LARGE SPARSE MATRIX REORDERING SCHEMES APPLIED TO A FINITE ELEMENT ANALYSIS

Brenno Lugon, Lucia Catabriga
Universidade Federal do Espírito Santo - Brazil
Corresponding Author: **Lucia Catabriga**

brennolugon@gmail.com,luciac@inf.ufes.br

1 Introduction

The finite element method is one of the most used numerical techniques for finding approximated solutions of partial differential equations. Generally, the finite element formulations may require the solution of linear systems of equations involving millions of unknowns that are usually solved by Krylov space iterative update techniques, from which the most used is the Generalized Minimal Residual method (GMRES). In this work, we analyze reordering strategies applied to an incomplete **LU** preconditioner for the resulting system of a SUPG finite element formulation considering two free softwares for unstructured mesh generation. The first one, EASYMESH (`web.mit.edu/easymesh_v1.4/www/easymesh.html`), perform a renumeration of nodes in order to decrease the matrix bandwidth while the second one, GMSH (`geuz.org/gmsh/`), consider a naive numeration.

2 ILU(p) preconditioner and reordering schemes

The preconditioner basic idea is to replace the given system $Ax = b$ by the system $M^{-1}Ax = M^{-1}b$, where M is a "suitable" approximation to A such that $M^{-1}A$ is well conditioned. From a practical point of view, preconditioner operations should be memory-efficient and require few arithmetic operations. Hence a sparse incomplete factorization strategy in the form of $M = \bar{L}\bar{U}$, where $\bar{L}$ and $\bar{U}$ are the incomplete LU factors might be an appropriate candidate. Incomplete factorization algorithms are sensitive to the ordering of equations, therefore the efficiency of elimination and consequently the construction of an incomplete factorization applied to a preconditioned iterative solver will be influenced by the numeration of the nodal unknowns [1].

There are several approaches to reordering nodal unknowns for efficient linear system solution. These reorderings correspond to row and column matrix interchanges. The goal is to reduce the matrix bandwidth, i.e., reduce the maximum of distances between the first nonzero element of a row i and the main diagonal. This is NP-complete problem and several algorithms exist that are generally able to finding a relatively good solution in a reasonable amount of time. It is well known that reordering techniques are efficient choices when we use incomplete factorization algorithms [2]. However, we want to show how advantageous these reorderings can be in final CPU time when we examine softwares for mesh generation with distinct characteristics for nodal numeration.

3 Test problem

For the experiments, we consider a benchmark problem described by the following convection-diffusion equation:

$$\boldsymbol{\beta}.\nabla u - \nabla.(\boldsymbol{\kappa}\nabla u) = f, \quad \text{in } \Omega = [0,1] \times [0,1] \tag{1}$$

where u represents the quantity being transported (e.g. temperature, concentration). The problem described a pure convection of a scalar on the domain Ω, where the diffusivity is given by $\boldsymbol{\kappa} = 1 \times 10^{-6} I$, the flow direction is 45° from the x-axis, $\|\boldsymbol{\beta}\| = 1$ and the function $f = 0$. Figure 1 shows the problem set up and the boundary conditions and Figure 2 shows the approximated solution.

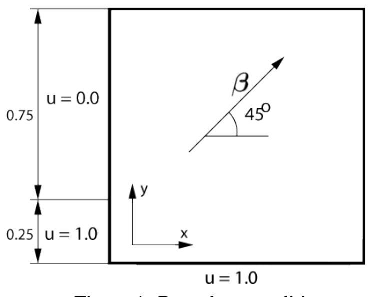

Figure 1: Boundary conditions of the related problem.

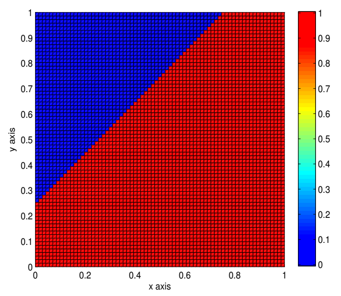

Figure 2: Approximated solution for a 10.000 mesh.

4 Experimental results

All programs were developed by the author in C language and the tests were run on an Intel Core i5 2.53GHz processor with 4GB of RAM, under Ubuntu 12.10. To store the resulting matrix we use the well-known CSR optimized storage scheme. Figures 3 and 4 show the sparsity pattern of resulting matrices derived from GMSH and EASYMESH.

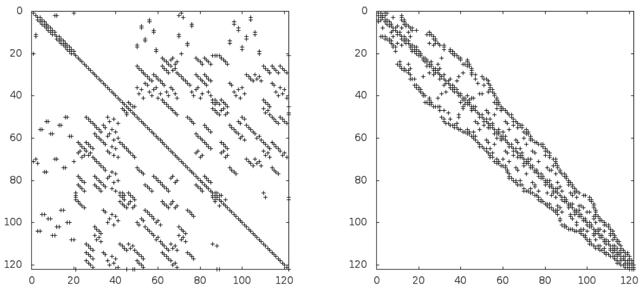
Figure 3: Sparsity pattern of resulting matrices by GMSH
Left: without reordering **Right**: reordering with RCM

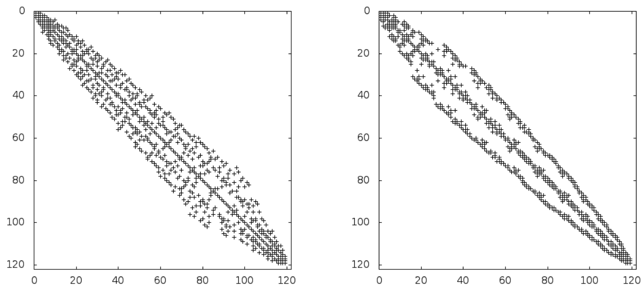
Figure 4: Sparsity pattern of resulting matrices by EASYMESH
Left: without reordering **Right**: reordering with RCM

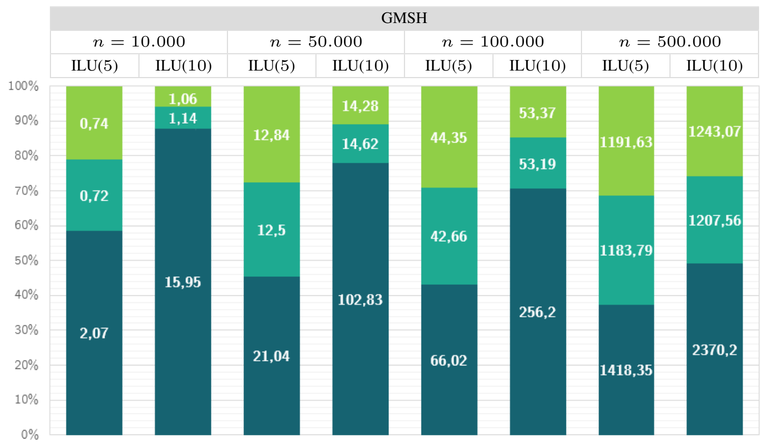

Figure 5: CPU time (seconds) to solve GMSH resulting matrices with no reordering ■, reordering with RCM ■ and with Sloan ■, for different ILU preconditioners.

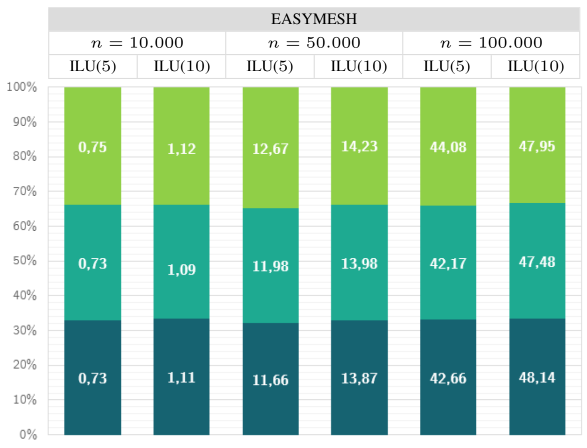

Figure 6: Same as Figure 4, but now considering EASYMESH.

	$n = 10.000$	$n = 50.000$	$n = 100.000$	$n = 500.000$
GMSH	0,65	4,43	10,58	65,23
EASYMESH	47,61	2.813,56	11.266,33	more than 48h

Table 1: CPU time (seconds) to generate meshes with GMSH and EASYMESH for different sizes.

5 Analysis and conclusions

Two reordering algorithms, Reverse Cuthill-McKee (RCM) and Sloan, were used for tests as well as two levels of fill-in for ILU(p) preconditioner. Figure 5 shows us that, for all cases, reordering strategies cause significant gain of CPU time for matrices derived from GMSH. On the other hand, for matrices derived from EASYMESH in Figure 6, the reordering does not cause any positive impact. This fact has to do with the nodal unknowns numeration, i.e., the matrix sparsity pattern, that may cause substantial fill-in during the preconditioner calculations and raise the number of arithmetic operations. Regarding the reordering algorithms, RCM and Sloan had comparative results. If we examine the ILU preconditioner effectiveness in GMSH matrices, we can see in Figure 5 that, as the p parameter increases, higher is the impact of the reordering in the final CPU time. Now, with respect to the mesh generators, Table 1 shows that the computational time to generate meshes were quite different for each software. Meshes generated by EASYMESH demanded much more CPU time than meshes generated by GMSH, which was expected since GMSH does not perform any kind of optimization for numeration nodal unknowns.

We can conclude that it is not necessary to use a reordering if you choose to use a mesh generator that apply an optimization (renumeration of nodes). However, we notice that it is much more advantageous to decide for a software that does not implement any optimization during the mesh generation and make use of reordering strategies during the solution process, since the CPU time to generate an optimized mesh, as does EASYMESH, can be incredibly high.

Acknowledgement

This work was partly supported by FAPES n^o 48511579/2009, CNPq 552630/2011-0 and CNPq 307020/2012-6.

References

[1] J. Camata, A. Rossa, A. Valli, L. Catabriga, G. Carey, and A. Coutinho, “Reordering and incomplete preconditioning in serial and parallel adaptive mesh refinement and coarsening flow solutions,” *International Journal for Numerical Methods in Fluids*, vol. 69, no. 4, pp. 802–823, 2012.

[2] M. Benzi, “Preconditioning techniques for large linear systems: A survey,” *J. Comput. Phys.*, vol. 182, pp. 418–477, Nov. 2002.