IADIS International Journal on Computer Science and Information Systems
Vol. 1, No. 2, pp. 76-87
ISSN: 1646-3692

FREE DISTRIBUTION PARALLEL SPARSE SOLVERS. APPLICATION TO LAMBDA MODES EQUATION

Omar Flores Sánchez[1,2] and Vicente E. Vidal Gimeno[1]
[1]*Universidad Politécnica de Valencia, Departamento de Sistemas y Computación—DSIC, Camino de Vera s/n 46022, Valencia, España*
[2]*Instituto Tecnológico de Tuxtepec, Departamento de Sistemas y Computación—DSC, Av. Dr. Victor Bravo Ahuja s/n, Col. 5 de Mayo, A.P. 69, C.P.68300, Tuxtepec, Oaxaca, México*

ABSTRACT

This paper discusses how High Performance Computing can help to solve Engineering problems, where it is necessary to reduce the execution time spent in the solution of sparse linear systems. We have chosen three free-distribution numerical parallel libraries called PETSc (Portable, Extensible Toolkit for Scientific Computation) [Satish2001], pARMS (Parallel Algebraic Recursive Multilevel Solver) [Saad2003], and one direct sparse solver named SuperLU [Demmel2003]. These libraries have been applied to a realistic test case of Nuclear Engineering where it is necessary to solve efficiently very-large sparse linear systems to study steady-state neutron diffusion processes. Numerical experiments have shown the effectiveness of using parallel and distributed computing.

KEYWORDS

Parallel Computing, Krylov Subspace Methods, Very-Large Sparse Linear Systems, Lambda Modes Equation.

1. INTRODUCTION

Physical phenomena are often modelled by equations that relate several partial derivatives of physical quantities, such as forces, momentums, velocities, energy, temperature, etc. These equations rarely have a closed-form (explicit) solution. The Partial Differential Equations (PDEs) constitute the biggest source of sparse matrix problems. The typical way to solve such equations it is to discretize them, i.e., to approximate them by equations that involve a finite number of unknown. The matrices that arise from these discretizations are generally large and sparse, i.e., they have very few nonzero entries. There are several different ways to discretize a

Partial Differential Equation. The simplest method uses finite difference approximations for the partial differential operators. The Finite Element Method replaces the original function by a function which has some degree of smoothness over the global domain, but which is piecewise polynomial on simple cells, such as small triangles or rectangles. In between these two methods, there are a few conservative schemes called Finite Volume Methods, which attempt to emulate continuous conservation laws of physics.

Traditionally, direct methods [Duff1986][Demmel2003] have been used for solving linear systems of equations due to their robustness and predictable behaviour. However, iterative methods [Saad1996] have shown a good competency when they are combined with preconditioning techniques and Krylov subspace iterations, giving rise to efficient and simple general purpose procedures.

The main objective of this work, it is to show the advantages of using High Performance Computing tools such as numerical parallel libraries and PCs clusters to accelerate computing processes in Engineering problems. In particular, we have studied the PETSc, pARMS and SuperLU parallel libraries applied to the solution of the linear systems of equations related to the lambda modes equation that appears in stability and security analysis of nuclear reactors. In order to attain our objective, we have made numerical experiments with the above libraries. They are analyzed and evaluated from speedup and efficiency [Kumar1994] points of view.

The rest of this paper is organized as follows. Section 2 gives a general overview of the methods and preconditioners contained into PETSc and pARMS, as well as some important aspects of SuperLU. The realistic test case is covered in Section 3. Section 4 is devoted to parallel numerical experiments. Some conclusions are drawn in Section 5.

2. SPARSE SOLVERS

At the moment, many modern computational problems that arise in science and engineering should efficiently utilize the combined power of multi-processor computer architectures and effective algorithms. For many large-scale applications, solving large sparse linear systems is the most intensive computational task. The important criteria for a suitable solver include numerical efficiency, robustness, and good parallel performance. There is a limited selection of general-purpose sparse solvers. Among the implementations that contain general purpose solvers are PETSc and pARMS. These tools implement iterative solvers in their majority. By other hand, SuperLU is a general purpose library for the direct solution of large, sparse, nonsymmetric systems of linear equations on high performance machines.

2.1 Review of PETSc

The Portable, Extensible Toolkit for Scientific Computation (PETSc) is a suite of data structures and routines that provide the building blocks for the implementation of large-scale application codes on parallel (and serial) computers. PETSc uses the MPI [Groupp1994] standard for all message-passing communication.

Some of the PETSc modules deal with vectors, matrices (generally sparse), distributed arrays (useful for parallelizing regular grid-based problems), Krylov subspace methods, preconditioners including multigrid and sparse direct solvers, etc.

Figure 1 illustrates the library's hierarchical organization, which enables users to employ the level of abstraction that is most appropriate for a particular problem.

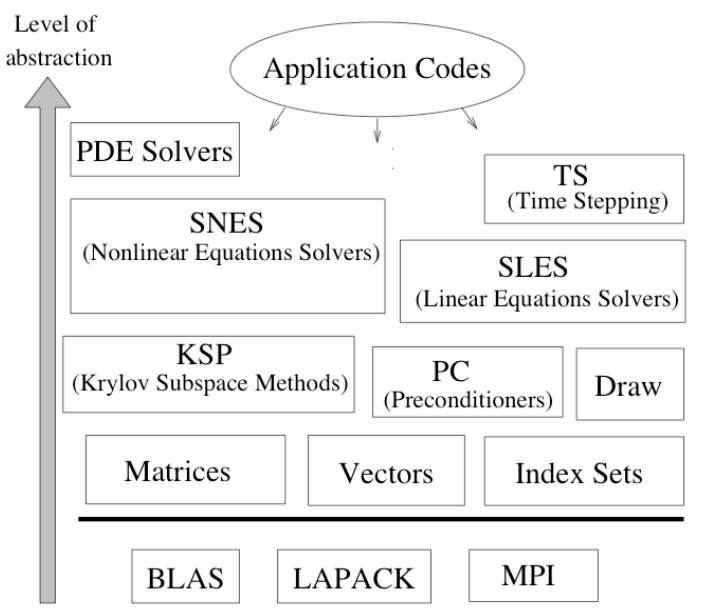

Figure 1. Organization of the PETSc libraries.

Table 1 shows the most popular iterative methods contained in PETSc. The preconditioners that have been applied to the real test case are Jacobi and Block-Jacobi.

Table 1. PETSc iterative methods.

Method	PETSc option
Conjugate Gradient	KSPCG
Bi-Conjugate Gradient	KSPBICG
Generalized Minimal Residual	KSPGMRES
BiCGSTAB	KSPBCGS
Conjugate Gradient Squared	KSPCGS
Transpose-Free Quasi-Minimal Residual	KSPTFQMR
Conjugate Residual	KSPCR

2.2 Review of pARMS

The Parallel Algebraic Recursive Multilevel Solver (pARMS) offers a suite of distributed-memory iterative accelerators and local preconditioners targeting the solution of general sparse linear systems. Multi-level Schur complement techniques available in pARMS, they are based on techniques which exploit block independent sets.

Table 2 shows the available preconditioners in pARMS, where add_X indicates an Additive Schwarz preconditioner with local preconditioner X, where X can be ILU0, ILUT or ARMS. lsch_X indicates left Schur complement preconditioner with X as local preconditioner. rsch_X indicates right Schur complement preconditioner with X as local preconditioner. sch_gilu0 stands for distributed ILU0 preconditioner on interface nodes.

sch_sgs stands for distributed Gauss-Seidel preconditioner on interface nodes. The accelerators employed during the experiments were a distributed version of flexible GMRES (fgmresd), a distributed version of deflated GMRES (dgmresd) and a distributed version of bi-CG stabilized (bcgstabd).

Table 2. pARMS preconditioners.

Schwarz Additive	Schur Complement based	Schur Complement based with enhancements
add_ilu0	lsch_ilu0	sch_gilu0
add_ilut	lsch_ilut	sch_sgs
add_iluk	lsch_arms	
add_arms	rsch_ilu0	
	rsch_ilut	
	rsch_iluk	
	rsch_arms	

2.3 Review of SuperLU

SuperLU is a general library for the direct solution of large, sparse, nonsymmetric systems of linear equations on high performance machines. The library is callable from either C or Fortran. The library routines will perform an LU decomposition with partial pivoting and triangular system solves through forward and back substitution. The LU factorization routines can handle non-square matrices but the triangular solves are performed only for square matrices. The matrix columns may be preordered (before factorization) either through library or user supplied routines. This preordering for sparsity is completely separate from the factorization. Working precision iterative refinement subroutines are provided for improved backward stability. Routines are also provided to equilibrate the system, estimate the condition number, calculate the relative backward error, and estimate error bounds for the refined solutions.

The SuperLU_DIST library is designed for distributed memory parallel computers. The parallel programming model is SPMD and the library is implemented in ANSI C, using MPI for communication. The library includes routines to handle both real and complex matrices in double precision.

To use SuperLU_DIST, five basic step are required:

- Initialize the MPI environment and the SuperLU process grid.
- Set up the input matrix and the right-hand side
- Set the options argument
- Call the SuperLU routine `pdgssvx`
- Release the process grid and terminate the MPI environment

In Section 4.3, we show some numerical results with the SuperLU_DIST library, when it is applied to our particular test case.

3. TEST CASE

In this section we introduce the Lambda Modes Equation and its solution using a classic iterative algorithm, where the most expensive step is represented by a matrix-vector operation. For the test case chosen and introduced in this section, the matrix is not in explicit form due to the use of two energy groups.

3.1 Lambda modes equation

Reactors calculations are based on the multigroup neutron diffusion equation [Hébert1987]. If this equation is modeled with two energy groups, then the problem to be dealt with is to find the eigenvalues and eigenfunctions of

$$\mathcal{L}\psi_i = \frac{1}{\lambda_i}\mathcal{M}\psi_i \ , \tag{1}$$

which is known as the Lambda Modes equation [Vidal1997], where

$$\mathcal{L} = \begin{bmatrix} -\vec{\nabla}(D_1\vec{\nabla}) + \Sigma_{a1} + \Sigma_{12} & 0 \\ -\Sigma_{12} & -\vec{\nabla}(D_2\vec{\nabla}) + \Sigma_{a2} \end{bmatrix}, \tag{2}$$

$$\mathcal{M} = \begin{bmatrix} \nu\Sigma_{f1} & \nu\Sigma_{f2} \\ 0 & 0 \end{bmatrix} \quad \text{and} \quad \phi_i = \begin{bmatrix} \phi_{f_i} \\ \phi_{t_i} \end{bmatrix} \tag{3}$$

with boundary conditions $\phi_i|_\Gamma = 0$ where Γ is the reactor border.

For a numerical treatment, this equation must be discretized in space. Nodal methods are extensively used in this case. These methods are based on approximations of the solution in each node in terms of a suitable base of functions such as Legendre's polynomials [Verdú1993]. It is assumed that nuclear properties are constant in every cell. Finally, suitable continuity conditions for fluxes and currents are enforced.

This process allows the transformation of the original system of partial differential equations (1) into an algebraic large sparse generalized eigenvalue problem

$$L\psi_i = \frac{1}{\lambda_i}M\psi_i \ , \tag{4}$$

where *L* and *M* are matrices of order *2N* with the following block structure

$$\begin{bmatrix} L_{11} & 0 \\ -L_{21} & L_{22} \end{bmatrix}\begin{bmatrix} \psi_{1_i} \\ \psi_{2_i} \end{bmatrix} = \frac{1}{\lambda_i}\begin{bmatrix} M_{11} & M_{12} \\ 0 & 0 \end{bmatrix}\begin{bmatrix} \psi_{1_i} \\ \psi_{2_i} \end{bmatrix}. \tag{5}$$

Depending on flux continuity conditions imposed among the discretization cells of the nuclear reactor, the matrices L_{11} and L_{22} will be symmetric or not. Due to intrinsic properties of the physical problem, these matrices are diagonal dominant and positive definite. On the other hand, under certain conditions, the null block of operator *L* can be a diagonal matrix as L_{21}, M_{11}, and M_{12} blocks. All matrices are *sparse* and *very large*. This problem must be solved

in modal methods of transient analysis. By eliminating ψ_{2_i} in (5), we obtain the following *N*-dimensional standard eigenvalue problem

$$S\psi_{1_i} = \lambda_i \psi_{1_i}, \tag{6}$$

where the matrix *S* is given by

$$S = L_{11}^{-1}(M_{11} + M_{12}L_{22}^{-1}L_{21}). \tag{7}$$

One possible method for solving the standard eigenvalue problem represented in (6), is the Arnoldi Iteration [Saad1996] (Algorithm 1) where the most expensive operation is represented by a sparse matrix-vector operation Sv_j (line 3).

Algorithm 1. (*Arnoldi Iteration)*

```
1. Choose a vector v1 of norm 1
2. For j=1,2, …, m Do:
3.   Compute hij=(Svj,vi)for i=1,2,…,j /* expensive matrix-vector
       operation */
4.   Compute wj:=Svj- Σ_{i=1}^{j} hijvi
5.   hj+1,j= ||wj||2
6.   If hj+1,j=0 then Stop
7.   vj+1=wj/hj+1,j
8. EndDo
```

Nevertheless, due to the matrix *S* is not in explicit form, it is necessary to carry out the steps of the Algorithm 2, which consists of three diagonal-vector operations, one sum of vectors, and the *solution* of large sparse linear systems related to the matrices L_{11} and L_{22} (lines 2 and 5). These last operations are the most expensive and their fast solution might decrease the computation time of the eigenvalue problem established in (6).

Algorithm 2. (*Sv operation*)

```
1. w1:=L21v
2. w2:=L22^-1w1  /* solve a large sparse linear system */
3. w3:=M12w2
4. w4:=M11v
5. r:=L11^-1(w3+w4) /* solve a large sparse linear system */
```

3.2 Ringhals-I reactor

The test case corresponds to the Sweden nuclear reactor of Ringhals-I [Lefvert1996]. It has been discretized in a 3D form using a nodal collocation method as discretization technique

based on second order Legendre polynomials [Hébert1987] [Verdú1994]. The dimension of the associated matrix, L in (4), is 157248 with 1811072 non-zero elements (see Figure 2).

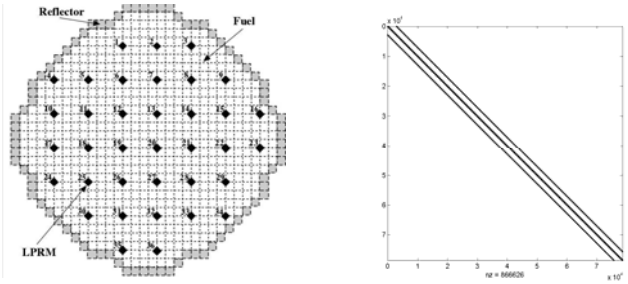

Figure 2. Ringhals-I reactor axial plane and block L_{ii} non-zero patern.

4. NUMERICAL RESULTS

Numerical experiments have been performed on a cluster of PCs, formed by ten PCs at the Polytechnic University of Valencia. Each computing node has two 2 GHz Intel Xeon processors, and each processor has 1 GB of RAM memory. The tolerance used for the stopping criterion is ε = 1.e-10.

4.1 Solution with pARMS

We have observed that the pARMS methods based on Schur Complement techniques, with or without enhancements, did not converge for any number of processors whereas the most methods based on Additive Schwarz, with the exception of the method add_arms, converged to the solution with the requested tolerance.

Table 3 shows the execution times using one, eight and ten processors. For the case of p=1 processor, the combination of add_ilu0 preconditioner with bcgstabd accelerator was the fastest one. With this combination, the solution of the large sparse linear systems L_{11} and L_{22} converged in 28 and 21 iterations, respectively. Use of add_iluk preconditioner presented memory problems due to fill-in effect. This problem has been solved with the use of more than one processor. Execution times with more than one processor (p= 8, 10) have shown that both fgmresd and add_ilu0 appear to be competitive. In addition, it has observed that add_iluk preconditioner was the most expensive due to *k* level fill-in.

Table 3. Execution times (seconds) in pARMS.

	(Using p=1 processor)			(Using p=8 processors)			(Using p=10 processors)		
Preconditioners	bcgstabd	dgmresd	fgmresd	bcgstabd	dgmresd	fgmresd	bcgstabd	dgmresd	fgmresd
add_ilu0	1.77	2.07	1.95	0.47	0.55	0.47	0.44	0.51	0.39
add_ilut	2.33	2.27	2.49	0.82	0.86	0.71	0.70	0.75	0.64
add_iluk	Memory problems	Memory problems	Memory problems	39.64	35.99	34.71	28.93	26.98	25.32

Table 4 shows the parallel performance achieved with *p=4, 8, 10* processors and the Figure 3 shows the execution times of this experiments. We can observe that the coefficients of speedup and efficiency are poor. However, we believe that using a different discretization method, we can achieve better execution times.

Table 4. Speedup and efficiency with pARMS.

P	T_p	S_p	E_p
1	1.76	1.00	100%
4	0.77	2.29	57%
8	0.47	3.79	47%
10	0.39	4.53	45%

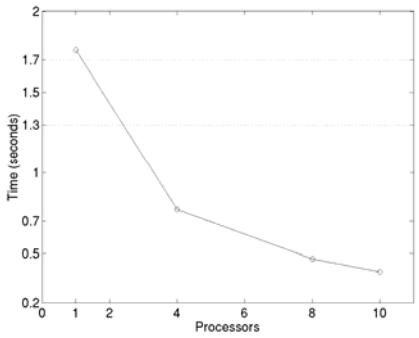

Figure 3. Execution times with p=1,4,8,10 in pARMS.

4.2 Solution with PETSc

We have tested many Krylov subspace methods and preconditioners with the PETSc library. Table 5 shows execution times with *p=1,4,8,10* processors without preconditioning. In these cases, the use of parallel computing is advantageous in the solution of very-large sparse linear systems. For example, from an execution time of 2.02 seconds with *p=1* processor, it decreases to 0.33 seconds using *p=10* processors. Also, we have observed that the most competitive method is the Conjugate Gradient (CG) for any number of processors, due to the L_{11} and L_{22} block properties. It is interesting to note that the Conjugate Residual (CR) method is competitive as well.

Table 5. Execution times (seconds) in PETSc (without Preconditioner).

Methods	p=1	p=4	p=8	p=10
CG	2.02	0.64	0.38	0.33
BICG	3.94	1.26	0.73	0.63
GMRES	4.42	1.26	0.63	0.55
BCGS	2.67	0.82	0.48	0.44
CGS	2.60	0.79	0.47	0.41
TFQMR	2.91	0.87	0.49	0.43
CR	2.23	0.69	0.42	0.36

We have combined the above methods with the Jacobi preconditioner. The execution times are showed in the Table 6, where we can see the effect of the preconditioner on the

convergence rate. For example, with p=1 processor, the execution time using non-preconditioned CG is 2.02 seconds (Table 5), whereas using a Jacobi-preconditioner CG the execution time decreases to 1.42 seconds. In addition, it is important to note that with ten processors (p=10), the execution time is reduced even more. This means a time reduction of almost 83% with regard to the sequential time using the same method and preconditioner.

Table 6. Execution times (seconds) in PETSc (with Jacobi preconditioner).

Methods	p=1	p=4	p=8	p=10
CG	1.42	0.45	0.26	0.24
BICG	2.71	0.86	0.50	0.44
GMRES	2.80	0.80	0.42	0.36
BCGS	1.88	0.58	0.35	0.30
CGS	1.64	0.50	0.30	0.26
TFQMR	1.84	0.54	0.31	0.28
CR	1.53	0.48	0.28	0.26

Applying a Block Jacobi preconditioner in combination with the PETSc methods, we obtain the execution times showed in Table 7. For p=1 processor, the runtime is 1.20 seconds, this means a reduction of 41% with regard to the non-preconditioned case. Using p=10 processors, we obtain the same benefit that in the Jacobi preconditioner case.

Table 7. Execution times (seconds) in PETSc (with Block-Jacobi preconditioner).

Methods	p=1	p=4	p=8	p=10
CG	1.20	0.45	0.24	0.21
BICG	2.32	0.81	0.44	0.37
GMRES	2.75	0.64	0.30	0.28
BCGS	1.38	0.51	0.28	0.24
CGS	1.36	0.46	0.25	0.22
TFQMR	1.47	0.47	0.26	0.23
CR	1.26	0.45	0.25	0.22

Conjugate Gradient (CG) and Conjugate Residual (CR) represented the most competitive methods mostly. The overall parallel performance is represented in Table 8 and we can see that the results are good enough. The execution times are showed in Figure 4.

Table 8. Parallel performance obtained with PETSc.

P	T_p	S_p	E_p
1	1.20	1.00	100%
4	0.45	2.68	67%
8	0.24	5.04	63%
10	0.21	5.68	57%

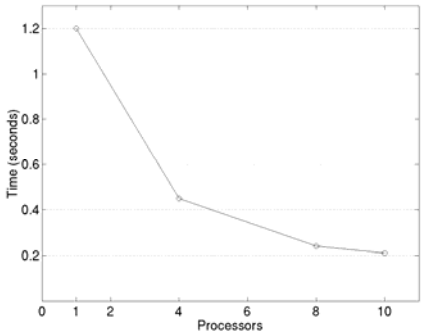

Figure 4. Execution times with p=1,4,8,10 in PETSc.

4.3 Solution with SuperLU

From the application of SuperLU to the test case, we found the parallel performance results showed in Table 9. However, the parallel execution times (T_p) are not specified in seconds, but in minutes. Also, we need to say that it was not possible to calculate the sequential execution time of SuperLU applied to our test case, due to memory problems and fill-in effect. So, in order to present the parallel performance using SuperLU, we consider the execution time with two processors as the basic execution time and from it, we derive the others parallel performance parameters.

Table 9. Parallel performance with SuperLU.

P	T_p	S_p	E_p
2	9.73	1.00	100%
4	5.00	1.89	94%
6	3.67	2.65	88%
8	2.95	3.20	80%
10	2.48	3.81	76%

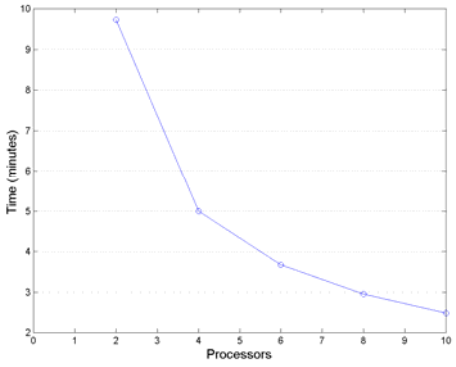

Figure 5. Execution times (minutes) with 2, 4, 6, 8 and 10 processors using SuperLU.

From Table 9, we can observe that the use of the parallel SuperLU library applied to our particular test case it is not sufficient to attain the objective of accelerate the solution process because it is not efficient. However, aspects such as *scalability*, *efficiency* and *speedup* are good using this library as we can see in Figure 5.

5. CONCLUDING REMARKS

In this work we have applied the numerical parallel libraries of PETSc, pARMS and SuperLU to solve the sparse linear systems related to a realistic case of Nuclear Engineering, specifically to a problem of Neutron Diffusion equation in steady-state. The numerical parallel libraries of PETSc and pARMS implement efficient iterative methods. By other hand, SuperLU implements a sparse LU factorization. In this work, we have used a distributed SuperLU version named SuperLU_DIST. Also, all these libraries are free distribution solvers.

Numerical experiments have shown that the PETSc implementation codes have been the most efficient ones for the test case. On the other hand, pARMS implementation codes are not so fast, but we think that using a different discretization method we can attain a better performance.

When we use a sparse direct solver such as SuperLU to solve the sparse linear systems we found that it does not accelerate the solution process. For that reason, the use of SuperLU is not appropriate for our test case. However, all libraries have shown a good accuracy.

Mostly, the fastest method has been the Conjugate Gradient method (CG), combined with a Block Jacobi preconditioner in PETSc, due to the properties of L_{11} and L_{22} matrices.

In the application of pARMS methods to the test case, we have found that the fastest method has been the Additive Schwarz method combined with an ILU0 preconditioner type, but overall performance has been poor.

The use of preconditioners combined with iterative methods has helped to accelerate the rate of convergence in the solution of the sparse linear systems of the test case. Parallel performance of PETSc codes will allow decreasing the computing time of the global process of finding the eigenvalues and eigenfunctions of the Lambda Modes Equation in large-scale simulations.

For this particular test case, iterative methods have showed to be supreme with regard to a direct sparse solver such as SuperLU.

Future works will contemplate experiments with the implementation of others types of methods and preconditioners such as second order methods and inverse preconditioners.

ACKNOWLEDGEMENT

This work has been supported by Spanish MCYT under Grant ENE2005-09219-C02-02/CON and by DGIT-SUPERA-ANUIES (Mexico).

REFERENCES

[Demmel2003] Demmel, J.W. et al., 2003, SuperLU_DIST: A Scalable Distributed-Memory Sparse Direct Solver for Unsymmetric Linear Systems, *ACM Trans. Mathematical Software*, Vol. 29, No. 2, pp. 110-140.

[Duff1986] Duff I.S. et al., 1986, *Direct Methods for Sparse Matrices* , Clarendon Press, Oxford.

[Groupp1994] Groupp W. et al., 1994, *Using MPI: Portable Parallel Programming with Message Passing Interface*, MIT Press.

[Hébert1987] Hébert A., 1987, Development of the Nodal Collocation Method for Solving the Neutron Diffusion Equation, *Ann. Nucl. Energy*, Vol. 14, No. 10, pp. 527-541

[Kumar1994] Kumar V. et al., 1994. *Introduction to parallel computing*. The Benjamin/Cummings Publishing Company, Inc., Redwood city California.

[Lefvert1996] Lefvert T., 1996. RINGHALS I Stability Benchmark – Final Report, NEA/NSC/DOC (96) 22, OECD Nuclear Energy Agency, Paris, France.

[Li2003] Li Z. et al., 2003. pARMS: A parallel version of the algebraic recursive multilevel solver, *Numerical Linear Algebra with Applications*, Vol. 10, pp. 485-509.

[Saad1996] Saad Y., 1996. *Iterative Methods for Sparse Linear Systems*, PWS Publishing Company, Boston, MA.

[Saad2003] Saad Y. et al., 2003. pARMS: A Package for the Parallel Iterative Solution of General Large Sparse Linear Systems: User's Guide, Technical Report, Minnesota Supercomputer Institute, University of Minnesota.

[Satish2001] Satish B. et al., 2001. PETSc Users Manual, Technical Report ANL-95/11 - Revision 2.1.0, Argonne National Laboratory.

[Verdú1993] Verdú G. et al., 1993, Lambda Modes of the Neutron-Diffusion Equation. Application to BWRs Out-of-Phase Instabilities, *Ann. Nucl. Energy*, Vol. 20, No. 7, pp. 477-501.

[Verdú1994] Verdú G. et al., 1994, 3D λ-Modes of the Neutron Diffusion Equation, *Ann. Nucl. Energy*, Vol. 21, pp. 405-421.

[Vidal1997] Vidal G., 1997. Métodos Numéricos para la Obtención de los Modos Lambda de un Reactor Nuclear. Técnicas de Aceleración y Paralelización. PhD. Thesis, Universidad Politécnica de Valencia, Valencia, Spain.