Efficient Parallel Solution of the 3D Stationary Boltzmann Transport Equation for Diffusive Problems

Salli Moustafa, François Févotte, Mathieu Faverge, Laurent Plagne, Pierre Ramet

Efficient Parallel Solution of the 3D Stationary Boltzmann Transport Equation for Diffusive Problems

Salli Moustafa[a,b], François Févotte[a], Mathieu Faverge[b,c], Laurent Plagne[a], Pierre Ramet[b,d]

[a] *EDF Lab Paris-Saclay - 7, Boulevard Gaspard Monge 91120 Palaiseau France*
[b] *INRIA Bordeaux - Sud-Ouest, LaBRI, Talence, France*
[c] *Bordeaux INP, Talence, France*
[d] *University of Bordeaux, Talence, France*

Abstract

This paper presents an efficient parallel method for the deterministic solution of the 3D stationary Boltzmann transport equation applied to diffusive problems such as nuclear core criticality computations. Based on standard MultiGroup-Sn-DD discretization schemes, our approach combines a highly efficient nested parallelization strategy [1] with the PDSA parallel acceleration technique [2] applied for the first time to 3D transport problems. These two key ingredients enable us to solve extremely large neutronic problems involving up to 10^{12} degrees of freedom in less than an hour using 64 super-computer nodes.

1. Introduction

This paper presents an efficient parallel deterministic solution of the stationary Boltzmann Transport Equation (BTE) applied to 3D diffusive problems.

1.1. Deterministic 3D stationary Boltzmann transport equation solver

The BTE governs the statistical evolution of gas-like collections of neutral particles described by phase-space densities $f(\vec{r}, \vec{p}, t)$ proportional to the number of particles at a position $\vec{r}$, with momentum $\vec{p}$ at a given time t. This one-body description is widely used to simulate the transport of particles like neutrons or photons through inhomogeneous reactive media. The material properties of the media are characterized by cross-sections that measure the probability of various particle interactions: absorption, scattering, emission, etc.

Lying in a six-dimensional (6D) space (3 for space, 2 for direction and 1 for energy), a precise mesh-based discretization of the stationary BTE solutions $f(\vec{r}, \vec{p})$ can be very large for *true* 3D cases where the considered physical problem offers no particular spatial symmetry. As an example, a best-estimate simulation for Pressurized Water Reactor (PWR) physics might feature in the order of 300 cells for angular discretization, 300 for energy, and 10^7 for space. This leads to an order of 10^{12} phase-space cells, which may each contain several

Degrees Of Freedom (DoFs). Considering the scale of a BTE solution, one can easily infer that its solving procedure may rapidly exhaust the capability of the largest supercomputers. As a consequence, probabilistic methods (Monte-Carlo) that avoid the phase-space mesh problems, were the only approaches able to deal with true 3D cases until the beginning of this century. Unfortunately, probabilistic methods converge slowly with the number N of pseudo-particles ($\propto N^{-1/2}$) and the computational demand increases strongly with the desired accuracy. During the last two decades the peak performance of super-computers has been multiplied by a factor of 10^4. Modern supercomputer capabilities have made deterministic methods a credible alternative to probabilistic methods for 3D problems and has allowed for unprecedented accuracy levels for BTE approximate solutions.

1.2. Reference criticality computations for nuclear diffusive problems

BTE solvers are used in different physical contexts and optimal numerical methods differ from one application to another. In this paper we address the specific problem of solving the stationary BTE in diffusive media. Diffusive problems arise when the mean-free path of particles becomes small compared to the characteristic scale of the considered problem. For such media, and considering an optically thick enough geometry, one may neglect the advective part of the transport and replace the original BTE by the much simpler diffusion equation.

This work takes place in the context of nuclear reactor simulations. We consider the transport of neutrons inside nuclear reactor cores which contain optically thick diffusive media. More specifically, we address the problem of nuclear core criticality computations. Because nuclear cross-sections mainly depend on the particle energy, the phase-space density variable $f(\vec{r},\vec{p})$ is replaced by the angular neutron flux $\psi(\vec{r},E,\vec{\Omega}) = vf(\vec{r},\vec{p})$ where $\vec{\Omega}$ stands for the particle momentum direction, v its velocity and E its kinetic energy. Nuclear operators need to complete many criticality computations that correspond to stationary BTE solutions. Industrial routine computations, which are primarily used to conduct operational and safety studies and to optimize nuclear reactor core designs, are often based on the diffusion equation approximation. In order to assess this approximation, the solution of the original BTE problem is required. More generally, nuclear operators need accurate reference transport solutions in order to control the accuracy of their simulations.

1.3. Starting from a classical numerical scheme

The proposed method is based on nested algorithms classically used for nuclear criticality computations. The external loop is a Chebyshev-accelerated Power Iteration (PI) algorithm that solves the eigenvalue problem $H\psi = k^{-1}F\psi$ where H is the transport operator and F the fission operator. The kinetic energy of neutrons is discretized in well chosen slices called energy groups and, for each power iteration, an iterative Gauss-Seidel (GS) algorithm is used to solve the multigroup linear problem. For each energy group, the angular variable is

treated with the discrete ordinates method (S_N) and a Source Iteration (SI) algorithm deals with the coupling between angular components of the flux. For diffusive problems the SI procedure converges slowly and is classically accelerated by the Diffusion Synthetic Acceleration method (DSA) [3]. In this paper, we introduce a parallel extension of the DSA method (PDSA) where an efficient single-domain diffusion solver is required. Finally, the space is discretized over 3D Cartesian meshes, and all the examples of the paper use the lowest order Diamond Differencing spatial discretization scheme (DD_0), which appears to be sufficient for diffusive nuclear core simulations [4]. Note that the PDSA method does not depend on the DD_0 choice and a higher order numerical scheme could have been used. The only condition is that these alternative schemes must be consistent with the single-domain DSA solver.

1.4. New metrics for efficient numerical algorithms

The tremendous peak power of modern supercomputers, that commonly exceeds 10^{16} floating point operations per second (flop/s), is accompanied by a high architecture complexity. Indeed, recent architectures exhibit a hierarchical organization (cluster of nodes of multicore processors with vector units) which requires a mix of different parallel programming paradigms (Message Passing, Multi-Threading, SIMD) to achieve optimal efficiency.

In addition to this mixture of parallel programming models, constraints on the data movements play a large role in computational efficiency. A direct consequence of this machine evolution is that numerical algorithms should not be evaluated on their *parallel scalability* (*i*) alone.

The *arithmetic intensity* (*ii*) which measures the ratio between the number of floating point (FP) operations and the number of data movements from the off-chip memory to CPU registers is a new metric that must be considered to evaluate the efficiency of a given algorithm. Finally, the *vectorization potential* (*iii*) of a given algorithm will determine its ability to benefit from the ever increasing Single Instruction Multiple Data (SIMD) width of dedicated CPU FP SIMD instructions (SSE2, AVX, AVX512). The combination of these three algorithm characteristics (*i*, *ii*, *iii*) will eventually result in an efficient numerical solver. In this paper we put a particular focus on the efficiency of the proposed BTE solver and explain how *parallel scalability*, *arithmetic intensity* and *vectorization potential* are taken into account.

1.5. Paper contributions

In this paper we propose a parallel and efficient solution method for the stationary BTE that allows one to carry out very large criticality computations for diffusive problems on moderately large supercomputers. These *affordable* full 3D transport computations result from an uncommonly high effective FP performance that can exceed 20% of the available theoretical peak performance of the computing nodes. As a representative example, we show that a 3D PWR k_{eff} computation with 26 energy groups, 288 angular directions, and $578 \times 578 \times 140$ space cells, can be completed in less than an hour using 64 cluster

nodes. Two main ingredients are combined in the proposed method that has been implemented in DOMINO [5, 6], our in-house neutron transport solver.

- A very high performance **sweep** algorithm including 3 nested levels of parallelism with good data locality and fine grained synchronization that has been described in detail in [1].
- A novel scalable PDSA acceleration technique for diffusive problem introduced in [2] and applied for the first time to 3D transport computations. This method is easy to implement provided a fast single-domain shared-memory diffusion solver. Hence PDSA allows one to avoid the complex task of building fully distributed diffusion solvers as implemented in [7, 8].

Recently, important progress has been made for increasing the scalability of BTE solvers. In [9] the authors replace the Power Iterations by advanced eigenvalue algorithms and treat the energy groups in parallel. The scalability of this approach is impressive and parallel computations involving more than 10^5 computing cores are presented. In this current paper we show that, for a moderately high number of energy groups[1] (≤ 26), the proposed method results in fast criticality computations with more modest numbers of computing cores ($10^2 - 10^3$) thereby making 3D stationary computation more affordable.

This result should have an impact on the acceleration strategies for other kinds of BTE solvers like unstructured mesh based transport solvers or the accelerated Monte-Carlo approach for criticality nuclear computations.

The paper is organized as follows. In Section 2, we describe the equations to be solved, the different discretization schemes, the main algorithm and the three nested levels of parallelism used in the sweep implementation described in [1]. In Section 3, the PDSA algorithm and its implementation are described and some details are given regarding the correct coupling between the Transport DD_0 discretization and the Finite Element method used in the PDSA. Section 4 describes the parallel performance achieved by DOMINO for different PWR criticality computation configurations. Some conclusions and outlooks are given in section 5.

2. The Discrete Ordinates Method for Neutron Transport Simulation

2.1. Source Iterations Scheme

We consider the monogroup transport equation as defined in equation (1).

$$\underbrace{\boldsymbol{\Omega}\cdot\nabla\psi(\mathbf{r},\boldsymbol{\Omega})+\Sigma_t(\mathbf{r},\boldsymbol{\Omega})\psi(\mathbf{r},\boldsymbol{\Omega})}_{L\psi(\mathbf{r},\boldsymbol{\Omega})}-\overbrace{\int_{S_2}d\boldsymbol{\Omega}'\Sigma_s(\mathbf{r},\boldsymbol{\Omega}'\cdot\boldsymbol{\Omega})\psi(\mathbf{r},\boldsymbol{\Omega}')}^{R(\mathbf{r},\boldsymbol{\Omega})}=Q(\mathbf{r},\boldsymbol{\Omega}),\quad(1)$$

[1] with respect to the $\simeq 300$ energy groups typically needed to represent fine spectral effects in 2D assembly calculations for PWRs

Input : ϕ^k
Output: $\phi^{k+\frac{1}{2}}$
while *Non convergence* **do**
  for $i = 1, \ldots, N_{dir}$ **do**
    $S(\mathbf{r}, \mathbf{\Omega}_i) = Q(\mathbf{r}, \mathbf{\Omega}_i) + \Sigma_s(\mathbf{r})\phi^k(\mathbf{r})$;
4   $L\psi^{k+\frac{1}{2}}(\mathbf{r}, \mathbf{\Omega}_i) = S(\mathbf{r}, \mathbf{\Omega}_i)$;
    $\phi^{k+\frac{1}{2}}(\mathbf{r}) = \sum_{j=1}^{N_{\text{dir}}} w_j \psi^{k+\frac{1}{2}}(\mathbf{r}, \mathbf{\Omega}_j)$;

Algorithm 1: Source iterations

where $Q(\mathbf{r}, \mathbf{\Omega})$ gathers both monogroup fission and inter-group scattering sources. The angular dependency of this equation is resolved by looking for solutions on a discrete set of carefully selected angular directions $\{\mathbf{\Omega}_i \in S_2, i = 1, 2, \cdots, N_{\text{dir}}\}$, called discrete ordinates; each one being associated to a weight w_j. In general, the discrete ordinates are determined thanks to a numerical quadrature formula as defined in equation (2). For any summable function g over S_2:

$$\bar{g} \equiv \int_{S_2} g(\mathbf{\Omega}) d\mathbf{\Omega} \simeq \sum_{j=1}^{N_{\text{dir}}} w_j g(\mathbf{\Omega}_j). \tag{2}$$

In DOMINO, we use the *Level Symmetric* [10] quadrature formula, which leads to $N_{\text{dir}} = N(N+2)$ angular directions, where N stands for the *Level Symmetric* quadrature formula order.

Therefore, considering the cross-sections to be isotropic, equation (1) becomes:

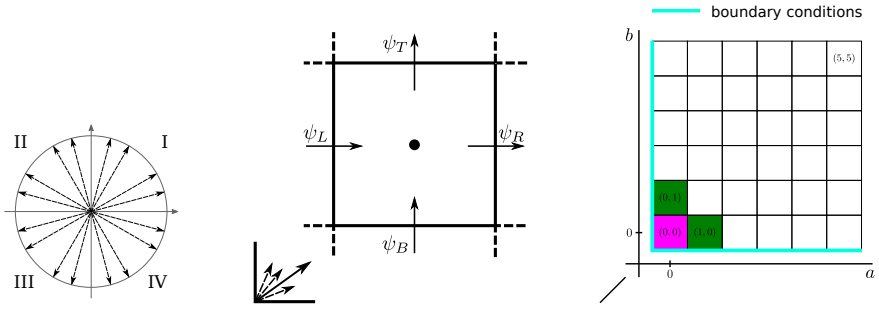
$$\mathbf{\Omega}_i \cdot \nabla\psi(\mathbf{r}, \mathbf{\Omega}_i) + \Sigma_t(\mathbf{r})\psi(\mathbf{r}, \mathbf{\Omega}_i) = \overbrace{Q(\mathbf{r}, \mathbf{\Omega}_i) + \Sigma_s(\mathbf{r})\phi(\mathbf{r})}^{S(\mathbf{r}, \mathbf{\Omega}_i)}, \tag{3}$$

where $\phi(\mathbf{r})$ is the scalar flux and defined by:

$$\phi(\mathbf{r}) \equiv \bar{\psi}(\mathbf{r}, \cdot) = \int_{S_2} \psi(\mathbf{r}, \mathbf{\Omega}) d\mathbf{\Omega} \simeq \sum_{j=1}^{N_{\text{dir}}} w_j \psi(\mathbf{r}, \mathbf{\Omega}_j). \tag{4}$$

Equation (3) is solved by iterating over the scattering source as described in Algorithm 1.

Each source iteration (SI) involves the resolution of a fixed-source problem (Line 4), for every angular direction. This is done by discretizing the spatial variable $\mathbf{r}$ of the streaming operator L. In this work, we focus on a 3D reactor core model, represented by a 3D Cartesian domain $\mathcal{D}$, and L is discretized using a Diamond Difference scheme (DD), as presented in [11]. The discrete form of the fixed-source problem is then solved by "walking" step by step throughout the whole spatial domain and to progressively compute angular fluxes in the spatial cells. In the literature, this process is known as the **sweep operation**. The vast majority of computations performed in the S_{N} method are part of the sweep operation.



(a) Angular quadrature in 2D. Directions are grouped in quadrants.

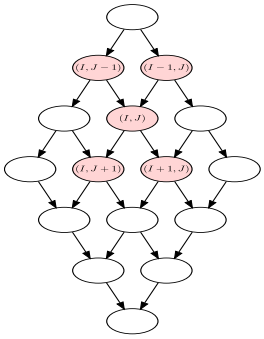
(b) In each direction, cells have two incoming components of the flux (Here, from the left and bottom faces: ψ_L and ψ_B), and generates two outgoing components of the flux (Here, on the right and top faces: ψ_R and ψ_T).

(c) Domain decomposition and boundary conditions. The corner cell $(0, 0)$ is the first to be processed for a quadrant, and its processing then allows for the processing of its neighbors (Here $(0, 1)$ and $(1, 0)$).

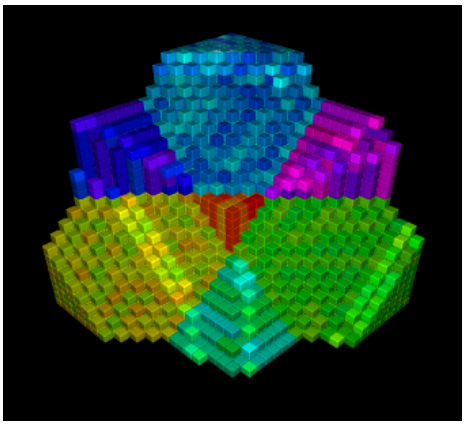
Figure 1: Illustration of the sweep operation over a 6×6 2D spatial grid for a single direction.

2.2. Sweep Operation

The sweep operation is used to solve the space-angle problem on Line 4 of Algorithm 1. It computes the angular neutron flux inside all cells of the spatial domain, for a set of angular directions. These directions are grouped into four quadrants in 2D (or eight octants in 3D). In the following, we focus on the first quadrant (labeled I in Figure 1a). As shown in Figure 1b, each cell has two incoming dependencies ψ_L and ψ_B for each angular direction. At the beginning, incoming fluxes on all left and bottom faces are known as indicated in Figure 1c. Hence, the cell $(0, 0)$ located at the bottom-left corner is the first to be processed. The treatment of this cell allows for the update of outgoing fluxes ψ_R and ψ_T, that satisfy the dependencies of cells $(0, 1)$ and $(1, 0)$. These dependencies on the processing of cells define a sequential nature throughout the progression of the sweep operation: two adjacent cells belonging to successive diagonals cannot be processed simultaneously. However, all cells belonging to a same diagonal can be processed in parallel. Furthermore, treatment of a single cell for all directions of the same quadrant can be done in parallel. Hence, step by step, fluxes are evaluated in all cells of the spatial domain, for all angular directions belonging to the same quadrant. The same operation is repeated for all the four quadrants. When using vacuum boundary conditions, there are no incoming neutrons to the computational domain and therefore processing of the four quadrants can be done concurrently. This sweep operation is subject to numerous studies regarding design and parallelism to reach highest efficiency on parallel architectures.



(a) A 2D single-quadrant Sweep's DAG over a 4x4 grid of `MacroCells`.



(b) Snapshot of an execution of the Sweep operation implemented on top of PaRSEC. `MacroCells` of similar colors are processed on the same node and highlighted ones are those that are in the process of being executed (at the time the snapshot was taken).

Figure 2: Sweep's DAG and snapshot.

2.3. Hierarchical Parallelization of the Sweep

In this section, we briefly describe the parallelization of the S_N-sweep operation on distributed multicore-based architectures. A detailed description of the Domino S_N-sweep can be found in [1] and [12].

As one can see from Figure 1, a space cell $c_{i,j}$ with Cartesian indices i and j can be processed as soon as both cells $c_{i-1,j}$ and $c_{i,j-1}$ have been computed. In order to reduce the cost of parallel communications, we do not consider individual cells but group them into `MacroCells` $C_{I,J}$ that correspond to rectangular sets of cells. Let $T_{I,J}$ be the task corresponding to the sweep inside a `MacroCell` $C_{I,J}$. The dependency between all the tasks:

$$(T_{I-1,J}, T_{I,J-1}) \rightarrow T_{I,J}$$

defines a Directed Acyclic Graph (DAG) which corresponds to the complete sweep from one corner of the spatial mesh to the opposite corner. An illustration of this DAG is presented in Figure 2a.

This DAG description, the task implementation and the data distribution over the computing nodes, are passed to a parallel runtime system. Here, we choose the PaRSEC [13] runtime system and its specific parameterized task graph to describe the algorithm. This format corresponds well with the regular pattern of our regular domain decomposition and allows the runtime system to schedule the tasks in a fully distributed manner without discovering integrally the graph of dependencies. In practice, PaRSEC exploits this pattern regularity to automatically schedule all computations on a set of threads per node (usually one thread per core), and triggers communications through an

```
   forall c ∈ MacroCell do
   |  ▷ c = (i, j, k)
3  |  forall d ∈ Directions[o] do
   |  |  ▷ d = (ν, η, ξ)
5  |  |  ϵ_x = 2ν/Δx;   ϵ_y = 2η/Δy;   ϵ_z = 2ξ/Δz;
6  |  |  ψ_vol = (ϵ_x ψ_L + ϵ_y ψ_B + ϵ_z ψ_F + S) / (ϵ_x + ϵ_y + ϵ_z + Σ_t);
7  |  |  ψ_R[c][d] = 2ψ_vol − ψ_L[c][d];
8  |  |  ψ_T[c][d] = 2ψ_vol − ψ_B[c][d];
9  |  |  ψ_BF[c][d] = 2ψ_vol − ψ_F[c][d];
10 |  |  φ[k][j][i] = φ[k][j][i] + ψ_vol ω[d];
```

Algorithm 2: `MacroCell` sweep

MPI layer when necessary. A snapshot of the execution on top of PaRSEC is depicted in Figure 2b.

In the case of vacuum boundary conditions, all 4 (resp. 8) angular sweep quadrants (resp. octants) are processed in parallel and again, handled via PaRSEC.

2.4. Arithmetic Intensity and Vectorization of the Tasks

We consider a single task $T_{I,J}$ corresponding to the sweep inside a `MacroCell` $C_{I,J}$. A `MacroCell` is a rectangular block of cells. Let A_x, A_y and A_z be the `MacroCell` sizes along the three dimensions. For the sake of readability, we will only consider a cubic `MacroCell` where $A = A_x = A_y = A_z$. The basic sweep algorithm for a given octant o containing a set of N_{do} directions `Directions[o]` is shown in algorithm 2.

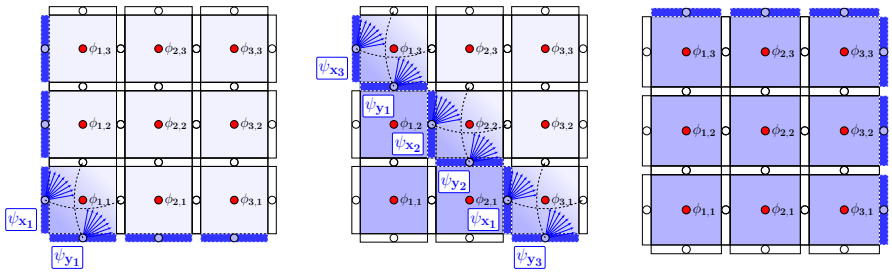
The aim of this algorithm is to solve the space-angle problem, by inverting the streaming operator of the monogroup transport equation. The volumic flux computation inside the cell c (line 6) needs to know incoming data for this cell: incoming angular fluxes, ψ_L, ψ_B, ψ_F, the total cross section Σ_t and the source term S. Outgoing angular fluxes are then updated on lines 7, 8 and 9. Finally we add the contribution of the volumic flux to the scalar flux on line 10, using the weight $\omega[d]$ associated to the direction d.

At this point, we can observe that, once a cell has been swept for a given octant, incoming angular fluxes are not longer used. This property allows for an optimization on the memory footprint of the solver: incoming and outgoing angular fluxes are stored on the same memory location, which dramatically increases the arithmetic intensity of the code. This feature is key in obtaining a code that leverages the full potential of modern multi-core architectures. This leads to the algorithm 3 where the $N_{do}A^3$ angular fluxes $\psi_{R,L,T,B,F,BF}$ defined for all cells (i, j, k) are replaced by $N_{do}A^2$ surface-based angular fluxes $\psi_{X,Y,Z}$. A 2D representation of this new algorithm is shown in Figure 3.

Let us count the total number of floating point operations (flop) per angular direction and per spatial cell in this sweep algorithm. We have 18 basic operations (add and mult) (the quantities $\frac{2}{\Delta u},\ u = x, y, z$ can be computed once

forall $c \in$ *MacroCell* **do**
  $\triangleright\ c = (i, j, k)$
  forall $d \in$ *Directions[o]* **do**
    $\triangleright\ d = (\nu, \eta, \xi)$
    $\epsilon_x = \frac{2\nu}{\Delta x}; \quad \epsilon_y = \frac{2\eta}{\Delta y}; \quad \epsilon_z = \frac{2\xi}{\Delta z};$
    $\psi_{\text{vol}} = \frac{\epsilon_x \psi_X + \epsilon_y \psi_Y + \epsilon_z \psi_Z + S}{\epsilon_x + \epsilon_y + \epsilon_z + \Sigma_t};$
    $\psi_X[k][j][d] = 2\psi_{\text{vol}} - \psi_X[k][j][d];$
    $\psi_Y[k][i][d] = 2\psi_{\text{vol}} - \psi_Y[k][i][d];$
    $\psi_Z[j][i][d] = 2\psi_{\text{vol}} - \psi_Z[j][i][d];$
    $\phi[k][j][i] = \phi[k][j][i] + \psi_{\text{vol}}\omega[d];$

Algorithm 3: `MacroCell` sweep with surface based angular flux DoFs



(a) Start of sweep (b) Sweep (c) End of sweep

Figure 3: `MacroCell` sweep in 2D with surface-based angular fluxes. At the beginning, the angular flux DoFs ψ_X and ψ_Y are known at the left and bottom sides. The blue arrows represent the angular flux DoFs corresponding to different angular directions. During the sweep, the angular DoFs for inner cells are computed and stored in the same surface-storage drawn in blue.

per spatial cell) and 1 floating point division (line 6). The question how many flop to count for one division operation is a tricky one as the answer depends on the target architecture. For our studies and following [14], we choose to count 7 flop per floating point division; this leads to a total of 25 flop for the sweep operation. The number of operations for a `MacroCell` sweep is then given by:

$$n_{\text{flop}} = 25A^3N_{do}. \tag{5}$$

A `MacroCell` contains A^3 different values for S, Σ and ϕ. In addition, a `MacroCell` contains A^2N_{do} DoFs for each surface-based angular flux component $\psi_{X,Y,Z}$. Hence the total number of floating point values to be accessed from the RAM for a `MacroCell` sweep is given by:

$$n_{\text{float}} = 3A^3 + 3A^2N_{do}, \tag{6}$$

which requires a volume of data transfers given by $s_{\text{float}}\, n_{\text{float}}$, where s_{float} represents the size of an FP number: 4 Bytes in Single Precision (SP), and 8 Bytes in Double Precision (DP). If we consider small enough `MacroCell` sizes A and typical numbers of directions per octant $N_{do} < 50$, then the input data should remain in cache memory during the `MacroCell` sweep computation. For a typical S_{12} quadrature ($N_{do} = 21$) and a `MacroCell` of size $A = 16$, a Single Precision sweep requires $4 \times 3(16^3 + 16^2 \times 21) = 111$ kB which fits within the 256 kB L2 cache memory attached to each core of our Xeon E5-2697v2 target computing node (microarchitecture codename Ivy Bridge).

The arithmetic intensity of the sweep, $\mathcal{I}_{\text{sweep}}$, is then given by the ratio of n_{flop} by the size of reads (S, Σ, ϕ and $\psi_{X,Y,Z}$) and writes (ϕ and $\psi_{X,Y,Z}$) during the sweep:

$$\mathcal{I}_{\text{sweep}} = \frac{25A^3N_{do}}{s_{\text{float}}\,(4A^3 + 6A^2N_{do})} = \frac{25}{2\,s_{\text{float}}}\frac{AN_{do}}{2A + 3N_{do}}. \tag{7}$$

For large enough `MacroCell` sizes A and numbers of directions by octant N_{do}, one can see that the arithmetic intensity can be very large. For our previous example, $N_{do} = 21$ and $A = 16$ in Single Precision, we obtain:

$$\mathcal{I}_{\text{sweep}} \simeq 11 \text{ flop/Byte}. \tag{8}$$

In [15] and following the so-called Roofline model [16], the authors evaluate the critical arithmetic intensity $\mathcal{I}^{\text{node}}_{\text{critical}}$ of our E5-2697v2 target computing node to be:

$$\mathcal{I}^{\text{node}}_{\text{critical}} = \frac{\text{Peak Perf}}{\text{Bandwidth}} = \frac{1000 \text{ Gflop/s}}{100 \text{ GB/s}} = 10 \text{ flop/Byte}. \tag{9}$$

Since $\mathcal{I}_{\text{sweep}}$ exceeds $\mathcal{I}^{\text{node}}_{\text{critical}}$, our `MacroCell` sweep kernel is said to be *compute bound* and its performance is only limited by the peak performance of the node. Consequently this kernel property allows for leveraging the full potential of SIMD acceleration.

Let $\text{nop}_{p,s}$ be the number of identical flop ($+$, $*$) that can be simultaneously applied to floating point values of precision $p \in \{\text{SP}, \text{DP}\}$ by a given

Angular Quadrature		S_2	S_4	S_8	S_{12}	S_{16}
Directions per octant (N_{do})		1	3	10	21	36
AVX,SP (nop = 8)	SpeedUp:	1	3	5	7	7.2
	Efficiency:	12.5%	37.5%	62.5%	87.5%	90%
AVX,DP (nop = 4)	SpeedUp:	1	3	3.3	3.5	4
	Efficiency:	25%	75%	83.3%	87.5%	100%

Table 1: Maximal SIMD-AVX speed-ups and efficiencies for Single and Double Precision and different S_N quadratures.

SIMD instruction set $s \in \{\text{SSE}, \text{AVX}, \text{AVX2}, \dots\}$. Each s is characterized by a given SIMD width w_s and one can compute $\text{nop}_{p,s} = w_s/s_p$. On our target architecture (AVX) we have:

$$\text{nop}_{\text{SP, AVX}} = \frac{w_{\text{AVX}}}{s_{\text{SP}}} = \frac{256 \text{ bits}}{32 \text{ bits}} = 8. \tag{10}$$

In addition, for each core, two simultaneous pipelines can execute $\text{nop}_{p,s}$ additions and $\text{nop}_{p,s}$ multiplications at each clock cycle. Consequently, the SP peak performance of one E5-2697v2 node with 24 cores running at 2.7 GHz is:

$$\text{Peak}_{\text{SIMD}} = 2.7 \times 24 \times 8 \times 2 = 1036 \text{ Gflop/s}. \tag{11}$$

Without SIMD operations, a maximum of two SP operations are done at each clock cycle. This reduces the maximal peak performance to:

$$\text{Peak}_{\text{scalar}} = \frac{\text{Peak}_{\text{SIMD}}}{\text{nop}_{\text{SP, AVX}}} = 128 \text{ Gflop/s} = 12\% \text{ Peak}_{\text{SIMD}}. \tag{12}$$

SIMD parallelism is applied to perform the sweep operations that correspond to the N_{do} different angular directions of the same octant inside each spatial cell. For S_N *Level Symmetric* quadratures, we have $N_{do} = N_{\text{dir}}/8 = N(N+2)/8$. Note that SIMD parallelism imposes to perform exactly $\text{nop}_{p,s}$ simultaneous operations at a time. Therefore the maximal expected SIMD speed-ups and efficiencies are obtained for N_{do} being a multiple of $\text{nop}_{p,s}$. Table 1 summarizes the ideal speed-ups and efficiencies of this angular-based vectorization strategy.

A detailed description of this *vectorized* implementation based on Eigen, a C++ template library, can be found in [6] and [12]. In particular in Chapter 2 of [12], the author shows that the SIMD (SSE) vectorized version of DOMINO reaches a speed-up of 4 compared to the scalar version. Our sweep kernel performance strongly depends on SIMD operations and section 4.1 shows that an overall performance of 6.6 Tflop/s of the sweep is reached using 32 computing nodes. This performance corresponds to 20% of the SIMD peak performance ($32 \times 1.036 = 33$ Tflop/s). Although all the parallel overheads of the many nodes parallelism is included in this measurement, this performance ratio (20%) exceeds by a large factor the maximal ratio (12%) that can be reached with scalar operations only.

3. Acceleration of Scattering Iterations using PDSA

In highly diffusive media ($\Sigma_s \approx \Sigma_t$), the convergence of the Source Iteration algorithm (alg. 1) is very slow, and therefore a numerical acceleration scheme must be combined with this algorithm in order to speed-up its convergence. One of the widely used acceleration schemes in this case is the Diffusion Synthetic Acceleration (DSA) [17].

3.1. Diffusion Synthetic Acceleration

Here we just recall the basics of this method, and the reader can refer to the paper [17] for more details regarding its effectiveness and the Fourier analysis characterizing its convergence properties. Let us define $\epsilon^{k+\frac{1}{2}} = \psi - \psi^{k+\frac{1}{2}}$, as the error of the solution obtained after the $k + \frac{1}{2}^{\text{th}}$ iteration of the SI scheme, relative to the exact solution ψ, as defined by equation (3). The error ϵ satisfies the following transport equation:

$$L\epsilon^{k+\frac{1}{2}}(\mathbf{r}, \mathbf{\Omega}_i) = \Sigma_t(\mathbf{r})\bar{\epsilon}^{k+\frac{1}{2}} + \Sigma_s(\mathbf{r})\left(\phi^{k+\frac{1}{2}}(\mathbf{r}) - \phi^k(\mathbf{r})\right), \tag{13}$$

where $\bar{\epsilon}$ is the scalar field associated to ϵ and defined as in equation (2). However, equation (13) is as difficult to solve as the original fixed-source transport problem (3). Nevertheless, if an approximation $\tilde{\epsilon}$ of $\bar{\epsilon}$ was available, then the scalar flux could be updated to:

$$\phi^{k+1}(\mathbf{r}) = \phi^{k+\frac{1}{2}}(\mathbf{r}) + \tilde{\epsilon}^{k+\frac{1}{2}}(\mathbf{r}).$$

The idea of the DSA method is then to use a diffusion approximation, yielding an approximate correction term $\tilde{\epsilon}$, instead of solving the transport equation (13). In Domino, the diffusion approximation is obtained using the Diabolo solver [18], which implements the Simplified P_{N} (SP_{N}) method as presented in [19], in a mixed-dual formulation. When approximating equation (13) with a diffusion operator, the problem solved by Diabolo can be stated as the following mixed dual formulation:

Find $(\tilde{\epsilon}^{k+\frac{1}{2}}, \mathbf{j}^{k+\frac{1}{2}}) \in L^2(\mathcal{D}) \times H(\mathcal{D}, \text{div})$ such that:

$$\begin{cases} \text{div}\,\mathbf{j}^{k+\frac{1}{2}}(\mathbf{r}) + \Sigma_a\tilde{\epsilon}^{k+\frac{1}{2}}(\mathbf{r}) = \Sigma_s(\mathbf{r})\left(\phi^{k+\frac{1}{2}}(\mathbf{r}) - \phi^k(\mathbf{r})\right) & \text{in } \mathcal{D}, \\ \dfrac{1}{D}\mathbf{j}^{k+\frac{1}{2}}(\mathbf{r}) + \nabla\,\tilde{\epsilon}^{k+\frac{1}{2}}(\mathbf{r}) = \mathbf{0} & \text{in } \mathcal{D}, \\ \tilde{\epsilon}^{k+\frac{1}{2}} = 0 & \text{on } \partial\mathcal{D}, \end{cases} \tag{14}$$

in which we introduced the diffusion coefficient D and the neutronic current $\mathbf{j}^{k+\frac{1}{2}}$ associated to $\tilde{\epsilon}^{k+\frac{1}{2}}$. Within Diabolo, these equations are spatially discretized using an RTk finite elements scheme [20, 21] (see Figure 4), which is consistent with the DD scheme used for the discretization of the transport equation as proven in [11]. Therefore, the stability of the acceleration scheme is ensured.

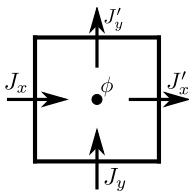


Figure 4: RT0 finite element in 2D: 5 DoFs (4 for the currents J_x, J'_x, J_y, J'_y and 1 for the scalar flux ϕ). DSA is applied using an RT0 element.

3.2. Discussion on DSA in industrial and parallel contexts

When integrated into a parallelized transport solver, DSA may become a bottleneck for the scalability of the transport solver if, for instance, a serial implementation of the diffusion solver is used. Several techniques may be used in order to prevent the acceleration solver from negatively impacting the performance of the transport.

First, one could consider departing from the standard Source Iteration & DSA scheme. For example, DENOVO uses a Krylov solver [9], which converges faster than the traditional multi-group Gauss-Seidel algorithm and angular Source Iteration and alleviates the need for an acceleration scheme. Such a Krylov solver can still be further preconditioned, for example using multigrid methods in energy [22]. While very efficient and scalable, the implementation of such techniques makes the reference neutron transport code share few software components, or even algorithms, with the rest of the industrial platform. This makes the development, maintenance and verification price heavy to pay for the industry.

A more natural solution to consider would consist in keeping the traditional DSA scheme, and parallelizing it alongside the transport solver. Such a strategy is most interesting if it allows reusing existing neutron diffusion solvers, as was the case in the situation described above, where the sequential version of DOMINO makes use of DIABOLO. However, industry-grade neutron diffusion solvers are in general sequential, or limited to shared-memory parallelism. This is in particular the case of DIABOLO, which features a high performance implementation making use of vectorization and multi-thread parallelism, but no distributed-memory parallelism. This is due to the elliptic nature of the diffusion equation which makes the parallelization of such solvers a challenging task. Although parallel diffusion solvers can be implemented [7, 8], the induced code complexity is often considered a heavy price to pay.

DOMINO uses a third option, the Piecewise Diffusion Synthetic Acceleration (PDSA), in which only local diffusion problems are solved, and there is no need to set-up a global, parallel resolution process for the diffusion problem. Indeed, when keeping the principle of source iteration with synthetic acceleration, all that is needed is that the acceleration operator produces approximate solutions to equation (13), in such a way that the Source Iteration process is accelerated. In a parallel context, one may also want to require the acceleration operator to

be easily parallelizable, if possible in a scalable way. Finally, in an industrial context, an additional desirable property of the acceleration operator would be to re-use as much as possible the existing features of a sequential (or shared-memory-parallelized) industrial solver such as DIABOLO.

As detailed in [2], the Piecewise Diffusion Synthetic Acceleration fulfills all these requirements. The general principle of PDSA will be summarized below in section 3.3. The steps required for its implementation will be further presented in sections 3.4 and 3.5. Section 3.6 prolongs the present discussion by comparing PDSA to the above mentioned strategy of parallelizing a neutron diffusion solver using Domain Decomposition techniques.

3.3. Piecewise Diffusion Synthetic Acceleration

The general presentation and the convergence proof of PDSA are given in [2]. We recall that the purpose of this method is similar to that of DSA: evaluate an approximation, $\tilde{\epsilon}^{k+\frac{1}{2}}$, of the error on the scalar flux, $\bar{\epsilon}^{k+\frac{1}{2}}$, to be used for correcting the scalar flux, $\phi^{k+\frac{1}{2}}$. In the following, iteration indices $k+\frac{1}{2}$ will be dropped for the sake of readability.

We assume that the spatial domain $\mathcal{D}$ is split, along the 3 spatial dimensions, into $N = P \times Q \times R$ non-overlapping subdomains $\mathcal{D}_I$ such that: $\mathcal{D} = \cup_{I\in\mathcal{I}}\mathcal{D}_I$, where

$$\mathcal{I} = [\![1, P]\!] \times [\![1, Q]\!] \times [\![1, R]\!].$$

We set: $\Gamma_{IJ} = \partial\mathcal{D}_I \cap \partial\mathcal{D}_J$ the non-empty interfaces between subdomains of index I and J; $\Gamma_I = \partial\mathcal{D} \cap \partial\mathcal{D}_I$ and $\mathbf{n}_I$ the unit normal vector to $\partial\mathcal{D}_I$ and $\tilde{\epsilon}^I = \tilde{\epsilon}|_{\mathcal{D}_I}$ and $\mathbf{j}^I = \mathbf{j}|_{\mathcal{D}_I}$, the respective restrictions of $\tilde{\epsilon}$ and $\mathbf{j}$ to subdomain $\mathcal{D}_I$.

Unlike the DSA method which consists in solving a single diffusion problem on the global domain $\mathcal{D}$, the PDSA method is based on the resolution of two diffusion problems on each of the subdomains $\mathcal{D}_I$. These diffusion problems differ with respect to the boundary conditions applied on the subdomains: the first problem uses homogeneous Neumann boundary conditions (equation (15)), and the second one uses non-homogeneous Dirichlet boundary conditions (equation (16)). In both equations, notations were shortened by using $S^I(\mathbf{r})$ to denote the right-hand side of equation (14).

$$\begin{cases} \operatorname{div}\mathbf{j}_N^I(\mathbf{r}) + \Sigma_a\tilde{\epsilon}_N^I(\mathbf{r}) = S^I(\mathbf{r}) & \text{in } \mathcal{D}_I \\ \nabla\,\tilde{\epsilon}_N^I(\mathbf{r}) + \dfrac{1}{D}\mathbf{j}_N^I(\mathbf{r}) = \mathbf{0} & \text{in } \mathcal{D}_I \\ \tilde{\epsilon}_N^I = 0 & \text{on } \Gamma_I \\ \nabla\tilde{\epsilon}_N^I \cdot \mathbf{n}_I = 0 & \text{on } \Gamma_{IJ} \end{cases} \tag{15}$$

$$\begin{cases} \operatorname{div}\mathbf{j}_D^I(\mathbf{r}) + \Sigma_a\tilde{\epsilon}_D^I(\mathbf{r}) = S^I(\mathbf{r}) & \text{in } \mathcal{D}_I \\ \nabla\,\tilde{\epsilon}_D^I(\mathbf{r}) + \dfrac{1}{D}\mathbf{j}_D^I(\mathbf{r}) = \mathbf{0} & \text{in } \mathcal{D}_I \\ \tilde{\epsilon}_D^I = 0 & \text{on } \Gamma_I \\ \tilde{\epsilon}_D^I = \dfrac{\tilde{\epsilon}_N^I + \tilde{\epsilon}_N^J}{2} & \text{on } \Gamma_{IJ} \end{cases} \tag{16}$$

It is important to note that this two-step PDSA process is not an iterative one: equations (15)–(16) are solved once per outer transport iteration, instead of solving problem (14) in a standard DSA process. The two PDSA steps produce a correction term $\tilde{\epsilon}_D$, which can be used to compute an accelerated S_N flux

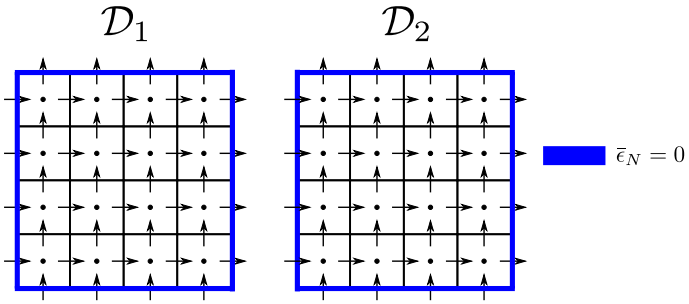
$$\phi^{k+1} = \phi^{k+\frac{1}{2}} + \tilde{\epsilon}_D.$$

It is shown in [2] that, for sufficiently diffusive and optically thick problems, this two-step PDSA process is close enough from a global diffusion resolution that it preserves the good properties of a standard DSA. In particular, a theoretical threshold is given on the optical thickness of any subdomain, above which PDSA is guaranteed to provide the same spectral radius (and therefore the same number of outer iterations) as standard DSA. As shall be seen in the experimental results of section 4.1, PWRs seem to be optically thick enough for PDSA to work effectively for the targeted number of subdomains.

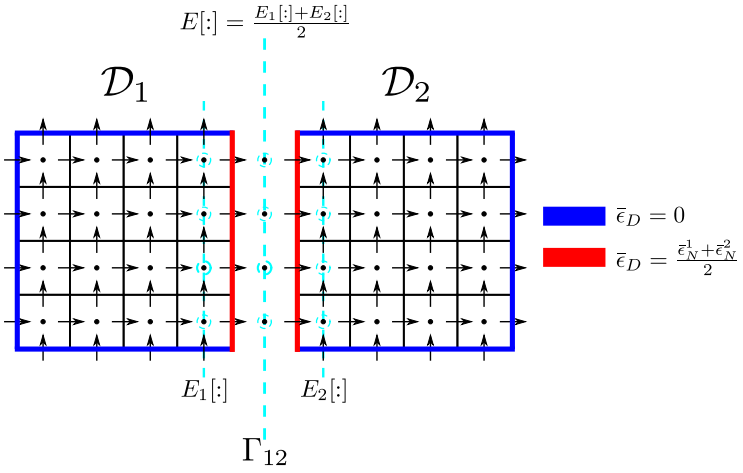
3.4. Practical implementation of PDSA

On the practical side, it is important to note that the application of the PDSA method using a classical diffusion solver does not require many changes. Homogeneous Dirichlet boundary conditions used over Γ_I are classically implemented to simulate whole cores; homogeneous Neumann boundary conditions used over Γ_{IJ} in equation (15) are likewise featured by most diffusion solvers to implement symmetric domains. However, the second PDSA step (16) requires implementing a non-homogeneous Dirichlet boundary condition, which is not standard. In our case we are using a mixed-dual formulation of the SP_N equations, therefore these boundary conditions are natural and not essential and their implementation is straightforward. An illustration of the processing of the boundary conditions in the case of two subdomains is presented in Figure 5. This figure shows how the half-sum of cell-based flux DoFs from the first (Neumann) step, is injected into edge- or face-based values for the expression of Dirichlet boundary conditions in the second step.

Another important aspect to account for is the fact that DIABOLO internally uses iterative solvers to compute the solution to neutron diffusion problems. Care should therefore be taken to correctly feed this iterative solver with appropriate initial values and stopping criteria. Since both steps of PDSA share the same equation and source term, and differ only with respect to their boundary conditions, it has been found that the number of iterations of the second (Dirichlet) PDSA step could be reduced by initializing it with the solution to the first (Neumann) PDSA step. As for the stopping criteria of the SP_N iterative solver, it has also been found that limiting the number of iterations to fixed values had negligible impact on the effectiveness of the acceleration, while reducing the time spent in the PDSA calculations. In the experimental results of section 4.1, the DIABOLO SP_N solver has been set to perform only one iteration per PDSA step. Even though it produces very approximate solutions to equations (15) and (16), this has been found to have negligible effect on the global number of outer iterations.



(a) The first step consists of solving two diffusion problems in parallel on $\mathcal{D}_1$ and $\mathcal{D}_2$, with Neumann boundary conditions.



(b) The second step also solves two diffusion problems, but with non-homogeneous Dirichlet boundary conditions: null flux boundary conditions on the external boundary of the domain and an average value of the flux at the inner interface.

Figure 5: Illustration of the PDSA method on a domain split in two.

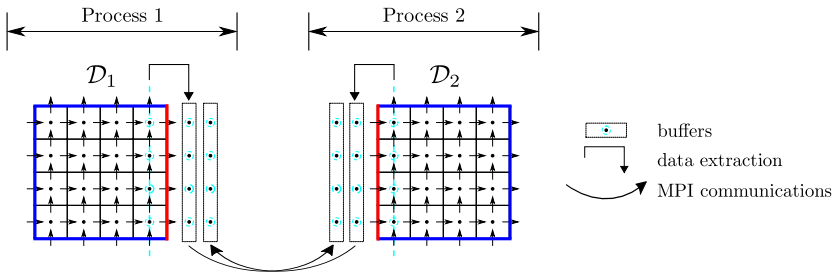


Figure 6: Illustration of the communication pattern in the PDSA method on a domain split in two. Two point-to-point communications are needed to exchange flux information at the interface between the two subdomains.

3.5. Parallelization of the PDSA Method

Figure 6 illustrates a parallel implementation of the PDSA method in 2D, when the global domain is partitioned in two subdomains.

The partitioning of the global domain uses the same block data distribution as for the sweep operation. As we mentioned previously in section 3.1, the diffusion problem on each subdomain is solved using our SP_N solver Diabolo which is parallelized on shared memory systems using the Intel TBB framework.

Hence, by mapping each subdomain to a single process, the resolution of the diffusion problems on $\mathcal{D}_1$ and $\mathcal{D}_2$, when applying the PDSA method, is naturally performed in parallel. Moreover, for the first step, the use of Neumann boundary conditions requires no communication with the neighboring processes. However, in the second step, each process needs to have the average value of the scalar flux at the interfaces between its neighbors. Therefore, each process must perform send and receive operations to exchange data with its neighbors.

These data exchanges are point-to-point communications as only two processes are involved for each data exchange. Moreover, the diffusion problems being considered here are much smaller than the transport sweeps, in that they only consider the space variable and not the angle. It is therefore expected that this process scales very well in parallel, and will be solved in negligible time with respect to the transport sweeps. Both these expectations will be verified in the scalability study performed in section 4.1. In any case, it follows from this that the subdomains definition should be optimized for the transport only; PDSA should be used with the same decomposition in order to minimize data transfers.

3.6. Comparison of PDSA and Domain Decomposition techniques

The PDSA technique described here might be considered extremely similar to a parallel implementation of standard DSA using Domain Decomposition techniques. Indeed, the two steps of Eqs. (15) & (16) in PDSA correspond to the first iteration of a Domain Decomposition technique called the Dirichlet-Dirichlet algorithm in [23].

There exist however significant differences between PDSA and DD techniques. First, DD methods converge to the solution of the global diffusion equation. The number of iterations required to produce a solution might lead to poor global scalability [24]. In contrast, PDSA uses only two steps[2] to produce a correction which is not the solution to a global diffusion problem – but still fulfills in practice the required conditions to accelerate Source Iterations. For the same reasons, any method which might increase the efficiency of the diffusion solver (such as multigrid methods for example) will be best used at the level of the sub-domain, but the formalism of PDSA should be kept for parallelization between subdomains.

From the scalability standpoint, PDSA only requires one local point-to-point communication to exchange boundary fluxes between neighbouring subdomains. This is the least possible amount of communication any parallel process might need, and PDSA is thus optimal in this sense.

Last, from the vantage point of the implementation, PDSA requires very limited work. The only requirement which might not be readily available in a neutron diffusion solver is the availability of non-homogeneous Dirichlet boundary conditions.

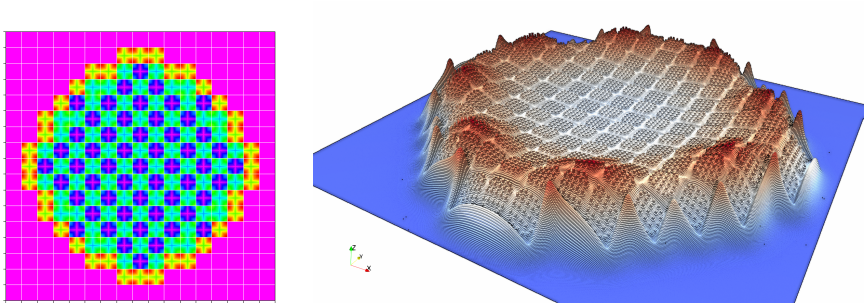
For these reasons, while DD methods for a standard DSA might have provided viable alternatives, PDSA has been considered as the primary choice in the parallel version of Domino.

4. Performance of Domino

The performance and accuracy of the single-domain Domino implementation, based on standard DSA acceleration, has been assessed in [12] and [5] where comparisons with both reference Monte-Carlo and deterministic solvers have been conducted. In this section, we assess the performance of the present multi-domain Domino implementation based on the PDSA method for solving a set of PWR nuclear core benchmarks.

These benchmarks correspond to a PWR 900 MW core, and enable 2, 8 and 26 energy groups calculations to be performed. A full description of these benchmarks is available in [25] and [26]. All benchmarks represent a simplified 3D PWR first core loaded with 3 different types of fuel assemblies characterized by different Uranium-235 enrichment levels (low, medium and highly enriched uranium ranging from 1.5% to 3.25%), in a configuration where all control rods are extracted. Along the z-axis, the 360 cm assembly is axially reflected with 30 cm of water which results in a total core height of 420 cm. The 3 types of fuel assemblies appear on Figure 7a where the central assembly corresponds to the lowest enrichment, while the last row of fuel assemblies has the highest enrichment to flatten the neutron flux. Each fuel assembly is a 17×17 array of fuel pins, with a lattice pitch of 1.26 cm that contains 264 fuel pins and 25

[2] which is optimal, in the sense that any DD method is also expected to perform at least two iterations before it reaches convergence



(a) Material Radial Map (b) Neutron Thermal Flux

Figure 7: Illustration of the 2-Group PWR 900 MW model [25]

water holes. The boundary condition associated with this benchmark problem is a pure leakage condition without any incoming angular flux. The associated nuclear data, 2-group, 8-group and 26-group libraries are derived from a fuel assembly heterogeneous transport calculation performed with the cell code DRAGON [27]. As an example, Figure 7b presents a visualization of the thermal flux in the central radial plane, as obtained from a 2-group calculation.

Table 2 summarizes the discretization parameters for the considered benchmarks, where the following notations are used:

- N_{G} is the number of energy groups.
- N_x, N_y and N_z define the number of spatial cells along the three dimensions of the spatial domain.
- N_{dir} is the number of angular directions according to the order of angular quadrature in use.
- N_{DOF} is the number of degrees of freedoms (DoFs). The calculation of DoF numbers consider 3 DoFs per cell, per energy group and per angular direction. These DoFs correspond to the 3 incoming angular fluxes ψ_R, ψ_L and ψ_F introduced in section 2.4. The DoFs corresponding to the cell average flux ϕ, independent from angle, are neglected for computing N_{DOF}.
- flop is the number of floating point operations required to perform a single complete sweep operation, for all energy groups. Note that the sweep of a single spatial cell for a single angular direction requires 25 flop (see section 2.4).
- A_x, A_y and A_z define the `MacroCell` sizes along the three dimensions. Experimentally, $A_{x,y,z} = 16$ was shown to be the most effective choice for the 2-group benchmark.
- $\epsilon_{k_{\mathrm{eff}}}$ and ϵ_{ψ} define the thresholds used to check the stopping criteria at iteration $n+1$ of the power algorithm for the eigenvalue and on the fission

Case name	N_{G}	N_x	N_y	N_z	N_{dir}	N_{DOF} $\times 10^{12}$	flop $\times 10^{12}$	$A_{x,y,z}$	$\epsilon_{k_{\mathrm{eff}}}$	ϵ_{ψ}	I_g
S_{12} 2-group	2	578	578	756	168	0.254	2.12	16	10^{-6}	10^{-5}	1
S_8 8-group	8	578	578	168	80	0.108	0.90	20	10^{-5}	10^{-5}	5
S_{16} 26-group	26	578	578	140	288	1.051	8.75	20	10^{-5}	10^{-5}	4

Table 2: Description of PWR benchmarks and calculation parameters.

source respectively as follows:

$$\frac{|k_{\mathrm{eff}}^{n+1} - k_{\mathrm{eff}}^{n}|}{k_{\mathrm{eff}}^{n}} < \epsilon_{k_{\mathrm{eff}}}, \quad \frac{||\mathcal{F}\psi^{n+1} - \mathcal{F}\psi^{n}||}{||\mathcal{F}\psi^{n}||} < \epsilon_{\psi}. \tag{17}$$

- I_g is the fixed number of Gauss-Seidel iterations for the multigroup problem.

Note that the spatial mesh used for the PWR benchmarks is based on a pin-cell mesh in the $x-y$ plane. Each pin-cell is then subdivided into 70 (resp. 84) slices in the z direction for the 26-group (resp. 2-group and 8-group). The spatial mesh is then further refined by $2 \times 2 \times 2$ for the 8-group and 26-group, and by $2 \times 2 \times 9$ for the 2-group. The larger spatial mesh for the 2-group case enables the study of the strong scalability of our implementation at high core count.

The experiments have been conducted on computing nodes (dual Intel Xeon E5-2697v2 processors) of the ATHOS cluster at EDF. The theoretical peak performance of each node is 518 (resp. 1036) Gflop/s in double (resp. single) precision (2×12 AVX cores at 2.7 GHz). The following experiments were conducted by launching one MPI process per computing node and as many threads as available cores; keeping one core per node for the PaRSEC communication thread. This configuration has been shown in [1, 12] to be the most efficient for the PaRSEC environment, especially with respect to the configuration with one MPI process per core. All experiments were conducted in single precision. Computation times do not include setup (reading of cross-section files from the hard disk), but include all communications and stopping criterion checks. For all the experiments presented in the following sections, the setup time is less than a minute. Note that the `MacroCell` size chosen ($A_{x,y,z} = 16$) is large enough to ensure that the runtime overhead to schedule one task is negligible compared to the cost of computing a single task.

4.1. Strong scalability of 2-group PWR k_{eff} computation

In this section, we present full-core k_{eff} computations using the S_{12} 2-group 3D PWR core model. As explained in section 3.4, a preliminary study reveals that the SP_{N} iterative solver can be set up to perform only one iteration per resolution of equations (14), (15) or (16). All results presented hereafter were obtained using this parameter.

(P,Q,R) Accel		$(1,1,1)$ No DSA	$(1,1,1)$ DSA	$(4,2,1)$ PDSA	$(4,4,1)$ PDSA	$(4,4,2)$ PDSA	$(4,4,4)$ PDSA
N_{cores}		24	24	192	384	768	1536
N_{outer}		315	56	59	66	69	71
T_{sweep}	(s)	3610	598.0	83.2	50.0	29.4	23.5
T_{SPN}	(s)	-	114.7	14.6	8.0	4.2	2.2
T_{PDSA}	(s)	-	147.0	22.8	11.6	7.1	3.4
T_{total}	(s)	4547	916.4	130.5	75.5	43.8	31.1
% sweep		79	65	64	66	67	75

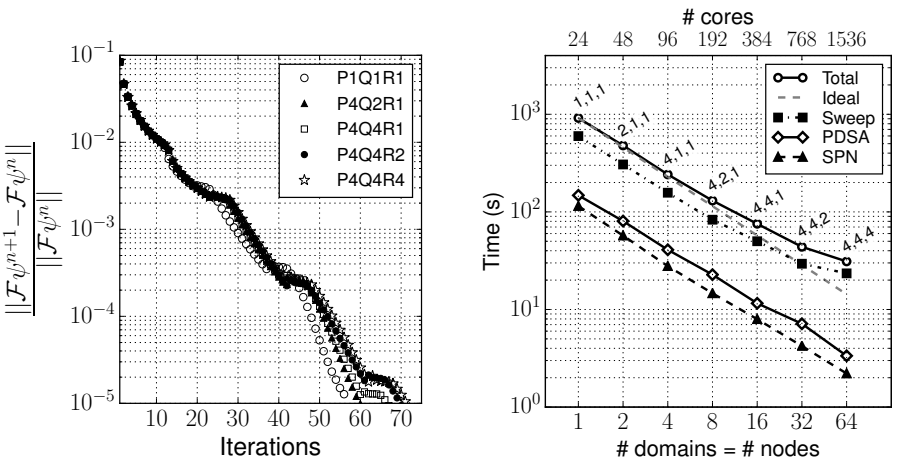
Table 3: Solution times for a S_{12} 2-group 3D PWR k_{eff} computation on the ATHOS platform.

Figure 8a compares the convergence of the standard DSA method with one subdomain and those of PDSA with 4, 16, 32 and 64 subdomains. For each case, the domain is split into a regular grid of $x \times y \times z$ subdomains, denoted by $PxQyRz$ in the labels of Figure 8a, and denoted by x, y, z on each dot of our PDSA algorithm in Figure 8b. The distribution of each case is chosen based on the model introduced in [1]. All the computations lead to the same k_{eff} (1.019574) and to the same fluxes. The outer iteration number increases from 56 for one subdomain to 71 for 64 subdomains. This small increase demonstrates that the PDSA method is a suitable parallel acceleration technique for PWR core problems. Table 3 summarizes these DSA and PDSA results and compares them to the non-accelerated computation which requires 315 outer iterations to reach the convergence criterion. Figure 8b illustrates DOMINO's strong scalability. The total computing time evolution and its main components are displayed. T_{SPN} corresponds to the time spent in the local solution process of both PDSA diffusion subproblems in all subdomains. It scales perfectly and remains negligible for all the parallel range. T_{PDSA} represents the time spent within the whole PDSA algorithm. It includes, besides T_{SPN}:

- the time required for local data transfers, between S_{N} and SP_{N} solvers;
- the communication time required for sending/receiving data to/from neighbouring processors during the second step of the algorithm.

The global parallel efficiency is mainly limited by the scalability of the sweep. These results show that the PDSA method allows DOMINO to achieve very good performance for PWR criticality computations.

The distributed memory nodes are efficiently used by this domain decomposition approach. Each multi-core node parallel potential is efficiently exploited by multi-threaded implementation of the sweep and mono-domain diffusion solvers. The core SIMD units, handling the third and innermost parallel level, are efficiently used to simultaneously compute several components of the angular flux. These three nested levels of parallelism allow DOMINO to exploit a large fraction of the computing power of the parallel platform. For this S_{12} (resp. S_{16}) 2-group PWR Benchmark running on 768 cores, the performance of the sweep



(a) PDSA convergence

(b) PDSA timings

Figure 8: Convergence and elapsed CPU time of DOMINO using the S_{12} 2-group PWR benchmark for various multi-domain configurations. A (P, Q, R) configuration divides the spatial domain in P (resp. Q,R) slices in the X (resp. Y,Z) direction.

Case name	N_{outer}	T_{sweep} (s)	T_{SPN} (s)	$T^{\text{comm}}_{\text{PDSA}}$ (s)	T_{total} (s)
S_8 8-group	65	91.53	7.84	0.86	128.85
S_{16} 26-group	126	2226.42	56.56	147.2	2763.52

Table 4: Solution times for a S_{12} 8-group and S_{16} 26-group 3D PWR k_{eff} computation on 64 cluster nodes (4,4,4).

operation reaches 5.0 (resp. 6.6) Tflop/s which corresponds to 15% (resp. 20%) of the peak performance of the corresponding 32 nodes.

4.2. PWR core mode with 8 *and* 26*-groups*

Table 4 presents performance results of S_8 8-group and S_{16} 26-group 3D PWR k_{eff} computations using 64 computing nodes of the ATHOS cluster, partitioned into $(4, 4, 4)$.

The convergence on the 8-group benchmark is reached in 65 external iterations, and the obtained eigenvalue is $k_{\text{eff}} = 1.009408$. This number of external iterations is similar to what was obtained for a run with a single subdomain. The total computation time is 128.85 s of which 91.53 s comes from the sweep operation, illustrating that the sweep operation is still dominant (71% of the total time).

For the 26-group case, the convergence is reached in 126 outer iterations, for a global solver time of 2763.52 s. The obtained eigenvalue is $k_{\text{eff}} = 1.008358$. As in the case with 8-groups, we did not observe any increase in the number of external iterations as compared to a run with a single domain. This is a

remarkable result, highlighting the very good suitability of the PDSA method on representative benchmarks of our target applications.

5. Conclusion

In this paper, we studied the performance of our massively parallel approach for solving the neutron transport equation according to the discrete ordinates method. We first presented our task-based implementation of the sweep with PaRSEC, as implemented in the Domino solver. Then we presented an application of PDSA, a new piecewise diffusion acceleration scheme for the scattering iterations. This is required to speed-up the convergence for strongly diffusive problems. The PDSA approach has been shown to effectively accelerate the different PWR nuclear core criticality computations on which it has been tested, since it matches the standard DSA convergence rate on sequential runs, and does not degrade the convergence properties of parallel runs. These results are very satisfying, especially considering that the implementation of PDSA requires very limited work. The Cartesian transport solver Domino, implementing the PDSA scheme, exhibits three nested levels of parallelism and exploits a large fraction of the theoretical peak performance of thousands of SIMD computing cores. As a result, Domino can complete very large and high-fidelity criticality computations involving more than 10^{12} degrees of freedom in less than an hour using 64 super-computer nodes.

This result allows us now to consider future fast and accurate 3D time-dependent transport solutions for diffusive problems.

Acknowledgment

The authors would like to thank the reviewers for their thoughtful comments and efforts towards improving our manuscript.

References

References

[1] S. Moustafa, M. Faverge, L. Plagne, P. Ramet, 3D Cartesian Transport Sweep for Massively Parallel Architectures with PaRSEC, in: Parallel and Distributed Processing Symposium (IPDPS), 2015 IEEE International, 2015, pp. 581–590. doi:10.1109/IPDPS.2015.75.

[2] F. Févotte, PDSA: a piecewise diffusion synthetic acceleration scheme for neutron transport simulations in diffusive media, in preparation for submission to Journal of Computational Physics.

[3] R. E. Alcouffe, Diffusion synthetic acceleration methods for the diamond-differenced discrete-ordinates equations, Nuclear Science and Engineering 64 (2) (1977) 344–355.

[4] N. Martin, A. Hébert, A three-dimensional high-order diamond differencing discretization with a consistent acceleration scheme, Annals of Nuclear Energy 36 (11-12) (2009) 1787 – 1796. doi:10.1016/j.anucene.2009.08.014.

[5] T. Courau, S. Moustafa, L. Plagne, A. Ponçot, DOMINO: A fast 3D Cartesian discrete ordiantes solver for reference pwr simulations and SPN validations, in: International Conference on Mathematics and Computational Methods Applied to Nuclear Science & Engineering (M&C 2013), USA, 2013.

[6] S. Moustafa, I. Dutka-Malen, L. Plagne, A. Ponçot, P. Ramet, Shared memory parallelism for 3D Cartesian discrete ordinates solver, Annals of Nuclear Energy 82 (2015) 179 – 187, joint International Conference on Supercomputing in Nuclear Applications and Monte Carlo 2013, SNA + MC 2013. Pluri- and Trans-disciplinarity, Towards New Modeling and Numerical Simulation Paradigms. doi:http://dx.doi.org/10.1016/j.anucene.2014.08.034.
URL http://www.sciencedirect.com/science/article/pii/S0306454914004265

[7] E. Jamelot, P. Ciarlet Jr, Fast non-overlapping Schwarz domain decomposition methods for solving the neutron diffusion equation, Journal of Computational Physics 241 (2013) 445 – 463. doi:http://dx.doi.org/10.1016/j.jcp.2013.01.026.
URL http://www.sciencedirect.com/science/article/pii/S0021999113000703

[8] M. Barrault, B. Lathuilière, P. Ramet, J. Roman, Efficient parallel resolution of the simplified transport equations in mixed-dual formulation, Journal of Computational Physics 230 (5) (2011) 2004 – 2020. doi:http://dx.doi.org/10.1016/j.jcp.2010.11.047.
URL http://www.sciencedirect.com/science/article/pii/S0021999110006625

[9] G. G. Davidson, T. M. Evans, J. J. Jarrell, R. N. Slaybaugh, Massively parallel, three-dimensional transport solutions for the k-eigenvalue problem, in: International Conference on Mathematics and Computational Methods Applied to Nuclear Science & Engineering (M&C 2011), Brazil, 2011.

[10] B. Carlson, C. Lee, Mechanical quadratures and the transport equation, Tech. Rep. LA-2573, Los Alamos Scientific Laboratory (1961).

[11] A. Hébert, High order diamond differencing schemes, Annals of Nuclear Energy 33 (17–18) (2006) 1479 – 1488. doi:10.1016/j.anucene.2006.10.003.

[12] S. Moustafa, Massively parallel cartesian discrete ordinates method for neutron transport simulation, Ph.D. thesis, Université de Bordeaux (December 2015).

[13] G. Bosilca, A. Bouteiller, A. Danalis, T. Herault, P. Lemarinier, J. Dongarra, DAGuE: A generic distributed DAG engine for High Performance Computing, Parallel Computing 38 (1–2) (2012) 37 – 51, extensions for Next-Generation Parallel Programming Models.

doi:https://doi.org/10.1016/j.parco.2011.10.003.
URL http://www.sciencedirect.com/science/article/pii/S0167819111001347

[14] J. Hofmann, G. Hager, G. Wellein, D. Fey, An analysis of core-and chip-level architectural features in four generations of intel server processors, in: International Supercomputing Conference, Springer, 2017, pp. 294–314.

[15] J. Reinders, J. Jeffers, High Performance Parallelism Pearls: Multicore and Many-core Programming Approaches, 1st Edition, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2014.

[16] S. Williams, A. Waterman, D. Patterson, Roofline: An insightful visual performance model for multicore architectures, Commun. ACM 52 (4) (2009) 65–76. doi:10.1145/1498765.1498785.
URL http://doi.acm.org/10.1145/1498765.1498785

[17] E. W. Larsen, J. E. Morel, Advances in discrete-ordinates methodology, in: Nuclear Computational Science, Springer, 2010, pp. 1–84.

[18] W. Kirschenmann, L. Plagne, A. Ponçot, S. Vialle, Parallel spn on multicore cpus and many-core gpus, Transport Theory and Statistical Physics 39 (2-4) (2011) 255–281.

[19] J. Lautard, D. Schneider, A. Baudron, Mixed dual methods for neutronic reactor core calculations in the cronos system, in: Proc. Int. Conf. Mathematics and Computation, Reactor Physics and Environmental Analysis of Nuclear Systems, 1999, pp. 27–30.

[20] P.-A. Raviart, J.-M. Thomas, A mixed finite element method for 2-nd order elliptic problems, in: Mathematical aspects of finite element methods, Springer, 1977, pp. 292–315.

[21] J.-C. Nédélec, A new family of mixed finite elements in r3, Numerische Mathematik 50 (1) (1986) 57–81.

[22] R. Slaybaugh, T. Evans, G. Davidson, P. Wilson, Multigrid in energy preconditioner for Krylov solvers, Journal of Computational Physics 242 (2013) 405–419.

[23] A. Toselli, O. Widlund, Domain Decomposition Methods – Algorithms and Theory, Springer Series in Computational Mathematics, Springer, 2000.

[24] M. Yavuz, E. W. Larsen, Iterative methods for solving x-y geometry SN problems on parallel architecture computers, Nuclear science and engineering 112 (1) (1992) 32–42.

[25] T. Courau, Specifications of a 3D PWR core benchmark for neutron transport, Tech. rep., Technical Note CR-128/2009/014 EDF-SA (2009).

[26] T. Courau, G. Sjoden, 3D neutron transport and HPC: A PWR full core calculation using pentran SN code and IBM BLUEGENE/P Computers, Progress in Nuclear Science and Technology 2 (2011) 628–633.

[27] G. Marleau, A. Hébert and R. Roy, A User's Guide for DRAGON 3.05, Tech. Rep. IGE-174 Rev.6, Institut de Génie Nucléaire, École Polytechnique de Montréal (2006).