High Performance Preconditioning Techniques for the Solution of Two-Group Transient Neutron Diffusion Equation

Omar Flores[12], Vicente Vidal[1], L.A. Drummond[3], and Gumersindo Verdú[4]

[1] Departamento de Sistemas Informáticos y Computación
Universidad Politécnica de Valencia
Camino de Vera s/n, 46022 Valencia, España
{oflores,vvidal}@dsic.upv.es
[2] Departamento de Sistemas y Computación
Instituto Tecnológico de Tuxtepec
Av. Bravo Ahuja s/n, Col. 5 de Mayo, C.P. 68300, Tuxtepec, Oaxaca, México
[3] Computational Research Division
Lawrence Berkeley National Laboratory
One Cyclotron Road
Berkeley, CA 94720
LADrummond@lbl.gov
[4] Departamento de Química e Ingeniería Nuclear
Universidad Politécnica de Valencia
Camino de Vera s/n, 46022 Valencia, España
gverdu@iqn.upv.es

Abstract. We evaluate the performance of preconditioned Second-Order degree iterative methods for the solution of large-sparse linear systems of equations related to the 3D multi-group time-dependent Neutron Diffusion Equation. Efficient solutions to these problems are important for studies of stability and security of nuclear reactors. For this, first we present the results of a comparison among preconditioners available in *Hypre*[1][2][3] that are more appropriated for our problem. Secondly, they are compared against results obtained from preconditioners available in the *PETSc* library[4][5][6]. Our test examples are based on the commercial nuclear reactor of Leibstadt.

Key words: Second-Order Iterative Methods, Transient Neutron Diffusion Equation, Parallel Computing.

1 Introducción

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[7][8], 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 to perform faster and more accurate safety analysis of Nuclear Reactors [9].

Bru et al in [10] apply two Second-Degree iterative methods [11] to solve the linear system of equations related to a 2D Neutron-Diffusion equation case. These same techniques with new modifications have been applied to a 3D real test case and presented in [12]. The focus of this paper is to try and find the preconditioners that offer the best performance. As we have mentioned before, we have choosen the HYPRE and *PETSc* libraries due to their facility of use. Also we compare the performance of such preconditioners.

This papers is organized as follows. In section 2, we introduce some basic knowledge of the mathematical model of the Time-dependent Neutron Diffusion Equation and its discretisation. Then we give some details about second-degree iterative methods in section 3. In section 4 we describe the hardware and software used. The test case is presented in section 5. In section 6 some numerical results are presented. Finally, section 7 contains some brief concluding remarks.

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[13]:

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

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

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[14][15]. 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[16], 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 and 3D neutron diffusion equation case (see [10][12].

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; $\quad \{\psi_2^0 := \psi_2^*\}$
(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 $l = 1, 2, \ldots$
    (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. General theorems about the convergence of second-degree methods can be found in [11].

3.3 Method C

As we can see, the solution process with T_{11} and T_{22} blocks can be carried out independently each one of other. This fact has motivated experiments with the following operations

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

In this new scheme, method C, we have added two different parameters ω_1 and ω_2 for each system to be solved, in order to accelerate its convergence. From application of method C to the test case, we find that the optimum value of ω_1 is 1.0 and for ω_2 is 1.9 (See [12]).

3.4 Method D

Besides to method C, we have carried out experiments with an *'adaptable'* technique, achieving some improvements in the process efficiency. This technique computes the solution of the systems $T_{11}\psi_1^{l+1}$ and $T_{22}\psi_2^{l+1}$ with a cheap precision ϵ_i at initial stages of the method. Then, this precision is 'adapted' or 'improved' towards a more demanding one in successive iterations. The application of this technique to method C gives rise to the following algorithm (method D), where r_l means 'precision' r achieved in stage l. We believe that application of preconditioning techniques to the *solve* steps can leads to speedup the convergence rate of these methods.

Second-Degree Iterative Method Algorithm (Adaptable version)

(1) Set ψ_2^0; $\{\psi_2^0 := \psi_2^*\}$
(2) Set $\epsilon_i = \{\epsilon_1, \epsilon_2, \ldots, \epsilon_n\}$ where $\epsilon_i > \epsilon_{i+1}$;
(3) Set $i = 1$;
(4) Solve $T_{11}\psi_1^1 = e_1 - T_{12}\psi_2^0$ with tolerance ϵ_i
(5) Solve $T_{22}\psi_2^1 = e_2 - T_{21}\psi_1^1$ with tolerance ϵ_i
(6) Do $l = 1, 2, \ldots$
(6a) Solve for ψ_1^{l+1} with tolerance ϵ_i
(6b) Solve for ψ_2^{l+1} with tolerance ϵ_i
(6c) if $r_{l+1} \leq r_l$
$i := i + 1$
end if
until $\|\psi_1^{l+1} - \psi_1^l\| < tol$ and $\|\psi_2^{l+1} - \psi_2^l\| < tol$

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

4 Computer Platform

This section presents a brief overview of the hardware and software used to carry out the numerical experiments.

4.1 Hardware

Sequential and parallel computations have been performed on a 8-node biprocessor cluster subcomplex of an AMD Opteron supercluster named Jacquard[18] running a Linux operating system at National Energy Research Scientific Computing Center[19]. The machine is named in honor of inventor Joseph Marie Jacquard. Each processor runs at a clock speed of 2.2GHz, and has a theoretical peak performance of 4.4 GFlop/s. Processors on each node share 6GB of memory. The nodes are interconnected with a high-speed InfiniBand network.

4.2 Software

The *PETSc* Library The Portable, Extensible Toolkit for Scientific Computation library (*PETSc*)[4][5][6], it 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. Krylov subspace methods, preconditioners including multigrid and sparse direct solvers, etc.

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

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

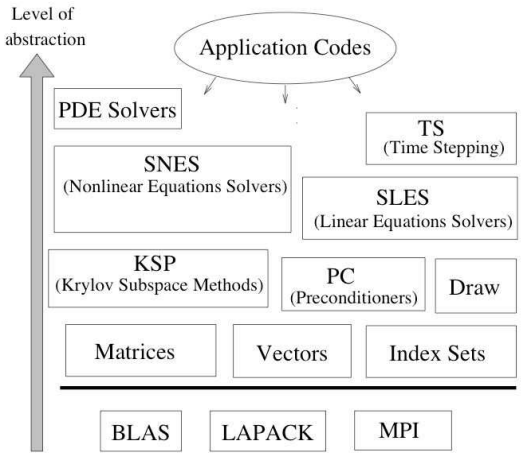

Fig. 1. Organization of *PETSc* library.

with an array of corresponding column numbers and an array of pointers to the beginning of each row. For example, the *PETSc* partitioning scheme using the parallel sparse matrix AIJ format for operation Ax, must be as follows

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

where we can see the local parts of matrix A and vector x stored in processor p_0.

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[17]. In addition, *PETSc* offers preconditioners as *Additive Schwarz, Block Jacobi, Jacobi, ILU, ICC*, etc.

The *Hypre* Library The High Performance Preconditioners (*Hypre*)[1][2][3] is a library of high performance preconditioners and solvers for the solution of large, sparse linear systems of equations on massively parallel computers. The *Hypre* library was created with the primary goal of providing users with advanced parallel preconditioners. The library features parallel multigrid solvers for both structured and unstructured grid problems.

The available solvers and preconditioners in *Hypre* are accessed from the application code via *Hypre*'s conceptual linear system interfaces, which allow a variety of natural problem descriptions (see Fig. 2).

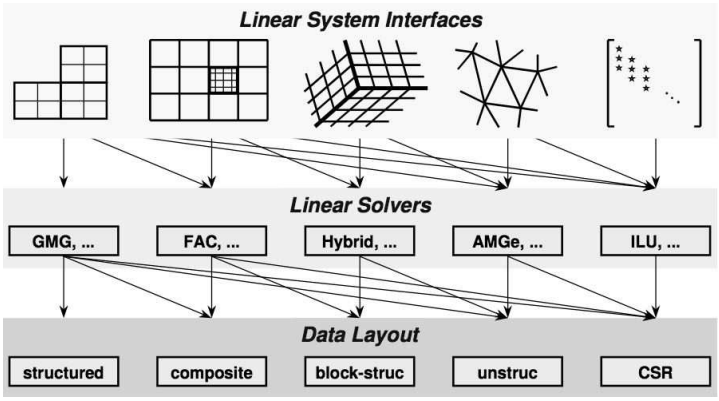

Fig. 2. Organization of conceptual interfaces in *Hypre*.

In order to use *Hypre* we have chosen the IJ interface due to our test matrices are specified in CSR format. Matrices are assumed to be distributed by blocks of rows as follows:

$$\begin{bmatrix} A_0 \\ A_1 \\ \vdots \\ A_{P-1} \end{bmatrix}. \tag{13}$$

The above example shows that matrix is distributed accross the P processes, $0, 1, \ldots, P-1$ by blocks of rows. Each submatrix A_p is "owned"" by a single process.

For IJ conceptual interface, *Hypre* provides popular Krylov subspace iterative methods such as *Preconditioned Conjugate Gradient, Generalized Minimum Residual* and *Biconjugate Gradient Stabilized (BICGSTAB)*. In addition *Hypre* contains several families of preconditioner algorithms such as *BoomerAMG, ParaSails, Euclid* and so on[3][2].

5 Test Case

The test case chosen is the comercial reactor of Leibstadt[21], 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 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. Timing is obtained through the use of `MPI_Wtime` function available in MPI library. For all methods, we have verified their accuracy and precision with regard to the global system $T\psi = e$ using the Matlab software.

6 Numerical Results

We have used the same vector as initial solution (ψ^*) for all methods. Also, we have chosen the Krylov subspace method of Conjugate Gradient, because the blocks M_{11} and M_{22} are nearly-symmetric positive-definite matrices due to the continuity conditions imposed on the flux among cells. In addition, we have required that the relative error be less than 10^{-7} as the criterion for convergence for the solve phases. Parallel computations were carried out with $p = 1, 2, 4, 6, 8, 10, 14, 16$ processors.

6.1 Sequential Execution Times

For the case of *Hypre*, we have carried out test with preconditioners such as *Diagonal Scaling(DS)*, *BoomerAMG*, *ParaSails* and *Euclid* and without preconditioning (*PCNONE*). From the application of these preconditioners we have obtained the execution times showed in Table 1. From this table, we can comment: first, we see that all methods present the same performance observed in a prior work, where efficiency of method D is better than method A (see [12]). Secondly, we can observe that the application of preconditioning techniques had speed up the convergence rate for some *Hypre* preconditioners. For example, the execution time of method D with an *Euclid* preconditioner represents a reduction of almost 40% with respect to the case without preconditioning. This fact shows that use of preconditioning techniques had improved the performance substantially for our test case. It is important pointed out, that application of a sparse approximate inverse preconditioner (*ParaSails*) is slightly cheaper compared to Diagonal Scaling (*DS*) for our test matrices. However, numerical test using higher level sparsity patterns, had lead to worse execution times.

Table 1. Sequential execution times (*secs*) in *Hypre*

METHOD	PCNONE	DS	BoomerAMG	ParaSails	Euclid
A	403.12	310.43	1955.17	290.23	**236.66**
B	192.18	147.81	885.01	133.50	**111.73**
C	112.48	86.57	433.25	77.87	**63.60**
D	79.08	61.45	434.69	56.04	**48.27**

On the other hand, we are interested in knowing the performance achieved with application of preconditioners available in the *PETSc* library. So, under

same conditions, we have obtained the times represented in Table 2. In the same manner, we can observe the advantage of use preconditioning techniques. If we make an analogous analysis as before paragraph, we will find that we have reduced the execution time of method D to almost 60% when it is combined with a *Block Jacobi* preconditioner.

Table 2. Sequential execution times (*secs*) in *PETSc*

METHOD	PCNONE	JACOBI	BJACOBI
A	290.17	218.00	**176.28**
B	148.72	104.71	**84.23**
C	85.45	62.19	**47.09**
D	61.81	45.21	**36.39**

6.2 Parallel Execution Times

For the parallel case with *Hypre*, we present only those with the preconditioner *Euclid*, due to presents the best performance. So, analysis of Table 3 shows the advantage of use parallel computing to reduce even more the sequential execution times for all methods. For our case, this reduction is almost 10% for all methods over their sequential time when we use $p = 16$ processors.

Also, it is necessary to emphasize, that in the sequential case, the influence of preconditioning in the methods considered. For example, the execution time for method A with $p = 16$ processors without preconditioning (*PCNONE*) registered 36.71 *secs* (This value is not showed in Table 3), but when method A is combined with *Euclid* preconditioning and equal number of processors, this time is reduced to a 60% of its value.

Table 3. Parallel execution times (*secs*) in *Hypre* using *Euclid* preconditioner

METHOD	$p = 1$	$p = 2$	$p = 4$	$p = 6$	$p = 8$	$p = 10$	$p = 14$	$p = 16$
A	236.66	128.22	71.94	49.23	38.53	31.68	24.11	22.06
B	111.73	62.82	33.54	23.27	18.16	14.95	11.38	10.51
C	63.60	35.39	19.62	13.46	10.28	8.70	6.84	6.12
D	48.27	26.02	14.64	9.84	7.89	6.39	4.88	4.45

For parallel execution times in *PETSc*, we represent only execution times with preconditioner *Block Jacobi*, because it showed the best ones. These times are represented in Table 4. As in the case of the *Hypre* library, we can observe a reduction of the sequential time almost 12% of its value for most cases in all methods. Now, with respect the use of preconditioning, we can observed that sequential times has been reduced almost 8% for all methods too.

Table 4. Parallel execution times (*secs*) in *PETSc* using *Block Jacobi* preconditioner

METHOD	$p=1$	$p=2$	$p=4$	$p=6$	$p=8$	$p=10$	$p=14$	$p=16$
A	176.28	102.18	58.27	42.49	33.70	29.34	23.25	21.93
B	84.23	48.91	27.51	19.95	15.87	13.85	10.88	10.03
C	47.09	28.69	16.31	11.65	9.19	8.04	6.50	5.95
D	36.39	21.37	12.04	8.74	6.92	6.06	4.77	4.60

6.3 Speedup and Efficiency

Parallel performance metrics such as *speedup* and *efficiency*[20] of the best combination of solver and preconditioner are presented in Table 5. Generally speaking, we can see that parallel performance does not degrade for down to 16 processors in both libraries. However, for the case of *Hypre* library, even when registered slightly execution times higher than *PETSc*, it shows a better parallel performance. For example, for the case of $p = 16$ processors using *Hypre*, shows a speedup of 10.8 against the value 7.9 registered using *PETSc*. This fact, talk us about the efficiency of the library in use of the cluster resources.

Table 5. Speedup and Efficiency

p	*Hypre*			*PETSc*		
	T_p	S_p	E_p	T_p	S_p	E_p
1	48.27	1.0	100%	36.39	1.0	100%
2	26.02	1.9	93%	21.37	1.7	85%
4	14.64	3.3	82%	12.04	3.0	76%
6	9.84	4.9	82%	8.74	4.2	69%
8	7.89	6.1	76%	6.92	5.3	66%
10	6.39	7.6	76%	6.06	6.0	60%
14	4.88	9.9	71%	4.77	7.6	54%
16	4.45	10.8	68%	4.60	7.9	49%

If we make a comparison of parallel execution times for both libraries , we shall observe that in terms of performance, *PETSc* library is slightly better than *Hypre*(see Fig. 3). However, in general both libraries had showed acceptable performance for our test case.

7 Conclusions

Efficient solution of large-sparse linear systems of equations related to the 3D multi-group time-dependent Neutron Diffusion Equation play a central role in stability and security studies of nuclear reactors. We have shown in this paper the results of four Second-Degree Iterative Methods applied to test matrices of

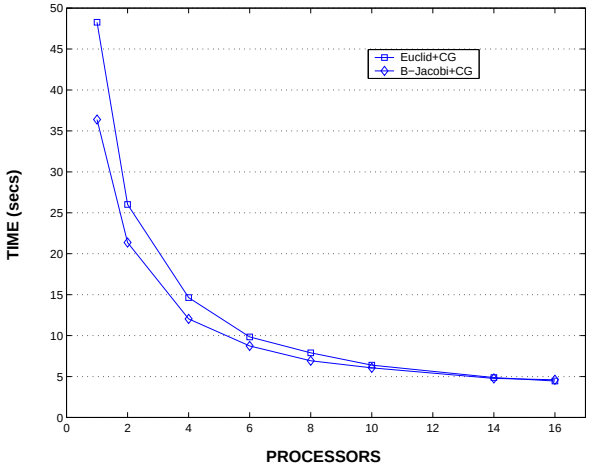

Fig. 3. Parallel execution times for *Hypre* and *PETSc*

real nuclear reactor. In addition, these methods have been combined with a set of preconditioners contained in two important libraries: *PETSc* and *Hypre*. For *PETSc* library, the preconditioners applied were *Jacobi* and *block Jacobi*. For *Hypre* library, the preconditioners applied were Diagonal Scaling, BoomerAMG, ParaSails and Euclid. All the preconditioners were combined with the method Conjugate Gradient. The preconditioners more efficient were *Block Jacobi* and *Euclid* for *PETSc* and *Hypre* libraries respectively. We have also found that *PETSc* preconditioners presented a better performance with respect to *Hypre* preconditioners.

We have used parallel computing to decrease the sequential time and we have achieved acceptable parallel performance for our test case.However, we have found higher values of speedup and efficiency in the case of *Hypre*. This means, that the algorithms of *Hypre* use the computer resources in a more efficient manner.

In general, the *Hypre* and *PETSc* libraries have shown a good parallel performance for our test case. Application of Second-Degree Iterative Methods combined with preconditioning techniques have helped to speedup their convergence rate in the solution of our test matrices.

Acknowledgments. This work has been supported by Spanish MEC and FEDER under Grants ENE2005-09219-C02-02, ENE2005-09219-C02-01 and SEIT-SUPERA-ANUIES (México).

References

1. Scalable Linear Solvers HYPRE High-Performance Preconditioning, `https://computation.llnl.gov/casc/linear_solvers/sls_hypre.html`

2. HYPRE team: HYPRE Reference Manual. UCRL-CODE-222953 - Software Version 2.2.0, Center for Applied Scientific Computating Lawrence Livermore National Laboratory (2006)
3. HYPRE team: HYPRE User's Manual. UCRL-CODE-222953 - Software Version 2.2.0, Center for Applied Scientific Computating Lawrence Livermore National Laboratory (2007)
4. Balay S., Gropp W.D., McInnes L.C., Smith B.F.: *PETSc* home page http://www.mcs.anl.gov/*PETSc* (2002)
5. 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)
6. 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
7. Weston J.R., Stacey M.: SpaceTime Nuclear Reactor Kinetics. Academic Press (1970)
8. Henry A.F.: Nuclear Reactor Analysis. The M.I.T Press (1975)
9. 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)
10. 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
11. D.M. Young.: Iterative Solution of Large Linear Systems. Academic Press Inc.,New York, N.Y. (1971)
12. Flores-Sánchez, O., Vidal, V., García, V., Flores-Sánchez, P.: Sequential and Parallel Resolution of the Two-Group Transient Neutron Diffusion Equation using Second-Degree Iterative Methods. In: 7th International Conference on High-Performance Computing for Computational Science - VECPAR 2006, pp. 426–438. Springer-Verlag, Berlin Heidelberg (2007)
13. Stacey W.M.: Space-Time Nuclear Reactor Kinetics. Academic Press, New York (1969)
14. 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
15. 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
16. Hébert A.: Development of the Nodal Collocation Method for Solving the Neutron Diffusion Equation. Ann. Nucl. Energy, 14(10), (1987) 527-541
17. Y. Saad. Iterative Methods for Sparse Linear Systems PWS Publishing Company, Boston, MA (1996)
18. Jacquard - Opteron Cluster`http://www.nersc.gov/nusers/systems/jacquard/`
19. National Energy Research Scientific Computing Center `http://http://www.nersc.gov/`
20. 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)
21. Blomstrand J.: The KKL Core Stability Test, conducted in September 1990. ABB Report, BR91-245 , (1992)
22. D. Ginestar and J. Marín and G. Verdú.: Multilevel methods to solve the neutron diffusion equation. Applied Mathematical Modelling, 25, (2001) 463-477