping edges in the evaluation of residuals for the Laplacian operator, aiming to reduce the number i/a operations. More recently, Ribeiro et al. (2001) presented an edge-based implementation for stabilized semi-discrete and space-time finite element formulations for shallow water equations. Other recent works on the subject include those of Coutinho et al. (2001), with applications to nonlinear solid mechanics, Catabriga and Coutinho (2002), for the implicit SUPG solution of the Euler equations, and Soto et al. (2004) for incompressible flow problems. It has been shown by Ribeiro and Coutinho (2005) that for unstructured grids composed by tetrahedra, edge-based data structures offer more advantages than CSR, particularly for problems involving many degrees of freedom.

When dealing with large scale problems the use of parallel solvers is an essential condition and the use of an algorithm able to run efficiently in shared, distributed or hybrid memory systems has been a motivation for many researchers to turn the solver strategy more independent of the hardware resources.

The present paper is organized as follows. Sections 2 present the SUPG/PSPG finite element formulation and nonlinear solution adopted. In section 3 we describe the edge-based data structure. Some parallel aspects are briefly introduced in section 4 and the last two sections present the results for two benchmarks problems considered and our final discussions.

2 GOVERNING EQUATIONS

Let $\Omega \subset \mathbb{R}^{n_{sd}}$ be the spatial domain, where n_{sd} is the number of space dimensions. Let Γ denote the boundary of Ω. We consider the following velocity-pressure formulation of the Navier-Stokes equations governing steady incompressible flows:

$$\rho\left(\mathbf{u}\cdot\nabla\mathbf{u}-\mathbf{f}\right)-\nabla\cdot\boldsymbol{\sigma}=\mathbf{0} \qquad \text{on } \Omega \tag{1}$$

$$\nabla\cdot\mathbf{u}=\mathbf{0} \qquad \text{on } \Omega \tag{2}$$

where ρ and $\mathbf{u}$ are the density and velocity, $\boldsymbol{\sigma}$ is the stress tensor given as

The essential and natural boundary conditions associated with equations (1) and (2) can be imposed at different portions of the boundary Γ and represented by,

$$\mathbf{u} = \mathbf{g} \qquad \text{on } \Gamma_g \tag{3}$$

$$\mathbf{n} \cdot \boldsymbol{\sigma} = \mathbf{h} \qquad \text{on } \Gamma_h \tag{4}$$

where Γ_g and Γ_h are complementary subsets of Γ.

Let us assume following Tezduyar (1991) that we have some suitably defined finite-dimensional trial solution and test function spaces for velocity and pressure, $S_{\mathbf{u}}^h$, $V_{\mathbf{u}}^h$, S_p^h and $V_p^h = S_p^h$. The finite element formulation of equations (1) and (2) using SUPG and PSPG stabilizations for incompressible fluid flows can be written as follows: Find $\mathbf{u}^h \in S_{\mathbf{u}}^h$ and $p^h \in S_p^h$ such that $\forall \mathbf{w}^h \in V_{\mathbf{u}}^h$ and $\forall q^h \in V_p^h$:

$$\begin{aligned}
&\int_\Omega \mathbf{w}^{\mathrm{h}} \cdot \rho\left(\mathbf{u}^{\mathrm{h}} \cdot \nabla \mathbf{u}^{\mathrm{h}} - \mathbf{f}\right) d\Omega + \int_\Omega \boldsymbol{\varepsilon}\left(\mathbf{w}^{\mathrm{h}}\right) : \boldsymbol{\sigma}\left(p^{\mathrm{h}}, \mathbf{u}^{\mathrm{h}}\right) d\Omega - \int_\Gamma \mathbf{w}^{\mathrm{h}} \cdot h \, d\Gamma + \int_\Omega q^{\mathrm{h}} \nabla \cdot \mathbf{u}^{\mathrm{h}} d\Omega \\
&\quad + \sum_{e=1}^{n_{el}} \int_{\Omega^e} \tau_{SUPG} \mathbf{u}^h \cdot \nabla \mathbf{w}^h \cdot \left[\rho\left(\mathbf{u}^h \cdot \nabla \mathbf{u}^h\right) - \nabla \cdot \boldsymbol{\sigma}\left(p^h, \mathbf{u}^h\right) - \rho \mathbf{f}\right] d\Omega \\
&\quad + \sum_{e=1}^{n_{el}} \int_{\Omega^e} \frac{1}{\rho} \tau_{PSPG} \nabla q^h \cdot \left[\rho\left(\mathbf{u}^h \cdot \nabla \mathbf{u}^h\right) - \nabla \cdot \boldsymbol{\sigma}\left(p^h, \mathbf{u}^h\right) - \rho \mathbf{f}\right] d\Omega = 0
\end{aligned} \tag{5}$$

In the above equation the first four integrals on the left hand side represent terms that appear in the Galerkin formulation of the problem (1)-(4), while the remaining integral expressions represent the additional terms which arise in the stabilized finite element formulation. Note that the stabilization terms are evaluated as the sum of element-wise integral expressions, where n_{el} is the number of elements in the mesh. The first summation corresponds to the SUPG (Streamline Upwind Petrov/Galerkin) term and the second to the PSPG (Pressure Stabilization Petrov/Galerkin) term. We have calculated the stabilization parameters according to Tezduyar et al. (1992), as follows:

$$\tau_{SUPG} = \tau_{PSPG} = \left[\left(\frac{2\left\| \mathbf{u}^h \right\|}{h^\#} \right)^2 + 9 \left(\frac{4\nu}{\left(h^\#\right)^2} \right)^2 \right]^{-1/2} \tag{6}$$

Here $\mathbf{u}^h$ is the local velocity vector, ν represent the kinematic viscosity and the "element length" $h^\#$ is defined to be equal to the diameter of the sphere which is volume-equivalent to the element.

The spatial discretization of Eq. (11) leads to the following system of nonlinear equations,

$$\begin{aligned} &\mathbf{N}(\mathbf{u}) + \mathbf{N}_{\delta}(\mathbf{u}) + \mathbf{K}\mathbf{u} - (\mathbf{G} + \mathbf{G}_{\delta})\,\boldsymbol{p} = \mathbf{f}_{\mathbf{u}} \\ &\mathbf{G}^{T}\mathbf{u} + \mathbf{N}_{\psi}(\mathbf{u}) + \mathbf{G}_{\psi}\boldsymbol{p} = \mathbf{f}_{\boldsymbol{p}} \end{aligned} \tag{7}$$

where $\mathbf{u}$ is the vector of unknown nodal values of $\mathbf{u}^h$ and $\boldsymbol{p}$ is the vector of unknown nodal values of $\boldsymbol{p}^h$. The non-linear vectors $\mathbf{N}(\mathbf{u})$, $\mathbf{N}_{\delta}(\mathbf{u})$, $\mathbf{N}_{\psi}(\mathbf{u})$, and the matrices $\mathbf{K}$, $\mathbf{G}$, $\mathbf{G}_{\delta}$, and $\mathbf{G}_{\psi}$ emanate, respectively, from the convective, viscous and pressure terms. The vectors $\mathbf{f}_{\mathbf{u}}$ and $\mathbf{f}_{\boldsymbol{p}}$ are due to the boundary conditions (3) and (4). The subscripts δ and ψ identify the SUPG and PSPG contributions respectively. In order to simplify the notation we denote by $\mathbf{x} = (\mathbf{u}, \boldsymbol{p})$ a vector of nodal variables comprising both nodal velocities and pressures. Thus, Eq. (7) can be written as,

$$\mathbf{F}(\mathbf{x}) = \mathbf{0} \tag{8}$$

where $\mathbf{F}(\mathbf{x})$ represents a nonlinear vector function.

A particularly simple scheme for solving the nonlinear system of equations (8) is a fixed point iteration procedure known as the *successive substitution*, (also known as Picard iteration, functional iteration or successive iteration) which may be written as

$$\mathbf{J}_0(\mathbf{x}_k)\,\mathbf{x}_{k+1} = -\mathbf{r}_k \tag{9}$$

here the nonlinearity is evaluated at the known iterate $\mathbf{x}_k$ where $\mathbf{J_0}$ is a first order approximation of the Jacobian $\mathbf{J}(\mathbf{x})$, $\mathbf{r}_k$ is the residual vector, and a non-symmetric linear system must be formed and solved at each iteration. Although the convergence rate of this scheme can be slow (its convergence rate is only asymptotically linear), the method converges for a fair range of Reynolds numbers. This, together with its simplicity of implementation and relativity insensitivity to the initial iterate $\mathbf{x}_0$, accounts for its popularity and recommends its use for the solution of a wide variety of problems. However, as observed by Elias et al (2004), the fully coupled Jacobian may be numerically approximated by a truncated Taylor expansion, reducing the number of nonlinear iterations.

One drawback of nonlinear methods is the need to solve a local linear system (9) at each stage. Computing the exact solution using a direct method can be expensive if the number of unknowns is large and may not be justified when $\mathbf{x}_k$ is far from a solution (Dembo et al., 1982). Thus, one might prefer to compute some approximate solution, leading to the following algorithm:

For $k = 0$ *step 1 until convergence do*
  Find some $\eta_k \in [0,1)$ *AND* $\mathbf{s}_k$ *that satisfy*

$$\|\mathbf{r}_k + \mathbf{J}_0(\mathbf{x}_k)\,\mathbf{s}_{k+1}\| \le \eta_k \|\mathbf{r}_k\| \tag{10}$$

  update $\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{s}_k$

for some $\eta_k \in [0,1)$, where $\|\bullet\|$ is a norm of choice. This formulation naturally allows the use of an iterative solver like GMRES or BiCGSTAB: one first chooses η_k and then applies the iterative solver to (9) until a $\mathbf{s}_k$ is determined for which the residual norm satisfies (10). In this context η_k is often called the forcing term, since its role is to force the residual of (9) to be

suitably small. The forcing term can be specified in several ways (see, Eisenstat and Walker, 1996 for details) and in this work the following definition was employed

$$\eta_k = 0.9\|\mathbf{F}(\mathbf{x}_k)\|^2 \Big/ \|\mathbf{F}(\mathbf{x}_{k-1})\|^2 . \tag{11}$$

3 EDGE-BASED DATA STRUCTURE

Edge-based finite element data structures have been introduced for explicit computations of compressible flow in unstructured grids composed by tetrahedra (Peraire et al. 1993; Luo and Löhner, 1994). It was observed in these works that residual computations with edge-based data structures were faster and required less memory than standard element-based residual evaluations. Following these ideas, Coutinho et al (2001) and Catabriga and Coutinho (2002) derived edge-based formulations respectively for elasto-plasticity and the SUPG formulation for inviscid compressible flows. They used the concept of disassembling the finite element matrices to build the edge matrices. For three dimensional viscoplastic flow problems on unstructured meshes, we may derive an edge-based finite element scheme by noting that the element matrices can be disassembled into their edge contributions as,

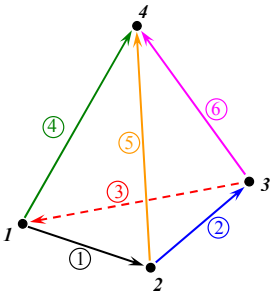
$$\mathbf{J}^e = \sum_{s=1}^{m} \mathbf{T}_s^e \tag{12}$$

where $\mathbf{T}_s^e$ is the contribution of edge *s* to $\mathbf{J}^e$ and *m* is the number of element edges per element (six for tetrahedral). For instance, Figure 1 shows a tetrahedron and its edge disassembling into its six edges.

In Figure 1, each $\mathbf{T}_{ij}^s$ is a *ndof* × *ndof* sub-matrix, where *ndof* is the number of degrees of freedom (four for fully coupled incompressible fluid flow). The arrows represent the adopted edge orientation. Denoting by $\mathcal{E}$ the set of all elements sharing a given edge *s*, we may add their contributions, arriving to the edge matrix,

$$\mathbf{J}_s = \sum_{s \in \mathcal{E}} \mathbf{T}_s^e \tag{13}$$

The resulting matrix for incompressible fluid flow is non-symmetric, preserving the structure of the edge matrices given in Figure 1. Thus, in principle we need to store two off-diagonal 4×4 blocks and two 4×4 diagonal blocks per edge. As we shall see below this storage area can be reduced.



$$\mathbf{J}^e = \begin{bmatrix} \mathbf{J}_{11} & \mathbf{J}_{12} & \mathbf{J}_{13} & \mathbf{J}_{14} \\ \mathbf{J}_{21} & \mathbf{J}_{22} & \mathbf{J}_{23} & \mathbf{J}_{24} \\ \mathbf{J}_{31} & \mathbf{J}_{32} & \mathbf{J}_{33} & \mathbf{J}_{34} \\ \mathbf{J}_{41} & \mathbf{J}_{42} & \mathbf{J}_{43} & \mathbf{J}_{44} \end{bmatrix}$$

Element stiffness matrix

$$\mathbf{T}_1^e = \begin{bmatrix} \mathbf{T}_{11}^1 & \mathbf{T}_{12}^1 & 0 & 0 \\ \mathbf{T}_{21}^1 & \mathbf{T}_{22}^1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Edge matrix 1

$$\mathbf{T}_2^e = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & \mathbf{T}_{22}^2 & \mathbf{T}_{23}^2 & 0 \\ 0 & \mathbf{T}_{32}^2 & \mathbf{T}_{33}^2 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Edge matrix 2

$$\mathbf{T}_3^e = \begin{bmatrix} \mathbf{T}_{11}^3 & 0 & \mathbf{T}_{13}^3 & 0 \\ 0 & 0 & 0 & 0 \\ \mathbf{T}_{31}^3 & 0 & \mathbf{T}_{33}^3 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Edge matrix 3

$$\mathbf{T}_4^e = \begin{bmatrix} \mathbf{T}_{11}^4 & 0 & 0 & \mathbf{T}_{14}^4 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \mathbf{T}_{41}^4 & 0 & 0 & \mathbf{T}_{44}^4 \end{bmatrix}$$

Edge matrix 4

$$\mathbf{T}_5^e = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & \mathbf{T}_{22}^5 & 0 & \mathbf{T}_{24}^5 \\ 0 & 0 & 0 & 0 \\ 0 & \mathbf{T}_{42}^5 & 0 & \mathbf{T}_{44}^5 \end{bmatrix}$$

Edge matrix 5

$$\mathbf{T}_6^e = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & \mathbf{T}_{33}^6 & \mathbf{T}_{34}^6 \\ 0 & 0 & \mathbf{T}_{43}^6 & \mathbf{T}_{44}^6 \end{bmatrix}$$

Edge matrix 6

Figure 1- Edge disassembling of a tetrahedral finite element for incompressible flow problems.

When working with iterative solvers like GMRES, it is necessary to compute sparse matrix-vector products at each iteration. A straightforward way to implement the edge-by-edge (or element-by-element) matrix-vector product is,

$$\mathbf{J}\,\mathbf{p} = \sum_{l=1}^{ne} \mathbf{J}^l \mathbf{p}^l \tag{14}$$

where *ne* is the total number of local structures (edges or elements) in the mesh and $\mathbf{p}^l$ is the restriction of $\mathbf{p}$ to the edge or element degrees-of-freedom.
Considering that we may add all the local nodal block diagonals in a global nodal block diagonal matrix and we may store only the off-diagonal blocks (by edges or elements), it is possible to improve the matrix-vector product (14) disregarding redundant nodal block diagonal coefficients for each local structure (Catabriga and Coutinho, 2002). This scheme is an extension of that proposed by Gijzen (1995) where is considered only the main diagonal instead of the nodal block diagonal. Thus, the matrix-vector product may be rewritten as

$$\mathbf{J}\mathbf{p} = \mathbf{B}(\mathbf{J})\mathbf{p} + \mathop{\mathbf{A}}_{l=1}^{ne} \left[\mathbf{J}^l - \mathbf{B}(\mathbf{J}^l)\right]\mathbf{p}^l \tag{15}$$

where **B**(**J**) stores the nodal block diagonal of **J** and $\mathbf{A}$ is the assembling operator. In the above matrix-vector product the first sum involves only global quantities while the second is very similar to the standard matrix-vector product (14). Note that we need to store only the off-diagonal blocks associated to each structure (edge or element), thus reducing memory requirements. It is also important to observe that in the case of the edge-based matrix-vector product (15) only global coefficients are involved in the computations. The resulting algorithm is,

For each local structure l *do:*

Recover the global numbering of the local degrees of freedom.

Gathering operation: $\mathbf{p}^l \leftarrow \mathbf{p}$

Perform product: $\mathbf{jp}^l = \left[\mathbf{J}^l - \mathbf{B}(\mathbf{J}^l)\right] \cdot \mathbf{p}^l$

Scattering and accumulation operations: $\mathbf{jp} \leftarrow \mathbf{jp} + \mathbf{jp}^l$

end for l

$$\mathbf{jp} = \mathbf{B}(\mathbf{J})\mathbf{p} + \mathbf{jp}$$

The **B(J)p** product performed at nodal level follows the same ideas described in the above algorithm (gather, local computations and scatter and accumulation). Considering the standard pointers, localization matrices and destination arrays of finite element implementations (Hughes, 1987) applied to edge data structures, it is easy to verify that pointers, localization matrices and destination arrays correspond to a two-node element, that is, an edge. In Table 1 we compare the storage requirements to hold the coefficients of the element and edge matrices as well as the number of floating point (*flop*) and indirect addressing (*i/a*) operations for computing matrix-vector products (15) using element and edge-based data structures.

Table 1. Memory to hold matrix coefficients and computational costs for element and edge-based matrix-vector products for tetrahedral finite element meshes.

Data structure	Memory	*flop*	*i/a*
Element	1056 *nnodes*	2112 *nnodes*	1408 *nnodes*
Edge	224 *nnodes*	448 *nnodes*	448 *nnodes*

All data in these tables are referred to *nnodes*, the number of nodes in the finite element mesh. According to Löhner (1994), the following estimates are valid for unstructured 3D grids, $nel \approx 5.5 \times nnodes$, $nedges \approx 7 \times nnodes$.

Clearly data in Table 1 favors the edge-based scheme. However, compared to the element data structure, the edge scheme does not present a good balance between *flop* and *i/a* operations. Minimizing indirect addressing and improving data locality are major concerns to achieve high performance on current processors. Löhner and Galle (2002) and Coutinho et al (2005) discuss several enhancements to the simple edge scheme introduced here, that mitigate the effects of indirect addressing and bad data locality. In the next section we describe the edge and node reordering techniques used here, which are less involved than those of Coutinho et al (2005).

4 PARALLEL IMPLEMENTATION

The parallel inexact nonlinear solver presented in the previous section was implemented based in the message passing parallelism model (MPI). The original unstructured grid was partitioned into non-overlapped sub-domains by the use of the METIS_PartMeshDual routine provided by Metis package (Karypis and Kumar, 1998). Afterwards, the partitioned data is reordered to avoid memory dependency with a mesh coloring algorithm and the data locality is improved through a Reverse Cuthill Mckee algorithm. Finally, the nodes shared among partitions are indexed to be used on message passing communications.

Most of the computational effort spent during the iterative solution of linear systems is due to evaluations of matrix-vector products or matvec for short. In our tests matvec operations achieved 92% of the total computational costs for element-by-element and 60% for edge-by-edge data structure. In element-by-element (EBE) and edge-by-edge (EDE) data structures this task is message passing parallelizable by performing matvec operations at each partition level, then assembling the contribution of the interface equations calling MPI_AllReduce routine. Finally, it is important to note that edge (and element) matrix coefficients are computed in single DO-LOOPS also in each partition. Figure 2 sketches how we have been computing hybrid (shared and distributed parallelisms simultaneously) matvec operations. In this figure we show a mesh with two partitions and colored in disjoint blocks to avoid data dependency for vectorization, pipelining and/or shared memory parallelism purposes. Note that by using pre-compiler directives one can generate code for shared only, distributed only and hybrid programming models. Results will be presented here only for the distributed memory model due the characteristics of the hardware employed. However, *ivdep* directives and the necessary mesh coloring were used to improve Itanium-2 performance through an effective use of its pipelines with no data dependencies.

```
iside = 0
DO iblk = 1, nedblk
nvec  = iedblk(iblk)

!dir$ ivdep
!$OMP PARALLEL DO
   DO i = iside+1, iside+nvec, 1
      ...MATVEC computations...
   ENDDO
!$OMP END PARALLEL DO

iside = iside + nvec
ENDDO

...over interface nodes...
#ifdef MPICODE
call MPI_AllReduce
#endif
```

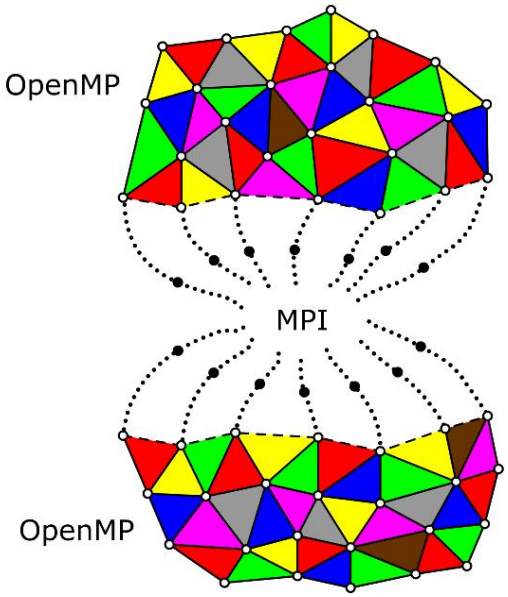


Figure 2 – Matrix-vector product for distributed (Metis partitioning) and shared parallelisms (mesh coloring).

5 RESULTS

In the following sections two benchmark problems are shown to evaluate the performance improvements, due the edge-based data structure, and accuracy of our implementation. The first problem is the well known lid driven cavity problem extended for the three dimensional case and the second complies a hypothetical laminar flow through the body of a Le Mans race car.

All computations were made on a SGI Altix 350 system (Intel Itanium-2 CPUs with 1.5 GHz and 24 Gb of NUMA flex memory). The system run Linux and Intel Fortran compiler 8.1. No optimizations further those provided by standard compiler flags (-O3) were made.

For all tests the numerical procedure considered a fully coupled **u**-*p* version of the stabilized formulation using linear tetrahedron elements. The parallel solver is composed by an outer inexact-Newton loop and an inner nodal block diagonal preconditioned GMRES linear solver.

5.1 Three dimensional leaky lid-driven cavity flow

In this well known problem the fluid confined in a cubic cavity is driven by the motion of a leaky lid. Boundary conditions consist in a unitary velocity specified along the entire top surface and zero velocity on the other surfaces. The model built with 5,151,505 tetrahedral elements; 6,273,918 edges and 1,061,208 nodes summarizing 4,101,106 equations is shown in Figure 3.

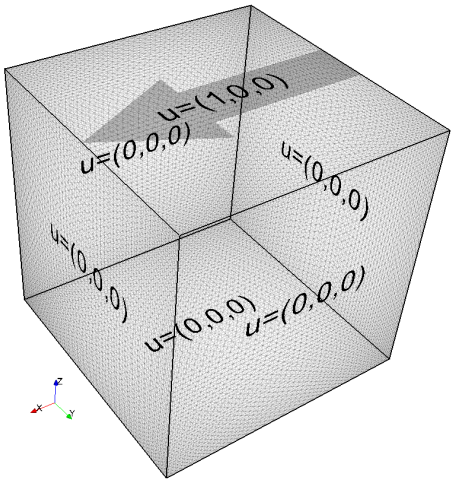


Figure 3 – Flow in a leaky lid-driven cavity. Geometry and boundary conditions.

Figure 4 show the computed vertical and horizontal velocities at the centerline for Reynolds 1000 (left) and Reynolds 2000 (right). The results are compared with those recently presented by Lo et al. (2005). We may observe that all results are in good agreement.

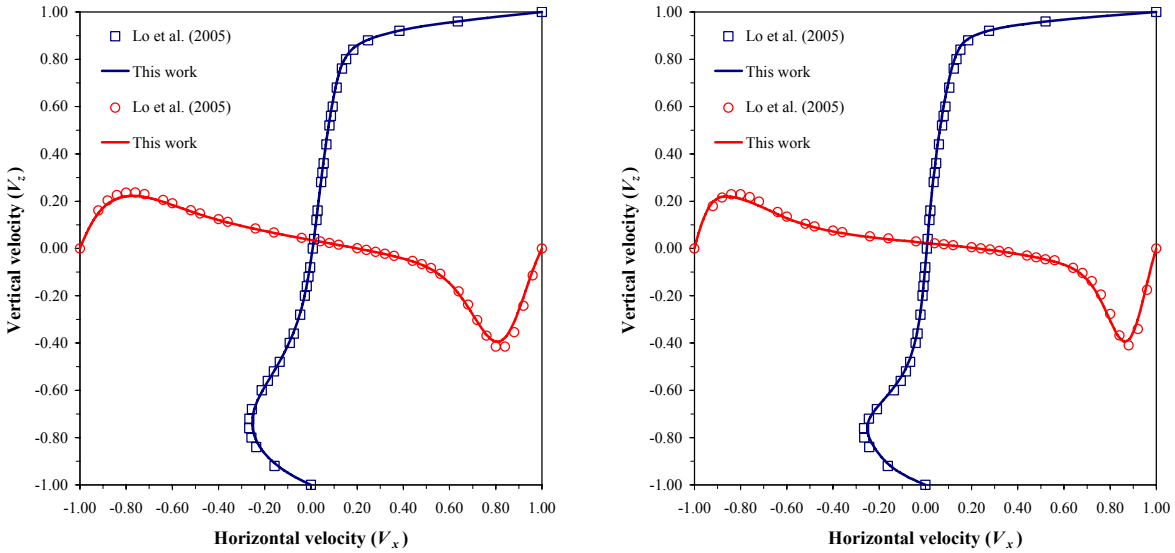


Figure 4 – Three dimensional lid driven cavity flow validation (left) Re 1000 (right) Re 2000

In Figure 5 we may observe the streamlines (left) for the case of Re=1000 and the corresponding pressure and velocity fields (right) at the cavity midplane. Due to the particular boundary conditions this problem presents strongly three-dimensional features.

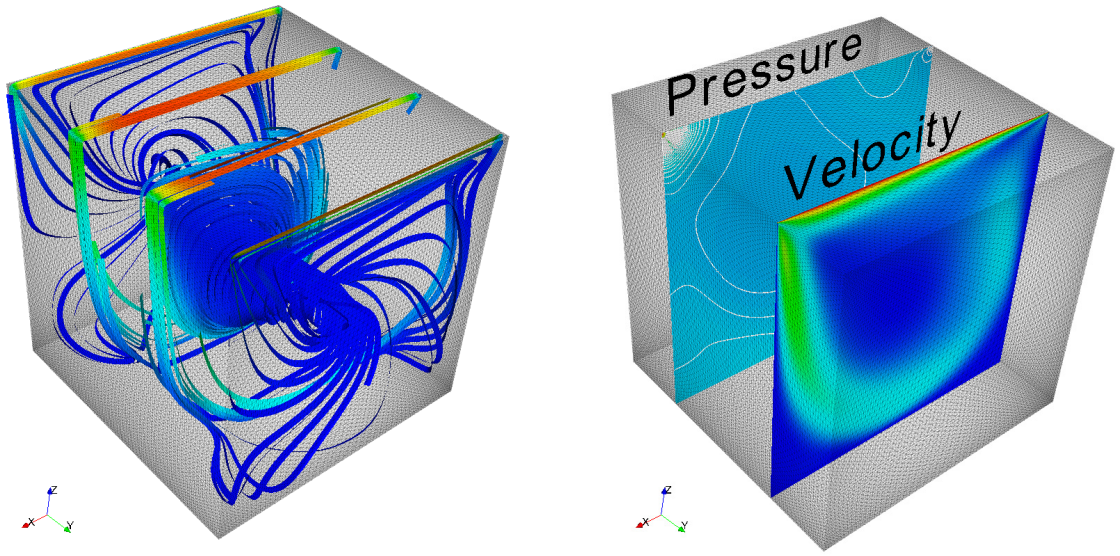


Figure 5 – Streamlines (left), pressure and velocity at the midplane for Re=1000

In Figure 6 we show the wall clock times for the EBE and EDE parallel solutions. Note that the EDE solutions are always faster than the EBE, but more pronounced for few processors. The EDE solution with one processor is faster than the EBE solution with two processors. However, we observed gains for the EDE solution until 16 processors, while for the EBE wall clock times diminish until 32 processors.

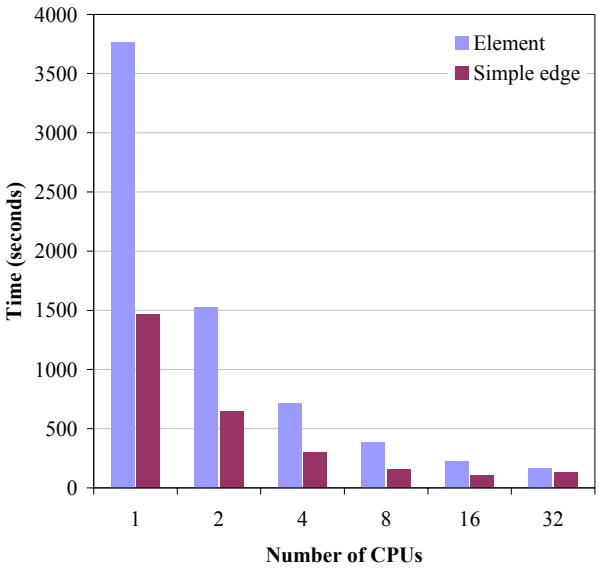


Figure 6 – Wall time comparison for element-by-element (EBE) and edge-by-edge (EDE) data structures

5.2 Laminar flow through a Le Mans race car

This problem consists on a hypothetical three-dimensional simulation of a laminar flow through the body of a Le Mans race car model for robustness and performance evaluation purposes. The model with 10,264,863 elements; 1,858,246 nodes; 12,434,433 edges resulting in 7,720,432 equations is shown in Figure 7. The computations were carried out up to the relative residual ($||\mathbf{r}||/||\mathbf{r}_o||$) and the relative error ($||\Delta\mathbf{u}||/||\mathbf{u}||$) decreased 5 orders of magnitude. Some qualitative results are presented in Figure 8 with the streamlines around the car and pressure contour plotted over the car body.

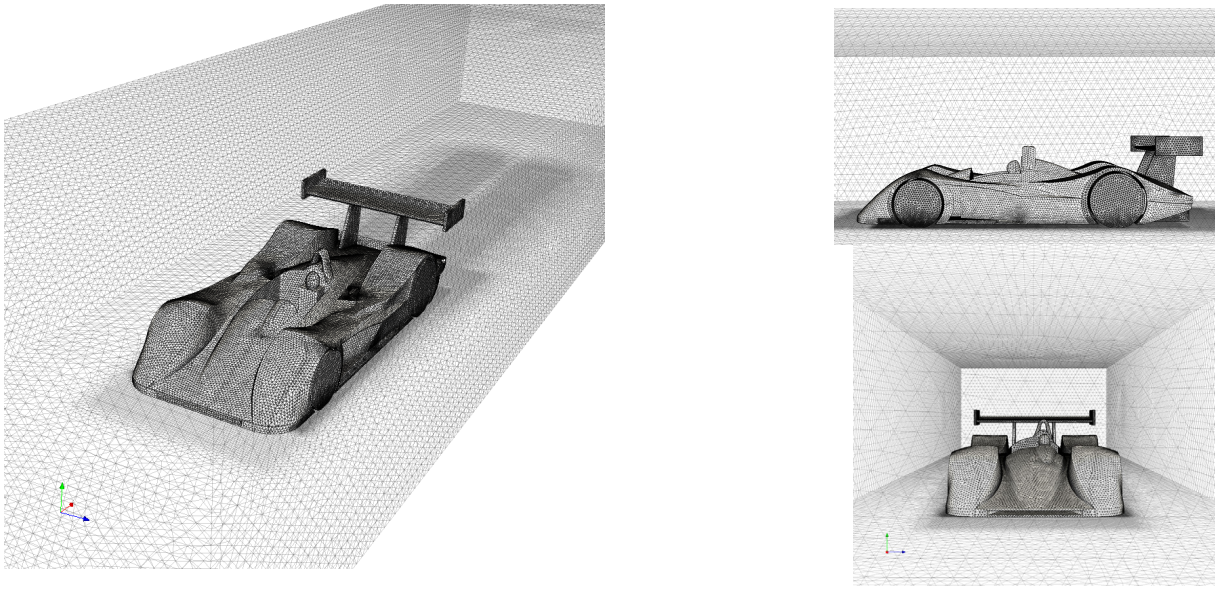


Figure 7 – Le Mans race car model and mesh.

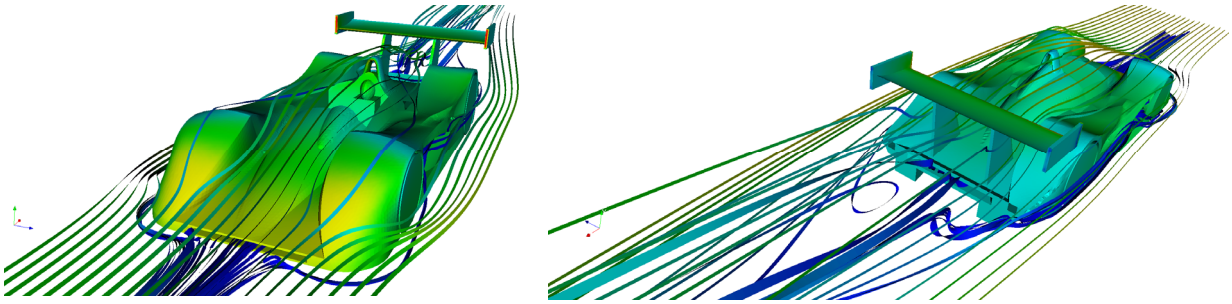


Figure 8 – Results for the laminar flow through the Le Mans race car

Table 2 lists the total memory, the memory per processor and the wall clock times for the EDE solution of this problem with an increasing number of processors. Since the SGI Altix provides us with a single image of the operational system, we were able to run this problem in only one processor even spending more memory than that available for each computing node. For the system employed, each computing node is built with 2 CPUs and 2 GB of memory and SGI refers this node as a C-Brick part of the system. As the number of processors is increased, total memory increases, but the memory per processor diminishes and reaches the memory available locally at each SGI Altix 350 C-Brick In Figure 9 we show the scaled speedups and efficiencies computed according to the Gustafson's law (see Gustafson et al., 1988 for details) and defined by $S_s = n + (1-n)\, s$, where n is the number of processors and s corresponds to the normalized time spent in the serial portion of the program. Theses results show good parallel performance despite the case with 2 processors where the parallel performance has fallen down below of expected.

Table 2. Memory and timings for the Le Mans car problem.

	Total memory Gb	Memory per CPU Gb	Wall time hh:mm:ss
1	7.07	7.07	06:58:34
2	7.10	3.55	05:16:05
4	7.21	1.80	02:00:10
8	7.56	0.95	01:05:41
12	8.04	0.67	00:53:03

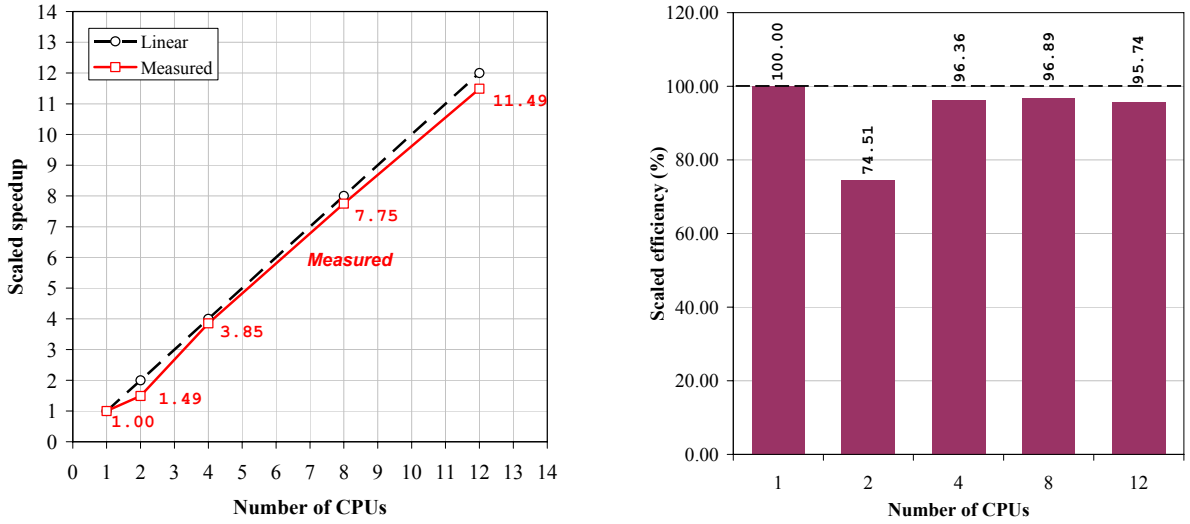


Figure 9 – Parallel performance on SGI Altix 350 (left) Scaled speedup and (right) Scaled parallel efficiency

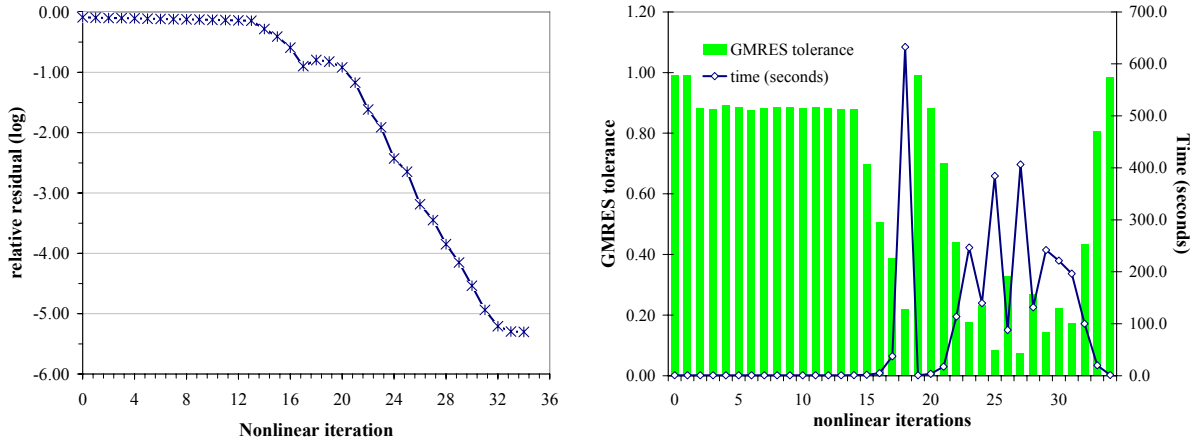


Figure 10 – Inexact nonlinear method behavior (left) Relative nonlinear residual. (right) GMRES tolerance (bars) controlled by inexact nonlinear method and time per nonlinear iteration (lines)

In Figure 10 we may observe the behavior of the inexact nonlinear solution method. The stopping criterion was satisfied after 35 nonlinear iterations and the linear tolerance ranged from 0.99 to 0.075 at nonlinear iteration 27. The first 15 iterations were very fast, and as soon as the relative residual starts to drop, the linear tolerance is tightened and oscillates until convergence.

6 CONCLUSIONS

We have tested the performance of an edge-based inexact Newton-type algorithm to solve nonlinear systems of equations arising from the SUPG/PSPG finite element formulation of steady incompressible flows. Numerical tests in two 3D problems, the lid driven cavity flow at Reynolds numbers 1000 and 2000 and the flow around a Le Mans car, have shown the code good parallel performance in the SGI Altix. We have observed that the inexact methods adapt the inner Krylov space iterative method tolerance to follow the solution progress towards the solution. The code supports shared, distributed and hybrid programming models with standard

libraries and interfaces and thanks to the edge-based data structures, we have been able to solve complex flow problems involving 12 million of tetrahedra in moderate sized machines.

Acknowledgements

The authors would like to thank the financial support of the Petroleum National Agency (ANP, Brazil) and MCT/CNPq, The Brazilian Council for Scientific Research. The Center for Parallel Computations (NACAD) at the Federal University of Rio de Janeiro and SGI Brazil provided the computational resources for this research.

REFERENCES

Catabriga L and Coutinho ALGA. , 2002. Implicit SUPG solution of Euler equations using edge-based data structures. *Computer Methods in Applied Mechanics and Engineering*, 32:3477-3490.

Coutinho ALGA, Alves JLD and Ebecken NFF, 1991. A study of implementation schemes for vectorized sparse EBE matrix-vector multiplication. *Advances in Engineering Software and Workstations*, 13(3):130-134.

Coutinho ALGA, Martins MAD, Alves JLD, Landau L and Moraes A, 2001. Edge-based finite element techniques for non-linear solid mechanics problems. *Int. J. for Num. Meth. in Engrg*, 50(9):2053-2068

Coutinho ALGA, Martins MAD, Sydenstricker R and Elias RN. Performance comparison of data reordering algorithms for sparse matrix-vector multiplication in edge-based unstructured grid computations, submitted__

Dembo, RS, Eisenstat, SC and Steihaug, 1982 T. Inexact Newton methods, *SIAM J. Numer. Anal.*, 19: 400-408.

Eisenstat, SC and Walker, H F, 1996. Choosing the forcing terms in inexact Newton method. *SIAM. J. Sci. Comput.*, 17(1): 16–32.

Elias, R. N., Coutinho, 2003, Inexact Newton-type methods for non-linear problems arising from the supg/pspg solution of steady incompressible Navier-Stokes equations. *J. of the Braz. Soc. of Mech. Sci. & Eng.*, 26(3): 330-339.

Ferencz RM and Hughes TJR, 1998. Iterative finite element solutions in solid mechanics. *Handbook for Numerical Analysis,* Vol. VI, eds, P.G. Ciarlet and J.L. Lions, Elsevier Science.

Gijzen MB, 1995. Large scale finite element computations with GMRES-like methods on a CRAY Y-MP. *Applied Numerical Mathematics*, 19:51-62.

Gustafson, J. L., Montry, G. R. and Benner, R. E., 1988. Development of Parallel Methods for a 1024-Processor Hypercube, *SIAM J. on Sci. and Stat. Comp.*, 9(4):609-638

Hughes TJR, 1987. The finite element method: linear static and dynamic finite element analysis. *Prentice-Hall.* NJ.

Karypis G. and Kumar V., 1998, Metis 4.0: Unstructured Graph Partitioning and Sparse Matrix Ordering System. *Technical report*, Department of Computer Science, University of Minnesota, Minneapolis;
(http://www.users.cs.umn.edu/~karypis/metis).

Lo, DC, Murugesan, K and Young, DL, 2005. Numerical solution of three-dimensional velocity-vorticity Navier-Stokes equations by finite difference method. *Int. J. Numer. Meth. Fluids*, 47:1469-1487.

Lohner R, 1994. Edges, stars, superedges and chains. *Computer Methods in Applied Mechanics and Engineering*, 111(3-4): 255-263.

Löhner, R., Galle, 2002. M. Minimization of indirect addressing for edge-based field solvers. *Communications in Numerical Methods in Engineering*, 18: 335-343.

Luo H, Baum JD, Löhner R, 1994. Edge-based finite element scheme for the Euler equations, *AIAA Journal*, 32(6):1183-1190.

Lyra PRM, Manzari MT, Morgan K, Hassan O and Peraire J, 1995. Side-based unstructured grid algorithms for compressible viscous flow computations. *International Journal for Engineering Analysis and Design*. 2:88-102.

Peraire J, Peiro J, Morgan K, 1993. Multigrid solution of the 3d-compressible Euler equations on unstructured grids. *Int. J. Num. Meth. Engrg.*. 36(6): 1029-1044.

Ribeiro FLB and Coutinho ALGA, 2005. Comparison between element, edge and compressed storage schemes for iterative solutions in finite element analyses. *Int. J. Num. Meth. Engrg.* 63(4): 569-588.

Ribeiro FLB, Galeão AC and Landau L, 2001. Edge-based finite element method for shallow-water equations. *Int. J. for Num. Meth. in Fluids*, 36: 659-685.

Saad Y, 1996. Iterative methods for sparse linear systems. *PWS Publishing Company*: Boston.

Soto O, Löhner R, Cebral JR and Camelli F, 2004. A stabilized edge-based implicit incompressible flow formulation. *Computer Methods in Applied Mechanics and Engineering*, 193: 2139-2154.

Tezduyar, TE, 1991. Stabilized finite element formulations for incompressible flow computations, *Advances in Applied Mechanics*, 28: 1-44.

Tezduyar, TE, Mittal S, Ray SE and Shin R, 1992. Incompressible flow computations with stabilized bilinear and linear equal-order-interpolation velocity-pressure elements. *Comput. Methods Appl. Mech. Engrg.*, 95: 221-242.