Shared Memory Parallelism for 3D Cartesian Discrete Ordinates Solver

Salli Moustafa, Ivan Dutka Malen, Laurent Plagne, Angélique Ponçot, Pierre Ramet

Joint International Conference on Supercomputing in Nuclear Applications and Monte Carlo 2013 (SNA + MC 2013)
La Cité des Sciences et de l'Industrie, Paris, France, October 27–31, 2013

Shared Memory Parallelism for 3D Cartesian Discrete Ordinates Solver

Salli Moustafa[1], Ivan Dutka-Malen[1], Laurent Plagne[1], Angélique Ponçot[1], and Pierre Ramet[2]

[1]*EDF R&D 1, Av du Général de Gaulle F92141 CLAMART CEDEX France*
[2]*INRIA-University of Bordeaux, France*

This paper describes the design and the performance of DOMINO, a 3D Cartesian SN solver that implements two nested levels of parallelism (multicore+SIMD) on shared memory computation nodes. DOMINO is written in C++, a multi-paradigm programming language that enables the use of powerful and generic parallel programming tools such as Intel TBB and Eigen. These two libraries allow us to combine multi-thread parallelism with vector operations in an efficient and yet portable way. As a result, DOMINO can exploit the full power of modern multi-core processors and is able to tackle very large simulations, that usually require large HPC clusters, using a single computing node. For example, DOMINO solves a 3D full core PWR eigenvalue problem involving 26 energy groups, 288 angular directions (S16), 46×10^6 spatial cells and $\mathbf{1 \times 10^{12}}$ DoFs within 11 hours on a single 32-core SMP node. This represents a sustained performance of 235 GFlops and 40.74% of the SMP node peak perfomance for the DOMINO sweep implementation. The very high Flops/Watt ratio of DOMINO makes it a very interesting building block for a future many-nodes nuclear simulation tool.

KEYWORDS: Deterministic transport, SN, multicore processor, wavefront, TBB, SIMD

I. INTRODUCTION

Industrial Context

As part of its activity, EDF R&D is developing a new nuclear core simulation code named COCAGNE. This code relies on DIABOLO, a Simplified PN (SPN) method to compute the neutron flux inside the core for eigenvalue calculations.[(1)] In order to assess the accuracy of SPN results, a 3D Cartesian model of PWR nuclear cores has been designed and a reference neutron flux inside this core has been computed with the MCNP5 Monte Carlo transport code.[(2)] This kind of 3D whole core probabilistic evaluation of the flux is computationally very demanding. An efficient deterministic approach is therefore required to reduce the computation effort dedicated to reference simulations. In this paper we introduce DOMINO, a new shared memory parallel 3D Cartesian SN solver specialized for PWR core reactivity computations which is fully integrated in the COCAGNE system.

A First Step Toward Efficient Use of the HPC Clusters

Our aim is to build a massively parallel deterministic neutron transport solver such as Denovo[(3)] which is under active development at ORNL[1], allowing simulations on large distributed memory supercomputers. To achieve this goal, we first focus on designing an efficient implementation for one supercomputing node. Modern supercomputers are generally built upon hierarchical architectures that can be characterized by their number n_{nodes} of computing nodes, the number n_{sockets} of processors in each node, the number n_{cores} of cores in each processor, and finally the number n_{simd} of Single Instruction on Multiple Data (SIMD) units in each core. Although we plan to use these four levels of parallelism in the future, the DOMINO implementation is, at the present time, limited to the inner three levels (multi-socket computer nodes) which can be addressed with shared memory parallel programming paradigms. Obviously, an efficient implementation for supercomputing nodes is required to attain an efficient use of many-node HPC facilities. This hierarchical path for building massively parallel solvers on modern supercomputers is well illustrated in ref[(4)] where the authors give algorithmic and implementation strategies for optimizing the Sweep3D[(5)] kernel on the IBM Cell BE processor.

Proper Tools For Multi-Core Processors Programming

DOMINO is written in C++, a multi-paradigm programming language that enables the use powerful and generic parallel programming tools such as Intel TBB[(6)] and Eigen.[(7)] These two libraries allow us to combine multi-thread parallelism with vector operations in an efficient and yet portable way. As a result, DOMINO can exploit the full power of modern multi-core processors and is able to tackle very large simulations, that usually require large HPC clusters, using a single computing node. For example, DOMINO solves a 3D full core PWR k_{eff} problem involving 26 energy groups, 288 angular directions (S_{16}), 46×10^6 spatial cells and $\mathbf{1 \times 10^{12}}$ DoFs within 11 hours on a single 32-core SMP node. This represents a sustained performance of 235 GFlops and 40.74% of the SMP node peak performance for the DOMINO sweep implementation. The very high Flops/Watt ratio of DOMINO makes it a very interesting building block for a future distributed memory nuclear simulation tool.

[1]Oak Ridge National Laboratory

Paper Outline

In this paper, we describe the design of DOMINO and assess its performance. Section II briefly introduces the discrete ordinates numerical scheme and algorithms used in DOMINO. Section III describes the two nested levels of parallelism (multicore+SIMD) of DOMINO. Then section IV analyses the parallel scalability of the sweep kernel that represents the computationally most demanding part of code. In section V we present 3D full core PWR k_{eff} computations carried out with DOMINO. Both k_{eff} and flux accuracies are assessed by comparison with Monte-Carlo MCNP computations. Then we compare the performances of DOMINO with those of PENTRAN[8] and Denovo,[9] two massively parallel deterministic transport solvers. Section VI concludes the paper by describing our ongoing work devoted to extending DOMINO to distributed memory machines.

II. THE DOMINO COCAGNE SN SOLVER

COCAGNE is the name of the new nuclear core simulation system developed at EDF R&D. The COCAGNE system mostly relies on DIABOLO, an approximate solver based on the Simplified PN equations (SPN)[1,10] for its industrial simulations. Recently, an additional reference solver named DOMINO and based on a discrete ordinates method has been introduced into the COCAGNE system. These two solvers share a common interface and a similar design within the COCAGNE system. In particular, both DOMINO and DIABOLO are built upon `Legolas++`, a C++ library dedicated to solving multilevel blocked linear algebra systems.[11] This design similarity facilitates the interoperability between the two solvers. This section starts with a description of the numerical schemes used in DOMINO and then gives an overview of its algorithmic structure.

1. DOMINO Numerical Schemes

DOMINO (Discrete Ordinates Method In NeutrOnics) implements the Discrete Ordinates Method for the stationary neutron transport equation (1) in multi-dimensional Cartesian geometries. This equation depends on energy (E), the angular directions ($\vec{\Omega}$) and the spatial position ($\vec{r}$) of the particles, defining respectively the velocity, propagation direction and the spatial localization of particles.

$$\vec{\Omega}\cdot\overrightarrow{\nabla}\psi(\vec{r},E,\vec{\Omega})+\Sigma_t(\vec{r},E)\psi(\vec{r},E,\vec{\Omega}) = \int_0^\infty dE'\int_{S_2} d\vec{\Omega}'\Sigma_s(\vec{r},E'\to E,\vec{\Omega}'\cdot\vec{\Omega})\psi(\vec{r},E',\vec{\Omega}') + \frac{1}{k}\frac{\chi(E)}{4\pi}\int_0^\infty dE'\int_{S_2} d\vec{\Omega}'\nu\Sigma_f(\vec{r},E')\psi(\vec{r},E',\vec{\Omega}'). \quad (1)$$

This is an eigenvalue problem with k representing the multiplication of neutrons in successive fission generations; it can be rewritten as:

$$H\psi(\vec{r},E,\vec{\Omega}) = \frac{1}{k}F\psi(\vec{r},E,\vec{\Omega}), \quad (2)$$

where H and F represent transport and fission operators.

The problem (2) is solved using an inverse power algorithm, which leads to the computation of the neutron flux ψ and k, by iterating on the fission term:

$$H\psi^{n+1} = \frac{1}{k^n}F\psi^n, \quad k^{n+1} = k^n\frac{<F\psi^{n+1},F\psi^{n+1}>}{<F\psi^{n+1},F\psi^n>}.$$

Each power iteration solves a multi-group problem by using the Gauss-Seidel algorithm, which is the same as solving G one-group space-angle problems:

$$H_{gg}\psi_g(\vec{r},\vec{\Omega}) = -\sum_{g'\neq g} H_{gg'}\psi_{g'}(\vec{r},\vec{\Omega}) + S_g(\vec{r}), \quad (3)$$

where $S_g(\vec{r})$ represents the fission sources for the group g. We refer to these one-group problems as *monogroup* equations. In the S_N method, each of these equations is discretized on a finite number of angular directions defined by the quadrature formula used. DOMINO supports both *Level Symmetric* and *Gauss-Legendre* quadrature formulas, which lead respectively to $N(N+2)$ and $m\times n$ angular directions; N stands for the *Level Symmetric* quadrature formula order, m and n represent respectively the number of azimutal and polar directions used in the *Gauss-Legendre* quadrature formula. For both quadrature formulas, each angular direction is associated to a weight w_i for integral calculation on the unit sphere S_2. Hence, we have the following transport equations coupled by the scattering term:

$$L\psi^{n+1} = B - R\phi^n, \quad \phi^{n+1}(\vec{r}) = \sum_{j=1}^{ndir} w_j\psi^{n+1}(\vec{r},\vec{\Omega}_j),$$

where L is the spatial stream matrix, composed of the diagonal of H, $R = H - L$ is the spatial scattering matrix, and $B = S$ the source term; *ndir* is the total number of angular directions depending on the quadrature formula used.

The general structure of the DOMINO solver is summarized by Algorithm 1. The first level of iterations solves the eigenvalue problem by an inverse power algorithm with a Chebychev acceleration. Then a Gauss-Seidel algorithm is used to solve the multigroup problem coming from the discretization of the energetic variable. Inside each multigroup iteration, scattering iterations with a Diffusion Synthetic Acceleration,[12] solve the spatial problem by the well-known sweep algorithm.

The space discretization corresponds to the Diamond Differencing scheme (DD).[13] Up to now, only the order 0 has been implemented. In three dimension, the DD0 element has 1 moment and 3 mesh-edge incoming fluxes per cell as indicated in Figure 1.

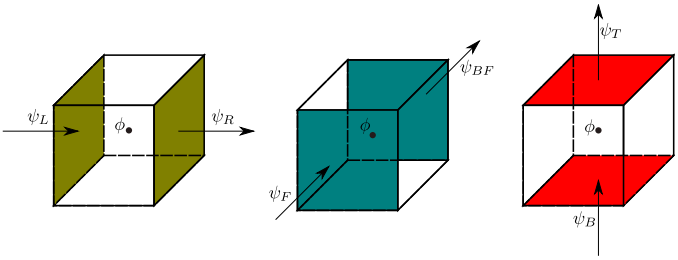

Figure 1: Degrees of freedom for the DD0 element

Algorithm 1: General Structure of DOMINO Solver

▷ *Initialization of external iterations*
$\phi = 1$;
$k = 1$;
$C = \sum_g \nu\Sigma_{f,g}.\phi_g$; ▷ *Fission source computation*

▷ *External iterations: inverse power algorithm*
while *Non convergence* **do**
  $\nu S = [\chi_g]\, C$
  ▷ *Multigroup iterations: Gauss-Seidel*
  while *Non convergence* **do**
    for $g \in [\![1, N_g]\!]$ **do**
      ▷ *External sources*
      $Q_{ext} = \nu S[g] + \sum_{g' \neq g} \Sigma_s^{g' \to g}.\phi_{g'}$;
      ▷ *Scattering iterations*
      while *Non convergence* **do**
        $Q = Q_{ext} + \Sigma_s^{g \to g}\phi_g$;
        $\vec{\Omega}_k.\vec{\nabla}\psi_k + \Sigma\psi_k = Q \quad \forall \vec{\Omega}_k \in S_N$;
        $\phi_g = \sum_{\Omega_k} \omega_k\psi_k$;
  $C = \sum_g \nu\Sigma_{f,g}.\phi_g$; ▷ *Fission sources update*
  $k = \frac{<C,C>}{<Cold,Cold>}$

Algorithm 2: The Sweep Operation

forall the $o \in$ `Octants` **do**
  forall the $c \in$ `Cells` **do**
    ▷ $c = (i, j, k)$
    forall the $d \in$ `Directions[o]` **do**
      ▷ $d = (\nu, \eta, \xi)$
      $\epsilon_x = \frac{2\nu}{\Delta x}$; $\epsilon_y = \frac{2\eta}{\Delta y}$; $\epsilon_z = \frac{2\xi}{\Delta z}$;
      $\psi[o][c][d] = \frac{\epsilon_x\psi_L + \epsilon_y\psi_B + \epsilon_z\psi_F + S}{\epsilon_x + \epsilon_y + \epsilon_z + \Sigma_t}$;
      $\psi_R[o][c][d] = 2\psi[o][c][d] - \psi_L[o][c][d]$;
      $\psi_T[o][c][d] = 2\psi[o][c][d] - \psi_B[o][c][d]$;
      $\psi_{BF}[o][c][d] = 2\psi[o][c][d] - \psi_F[o][c][d]$;
      $\phi[k][j][i] = \phi[k][j][i] + \psi[o][c][d] * \omega[d]$;

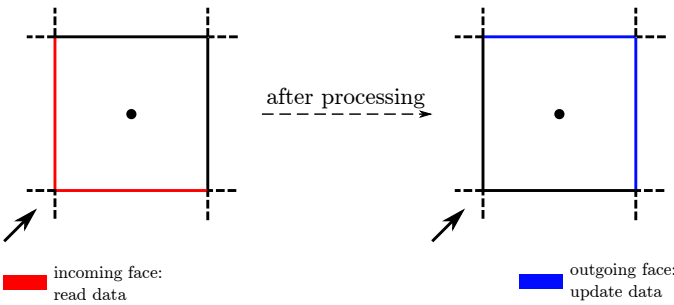

Figure 2: Data dependencies over one spatial cell in 2D

2. The Sweep Algorithm

Before going into details of the parallel implementation, Algorithm 2 describes the sweep operation which represents the most time-consuming operation inside the solver.

The aim of this algorithm is to solve the space-angle problem, by inverting the streaming operator of the monogroup transport equation: lines 0-0 of Algorithm 1. The volumic flux computation inside the cell c (line 0) 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 flux are then updated on lines 0, 0 and 0. At this point, we can observe that once a cell has been swept for a given octant, incoming angular flux are not longer used. This property allows 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 the key factor that allows obtaining a code that uses the full potential of modern multi-core architectures (see section 3). Finally we add the contribution of the volumic flux to the scalar flux on line 0, using the weight associated to the direction d, $\omega[d]$.

Let us count the total number of arithmetic operations per angular direction and per spatial cell in this sweep algorithm. We have 20 add/mult operations (the quantities $\frac{2}{\Delta u}$, $u = x, y, z$ can be computed once per spatial cell) each corresponding to 1 (flop) and 1 floating point division (line 0). The question how many flops to count for one division operation is a tricky one as the answer depends on the target architecture. In Ref,[14] authors show that the Sandy Bridge microarchitecture gives no performance gain for the division compared to the Nehalem microarchitecture, that is to say it costs exactly the same time to perform 8 packed single precision floating point divisions using AVX as 4 packed single precision floating point divisions using SSE. For our studies, we choose to count 5 flops per floating point division; this leads to a total of 25 flops for the sweep operation, determining the arithmetic intensity.

3. The Critical Arithmetic Intensity Issue

During the last decade, successive supercomputer node generations brought a regular and impressive improvement of their peak performance. Since their operating frequency remained almost unchanged, the multi-socket processors peak performance resulted mainly from their parallelism. In the context of transport simulations, the parallel computing power of a processor is proportional to the number n_{FPU} of its parallel floating points units (FPU): $n_{\text{FPU}} = n_{\text{sockets}} \times n_{\text{cores}} \times n_{\text{simd}}$.

Surprisingly enough, the computer bandwidth that measures the maximal data flow between the computer RAM and the FPUs did not increase as fast as the peak performance. The consequence of the broadening gap between the node bandwidth and its peak performance is to put a dramatic emphasis on the *arithmetic intensity* of the algorithms. If the *arithmetic*

intensity i of a given algorithm, defined by:

$$i = \frac{\text{Number of floating points operations}}{\text{Number of RAM access (Read+Write)}},$$

is lower than a critical value i_c, then its performance does not depend on the computational power of a target processor but mainly on the system memory bandwidth: the algorithm is then said to be *memory bound*. The critical arithmetic intensity of processors increases with each generation causing an ever larger fraction of algorithms to be *memory bound*. As a consequence of this processor evolution, the whole design of DOMINO is aimed at maximizing the *arithmetic intensity* in order to exploit the full parallel power of multicore processors.

III. TWO LEVELS OF PARALLELISM INSIDE THE DOMINO SOLVER

A shared memory supercomputing node can be seen as a multicore processor. To achieve a full utilization of its computing resources, we need to use both the multicore parallelism and the vector level parallelism also known as the Single Instruction Multiple Data model (SIMD). This latter level is a real bottleneck for performance issues, and its implementation constraints, involving hardware specifications of the chip, give guidelines to the algorithmic design of an application.

1. The Multicore Parallelism

The sweep operation is the most computationally intensive portion of DOMINO. For each incoming direction of the angular quadrature, the angular and volumic flux Degrees of Freedom (DoFs) of each spatial mesh cell must be updated. For simplicity's sake, we base the following sweep parallel description on a 2D example with a DD0 spatial discretization scheme. Let $\{c_{ij}\}$ be the $n_x \times n_y$ cells of a 2D Cartesian spatial mesh. Let us consider Ω_d an angular direction coming from the bottom left corner. Let $\{{}^d\psi^x_{ij}, {}^d\psi^y_{ij}, \phi_{ij}\}$ be the corresponding incoming angular and volumic flux DoFs. In the sweep operation one has to process each cell c_{ij} by updating the volumic ϕ_{ij} and the angular outgoing ${}^d\psi^x_{i+1\,j}$ and ${}^d\psi^y_{ij+1}$ DoFs that depend on the three ${}^d\psi^x_{ij}, {}^d\psi^y_{ij}$ and ϕ_{ij} input values. This implies an ordering constraint for the sweep operation: a cell c_{ij} can only be processed if the following two conditions are fulfilled:

$$c_{i-1,j} \text{ has been already processed or } i = 0,$$
$$c_{i,j-1} \text{ has been already processed or } j = 0.$$

Obviously c_{00} is the first cell that can be processed but the second can be either c_{10} or $c_{01}\ldots$ or both can be processed in parallel. We rely on the Intel TBB primitive `parallel_do`[15] that enables a dynamic scheduling of the parallel tasks to implement the sweep. This parallel function allows a pool of threads to execute the tasks from a task list which is dynamically updated. In the beginning, the task list contains the cell c_{00}. One of the running threads processes this cell and updates the task list to $\{c_{10}, c_{01}\}$ and so on. In order to reduce the overhead due to the thread scheduling, the cells are not processed individually but they are packed into groups of cells, called macrocells in the DOMINO implementation. The choice of the macrocell size depends of the available cache memory. In order to benefit from data reuse, data must be kept as long as possible in cache. If the macrocell size is too small, overheads due to thread scheduling deteriorate performances; the same consequence occurs when it is too big because of cache misses. For our studies, we perform an heuristic for the target architecture and determine the best size by trying several sizes. Figure 3 illustrates this parallel sweep strategy.

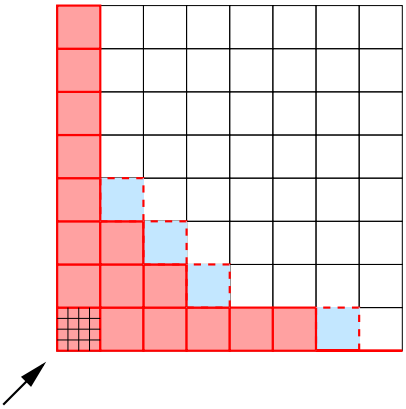

Figure 3: Wave front algorithm applied to sweep across the mesh macrocells. The incoming fluxes come from the bottom left corner. The processed macrocells are pink colored while the white macrocells are still waiting. Only blue dashed border macrocells are ready for processing and their respective indexes are stored in a task list from which a pool of threads concurrently pick their work item. This list is dynamically updated each time a new macrocell has been processed.

All the $n_{\text{sockets}} \times n_{\text{cores}}$ cores of the supercomputing node are used in this parallel wavefront algorithm.

When we are using the vacuum condition, with no external neutron source, we can extract more parallelism from the sweep operation. In fact, we can start the sweep for all 8 octants in parallel as indicated on Figure 4, provided the scalar flux reduction is performed by using mutexes, in order to avoid race conditions which can lead to wrong results. This corresponds to the parallelization of the most external `forall` loop of the Algorithm 2. By so doing, the potential parallelism is multiplied by 8 which can significantly improve the speed-up especially when a large number of cores are used to sweep small geometries. It is an ongoing work to extend this strategy to symmetric boundary conditions. Basically you start the sweep of an octant as soon as its first macrocell has been swept for the previous octant: it follows a pipeline of the sweep over the octants.

2. SIMD Parallelism

During each sweep, the angular directions that belong to the same octant are computed via SIMD instructions allowing to process simultaneously up to 8 simple precision floating point operations when using Intel AVX (Advanced Vector Extensions) instructions. In this section we recall the basic of the SIMD programming model with emphasis on Intel SSE and AVX; then we present how we handle these instructions in DOMINO.

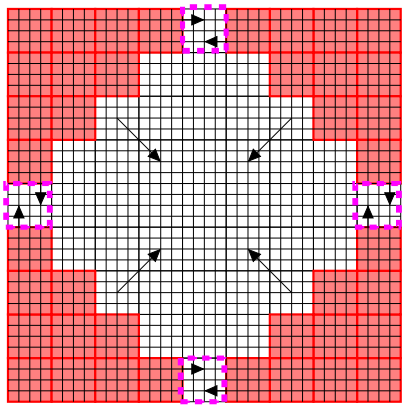

Figure 4: Parallelization of the sweep operation: all 4 octants are swept simultaneously. Pink colored macrocells are concurrently processed using mutexes: the first thread that starts the sweep of a given macrocell activates a lock that prevents other threads from starting to process the same macrocell and at the end, the lock is released so that another thread can start processing this macrocell.

SIMD Programming Model Vectorization is a computing model which consists in applying a single instruction to multiple data (SIMD). Most modern processors provide dedicated functional units that implement such instructions. Exploitation of these units can be done by relying on automatic vectorization by the compiler. However, while automatic vectorization is possible for simple loops, it is generally not the case for complex kernels. Meanwhile, by explicitly using assembly instructions or compiler intrinsics corresponding to the target architecture, one can make use of these vector units with the expense of some hardware constraints specific to vector instructions. In fact, vector instructions operate on packed data loaded inside specialized registers of fixed size. To perform fast load and store operations, data items need to be well aligned on cache boundary: 16 bytes for Intel SSE and 32 bytes for Intel AVX. Sometimes we have to resort to padding to satisfy this requirement. As an example, when loading 256 bits packet data with Intel AVX in a contiguous memory region of size $256 + 32 \times 3 = 352$ bits, we need to extend this region with $512 - 352 = 160$ bits at the memory allocation stage. Attention should be ported on padding as it increases global memory consumption and useless computations.

Angular Vectorization For the octant currently being swept, inside each spatial cell, we compute simultaneously several angular directions belonging to the octant (the `forall` loop on line 0 of Algorithm 2). SIMD instructions operate on packs of directions. The pack size, which depends on the precision and on the target architecture, is 8 in single precision and 4 in double precision on AVX enabled processors; on SSE enabled processors this pack size is divided by 2. Handling any angular quadrature order requires us to set up a padding system: for example, S_{16} *Level Symmetric* angular quadrature formula gives $16(16 + 2) = 288$ angular directions or $288/8 = 36$ directions per octant; when using single precision Intel AVX, as 36 is not a multiple of 8, we perform 40 angular directions processing per spatial cell corresponding to an efficiency of $36/40 = 0.9$. On the other hand, product quadrature formulas, such as *Gauss-Legendre*, give more flexibility to overcome this limitation as you can choose a combination of the number of azimutal and polar directions that give a total number of directions divisible by, say, 8.

C++ programming language is well suited to deal with this vectorization procedure. Instead of hard-coding compiler intrinsics in the code, the arithmetic operators needed are overloaded by their corresponding intrinsics; we rely on Eigen[2] for doing this job. Hence we are able to benefit from new vector instructions of the next generation of processors without having to modify the code, improving maintainability and readability of the code. Listing 1 shows a snapshot of the SIMD implementation of the sweep algorithm, which features some constructs of then Eigen library.

```
...
typedef Eigen::Array<RealType, blockSize, 1>
        BlockArray;
typedef Eigen::Map<BlockArray,Eigen::Aligned>
        BlockArrayView;
...
const int nblocks=directionPerOctant/blockSize;
for (int b=0; b<nblocks; b++){
  ...
  BlockArrayView psiX(&psiXd[dir]);
  psiOut+= epsX*psiX+epsY*psiY+epsZ*psiZ;
  BlockArray denom(sigmaIJK);
  denom+=epsX; denom+=epsY; denom+=epsZ;
  psiOut/=denom;
  phi+=psiOut*ConstBlockArrayView(&weight[dir]);
}
```

Listing 1: The SIMD implementation of the sweep algorithm: `BlockArray` is a type representing an array of packed data.

IV. Sweep Scalability and Efficiency

We analyze the performance of the sweep operation for a simple 2-group k_{eff} computation with $480 \times 480 \times 480$ spatial cells with different angular quadratures on the A and B computing nodes described in Table 1.

	Node A	Node B
Name	Intel X7560	Intel E5-2670
Architecture	Nehalem	Sandy Bridge
SIMD Inst.	SSE4	AVX
SIMD Units/Core	2	2
SIMD width (`float`)	4	8
SIMD width (`double`)	2	4
N_{core}	32	16
Frequency (GHz)	2.26	2.6
`float` Peak Performance (GFLOPS)	578	666
`double` Peak Performance (GFLOPS)	289	332

Table 1: Characteristics of two computing nodes. The theoretical peak performance figures are computed from Eq. (5).

Figure 5 summarizes the performance of the 2-level parallel implementation (multicore+SIMD) of DOMINO for different Level-Symmetric angular quadratures on Node A. Since

[2]Eigen is a C++ template library for linear algebra: matrices, vectors, numerical solvers, and related algorithms.

the arithmetic intensity of the computation increases with the number of direction, the sweep performance improves with the angular quadrature order. Because of their relatively low arithmetic intensities, the parallel speed-ups of the S_2 and S_4 computations saturate above 16 cores while it is not the case for the S_{8-16} quadratures. Note that S_2 and S_4 timings are exactly the same due to our padding strategy (see section 2). Here we use SIMD units that process at least 4 angular directions that belong to the same octant simultaneously. Since a given S_N Level-Symmetric defines $d_o = N(N+2)/8$ angular directions per octant, we obtain $d_o = 1$ (resp. $d_o = 3$) for the S_2 (resp. S_4) quadrature leading to $4-1=3$ (resp $4-3=1$) padded directions.

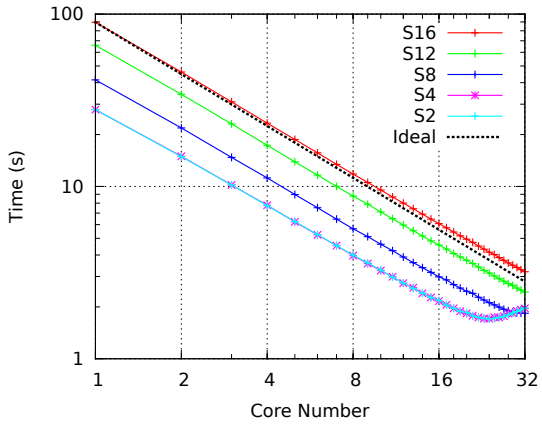

Figure 5: Time per sweep over all angular directions as a function of the core number on Node A (mesh: $480 \times 480 \times 480$).

In order to assess the sweep perfomance we compare the corresponding GFLOPS (Giga FLOating Point operations per Second) to the theoretical Peak performance of the computing Nodes. These two metrics are defined as:

- GFLOPS value is estimated by dividing the number of floating point operations by the completion time:

$$\text{GFLOPS} = \frac{25 \times N_{cells} \times N_{dir}}{\text{Time in nanoseconds}}, \tag{4}$$

  where the factor 25 is the total number of the floating point operations in the sweep operation, counted in section 2.

- The peak performance of a supercomputing node is evaluated by taking into account all the operations that all the available floating point arithmetic units can complete per clock cycle:

$$\text{Peak} = \text{ncores} \times \text{frequency} \times \text{SIMD width} \times \text{SIMD units}. \tag{5}$$

Table 2 summarizes the 32-core performance corresponding to Figure 5. While the S_2 and S_4 timings are identical (27.9s), the corresponding GFLOPS figures (11.2 and 33.8) differ since we only count *useful* angular directions (1 and 3) in the GFLOPS evaluation (Eq 4). In the S_{16} case ($N_{\text{dir}} = 288$), the sweep performance reaches 248.9 GFLOPS which corresponds to 43% of the Node A peak performance in single precision.

	N_{dir}	Seq. Time (s)	32-Core Time (s)	Speed Up	GFLOPS (Eq 4)	% Peak Perf.
S2	8×1	27.9	1.96	14.2	11.2	1.94%
S4	8×3	27.9	1.96	14.2	33.8	5.8%
S8	8×10	41.5	1.84	22.6	120.4	20.8%
S12	8×21	65.0	2.4	27.0	190.6	32.9%
S16	8×36	89.7	3.2	28.0	248.9	43.0%

Table 2: Time per sweep over all angular directions using 1 and 32 cores on Node A (mesh: $480 \times 480 \times 480$).

Impact of Parallelizing Over the Octants

Figure 6 shows the performance behavior that one can observe when allowing to compute all the octants in parallel as explained in the previous section. For the large spatial mesh, the parallelism over the octant is negligible. On the contrary, for the smaller mesh the additional amount of task-parallelism helps to maintain a speed-up almost linear up to 32 cores.

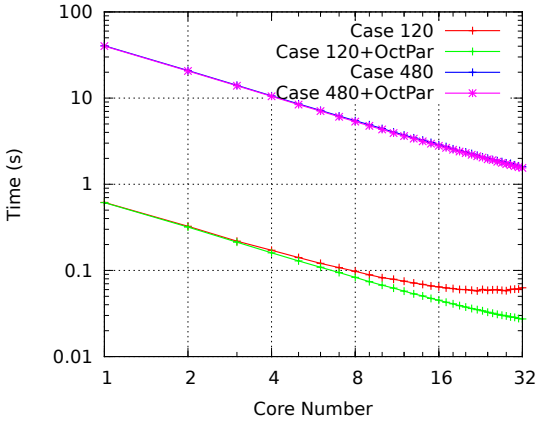

Figure 6: Performance impact of the octant parallelization (mesh: $120 \times 120 \times 120$ and $480 \times 480 \times 480$, using S_8 Level Symmetric quadrature on Node A)

Performance on Node B: From SSE To AVX Instructions

The same experiments were carried out using Node B. Here we use Gauss-Legendre quadratures. The corresponding result are summarized in Figure 7. This nodes accepts AVX instructions but is still compatible with SSE4 version of the sweep. Scalar, SSE and AVX versions of the sweep have been evaluated on Node B using 16 threads. The results are summarized in Table 3. One can see that, in this case, the performance does not increase much when switching from SSE to AVX. We are currently investigating this issue.

The performance of the sweep operation reaches 237.7 GFLOPS which corresponds to 35.71% of the peak performance of the supercomputing node B in single precision. As a general trend, the 16-core Sandy Bridge node leads to sweep performances that are on par with the 32-core Nehalem node.

In the next section we show that the good performance of the sweep kernel allows DOMINO to solve 3D PWR reactivity problems efficiently.

	Node A (32 cores)		Node B (16 core)	
	Time (s)	Speed-Up	Time (s)	Speed-Up
Scalar	21.46	×1	28.25	×1
SSE	5.62	×3.81	6.82	×4.14
AVX	-	-	5.95	×4.74

Table 3: Impact of SIMD parallelism for a 32×16 Gauss-Legendre quadrature sweep performance (mesh: $480 \times 480 \times 480$).

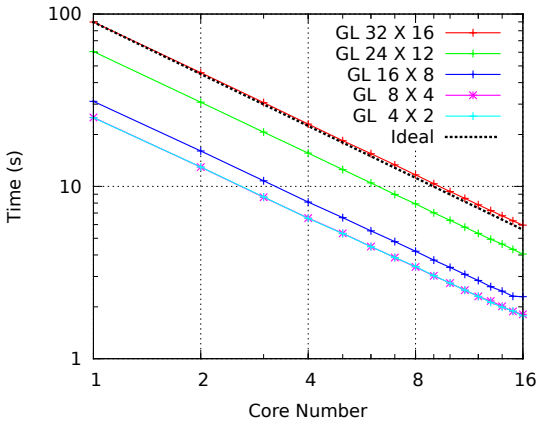

Figure 7: Time per sweep over all angular directions, using Gauss-Legendre quadrature formula, as a function of the core number on Node B (mesh: $480 \times 480 \times 480$)

V. 3D PWR Full Core Performance

The benchmark used for calculations is described in Ref.[2] It corresponds to a simplified 3D PWR first core loaded with 3 different types of fuel assemblies characterized by a specific 235 U enrichment (low, medium and highly enriched uranium). No inserted control device is considered in this core model. 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 in Figure 8 where the central assembly corresponds to the lowest enrichment, while the last row of fuel assemblies have the highest enrichment to flatten the neutron flux.

Each fuel assembly is a 17×17 array with a lattice pitch of 1.26 cm that contains 264 fuel pins and 25 water holes. The boundary condition associated with this benchmark problem is a pure leakage without any incoming angular flux. The associated nuclear data, an 8-group and a 26-group libraries, were obtained from a fuel assembly heterogeneous transport calculation performed with the cell code DRAGON.[16]

A 26-group computation has been carried out with DOMINO using a S_{16} Level-Symmetric angular quadrature. Table 4 summarizes the results of this k_{eff} computation. We assess the accuracy of DOMINO by comparing obtained neutron flux and k_{eff} to a reference obtained with MCNP5. To make easier the comparisons to MCNP results, the 3D 26-group fluxes are integrated over energy and space. Practically, a group collapsing is performed from 26 to 2 energy groups with a boundary fixed

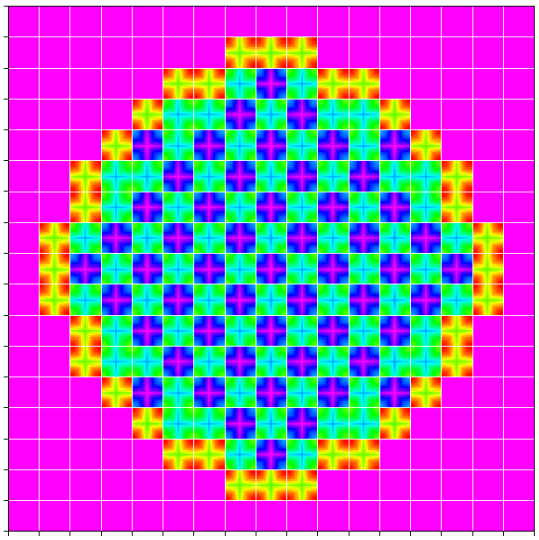

Figure 8: 2D-View of the PWR Core Model

at 0.625 eV, and a spatial integration over each pin-cell along z-axis is done. Comparisons between DOMINO and MCNP presented in Table 4 are computed in the form:

$$\Delta k_{eff} = \frac{k_{eff}^{DOMINO} - k_{eff}^{MCNP}}{k_{eff}^{MCNP}} \times 10^5 \text{ in pcm},$$

$$|\delta\phi_g| = \max_{\{p_i\}} \left| \frac{\phi_g^{DOMINO}(p_i) - \phi_g^{MCNP}(p_i)}{\phi_g^{MCNP}(p_i)} \right|,$$

where p_i stands for the pin cell index.

	26-group
N_{dir}	288
X Mesh Size	2×289
Y Mesh Size	2×289
Z Mesh Size	2×70
N_{DoF}	1.05×10^{12}
k_{eff}	1.008361
Δk_{eff} (pcm)	12
$\lvert\delta\phi_1\rvert$ (%)	0.69
$\lvert\delta\phi_2\rvert$ (%)	0.34
Wall-clock time (min)	658
Sweep Perf.	235 GFLOPS
(%) Peak Perf.	40.7%
DOMINO Perf.	111 GFLOPS
(%) Peak Perf.	19.2%

Table 4: 3D PWR k_{eff} computation with S_{16} angular quadrature on Node A (see Table 1). The k_{eff} tolerance is set to 1×10^{-5}. The flux and k_{eff} obtained with DOMINO are compared to a MCNP5 reference.

The 26-group k_{eff} computation completes in less than 11h, and corresponds to a sweep performance of 235 GFLOPS which is equivalent to 40.74% of the peak performance of the supercomputing node A. In order to evaluate the performance of

the complete DOMINO k_{eff} computation we divide the sweep operation number (Eq.4) by the complete wall-clock time[3]. It leads to an average performance of 111 GFLOPS.

Based on our 3D PWR model, the comparison with MCNP and the performance evaluation in terms of GFLOPS show that DOMINO is both accurate and computationally efficient. However it is important to compare DOMINO to other deterministic solvers in order to check if all the counted floating point operations correspond to a truly efficient algorithm.

1. DOMINO Compared With Pentran And Denovo

There exists several deterministic neutron transport solvers based on discrete ordinates method; we propose here a comparison of DOMINO with two solvers that can address 3D Cartesian meshes.

Pentran

PENTRAN[(8)] is a 3D discrete ordinates code that can automatically distribute the problem phase space among the angular (A), energy (E), and spatial variables (S) using a parallel memory and task allocation across a virtual processor array (VPA). One has the possibility to focus on one of the axes of the VPA by assigning a specific weight for this axis; the parallelism is implemented using a pure MPI programming model to exploit both shared and distributed memory supercomputers.

We compare S_8 DOMINO and PENTRAN k_{eff} computations for the 8-group version of our PWR Benchmark.[(8)] The results, obtained on Ivanoe, the EDF Intel Xeon based supercomputer[4], are summarized in Table 5. Note that different values for the stopping criterion associated to the flux (ϵ_ϕ) has been used for the different computations. The PENTRAN results have been extracted from ref.[(2)]

Using a single node DOMINO is faster (67 min) than PENTRAN running on 289 nodes (4752 min). Actually, PENTRAN is a general-purpose transport code and is not specially adapted to 3D PWR simulations. In particular, PENTRAN lacks an efficient acceleration scheme that is required to cope efficiently with highly diffusive media. In order to compare the performances of the parallel implementations of the codes, a non-accelerated DOMINO computation has been carried out (third column in Table 5). This non-accelerated DOMINO computation remains faster (573 min) than PENTRAN.

Several factors can explain this large performance difference. The PENTRAN code is built upon a single programming paradigm and does not explicitly address shared memory and vector issues that tend to become more and more important on modern processors. In addition, the spatial discretization scheme of DOMINO (DD0) is simpler than the DTW scheme used in PENTRAN and should consume less CPU cycles. At last it is difficult to evaluate the efficiency penalty arising from using a distributed architecture (289 nodes) compared to our shared memory approach. We are about to complete a distributed version of DOMINO that would allow us to conclude on this point.

[3]Here we neglect all the other flops of the rest of the code.
[4]Ivanoe was ranked 213 in the June 2013 Top 500 list.

	PENTRAN	DOMINO	DOMINO No Accel
	PWR 8g[(8)]	PWR 8g	PWR 8g
Computer	Ivanoe	Ivanoe	Ivanoe
XY Mesh	578×578	578×578	578×578
Z Mesh	168	168	168
N_{dir}	80	80	80
N_{group}	8	8	8
ϵ_ϕ tol	5×10^{-5}	1×10^{-5}	5×10^{-5}
N_{DoF} ($\times 10^9$)	35.9	35.9	35.9
k_{eff}	1.00867	1.00940	1.00940
δk_{eff} (pcm)	57	14	14
N_{cores}	3468	32	32
Wall-clock time (min)	4752	67	573
DoF/min ($\times 10^6$)	7.6	536	62.6
DoF/min/core ($\times 10^6$)	2.2×10^{-3}	16.7	1.96

Table 5: DOMINO-PENTRAN Comparison for a S_8 8-group 3D PWR k_{eff} computation.

Denovo

Denovo[(9)] is another deterministic neutron transport solver based on the discrete ordinates method, for radiation shielding and reactor physics applications under active development at ORNL. It implements a multilevel parallel decomposition on the phase space. This decomposition allows concurrency over energy in addition to space-angle parallelism. The spatial parallelism uses the KBA-based parallel decomposition; eigenvalue solvers implemented in Denovo are power iteration, as in DOMINO, and an Arnoldi solver.

Extracted from ref,[(9)] we compare S_{12} calculations performed with the Denovo solver on a 2-group version of the PWR-900 benchmark[(17)] with DOMINO. The Denovo computation was run on the Jaguar XT5 supercomputer (18688 compute nodes, each with dual 2.6 GHz AMD 6-core Istanbul processor). Please note that this machine differs from Ivanoe and that the performance comparison is not very precise. In addition, the axial meshes used in DOMINO and Denovo are slightly different (756 vs 700). Nonetheless, we hope that this comparison provides a correct trend on how DOMINO compares to Denovo. The results are summarized in Table 6.

Denovo is able to use efficiently a large number of nodes and thus solves the 2-group problem much faster than DOMINO. This illustrates our need to address many-node computations with DOMINO. As a positive point, we confirm that DOMINO makes an efficient use of the available 32 cores. Although the number of solved DoF per minute and per core (DoF/min/core) of DENOVO (1.88×10^6) is lower than the one of DOMINO (64.7×10^6), one should keep in mind that Amdahl's law imposes this metric to decrease with the number of cores. Note that this number (64.7×10^6) is significantly higher than what we measure for the 8-group computation (16.7×10^6). Indeed one needs more inner-group iterations when the group number

	DENOVO PWR 2g[3]	DOMINO PWR 2g
Computer	Jaguar	Ivanoe
XY Mesh	578×578	578×578
Z Mesh	700	756
N_{dir}	168	168
N_{group}	2	2
k_{eff} tol	1×10^{-3}	1×10^{-5}
N_{DoF} ($\times 10^9$)	78.6	84.9
N_{cores}	20400	32
Wall-clock time (min)	2.05	41
DoF/min ($\times 10^6$)	38330	2069
DoF/min/core ($\times 10^6$)	1.88	64.7

Table 6: DOMINO-Denovo Comparison for a S_{12} 2-group 3D PWR k_{eff} computation.

increases.

VI. Conclusion

The two-level (multi-core+SIMD) parallel implementation of the sweep algorithm in DOMINO has been described. It leads to a very efficient deterministic Cartesian transport solver that runs on shared-memory HPC nodes. Compared to Monte-Carlo solution (MCNP), DOMINO exhibits very accurate results both on 8 and 26 energy groups 3D PWR Benchmark. On comparable 3D nuclear core reactivity computations, DOMINO has shown to be very fast compared to PENTRAN and not very far from Denovo, a solver that addresses much larger HPC devices. We believe that the very high Flops/Watt ratio of DOMINO makes it a very promising building block for a future distributed memory nuclear simulation tool.

ACKNOWLEDGMENTS

The authors wish to thank Stanislas Odinot from Intel Corporation for providing access to different Intel devices. We also thank our reviewers who encouraged us to compare DOMINO with other transport codes.

References

1) W. Kirschenmann, L. Plagne, A. Ponçot, and S. Vialle, "Parallel SPN on Multi-Core CPUS and Many-Core GPUS," *Transport Theory and Statistical Physics*, **39**, *2-4*, 255-281 (2011).
2) T. Courau, L. Plagne, A. Ponçot, and G. Sjoden, "Hybrid Parallel Code Acceleration Methods in Full-Core Reactor Physics Calculations," *Proc. Physor 2012 - Advances in Reactor Physics*, 2010.
3) G. G. Davidson, T. M. Evans, J. J. Jarrell, and R. N. Slaybaugh, "Massively Parallel, Three-Dimensional Transport Solutions for the k-Eigenvalue Problem," *Proc. International Conference on Mathematics and Computational Methods Applied to Nuclear Science & Engineering (M&C 2011)*, Brazil, May, 2011.
4) F. Petrini et al., "Multicore surprises: Lessons learned from optimizing Sweep3D on the Cell Broadband Engine," *Proc. Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*, p. 1–10, IEEE, 2007.
5) K. R. Koch, R. S. Baker, and R. E. Alcouffe, "Solution of the first-order form of the 3-D discrete ordinates equation on a massively parallel processor," *Transactions of the American Nuclear Society*, **65**, *108*, 198–199 (1992).
6) Intel TBB: a C++ template library for task parallelism, "https://www.threadingbuildingblocks.org,".
7) Eigen: a C++ template library for linear algebra, "http://eigen.tuxfamily.org/index.php?title=Main_Page,".
8) T. Courau and 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**, 628–633 (2011).
9) T. M. Evans, G. G. Davidson, and R. N. Slaybaugh, "Three-dimensional full core power calculations for pressurized water reactors," *Proc. Journal of Physics: Conference Series, SciDAC*, volume 68, 2010.
10) T. Courau, S. Moustafa, L. Plagne, and A. Ponçot, "DOMINO: A Fast 3D Cartesian Discrete Ordiantes Solver for Reference PWR Simulations and SPN Validations," *Proc. International Conference on Mathematics and Computational Methods Applied to Nuclear Science & Engineering (M&C 2013)*, USA, May, 2013.
11) L. Plagne and A. Ponçot, "Generic Programming for Deterministic Neutron Transport Codes," *Proc. Mathematics & Computation, Supercomputing, Reactor Physics and Nuclear and Biological Applications*, Palais des Papes, Avignon, France, September, 2005.
12) E. Larsen, "Unconditionally stable diffusion synthetic acceleration methods for the slab geometry discrete ordinates equations," *Nuclear Science and Engineering* (1982).
13) N. Martin and A. Hébert, "A three-dimensional high-order diamond differencing discretization with a consistent acceleration scheme," *Annals of Nuclear Energy*, **36**, *11-12*, 1787 - 1796 (2009).
14) A. Vladimirov, "Arithmetics on Intel's Sandy Bridge and Westmere CPUs: not all FLOPS are created equal," *Colfax International* (2012).
15) A. Robison, M. Voss, and A. Kukanov, "Optimization via reflection on work stealing in TBB," *Proc. Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*, p. 1–8, IEEE, 2008.
16) G. Marleau, A. Hébert and R. Roy, "A User's Guide for DRAGON 3.05," IGE-174 Rev.6, Institut de Génie Nucléaire, École Polytechnique de Montréal (2006).
17) T. Courau, "Specifications of a 3D PWR Core Benchmark for Neutron Transport," Technical Note CR-128/2009/014 EDF-SA (2009).