Performance study of a parallel domain decomposition method

N. Odry, J.-F. Vidal, G. Rimpault, J.-J. Lautard, A.-M. Baudron

PERFORMANCE STUDY OF A PARALLEL DOMAIN DECOMPOSITION METHOD

Nans ODRY, Jean-François VIDAL and Gérald RIMPAULT
Commissariat à l'Énergie Atomique,
CEA/DEN/DER/SPRC/LEPh Cadarache, F-13108 Saint Paul-Lez-Durance, France.
nans.odry@cea.fr ; jean-francois.vidal@cea.fr ; gerald.rimpault@cea.fr

Jean-Jacques LAUTARD and Anne-Marie BAUDRON
Commissariat à l'Énergie Atomique,
CEA/DEN/DANS/DM2S/SERMA/LLPR Saclay, F-91191 Gif sur Yvette, France.
jean-jacques.lautard@cea.fr ; anne-marie.baudron@cea.fr

ABSTRACT

This paper studies the influence of various parameters, in order to improve the performances of a *parallel* Domain Decomposition Method (*aka* DDM). If introducing more parallelism represents an opportunity to heighten the performance of deterministic schemes, substantial modifications of their architecture are required. In this context, DDM has been implemented into the APOLLO3®multigroup S_n solver, Minaret. The fundamental idea involves splitting a large boundary value problem into several *independent* subproblems, that can be computed in parallel. Two *DDM* algorithms are considered. The first one solves a one-group problem per subdomain. The second one is a multigroup block-Jacobi algorithm. To improve performances of these DDM, various parallelism strategies are implemented and compared, depending on the internal structure of the DDM algorithm, the technology chosen (MPI or OpenMP), and the variable parallelized (angular direction or subdomain). Based on these considerations, an efficient *hybrid* parallelism, suitable for *HPC* is built: a parallel multigroup Jacobi iteration algorithm, using a two layer MPI/OpenMP architecture, gives the best performances for the reactor configuration studied.

Key Words: **domain decomposition, parallel computing, APOLLO3®**

1. INTRODUCTION

1.1. Elements of context

A new neutronics platform, APOLLO3® [1], is currently under development at the French Atomic Energy Commision *CEA*. It provides an integrated tool, able to deal with a large range of reactor concepts, both for core and lattice calculations. APOLLO3® benefits from substantial progresses in computer science, from the modern software conception (object oriented programming langages...) to the hardware evolution. In this context, massively parallel computing represents an opportunity to improve performances. So much that introducing more parallelism into deterministic schemes is nowadays a major trend in the new simulation codes.

Various parallelization strategies are found in litterature. Three levers can be distinguished, matching the variables of the transport equation: angle, space and energy. These strategies are compatible and can be combined, as in the mini-app *Kripke* [2] or the 3D S_n solver *Partisn* [3].

- it is very natural to parallelize on angular directions, particularly when S_n methods are used.
- it is more delicate to break the multigroup approximation, in order to deal simultaneously with various energy groups. For instance, [3] proposes a 'point Jacobi' algorithm.
- considering the local dependency of the spatial unknowns, parallelism on space is very promising. The sweeping of the spatial mesh is parallelized through a 'KBA[4] algorithm' in [3, 5, 6].

Another complementary strategy is based on domain decomposition. This paper deals with the performances of such a parallel *Domain Decomposition Method* (*aka* DDM).

1.2. General principle of domain decomposition

Nowadays, domain decomposition arouses a large interest since it efficiently handles large numerical problems on parallel computers. The fundamental idea involves splitting a large boundary value problem into several but smaller sub-problems.The DDM iterates between local resolution on each subdomain (using local boundary conditions) and a flux exchange step between subdomains. Since the sub-problems are only connected through their boundary conditions, each of them can be solved independently.

DDM is a well-tried methodology and a substantial litterature can be found on the subject. For instance, works presented in [7–11] developed a non-overlapping parallel block-Jacobi algorithm, completed by a diffusion-based acceleration of the CMFD-kind, for various solvers (Newtrnx (S_n), Denovo (S_n), Mpact (MOC), IDT (S_n)...). In the same scope, non-overlapping domain decompositions have been implemented in the SP_n solver Minos [12, 13], and in the P_n solver Parafish.

1.3. Context of the present work

The DDM we study in this paper has been implemented inside the Apollo3® solver *Minaret* [14]. *Minaret* is a 2D/3D core solver, built on the discrete ordinate method S_n and relying on a Discontinuous Galerkin Finite Element Method. The mesh is cylindrical, with an unstructured radial basis. *Minaret* is then able to deal with a large variety of core geometries. A previous proceeding [15] presented the equations, the implementation and the numerical verification of the DDM inside the solver.

The aim of the present paper is to show how performances of the DDM can be improved. Two DDM have been implemented. They are presented in a first section. Then, various parallelism strategies, applied to both domain decomposition and reference algorithms, are considered. The aim is to build an efficient *hybrid* parallelism, suitable for *HPC*, by mixing MPI and OpenMP standards. A performance study is included. The test cases presented in this proceeding are all of the hexagonal periodicity kind.

2. IMPLEMENTATION OF TWO DDM ALGORITHMS

Two different DDM algorithms have been implemented. They differ by the way the Jacobi iterations are integrated inside the standard inverse power iteration scheme. Equations are not presented here, but can be found in [15].

2.1. One-group Jacobi iterations

The first DDM algorithm is based on one-group Jacobi iterations to solve the k-eigenvalue problem. The Jacobi iterations are applied to monoenergetic problems, waiting the full convergence of the within-group scattering source for each energy block to tackle the next one. This algorithm is refered as 1g-DDM.

2.2. Multigroup Jacobi iterations

To obtain the second algorithm, we modify the 1g-DDM in order to deal with multigroup problems per subdomain. To do so, the multigroup loop is switched with the subdomain one, as shown in 2. The scattering source is then locally and independently estimated on each subdomain. It is of the utmost importance when parallelism is introduced. This new algorithm is refered as mg-DDM.

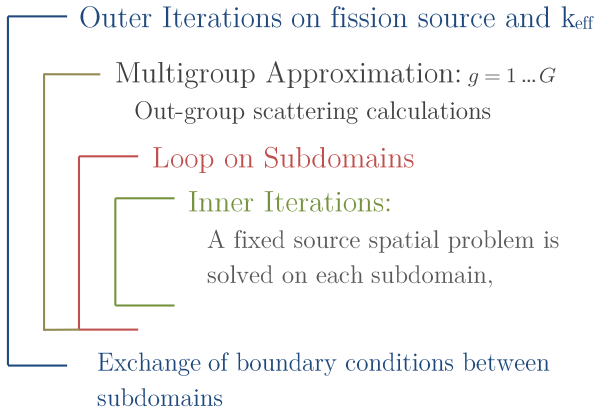

Figure 1: Domain Decomposition algorithm using one-group Jacobi iterations

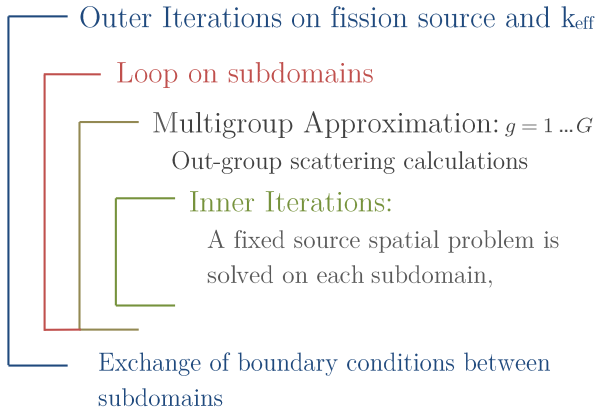

Figure 2: Domain Decomposition algorithm using multigroup Jacobi iterations

In both cases, the boundary flux exchange is performed once each subdomain and each energy group is converged. Doing so, the algorithm is well-suited for 'full' parallelism (*Gauss-Jacobi* algorithm).

A numerical verification shows how perfectly consistent these two DDM algorithms are, particularly when compared with the reference scheme. Such a numerical verification has already been tackled in [15]. Performances of 1g-DDM and mg-DDM algorithms are compared in section 3.

3. VARIOUS LEVELS OF PARALLELISM

In order to benefit from the domain decomposition, the new algorithm needs to be parallelized. Indeed, using a sequential calculation, the DDM *exactly* converges to the standard *Gauss-Seidel* algorithm, but both the number of outer iterations and the computing time increase [15]. This rise needs to be counterbalanced thanks to parallelism.

Pros of parallel computing are twofold. It helps to substantially decrease computing time. Moreover, parallelizing also enables to divide the storage requirement between several memory resourses. Doing so, one can hope to deal with much bigger problems than nowadays (reactor core calculation with a fine spatial and energy mesh for instance).

3.1. Various parallelism strategies

The aim of this section is to discriminate between several parallelism strategies, by selecting the most efficient depending on the context. These strategies are built by choosing between two programming technologies, two parallelized variables and two DDM algorithms:

1g-DDM *vs* mg-DDM When a spatial parallelism is chosen, one can use either the 1g-DDM or the mg-DDM algorithms previously introduced.

Spatial parallelism *vs* Angular parallelism As mention earlier, different levels of parallelism can be chosen: energy, space and angular direction. In this paper, we only implemented an angular and a DDM-based spatial parallelism:

- The resort to spatial parallelism is very natural using DDM, since once boundary conditions are set, each local sub-problem can be solved independently. The integration of more parallelism has been foreseen in the DDM developments.
- On the other hand, one can easily benefits from the independance property of angular directions used in the discrete ordinate method S_n. This kind of parallelism does not require a domain decomposition algorithm.

Two parallel programming standards Usually, two programming standards are considered for parallel computing, depending on both the computer architecture and the way memory is managed. A detailled review of these parallel computing standards can be found in [16]. In this paper, we focus on a shared-memory parallelism, using the OpenMP protocole, and on a distributed-memory parallelism, built on the MPI protocole.

- On the one hand, the application programming interface OpenMP is used for shared-memory machines: the program considers the memory as a unique and shared resource, accessible simultaneously by all the independant processes, known as *threads*. OpenMP is easy to implement and does not require a deep restructuration of the algorithm. Nevertheless, an attention is required to avoid conflicts in the way shared data are used and modified. To do so, temporary duplication of some local data is sometimes needed.
Moreover, OpenMP suffers an inherent constraint. The number of available independent processes is limited by a hardware parameter, the number of core available. Indeed, since OpenMP uses a

shared memory, parallelism is limited to the calculation units pluged on it. For instance, standard microcomputers rarely exceed 24 cores, which limits the parallelism acceleration to a factor 24.

• On the opposite, the Message-Passing Interface parallelism (MPI) considers memory as a set of 'private' separated areas, each of them being reachable by a unique process. Doing so, MPI creates local information for each process, which communicates with its counterparts by exchanging data. The parallelism efficiency is often limited by the slowness of this data exchange.
On the other hand, sharing out the global data on a set of independent local memories is a great opportunity for the DDM. Distributing enough memory resources (on High Performance Computer *HPC*), memory requirements are no longer a limiting factor. The problem size can significantly increase.
At last, it is up to the developer to define which information is sent or received, and the moment at which it occurs. MPI is then much more delicate to implement than OpenMP.

As a conclusion, OpenMP represents an efficient and easy way to introduce a 'small magnitude' parallelism on shared-memory computers. On the contrary, MPI is particularly well-suited for distributed-memory multiprocessors, such as *HPC*, coupled to 'large magnitude' parallelism.

3.2. Performance study of 'elementary' parallelism strategies

What we call 'elementary' parallelism configurations are considered: strategies are built by combinations of the options presented above. However, mixing MPI and OpenMP or mixing parallelism on subdomain and parallelism on angular direction is temporarily forbidden.

3.2.1. Performances

The results presented are obtained from an academic test case, made of two rings of fast reactor hexagonal fuel assembly (7 assemblies) in 2D. The choice is made to match assembly and subdomain. A S_2 approximation is chosen, creating 6 independent angular directions. Doing so, up to 7 (respectively 6) independent processes can be computed simultaneously on space (respectively angular direction). Such a small test case has been chosen since it is the only configuration which enables to *fully* parallelize the problem with limited quantity of computing resources. More ambitious test cases will be presented in the next section. Calculations are performed on a 16 cores computer with 64GB of RAM memory.

In order to quantify and compare the performances of each 'elementary' parallelism, the *efficiency* criterium is introduced. Considering a parallel computation on p processors, efficiency is defined as:

$$E = \frac{t_1/t_p}{p}$$

where t_1 the sequential computating time and t_p the parallel computing time using p processing units. Sequential and parallel computation are performed using the same algorithm (standard, 1g-DDM or mg-DDM). E is a value between 0 and 1, estimating how well-used the processors are in solving the problem, compared to how much effort is wasted in communication and synchronization. The best performances are, the closer to 1 the efficiency will be (case of linear speed up).

Table I: Comparison of computing time for various basic parallelisation strategies
16 cores available - 7 subdomains - 6 angular directions

Algorithm		**number of procs MPI**		**number of threads OpenMP**		**computing time**	***E***
		Subdo.	*Ang. Dir.*	*Subdo.*	*Ang. Dir.*		
a/	**standard**	1	1	1	1	420 s	1
b/	**standard**	1	1	1	6	104 s	0.58
c/	**standard**	1	6	1	1	122 s	0.57
d/	**1g-DDM**	1	1	1	1	1619 s	1
e/	**1g-DDM**	1	1	1	6	534 s	0.51
f/	**1g-DDM**	1	1	7	1	289 s	0.80
g/	**1g-DDM**	7	1	1	1	852 s	0.27
h/	**mg-DDM**	1	1	1	1	1561 s	1
i/	**mg-DDM**	1	1	1	6	469 s	0.55
j/	**mg-DDM**	1	1	7	1	263 s	0.85
k/	**mg-DDM**	7	1	1	1	367 s	0.61

• **reference algorithm**: The configuration (a/) is the reference sequential calculation, obtained with the standard *Gauss-Seidel* algorithm. The aim of parallelism is to improve this computing time. To do so, parallelizing on angular directions does not require a domain decomposition. Both OpenMP (b/) and MPI (c/) have been tested, leading to a significant improvement of computing time, but without any significative difference between the two technologies for such a 'size-limited' test case.

• **sequential - DDM algorithms** : Let us compare the sequential performances of the two DDM algorithms (d/, h/) and the reference scheme (a/). The computing time increases by a factor 3.7/3.8 using domain decomposition. It is a fundamental property of domain decomposition, due to the switch from a *Gauss-Seidel* to a *Gauss-Jacobi* algorithm: information is spread from one subdomain to its neighbor at each outer iteration. On the contrary, an optimized *full-core* mesh sweeping is implemented in the Gauss-Seidel standard algorithm, describing the entire core in a single outer iteration. The number of outer iterations needed is then much smaller. Moreover, sequentially, there is no interest to favor 1g-DDM or mg-DDM, since performances are quite comparable.

• **DDM - MPI on subdomains**: It is no longer the case considering a MPI parallelism on subdomains. Doing so, performances of the mg-DDM algorithm (k/) are nearly twice more efficient than using 1g-DDM(g/), both in term of computing time and efficiency. Indeed, the multigroup algorithm enables to locally build the scattering for each subdomain. First, this construction step can be parallelized. Secondly, the amount of communications needed is reduced, compared to 1g-DDM, where a restricted source map is sent to each processing unit at every outer iteration.

• **DDM - OpenMP on angular directions**: The efficiency of an OpenMP parallelism on angular directions is very similar using the reference algorithm(b/) or the domain decomposition algorithms (e/, i/). So, it necessarily fails to catch up the additional cost of domain decomposition due to the *Gauss-Jacobi* iterations.

• **DDM - OpenMP on subdomains**: Last but not least, OpenMP parallelism on subdomains

reaches by far the best efficiency values(f/, j/). Parallelism using OpenMP is then greatly recommanded, particularly when knowing that this efficiency is quite preserved while the problem size rises ($E = 0.71$ for a S_4 test case, made of 37 subdomains and using 9 threads).

3.2.2. Influence of the number of processing units

A parametric study has been performed for various configurations, plotting the influence of the number of processing units on computing time, for the small test case. Comparison of the various parallelism approaches reveals similar trends than previously. One phenomenon is notable: a stage appears on the plot around 3, 4 and 5 processing units used. When parallelizing on 6 angular directions (respectively 7 subdomains), two steps are needed to describe them all, no matter if 3, 4 or 5 processors are available.

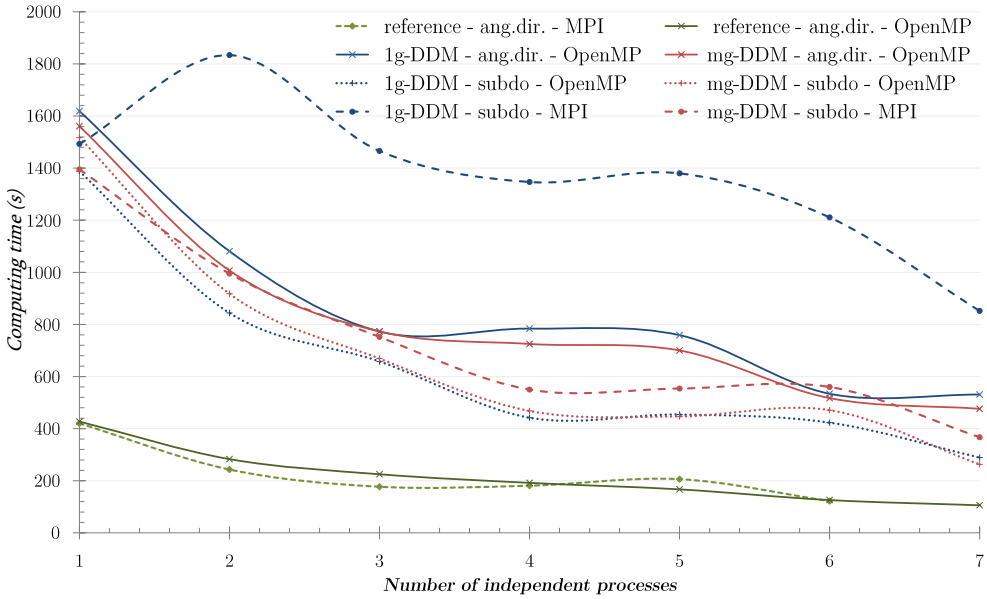

Figure 3: Influence of various parameters on computing time

Increasing the number of processing available units is then quite useless if the number of processing 'waves' remains unchanged. The calculation time is preserved while the efficiency strongly decreases. This phenomenon is particularly important when the number of independent variables rises. For instance, using 9 or 16 processors both needs two 'waves' to process 18 angular directions. A judicious load balance of variables on processors is then hightly recommended.

In order to maximize the efficiency of the domain decomposition algorithm, and then have an optimal management of computing resources, the choice of the 'mg-DDM' is obvious, particularly when parallelized on subdomains. Performances are very satisfying using an OpenMP parallelism. Yet, the number of threads allowed is limited by the computer architecture. So, since MPI performances on subdomains are quite good too, it would be of the utmost interest to couple the two parallel programming standards.

3.3. Hybrid parallelism

To benefit from the advantages of both MPI and OpenMP, the two strategies are often coupled in a two-layer parallelism. It is particularly well-suited on *HPC* which uses simultaneously a shared-memory inside each node, and a distributed memory among them. A two layer 'hybrid' parallelism has then been developed in *Minaret*. The first layer splits data on independent nodes using MPI, thanks to the independence property of subdomains. Over a second level, an OpenMP computation is performed, either on subdomains or on angular directions. Doing so, the number of independent processes is potentially unlimited.

Table II: Comparison of computing time for various basic parallelisation strategies
16 cores available - 37 subdomains - 18 angular directions

Algorithm	**number of procs MPI**		**number of threads OpenMP per MPI**		**total number of processing units**	**computing time**	***E***
	Subdo.	*Ang. Dir.*	*Subdo.*	*Ang. Dir.*			
a/ **reference**	1	1	1	1	1	10675 s	1
b/ **reference**	1	1	1	16	16	2499 s	0.26
c/ **reference**	1	16	1	1	16	2104 s	0.32
d/ **mg-DDM**	1	1	1	1	1	34376 s	1
e/ **mg-DDM**	3	1	1	9	27	3921 s	0.32
f/ **mg-DDM**	1	1	13	1	13	4081 s	0.64
g/ **mg-DDM**	3	1	13	1	39	2339 s	0.37

The presented test case is made of 37 subdomains and 18 S_n angular directions in $2D$ (675000 meshes). It is computed on an *HPC*, made of a set of computing nodes, each of them having 16 cores and 64GB of RAM.

Performances are quite encouraging, since the sequential reference calculation time (a/) is largely improved using 'hybrid' parallelism on either subdomains (g/) or angular directions (e/). Nevertheless, a similar computing time to (g/) can be found using the parallel reference scheme, with significativly less processing units (b/, c/). Efficiency is better using Domain Decomposition, yet the method suffers from both the additional number of outer iterations and the cost of MPI communications between nodes.

Actually, the test case is not large enough to show the real interest of domain decomposition. However, it is essential to notice there is no more room for improvement in the reference calculation, particularly when the problem size rises. The amount of independent processes is limited by the number of angular directions.Conversely, the number of subdomains can be increase as much as wanted. Even better, the larger the problem to solve is, and the most valuable the domain decomposition will be.

4. CONCLUSION

This paper studies the influence of various parameters on the performances of a *parallel* Domain Decomposition Method. The aim is to provide an efficient solver, able to efficiently perform on *HPC*. Developments have been performed inside the S_n *Minaret* solver, from the APOLLO3® neutronics platform.

A parallel one-group block Jacobi algorithm and a parallel multigroup block Jacobi algorithm have been efficiently implemented in Minaret. Various parallelism strategies have been tested for the reference and the two domain decomposition algorithms, depending on the variable parallelized (angular direction or subdomain) and the programming standard used (MPI or OpenMP). A performance study is given, showing how important it is to reduce the amount of MPI communications. On the other hand, OpenMP efficiency is very satisfying. At last, it is of the utter interest to couple MPI and OpenMP in a two layers '*hybrid*' parallelism, suitable for *HPC*. Doing so, the number of independent processes is potentially unlimited. Especially, the larger the problem to solve is, the most valuable this 'hybrid' computation will be.

These first results are very encouraging for future developments. Nethertheless, there is still room for improvement. One can notice that the performances are still restrained by the extra number of outer iterations required to converge the DDM. A diffusion-based acceleration is currently under development to overcome this issue.

ACKNOWLEDGMENTS

The authors would like to acknowledge AREVA and Électricité de France (EDF) for providing partial financial support. The first author also wishes to thank the CEA/DEN/DANS/DM2S/SERMA and CEA/DEN/DER/SPRC teams for their technical support.

REFERENCES

[1] D. Schneider *et al.* "APOLLO3®: CEA/DEN deterministic multi-purpose code for reactor physics analysis." In: *this PHYSOR 2016 conference*. Sun Valley, Idaho, USA (2016).

[2] A. Kunen, T. Bailey, and P. N. Brown. "Kripke: A massively parallel transport mini-app." In: *Proc. Int. Conf. on Math., Computational Meth., M&C2015*. Nashville (TN, USA) (2015).

[3] R. Baker. "An Sn algorithm for modern architectures." In: *Proc. Int. Conf. on Math., Computational Meth., M&C2015*. Nashville (TN, USA) (2015).

[4] K. Koch, R. Baker, and R. Alcouffe. "A parallel algorithm for 3D Sn transport sweep." *LA-CP-92-406* (1992).

[5] S. Pautz and T. Bailey. "Parallel deterministic transport sweeps of structured and unstructured meshes with overloaded mesh decompositions." In: *Proc. Int. Conf. on Math., Computational Meth., M&C2015*. Nashville (TN, USA) (2015).

[6] M. Adams *et al.* "Provably optimal parallel transport sweeps with non-contiguous partitions." In: *Proc. Int. Conf. on Math., Computational Meth., M&C2015*. Nashville (TN, USA) (2015).

[7] K. T. Clarno. "Implementation of generalized coarse-mesh rebalance in NEWTRNX for acceleration of parallel block-jacobi transport." *Trans. Am. Nucl. Soc.*, **97**: p. 498 (2007).

[8] B. Kelley and E. Larsen. "CMFD acceleration of spatial domain-decomposed neutron transport problems." In: *Proc. Int. Conf. on Physics of Reactors (PHYSOR2014)*. Knoxville(USA) (2012).

[9] B. Kochunas *et al.* "Application of the SDD-CMFD acceleration method to parallel 3D MOC transport." In: *Proc. Int. Conf. on Physics of Reactors (PHYSOR2014)*. Kyoto (Japan) (2014).

[10] S. Yuk and N. Z. Cho. "p-CMFD acceleration and non-overlapping local/global iterative transports methods with 2D/1D fusion kernel." In: *Proc. Int. Conf. on Physics of Reactors (PHYSOR2014)*. Kyoto (Japan) (2014).

[11] R. Lenain *et al.* "Domain decomposition method for 2D and 3D transport calculations using hybrid MPI/OPENMP parallelism." In: *Proc. Int. Conf. on Math., Computational Meth., M&C2015*. Nashville (TN, USA) (2015).

[12] P. Guérin, A.-M. Baudron, 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).

[13] E. Jamelot and P. Ciarlet. "Fast non-overlapping Schwarz domain decomposition methods for solving the neutron diffusion equation." *Journal of Computational Physics*, **241**: pp. 445–463 (2013).

[14] J.-Y. Moller and J.-J. Lautard. "Minaret, a deterministic neutron transport solver for nuclear core calculations." In: *Proc. Int. Conf. on Math., Computational Meth., M&C2011*. Rio de Janiero (Brasil) (2011).

[15] N. Odry *et al.* "A Domain Decomposition Method in the APOLLO3 solver, Minaret." In: *Proc. Int. Conf. on Math., Computational Meth., M&C2015*. Nashville (TN, USA) (2015).

[16] D. Griesheimer *et al.* "Strategies and algorithms for hybrid-shared memory/message passing in Monte-Carlo radiation transport code." In: *Proc. Int. Conf. on Math., Computational Meth., M&C2015*. Nashville (TN, USA) (2015).