Sequential and Parallel Resolution of the Two-Group Transient Neutron Diffusion Equation using second-degree Iterative Methods

Omar Flores-Sánchez[1,2], Vicente E. Vidal[1], Victor M. García[1], and Pedro Flores-Sánchez[3]

[1] Departamento de Sistemas Informáticos y Computación
Universidad Politécnica de Valencia
Camino de Vera s/n, 46022 Valencia, España
{oflores, vvidal, vmgarcia}@dsic.upv.es
[2] Departamento de Sistemas y Computación
Instituto Tecnológico de Tuxtepec
Av. Dr. Victor Bravo Ahuja, Col. 5 de Mayo, C.P. 68300, Tuxtepec, Oaxaca, México
oflores70@hotmail.com
[3] Telebachillerato "El Recreo"
Tierra Blanca, Veracruz, México
pedrofs080374@hotmail.com

Abstract. We present an experimental study of two versions of a second-degree iterative method applied to the resolution of the sparse linear systems related to the 3D multi-group time-dependent Neutron Diffusion Equation (TNDE), which is important for studies of stability and security of nuclear reactors. In addition, the second-degree iterative methods have been combined with an adaptable technique, in order to improve their convergence and accuracy. The authors consider that second-degree iterative methods can be applied and extended to the study of transient analysis with more than two energy groups and they might represent a saving in spatial cost for nuclear core simulations. These methods have been coded in PETSc [1][2][3].

1 Introduction

For design and safety reasons, nuclear power plants need fast and accurate plant simulators. The centre point of concern in the simulation of a nuclear power plant is the reactor core. Since it is the source of the energy that is produced in the reactor, a very accurate model of the constituent processes is needed. The neutron population into the reactor core is modeled using the Boltzmann transport equation. This three-dimensional problem is modeled as a system of coupled partial differential equations, the multigroup neutron diffusion equations[4][5], that have been discretised using a nodal collocation method in space and one-step Backward-Difference Method in time. The solution of these equations can involve very intensive computing. Therefore, it is necessary to find effective algorithms for the solution of the three-dimensional model. The progress in the area

of multiprocessor technology suggests the application of High-Performance Computing to enable engineers to perform faster and more accurate safety analysis of Nuclear Reactors [6].

Bru et al in [7] apply two Second-Degree methods [8] to solve the linear system of equations related to a 2D Neutron-Diffusion equation case. Thus, the main goal of this paper is the application of those methods and some modifications that we have proposed to decrease the computational work, but applied to a 3D real test case.

The outline of the paper is as follows. The mathematical model of the Time-dependent Neutron Diffusion Equation and its discretisation are described in Section 2. The second-degree iterative methods are introduced at Section 3. Section 4 describes hardware and software platform used. The test case is presented in Section 5. Section 6 presents a sequential study of the second-degree iterative methods and the modifications proposed. In Section 7 numerical parallel results are presented. Finally, we will draw some conclusions in Section 8.

2 Problem Description

Plant simulators mainly consist of two different modules which account for the basic physical phenomena taking place in the plant: a neutronic module which simulates the neutron balance in the reactor core, and the evaporation and condensation processes. In this paper, we will focus on the neutronic module. The balance of neutrons in the reactor core can be approximately modeled by the time-dependent two energy group neutron diffusion equation, which is written using standard matrix notation as follows[9]:

$$[v^{-1}]\dot{\phi} + \mathcal{L}\phi = (1-\beta)\mathcal{M}\phi + \chi \sum_{k=1}^{K} \lambda_k \mathcal{C}_k \tag{1}$$

$$\dot{\mathcal{C}_k} = \beta_k [\nu\Sigma_{f_1} \nu\Sigma_{f_2}]\phi - \lambda_k \mathcal{C}_k, \qquad k = 1, \ldots, K \tag{2}$$

where

$$\mathcal{L} = \begin{bmatrix} -\boldsymbol{\nabla}\cdot(D_1\boldsymbol{\nabla}) + \sum_{a1} + \sum_{12} & 0 \\ -\sum_{12} & -\boldsymbol{\nabla}\cdot(D_2\boldsymbol{\nabla}) + \sum_{a2} \end{bmatrix}, [v^{-1}] = \begin{bmatrix} \frac{1}{v_1} & 0 \\ 0 & \frac{1}{v_2} \end{bmatrix},$$

and

$$\mathcal{M} = \begin{bmatrix} \nu\Sigma_{f1} & \nu\Sigma_{f2} \\ 0 & 0 \end{bmatrix}, \phi = \begin{bmatrix} \phi_f \\ \phi_t \end{bmatrix}, \chi = \begin{bmatrix} 1 \\ 0 \end{bmatrix},$$

where

- ϕ is the neutron flux on each point of the reactor; so, it is a function of time and position.

- $\mathcal{C}_k$ is the concentration of the k-th neutron precursor on each point of the reactor (it is as well a function of time and position). $\lambda_k \mathcal{C}_k$ is the decay rate of the k-th neutron precursor.
- K is the number of neutron precursors. β_k is the proportion of fission neutrons given by transformation of the k-th neutron precursor; $\beta = \sum_{k=1}^{K} \beta_k$.
- $\mathcal{L}$ models the diffusion ($-\boldsymbol{\nabla} \cdot (D_1 \boldsymbol{\nabla})$), absorption ($\sum_{a1}$, $\sum_{a2}$) and transfer from fast group to thermal group ($\sum_{12}$).
- $\mathcal{M}$ models the generation of neutrons by fission.
- $\nu \sum_{fg}$ gives the amount of neutrons obtained by fission in group g.
- v^{-1} gives the time constants of each group.

To study rapid transients of neutronic power and other space and time phenomena related to neutron flux variations, fast codes for solving these equations are needed. The first step to obtain a numerical solution of these equations consists of choosing a spatial discretization for equation (1). For this , the reactor is divided in cells or nodes and a nodal collocation method is applied[10][11]. In this collocation method, neutron flux is expressed as a series of Legendre Polynomials.

After a relatively standard process (setting boundary conditions, making use of the orthonormality conditions, using continuity conditions between cells) we obtain the following systems of ordinary differential equations:

$$[v^{-1}]\dot{\psi} + L\psi = (1-\beta)M\psi + X\sum_{k=1}^{K} \lambda_k C_k, \tag{3}$$

$$\dot{C}_k = \beta_k [M_{11} M_{12}]\psi - \lambda_k C_k, \qquad k = 1, \ldots, K, \tag{4}$$

where unknowns ψ and C_k are vectors whose components are the Legendre coefficients of ϕ and $\mathcal{C}_k$ in each cell, and L, M, $[v^{-1}]$ are matrices with the following block structure:

$$L = \begin{bmatrix} L_{11} & 0 \\ -L_{21} & L_{22} \end{bmatrix}, M = \begin{bmatrix} M_{11} & M_{12} \\ 0 & 0 \end{bmatrix}, v^{-1} = \begin{bmatrix} v^{-1} & 0 \\ 0 & v^{-1} \end{bmatrix}, X = \begin{bmatrix} I \\ 0 \end{bmatrix}.$$

Depending on flux continuity conditions imposed among the discretisation cells of the nuclear reactor, the blocks L_{11} and L_{22} can be symmetric or not. For our test case, these blocks are symmetric positive definite matrices[12], while blocks L_{21}, M_{11} and M_{12} are diagonal.

The next step consists of integrating the above ordinary differential equations over a series of time interval, $[t_n, t_{n+1}]$. Equation (4) is integrated under the assumption that the term $[M_{11}M_{12}]\psi$ varies linearly from t_n to t_{n+1}, obtaining the solution C_k at t_{n+1} expressed as

$$C_k^{n+1} = C_k^n e^{\lambda_k h} + \beta_k (a_k [M_{11} M_{12}]^n \psi^n + b_k [M_{11} M_{12}]^{n+1} \psi^{n+1} \tag{5}$$

where $h = t_{n+1} - t_n$ is a fixed time step size, and the coefficients a_k and b_k are given by

$$a_k = \frac{(1+\lambda_k h)(1-e^{\lambda_k h})}{\lambda_k^2 h} - \frac{1}{\lambda_k}, b_k = \frac{\lambda_k h - 1 + e^{\lambda_k h}}{\lambda_k^2 h}.$$

To integrate (3), we must take into account that it constitutes a system of stiff differential equations, mainly due to the elements of the diagonal matrix $[v^{-1}]$. Hence, for its integration, it is convenient to use an implicit backward difference formula (BDF). A stable one-step BDF to integrate (3) is given by

$$\frac{[v^{-1}]}{h}(\psi^{n+1} - \psi^n) + L^{n+1}\psi^{n+1} = (1-\beta)M^{n+1}\psi^{n+1} + X\sum_{k=1}^{K}\lambda_k C_k^{n+1} \tag{6}$$

Taking into account equation (5) and the structure of matrices L and M, we rewrite (6) as the system of linear equations

$$\begin{bmatrix} T_{11} & T_{12} \\ T_{21} & T_{22} \end{bmatrix} \begin{bmatrix} \psi_1^{n+1} \\ \psi_2^{n+1} \end{bmatrix} = \begin{bmatrix} R_{11} & R_{12} \\ 0 & R_{22} \end{bmatrix} \begin{bmatrix} \psi_1^{n} \\ \psi_2^{n} \end{bmatrix} + \sum_{k=1}^{K}\lambda_k e^{-\lambda_k h} \begin{bmatrix} C_k^n \\ 0 \end{bmatrix}, \tag{7}$$

where

$$\begin{aligned}
T_{11} &= \frac{1}{h}v_1^{-1} + L_{11}^{n+1} - (1-\beta)M_{11}^{n+1} - \sum_{k=1}^{K}\lambda_k\beta_k b_k M_{11}^{n+1}, \\
T_{21} &= -L_{21}^{n+1}, \\
T_{12} &= -(1-\beta)M_{12}^{n+1} - \sum_{k=1}^{K}\lambda_k\beta_k b_k M_{12}^{n+1}, \\
T_{22} &= \frac{1}{h}v_2^{-1} + L_{22}^{n+1}, \\
R_{11} &= \frac{1}{h}v_1^{-1} + \sum_{k=1}^{K}\lambda_k\beta_k a_k M_{11}^{n}, \\
R_{12} &= \sum_{k=1}^{K}\lambda_k\beta_k a_k M_{12}^{n}, \qquad R_{22} = \frac{1}{h}v_2^{-1}.
\end{aligned}$$

Thus, for each time step it is necessary to solve a large and sparse system of linear equations, with the following block structure:

$$\begin{bmatrix} T_{11} & T_{12} \\ T_{21} & T_{22} \end{bmatrix} \begin{bmatrix} \psi_1 \\ \psi_2 \end{bmatrix} = \begin{bmatrix} e_1 \\ e_2 \end{bmatrix} \tag{8}$$

where the right-hand side depends on both the solution in previous time steps and the backward difference method used. Usually, the coefficients matrix of system (8) has similar properties as the matrices L and M in equation (3), namely blocks T_{11}, T_{22} are symmetric positive definite matrices, and blocks T_{12}, T_{21} are singular diagonal matrices. System (8) will be also denoted as

$$T\psi = e. \tag{9}$$

3 Second-Degree Iterative Methods.

We begin this section with a brief introduction to the second-degree methods presented and applied to a 2D neutron diffusion equation case in [7].

3.1 Second Degree Method A

Consider the coefficient matrix T of the linear system (9) and the Jacobi splitting, $T = M - N$, with matrices M and N given by

$$M = \begin{bmatrix} T_{11} & 0 \\ 0 & T_{22} \end{bmatrix}, N = \begin{bmatrix} 0 & -T_{12} \\ -T_{21} & 0 \end{bmatrix}$$

where iteration matrix B_J is represented by

$$B_J = M^{-1}N = \begin{bmatrix} 0 & -T_{11}^{-1}T_{12} \\ -T_{22}^{-1}T_{21} & 0 \end{bmatrix}$$

Now, considering the matrices $G_1 = \omega B_J$, $G_0 = (1-\omega)B_J$, where ω is an extrapolation factor, and the vector $k = M^{-1}e$, we can write the following second degree method based on the Jacobi Over-relaxation (JOR) splitting

$$\psi^{(n+1)} = G_1\psi^{(n)} + G_0\psi^{(n-1)} + k = B_J(\omega\psi^{(n)} + (1-\omega)\psi^{(n-1)}) + k,$$

which corresponds to the following operations

$$\begin{aligned} T_{11}\psi_1^{l+1} &= e_1 - T_{12}(\omega\psi_2^l + (1-\omega)\psi_2^{l-1}), \\ T_{22}\psi_2^{l+1} &= e_2 - T_{21}(\omega\psi_1^l + (1-\omega)\psi_1^{l-1}). \end{aligned} \tag{10}$$

Let us identify these operations as Method A.

3.2 Second Degree Method B

In the same manner, we can construct another method based on the accelerated Gauss-Seidel splitting, whose iteration matrix B_{GS} is given by

$$B_{GS} = M^{-1}N = \begin{bmatrix} T_{11} & 0 \\ T_{21} & T_{22} \end{bmatrix}^{-1} \begin{bmatrix} 0 & -T_{12} \\ 0 & 0 \end{bmatrix},$$

The operations that correspond to this method are represented by

$$\begin{aligned} T_{11}\psi_1^{l+1} &= e_1 - T_{12}(\omega\psi_2^l + (1-\omega)\psi_2^{l-1}), \\ T_{22}\psi_2^{l+1} &= e_2 - T_{21}(\omega\psi_1^{l+1} + (1-\omega)\psi_1^l). \end{aligned} \tag{11}$$

Methods A and B, can be described by the following algorithmic scheme

Second-Degree Iterative Method Algorithm

(1) Set ψ_2^0; $\{\psi_2^0 := \psi_2^*\}$
(2) Solve $T_{11}\psi_1^1 = e_1 - T_{12}\psi_2^0$
(3) Solve $T_{22}\psi_2^1 = e_2 - T_{21}\psi_1^1$
(4) Do $l = 0, 1, 2, \ldots$
    (4a) Solve ψ_1^{l+1} in accordance with A or B method.
    (4b) Solve ψ_2^{l+1} in accordance with A or B method.
  until $\|\psi_1^{l+1} - \psi_1^l\| < tol$ and $\|\psi_2^{l+1} - \psi_2^l\| < tol$

where, ψ_2^* represents the solution of a previous time step.

As we can see, the main difference between methods (10) and (11), is that in (11), the new solution for ψ_1 is used as soon as it is available to compute ψ_2. Therefore, a faster convergence rate may be expected. In both methods, we distinguish between outer and inner iterations. The outer iterations are identified by the Step (4), and inner iterations are represented by Step (4a) and (4b), which correspond to iterations needed for solving the linear systems with matrices T_{11} and T_{22} respectively. Since these blocks are symmetric positive-definite matrices the *Conjugate-Gradient method*[16] was applied. General theorems about the convergence of second-degree methods can be found in [8].

The next section presents the hardware and software tools that we have used to carry out the numerical experiments.

4 Hardware and Software Platform

Sequential and parallel experiments have been performed on a 12-node biprocessor cluster with Red Hat 8.0 operating system, using only one CPU per node at the Polytechnic University of Valencia. Each CPU is a 2 GHz Intel Xeon processor and has 1 GB of RAM memory. All nodes are connected by a SCI network with a Torus 2D topology in a 4x5 mesh.

The Portable, Extensible Toolkit for Scientific Computation (PETSc)[1][2][3], 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 Standard for all message-passing communication. Some of the PETSc modules deal with vectors, matrices (generally sparse), distributed arrays, Krylov subspace methods, preconditioners including multigrid and sparse direct solvers, etc.

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

PETSc uses the message-passing model for parallel programming and employs MPI for all interprocessor communication. In PETSc the user is free to employ MPI routines as needed throught an application code. However, by default the user is shielded from many of the details of message passing within PETSc, since these are hidden within parallel objects, such as vectors, matrices, and solvers. In addition, PETSc provides tools such as generalized vector

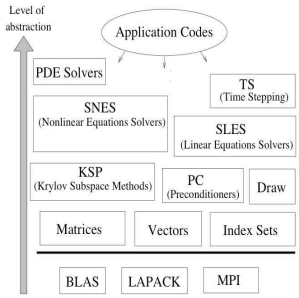

Fig. 1: Organization of PETSc library.

scatter/gathers and distributed arrays to assist in the management of parallel data.

PETSc provides a variety of matrix implementations because no single matrix format is appropriate for all problems. Currently PETSc supports dense storage and compressed sparse row storage, as well as several specialized formats. There are sequential and parallel AIJ sparse matrix format in PETSc. In the sequential AIJ sparse matrix, the nonzero elements are stored by rows, along with an array of corresponding column numbers and an array of pointers to the beginning of each row. Parallel sparse matrices with the AIJ format can be created with the command

```
MatCreateMPIAIJ(MPI_Comm comm,int m, int n, int M,int N,
                int d_nz,int *d_nnz,int o_nz,int *o_nnz, Mat *A);
```

`A` is the newly created matrix, while the arguments `m, M` and `N`, indicate the number of local rows and the number of global rows and columns, respectively. In the PETSc partitioning scheme, all the matrix columns are local and `n` is the number of columns corresponding to local part of a parallel vector. Either the local o global parameters can be replaced with PETSC_DECIDE, so that PETSc will determine them. The matrix is stored with a number of rows on each process, given by `m`, or determined by PETSc if `m` is PETSC_DECIDE. If PETSC_DECIDE is not used for the arguments `m` and `n`, then the user must ensure that they are chosen to be compatible with the vectors. To do this, one first considers the matrix-vector product $y = Ax$. The `m` that is used in the matrix creation routine MatCreateMPIAIJ() must match the local size used in the vector routine VecCreateMPI() for `y`. Likewise, the `n` used must match that used as the local size in VecCreateMPI() for `x`. For example, the PETSc partitioning scheme using the parallel sparse matrix AIJ format for operation Ax, must be as follows

$$
\begin{array}{ll}
p_0 & \\
p_1 & Ax = \\
p_2 &
\end{array}
\left(\begin{array}{ccc|ccc|cc}
1 & 2 & 0 & 0 & 3 & 0 & 0 & 4 \\
0 & 5 & 6 & 7 & 0 & 0 & 8 & 0 \\
9 & 0 & 10 & 11 & 0 & 0 & 12 & 0 \\
\hline
13 & 0 & 14 & 15 & 16 & 17 & 0 & 0 \\
0 & 18 & 0 & 19 & 20 & 21 & 0 & 0 \\
0 & 0 & 0 & 22 & 23 & 0 & 24 & 0 \\
\hline
25 & 26 & 27 & 0 & 0 & 28 & 29 & 0 \\
30 & 0 & 0 & 31 & 32 & 33 & 0 & 34
\end{array}\right)
\left(\begin{array}{c}
1 \\ 0 \\ 5 \\ \hline 7 \\ 9 \\ 0 \\ \hline 10 \\ 11
\end{array}\right)
\begin{array}{l}
p_0 \\ \\ \\ p_1 \\ \\ \\ p_2 \\ \\
\end{array}
$$

where the local parts of matrix A and vector x stored in processor p_0 are

$$
A_{p_0} = \left(\begin{array}{ccc|ccc|cc}
1 & 2 & 0 & 0 & 3 & 0 & 0 & 4 \\
0 & 5 & 6 & 7 & 0 & 0 & 8 & 0 \\
9 & 0 & 10 & 11 & 0 & 0 & 12 & 0
\end{array}\right) \text{ and } \begin{pmatrix} 1 \\ 0 \\ 5 \end{pmatrix} = x_{p_0},
$$

respectively.

In PETSc, user must specify a communicator upon creation of any PETSc object (such as a vector, matrix or solver) to indicate the processors over which the object is to be distributed.

Among the most popular Krylov subspace iterative methods contained in PETSc are *Conjugate Gradient, Bi-Conjugate Gradient, Stabilized BCG, Transpose Free Quasi-Minimal Residual, Generalized-Minimal Residual* and so on[16]. PETSc offers preconditioners as *Additive Schwarz, Block Jacobi, Jacobi, ILU, ICC*, etc. We do not apply preconditioning for our test case due to the good spectral properties of T_{11} and T_{22} blocks, as we can see in the convergence curves represented in the Figure 2.

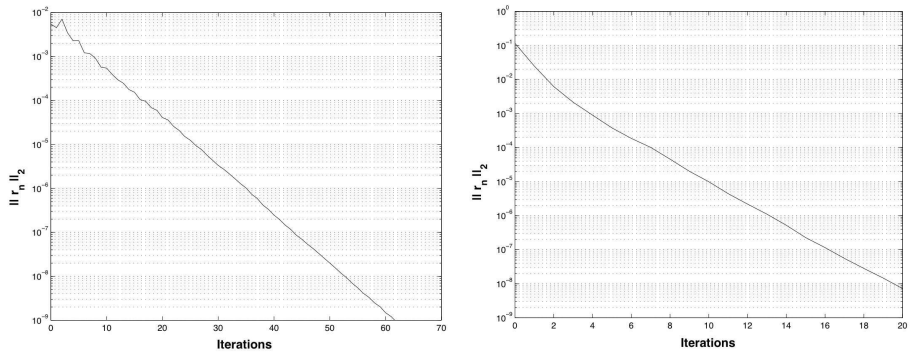

Fig. 2: CG convergence curves for T_{11} and T_{22} blocks.

Timing is obtained through the use of real-time (wall-clock time) clock function available in PETSc library. For all methods, we have verified their accuracy

and precision with regard to the global system $T\psi = e$ using the Matlab software.

5 Test Case

The test case chosen is the comercial reactor of Leibstadt[13], which has been discretised in a 3D form. The spatial discretisation has 32*32*27 cells, so that the total number of equations and cells is quite large: 157248 equations and 796080 nonzero elements in the Jacobian matrix.

We have applied all methods presented here to the set of matrices belongs to time step $t = 0$, which corresponds to a stability test carried out in 1990 where the reactor oscillates out of phase, to test their robustness and efficiency.

In the next section, the sequential performance is analysed, and some variations to A and B methods are introduced.

6 Sequential study of methods A and B and some variations.

In order to identify the optimum ω for A and B methods in our test case, we have carry out a heuristic study. Some results of this study are described in Table 1.

Table 1: Sequential execution times (secs) for method A and B with different ω values. The symbol † indicates that convergence was not attained.

ω	0.1	0.5	0.8	0.9	1.0	1.3	1.5	1.9
A	252.46	223.97	196.75	187.39	**138.08**	†	†	†
B	211.08	175.44	143.61	130.40	116.28	**63.98**	290.85	†

In accordance with CPU execution times of this table, the optimum ω for method A is 1.0 and for method B is 1.3.

The errors attained with methods A and B are represented in Table 2, where r_{l+1} and $\|\cdot\|$ represent the residual $e - T\psi^{l+1}$ and the Euclidean norm, respectively.

From Tables 1 and 2, we can observe that method B is twice more efficient than method A as we had expected.

In order to reduce even more the computational work of method B, we have modified the operations as follows

$$\begin{aligned} T_{11}\psi_1^{l+1} &= e_1 - T_{12}(\omega_1\psi_2^l + (1-\omega_1)\psi_2^{l-1}), \\ T_{22}\psi_2^{l+1} &= e_2 - T_{21}(\omega_2\psi_1^{l+1} + (1-\omega_2)\psi_1^l). \end{aligned} \tag{12}$$

Under this scheme, we have added two different parameters ω_1 and ω_2. Let us identify (12) as method C.

From application of method C to the test case, the optimum value of ω_1 is 1.0 and for ω_2 is 1.9. Table 3 shows a comparison of the number of iterations and execution times registered by method B and C, and we can see that the goal of decrease the computational work has been reached without lost of accuracy. From Table 3 we can observe a time reduction of 43% with regard to method B.

Table 2: Precision of methods A and B with optimum ω value.

	$\|r_{l+1}\|_2$	$\|r_{l+1}\|_2/\|e\|_2$	Its.
A	8.83e-6	7.58e-5	394
B	4.91e-6	4.21e-5	183

Table 3: Performance comparison between method B and C

	Its.	CPU Time (secs.)	$\|r_{l+1}\|_2/\|e\|_2$
B	183	63.98	4.21e-5
C	**89**	**36.29**	5.42e-5

Since the precision of method C is good as A and B methods, we have experimented with an 'adaptable' precision technique, achieving some improvements in the efficiency of the process. This technique solves T_{11} and T_{22} blocks with a cheap precision (ϵ_{ρ_i}) at initial stages of the method. Then, this precision is 'adapted' or 'improved' towards a more demanding one ($\epsilon_{\rho_{i+1}}$) in successive iterations. Application of this technique to method C give rise to the following algorithm (method D in this work).

Second-Degree Iterative Algorithm (Adaptable version)

(1) Set ψ_2^0; $\{\psi_2^0 := \psi_2^*\}$
(2) Set $\epsilon_\rho = \{\epsilon_{\rho_1}, \epsilon_{\rho_2}, \ldots, \epsilon_{\rho_n}\}$ where $\epsilon_{\rho_i} > \epsilon_{\rho_{i+1}}$;
(2) Solve $T_{11}\psi_1^1 = e_1 - T_{12}\psi_2^0$
(3) Solve $T_{22}\psi_2^1 = e_2 - T_{21}\psi_1^1$
(4) Do $l = 0, 1, 2, \ldots$
    (4a) Solve for ψ_1^{l+1} with tolerance ϵ_{ρ_i}
    (4b) Solve for ψ_2^{l+1} with tolerance ϵ_{ρ_i}
    (4c) if *precision of* $r_{l+1} \leq r_l$
        $i := i + 1$
    end if
until $\|\psi_1^{l+1} - \psi_1^l\| < tol$ and $\|\psi_2^{l+1} - \psi_2^l\| < tol$

Numerical experiments have shown that Method D is 25% more efficient than method C. Also, it is as exact as the rest of methods for the test case (See Table 4).

Table 4: Performance comparison between method C and D

	Its.	CPU Time (secs.)	$\|r_{l+1}\|_2/\|e\|_2$
C	89	36.29	5.42e-5
D	90	**27.31**	5.42e-5

Next section presents the parallel numerical results for all methods.

7 Parallel numerical results

All methods have been coded using the following PETSc operations facilites:

- `VecNorm` Computes the vector norm.
- `VecPointwiseMult` Computes the componentwise multiplication w = x*y.
- `VecAYPX` Computes y = x + alpha y.
- `VecCopy` Copies a vector.
- `KSPSolve` Solves a linear system. Steps (4a) and (4b) are carry out through the use of this operation.

Method A presents a good parallelism degree because the different linear systems of equations in (10) can be simultaneously solved by different groups of processors, and then interchange their solutions. For that reason, we have implemented two different parallel versions of this method, based on two different MPI communication routines: *gather/scatter* and *send/recv*. Also, we use the MPI facility to manage groups of processes through the use of communicators. For example, for the case of use $p = 2$ processors in method A, processor p_0 is dedicated to solve system T_{11} and processor p_1 is dedicated to solve system T_{22}. For the case of use $p = 4$ processors, $\frac{p}{2}$ processors are dedicated to solve system T_{11} and the rest dedicated to solve T_{22}, and so on.

The timing results for different number of processors (p) are registered in Table 5. As we can see, version based on *send/recv* is slightly more efficient than version based on *gather/scatter* primitives.

Table 5: Parallel execution times (secs) with method A

p	1	2	4	6	8	12
gather/scatter	138.08	89.33	54.90	40.51	33.86	25.47
send/recv	138.08	**88.27**	**53.07**	**38.05**	**31.30**	**22.60**

We have attempted to implement an asynchronous version of method A, but this was not possible, due to the strong dependency between T_{11} and T_{22} blocks.

We present a summary of parallel execution times with all methods in Table 6, where we can observe that the use of High-Performance Computing has decreased the sequential execution times for all methods. For example, for A and B methods the sequential execution times have been reduced until 16% of the original time value when we use $p = 12$ processors. For C and D methods, the execution time was reduce to 14% for the same number of processors.

Table 6: Parallel execution times (secs) for all methods

Method	$p = 1$	$p = 2$	$p = 4$	$p = 6$	$p = 8$	$p = 12$
A	138.08	88.27	53.07	38.05	31.30	22.60
B	63.98	36.83	28.78	20.26	11.94	10.41
C	36.29	21.11	12.01	8.68	6.91	5.23
D	**27.31**	**15.83**	**8.90**	**6.51**	**5.17**	**3.90**

From Table 6 we observe that method D offers the best execution time. The *speedup* and *efficiency*[15] of method D are represented in Figure 3, where for all values of p, parallel efficiency remains above of 50%.

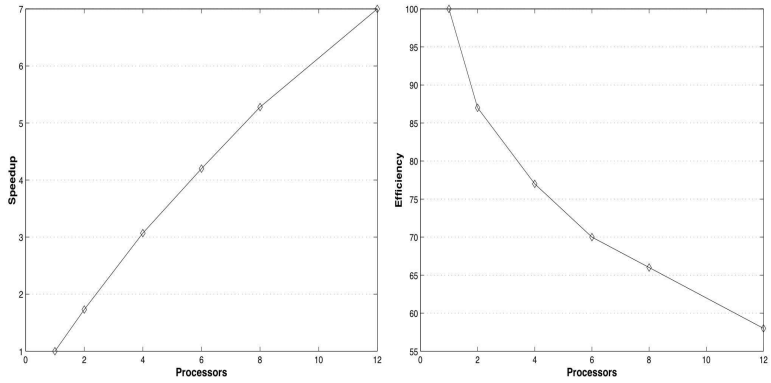

Fig. 3: Speedup and efficiency of method D.

8 Conclusions

We have presented the application and parallelisation of two second-degree methods (A and B methods) to solve the sparse linear system related to a 3D Neutron-

Diffusion equation of a real nuclear reactor using the numerical parallel library of PETSc.

In addition, we have modified A and B methods in order to reduce the computational work. For this, we have implemented two versions: the first one, based on two different relaxation parameters for each energy group obtaining a great performance which we call method C; and a second one, named method D, which is based on an *adaptable* technique that improves even more the performance with regard to the others methods.

We have carry out a heuristic study of the optimum relaxation parameter for each one of the methods presented and for our particular test case. These parameters have helped to accelerate the methods, specially C and D methods.

The main advantage of the second-degree methods presented in this work, is that matrix T do not need be formed explicitly; thus, simulation with more than 2 energy groups can be feasible.

It is important to emphasize that the application of High Performance Computing has reduced the sequential time of the different methods presented in this work.

Future works will contain the integration of these methods to DDASPK and FCVODE routines and the simulation of a full transient.

Acknowledgement

This work has been supported by Spanish MEC and FEDER under Grant ENE2005-09219-C02-02 and SEIT-SUPERA-ANUIES (México).

References

1. Balay S., Gropp W.D., McInnes L.C., Smith B.F.: PETSc home page http://www.mcs.anl.gov/petsc (2002)
2. Balay S., Gropp W.D., McInnes L.C., Smith B.F.: PETSc Users Manual. ANL-95/11 - Revision 2.1.5, Argonne National Laboratory (1997)
3. Balay S., Gropp W.D., McInnes L.C., Smith B.F.: Efficient Management of Parallelism in Object Oriented Numerical Software Libraries. Modern Software Tools in Scientific Computing (1997) 163-202
4. Weston J.R., Stacey M.: SpaceTime Nuclear Reactor Kinetics. Academic Press (1970)
5. Henry A.F.: Nuclear Reactor Analysis. The M.I.T Press (1975)
6. García V.M., Vidal V., Verdú G., Miró R.: Sequential and Parallel Resolution of the 3D Transient Neutron Diffusion Equation. Mathematics and Computation, Supercomputing, Reactor Physics and Nuclear and Biological Applications, on CD-ROM, American Nuclear Society (2005)
7. Bru R., Ginestar D., Marín J., Verdú G., Mas J., Manteuffel T.: Iterative Schemes for the Neutron Diffusion Equation. Computers and Mathematics with Applications, Vol.44, (2002) 1307-1323
8. D.M. Young.: Iterative Solution of Large Linear Systems. Academic Press Inc.,New York, N.Y. (1971)

9. Stacey W.M.: Space-Time Nuclear Reactor Kinetics. Academic Press, New York (1969)
10. Verdú G., Ginestar D., Vidal. V., Muñoz-Cobo J.L.: A Consistent Multidimensional Nodal Method for Transient Calculation. Ann. Nucl. Energy, 22(6), (1995) 395-410
11. Ginestar D., Verdú G., Vidal V., Bru R., Marín J., Muñoz J.L.: High order backward discretization of the neutron diffusion equation. Ann. Nucl. Energy, 25(1-3), (1998) 47-64
12. Hébert A.: Development of the Nodal Collocation Method for Solving the Neutron Diffusion Equation. Ann. Nucl. Energy, 14(10), (1987) 527-541
13. Blomstrand J.: The KKL Core Stability Test, conducted in September 1990. ABB Report, BR91-245 , (1992)
14. D. Ginestar and J. Marín and G. Verdú.: Multilevel methods to solve the neutron diffusion equation. Applied Mathematical Modelling, 25, (2001) 463-477
15. V. Kumar and A. Grama and A. Gupta and G. Karypis.: Introduction to parallel computing:design and analysis of parallel algorithms. The Benjamin/Cummings Publishing Company, Inc.,Redwood City, CA (1994)
16. Y. Saad. Iterative Methods for Sparse Linear Systems PWS Publishing Company, Boston, MA (1996)