

PARAFISH: a Parallel $FE - P_N$ Neutron Transport Solver based on Domain Decomposition

S. Van Criekingen ^a, F. Nataf ^b and P. Havé ^c

^a Karlsruhe Institute of Technology, Germany , serge.criekingen@kit.edu

^b J.-L. Lions Laboratory, Paris VI University, CNRS UMR 7598, France

^c Institut Français du Pétrole, Rueil-Malmaison, France

August 2010, accepted by Annals of Nuclear Energy

A new core solver named PARAFISH is presented for the solution of large neutron transport core calculations. The second-order even-parity form of the time-independent Boltzmann transport equation is solved using an innovative algebraic domain-decomposition method. The spatio-angular discretization is performed using non-conforming finite elements and spherical harmonic expansions (P_N method). The PARAFISH code allows one processor to handle more than one domain. This enables proper evaluations of the speed-up. Also, this enables to show that the domain-decomposition method not only performs well in parallel calculations, but also has an inherent acceleration potential. That is, it yields acceleration even without increasing the number of processors.

KEYWORDS: domain decomposition, parallel 3-D transport, even-parity neutron transport equation, spherical harmonics, non-conforming finite elements

1 Introduction

Domain decomposition (DD) methods were introduced by Schwarz in the late nineteenth century (Schwarz 1869). About one century later, they arouse a new wave of interest due to the development of parallel computers . Application of such methods to neutron transport was already addressed by de Oliveira et al. (1995), Park et al. (2005) and by Guérin et al. (2007).

The basic idea of DD consists first in splitting up a given global domain into several subdomains, which can either overlap (“overlapping DD”) or simply share one or more interfaces (“non-overlapping DD”, in fact minimal overlapping). Note that the word

“subdomain” is often simply replaced by the word “domain”, when the context makes it clear whether the whole domain or only one (sub)domain is meant. After splitting, one proceeds iteratively with local resolutions (“local solves”) performed in each domain: at each iteration, one uses as boundary condition in one domain, the solution values computed in the neighboring domains. Schwarz (1869) proved the convergence of this iterative procedure (for the Poisson equation), and showed that the convergence speed is proportional to the amount of values “transferred” at each iteration of the procedure, that is, proportional to the overlap size. The larger the overlap, the faster the iterative process will converge. While the local solves can be performed alternately, they can also be performed simultaneously on parallel computers, and that is what we do here.

Increasing the overlap size to accelerate the DD procedure comes at the cost of more unknowns being duplicated and more data being transferred between processors. Many improvements were therefore developed to obtain fast convergence with small or no overlap. Considering non-overlapping domains, only interfaces are shared, and one can transfer not only point solution values, but also normal derivatives. One can moreover transfer a combination of point values and normal derivatives:

$$(\partial_n \Psi + \alpha \Psi), \tag{1}$$

with α a tunable coefficient. Using such Robin (or mixed) interface conditions, P.L. Lions (1990) proved the convergence of the iterative procedure on non-overlapping domain decompositions (for the generalized Poisson equation). P.L. Lions’ idea was implemented by Guérin et al. (2007) for the simplified transport SP_N approximation and using one processor per domain.

Here, we present in Section 2 an algebraic non-overlapping DD method (Section 2.1), and afterwards show that this method can be related to P.L. Lions’ idea (Section 2.2). The method is algebraic in the sense that it is defined at the matrix level. Next in Section 3, this method is implemented on the even-parity formulation of the Boltzmann transport equation (Lewis and Miller, Jr. 1984), yielding a new core solver named PARAFISH. The spatial discretization is performed using finite elements, more precisely non-conforming NC_4 and NC_6 finite elements (Lautard 1981; Van Criekingen 2007), since they appear particularly suitable for our DD method (Section 2.3). The angular discretization is performed using spherical harmonic expansions (P_N method), and any order N can be used. The PARAFISH code is implemented in C++, using MPI for parallel communications. Sequential and parallel numerical results on the 3-D Takeda 1 benchmark (Takeda et al. 1989) are respectively presented in Section 3.2 and 3.3.

2 Algebraic formulation

2.1 A Two-domain example in two dimensions

With \mathcal{A} a linear operator (e.g., the Boltzmann transport operator), we consider the problem

$$\mathcal{A}\Psi = Q \quad (2)$$

defined on two non-overlapping (sub)domains E^1 and E^2 sharing one common edge $\Gamma = E^1 \cap E^2$. In view of simplification, we consider in this section that, as depicted in Fig. 1, both domains E^1 and E^2 are made out of one single finite element (FE), namely at this point a lowest-order (i.e., linear) conforming Lagrangian FE with 4 nodes located at the vertices. The nodal point values of Ψ in E^i ($i = 1, 2$) are denoted by φ_j^i ($j = 1..4$), as illustrated in Fig. 1. Applying the FE method separately to E^1 and E^2 results in two

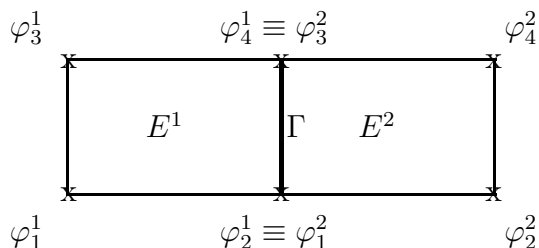


Figure 1: A two-domain example, where each domain is made out of one lowest-order conforming Lagrangian FE.

symmetric (assuming \mathcal{A} is self-adjoint) linear systems of the form

$$\begin{pmatrix} a_{11}^i & a_{21}^i & a_{31}^i & a_{41}^i \\ a_{21}^i & a_{22}^i & a_{32}^i & a_{42}^i \\ a_{31}^i & a_{32}^i & a_{33}^i & a_{43}^i \\ a_{41}^i & a_{42}^i & a_{43}^i & a_{44}^i \end{pmatrix} \begin{pmatrix} \varphi_1^i \\ \varphi_2^i \\ \varphi_3^i \\ \varphi_4^i \end{pmatrix} = \begin{pmatrix} q_1^i \\ q_2^i \\ q_3^i \\ q_4^i \end{pmatrix}, \quad i = 1, 2.$$

To couple these two linear systems together, we use the fact that, since they represent the unknown Ψ at the same physical point, the values φ_2^1 and φ_1^2 must be identical. (More precisely, these two values will asymptotically tend to the same value in an iterative process involving parallel calculations on the two domains.) Similarly, the values φ_4^1 and φ_3^2 must be identical. We therefore write

$$\begin{pmatrix} a_{11}^1 & a_{21}^1 & a_{31}^1 & a_{41}^1 & 0 & 0 & 0 & 0 \\ a_{21}^1 & a_{22}^1 & a_{32}^1 & a_{42}^1 & a_{11}^2 & a_{21}^2 & a_{31}^2 & a_{41}^2 \\ a_{31}^1 & a_{32}^1 & a_{33}^1 & a_{43}^1 & 0 & 0 & 0 & 0 \\ a_{41}^1 & a_{42}^1 & a_{43}^1 & a_{44}^1 & a_{31}^2 & a_{32}^2 & a_{33}^2 & a_{43}^2 \\ 0 & -1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & a_{21}^2 & a_{22}^2 & a_{32}^2 & a_{42}^2 \\ 0 & 0 & 0 & -1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & a_{41}^2 & a_{42}^2 & a_{43}^2 & a_{44}^2 \end{pmatrix} \begin{pmatrix} \varphi_1^1 \\ \varphi_2^1 \\ \varphi_3^1 \\ \varphi_4^1 \\ \varphi_1^2 \\ \varphi_2^2 \\ \varphi_3^2 \\ \varphi_4^2 \end{pmatrix} = \begin{pmatrix} q_1^1 \\ q_2^1 + q_1^2 \\ q_3^1 \\ q_4^1 + q_3^2 \\ 0 \\ q_2^2 \\ 0 \\ q_4^2 \end{pmatrix}. \quad (3)$$

In the matrix on the left-hand side of Eq. (3), the diagonal blocks can be made both symmetric by adding and subtracting the lines corresponding to a same duplicated unknown. After re-organization, the system (3) can be written in a general two-domain formulation:

$$\begin{pmatrix} A^{11} & A^{1\Gamma} & 0 & 0 \\ A^{\Gamma 1} & A_1^{\Gamma\Gamma} + I & A^{\Gamma 2} & A_2^{\Gamma\Gamma} - I \\ 0 & 0 & A^{22} & A^{2\Gamma} \\ A^{\Gamma 1} & A_1^{\Gamma\Gamma} - I & A^{\Gamma 2} & A_2^{\Gamma\Gamma} + I \end{pmatrix} \begin{pmatrix} \varphi^1 \\ \varphi^{\Gamma,1} \\ \varphi^2 \\ \varphi^{\Gamma,2} \end{pmatrix} = \begin{pmatrix} Q^1 \\ Q_1^{\Gamma\Gamma} + Q_2^{\Gamma\Gamma} \\ Q^2 \\ Q_1^{\Gamma\Gamma} + Q_2^{\Gamma\Gamma} \end{pmatrix} \quad (4)$$

where, if as above each domain contains only one FE, one has

$$\varphi^1 = \begin{pmatrix} \varphi_1^1 \\ \varphi_3^1 \end{pmatrix}, \varphi^{\Gamma,1} = \begin{pmatrix} \varphi_2^1 \\ \varphi_4^1 \end{pmatrix}, \varphi^2 = \begin{pmatrix} \varphi_2^2 \\ \varphi_4^2 \end{pmatrix}, \varphi^{\Gamma,2} = \begin{pmatrix} \varphi_1^2 \\ \varphi_3^2 \end{pmatrix},$$

$$A^{11} = \begin{pmatrix} a_{11}^1 & a_{31}^1 \\ a_{31}^1 & a_{33}^1 \end{pmatrix}, A^{1\Gamma} = A^{\Gamma 1T} = \begin{pmatrix} a_{21}^1 & a_{41}^1 \\ a_{32}^1 & a_{43}^1 \end{pmatrix},$$

$$A_1^{\Gamma\Gamma} = \begin{pmatrix} a_{22}^1 & a_{42}^1 \\ a_{42}^1 & a_{44}^1 \end{pmatrix}, I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \dots$$

Note that the global system matrix in (4) is not symmetric, but that the diagonal blocks

$$\mathbf{A}_1 = \begin{pmatrix} A^{11} & A^{1\Gamma} \\ A^{\Gamma 1} & A_1^{\Gamma\Gamma} + I \end{pmatrix} \quad \text{and} \quad \mathbf{A}_2 = \begin{pmatrix} A^{22} & A^{2\Gamma} \\ A^{\Gamma 2} & A_2^{\Gamma\Gamma} + I \end{pmatrix} \quad (5)$$

are both symmetric. Our DD method then consists in solving the global system (4) using an iterative method. Since the system matrix in (4) is not symmetric, Krylov-type methods such as BiCGStab and GMRES (Saad 2003) can be considered at this global level. These Krylov methods require preconditioning. We here implement a block-diagonal preconditioning, which implies that the blocks \mathbf{A}_1 and \mathbf{A}_2 have to be “inverted” (in fact, linear systems have to be solved) at each iteration. These “inversions” are performed in parallel on different processors, and will be therefore referred to as “local solves” in the sequel. The local solves can be performed using a direct or an iterative method, and can make use of the symmetry property of \mathbf{A}_1 and \mathbf{A}_2 . Because the local solves are meant only at preconditioning, iterative methods with relatively low convergence criteria are here the method of choice. In this view, we opted for the preconditioned conjugate gradient (PCG) method, with incomplete lower-upper (ILU, also known as incomplete Cholesky (IC) when symmetric) preconditioning. Such choice however means that the preconditioning of the global level is not constant, and in turn the flexible GMRES (FGMRES) (Saad 1993) is preferred to the BiCGStab method at that level.

Note also that, compared to a method in which the whole domain ($E_1 \cup E_2$ in Fig. 1) is not split into subdomains, the DD method presented here has a higher total number of unknowns since the interface unknowns are duplicated among the subdomains. However, this duplication enables the local solves to be performed in parallel.

2.2 Interpretation

As stated above, we solve Eq. (4) iteratively using the FGMRES method. However, an interpretation of our DD method requires to consider a more basic iterative approach. The most basic method to solve (4) iteratively consists in doing

$$\begin{pmatrix} A^{11} & A^{1\Gamma} & 0 & 0 \\ A^{\Gamma 1} & A_1^{\Gamma\Gamma} + I & 0 & 0 \\ 0 & 0 & A^{22} & A^{2\Gamma} \\ 0 & 0 & A^{\Gamma 2} & A_2^{\Gamma\Gamma} + I \end{pmatrix} \begin{pmatrix} \varphi^1 \\ \varphi^{\Gamma,1} \\ \varphi^2 \\ \varphi^{\Gamma,2} \end{pmatrix}^{n+1} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & -A^{\Gamma 2} & -A_2^{\Gamma\Gamma} + I \\ 0 & 0 & 0 & 0 \\ -A^{\Gamma 1} & -A_1^{\Gamma\Gamma} + I & 0 & 0 \end{pmatrix} \begin{pmatrix} \varphi^1 \\ \varphi^{\Gamma,1} \\ \varphi^2 \\ \varphi^{\Gamma,2} \end{pmatrix}^n + \begin{pmatrix} Q^1 \\ Q_1^{\Gamma\Gamma} + Q_2^{\Gamma\Gamma} \\ Q^2 \\ Q_1^{\Gamma\Gamma} + Q_2^{\Gamma\Gamma} \end{pmatrix}. \quad (6)$$

Following Nataf (1999), Eq. (6) can be interpreted as the matrix formulation of an algorithm whose continuous version is

$$\mathcal{A}\Psi^{n+1} = Q \quad \text{in } E^1 \text{ and } E^2, \quad (7)$$

$$(\mathcal{A}^\Gamma + \mathcal{I})\Psi_{E^1}^{n+1} = (\mathcal{A}^\Gamma + \mathcal{I})\Psi_{E^2}^n \quad \text{on } \Gamma, \quad (8)$$

$$(\mathcal{A}^\Gamma + \mathcal{I})\Psi_{E^2}^{n+1} = (\mathcal{A}^\Gamma + \mathcal{I})\Psi_{E^1}^n \quad \text{on } \Gamma, \quad (9)$$

where \mathcal{A} and Q are as in Eq. (2), the operator \mathcal{A}^Γ is the part of \mathcal{A} acting on Γ , and the operator \mathcal{I} is the identity operator. The interpretation of algorithm (7)-(9) is that, at each iteration,

- the considered problem (2) is solved in each domain (Eq. (7)),
- the boundary condition in one domain is given as a function of the unknown Ψ in the neighboring domain at the previous iteration (Eq. (8) and (9)).

The boundary conditions at domain interfaces is the way information is exchanged between domains in the iterative process. To illustrate these interface conditions, we further simplify our example by considering the Poisson equation, namely the case where $\mathcal{A} = \Delta$, the Laplacian operator. In this case, the integration by parts performed to obtain the weak form (standard in the FE theory) yields $\mathcal{A}^\Gamma = \partial_n$, that is, \mathcal{A}^Γ is the normal derivative on the interface. The interface condition enforces the continuity of

$$\partial_n \Psi + \Psi$$

through interfaces. This is a so-called Robin (or mixed) interface condition, already introduced in Eq. (1), here with a proportionality factor $\alpha = 1$. At each iteration, point values as well as normal derivatives of the unknown Ψ are transferred between neighboring domains, with here the same weight given to ‘‘Dirichlet’’ (point values) and ‘‘Neumann’’

(normal derivatives) data. Such procedure was shown to be convergent by P.L. Lions (1990). As mentioned in the introduction, the use of “enhanced” interface conditions (namely Robin conditions instead of Dirichlet) compensates the lack of large overlapping.

When \mathcal{A} is the Boltzmann transport operator, the situation is more complicated since \mathcal{A}^Γ also involves the angular variable. However, the idea presented here remains valid.

Note finally that Robin interface conditions are not implemented as such in PARAFISH. This is only a simplified interpretation of the algebraic procedure presented in Section 2.1.

2.3 Non-conforming finite elements

In two dimensions, we now consider the NC_4 non-conforming FE, defined on a rectangle with 4 nodes placed each in the middle of an edge. This element differs from the lowest-order conforming Lagrangian FE, whose 4 nodes are each placed on a vertex. (In the finite element literature, the lowest-order conforming Lagrangian FE is known as “ Q_1 ”, and the NC_4 non-conforming FE as “rotated Q_1 ” element.) Note that in a $N \times N$ Cartesian FE mesh, the number of NC_4 FE nodes is $2N(N + 1)$. Fig. 2 depicts the two-domain configuration when NC_4 elements are used.

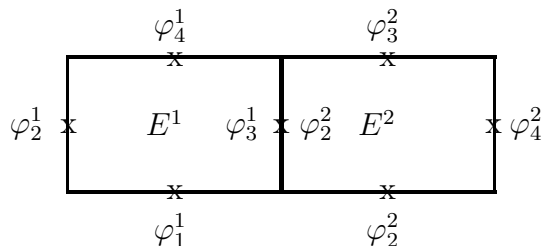


Figure 2: A two-domain example, where each domain is made out of one non-conforming FE. The two domains are coupled by enforcing that φ_3^1 be equal to φ_2^2 .

In three dimensions, we equivalently consider the NC_6 non-conforming FE, defined on a rectangular parallelepiped (cube in the simplest case) with 6 nodes placed each in the middle of a face. In a $N \times N \times N$ Cartesian FE mesh, the number of NC_6 FE nodes is $3N^2(N + 1)$.

Applying the FE method separately in E^1 and E^2 , and following the procedure described in Section 2.1, yields a linear system similar to the one in Eq. (4). Non-conforming FEs appear particularly appropriate for our DD method, because placing the nodes away from the vertices avoids having interface nodes belong to more than 2 elements at a time. In turn, each FE node belongs at most to only two domains. This simplifies the coding of data transfer between domains. Indeed, this must be compared to what happens when conforming FEs are used, that is when certain nodes can belong to as much as 4 (in 2-D) or 8 (in 3-D) domains. Finally, note also that the use of non-conforming FEs has been showed to be valuable in P_N neutronic calculations (Van Criekingen 2007). This last reference also gives the non-conforming FE basis functions.

3 Application to neutron transport

We now consider \mathcal{A} to be the linear Boltzmann transport operator in its second-order even-parity formulation (Lewis and Miller, Jr. 1984). That is, making use of the even- and odd- (angular) parity decomposition reading

$$\Psi^\pm(\mathbf{r}, \boldsymbol{\Omega}) = \frac{1}{2} (\Psi(\mathbf{r}, \boldsymbol{\Omega}) \pm \Psi(\mathbf{r}, -\boldsymbol{\Omega}))$$

the operator \mathcal{A} applies to the even-parity flux $\Psi^+(\mathbf{r}, \boldsymbol{\Omega})$. In the within-group (i.e., energy-independent) case, we have $\mathcal{A} = \mathcal{H} + \mathcal{S}$ with the streaming operator \mathcal{H} defined by

$$\mathcal{H}\Psi^+(\mathbf{r}, \boldsymbol{\Omega}) = -\boldsymbol{\Omega} \cdot \nabla \frac{1}{\sigma(\mathbf{r})} \boldsymbol{\Omega} \cdot \nabla \Psi^+(\mathbf{r}, \boldsymbol{\Omega}) + \sigma(\mathbf{r}) \Psi^+(\mathbf{r}, \boldsymbol{\Omega})$$

where the macroscopic total cross-section $\sigma(\mathbf{r})$ is assumed strictly positive for all \mathbf{r} (this assumption is physically sound) and the scattering operator \mathcal{S} defined by

$$\mathcal{S}\Psi^+(\mathbf{r}, \boldsymbol{\Omega}) = \int_{4\pi} \sigma_s(\mathbf{r}, \boldsymbol{\Omega}, \boldsymbol{\Omega}') \Psi^+(\mathbf{r}, \boldsymbol{\Omega}') d\boldsymbol{\Omega}'$$

with $\sigma_s(\mathbf{r}, \boldsymbol{\Omega}, \boldsymbol{\Omega}')$ the differential scattering cross-section. This formulation has the advantage that \mathcal{A} can be shown to be self-adjoint. We moreover consider criticality calculations, that is, a flux-dependent fission source term $\mathcal{F}\Psi^+$ is introduced, as well as a k eigenvalue, to yield the generalized eigenvalue problem

$$\mathcal{A}\Psi^+ = \frac{1}{k} \mathcal{F}\Psi^+. \quad (10)$$

A detailed description of the weak form and its matrix formulation can be found in Van Criekingen (2007).

The eigenproblem (10) is solved by applying the well-known power iteration method. At each power iteration, a linear system in \mathcal{A} (“inversion” of \mathcal{A}) has to be performed. With the traditional multi-group energy discretization, the discretized \mathcal{A} operator tends to be block-triangular (it is completely block-triangular in the absence of up-scattering). In turn, the Gauss-Seidel method is used to deal with this energy structure. At each Gauss-Seidel iteration, one has to solve a global spatio-angular problem as in Eq. (2). This global solve is done using the above described DD method. As stated in Section 2.1, the DD method consists in iteratively solving a linear system whose matrix has symmetric diagonal blocks of the form given by Eq. (5). This is done with the FGMRES method. Thus, the FGMRES iterations are here the “inner” iterations of the classical outer/inner iteration procedure (the “outer” iterations correspond to the power iterations). When required, the Gauss-Seidel iterations are known as “thermal” iterations.

The FGMRES implementation we use here is the one by CERFACS (Frayssé, Giraud, and Gratton 2006). The FGMRES is block-diagonally preconditioned, and the

local solves, i.e., the “inversions” of the diagonal blocks, are performed in parallel on different processors. Each local solve corresponds to a spatio-angular problem defined on one domain. A tensor-product approach is used to couple space and angle, the angular moments being the innermost blocks.

To perform the local solves, we use the preconditioned conjugate gradient (PCG) iterative solver of the library SPARSELIB (Dongarra, Lumsdaine, Pozo, and Remington 1994) with incomplete Cholesky (IC) factorization. Since the local solves are wrapped-up in the FGMRES iterations, which themselves are wrapped-up in the outer power iterations, these solves do not need to be very tightly converged. A convergence criteria of 10^{-2} on the PCG iterations proved to yield the best results for the Takeda 1 benchmark that we consider below.

The different iteration loops are summarized in Fig. 3.

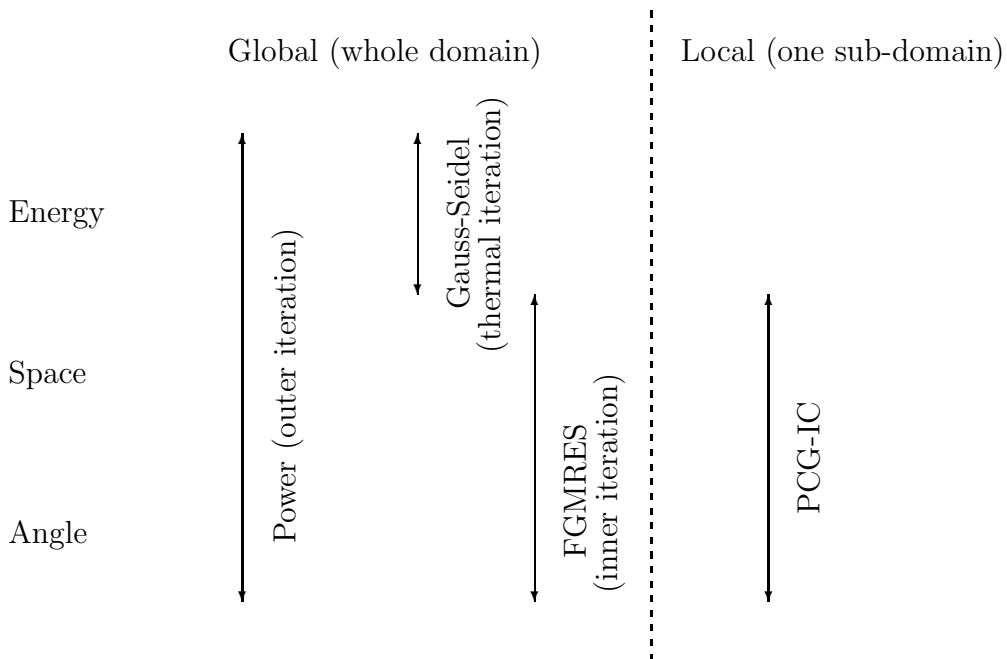


Figure 3: Schematic view of the different iteration loops in PARAFISH

Finally, note that an important parameter for the FGMRES is the restart parameter m (Saad 1993). In fact, this method searches for the solution in a (Krylov) space whose size increases at each iteration. However, the bigger the search space, the more costly the procedure becomes. Therefore, after m iterations, one restarts the procedure from scratch, using the obtained result as initial approximation. Numerical results showed that in 3-D, the m must be chosen bigger than in 2-D to obtain satisfying convergence. In the results presented below, we take $m = 25$ for the 2-D calculations, and $m = 250$ for the 3-D calculations.

3.1 The Takeda 1 benchmark

We here consider the 3-D Takeda 1 benchmark, which consists of a small LWR quarter-core of cubic shape, depicted in Fig. 4. Three zones appear in the geometry: the core zone, the reflector zone and the guide tube zone. The 2-group cross-section values can be found in Takeda et al. (1989). The benchmark has two cases, ‘case 1’ with rods out of the core, and ‘case 2’ with rods in the core. Therefore, in ‘case 1’, the zone 3 is voided (low-density zone), which makes this case more demanding - in fact, it makes the simplified transport method SP_N inappropriate to handle this benchmark.

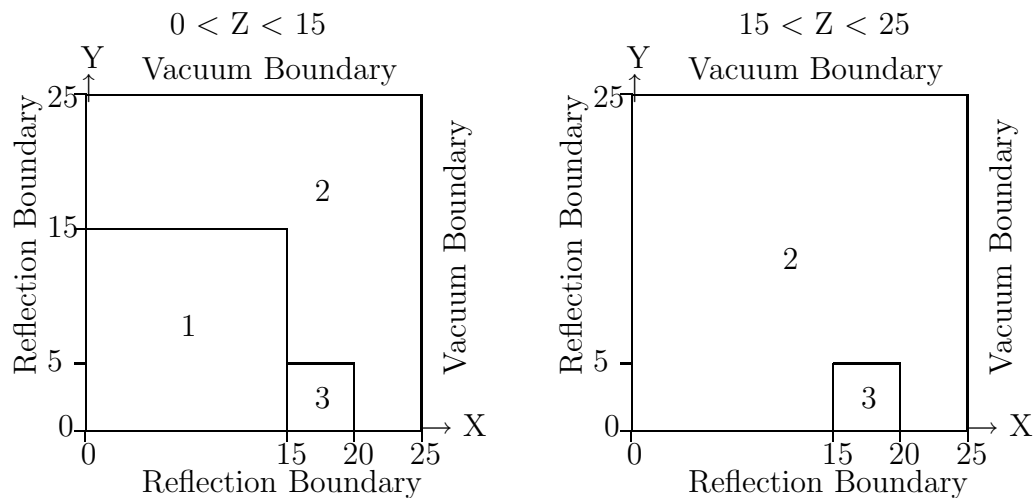


Figure 4: Geometry of the 3-D Takeda 1 benchmark. Zone 1 = core, zone 2 = reflector, zone 3 = guide-tube zone (low-density in case 1, control rod in case 2)

We also consider a scaled-down 2-D version of this benchmark in Section 3.2 to enable calculations on a single processor. This 2-D version is simply one ‘‘Z-layer’’ of the 3-D version taken for $Z < 15$.

For the spatial discretization, as described above we use the square NC_4 and cubic NC_6 non-conforming finite elements in 2- and 3-D, respectively. The capacity of these FEs to accurately solve the Takeda 1 benchmark has been assessed previously (Van Criekingen 2007). For the angular discretization, we use the P_N method, with N ranging for 3 to 7 as indicated.

For all the numerical results presented below, a Cartesian mesh with 50 FE in each direction was used (except for Table 3 where 25 FE are used in each direction). These nodes are distributed among the different domains according to several geometrical domain decompositions. The domain interfaces are always parallel to the cube faces. Note that since no upscattering is present, no Gauss-Seidel (thermal) iterations are needed.

3.2 Sequential numerical results

We here consider the 2-D version of this benchmark, with a P_5 angular flux approximation (thus 9 unknowns per FE node), a 50×50 FE mesh and several geometrical domain decompositions. Trying to achieve a good compromise between domain sizes and physical homogeneity, the 3×3 decomposition is performed using interfaces along $X = 15$ and 20 , and $Y = 5$ and 15 , which yields domains with homogeneous compositions but different sizes. The other decompositions are straightforward, with equally-sized and homogeneous domains. Also, the result for a single domain, i.e. using PCG on the whole quarter-core, is given.

The results in Table 1 clearly show that our DD method is effective as a sequential method: increasing the number of domains up to 25 accelerates the solution process. When increasing the number of domains even more, this is not true anymore. This might be partially due to the increase in the number of (duplicated) unknowns. Moreover, while theoretically one could increase the number of domains up to the number of FE, this would lead to duplicate all the internal FE nodes, which is intuitively clearly inefficient. It is therefore not surprising to find an optimum when varying the number of domains.

Note also that the total inner iteration count goes up with the number of domains, which might be another reason for the limit in the acceleration potential. As for the outer iteration count, it is highly dependent on the stopping criteria chosen (eigenvalue/vector variation or residual value). In this view, no conclusion should be drawn from the outer iteration count.

Table 1: PARAFISH sequential results for the case 2 (rods in) of the 2-D benchmark with P_5 approximation, using different DD and a single CPU: wall-clock time, total number of outer (i.e., power) and inner (i.e., FGMRES) iterations, total number of FE nodes and total number of degrees of freedom for both groups.

Domains	Time (s)	Outers	Inners	FE nodes	DOFs
1	121	20	44	5,100	91,800
9 (3×3)	62	12	147	5,300	95,400
25 (5×5)	41	9	165	5,500	99,000
100 (10×10)	41	8	214	6,000	108,000

3.3 Parallel numerical results

The results for the 3-D benchmark (case 2, P_3 (6 unknowns per FE node), $50 \times 50 \times 50$ FE mesh) are presented in Table 2. This problem could not be run on a single processor. We therefore present results obtained using at least 25 processors. Up to 1000 CPUs were used simultaneously.

Table 2: PARAFISH wall-clock times (in seconds) for the case 2 (rods in) of the 3-D benchmark with P_3 approximation, using different DD and a variable number of CPUs. The efficiency per added CPU compared to the 25 CPU case is given in parenthesis. The times obtained using the maximum number of CPUs are emphasized.

Domains	125 ($5 \times 5 \times 5$)	1,000 ($10 \times 10 \times 10$)	15,625 ($25 \times 25 \times 25$)
FE nodes	412,500	450,000	562,500
Total DOFs (2 groups)	4,950,000	5,400,000	6,750,000
<u>number of CPUs</u>			
25	199	94	734
125	57 (.70)	30 (.63)	650 (.23)
1000	-	92 (.03)	795 (<0.)

The inherent acceleration capability of the DD method, i.e. its ability to accelerate without increasing the number of processors, is also visible here. For instance, with 25 CPUs, using the $10 \times 10 \times 10$ decomposition instead of the $5 \times 5 \times 5$ accelerates the solution process (by a factor close to 2). This acceleration effect does not hold when using $25 \times 25 \times 25$ domains.

For a given decomposition, acceleration can be gained by increasing the number of CPUs. The efficiency per added CPU (compared to the 25 CPU case) is given in parenthesis. Using 125 instead of 25 CPUs is rather efficient. However, the acceleration by CPU increase is limited by the communication time increase. (cf. the dramatically low efficiencies achieved with 1000 processors).

Note that, typically when codes using one domain per processor are applied, one often computes speed-ups comparing the times obtained with as many CPUs as domains. In fact, this means using the maximum number of CPUs allowed by the decomposition. In our case, this would mean comparing the times bold-faced in Table 2. The speed-ups obtained in such a way are however not mathematically correct, since changing the decomposition means changing the linear system to solve (including its size since the number of duplicated FE nodes varies). Computing such “pseudo speed-ups” can in fact yield efficiencies bigger than one. No proper speed-up estimations can actually be done if one CPU can not handle more than one domain.

Finally, we give in Table 3 the complete speed-up results for the two cases of the 3-D benchmark (rods in and out) using a reduced $25 \times 25 \times 25$ FE mesh (which does not alter too much the spatial convergence) decomposed into $5 \times 5 \times 5$ domains. We also vary the P_N approximation, using P_3 , P_5 and P_7 approximations, which respectively means 6, 15 and 25 unknowns per FE node. The obtained efficiencies are quite decent.

Table 3: PARAFISH speed-up estimations for the 3-D benchmark with $25 \times 25 \times 25$ FE mesh, using the $5 \times 5 \times 5$ decomposition and P_3 to P_7 angular approximations.

CASE 1 (rods out)				CASE 2 (rods in)		
number of CPUs	wall-clock time (s)	speed-up	efficiency	wall-clock time (s)	speed-up	efficiency
<i>P₃ approximation</i> (Total DOFs: 675,000)						
1	102	1	1	76	1	1
5	27	3.8	.76	21	3.7	.73
25	9	11.9	.48	6	12.6	.50
125	6	17.3	.14	3	26.3	.21
<i>P₅ approximation</i> (Total DOFs: 1,687,500)						
1	734	1	1	525	1	1
5	237	3.1	.62	182	2.9	.58
25	71	10.3	.41	51	10.3	.41
125	29	25.4	.20	24	22.1	.18
<i>P₇ approximation</i> (Total DOFs: 3,150,000)						
1	2271	1	1	1975	1	1
5	843	2.7	.54	743	2.7	.53
25	255	8.9	.36	216	9.1	.37
125	95	24.0	.19	57	34.9	.28

4 Conclusions and outlook

The encouraging numerical results presented here show the inherent acceleration capability of the proposed DD method: acceleration can be obtained *without* increasing the number of processors. This acceleration potential is limited by the increase in the number of duplicated unknowns, and an optimum can be found. Note that this acceleration is achievable only if one processor can handle more than one domain. Moreover, proper speed-up evaluations can then be obtained, by varying the number of CPUs used with a fixed DD (and thus a fixed number of domains). All this is believed to be a novelty compared to previous investigations in the neutronic field (de Oliveira, Pain, and Goddard 1995; Guérin, Baudron, and Lautard 2007; Pattnaik and de Oliveira 2007).

As future prospect, it is planned to validate the PARAFISH code on the NEA suite of benchmarks for 3-D transport methods and codes over a range in parameter space (Azmy 2007). For the method itself, other types of preconditioning can be investigated for the local solves (improving on the ILU for PCG) as well as at the global level (improving on the block-diagonal for FGMRES).

A proportionality constant α can be introduced in equation (3), and tuned to find an optimal value. This is actually already implemented in PARAFISH, but numerical

tests did not enable us to find a value (or a method to find a value) consistently better than $\alpha = 1$. Some limited improvement could be gained, but always with problem- and discretization- specific α values. Moreover, there is to our knowledge no theoretical result - even in the diffusion case - that could guide our research in this direction.

Note that the ordering in the tensor-product approach used to couple space and angle could be changed (“angle times space” instead of “space times angle”). Also, the quasi-reflected interface conditions developed by Lewis (Lewis 2006) for the spherical harmonic discretization, could prove valuable in the present framework in that it could reduce the exchanges between processors. Finally, a systematic load-balancing strategy (Pattnaik and de Oliveira 2007) should be implemented.

Acknowledgments This work originates from post-doctoral work supported by the French Petroleum Institute. It is currently being pursued at the Karlsruhe Institute of Technology. The numerical results in Section 3.3 were obtained on the JuRoPA (Jülich Research on Petaflop Architectures) supercomputer installed at the Forschungszentrum Jülich (Germany).

The first author acknowledges Luc Giraud from CERFACS for his valuable help concerning the use of FGMRES, as well as the anonymous referee for his constructive remarks.

References

- Azmy, Y. Y. (2007). Benchmarking the accuracy of solution of three-dimensional transport codes and methods over a range in parameter space. NEA/NSC/DOC(2007)1/REV1. OECD, Nuclear Energy Agency. See updates on <http://www.nea.fr/html/science/eg3drtb>.
- de Oliveira, C. R. E., C. C. Pain, and A. J. H. Goddard (April 30 - May 4, 1995). Parallel domain decomposition methods for large-scale finite element transport modelling. In *Proc. Int. Conf. Math. Comp. Reactor Physics and Environmental Analysis of Nuclear System*, Portland, Oregon.
- Dongarra, J., A. Lumsdaine, R. Pozo, and K. Remington (1994). A sparse matrix library in C++ for high performance architectures. pp. 214–218.
- Frayssé, V., L. Giraud, and S. Gratton (2006). A Set of Flexible GMRES Routines for Real and Complex Arithmetics on HPC. Technical report, CERFACS. http://www.cerfacs.fr/algor/reports/2006/TR_PA_06_09.pdf.
- Guérin, P., A.-M. Baudron, and J.-J. Lautard (2007, April 15-19). Domain decomposition methods for core calculations using the MINOS solver. In *Proc. Joint International Topical Meeting on Mathematics & Computation, Supercomputing in Nuclear Applications (M & C + SNA 2007)*, Monterey, CA, U.S.A.
- Lautard, J.-J. (1981, April 27-29). New finite element representation for 3D reactor calculations. In *Proc. of the Topical Meeting on Advances in Mathematical Methods for the Solution of Nuclear Engineering Problems*, München, Germany.

- Lewis, E. E. (2006). Quasi-reflected interface conditions for variational nodal lattice calculations. In *Proceedings of the PHYSOR 2006 ANS Topical Meeting on Reactor Physics*, Vancouver, BC, Canada.
- Lewis, E. E. and W. Miller, Jr. (1984). *Computational methods of neutron transport*. John Wiley & Sons.
- Lions, P.-L. (1990). On the Schwarz alternating method III: a variant for nonoverlapping subdomains. In *Third International Symposium on Domain Decomposition Methods for Partial Differential Equations*, Philadelphia, pp. 202–223. SIAM.
- Nataf, F. (1999, March 20-25). Domain decomposition methods for non-symmetric problems. In *Computational Mechanics using High Performance Computing - Proceedings of the Third Euro-Conference on Parallel and Distributed Computing for Computational Mechanics*, Weimar, Germany, pp. 185–198. Saxe-Coburg Publications.
- Park, H.-K. and C. R. E. de Oliveira (2005). A block diagonal preconditioning strategy for the finite element-spherical harmonics method. *ANS Transactions* 92, 728–730.
- Pattnaik, A. and C. R. E. de Oliveira (2007, April 15-19). A load balancing strategy for the radiation transport code EVENT. In *Proc. Joint International Topical Meeting on Mathematics & Computation, Supercomputing in Nuclear Applications (M & C + SNA 2007)*, Monterey, CA, U.S.A.
- Saad, Y. (1993). A flexible inner-outer preconditioned gmres algorithm. *SIAM J. Sci Comput.* 14(2), 461–469.
- Saad, Y. (2003). *Iterative Methods for Sparse Linear Systems*. SIAM.
- Schwarz, H. (1869). Über einige Abbildungsaufgaben. *Ges. Math. Abh.* 11, 65–83.
- Takeda, T., M. Tamitani, and H. Unesaki (1989). Proposal of 3D neutron transport benchmark. NEACRPA-953 rev.1, Department of nuclear engineering, Osaka University, Japan.
- Van Criekingen, S. (2007). A 2D/3D cartesian geometry non-conforming spherical harmonic neutron transport solver. *Annals of nuclear energy* 34(3), 177–187.