Iterative versus direct parallel substructuring methods in semiconductor device modeling

L. Giraud[*], A. Marrocco[†], J.-C. Rioual[*]

TR/PA/02/114

Abstract

The numerical simulation of semiconductor devices is extremely demanding in term of computational time because it involves complex embedded numerical schemes. At the kernel of these schemes is the solution of very ill-conditioned large linear systems. In this paper, we present the various ingredients of some hybrid iterative schemes that play a central role in the robustness of those solvers when they are embedded in other numerical schemes. On a set of bidimensional unstructured mixed finite element problems representative of semiconductor simulation, we perform a fair and detailed comparison between parallel iterative and direct linear solution techniques. We show that iterative solvers can be robust enough to solve the very challenging linear systems that appear in those simulations.

1 Introduction

One route to the solution of large sparse linear systems is the use of hybrid methods that combine direct and iterative methods. For the solution of linear systems arising from the discretization of Partial Differential Equations (PDE), such techniques are referred to as domain decomposition methods (see for instance [15, 26, 31]) and are based on a splitting of the underlying mesh into several subdomains. In that framework, there are two main families depending whether there is, or not, an overlap between the different subdomains. The techniques based on overlapping subdomains are referred to as Schwarz methods [30], while the ones using non-overlapping subdomains often lead to a reduced linear system condensed on the interface and are usually referred to as substructuring techniques.

In this paper, we consider the second approach for the parallel solution of the linear systems involved in the numerical simulations of bidimensional semiconductor devices. Those problems are challenging to solve because they are large, very ill-conditioned, extremely badly scaled and many of them have to be solved in the nonlinear time dependent scheme involved in a complete simulation. In that respect, robust and efficient parallel linear solvers should be selected in order to reduce the elapsed time required for a complete simulation. In our work we follow the substructuring approach and combined it with both preconditioned iterative Krylov solvers and sparse factorization techniques. We show how some components of the iterative solvers should be tuned, not only to enable the convergence of the linear iterations, but also to ensure the convergence of the outer schemes.

The paper is organized as follows. In Section 2, we briefly described the main equations governing the device modeling as well as the unstructured mixed finite element discretization that is implemented [22]. In Section 3, we present the substructuring approach followed to implement the parallel iterative linear solvers. In particular, we briefly describe the state of the art sparse

[*]CERFACS, Toulouse, France
[†]INRIA, Rocquencourt, France

direct software that we used as building box for all the parallel implementations. In the next section, we illustrate the coupling that exists between the linear and nonlinear iterations. We report on numerical experiments that show the role played by some components of the iterative solvers in the design of a robust device simulation software tool. More precisely, we investigate the influence of the preconditioner and the selected scaling of the ill-conditioned linear systems and indicate the possible weakness of the classical stopping criterion implemented in many iterative packages. A performance comparison between iterative versus direct substructuring approaches is performed in Section 5. Finally, we conclude with some remarks and possible prospective works that might be undertaken to further develop this study.

2 A brief overview of semiconductor device modeling

The drift-diffusion equations that govern the semiconductor device modeling may be written in the form

$$\begin{cases} \text{Find } (\phi, \phi_n, \phi_p) \text{ so that} \\ -div(\epsilon \nabla \phi) + q[N(\phi, \phi_n) - P(\phi, \phi_p) - Dop] = 0, \\ -div(q\mu_n N(\phi, \phi_n)\nabla \phi_n) + qGR(\phi, \phi_n, \phi_p) = 0, \\ -div(q\mu_p P(\phi, \phi_p)\nabla \phi_p) - qGR(\phi, \phi_n, \phi_p) = 0, \\ \text{with mixed Dirichlet-Neumann boundary conditions.} \end{cases} \tag{1}$$

A detailed description of the numerical solution of the set of Equations (1) is out of the scope of this paper. We just remind the basic informations required to understand the results of this paper; more details can be found in [8, 22, 27]. This model describes several quantities inside a two-dimensional domain Ω which corresponds to an electronic device. The unknowns are ϕ the electrostatic potential, ϕ_n the quasi-Fermi level for the electrons and ϕ_p the quasi-Fermi level for the holes. The electron charge is denoted by q. The mobilities of the electrons and the holes are denoted by μ_n and μ_p and are material dependent. $GR(.)$ is a function which represents the mechanism of generation-recombination of electrons and holes. This function may have different expressions, depending on the physics taken into account. Dop is a given function only depending on the space variable. The concentration in electrons and holes are N and P. The model for N and P is nonlinear and based on the exponential function for the Boltzmann statistics. The high dynamics and nonlinearity of N and P leads to numerical difficulties that show up in the linear systems and make the iterative methods difficult to converge.

The solution scheme selected to solve Equations (1) consists in decoupling the three equations and recombining them in an iterative process as summarized in Algorithm 1. We notice that, because the nonlinear problems are too stiff, a stabilization technique is used based on the introduction of an artificial time dependent problem. The steady state of this problem is the solution of the nonlinear equations we aim at solving. The Equations (2), (3), (4) respectively are referred to as the nonlinear Poisson equation, the continuity equation for the electrons and the continuity equation for the holes respectively. They are discretized using mixed finite elements on an unstructured triangular mesh (see for instance Figure 1). The discretization of Equations (2), (3), (4) leads to three nonlinear systems of equations. At each time step, each of these nonlinear systems is then solved by a Newton-Raphson iterative method. At each step of the Newton method, a linear system has to be solved. This linear system is symmetric positive definite in the case of Equation (2), unsymmetric in the case of Equations (3) and (4). In the numerical scheme, the potentials are eliminated from the fluxes and therefore the unknowns are only associated with the edges of the triangles in the mesh. Therefore the linear systems are sparse with a maximum of

Knowing ϕ^k, ϕ_n^k, ϕ_p^k.

Step 1. Compute ϕ^{k+1} by solving the *nonlinear Poisson equation*

$$\frac{\phi^{k+1}-\phi^k}{\Delta t} - div(\epsilon\nabla\phi^{k+1}) + q[N(\phi^{k+1},\phi_n^k) - P(\phi^{k+1},\phi_p^k) - Dop] = 0 \tag{2}$$

Step 2. Compute ϕ_n^{k+1} by solving the *continuity equation for the electrons*

$$\frac{\phi_n^{k+1}-\phi_n^k}{\Delta t} - div(q\mu_n N(\phi^{k+1},\phi_n^{k+1})\nabla\phi_n^{k+1}) + qGR(\phi^{k+1},\phi_n^{k+1},\phi_p^k) = 0 \tag{3}$$

Step 3. Compute ϕ_p^{k+1} by solving the *continuity equation for the holes*

$$\frac{\phi_p^{k+1}-\phi_p^k}{\Delta t} - div(q\mu_p P(\phi^{k+1},\phi_p^{k+1})\nabla\phi_p^{k+1}) - qGR(\phi^{k+1},\phi_n^{k+1},\phi_p^{k+1}) = 0 \tag{4}$$

Step 4. Check convergence.

Algorithm 1: Numerical solution of Equation (1) based on a decoupling of the three unknowns combined with an artificial evolution process.

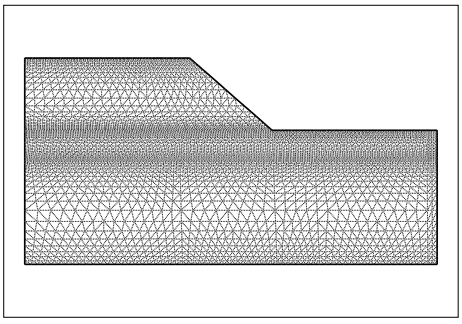

Figure 1: Triangular mesh with 5194 elements.

five nonzero elements per row. Moreover, these linear systems are badly scaled and ill-conditioned (particularly in the case of the continuity equations) due to the high dynamics of the quantities involved in the simulation. A preliminary sequential version of this numerical process has been implemented using a sparse direct solver based on old software technology. In that implementation the linear solver is the most time consuming part of the process with more than 99% of the CPU time. A problem based on meshes with 30000 elements may require 24 hours of computational time on a SGI O2000. Iterative methods based on Krylov solvers combined with incomplete LU (ILU) preconditioners [28] have been tested but were not robust enough to ensure the convergence of the nonlinear scheme even with a large level of fill-in. In the following sections we will describe the linear solvers that have been implemented to efficiently solve those linear systems on a distributed memory platform. The primary motivations to move on parallel platforms is first to reduce the elapsed time to perform a simulation and second to enable us to solve larger problems thanks to the use of more memory.

3 Substructuring methods

In this section, we present a domain decomposition algorithm and its implementation in a distributed memory environment enabling to perform in parallel the complete simulation. The formulation proposed in Section 2 for semiconductor device modeling is mainly vectorial and most of the resulting code is naturally parallelizable. Once the mesh has been partitioned, all the data defined on one subdomain is associated with one processor. The main difficulty consists then in implementing an efficient linear solver. For our implementation we select MPI [20, 21] as message passing library and the Fortran 90 as programming language.

3.1 Iterative substructuring algorithm

The technique we describe is often called the *substructuring* method, referring to structural mechanics problems which were historically its first area of application. It consists in splitting the mesh into non-overlapping sub-meshes/subdomains, first eliminating the unknowns associated with the interior nodes from the equations associated with the nodes on the interfaces, then solving the interface problem and finally solving back the internal problems on each subdomain.

Let us now further describe this technique and let $Au = f$ be the linear problem to solve. We assume that the domain Ω is partitioned into N non-overlapping subdomains Ω_1, ..., Ω_N with boundaries $\partial\Omega_1$, ..., $\partial\Omega_N$. Let B be the set of all indices of the discretized points which belong to the interfaces between the subdomains. Grouping the points corresponding to B in the vector u_B and those corresponding to the interior I of the subdomains in u_I, we get the reordered problem :

$$\begin{pmatrix} A_{II} & A_{IB} \\ A_{BI} & A_{BB} \end{pmatrix} \begin{pmatrix} u_I \\ u_B \end{pmatrix} = \begin{pmatrix} f_I \\ f_B \end{pmatrix}. \tag{5}$$

Eliminating u_I from the second block row of Equation (5) leads to the following reduced equation for u_B :

$$Su_B = f_B - A_{BI}A_{II}^{-1}f_I, \text{ where } S = A_{BB} - A_{BI}A_{II}^{-1}A_{IB} \tag{6}$$

is the Schur complement of the matrix A_{II} in A, and is usually referred to as the Schur complement matrix. Notice that if A is symmetric positive definite (SPD) the matrix S inherits this property. Algorithm 2 describes the Schur complement method. We define Γ_i the internal frontier of the subdomain Ω_i as $\Gamma_i = \partial\Omega_i \setminus \partial\Omega$ and the whole interface Γ as $\Gamma = \cup\Gamma_i$. Let $R_{\Gamma_i} : \Gamma \to \Gamma_i$ be the canonical pointwise restriction which maps full vectors defined on Γ into vectors defined on Γ_i, and

Step 1. Reorder the unknowns so that the unknowns on the interface (B) are the last ones and that A_{II} has a block diagonal structure.

$$\begin{pmatrix} A_{II} & A_{IB} \\ A_{BI} & A_{BB} \end{pmatrix} \begin{pmatrix} u_I \\ u_B \end{pmatrix} = \begin{pmatrix} f_I \\ f_B \end{pmatrix}$$

Step 2. Solve the problem on the interface

$$Su_B = f_B - A_{BI}A_{II}^{-1}f_I, \text{ with } S = A_{BB} - A_{BI}A_{II}^{-1}A_{IB}$$

Step 3. Solve the problem for the interior unknowns

$$A_{II}u_I = f_I - A_{IB}u_B$$

(Fully parallelizable in a distributed environment.)

Algorithm 2: Algorithm of the Schur complement method.

let $R_{\Gamma_i}^T : \Gamma_i \rightarrow \Gamma$ be its transpose. For a matrix A arising from a finite-element discretization, the Schur complement matrix (6) can also be written as

$$S = \sum_{i=1}^{N} R_{\Gamma_i}^T S^{(i)} R_{\Gamma_i}, \tag{7}$$

where

$$S^{(i)} = A_{\Gamma_i}^{(i)} - A_{\Gamma_i i}A_{ii}^{-1}A_{i\Gamma_i} \tag{8}$$

is referred to as the local Schur complement associated with the subdomain Ω_i. $S^{(i)}$ involves sub-matrices from the local matrix $A^{(i)}$, defined by

$$A^{(i)} = \begin{pmatrix} A_{ii} & A_{i\Gamma_i} \\ A_{\Gamma_i i} & A_{\Gamma_i}^{(i)} \end{pmatrix}, \tag{9}$$

where $A^{(i)}$ is the local discretization of the problem on the subdomain Ω_i.

When implemented on a distributed environment, the matrix S is never fully assembled. In the case of iterative substructuring algorithms the interface is solved using a distributed Krylov solver. Conjugate gradient algorithm (CG) [23] is the method of choice for SPD systems and we select GMRES [29] for the unsymmetric systems. For all our numerical experiments, CG is used for SPD systems and GMRES for unsymmetric systems. More precisely we consider full GMRES as the condensed problem on the interface remains reasonably small so that the extra storage of the full Arnoldi basis is affordable. Furthermore, we observed [27] that this was the most robust unsymmetric solver for this class of problem.

In modern parallel numerical libraries [17, 18], the Krylov solvers are implemented using the reverse communication mechanism such that only three computational kernels have to be implemented by the user. Those kernels are: the dot product calculation, the matrix-vector product, applying the preconditioner to a vector.

3.2 Explicit iterative substructuring implementation

Implicit implementation

The efficient implementation of a Krylov method for solving (6) strongly depends on the implementation of the matrix-vector product with the Schur complement. For the classical approach

the matrices A_{ii} involved in Equation (8) are factorized by a sparse direct solver independently on each subdomain, consequently in parallel on a distributed computer. In this implicit approach, the factors of A_{ii} are used to perform the local matrix-vector products for the local Schur complement defined by Equation (8). This is done via a sequence of sparse linear algebra computations, namely a sparse matrix-vector product by $A_{i\Gamma_i}$, then sparse forward/backward substitution using the computed factors of A_{ii}, and finally a sparse matrix-vector product by $A_{\Gamma_i i}$. Then a communication phase exchanges information between neighboring subdomains. Once the convergence of the Krylov solver is obtained the factors are used once again in Step 3 of Algorithm 2 to compute the solution inside the subdomains.

We have implemented this algorithm with the parallel distributed sparse direct multifrontal solver MUMPS (MUltifrontal Massively Parallel Solver) [2, 4]. It is written in FORTRAN 90 and uses the new functionalities of this language (modularity, dynamic memory management) and results in an efficient and easy to use modern code. In this first part of the work, we only exploit the sequential feature of MUMPS to implement Algorithm 2 as previously described.

Explicit implementation

Among the few available parallel distributed direct solvers MUMPS offers a unique feature that is the possibility to compute the Schur complements using efficient sparse calculation techniques. Let A be the partitioned matrix

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix},$$

where A_{11} and A_{22} are two square matrices coupled by the two rectangular matrices A_{12} and A_{21}. The matrix $C = A_{22} - A_{21}A_{11}^{-1}A_{12}$ is called the Schur complement matrix associated with A_{22} in A. MUMPS can return a Schur complement to the user. The user must specify the list of indices associated with A_{22}. The software then provides both factorization of the A_{11} matrix and the explicit Schur matrix S. The Schur matrix is returned as a dense matrix. The partial factorization that builds the Schur matrix can also be used to solve linear systems associated with the matrix A_{11}.

We can see that, by the use of this feature on the local discretization matrices (9), we obtain the local Schur complement matrices (8) in addition to the factorization of the local matrices A_{ii}. Therefore, in the case of explicit iterative substructuring the local Schur complement matrices $S^{(i)}$ are computed explicitly using the Schur complement feature of MUMPS concurrently on each processor. There are several advantages to the explicit algorithm. In the implicit case, the core of the matrix-vector product needs two sparse triangular solves on the internal unknowns of each subdomain. In the explicit case, it consists in a single call to DGEMV, the dense level 2 BLAS matrix-vector subroutine on each subdomain on a block of size the number of unknowns on the interface of the subdomain. On the other hand, there are also some drawbacks for this method. First, the factorization step is longer as we have more floating operations to perform in order to compute the Schur complement. This method also implies additional storage to hold the local Schur complement as a dense matrix. When the size of the interface is small relative to the number of internal unknowns, the explicit method will be more efficient than the implicit one. This is usually the case for 2D problems like the one we are treating. For 3D problems, the storage/computation of the local Schur complement might not be affordable. Furthermore, even if the factorization step is longer, only a small number of Krylov iterations is usually required to make the explicit method more efficient than the implicit one as illustrated in the next paragraph.

Comparison

In order to demonstrate the advantage of the explicit approach, we consider the following model problem. This model problem consists in the discretized Poisson equation on a regular rectangular

grid. The equation is the following

$$\begin{cases} -\dfrac{\partial^2 u}{\partial x^2} - \dfrac{\partial^2 u}{\partial y^2} = f \text{ in } \Omega \subset \mathbb{R}^2, \\ u = g \text{ on } \partial\Omega, \end{cases} \tag{10}$$

where the domain Ω is a rectangle discretized by a uniform grid and decomposed into 16 similar subdomains containing around 160 000 degree of freedoms each (see Figure 2).

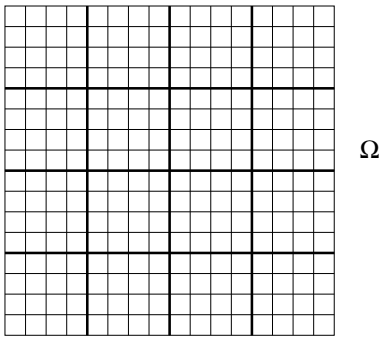

Figure 2: Regular discretization and decomposition into regular subdomains of a rectangular domain Ω (4×4 decomposition).

Using a Schur complement technique, the problem is then reduced to the solution of a SPD linear system. Each CG iteration requires a matrix-vector product that can be either performed explicitly, if the local Schur complement matrices are explicitly built, or implicitly otherwise. This 2D structured problem is similar to semiconductor problems concerning the ratio between the number of unknowns belonging to the interface and the number of unknowns belonging to the interior of the subdomains. Finally the patterns of the involved matrices are similar to those appearing in device modeling as the number of nonzero entries per row are comparable. In that respect, the behaviour of the sparse direct solver would also be very similar.

	Facto	Matvec
Implicit	10.19 s	1.60 s
Explicit	18.38 s	0.07 s

Table 1: Time for the factorization and of the matrix-vector product in the case of implicit and explicit iterative substructuring for a model problem (16 subdomains of size 400×400) on a SGI O2000.

In the first column of Table 1, we present the elapsed time required by MUMPS for the factorization of the internal subproblems, for the factorization and construction of the local Schur matrices, in the implicit and the explicit case respectively. In the explicit case, the local Schur complements are computed in addition to the factorizations of the internal subproblems. As expected, we see that this time is larger in the explicit situation. It is due to the additional cost for building the local Schur complement.

In the second column of Table 1, we display the elapsed time needed to perform one matrix-vector product by the Schur complement in the implicit and the explicit case. The use of the explicit method reduces the time spent in the matrix-vector product by a ratio of 22 compared with the implicit scheme. In the implicit case, two matrix-vector products by the sparse coupling blocks and a forward/backward substitution using the sparse factors of the internal problem are needed locally on each subdomain to compute the matrix-vector product by the local Schur complement. The matrix-vector product by the complete Schur matrix is then obtained by a communication

phase between neighboring subdomains. In the explicit case this communication phase is still the same, but the computation of the local matrix-vector product by the local Schur complement is obtained by a simple call to a level 2 BLAS subroutine that implements a dense matrix-vector product. In this latter case, the number of floating point operations is smaller and the access to the memory is more regular (i.e. dense versus sparse calculation), this explains the large decrease of computing time observed when using the explicit matrix-vector product.

Except during the factorization and the matrix-vector product, the operations performed by CG iterations using either the implicit or the explicit algorithms are identical. It is then enough to consider the computational time of these two kernels to evaluate their possible advantages when used in Krylov iterations. In this particular case, only 6 matrix-vector products, and therefore 6 Krylov iterations, are required to amortize the additional cost of the explicit calculation of the local Schur matrix (that includes the factorization). This number of 6 iterations is relatively small for solving a linear system of size about ten thousands (i.e. the size of the Schur complement system for such a decomposition).

Another attractive feature of the explicit approach is that it enables us to easy investigate the use of some preconditioners that would be difficult to implement if the local Schur complements were only implicitly known. The drawback is the additional memory cost due to the storage of the dense local Schur complement matrices. However for 2D computations this extra cost remains affordable since, as earlier mentioned, the main bottleneck is the elapsed time in semiconductor simulations.

In the sequel for all the experiments on semiconductor devices, we only consider iterative substructuring methods that implement the explicit computation of the local Schur complement matrices. Moreover, the results reported in Section 4 highlight other advantages of this choice.

3.3 Preconditioners for the Schur complement

In iterative substructuring the interface problem is solved using a Krylov solver. The possible weakness of iterative methods is their potential lack of robustness. However both the efficiency and the robustness can be improved by using preconditioning techniques.

The preconditioners presented in this section have been originally proposed in [11, 12, 13]. In order to describe these preconditioners, we need first to define a partition of B, the set of edges of the discretization belonging to the interface between the subdomains. Let U be the algebraic space of vectors where the Schur complement is defined and $(U_i)_{i=1,p}$ a set of subspaces of U such that

$$U = U_1 + U_2 + \cdots + U_p.$$

Let R_i be the canonical pointwise restriction from U to U_i. Its transpose extends grid functions in U_i by zero to the rest of U. Using the above notation, we can define a wide class of block preconditioners by:

$$M_{loc} = \sum_{i=1}^{p} R_i^T M_i^{-1} R_i, \tag{11}$$

where

$$M_i = R_i S R_i^T. \tag{12}$$

We note that, if $U = U_1 \oplus .... \oplus U_n$, then M_{loc} is a block Jacobi preconditioner. Otherwise, M_{loc} is a block diagonal preconditioner with an overlap between the blocks as $U_i \cap U_j \neq \emptyset$. In this case, the preconditioner can be viewed as an algebraic additive Schwarz preconditioner for the Schur complement.

The preconditioners are requested to be efficient on parallel distributed memory platforms. Therefore, we do mainly consider subspaces U_i that involve information mainly stored in the local

memory of the processors; that is information associated with only one subdomain and its closest neighbors. We present two decompositions of U :

1. each common interface $E_k = \partial\Omega_i \cap \partial\Omega_j$ between two subdomains of the decomposition giving rise to the block Jacobi preconditioner;
2. each interface Γ_i of the subdomains giving the subdomain preconditioner.

Block Jacobi preconditioner

We define the common interface E_i between subdomain Ω_j and subdomain Ω_l as the set of edges belonging to $(\partial\Omega_j \cap \partial\Omega_l)$. The set B can be partitioned into m common interfaces $E_i, i \in \{1, \ldots, m\}$, $B = (\bigcup_{i=1}^{m} E_i)$. For each common interface E_i we define $R_i \equiv R_{E_i}$ as the standard restriction from B to E_i. Its transpose extends vectors in E_i by zero to the rest of the interface. Thus, $M_i = R_{E_i} S R_{E_i}^T = S_{ii}$. Using the above notation we define the following local preconditioner by

$$M_{bJ} = \sum_{E_i} R_{E_i}^T S_{ii}^{-1} R_{E_i}. \tag{13}$$

This preconditioner aims at capturing the interaction between neighboring edges within the same common interface between two subdomains. This preconditioner is the straightforward block Jacobi that is well-known to be efficiently parallelizable.

Additive Schwarz preconditioner

In this alternative [12], we try to exploit all the information available on each subdomain and we associate each subspace U_i with the entire boundary Γ_i of subdomain Ω_i. Here, we have $R_i \equiv R_{\Gamma_i}$. The local matrix $M_i = \bar{S}^{(i)}$ is called the assembled local Schur complement and corresponds to the restriction of the complete Schur matrix to the interface of the subdomain Ω_i. This splitting $(U_i)_i$ is not a direct sum of the space U and we have introduced some overlap between the blocks defining the subdomain preconditioner M_{AS}. This preconditioner can be written as:

$$M_{AS} = \sum_{i=1}^{N} R_{\Gamma_i}^T (\bar{S}^{(i)})^{-1} R_{\Gamma_i} \ . \tag{14}$$

We refer to this preconditioner as the M_{AS} preconditioner because it can be viewed as an Additive Schwarz preconditioner for the Schur complement system. One advantage of using the assembled local Schur complements instead of the local Schur complements (like in the Neumann-Neumann case [9, 16]) is that in the SPD case the assembled Schur complements cannot be singular (as S is not singular).

We may remark that these two preconditioners are easy to build when an explicit storage scheme is selected for the Schur complement. Particularly, the additive Schwarz preconditioner may be difficult to obtain when the local Schur complements are only implicitly known.

4 Embedding the iterative substructuring solvers within the semiconductor simulation code

In this section we highlight the numerical difficulties faced when plugging the explicit iterative substructuring algorithm, presented in Section 3, in the semiconductor device simulation code, presented in Section 2. All the numerical experiments were performed in double precision arithmetic on a SGI Origin 2000. The set of selected test cases correspond to two different meshes defined on the same geometry of the heterojunction device depicted in Figure 1. The main characteristics of those examples are summarized in Table 2. The stopping criterion for the three embedded schemes are the following from the outest to the innest:

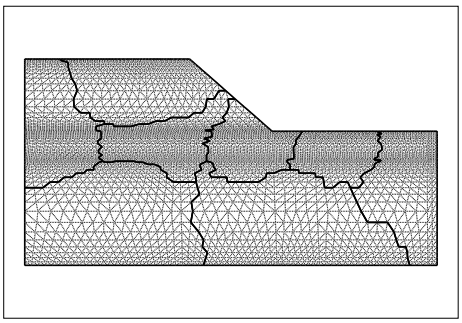

Figure 3: Triangular mesh with 5194 elements partitioned into 8 subdomains.

	Mesh size	# subdomains	Size Schur
Medium 16	365 701 edges	16	2273
Large 32	1 214 758 edges	32	5180

Table 2: Main characteristics of the test problems.

1. time dependent solver: the Euler scheme is stopped when the relative one-norm of the difference between two successive iterates is lower than a prescribed threshold,

2. the nonlinear solver: the Newton-Raphson scheme is stopped when the relative one-norm of the gradient of the nonlinear function is smaller that a prescribed threshold,

3. the linear solver : the Krylov solvers are stopped when the ratio between the two-norm of the residual divided by the two-norm of the right hand side becomes smaller than a prescribed threshold. Notice that this widely used stopping criterion can be viewed as a norm-wise backward error where only perturbations on the right hand side are considered [14, 34].

4.1 Preconditioning and scaling

In the case of the semiconductor application, the numerical difficulty is associated with the very high dynamics of the values to compute that leads to very badly scaled problems. Therefore we have investigated several scaling strategies. Even though row and/or column scaling exist we only consider symmetric techniques (i.e. row and column scaling) that are the only ones that can be used for SPD systems. In addition, for most of the linear systems the largest entries are located on the diagonal. We consider the symmetric diagonal scaling for the linear system $Ax = b$ defined as follows. The system $Ax = b$ is replaced by $DADy = Db,\ x = Dy$ where $D = (d_{ii})$ is the scaling matrix defined by $d_{ii} = \sqrt{|a_{ii}|}$.

We also consider a symmetric diagonal scaling on the Schur system $Su = g$. It can be shown that in the case of iterative substructuring the scaling on A is equivalent to scale the Schur complement system with the diagonal entries of A corresponding to the interface unknowns. In that respect, only using a scaling on A might be inappropriate. We may also remark that the scaling on S is easy to apply only in the case of explicit iterative substructuring.

The preconditioners considered are block Jacobi denoted by M_{bJ} and the additive Schwarz preconditioner denoted by M_{AS}. In Table 3 we display the total number of Newton steps (and therefore the total number of linear systems solved) required to obtain the stationary state of the simulation. We consider a diagonal scaling on the original systems or a diagonal scaling on the corresponding Schur complements. The "$\times$" symbol indicates that the simulation cannot be completed because the nonlinear scheme does not converge after 300 Newton steps. For the sake of completeness, the number of Newton steps needed to obtain the steady state with a direct

substructuring method is also reported. We consider the *Medium 16* test case. The required accuracy for the Newton solvers is set to 10^{-7} and the required accuracy for the Krylov solvers is set to 10^{-11}.

	no scaling	diag. on A	diag. on S
No prec.	$\times$	$\times$	$\times$
M_{bJ}	$\times$	$\times$	182
M_{AS}	$\times$	$\times$	176
Direct	173		

Table 3: Total number of Newton steps during a simulation on a mesh with 356701 edges decomposed into 16 subdomains. The varying parameters are the preconditioner and the scaling strategy. $\times$ means that the nonlinear scheme does not converge.

Table 3 shows that a combination of a preconditioner for S and a scaling on S is needed to ensure the convergence of the nonlinear scheme. Actually, the Schur complement matrices are effectively very badly scaled with values varying with more than 30 orders of magnitude. For example a Schur matrix deriving from one system associated with the continuity equation for the electrons have nonzero entries varying from 10^{-5} to 10^{25}. As already mentioned, the diagonal scaling on the original system is not efficient.

Even though, the scaling remove some ill-conditioning, the Schur complement matrices remain difficult to solve by unpreconditioned Krylov iterations. In that respect a good preconditioner has to be combined with the scaling to ensure the nonlinear convergence. Concerning the comparison between the two local preconditioners, we can see that the two preconditioners give comparable results concerning the nonlinear convergence. Futhermore, both iterative solvers lead to a nonlinear convergence comparable to the one obtained with a direct linear solver.

Finally to emphasize how crucial the scaling is for the nonlinear convergence, we mention that without any scaling and with the M_{AS} preconditioner, the nonlinear scheme does not converge while each linear system converges in norm-wise backward error down to 10^{-11}. Therefore, scaling is the key parameter to ensure the robustness of the complete numerical method. This also illustrates that the classical stopping criterion based on the norm-wise backward error is not always a faithful criterion to assess the actual quality of the computed solution. We should mention some promising ongoing research works [5, 6], that intend to relate the stopping criterion of the Krylov solver with some intrinsic properties of the underlying PDEs. Extending such techniques to the device modeling framework would deserve to be investigated but is out of the scope of this paper.

4.2 Comparison of block Jacobi and additive Schwarz preconditioners

Here we consider the comparison of the two preconditioners defined in Section 3.3 for the Schur complement system. In all the experiments the symmetric diagonal scaling is applied on the Schur system and the stopping criteria are set to 10^{-7} for the Newton solvers and 10^{-11} for the Krylov solvers. In Table 4 we display the total number of Newton iterations required, the average number of Krylov iterations required to solve one SPD Schur system or one unsymmetric Schur system and the total elapsed time in seconds to complete the simulation.

For *Medium 16* it is difficult to tell whether M_{bJ} or M_{AS} is the best. The number of Newton steps is almost the same. The average number of Krylov iterations per linear system is larger with M_{bJ} than with M_{AS}. So the time spent in matrix-vector products and in dot products is larger in the case of M_{bJ}. On the other hand, the cost of construction and application of the preconditioner is larger for M_{AS}. Actually, the average time needed to build the M_{AS} preconditioner is 0.26 seconds while the average time needed to build M_{bJ} is 0.05 seconds. The average time needed to apply the preconditioner M_{bJ} is 2.0 milliseconds while the average time needed for M_{AS} is 8.3 milliseconds. Finally, if we take into account the variability in the time measurements, we can

	Medium 16		*Large 32*	
	M_{bJ}	M_{AS}	M_{bJ}	M_{AS}
Newton its	182	176	228	175
iter CG	38	24	68	32
iter GMRES	34	24	94	34
time (s)	788	809	2892	1654

Table 4: Comparison of the preconditioners M_{bJ} and M_{AS} on two test cases.

say that in this case M_{AS} and M_{bJ} are equivalent. M_{AS} is more expensive but enables a faster convergence of the linear solvers than M_{bJ} that is cheaper in computation but less numerically efficient.

However, if we turn to Large 32, some differences appear. With M_{bJ} the number of Newton steps is larger than with M_{AS}, about 30 % more iterations. Moreover, the linear convergence is also deteriorated. Concerning the average number of Krylov iterations needed to solve a linear system, the gap between M_{bJ} and M_{AS} is much larger than in the case of *Medium 16*. The larger number of matrix-vector products is no longer mitigated by the cheaper construction and application of the M_{bJ} preconditioner. So the preconditioner M_{AS} is more robust than the preconditioner M_{bJ} and leads to better results. Even if M_{AS} and M_{bJ} are equivalent in many cases concerning computational time, the choice of M_{AS} ensures better condition number for the preconditioned matrix and therefore a better stability of the nonlinear schemes. In that respect M_{AS} is much more robust than M_{bJ}. This fact illustrates another advantage of the explicit iterative substructuring as M_{AS} is difficult to build when the local Schur complements are only implicitely known.

4.3 Influence of the accuracy of the linear solvers

Because the linear solvers are embedded in a nonlinear scheme, the accuracy required for the linear solvers may influence the convergence of the Newton method. We consider a simulation associated with the problem *Medium 16*. In this experiment the scaling and the preconditioner are frozen parameters. The additive Schwarz preconditioner M_{AS} is used combined with the symmetric diagonal scaling on the Schur complement. The varying parameter is the accuracy required for the Krylov solvers. In Table 5 we depict the number of Newton steps needed to obtain the steady state for the simulations.

	ε_{Krylov}					
	10^{-5}	10^{-7}	10^{-9}	10^{-11}	10^{-13}	10^{-15}
Newton Steps	$\times$	214	182	176	176	176

Table 5: Influence of the required accuracy of the linear solvers on the nonlinear convergence.

We see that the number of Newton steps increases when the threshold used for the stopping criterion of the Krylov solvers is relaxed. With $\epsilon_{Krylov} = 10^{-5}$ the nonlinear scheme does not converge any more. This degradation can be explained by the inequality :

$$\| \text{ Forward error } \| < \text{ Condition number} \times \text{ Backward error.} \tag{15}$$

In Inequality (15) the forward error is the distance between the computed solution and the exact solution of the problem in a certain norm $\| \, . \, \|$, the condition number is the quantity that indicates the sensitivity of the problem to perturbations on its data measured in the same norm $\| \, . \, \|$. The stopping criterion for the Krylov solvers is based on backward error analysis. When the stopping criterion for the Krylov solver is relaxed, the backward error for the Schur complement system increases and the forward error on the solution at the interface might increase as well. The

degradation of the solution accuracy at the interface propagates to the whole domain, and finally affects the convergence of the Newton schemes.

If the accuracy required for the Krylov solver is relaxed, then the number of Newton steps increases. At the same time, the number of iterations of the Krylov solvers decreases. So, each Newton step becomes less expensive and the overall simulation might become computationally cheaper. In Table 6 we consider the case of a backward error of 10^{-11}. We display the average time in seconds required to solve one Schur complement system and one complete system for simulations *Medium 16* and *Large 32*. One can see in Table 6 that the solution of the Schur system

	Medium 16	*Large 32*
Average time for $Su = g$	0.77	2.13
Average Time for $Ax = b$	3.31	7.35

Table 6: Average time in seconds required to solve one Schur system and one complete system for a threshold for the backward error set to 10^{-11}.

is less than a third of the time required to solve the complete system as a large part of the time is spent in computing the local Schur complement matrices. Therefore we can say that it is better to reduce the number of Newton iterations than to reduce the number of Krylov iterations. This explains why we set the stopping criterion threshold for the Krylov solver at a very low value.

5 Comparison with direct solvers

5.1 Direct substructuring

In explicit iterative substructuring the interface problem is solved by a distributed Krylov iterative method. Direct substructuring consists in replacing the Krylov solver by a parallel direct solver. In our implementation based on the MUMPS software the Schur complement matrix is unassembled and distributed over all the different processors as the local Schur complement matrices. Using another feature of MUMPS that is able to solve linear systems where the associated matrices are distributed, we can then solve the global Schur complement system using an additional parallel instance of MUMPS. For that latter parallel implementation we use MUMPS in two different places. First we call in parallel several sequential instances of MUMPS to build the local Schur complement matrices, then we call a single parallel instance of MUMPS to solve the distributed unassembled global Schur complement system.

Another possibility is to solve directly the original linear system A once all the local sub-matrices associated with the discretization on each subdomains have been computed. These local matrices can also be passed to MUMPS as an unassembled distributed matrix; the software will then factorize it in parallel. From a software point of view there are several differences between direct substructuring and an application of MUMPS to the matrix A distributed among the processors. In direct substructuring, the symbolic analysis phase is distributed for the internal problems but during the factorization phase load balancing is no longer possible and numerical pivoting is limited to the local subdomains.

In Table 7 we display the total number of Newton steps to obtained the steady state and the total elapsed time in seconds to complete the simulation. We compare the results obtained with the two parallel direct methods that are direct substructuring (referred to as *Dss*) and the use of MUMPS on the complete system as a multifrontal solver with the distributed entries option (referred to as *Sparse Direct*).

We can see that the two methods are numerically equivalent with respect to the number of Newton steps. But if we turned to the simulation time we can see that *Dss* is more efficient than *Sparse direct* (notably for *Large 32*). We will not enter into the details of the comparison between the two direct solvers but we can remark that the choice of the ordering defined during

	Medium 16		*Large 32*	
	Dss	*Sparse Direct*	*Dss*	*Sparse Direct*
Newton steps	173	173	166	166
Simulation time (s)	941	1022	2527	3995

Table 7: Performance of direct methods for the two simulations *Medium 16* and *Large 32*.

the symbolic analysis phase influences strongly the performance of the factorization and of the solve phase. For *Sparse Direct*, the ordering on which the factorization is based is an approximate minimum degree algorithm (AMD) [1] applied to the complete matrix. For *Dss*, it is a combination of nested dissection-like (corresponding to the mesh partitioning) and minimum degree within the subdomains and at the root on the complete Schur complement (see Section 5.1). For more details on the influence of the ordering on the performance of multifrontal solvers we refer to [3] and to [27] in the case of this specific application.

We can indicate that in the case of the semiconductor application the direct substructuring algorithm is an efficient direct solver. Moreover it gives us a framework to make a fair comparison with the iterative approaches.

5.2 Comparison of iterative and direct substructuring

In the case of iterative substructuring, the method that gives the most satisfactory results is explicit iterative substructuring with the symmetric diagonal scaling and the M_{AS} preconditioner for the Schur complement system. The accuracy required for the Krylov solver is set to 10^{-11}. In the case of parallel direct solvers, the results of Section 5.1 indicate that *Dss* is more attractive than *Sparse Direct* for this application. Moreover, the choice of *Dss* makes the comparison easier with iterative substructuring. The only difference between M_{AS} and *Dss* is the way the complete Schur system is solved. In the first case, the Schur system is solved by a distributed iterative Krylov solver and in the second case by a distributed direct multifrontal solver.

In Table 8, we display the total number of Newton steps and the total time in seconds required to obtain the stationary state of the simulation. We also display the average time in seconds required to solve one Schur complement system. We can see that *Dss* requires less Newton steps

	Medium 16		*Large 32*	
	M_{AS}	*Dss*	M_{AS}	*Dss*
Newton	176	173	175	166
time (s)	809	1140	1654	2527
Average Time for $Su = g$	0.77	3.31	2.13	7.34

Table 8: Total number of Newton steps, total simulation time in seconds, average time to solve one Schur system.

than M_{AS} to obtain the steady state. On the other hand, the total simulation time is larger with *Dss* than with M_{AS}. For our applications, this is due to the fact that solving the distributed Schur complement with an iterative solver is more efficient than solving it with a distributed direct solver.

Concerning memory requirements the iterative solver remains as expensive as *Dss* due to the additional storage cost of the additive Schwarz preconditioner (identical to the one of the Schur complement itself). However, we may remark again that this cost is affordable for 2D problems where the main bottleneck remains the computational elapsed time.

6 Conclusion and prospectives

In this study we have presented a robust linear iterative scheme for the solution of numerically difficult problems. This solution has been implemented on parallel distributed computers and a fully distributed code has been developed. A simulation that requires 24 hours of computational time with the initial code only requires a few minutes to be completed using the parallel implementation. This tremendous reduction of elapsed time is only partially due to the use of parallel computers but mainly induced by the use of efficient modern techniques for solving large sparse linear systems. We should also note that this work enables us to validate the some aspects of the modelization as we performed some numerical experiments out of reach with the preliminary version of the software. The parallel code has been successfully combined with an adaptative mesh refinement technique so that more reliable and accurate solution can be computed within moderate CPU costs [25].

Nevertheless, some problems are still open. As it can be seen in Table 9, where we display the average number of CG and GMRES iterations and the total simulation time in seconds, the M_{AS} preconditioner does not exhibit a good numerical scalability. Indeed, the number of Krylov

	Number of subdomains		
	8	16	32
iter CG	12	24	44
iter GMRES	14	24	42
Simulation time (s)	1295	809	584
Speed-Up	1	1.6	2.22

Table 9: Numerical and performance results for a simulation based upon an mesh with 365 701 edges decomposed into 8, 16 or 32 subdomains.

iterations increases almost linearly with respect to the number of subdomains. This might not be satisfactory from a linear algebra point of view. However from a computational view point the situation looks better. As the number of subdomains remains smaller than 32, this lack of numerical scalability of the preconditioner is compensated by the better efficiency of the sparse computations inside the subdomains (that becomes smaller when the number of domain is increased). Consequently the non scalability of the preconditioner is not too penalizing as for 2D simulations there is no need to go much beyond one million degree of freedom; then 32 processors is a reasonable maximum number of processors to use for a simulation.

Nevertheless, we should mention that we have performed experiments [27] with several classical two-level preconditioners [11, 12, 13], including the Balanced Neumann-Neumann preconditioner [24, 32, 33] developed at Parallab [7] for SPD systems. Unfortunately none of them enables us to obtain a satisfactory numerical scalability. Finally, we indicate that a preliminary study of a two-level preconditioner technique based upon spectral properties of the locally preconditioned matrix [10] can be found in [27]. The first results seem to be promising and would deserve more extensive experimentations to assess the effectiveness and robustness of this two-level preconditioner in the framework of device modeling.

Lastly, we would like to conclude by emphasizing that, in our context, the iterative solvers are more efficient than the direct ones, but the tuning of the various parameters that govern their behaviour might be difficult. Even though the use of MUMPS as a black box direct solver is the least efficient method concerning computational time but it was the easiest to interface with the semiconductor code. This is another illustration that sparse direct solver are mature enough to be implemented in black box packages where some additional effort still need to be developed to bring hybrid iterative solvers at the same level from a software point of view.

Acknowledgments

We would like to thank Parallab (Bergen, Norway) and CINES (Montpellier, France) for providing us with an access to their SGI O2000 platform.

References

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

[2] P. R. Amestoy, I. S. Duff, J. Koster, and J.-Y. L'Excellent. `MUMPS`: a general purpose distributed memory sparse solver. In A. H. Gebremedhin, F. Manne, R. Moe, and T. Sørevik, editors, *Proceedings of PARA2000, the Fifth International Workshop on Applied Parallel Computing, Bergen, June 18-21*, pages 122–131. Springer-Verlag, 2000. Lecture Notes in Computer Science 1947.

[3] P. R. Amestoy, I. S. Duff, J. Koster, and J.-Y. L'Excellent. A fully asynchronous multifrontal solver using distributed dynamic scheduling. *SIAM Journal on Matrix Analysis and Applications*, 23(1):15–41, 2001.

[4] P. R. Amestoy, I. S. Duff, and J.-Y. L'Excellent. Multifrontal parallel distributed symmetric and unsymmetric solvers. *Comput. Methods Appl. Mech. Eng.*, pages 501–520, 2000.

[5] M. Arioli. A stopping criterion for the conjugate gradient algorithm in a finite element method framework. Technical Report #1179, IAN, 2000.

[6] M. Arioli, D. Loghin, and A. J. Wathen. Stopping criteria for iterations in finite element methods. Technical Report In preparation, CERFACS, Toulouse, France, 2002.

[7] P. E. Bjørstad, J. Koster, and P. Krzyżanowski. Domain decomposition solvers for large scale industrial finite element problems. In *PARA2000 Workshop on Applied Parallel Computing*. Lecture Notes in Computer Science 1947, Springer-Verlag, 2000.

[8] A. El Boukili. *Analyse mathématique et simulation numérique bidimensionnelle des équations des semi-conducteurs par l'approche éléments finis mixtes*. PhD thesis, Université Paris VI, Paris, France, 1995.

[9] J.-F. Bourgat, R. Glowinski, P. Le Tallec, and M. Vidrascu. Variational formulation and algorithm for trace operator in domain decomposition calculations. In Tony Chan, Roland Glowinski, Jacques Périaux, and Olof Widlund, editors, *Domain Decomposition Methods*, pages 3–16, Philadelphia, PA, 1989. SIAM.

[10] B. Carpentieri, I.S. Duff, and L. Giraud. A class of spectral two-level preconditioners. Technical Report TR/PA/02/55, CERFACS, Toulouse, France, 2002. Preliminary version of the paper to appear in SISC.

[11] L. M. Carvalho. *Preconditioned Schur complement methods in distributed memory environments*. PhD thesis, INPT/CERFACS, France, october 1997. TH/PA/97/41, CERFACS.

[12] L. M. Carvalho, L. Giraud, and G. Meurant. Local preconditioners for two-level non-overlapping domain decomposition methods. *Numerical Linear Algebra with Applications*, 8(4):207–227, 2001.

[13] L. M. Carvalho, L. Giraud, and P. Le Tallec. Algebraic two-level preconditioners for the schur complement method. *SIAM J. Scientific Computing*, 22(6):1987–2005, 2001.

[14] F. Chaitin-Chatelin and V. Frayssé. *Lectures on Finite Precision Computations.* SIAM, Philadelphia, 1996.

[15] T. F. Chan and T. P. Mathew. Domain decomposition algorithms. In *Acta Numerica 1994*, pages 61–143. Cambridge University Press, 1994.

[16] Y.-H. De Roeck and P. Le Tallec. Analysis and test of a local domain decomposition preconditioner. In Roland Glowinski, Yuri Kuznetsov, Gérard Meurant, Jacques Périaux, and Olof Widlund, editors, *Fourth International Symposium on Domain Decomposition Methods for Partial Differential Equations*, pages 112–128. SIAM, Philadelphia, PA, 1991.

[17] V. Frayssé and L. Giraud. A set of conjugate gradient routines for real and complex arithmetics. Technical Report TR/PA/00/47, CERFACS, Toulouse, France, 2000. Public domain software availabe on www.cerfacs.fr/algor/Softs.

[18] V. Frayssé, L. Giraud, and S. Gratton. A set of GMRES routines for real and complex arithmetics. Technical Report TR/PA/97/49, CERFACS, Toulouse, France, 1997. Public domain software availabe on www.cerfacs.fr/algor/Softs.

[19] L. Giraud, J. Koster, A. Marrocco, and J.-C. Rioual. Domain decomposition methods in semiconductor device modeling. In N. Debit, M. Garbey, R. Hoppe, J. Périaux, D. Keyes, and Y. Kuznetsov, editors, *Domain decomposition methods in science and engineering.*, pages 469–476. CIMNE, 2001.

[20] W. Gropp, E. Lusk, and A. Skjellum. *Using MPI: Portable Parallel Programming with the Message-Passing Interface.* MIT press, 1994.

[21] W. D. Gropp, E. Lusk, N. Doss, and A. Skjellum. A high-performance, portable implementation of MPI message passing interface standard. *Parallel Computing*, 22(6), 1996.

[22] F. Hecht and A. Marrocco. Mixed finite element simulation of heterojunction structures including a boundary layer model for the quasi-fermi levels. *COMPEL*, 13(4):757–770, 1994.

[23] M. R. Hestenes and E. Stiefel. Methods of conjugate gradients for solving linear system. *J. Res. Nat. Bur. Stds.*, B49:409–436, 1952.

[24] J. Mandel. Balancing domain decomposition. *Comm. Numer. Meth. Engrg.*, 9:233–241, 1993.

[25] A. Marrocco. Simulations numériques de dispositifs électroniques via éléments finis mixtes, adaptation de maillages et décomposition de domaine. Technical Report In preparation, INRIA.

[26] A. Quarteroni and A. Valli. *Domain decomposition methods for partial differential equations.* Numerical mathematics and scientific computation. Oxford science publications, Oxford, 1999.

[27] J.C. Rioual. *Solving linear systems for semiconductor device simulations on parallel distributed computers.* PhD thesis, CERFACS, Toulouse, France, 2002.

[28] Y. Saad. *Iterative Methods for Sparse Linear Systems.* PWS Publishing Company, 1996.

[29] Y. Saad and M. H. Schultz. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM J. Sci. Stat. Comp.*, 7:856–869, 1986.

[30] H. A. Schwarz. Ueber einen Grenzübergang durch alternierendes Verfahren. *Vierteljahrsschrift der Naturforschenden Gesellschaft in Zürich*, 15:272–286, May 1870.

[31] B. F. Smith, P. E. Bjørstad, and W. Gropp. *Domain Decomposition: Parallel Multilevel Methods for Elliptic Partial Differential Equations.* Cambridge University Press, 1996.

[32] P. Le Tallec, J. Mandel, and M. Vidrascu. Balancing domain decomposition for plates. In *Domain Decomposition Methods in Scientific and Engineering Computing: Proceedings of the Seventh International Conference on Domain Decomposition*, volume 180 of *Contemporary Mathematics*, pages 515–524, Providence, Rhode Island, 1994. American Mathematical Society.

[33] P. Le Tallec, J. Mandel, and M. Vidrascu. A Neumann-Neumann domain decomposition algorithm for solving plate and shell problems. *SIAM Journal on Numerical Analysis*, 35(2):836–867, April 1998.

[34] J. H. Wilkinson. *The Algebraic Eigenvalue Problem*. Oxford Science Publications, 1965.