A Domain Decomposition Method Applied to The Simplified Transport Equations

BARRAULT Maxime
EDF R&D Clamart, France
maxime.barrault@edf.fr

LATHUILIÈRE Bruno
EDF R&D Clamart, France and
INRIA Bordeaux - Sud-Ouest
bruno.lathuiliere@edf.fr

RAMET Pierre
INRIA Bordeaux - Sud-Ouest
and LaBRI UMR5800, France
ramet@labri.fr

ROMAN Jean
INRIA Bordeaux - Sud-Ouest
and LaBRI UMR5800, France
roman@labri.fr

Abstract

The simulation of the neutron transport inside a nuclear reactor leads to the computation of the lowest eigen pair of a simplified transport operator. Whereas the sequential solution at our disposal today is really efficient, we are not able to run some industrial cases due to the memory consumption and the computational time. This problem brings us to study parallel strategies.

In order to re-use an important part of the solver and to bypass some limitations of conforming cartesian meshes, we propose a non overlapping domain decomposition based on the introduction of Lagrange multipliers. The method performs well on up to 100 *processors for an industrial test case.*

Keywords: *domain decomposition, neutron transport, Lagrange multiplers, Krylov based Uzawa algorithm.*

1 Introduction

The simulation of the neutron transport inside a nuclear reactor leads to the computation of the lowest eigen pair of a simplified transport operator [9]. This computation is done by an accelerated power inverse algorithm. At each iteration, a linear system is solved inexactly by a block Gauss-Seidel algorithm. For our application, one Gauss-Seidel iteration is already sufficient to ensure the right convergence of the power algorithm. For the approximate resolution of each linear system, we propose a non overlapping domain decomposition based on the introduction of Lagrange multipliers[12] in order to :

- get a parallel algorithm, which allows to get round memory consumption problem and to reduce the computational time;
- deal with different numerical approximations (mesh size and finite element order) in each subdomain. With this feature, we want to do the right amount of computation[1] for the requested accuracy. To use such non-conforming approximation, the physical knowledge of the problem is needed;
- minimize the code modifications in our industrial solver. The introduction of Lagrange multipliers allows us to use in each subdomain the sequential solver with the same boundary conditions as before.

Some other domain decomposition approaches have previously been studied in the context of neutron physics [4, 8]. In [7, 5], the authors propose a domain decomposition method based on an additive Schwartz algorithm with Robin transfer condition for diffusion approximation. This method is very efficient but implies to find a good parameter α for the Robin transfer condition.

In a first part, the simplified transport equations SPn and the basis of the domain decomposition method we propose are presented. The second part is dedicated to the description of the parallel implementation of a domain decomposition technique for our problem. Finally, numerical results, obtained with our parallel implementation, on a standard benchmark (BenchMarkAIEA [6]) are provided. Whereas the proposed domain decomposition method can get round the limitation of conforming cartesian meshes, the numer-

[1] However in a parallel context, it creates difficulties to find a well balanced partition.

ical results only point out the gain of the parallel method (computational time and memory consumption).

2 Introduction of the domain decomposition method

2.1 Presentation of the problem

The simplified transport equations SPn with two energy groups[2], lead for a reactivity calculation to the following algebraic problem :

Find the highest k_{eff} such that :

$$AX = \frac{1}{k_{eff}} FX \tag{1}$$

where

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \; F = \begin{pmatrix} F_{11} & 0 \\ 0 & F_{22} \end{pmatrix} \; X = \begin{pmatrix} X_1 \\ X_2 \end{pmatrix}.$$

- X_g corresponds to the degrees of freedom of neutron flux for energy group g;
- A is the transport matrix;
- F is called the fission matrix;
- k_{eff} is the highest eigenvalue of the operator $A^{-1}F$. It is called the effective multiplicative coefficient. $k_{eff} = 1$ means that the chain reaction is stable, $k_{eff} < 1$ means that the chain reaction slows down and $k_{eff} > 1$ means that the chain reaction spirals out of control.

In the $SP1$ approximation, the matrices A_{11} and A_{22} are four block matrices as they come from a discretization with Raviart-Thomas finite elements [1] of a mixed dual formulation of a diffusion problem. More precisely we have, for each group g,

$$A_{gg} = \begin{pmatrix} A_x^g & & & -B_x \\ & A_y^g & & -B_y \\ & & A_z^g & -B_z \\ {B_x}^t & {B_y}^t & {B_z}^t & T^g \end{pmatrix} \quad X_g = \begin{pmatrix} J_x^g \\ J_y^g \\ J_z^g \\ \phi^g \end{pmatrix}$$

where

- ϕ^g contains the degrees of freedom of the flux variable;
- J_x^g, J_y^g and J_z^g contain the degrees of freedom of the current variable for each space direction;
- A_x^g, A_y^g and A_z^g are banded matrices (tridiagonal with $RT0$ and pentadiagonal with $RT1$);
- T^g is a positive definite diagonal matrix;
- B_x, B_y and B_z are rectangular sparse matrices with 1 and -1 non zero values.

In the general case, the discretization of the SPn equations leads for each group g to a matrix A_{gg} made of $\frac{n+1}{2}$ coupled diffusion systems.

To solve problem (1), a Generalized Power Inverse Iteration Algorithm is used [11]. At each inverse power iteration, a linear system

$$AX = S \tag{2}$$

has to be solved. As A is very large and sparse, the resolution of (2) is done by a Gauss-Seidel algorithm which needs the computation of the product by A_{gg}^{-1}.
To solve each linear system with each matrix A_{gg}, the flux ϕ^g is eliminated in order to bring us to a positive definite linear system associated with a three-blocks matrix. Its resolution is done by a Gauss-Seidel algorithm where the inversion on each space direction is done exactly by an LU factorization.

The overall algorithm is resumed in the Algorithm 1 where three loops are imbricated[3] with an inner/outer process. For each iterative algorithm, the initial guest is the result of the previous resolution of the system with the same matrix. For our applications, we obtain the best performance by doing a single iteration for all Gauss-Seidel loops. One iteration is the best compromise between the iteration number of the inverse power algorithm and the computational time per iteration.

With a Gauss-Seidel algorithm restricted to one iteration there is only one forward/backward substitution for each dimension of space, so pipeline techniques can not be applied to increase the parallel efficiency. This is one reason for the introduction of domain decomposition techniques.

2.2 A domain decomposition method based on Lagrangian multipliers

The domain decomposition method we propose is based on the introduction of Lagrange multipliers to deal with different numerical approximations (mesh size and finite element order) between two adjacent subdomains. At each inverse power iteration, the linear system to solve (2) is then replaced by a new linear system that can be written, for a partition composed of two subdomains, as:

$$\begin{pmatrix} A_{d=1} & 0 & C_{1\to 2} \\ 0 & A_{d=2} & C_{2\to 1} \\ C_{1\to 2}^t & C_{2\to 1}^t & 0 \end{pmatrix} \begin{pmatrix} X_{d=1} \\ X_{d=2} \\ \Lambda \end{pmatrix} = \begin{pmatrix} S_{d=1} \\ S_{d=2} \\ 0 \end{pmatrix} \tag{3}$$

where

[2] The use of two energy groups corresponds to industrial cases

[3] In the general case (SPn equations), one new loop is added, due to the block Gauss-Seidel algorithm over the harmonics which are coming from angular discretization.

Algorithm 1: Global algorithm

```
/* Inverse power iteration algorithm */
while !Convergence do
    /* Inversion A by Gauss-Seidel */
    while !Convergence do
        foreach group g do
            /* Inversion of A_gg by Gauss-Seidel */
            while !Convergence do
                foreach space direction d do
                    Compute RhS ;
                    Solve (A_d^g - B_d^t T_g^{-1} B_d) J_d^g = RhS ;
```

- the unknowns are the solution vectors $X_{d=1}$ and $X_{d=2}$ defined on each subdomain;
- the matrices $C_{1\to 2}$ and $C_{2\to 1}$ are coupling matrices between the two subdomains, they depend on the numerical way to discretize each subdomain;
- Λ is the vector which contains the degrees of freedom associated with the Lagrange multipliers for the matching of $X_{d=1}$ and $X_{d=2}$ on the interface between the two subdomains.

More rigorously, (3) comes from the discretization of the variational form of the simplified transport equations written with the Lagrangian multipliers to take into account the non-matching grids at the interface of the subdomains [2]. To solve (3), we rewrite it as a saddle point system:

$$\begin{pmatrix} A_{DD} & C \\ C^t & 0 \end{pmatrix} \begin{pmatrix} X \\ \Lambda \end{pmatrix} = \begin{pmatrix} S \\ 0 \end{pmatrix} \tag{4}$$

with

$$A_{DD} = \begin{pmatrix} A_{d=1} & 0 \\ 0 & A_{d=2} \end{pmatrix} \quad C = \begin{pmatrix} C_{1\to 2} \\ C_{2\to 1} \end{pmatrix} \tag{5}$$

$$X = \begin{pmatrix} X_{d=1} \\ X_{d=2} \end{pmatrix} \quad S = \begin{pmatrix} S_{d=1} \\ S_{d=2} \end{pmatrix}. \tag{6}$$

3 Practical implementation of the domain decomposition method

To solve (4), a Krylov based Uzawa algorithm [3] is used. The principle of this algorithm is to solve the problem defined on the interface between the subdomains, that is to say, find Λ solution of

$$C^t A_{DD}^{-1} C\Lambda = C^t A_{DD}^{-1} S \tag{7}$$

with a Krylov algorithm where a linear combination of subdomain vectors associated to the Krylov interface vectors is added for each Krylov iteration to compute the solution. When two energy groups are considered, the matrix $C^t A_{DD}^{-1} C$ is not symmetric. That is why we use a Uzawa-BiCGStab algorithm (see Algorithm 2) that is an adaptation of a BiCGStab algorithm [10] (with the vector r_0^* set to 1) for saddle point system:

- at line 2, the residual r is computed using only one application of A_{DD}^{-1} as the computation of the right hand side $C^t A_{DD}^{-1} S$ is not necessary;
- at lines 3 and 4, we store an additional vector for each subdomain;
- at line 5, we compute X thanks to the additional vectors stored for each subdomain;
- since this algorithm is inside a inverse power algorithm, Λ is initialized at line 1 by Λ_0 provided by the previous iteration of the inverse power algorithm.

Algorithm 2: Uzawa-BiCGStab algorithm

1 $\Lambda = \Lambda_0; \quad r_0^* = 1$;
2 $X = A_{DD}^{-1}(S - C\Lambda); \quad r = C^T X$;
$p = r$;
while *!Convergence* **do**
3 $X_p = A_{DD}^{-1} Cp; \quad Sp = C^T X_p$;
$\alpha = \dfrac{< r, r_0^* >}{< Sp, r_0^* >}$;
$s = r - \alpha Sp$;
4 $X_s = A_{DD}^{-1} Cs; \quad Ss = C^T X_s$;
$\omega = \dfrac{< Ss, s >}{< Ss, Ss >}$;
$\Lambda = \Lambda + \alpha p + \omega s$;
5 $X = X - \alpha X_p - \omega X_s$;
$r_{old} = r$;
$r = s - \omega Ss$;
$\beta = \dfrac{< r, r_0^* >}{< r_{old}, r_0^* >} \times \dfrac{\alpha}{\omega}$;
$p = r + \beta(p - \omega Sp)$;
end

We use in parallel the sequential solver, in each subdomain, to compute the product by A_{DD}^{-1} (lines 2, 3, 4). Since the solver is iterative, we have to initialize carefully the inner iterative process:

- at line 2, the initial guest of the inner solver is the vector X which is provided by the previous iteration of the inverse power algorithm. Indeed at convergence this vector is invariant;

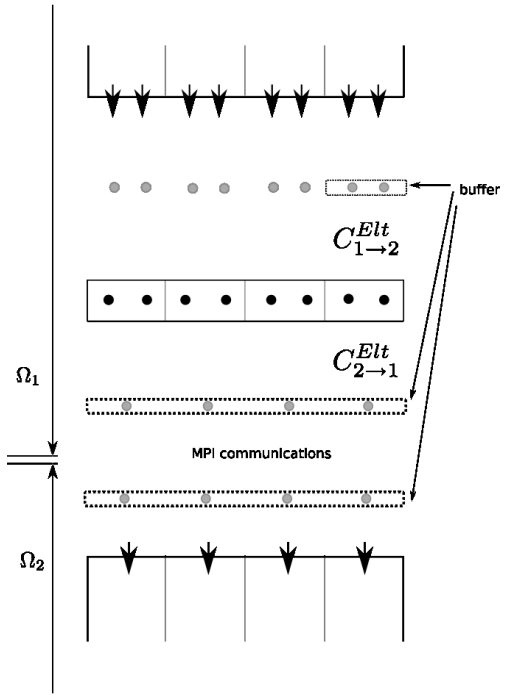

Figure 1. Communication Scheme

- at lines 3 and 4, the solvers use the null vector as initial guest.

Compared with algorithm 1, we finally add two nested loops: the first one, over subdomains, is implicit as we are using a SPMD[4] paradigm. The other loop is added to get convergence of Uzawa-BiCGStab algorithm. For this loop, we choose to set a fixed number of iterations in the same way as the original sequential solver. In futur works, we will have to define finer criteria to improve the convergence.

In our implementation, a subdomain is mapped on each processor. For instance, processor 1 has to store $A_{d=1}$, $F_{d=1}$ and $X_{d=1}$. Thanks to the specific finite elements and to the choice of the basis functions for the Lagrange multipliers Λ, each degree of freedom on the interface is linked between only two subdomains.

Interface vector Λ and interface vectors issued from BiCGStab algorithm are decomposed naturally following the interfaces between subdomains. For each interface vector ω, we denote $\omega_{i,j}$, the part of the interface vector between subdomains i and j ($\omega_{j,i}$ refers to the same part). Since in the case of non-matching grids (or different order of finite elements) the application of C can reduce the amount of data to transfer, the matrices $C_{i\to j}$ and $C_{j\to i}$ and $\omega_{i,j}$ (for each interface vector ω) are stored on the same processor. In order to minimize the communication, we choose to store this data on the processor with the finest discretization (see Figure 1). This processor is called the master for the interface between the subdomains i and j. A subdomain can be master for an interface and slave for another one. In the case of matching grids and same finite element order, the subdomain with the smallest index is defined as master. Since for each couple (i, j) of neighboring subdomains, $C_{i\to j}$ is made of small identical blocks, only the elementary block $C^{Elt}_{i\to j}$ is stored.

To ensure high-performance computing, there is no master/slave communication scheme: all the scalar products computed on solution vectors inside the inverse power algorithm and on the interface vectors inside the BiCGStab algorithm are computed locally and then reduced thanks to collective communications using "MPI_Allreduce" routine. Moreover, a communication will occur only between neighboring subdomains to compute the products by C and C^t. Finally, persistent communications are implemented: for each vectors w and X, the computations of Cw (see Algorithm 3) and $C^T X$ (see Algorithm 4) use the same buffer (see Figure 1).

Algorithm 3: Compute $X+ = C\Lambda$

forall *j master interface* **do**
  Compute $C_{j\to i}\Lambda_{i,j}$;
  Start Sending Data to j;
forall *j slave interface* **do**
  Start receiving Data from j;
forall *j master interface* **do**
  Compute $C_{i\to j}\Lambda_{i,j}$ to update X_i;
  Wait data sent to j;
forall *j slave interface* **do**
  Wait Received Data from j;
  Use $C_{i\to j}\Lambda_{i,j}$ to update X_i;

Algorithm 4: Compute $\omega = C^T X$

forall *j master interface* **do**
  Start receiving Data from j;
  Compute $\omega_{i,j} = C^t_{i\to j} X_i$;
forall *j slave interface* **do**
  Compute $tmp_{i,j} = C^t_{i\to j} X_i$;
  Start Sending $tmp_{i,j}$ to j ;
forall *j master interface* **do**
  Wait received data($tmp_{i,j}$) from j;
  $\omega_{i,j} += tmp_{i,j}$;
forall *j slave interface* **do**
  Wait Data sent to j ;

[4]Single Program Multiple Data

4 Experimental results

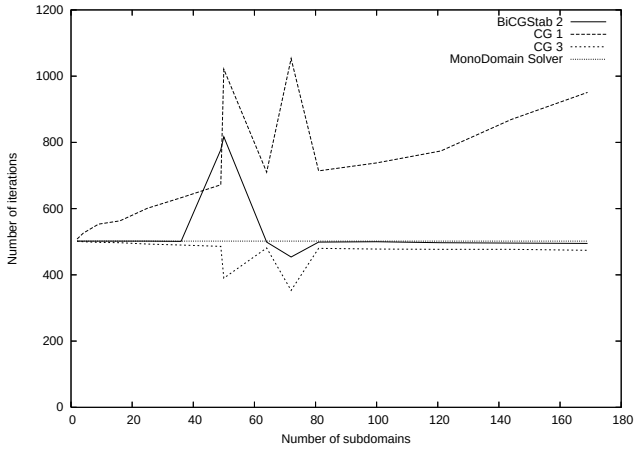

Figure 2. Number of iterations

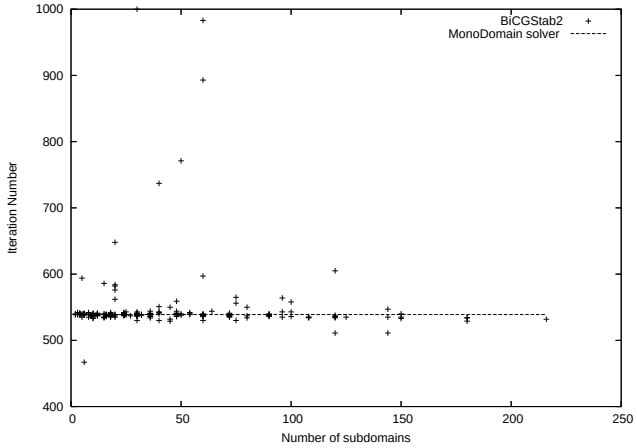

Figure 3. Iteration number for 215 partitions on a small test case ($17 \times 17 \times 19$)

To validate the proposed method we consider a 3D standard benchmark published by IAEA [6]. Two groups of energy are involved. The definition of physical constants is based on a cartesian mesh $17 \times 17 \times 19$ made of five materials. The computational mesh used is $289 \times 289 \times 38$ with the Raviart Thomas finite element $RT0$. The stiffness matrices are computed with a Gauss-Legendre integration. The axis x, y and z are partitioned into N_x, N_y and N_z parts. This leads to a partition of the domain into $N_x \times N_y \times N_z$ subdomains. The numerical results are

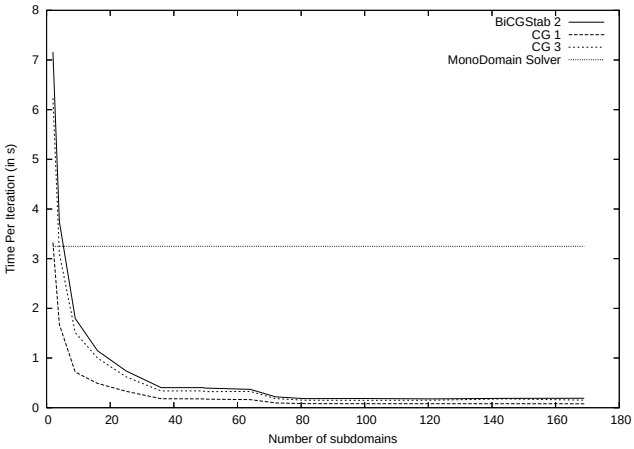

Figure 4. Time per iteration

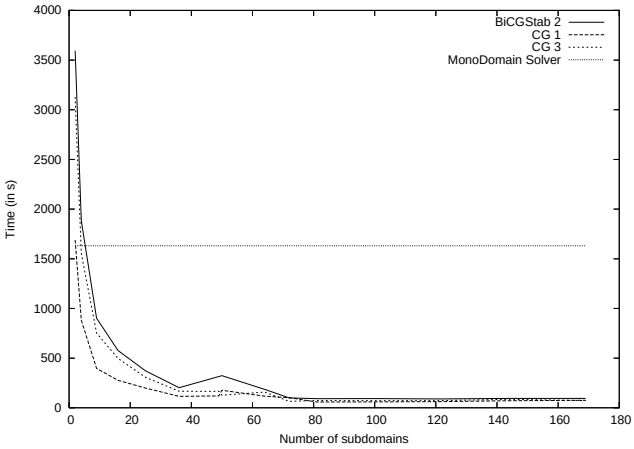

Figure 5. Total time

given with the following partitions: $(N_x \times N_y \times N_z) \in \{(2,1,1), (2,2,1), (3,3,1), (4,4,1), (5,5,1), (6,6,1), (7,7,1), (5,5,2), (8,8,1), (6,6,2), (9,9,1), (10,10,1), (11,11,1), (12,12,1), (13,13,1)\}$. With this choice, the jumps of physical data are located inside the subdomains. In order to compare with the sequential solver, we consider matching grids.

The inverse power algorithm has the following convergence criteria:

$$\frac{\|\phi_n - \phi_{n-1}\|}{\|\phi_n\|} < 10^{-6} \quad and \quad \frac{|\lambda_n - \lambda_{n-1}|}{|\lambda_n|} < 10^{-5}$$

where ϕ_n is the flux part of the estimated eigenvector and λ_n the estimated eigenvalue at iteration n.

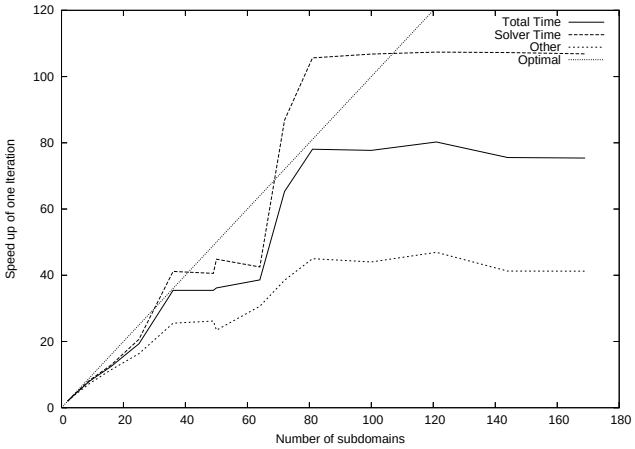

Figure 6. Speedup of one iteration of BiCGStab2

To run our code we used the cluster of EDF R&D Rend-Vous with the following properties:

- Processors: 208 nodes with Intel(R) Xeon(TM) CPU 3.40GHz 2MB Cache;
- Memory: 4 GB PC3200 DDR2 with DualChannel;
- Network: Infiniband openib-2.0.5;
- MPI Implementation: mpich 1.2.7;
- Compiler: g++ 4.1.2 (64 bit);
- OS: Debian etch.

During batch submissions, a node per subdomain is reserved to avoid concurrence for memory access.

Results with Uzawa-BiCGStab algorithm with two iterations are presented. Surprisingly, an Uzawa-CG algorithm (applied to a non-symmetric system) converges to the correct solution: the results with one and three iterations are also presented. Uzawa-CG algorithm is interesting because the computational cost is less important. In the mono-domain solver (sequential solver), only one call to local solver is done whereas five calls are needed in our multi-domain parallel solver with two iterations of BiCGStab algorithm (BiCGStab2)[5]. So, in parallel, there is at least a penalty by a factor of 5 compared to the sequential mono-domain solver. With CG1 the penalty per power iteration is only 2 (4 for CG3).

[5]In the same way CG1 (resp. CG3) refers to the multi-domain approach with one (resp. three) iteration of Conjugate Gradient

All the inner convergence loops have a fixed number of iterations. So to analyse the impact of domain decomposition on the convergence, the iteration number of the generalized inverse power algorithm is meaningful (see figure 2). Without reliable explanation, with two solvers (BiCGStab2, CG3), the number of iterations is not an increasing function of the number of subdomains neither of the size of interface problem.

For a given number of subdomains the choice of the partition has a great impact on the convergence speed. With a small test case (the computational mesh is set to $17 \times 17 \times 19$), we run 215 partitions and we observe that a same number of subdomains can lead to really different behaviours (see figure 3). At this time we do not know how to automatically choose a good partition. It explains some[6] pathological cases (for instance the partition with 49 or 50 subdomains for BiCGStab2 on figure 2). We have to mention that the number of iterations is remarkably stable with the CG3 solver (see figure 2).

Of course, the time spent inside each local solver is smaller when the number of subdomains increases, so the time spent in one iteration is decreasing (see figure 4). To measure the time, we consider only the execution of the inverse power iteration algorithm: it means that the stiffness matrices construction is not measured. The global behaviour is time decreasing with the number of subdomains but due to the jump in the iteration number of the inverse power algorithm, we observe one increasing zone (see figure 5).

Although our domain decomposition approach leads to a penalty factor, with 8 subdomains (resp. 4), the multi-domain approach BiCGStab2 (resp. CG1) is faster than the sequential mono-domain solver.

This domain decomposition method reduces significantly the execution time and, furthermore, we can run larger industrial test cases. For speedup studies, the multi-domain approach with two subdomains is defined as reference solver. On the figure 6, the speedup for one iteration of the inverse power algorithm is plotted. On this figure, we add one curve concerning only the time spent in the solver A_{DD}^{-1} and another one for the rest of the code (including products by F, C and C^t, scalar products, ...). The following results are obtained :

- on 16 subdomains, the global time is reduced by 6.2 with BiCGStab2 compared to the parallel two-domain solver (reduced by 6.1 with CG1). However, compared to the sequential mono-domain solver, the global time is only reduced by 2.8 with BiCGStab2 (reduced by 5.8 with CG1);

[6]On the small test case (Figure 3) there is only 4.2% (resp. 1.4%) of partitions which imply an increase in iterations number superior to 10% (resp. 50%) compared to the sequential solver)

- on 81 subdomains, the global time is reduced by 39.3 with BiCGStab2 compared to the parallel two-domain solver (reduced by 28.7 with CG1). In the same way, compared to the sequential mono-domain solver, the global time is only reduced by 17.8 with BiCGStab2 (reduced by 27.7 with CG1).

From 64 processors, the subdomains are enough small and thanks to cache effect (see figure 6), the local solver is super linear (for 72, 91 and 100 subdomains). The best efficiency is obtained for 81 subdomains. But for more than 81 subdomains there is not enough work in each subdomain and the time taken by the solver does not decrease.

On a test case which is, in term of size, not far from real industrial cases, these results prove the efficiency of our domain decomposition method and its implementation up to 100 processors. As expected, the global communications can not be neglected with more than 121 processors. However the main bottleneck remains the local solver with small subdomains. Some investigations need to be done to improve the efficiency of the sequential solver with small subdomains.

5 Conclusions

The proposed domain decomposition method in the difficult context of an approximate resolution of each linear system at each power iteration reveals very reliable. Besides, it exhibits a good efficiency which allows us to realize some computations beyond the reach of standard workstations. Finally, we obtain very good results in using a conjugate gradient algorithm instead of a BiCGStab algorithm for the resolution of the interface problem. We have to validate such behaviors on more complicated test cases (with more groups and more harmonics) and on more processors.

It is still a long way to obtain a very powerful method for industrial applications. On the one hand, the efficiency of using a conjugate gradient algorithm has to be investigated. On the other hand, we have to study the best way to introduce the process based on Chebyshev polynomials for the acceleration of the power algorithm. Indeed this process (when activated) leads to a significant reduction of the computational time for the sequential monodomain solver. Furthermore more experimentations have to be performed in order to find a way to choose a good partition of the computational mesh. Lastly, it is necessary to identify the best place in the overall algorithm of the subdomain loop. For the moment, it is placed between the inverse power and the energy group loops.

Finally we acknowledge ANRT[7] for financial support.

[7] Association Nationale de la Recherche Technique (http://www.anrt.asso.fr/)

References

[1] J.J. Lautard A.M. Baudron and D. Schneider. Mixed dual methods for neutronic reactor core calculations in the CRONOS system. In *Reactor Physics and Environmental Analysis of Nuclear Systems*, 1999.

[2] M. Barrault and N. Tonnel. Une méthode de décomposition de domaine pour le traitement SP1 de l'équation de transport des neutrons. Note Interne HI-23/05/035/A, EDF, France, 2005.

[3] F. Bertrand and P. A. Tanguy. Krylov-Based Uzawa Algorithms for the Solution of the Stokes Equations Using Discontinuous-Pressure Tetrahedral Finite Elements. *Journal of Computational Physics*, 181:617–638, September 2002.

[4] E. Girardi. *Couplage de méthodes et décomposition de domaine pour la résolution des équations de transport*. PhD thesis, Université d'Evry Val-d'Essonne, 1999.

[5] P. Guérin. *Méthode de synthèse modale pour la résolution numérique des équations de la neutronique en formulation mixte duale, application à des calculs 3D hétérogènes*. PhD thesis, Université Paris VI, 2007.

[6] B. Micheelsen. 3D IAEA Benchmark Problem. Technical report, IAEA, 1977.

[7] A.M. Baudron P. Guérin and J.J. Lautard. Domain decomposition methods for core calculations using the minos solver. In *Joint International Topical Meeting on Mathematics & Computation and super Computing in Nuclear Applications*, 2007.

[8] K. Pinchedez. *Calcul parallèle pour les équations de diffusion et de transport homogènes en neutronique*. PhD thesis, Université Paris XI, 1999.

[9] C.G. Pomraning. Asymptotic and variational derivation of the simplified pn equations. *American Nuclear Energy, 20, 9*, pages 623–637, 1993.

[10] Y. Saad. *Iterative Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2003.

[11] Henk A. van der Vorst. Computational methods for large eigenvalue problems. In J.L. Lions RG. Ciarlet, editor, *Handbook of Numerical Analysis*, volume 8, pages 3–179. North-Holland (Elsevier), 2002.

[12] B.I. Wohlmuth. *Discretization methods and iterative solvers based on domain decomposition*. Springer, 2001.