SëMA Journal
n°57(2012), 107–134

A SURVEY ON DIRECT SOLVERS FOR GALERKIN METHODS

DAVID PARDO[A,B], MACIEJ PASZYNSKI[C], NATHAN COLLIER[D],
JULEN ALVAREZ[A], LISANDRO DALCIN[E], AND VICTOR M. CALO[D]

[a] Department of Applied Mathematics, Statistics, and Operational Research, University of the Basque Country (UPV/EHU), Bilbao, Spain
[b] Ikerbasque, Bilbao, Spain
[c] AGH University of Science and Technology, Krakow, Poland
[d] King Abdullah University of Science and Technology (KAUST), Saudi Arabia
[e] CONICET, Santa Fe, Argentina

Corresponding author e-mail: dzubiaur@gmail.com (David Pardo)

Abstract

In this paper we describe the history, performance, and design concepts of direct solvers for algebraic systems resulting from Galerkin discretizations of partial differential equations. Popular direct solver implementations of Gaussian elimination (also known as LU factorization) are introduced and briefly analyzed. We discuss three of the most relevant aspects influencing the performance of direct solvers on this kind of algebraic systems. First, the ordering of the degrees of freedom of the algebraic system has a significant impact on the solver performance, solution speed and memory requirements. The impact of unknowns ordering for elimination is exemplified and alternative ordering algorithms are described and compared. Second, the effect of round-off error on the simulation results is discussed. We detail this effect for uniform grids where the impact of round-off error on the solution is controlled by the condition number of the matrix in terms of the element size, but is independent of the polynomial order of approximation. Additionally, we discuss the link between unknown ordering and round-off error. Third, we describe the impact of the connectivity pattern (graph) of the basis functions on the performance of direct solvers. Variations in the connectivity structure of the resulting discrete system have severe impact on performance of the solver. That is, the resources needed to factorize the system strongly depend on its connectivity graph. Less connected graphs are cheaper to solve, that is, C^0 finite element discretizations are cheaper to solve with direct solvers than C^{p-1} discretizations.

Key words: *LU factorization, Gaussian elimination, frontal solver, multi-frontal solver, hp-finite elements, isogeometric analysis, cost of regularity, parallel direct solvers*

AMS subject classifications: *35-02, 65-02, 65F05, 65F50*

Received: June 22, 2011. Accepted: November 15, 2011.

1 Introduction

Numerical simulation is fundamental to many technological and scientific endeavors. These applications affect our daily life and our understanding of our world. At the core of these scientific applications lies the efficient solution of linear systems of equations generated by the discretization technique adopted. Over the last twenty years, a revolution in Simulation-Based Engineering Science has occurred [71]. Simulations based on mathematical models that more faithfully account for the details of reality while using high-accuracy discretization methods are the norm presently (c.f., [63, 94]). Faster algorithms and faster hardware have made this revolution possible. Thus, simulation is increasingly important in medicine (drug delivery [18, 52, 62], stent design [6], tumor detection [37, 38], cardiovascular disease [9, 11]), environmental hazard assessment (earthquake and tsunami early warning systems, severe storm forecasting [69], climate change analysis [70]), energy (oil recovery and prospection, alternative energies such as wind turbine design [12, 54] and structural optimization [29, 30, 42]), and transportation, one of the biggest energy expenditures of humanity (car, train, and airplanes design [43]). Most of these simulations rely on the efficient solution of one or several large systems of linear equations. Additionally, quantitative assessment of the uncertainty induced to the simulation results by uncertainty in the data, error in the theoretical models as well as numerical error in the representation imply that these large systems of equations need to be repeatedly solved. For example, uncertainty in the simulation predictions can be quantified using Monte Carlo simulations, which require a complete simulation for every materialization of the model parameters [68].

In this paper, we focus on the solution of linear systems arising from the use of Galerkin methods such as Finite Elements (FE) [17, 22, 31, 32, 56], Discontinuous-Galerkin (DG) [23], Petrov-Galerkin (PG) [59], and Isogeometric Analysis (IGA) [27]. Galerkin methods are widely employed in the mathematical modeling of a variety of industrial applications. For example, FE methods are routinely used for: (a) the design of civil engineering structures such as bridges, skyscrapers, and other large buildings by solving, for example, elasticity equations, (b) prospection of oil, gas, and other precious materials in the Earth's subsurface using electromagnetic, acoustic, and nuclear measurements, (c) design of nanoparticles and meta-materials, (d) design of sonars, radars, and other detection systems, (e) weather simulation and other fluid-flow problems by solving Navier-Stokes equations, (f) image segmentation by solving convection dominated diffusion equations, and so on. Other Galerkin methods such as DG, PG, and IGA are also employed to solve either of the above applications, and thousands of articles are published monthly on different applications of Galerkin methods. The main difference between these Galerkin methods is the choice of basis functions, which has far-reaching consequences: While DG utilizes discontinuous basis functions, FE methods employ C^0-continuous functions and IGA makes use of basis functions with high regularity (C^k with $k \geq 0$). PG employs different trial and test basis function spaces to achieve stability and accuracy.

Although the focus is on linear systems arising from Galerkin discretizations, we emphasize that most of the content presented in this paper is extensible to other numerical methods such as finite differences (FD) and finite volumes (FV).

There exist two classes of methods for solving a linear system of equations: (a) direct methods, which deliver the exact solution in one step, and (b) iterative methods, which provide an approximate solution of prescribed quality by following an iterative process.

While the use of iterative solvers typically requires less computational resources (time and memory) than direct solvers, they suffer from a number of problems. First, iterative solvers often present severe convergence problems. Thus, different solvers are needed for each application (elasticity [36], electromagnetism [50], fluid dynamics [5]) and numerical methods (h-FEM [16], p-FEM [7], hp-FEM [75, 76], DG [46], PG, IGA [13, 20], etc.). Second, in addition to the convergence problems, iterative solvers may be slower than direct solvers when a problem with multiple right-hand-side needs to be solved, as it occurs in the case of gradient-based inverse methods in order to compute the Jacobian and Hessian matrices. Iterative solvers may also be slower than direct solvers when several matrices with a common set of rows and columns need to be solved, as it occurs in mesh-based methods when local grid-refinements are performed. Moreover, direct solvers are a main building block of most iterative solvers. Thus, direct solvers become essential in many applications. Thus, here we focus on the history, performance, and design of direct solvers.

There exist several direct methods for the solution of a linear system of equations, including LU factorization, QR factorization, and singular value decomposition [45]. The fastest method is LU factorization, also known as Gaussian elimination, which is by far the most used algorithm for the direct solution of a system of linear equations. While other methods such as QR factorization may offer added stability minimizing the effect of round-off error, they are simply non-competitive in terms of computational efficiency. The main principle of the LU factorization algorithm is to decompose the original matrix A into the product of a lower triangular matrix L with an upper triangular matrix U.

There exist several variations of the LU factorization algorithm. For the case of a symmetric matrix A, then U becomes L^T, and LU factorization is called *Choleski* factorization [45]. Additional alternatives are left-looking, right-looking, and Crout variants [33], which may be convenient for particular applications mainly in context of parallel computations. For the case of sparse matrices, it is also important to avoid operations with the zeros of the matrix, and to produce L and U factors that are as sparse as possible. State-of-the-art implementations of the LU factorization algorithm for sparse matrices include the frontal [34, 58] and multi-frontal solvers [35, 39]. The latest trends on this area include efficient parallelization techniques (see e.g., [3, 48, 64]) and application-specific implementations that take advantage of the data-structures of the Galerkin method, such as the work of [14, 61, 79–82]. In the literature there is a number of review papers about direct multi-frontal solvers (*e.g.*, [65]), while detailed performance comparison of the most prominent solver implementations can be found in [47].

The main approach of all existing papers, books, and reviews has been to design the best possible direct solver of linear equations for a given discretization. In here, in addition to provide a comprehensive review of direct solvers for non-expert readers, we present a unique approach that significantly differs from previous works: we analyze how the best existing direct solvers perform when applied to different

Galerkin discretizations. Thus, the work presented herein is an introduction and review of direct solvers as well as a guide to select the *optimal* Galerkin method (discretization) from the point of view of the direct solver performance. This analysis complements the existing one on different Galerkin methods, where they typically display only the discretization error versus problem size [1, 28]. The analysis of problem size versus computational resources described in this paper complements the existing results and provides a complete picture of the advantages and disadvantages of each Galerkin method (discretization). This description follows closely our work on the area [19, 24, 25]

In this review paper, we discuss the main advantages and disadvantages of most interesting variations of the LU solver when applied to systems arising from Galerkin discretizations. We describe in detail the main ingredient to achieve good performance, namely, correct ordering of the unknowns. We also provide a comparison of the performance of some of the most efficient direct solver implementations when applied to systems arising from different Galerkin discretizations. From this analysis, we conclude that some discretizations deliver matrices that are *easier to solve*.

Before closing this introduction we briefly describe a family of solvers introduced over the last ten years. This family of solvers based on the idea of H-matrices [49] and other compression methods has been proposed by several researchers (*e.g.*, [87, 88, 98]). Some authors referred to them as "direct solvers" or "fast direct solvers". Clearly, they cannot be classified as iterative solvers, since they provide a solution in one step (no iteration occurs). However, and in contrast to traditional direct solvers, this new family only provides an approximate solution. Furthermore, their accuracy depends upon the discretization and equation to be solved, and different compression strategies need to be designed for each equation. Thus, we believe a more adequate name perhaps could be "one step solvers" or "approximate direct solvers". In any case, we shall not consider them in this paper, in the same way we do not consider iterative solvers that converge in one step.

The structure of the paper is as follows. We first describe the frontal and multi-frontal solvers, including the most popular commercial and non-commercial direct solver implementations. Then, we analyze in detail the issue of ordering of the unknowns, which is at the core of obtaining superior performance. At this point, we review the latest trends on the newest mesh-based ordering algorithms. Since ordering also affects to the round-off error delivered by the solver, this issue is illustrated with a one-dimensional example. Then, we study the performance of two direct solvers when applied to different Galerkin methods. Following, we provide a short discussion on parallel direct solvers, we describe a number of real-world applications to motivate the open issues and future challenges on the topic, and we state some conclusions.

2 Frontal and Multi-frontal solvers

We begin with a short introduction of existing algorithms for performing LU factorization on sparse matrices. The most typical state-of-the-art algorithms are the frontal and multi-frontal solvers. The frontal solver [34, 58] browses blocks of unknowns (typically finite element contributions), one-by-one, and when possible it

performs Gaussian elimination of fully assembled unknowns from the single front matrix. The multi-frontal solver [35, 39] constructs the assembly tree based on the analysis of the connectivity (graph) of the matrix, that is, the topology of the computational mesh. Then, finite elements are joint into pairs and unknowns are eliminated within frontal matrices associated to multiple branches of the elimination tree. The process is repeated until the root of the assembly tree is reached. Finally, the common interface problem is solved and partial backward substitutions are recursively performed on the assembly tree.

The process of elimination of frontal and multi-frontal solvers is performed via the Schur complement operation. Let the square matrix A be decomposed into a block structure as:

$$A = \begin{bmatrix} \mathrm{B} & \mathrm{C} \\ \mathrm{D} & \mathrm{E} \end{bmatrix}, \tag{1}$$

where each block is full (all logical entries may be non-zero). The Schur complement method consists of performing partial LU factorization of the square submatrix B to obtain:

$$A = \begin{bmatrix} \mathrm{I} & 0 \\ \mathrm{DB}^{-1} & \mathrm{I} \end{bmatrix} \begin{bmatrix} \mathrm{B} & 0 \\ 0 & \mathrm{E\text{-}DB}^{-1}\mathrm{C} \end{bmatrix} \cdot \begin{bmatrix} \mathrm{I} & \mathrm{B}^{-1}\mathrm{C} \\ 0 & \mathrm{I} \end{bmatrix}, \tag{2}$$

where the term E-DB^{-1}C is known as the Schur complement.

Some of the most popular and efficient direct solvers of sparse matrices include MUMPS [67], PARDISO [72], and HSL [53]. All of them are designed for both sequential and parallel machines. Other solvers include SuperLU [95], SPOOLES [93] and UMFPACK [97]. A comparison on the performance and features of these solvers is given in, for example, [47]. Recently developed solvers that use the grid data structures to produce connectivity trees and elimination ordering are [14, 82].

3 The issue of ordering

A proper ordering of the unknowns is fundamental to obtain optimal performance from a direct solver. Although a different ordering does not modify the number of nonzero entries of the original matrix A, it strongly affects to the sparsity pattern of both factors L and U. In particular, different orderings of the same matrix may produce factorizations with dramatically different number of nonzero entries (the so-called "fill-in" of L and U). To illustrate this fact, we consider the following example:

$$A = \begin{pmatrix} 9 & 1.5 & 6 & 0.75 & 3 \\ 1.5 & 0.5 & 0 & 0 & 0 \\ 6 & 0 & 12 & 0 & 0 \\ 0.75 & 0 & 0 & 0.625 & 0 \\ 3 & 0 & 0 & 0 & 16 \end{pmatrix}$$

where

$$L = \begin{pmatrix} 3 & & & & \\ 0.5 & 0.5 & & & \\ 2 & -2 & 2 & & \\ 0.25 & -0.25 & -0.5 & 0.5 & \\ 1 & -1 & -2 & -3 & 1 \end{pmatrix}$$

Notice that $A = LL^T$. After proper re-ordering of A, we obtain an equivalent matrix $\tilde{A}$, whose factors $\tilde{L}$ and $\tilde{L}^T$ have a different and more convenient sparsity pattern with less nonzero entries:

$$\tilde{A} = \begin{pmatrix} 16 & 0 & 0 & 0 & 3 \\ 0 & 0.5 & 0 & 0 & 1.5 \\ 0 & 0 & 12 & 0 & 6 \\ 0 & 0 & 0 & 0.625 & 0.75 \\ 3 & 1.5 & 6 & 0.75 & 9 \end{pmatrix}$$

where

$$\tilde{L} = \begin{pmatrix} 4 & & & & \\ 0 & 0.707 & & & \\ 0 & 0 & 3.464 & & \\ 0 & 0 & 0 & 0.79 & \\ 0.75 & 2.121 & 1.732 & 0.949 & 0.194 \end{pmatrix}$$

In computations relevant to engineering science, a proper ordering of the unknowns may affect the performance of the direct solver in terms of time and memory by at least an order of magnitude. An example with just 100.000 unknowns for a 3D Laplace equation problem is described in Table 1.

SOLVER	MUMPS	MUMPS	MUMPS	PARDISO
ORDERING	N. O.	AMD	METIS	METIS
Time(sec)	686	332	73	73
Number of nonzero entries (millions)	488	181	95	95

Table 1: Time and number of nonzero entries in the factors L and U needed to solve the Laplace problem over a unit cube using a lowest order finite element (FE) method with 100.000 unknowns. (A detailed description of the problem setup is given in [24].) Different columns correspond to different solvers (MUMPS and PARDISO) and ordering algorithms: Natural ordering given by the FE software (N.O.), approximate minimum degree (AMD), and METIS.

We can divide the existing ordering algorithms into two groups. The first group of algorithms is based on the following idea. Once the matrix has been constructed, it is possible to create a graph with its connectivities, that is, for each unknown, one can easily determine the remaining unknowns interacting with it by just looking at

the nonzero entries of a particular row/column. After the connectivity graph has been built, the objective is to solve a global optimization problem that consists of selecting the ordering that minimizes the fill-in of matrices L and U. Approximate solutions to this problem give rise to different ordering algorithms. The most popular ordering algorithm is *nested dissection* [40], which is based on recursive graph partitioning. State-of-the-art implementations of nested dissection are available in METIS [66] and SCOTCH [90]. There also exist several other algorithms such as the minimum degree (see [41] and references therein) and the approximate minimum degree ordering algorithms AMD [2, 4] and their many variants. The main advantage of these ordering algorithms is that they provide an optimal ordering (up to a constant) for the case of matrices arising from uniform regular grids, as shown in [40, 51]. Thus, the most popular solvers of linear equations employ these ordering methods. Even though their optimality in matrices delivered by general unstructured meshes is unknown, our experience indicates that they perform reasonably well in any grid (we do not observe a major deterioration in performance for practical problems).

A second group of ordering algorithms is based on the idea that the information to build the connectivity graph is present in the numerical method employed for constructing the matrix. Therefore, it should be possible to design a proper grid-based ordering algorithm directly from the data-structures of the numerical method rather than by re-constructing the connectivity graph from the matrix. It is believed by many authors (including the authors of this article) that one should be able to achieve a better performance with this type of ordering algorithms, at least, in the case of high polynomial order methods on unstructured and irregular meshes. For this case, unknowns associated to the interior of an element should be eliminated first (this is called static condensation), since they do not interact with unknowns associated to other elements. Numerical experiments prove that performance of METIS and other general ordering algorithms significantly increases when one hard-codes static condensation [14] in the element assembly process. One disadvantage of grid-based ordering algorithms is that the algorithm is discretization-specific, since it is closely tied to the particular numerical scheme. In the following, we present recent efforts oriented towards designing efficient grid-based ordering algorithms for high-order finite element methods.

3.1 Next generation of solvers: ordering based on the grid

For high-order finite element methods, unknowns can be grouped into *supernodes*, that is, a set of unknowns (nodes) with the same connectivity pattern. Data-structures of any finite-element method easily identify supernodes. For example, all interior unknowns to a given element have the same connectivity structure, and therefore, they constitute a supernode. Similarly, unknowns associated to a single face, edge, or vertex constitute supernodes.

The structure of supernode has two clear advantages. First, it naturally gives rise to LU factorization algorithms in terms of blocks, that can be very efficiently executed by using BLAS3 routines, which are tuned to take advantage of the cache memory of the particular hardware available. Second, they facilitate proper ordering of the unknowns, since they reduce the size of the ordering problem from the total number of unknowns

to the total number of supernodes. This significant dimensional reduction for the optimization problem reduces the cost of the process while increasing the chances of finding the global minimum rather than one of the local minima.

Using finite element data structures, a multi-frontal solver algorithm can be designed in the following way. After static condensation, one assembles pairs of neighboring elements in what we will call a "super-element" in order to perform LU factorization of fully assembled supernodes. Then, one can join a set of two super-elements and continue with the LU factorization, recursively. This algorithm only requires a proper ordering of elements, reducing even more the size of the ordering problem, due to the close link between elements and their respective supernodes. A first attempt in this direction consists of using the natural ordering of elements provided by the finite element software. However, this ordering becomes inefficient when significant refinements around some types of singularities are present [61]. Other promising finite element based orderings are currently being studied by different research groups (e.g., [14, 61, 82]).

While a proper re-ordering algorithm is essential to obtain superior performance, it is also crucial in order to guarantee numerical stability of the solution. Otherwise, round-off errors could jeopardize the quality of the solution. In the next section, we briefly discuss a few basic concepts pertinent to round-off errors associated to direct solvers of linear equations.

4 Round-off errors

To reduce the impact of round-off errors on the solution of the algebraic system, it is necessary to employ some type of pivoting technique. Typically, most solvers employ some variation of partial row or column pivoting. To prevent deterioration of the solver performance (operation counts and memory requirements), one needs to ensure that "local" techniques are used, that is, the pivoting technique should only affect the ordering of unknowns.

A careful study of round-off error in context of Galerkin discretizations is cumbersome, since it is highly dependent upon the discretization method, that is, basis functions, quadrature rule, refinement method and so on. To simplify this analysis, keeping it tractable while showing the relevant features of the problem, we restrict our attention to a one-dimensional model problem. We choose to solve the Laplace equation on a uniform grid where all elements in the mesh have the same length (diameter) and polynomial order. The simulation results are shown in Figure 1. This figure plots the relative error of the approximation for $p = 1, 2, 3, 4$ versus the number of elements used in the uniform mesh. Note that the abscissa spans from several dozen to a million elements. The straight dashes line shown that the relative error grows proportional to the number of elements squared. That is, the approximation error is proportional to the matrix's condition number (ratio of largest to smallest eigenvalue which is proportional to the inverse of the mesh size squared), but it is almost insensitive to the polynomial order of approximation used. However, in the case of non-uniform grids, this conclusion is known to be false (the error is not proportional simply to the condition number), and a complete analysis becomes

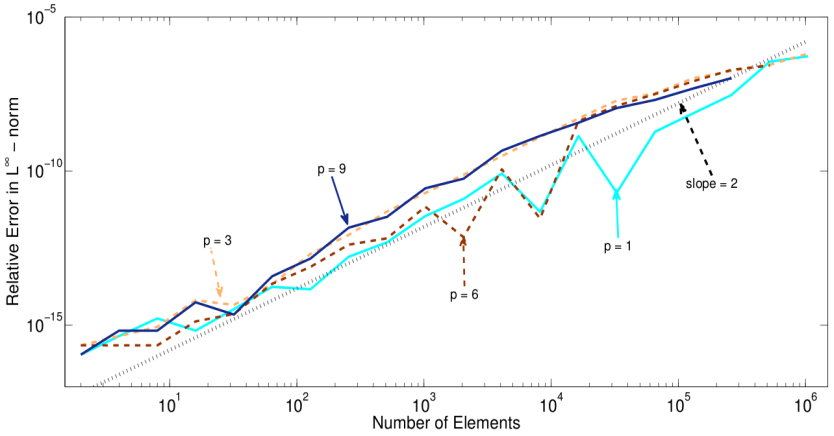


Figure 1: Round-off error for 1D Laplace problem discretized with uniform grids using Peano shape functions and MUMPS solver with PORD [89] ordering.

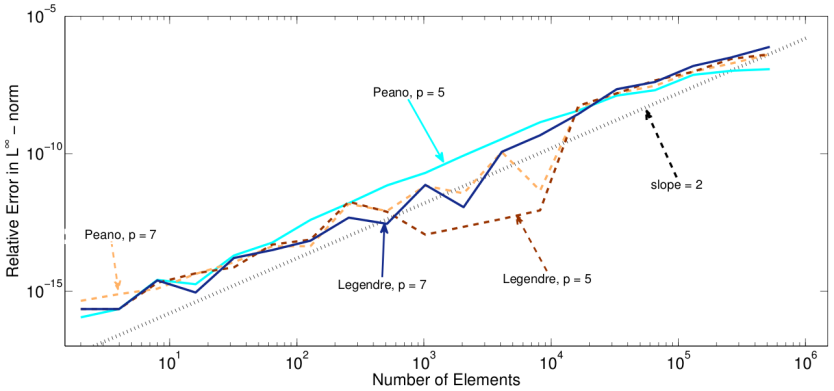


Figure 2: Round-off error for 1D Laplace problem discretized with uniform grids using MUMPS solver with PORD ordering.

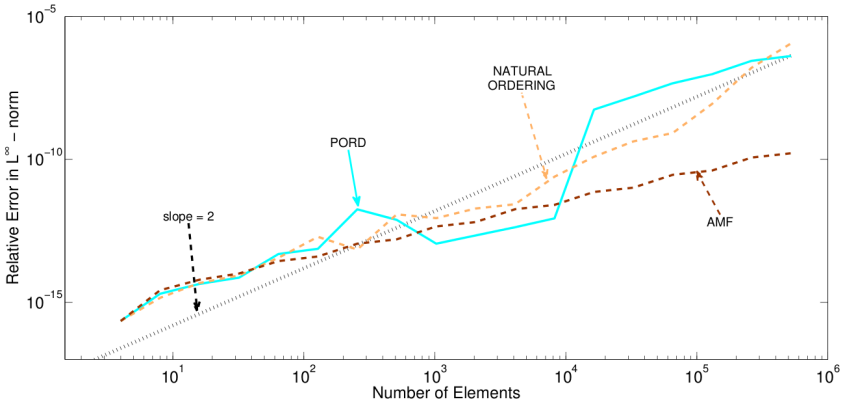


Figure 3: Round-off error for 1D Laplace problem discretized with uniform grids using integrated Legendre polynomial shape functions of order $p = 5$ and MUMPS solver with different ordering techniques.

challenging.

Furthermore, the choice of basis functions could also affect significantly to the condition number of the matrix [8, 21, 26, 55, 73], and therefore, one could also expect an increase on the round-off error of direct solvers. However, in the 1D case, we observe no significant influence of the choice of shape functions in the round-off error. As shown in Figure 2, results using Peano [60, 83, 84] shape functions are similar to those using integrated Legendre polynomials [96].

Nevertheless, when designing a solver one should be aware of the round-off error, since otherwise the solver may provide inaccurate results, specially in the case of some indefinite problems (with both positive and negative eigenvalues) such as wave propagation problems. In particular, at near resonant frequencies, a proper pivoting strategy (reordering of the unknowns) may be needed to control the round-off error, since different pivoting techniques affect to the performance of the solver.

Figure 3 illustrates how different ordering techniques may significantly affect round-off error. In particular, the AMF ordering minimizes this error for the considered 1D Laplace problem. Thus, while the most important aspect to control round-off error is the condition number of the original matrix (in terms of element size h), the reordering of the unknowns is also relevant.

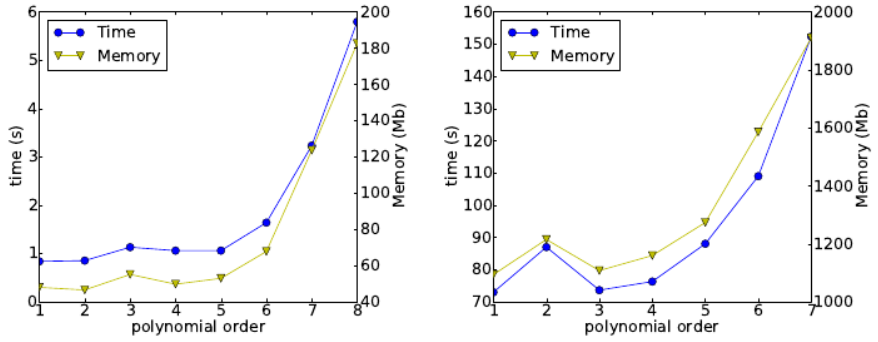
5 Performance of direct solvers when applied to different Galerkin methods

The amount of computational resources employed by a direct solver of linear equations is highly dependant upon the particular choice of the discrete Galerkin method. This choice has a significant effect on the structure (connectivity, condition number) of the linear system of equations. In this section we analyze the direct solver costs associated to some of the most popular Galerkin discretization schemes. All numerical examples correspond to solving the Laplace equation over the unit cube. A detailed description of the equation and boundary conditions used is given in [24].

For traditional C^0-continuous FEM, the performance of the solver is affected by the order of approximation within each element. For 3D problems, it is know that the number of floating point operations and amount of memory scale as [24]:

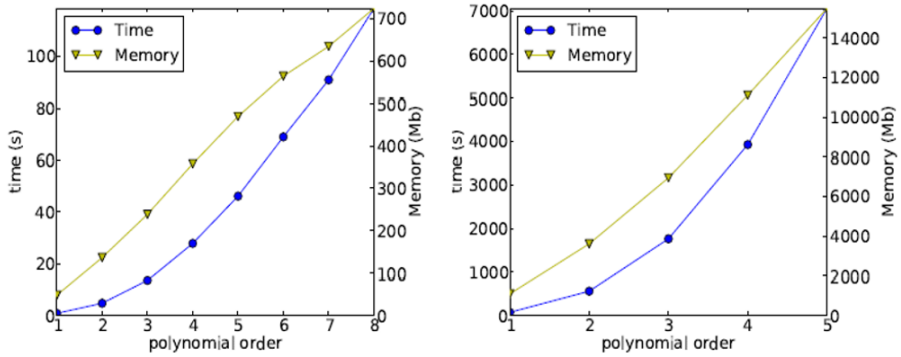
$$\begin{array}{ll} \text{FLOPS=} & \mathcal{O}(Np^6 + N^2), \\ \text{Memory=} & \mathcal{O}(Np^3 + N^{4/3}), \end{array} \tag{3}$$

where N refers to the number of degrees of freedom and p is a constant global polynomial order. The first term in each of the above estimates corresponds to the cost of "static condensation", that is, Gaussian elimination of the interior unknowns, while the second term corresponds to the cost of solving the remaining skeleton problem. We note that the estimates given in (3) were derived for regular meshes with approximately the same number of degrees of freedom in each coordinate direction and a constant polynomial order for all elements in the mesh. Nevertheless, these estimates are appropriate for other mesh configurations provided a sufficient number of elements is present in each coordinate direction. That is, the estimates can be used for arbitrary meshes if they have a sufficiently large number of unknowns, N, for a given polynomial order, p.



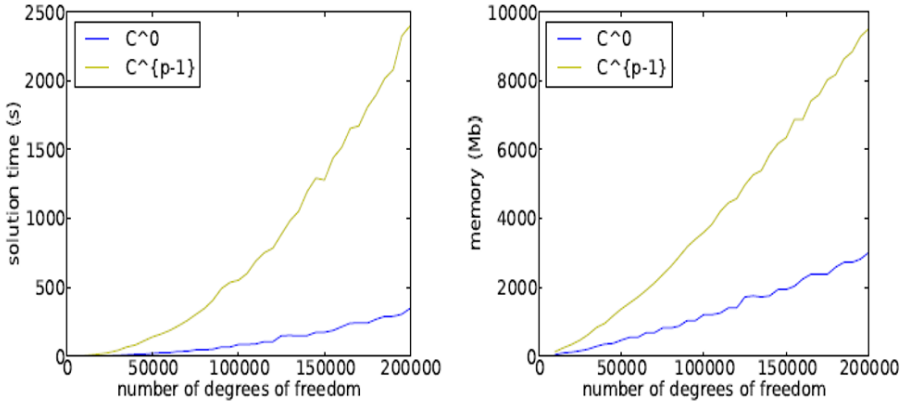
(a) 10,000 degrees of freedom (b) 100,000 degrees of freedom

Figure 4: Solution time and memory usage for 10,000 (left) and 100,000 (right) unknowns using C^0 continuous basis functions.



(a) 10,000 degrees of freedom (b) 100,000 degrees of freedom

Figure 5: Solution time and memory usage for 10,000 (left) and 100,000 (right) unknowns using C^{p-1} continuous basis functions.



(a) Solution time (b) Memory

Figure 6: Comparison of how time and memory scale with the number of degrees of freedom for $p = 2$

The above estimates indicate that if N is large enough with respect to p, then both time and memory are independent of the polynomial order of approximation. Figure 4 displays numerical results for 10,000 and 100,000 unknowns when solving Laplace equation over a unit cube. For the case $N = 10,000$, the solver requires 3 times more memory when using $p = 7$ than with $p = 1$. As we increase N to $100,000$ unknowns, this ratio decreases to a factor of 1.7. Similar results are observed in terms of CPU time.

The above estimates are dimension dependent. That is, for 2D and 1D problems, one obtains [19, 25]:

$$\begin{array}{ll} \text{FLOPS (2D)=} & \mathcal{O}(Np^4 + N^{1.5}) \\ \text{Memory (2D)=} & \mathcal{O}(Np^2 + N\log(N/p^2)) \\ \text{FLOPS (1D)=} & \mathcal{O}(Np^2), \\ \text{Memory (1D)=} & \mathcal{O}(Np). \end{array}$$

Another important example is the case of isogeometric analysis (IGA) [27, 57], which differs from traditional finite element analysis in the choice of basis functions. IGA uses the Non-uniform Rational B-spline (NURBS) basis, which is an important tool in commercial Computer Aided Design (CAD) packages [10, 57, 85]. While standard basis functions used in finite elements are C^0 continuous, NURBS basis can be constructed to possess an arbitrary order of continuity, C^k with $k < p$. In CAD, typically C^2 cubic spaces are used such that curves and surfaces may be designed where the curvature will be at least continuous. In terms of finite elements, the order of continuity k provides an added dimension to the choice of refinement, the hpk-finite element method.

The choice of higher continuous basis functions also has an effect on the structure of the resulting linear system. The higher order continuity of the basis extends the support of the basis functions into neighboring elements. This also means that there are fewer, if any, degrees of freedom fully assembled in the element stiffness matrix. Higher continuous methods, therefore, do not benefit as much from static condensation. Thus, the final complexity estimates for 3D problems using C^{p-1}-spaces become [24]:

$$\begin{array}{ll} \text{FLOPS=} & \mathcal{O}(N^2p^3) \\ \text{Memory=} & \mathcal{O}(p^2N^{4/3}) \end{array}$$

According to these estimates, the number of FLOPS of the C^{p-1} method is p^3 times more expensive than the C^0 if N is large enough. While the amount of memory required by the C^{p-1} B-spline spaces is p^2 times more expensive than that required by the C^0 spaces for sufficiently large N.

Figure 5 displays numerical results for 10,000 and 100,000 unknowns when solving 3D model problem. For the case $N = 10,000$, the solver requires 10 times more memory when using $p = 5$ than with $p = 1$. As we increase N to $100,000$ unknowns, this ratio increases to a factor of 15. Similar trends are observed in terms of CPU time.

To further illustrate the dependance of the performance of direct solvers with respect to the discretization method, we consider again a 3D Laplace problem over a unit cube, with the polynomial order of approximation $p = 2$ (see Fig. 6). For $C^{p-1} = C^1$-basis functions, we observe a great deterioration in performance with

respect to the C^0-counterparts. For $N = 200,000$ unknowns, the C^1-continuous method is about 6 times slower and consumes about 3 times more memory.

In summary, in traditional C^0 spaces, the time and memory needed to solve the system is relatively insensitive to the polynomial order p if N is large enough. However, the time and memory required to solve systems using C^{p-1} spaces is not only larger but also drastically increases with higher p. This effect is particularly noticeable when solving problems in three spatial dimensions. This dependence on p which the C^{p-1} spaces possess should be taken into account when selecting the discretization method or when designing a grid-refinement algorithm. Since the cost of solving systems using the hp finite element method is relatively insensitive to p, the cost of a refinement may be considered in terms of the number of degrees of freedom added, independently if they resulted from a h-refinement or a p-refinement. For higher-continuous spaces, the choice is not as simple, since the increased solution cost has to be factored into the analysis, that is, the reduction in the number of unknowns in the system that higher continuity of the basis functions grants [1, 28], does not directly translate in faster solution time when using direct methods. This open issue warrants further study.

Other finite element methods make different choices in the strong and weak form of the partial differential equation. Some of these methods include the discontinuous Galerkin method (DG) [23] and the discontinuous Petrov-Galerkin method (DPG). These methods lead to local problems which are linked to each other via traces and fluxes on element boundaries. In each method, the use of static condensation typically leads to a clear improvement in the performance of the solution of the algebraic system. In the past, this efficiency has mislead authors to believe that it is cheaper than traditional C^0 finite elements. However, the same is done in C^0 finite elements and so there is no real advantage of discontinuous methods over continuous ones, in terms of the direct solver.

6 Parallel Direct Solvers

Parallelization of multi-frontal solvers is typically performed by dividing the original matrix (domain) among different processors, and by starting with one or several elimination fronts within each processor. Then, sub-matrices of different subdomains are joint into pairs and, in the next level of the elimination tree, unknowns that do not interact with those of other processors are eliminated. This process is iterated until all unknowns are eliminated in the last step of the elimination tree. This procedure is a generalization of the multifrontal solver where different fronts are concurrently dealt in each processor. The hardware architecture (shared, distributed, or hybrid memory) imposes constraints on the particular algorithmic implementation to be used, while the multifrontal solver remains the core idea. Different memory architectures require different synchronization and communication constraints making the optimization process to construct the elimination tree harder.

The parallelization of the direct solver on distributed memory machines can be achieved using a sub-structuring method with non-overlapping sub-domains [44, 91, 92]. In this procedure, one first eliminates the internal unknowns of each sub-

domain with respect to the interface unknowns, then solves the interface problem, and finally performs backward substitution on each sub-domain. The sub-structuring method can be used as a way of parallelization of either the frontal or multi-frontal solver. If the sub-structuring method is used as a way of parallelization of the multi-frontal algorithm, an elimination tree is generated by the multi-frontal solver over each sub-domain independently, with the root nodes related to the sub-domain interface unknowns. On the contrary, if the sub-structuring method is used as a way of parallelization of the frontal algorithm, the frontal solver eliminates subdomain internal unknowns leaving the interface unknowns untouched. As discussed in Section 2, in each block, fully assembled degrees of freedom are eliminated and the Schur complement is formed for the partially assembled degrees of freedom. Figure 7 displays graphically the different steps taken by a parallel direct solver with multiple fronts.

In a distributed memory machine, as sub-matrices are joint into pairs, the number of processors in use is also divided by two. In the last step of the elimination, only one processor is working while all others remain idle. This fact clearly reflects the poor scalability that one should expect from conventional direct solvers, specially, for large and complex 3D problems. To overcome the above problem it is possible to redistribute the partially factorized matrix at each step of the elimination process. A naive redistribution based on calling an existing parallel library package such as ScaLAPACK [86] creates additional overhead that quickly deteriorates the parallel scalability. Thus, one needs to consider more sophisticated implementations of the redistribution step such as the one described in [64], where they achieve approx. a 60% scalability using a few thousand processors for a 2D problem. Notice that this scalability result is really high in context of direct solvers.

Another partial remedy to the above scalability problem is to use "hybrid memory machines", and most popular parallel solvers already incorporate the possibility of using multiple threads (e.g., [67, 72]).

Another option to increase scalability is to employ massively parallel, shared memory architectures, such as the ones provided by General-Purpose Graphical Processing Units (GPGPU's). The shared memory allows to store all frontal matrices and to perform multi-thread computations. If the number of threads is large enough, all the computations within a level of the elimination tree can be performed simultaneously. This implies that for a problem of size N, the computational cost in terms of time is of the order $\mathcal{O}(\log N)$ for 1D problems, since the depth of the elimination tree for a 1D problem is of order $\mathcal{O}(\log N)$. For 2D problems, the computational cost is of the order $\mathcal{O}(N \log N)$ if the number of threads is large enough. These estimates are illustrated in Figure 8 for a Laplace problem over regular 1D and 2D grids. Notice that in large and complex problems the number of processors is not large enough, and the above estimates are not achieved. Thus, when the number of unknowns is larger than the maximum number of threads, the above estimates $\mathcal{O}(\log N)$ and $\mathcal{O}(N \log N)$ for 1D and 2D problems, respectively, should be multiplied by N/t, where t is the number of threads available. Similar scalability has been obtained for higher-continuous isogeometric basis functions for 1d and 2d problems, when implemented in GPUs in application specific implementations [79–81].

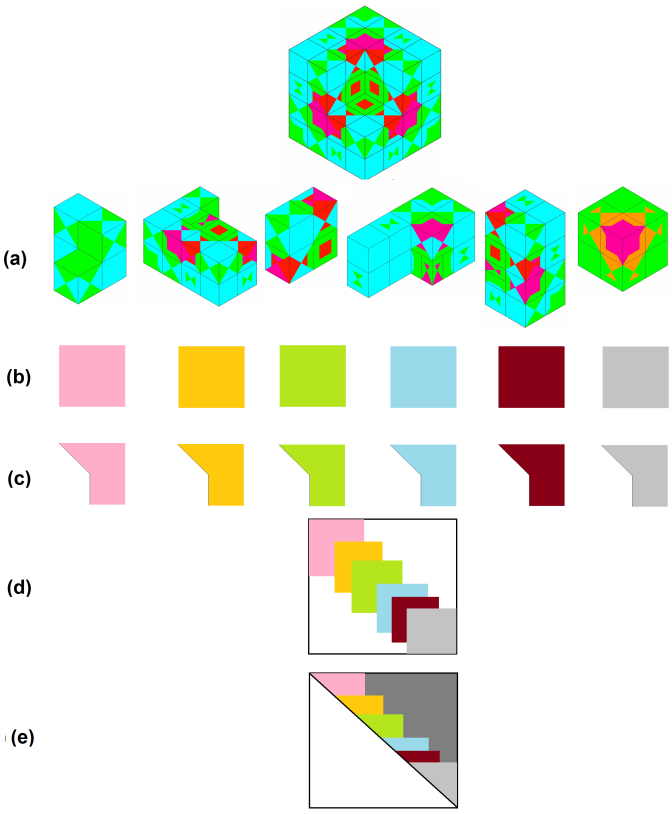


Figure 7: Summary of multiple front solver: (a) Distribution of the computational domain among different subdomains/processors, (b) matrix representation of each subdomain, (c) matrix representation of Gaussian elimination of interior unknowns of each subdomain, (d) matrix representation of the construction of the interface problem, and (e) matrix representation of Gaussian elimination of the interface problem.

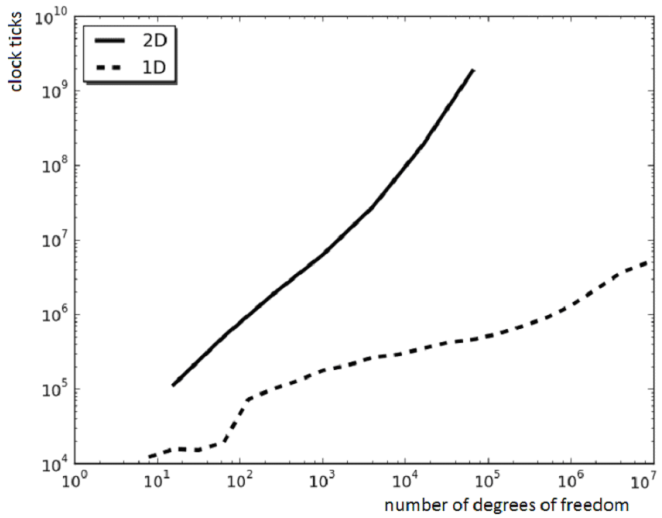


Figure 8: Computational cost of the parallel direct solver for a 1D and 2D Laplace problem over a regular grid. The experiment has been performed on NVIDIA GeForce 8800gt graphic card with 14 multiprocessors and 8 cores per processor.

7 Real-World Applications and Open Challenges

In this section, we illustrate the main properties of direct solvers by selecting a particular real-world application. Based on the numerical results obtained for that application, we discuss the main open challenges in the area of direct solvers that need to be overcome in the near future.

We consider the simulation and inversion of resistivity logging measurements in the Earth's subsurface, an application of great interest to quantify the amount and type of hydrocarbons possibly existing in a reservoir. We consider two different measurement systems: (a) marine controlled-source electromagnetic (CSEM) measurements [77] (see Figure 9, left panel), where the source an receivers are located at the bottom of the sea, and (b) borehole resistivity logging measurements in a deviated well (see Figure 9, right panel). Both types of measurements are extensively used nowadays by the oil industry.

To accurately solve the above applications we employ a discretization based on a Fourier series expansion in one spatial dimension, and a 2D self-adaptive hp-FE method [31, 32] in the remaining spatial dimensions, where both element size h and polynomial order of approximation p vary locally throughout the grid. This hp-Fourier Finite Element method has proven to be very efficient for simulation of resistivity geophysical measurements [74, 78].

For solving these applications, there exist several reasons to prefer the of direct solvers as opposed to iterative solvers. First, due to the presence of elongated elements, convergence of iterative solvers rapidly deteriorates. Second, since these applications are governed by Maxwell's equations, iterative solvers need to be properly designed

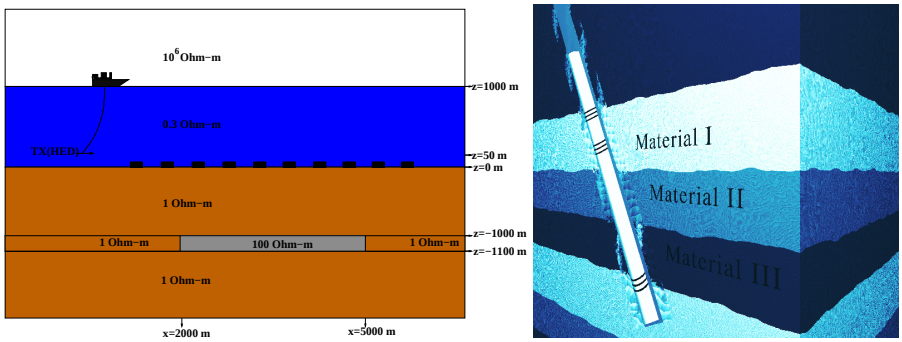


Figure 9: Left panel: marine controlled-source electromagnetics (CSEM) scenario composed of air, water, and a subsurface containing an oil-saturated layer. Right panel: logging instrument in a deviated well composed of several materials in the subsurface and a logging instrument equipped with one transmitter and two receiver antennas.

to deal with the kernel of the curl operator (see [5, 50]), which complicates the implementation. Finally, these applications require solution of a problem with multiple right hand sides for computing the Jacobian and Hessian matrices needed to invert the recorded measurements and obtain a map of the Earth's subsurface. The cost of iterative solvers increases with the number of right hand sides, while the cost of direct solvers is almost insensitive to this number.

The use of GPU's for solving these problems is not possible yet, due to the lack of RAM available in these devices, presently. Therefore, we solve them in a workstation equipped with a large amount of RAM (some applications exceeded 128 GB) or in a parallel computer.

Figure 10 compares the performance of two of the most popular direct solvers — MUMPS and PARDISO— when applied to a marine CSEM problem. We consider only the sequential version of both solvers, and we include in the comparison both the in-core and out-of-core versions. We report the CPU time (top panel of Figure 10) and random access memory (RAM) (bottom panel of Figure 10) necessary to solve the problems. Results indicate that the out-of-core version of PARDISO utilizes roughly twice as much RAM and is twice slower than MUMPS out-of-core. However, the in-core version of PARDISO is the fastest of all. These results indicate that there is still large room for improvement on the out-of-core implementations of the direct solver PARDISO. The use of efficient out-of-core implementations is important to reduce the needed amount of RAM. In our practical application, we could not solve the problem with 3.2 million unknowns using PARDISO, because the solver stopped with an error indicating the lack of available RAM (32 GB).

As described in previous sections, different performance of direct solvers can be due to implementation aspects and/or better ordering of the unknowns. In the case of solvers PARDISO and MUMPS, both solvers use the ordering of unknowns provided by METIS. Thus, it seems that in-core PARDISO is able to handle zeros in a more efficient way (reason why we observe less memory consumption), while the out-of-core implementation is clearly less efficient than the one performed by MUMPS.

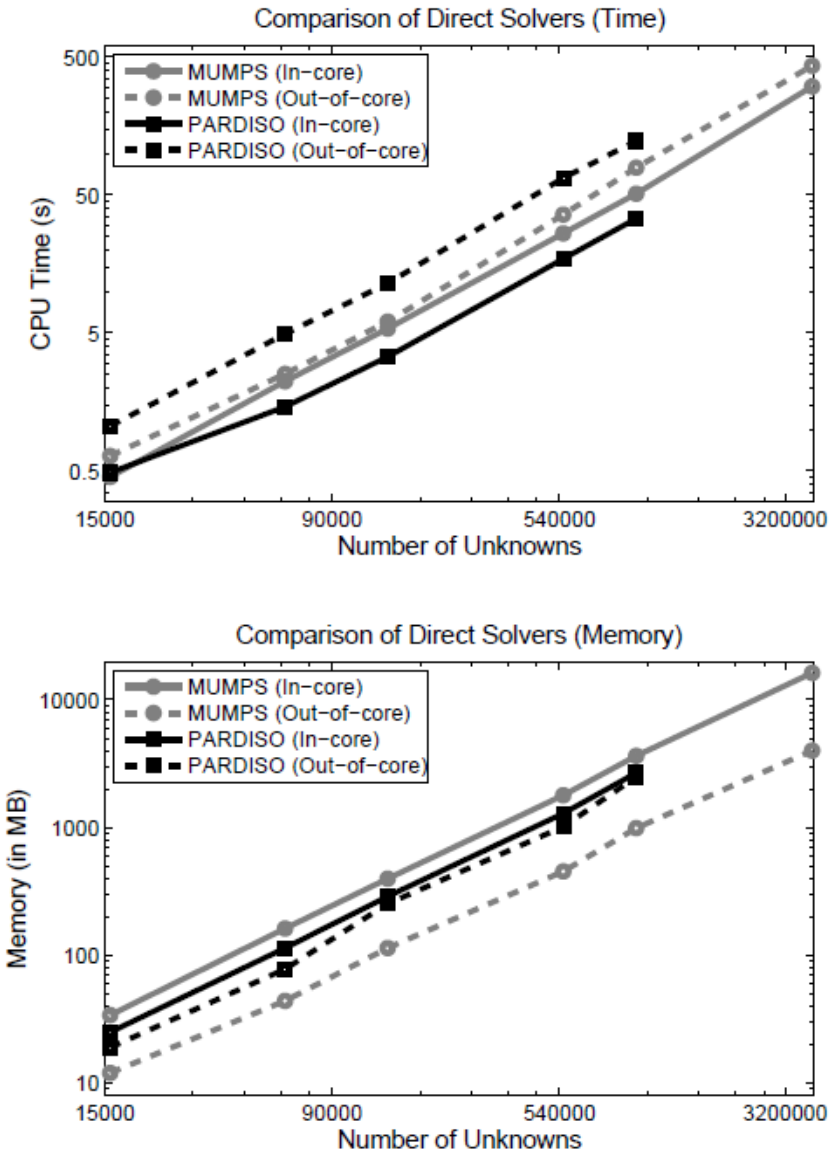


Figure 10: CPU time (top panel) and memory (bottom panel) used by different direct solvers (analysis and LU factorization) vs. problem size when applied to a marine CSEM problem using a Fourier hp-Finite Element Method. Tests performed on a computer equipped with 32 GB of RAM and using only one core of the available 2 GHz dual-core processor.

We now consider the in-core version of MUMPS, we analyze its parallel scalability when applied to our marine CSEM problem with a system of linear equations arising from the hp-Fourier Finite Element discretization. From the time (Figure 11, top panel) and memory (Figure 11, bottom panel) results, we observe a rapid decrease on the needed computational resources as we augment the number of available processors. However, the actual scalability attained by the solver is quite poor. Specifically, for 16 processors, the scalability is approx. 30% in terms of CPU time and about 50% in terms of CPU memory. For 64 processors, the CPU time scalability results further deteriorate to a level of about 10%, while the memory scalability results do not change drastically. These results indicate that the LU factorization for 64 processors is dominated by the communication costs, since otherwise, we would observe a very strong correlation between the CPU time and memory scalability results. Indeed, in our application, the problem size is too small for each processor, which results in computations dominated by the communication costs. Nevertheless, the memory scalability is *only* about 50%, and this is due to the problem of matrix redistribution described in Section 6 that naturally arises in LU factorization algorithms.

Another possible improvement of direct solvers when applied to real-world applications is related to the use of the so-called *supernodes*. A supernode is a set of unknowns (degrees of freedom) that share the same connectivities, and therefore, can be treated from the solver point of view as a single block square dense matrix of dimension equal to the number of unknowns of the supernode. For the case of a single equation discretized with a lowest order finite element method, all unknowns share different connectivities, and the use of supernodes becomes unnecessary. However, for high-order Galerkin discretizations, supernodes becomes critical, since they provide an additional boost in performance by enabling the use of BLAS3 [15] routines that efficiently utilize the computer cache memory. Additionally, supernodes reduce the size of the problem of ordering the unknowns, since they are treated as a single node with an additional weight accounting for the size of the supernode.

MUMPS (as well as other solvers) employ supernodes. However, numerical results in real-world applications suggest some inefficiencies on their implementation. Specifically, we considered the problem of simulating resistivity logging measurements in a deviated well mentioned above. We compared results obtained from solving the resulting linear system of equations with MUMPS against performing static condensation of the unknowns interior to each element and then calling MUMPS over the interface problem. This second approach was twice as fast, indicating that either the ordering of the unknowns using MUMPS with METIS was not optimal or that the identification or use of supernodes was suboptimal. This result also suggests that perhaps one should utilize the information available in the Galerkin software rather than ignoring it and then trying to reproduce it from the resulting matrix, which is a more difficult task. That is, the time is ripe to stop using direct solvers as black boxes. Their robustness has allowed us to treat them this way, but higher-order methods are testing this assertion.

To summarize, we encounter several challenges and open issues on the existing direct solver implementations when solving real-world applications. First, an efficient out-of-core implementation is often needed, since the amount of memory used by direct solvers rapidly increases, specially in the case of 3D simulations. These memory requirements also prevent the usage of GPU's. Second, direct solvers typically display

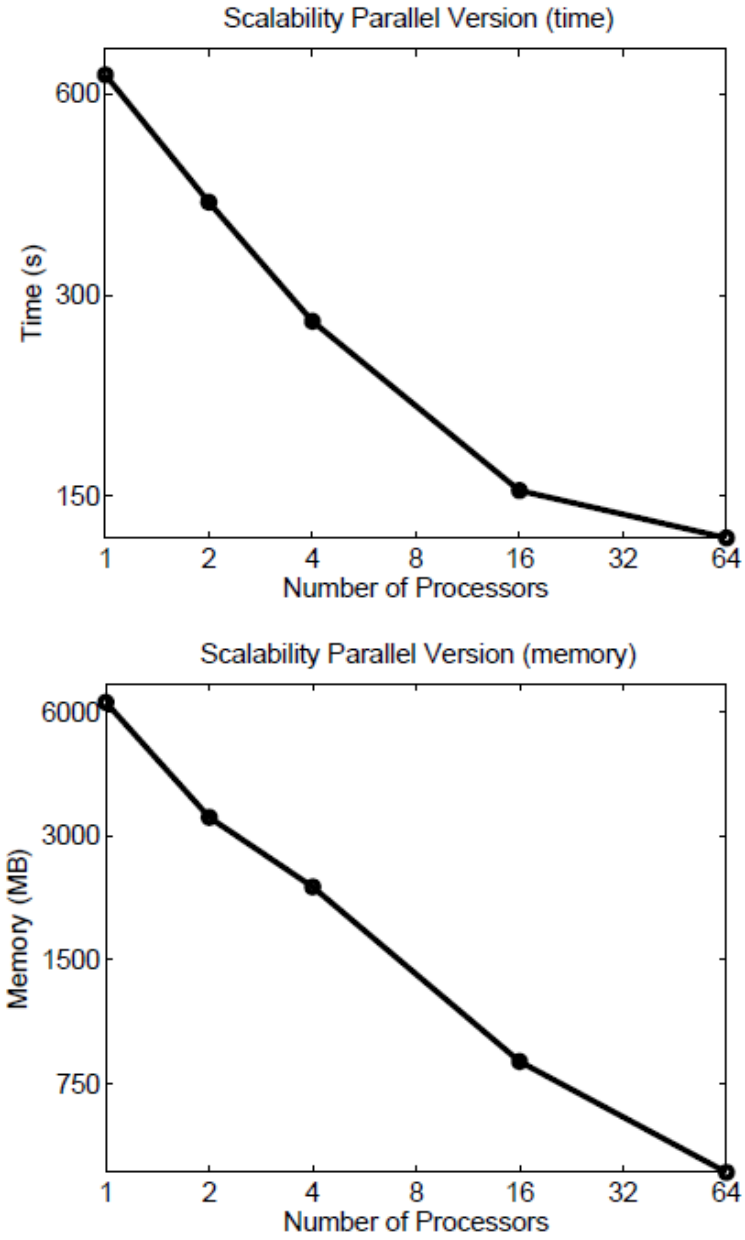


Figure 11: CPU time (top panel) and memory (bottom panel) used by the parallel execution of MUMPS when applied to a marine CSEM problem using a Fourier hp-Finite Element Method.

a non-optimal parallel scalability in distributed-memory machines, which is an area of active research. Finally, the efficient use of supernodes as well as better ordering techniques could lead to increased efficient. In this area, perhaps the use of the connectivity information already available in the Galerkin simulation software could lead to a simplification and even an improvement on the currently existing ordering algorithms. This will lead to further integration of the discretization with the solution strategy.

8 Conclusions

We give a brief introduction to direct solvers for linear algebraic systems resulting from Galerkin discretizations. We describe the most important variants of Gaussian elimination, which is the standard direct solution technique. We explain the frontal and multi-frontal approaches and their relation with Schur complement and static condensation concepts. State-of-the-art implementations of direct solvers are described, including MUMPS and PARDISO, which are among the most popular versions, due to their ease of access and robustness. Following, the effect of ordering in the solver performance is shown with a simple example, which leads to the presentation of the optimal graph-based ordering, such as the one produced by METIS. In the opinion of the authors, an optimal ordering that uses information from the discretization, that is, a grid-based ordering could be the most efficient. However, these optimal algorithms are more specific and thus less portable than graph-partitioned ones. Additionally, ordering algorithms should account for the pivoting needed to minimize the round-off error, which is analyzed and illustrated with a 1D problem in Section 4. Section 5 exemplifies how different Galerkin methods lead to different connectivities and in turn, this affects solver performance. In particular, systems with less connected graphs are easier to solve, as the comparison of standard C^0 against C^{p-1} finite elements exemplified. In the next section of the paper we discuss how parallel direct solvers are implemented and the effect of how the interplay between multi-threading and problem size affects performance. Finally, Section 7 is devoted to illustrate the performance of two popular direct solvers when applied to real-world applications, which leads to a discussion on the most important open problems in the area of direct solvers, namely, efficient: (a) ordering algorithms, (b) parallelization techniques, (c) out-of-core implementations, and (d) use of supernodes.

9 Acknowledgments

The work reported in this paper was partially funded by the Spanish Ministry of Sciences and Innovation under project MTM2010-16511. The work of the second author has been partially supported by Polish MNiSW grant no. 519 447 739.

References

[1] I Akkerman, Y Bazilevs, V M Calo, T J R Hughes, and S Hulshoff. The role of continuity in residual-based variational multiscale modeling of turbulence.

Computational Mechanics, 41(3):371–378, 2007.

[2] AMD. Approximate Minimum Degree (AMD). Webpage http://www.cise.ufl.edu/research/sparse/amd/, 2011.

[3] P. R. Amestoy, A. Guermouche, J.-Y. L'Excellent, and S. Pralet. Hybrid scheduling for the parallel solution of linear systems. *Parallel Computing*, 32:136–156, 2006.

[4] P.R. Amestoy, T.A. Davis, I.S. Duff, et al. An approximate minimum degree ordering algorithm. *SIAM Journal on Matrix Analysis and Applications*, 17(4):886–905, 1996.

[5] D. N. Arnold, R. S. Falk, and R. Winther. Multigrid in H(div) and H(curl). *Numer. Math.*, 85(2):197–217, 2000.

[6] F. Auricchio, M. Conti, S. Morganti, and A. Reali. Shape memory alloys: from constitutive modeling to finite element analysis of stent deployment. *Computer Modeling in Engineering & Sciences*, 57:225–243, 2010.

[7] I. Babuska, A. Craig, J. Mandel, and J. Pitkaranta. Efficient preconditioning for the p-version finite element method in two dimensions. *SIAM J. Numer. Anal.*, 28(3):624–661, 1991.

[8] Y Bazilevs, V Calo, J Cottrell, T Hughes, A Reali, and G Scovazzi. Variational multiscale residual-based turbulence modeling for large eddy simulation of incompressible flows. *Computer Methods in Applied Mechanics and Engineering*, 197(1-4):173–201, 2007.

[9] Y Bazilevs, V M Calo, Y Zhang, and T J R Hughes. Isogeometric Fluid Structure Interaction Analysis with Applications to Arterial Blood Flow. *Computational Mechanics*, 38(4-5):310–322, 2006.

[10] Y. Bazilevs, V.M. Calo, J.A. Cottrell, J.A. Evans, T.J.R. Hughes, S. Lipton, M.A. Scott, and T.W. Sederberg. Isogeometric analysis using T-splines. *Computer Methods in Applied Mechanics and Engineering*, page 34, 2010.

[11] Y Bazilevs, J R Gohean, T J R Hughes, R D Moser, and Y Zhang. Patient-specific isogeometric fluid-structure interaction analysis of thoracic aortic blood flow due to implantation of the Jarvik 2000 left ventricular assist device. *Computer Methods in Applied Mechanics and Engineering*, 198(45-46):3534–3550, 2009.

[12] Y Bazilevs, M Hsu, I Akkerman, S Wright, K Takizawa, and B Henicke. 3D simulation of wind turbine rotors at full scale. *International Journal for Numerical Methods in Fluids*, (August 2010):207–235, 2011.

[13] Y Bazilevs, C Michler, V M Calo, and T J R Hughes. Isogeometric variational multiscale modeling of wall-bounded turbulent flows with weakly enforced boundary conditions on unstretched meshes. *Computer Methods in Applied Mechanics and Engineering*, 199(13-16):780–790, 2010.

[14] P. Bientinesi, V. Eijkhout, K. Kim, J. Kurtz, and R. van de Geijn. Sparse Direct Factorizations through Unassembled Hyper-Matrices. *Computer Methods in Applied Mechanics and Engineering*, 199:430–438, 2010.

[15] BLAS. Basic linear algebra subprograms. http://netlib.org/blas, 2011.

[16] J. H. Bramble. *Multigrid Methods*. Pitman Research Notes in Mathematics Series. 294. Harlow: Longman Scientific & Technical. viii, 161 p. , 1993.

[17] F. Brezzi and M. Fortin. *Mixed and Hybrid Finite Element Methods*. Springer-Verlag, Berlin, 1991.

[18] V M Calo, N F Brasher, Y Bazilevs, and T J R Hughes. Multiphysics model for blood flow and drug transport with application to patient-specific coronary artery flow. *Computational Mechanics*, 43(1):161–177, 2008.

[19] V.M. Calo, N. O. Collier, D. Pardo, and M. Paszyński. Computational complexity and memory usage for multi-frontal direct solvers used in p finite element analysis. *Procedia Computer Science*, 4:1854–1861, 2011.

[20] V.M. Calo, H. Gómez, Y. Bazilevs, G.P. Johnson, and T.J.R. Hughes. Simulation of engineering applications using isogeometric analysis. In *TeraGrid08*, 2008.

[21] GF Carey and E. Barragy. Basis function selection and preconditioning high degree finite element and spectral methods. *BIT Numerical Mathematics*, 29(4):794–804, 1989.

[22] P.G. Ciarlet. *The finite element method for elliptic problems*. North-Holland, Amsterdam, 1978.

[23] B. Cockburn, G.E. Karniadakis, and C.-W. Shu (Eds.). In *Discontinuous Galerkin Methods*, Lecture Notes in Computational Science and Engineering 11. Springer, Berlin, 2000.

[24] N. Collier, D. Pardo, L. Dalcin, M. Paszynski, and V. M. Calo. The cost of continuity: a study of the performance of isogeometric finite elements using direct solvers. *Computer Methods in Applied Mechanics and Engineering*, submitted 2011.

[25] N.O. Collier, M.R D. Pardo, Paszyński, and V.M. Calo. Computational complexity and memory usage estimates for multi-frontal direct solvers for structured finite elements. *Journal of Computational Science*, 2011. Submitted.

[26] J A Cottrell, A Reali, Y Bazilevs, and T J R Hughes. Isogeometric analysis of structural vibrations. *Computer Methods in Applied Mechanics and Engineering*, 195(41-43):5257–5296, 2006.

[27] J. Austin Cottrell, T. J. R. Hughes, and Yuri Bazilevs. *Isogeometric Analysis: Toward Unification of CAD and FEA*. John Wiley and Sons, 2009.

[28] J.A. Cottrell, T.J.R. Hughes, and A. Reali. Studies of refinement and continuity in isogeometric structural analysis. *Computer Methods in Applied Mechanics and Engineering*, page 23, 2007.

[29] L Dede, T J R Hughes, and S Lipton. Isogeometric Analysis of Topology Optimization Problems Based on the Phase-Field Model Design Optimization. *Optimization*, 45(3):1630–1633, 2011.

[30] Luca Dede, T J R Hughes, Scott Lipton, and V M Calo. Structural topology optimization with isogeometric analysis in a phase field approach. In *USNCTAM2010, 16th US National Congree of Theoretical and Applied Mechanics*, 2010.

[31] L. Demkowicz. *Computing with hp-Adaptive Finite Elements. Volume I: One and Two Dimensional Elliptic and Maxwell Problems*. Chapman and Hall, 2006.

[32] L. Demkowicz, J. Kurtz, D. Pardo, M. Paszynski, W. Rachowicz, and A. Zdunek. *Computing with hp-Adaptive Finite Elements. Volume II. Frontiers: Three-Dimensional Elliptic and Maxwell Problems with Applications*. Chapman and Hall, 2007. chapters 8-12.

[33] J.J. Dongarra, L.S. Duff, D.C. Sorensen, and H.A.V. Vorst. *Numerical Linear Algebra for High Performance Computers*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1998.

[34] I. S. Duff and J. K. Reid. The multifrontal solution of indefinite sparse symmetric linear. *ACM Trans. Math. Softw.*, 9:302–325, 1983.

[35] I. S. Duff and J. K. Reid. The multifrontal solution of unsymmetric sets of linear systems. *SIAM J. Sci. Stat. Comput.*, 5:633–641, 1984.

[36] A. El maliki, M. Fortin, N. Tardieu, and A. Fortin. Iterative solvers for 3d linear and nonlinear elasticity problems: Displacement and mixed formulations. *International Journal for Numerical Methods in Engineering*, 83:1780 – 1802, 2010.

[37] M. Ferrari. Cancer nanotechnology: opportunities and challenges. *Nature Reviews Cancer*, 5(3):161–171, Mar 2005.

[38] M. Ferrari. Nanovector therapeutics. *Current Opinions in Chemical Biology*, 9(4):343–346, August 2005.

[39] P. Geng, T. J. Oden, and R. A. Van de Geijn. A parallel multifrontal algorithm and its implementation. *Comput. Methods in Appl. Mech. Eng.*, 149:289–301, 1997.

[40] A. George. Nested dissection of a regular finite element mesh. *SIAM Journal on Numerical Analysis*, pages 345–363, 1973.

[41] A. George and J.W.H. Liu. The evolution of the minimum degree ordering algorithm. *Siam review*, pages 1–19, 1989.

[42] M. Ghommem, M.R Hajj, D.T. Mook, B.K. Stanford, P.S. Beran, R.D. Snyder, and L.T. Watson. Global optimization of apping kinematics for micro air vehicles. *Journal of Fluids and Structures*, 2011. Under review.

[43] M. Ghommem, M.R. Hajj, C. L. Pettit, and P.S. Beran. Stochastic modeling of incident gust effects on aerodynamic lift. *Journal of Aircraft*, 47(5):1720–1729, 2010.

[44] L. Giraud, A. Marocco, and Rioual J.-C. Iterative versus direct parallel substructuring methods in semiconductor device modeling. *Numerical Linear Algebra with Applications*, 12:33–55, 2005.

[45] G.H. Golub and C.F. Van Loan. *Matrix computations (3rd ed.)*. Johns Hopkins University Press, Baltimore, MD, USA, 1996.

[46] J. Gopalakrishnan and G. Kanschat. A multilevel discontinuous galerkin method. *Numerische Mathematik*, 95:527–550, 2003. 10.1007/s002110200392.

[47] N.I.M. Gould, J.A. Scott, and Y. Hu. A numerical evaluation of sparse direct solvers for the solution of large sparse symmetric linear systems of equations. *ACM Trans. Math. Softw.*, 33, June 2007.

[48] L. Grigori, J.W. Demmel, and X.S. Li. Parallel symbolic factorization for sparse LU with static pivoting. *SIAM J. Scientific Computing*, 29(3):1289–1314, 2007.

[49] W. Hackbusch, L. Grasedyck, and S. Borm. An introduction to hierarchical matrices. *Math. Bohem*, 127(2):229–241, 2002.

[50] R. Hiptmair. Multigrid method for Maxwell's equations. *SIAM J. Numer. Anal.*, 36(1):204–225, 1998.

[51] A.J. Hoffman, M.S. Martin, and D.J. Rose. Complexity bounds for regular finite difference and finite element grids. *SIAM Journal on Numerical Analysis*, pages 364–369, 1973.

[52] S. Hossain, S.F.A. Hossainy, Y. Bazilevs amd V.M. Calo, and T.J.R. Hughes. Mathematical modeling of coupled drug and drug-encapsulated nanoparticle transport in patient-specific coronary artery walls. *Computational Mechanics, doi: 10.1007/s00466-011-0633-2*, 2011.

[53] HSL. Harwell Subroutine Library. http://www.cse.scitech.ac.uk/nag/hsl/, 2008.

[54] M.-C. Hsu, I. Akkerman, and Y. Bazilevs. High-performance computing of wind turbine aerodynamics using isogeometric analysis. *Computers and Fluids*, 2011.

[55] T J R Hughes, A Reali, and G Sangalli. Efficient quadrature for NURBS-based isogeometric analysis. *Computer Methods in Applied Mechanics and Engineering*, 199(5-8):301–313, 2010.

[56] T.J.R. Hughes. *The finite element method: Linear static and dynamic finite element analysis*. Prentice Hall, Englewood Cliffs, NJ, 1987.

[57] T.J.R. Hughes, J.A. Cottrell, and Y. Bazilevs. Isogeometric analysis: CAD, finite elements, NURBS, exact geometry, and mesh refinement. *Computer Methods in Applied Mechanics and Engineering*, 194:4135–4195, 2005.

[58] B. Irons. A frontal solution program for finite-element analysis. *Int. J. Num. Meth. Eng.*, 2:5–32, 1970.

[59] C. Johnson. *Numerical solution of partial differential equations by the finite element method*. Cambridge University Press, Sweden, 1987.

[60] I. N. Katz, A. G. Peano, and M. P. Rossow. Nodal variables for complete conforming finite elements of arbitrary polynomial order. *Computers Mathematics with Applications*, 4:85–112, 1978.

[61] K. Kim. Personal Communication, 2010.

[62] D.A. LaVan, T. McGuire, and R. Langer. Small-scale systems for in vivo drug delivery. *Nature Biotechnology*, 21:1184–1191, 2003.

[63] Luc L. Lavier and Gianreto Manatschal. A mechanism to thin the continental lithosphere at magma-poor margins. *Nature*, 440:324–328, 2006.

[64] L. Lin, C. Yang, J. Lu, L. Ying, and E. Weinan. A fast parallel algorithm for selected inversion of structured sparse matrices wtih application to 2D electronic structure calculations. *SIAM J. Scientific Computing*, 33:1329–1351, 2011.

[65] J.W.H. Liu. The multifrontal method for sparse matrix solution: Theory and practice. *Siam Review*, pages 82–109, 1992.

[66] METIS. Family of Multilevel Partitioning Algorithms. http://glaros.dtc.umn.edu/gkhome/views/metis, 2007.

[67] MUMPS. A multifrontal massively parallel sparse direct solver. http://graal.ens-lyon.fr/MUMPS/, 2010.

[68] F. Nobile, R. T. Rockafellar, C. Schwab, R. F. Tempone, and R. J-B Wets. Ima annual program year workshop, 2011, computing with uncertainty: Mathematical modeling, numerical approximation and large scale optimization of complex systems with uncertainty. http://www.ima.umn.edu/2010-2011/W10.18-22.10/, 2011.

[69] Intergovernmental Panel on Climate Change. http://srren.ipcc-wg3.de/, 2011.

[70] Intergovernmental Panel on Climate Change. http://www.ipcc.ch/index.htm, 2011.

[71] NSF Blue Ribbon Panel on Simulation-Based Engineering Science. www.nsf.gov/pubs/reports/sbes_final_report.pdf, 2006.

[72] PARDISO. Thread-safe solver of linear equations. http://www.pardiso_project.org/, 2008.

[73] D. Pardo. *Integration of hp-adaptivity with a two grid solver: applications to electromagnetics*. PhD thesis, The University of Texas at Austin, April 2004.

[74] D. Pardo, V. M. Calo, C. Torres-Verdín, and M. J. Nam. Fourier series expansion in a non-orthogonal system of coordinates for simulation of 3D DC borehole resistivity measurements. *Computer Methods in Applied Mechanics and Engineering*, 197(1-3):1906–1925, 2008.

[75] D. Pardo and L. Demkowicz. Integration of hp-adaptivity with a two grid solver for elliptic problems. *Computer Methods in Applied Mechanics and Engineering*, 195:674–710, 2006.

[76] D. Pardo, L. Demkowicz, and J. Gopalakrishnan. Integration of hp-adaptivity and a two grid solver for electromagnetic problems. *Computer Methods in Applied Mechanics and Engineering*, 195:2533–2573, 2006.

[77] D. Pardo, M. J. Nam, C. Torres-Verdín, M. Hoversten, and I. Garay. Simulation of Marine Controlled Source Electromagnetic (CSEM) Measurements Using a Parallel Fourier hp-Finite Element Method. *Computational Geosciences*, 15(1):53–67, 2011.

[78] D. Pardo, C. Torres-Verdín, M. J. Nam, M. Paszynski, and V. M. Calo. Fourier series expansion in a non-orthogonal system of coordinates for simulation of 3D alternating current borehole resistivity measurements. *Computer Methods in Applied Mechanics and Engineering*, 197:3836–3849, 2008.

[79] M. Paszyński, K. Kuźnik, and V.M. Calo. Graph grammar-based multi-frontal parallel direct solver for two-dimensional isogeometric analysis. *Submitted to 26th IEEE International Parallel & Distributed Processing Symposium,* Shanghai, China, May 21-25, 2012.

[80] M. Paszyński, K. Kuźnik, and V.M. Calo. Grammar-based multi-frontal solver for isogeometric analysis in 1d. *Scientific Programming*, 2011. Submitted.

[81] M. Paszyński, K. Kuźnik, and V.M. Calo. Parallel multi-frontal direct solver for isogeometric analysis of 2d problems. *Computer Methods in Applied Mechanics and Engineering*, 2011. Submitted.

[82] M. Paszynski, D. Pardo, C. Torres-Verdin, L. Demkowicz, and V.M. Calo. A parallel direct solver for the self-adaptive hp finite element method. *Journal of Parallel and Distributed Computing*, 70(3):270 – 281, 2010.

[83] A. G. Peano. *Hierarchies of Conforming Finite Elements*. PhD thesis, Sever Institute of Technology, Washington University, St. Luis, 1975.

[84] A. G. Peano. Hierarchies of conforming finite elements for plane elasticity and plate bending. *Computers Mathematics with Applications*, 2:211 –224, 1976.

[85] L. Piegl and W. Tiller. *The NURBS Book (Monographs in Visual Communication), 2nd ed.* Springer-Verlag, New York, 1997.

[86] ScaLAPACK. Scalable linear algebra package. http://netlib.org/scalapack, 2011.

[87] P. Schmitz and L. Ying. A fast direct solver for elliptic problems on Cartesian meshes in 2d, submitted, 2011. http://www.math.utexas.edu/users/lexing/publications/index.html.

[88] P. Schmitz and L. Ying. A fast direct solver for elliptic problems on Cartesian meshes in 3d, submitted, 2011. http://www.math.utexas.edu/users/lexing/publications/index.html.

[89] J. Schulze. Towards a tighter coupling of bottom-up and top-down sparse matrix ordering methods. *BIT*, 41:2001, 2001.

[90] SCOTCH. Graph partitioning, static mapping, and sparse matrix block ordering. http://www.labri.fr/perso/pelegrin/scotch/, 2011.

[91] A. Scott. Parallel frontal solvers for large sparse linear systems. *ACM Trans. on Math. Soft.*, 29:395–417, 2003.

[92] B. F. Smith, P. Bjorstad, and Gropp W. *Domain Decomposition, Parallel Multi-Level Methods for Elliptic Partial Differential Equations*. Cambridge University Press, New York, 1996.

[93] SPOOLES. SParse Object Oriented Linear Equations Solver. http://www.netlib.org/linalg/spooles/spooles.2.2.html, 2011.

[94] G. Stadler, M. Gurnis, C. Burstedde, L. C. Wilcox, L. Alisic, and O. Ghattas. The dynamics of plate tectonics and mantle flow: From local to global scales. *Science*, 329:1033–1038, 2010.

[95] SuperLU. A general purpose package for solution of large sparse systems of linear equations. http://crd.lbl.gov/%7Exiaoye/SuperLU/, 2008.

[96] B. A. Szabo and I. Babuska. *Finite Element Analysis*. John Wiley and Sons, Ney York, 1991.

[97] UMFPACK. Unsymmetric Multi-Frontal Package. http://www.cise.ufl.edu/research/sparse/umfpack/, 2011.

[98] J. Xia, S. Chandrasekaran, M. Gu, and X.S. Li. Superfast multifrontal method for large structured linear systems of equations. *SIAM J. Matrix Anal. Appl*, 31(3):1382–1411, 2009.