

3D Cartesian Transport Sweep for Massively Parallel Architectures with PARSEC

Salli Moustafa, Mathieu Faverge, Laurent Plagne, Pierre Ramet

▶ **To cite this version:**

Salli Moustafa, Mathieu Faverge, Laurent Plagne, Pierre Ramet. 3D Cartesian Transport Sweep for Massively Parallel Architectures with PARSEC. IEEE International Parallel & Distributed Processing Symposium (IPDPS 2015), May 2015, Hyderabad, India. pp.581-590, 10.1109/IPDPS.2015.75 . hal-01078362

HAL Id: hal-01078362
https://inria.hal.science/hal-01078362

Submitted on 16 Dec 2014

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

3D Cartesian Transport Sweep for Massively Parallel Architectures with PARSEC

Salli Moustafa*†, Mathieu Faverge†§, Laurent Plagne*, and Pierre Ramet†‡
*EDF R&D 1, Av du Général de Gaulle F92141 CLAMART CEDEX France
†INRIA Bordeaux - Sud-Ouest, LaBRI, Talence, France
‡University of Bordeaux, Talence, France
§Bordeaux INP, Talence, France

***Abstract*—High-fidelity nuclear power plant core simulations require solving the Boltzmann transport equation. In discrete ordinates methods, the most computationally demanding operation of this equation is the sweep operation. Considering the evolution of computer architectures, we propose in this paper, as a first step toward heterogeneous distributed architectures, a *hybrid* parallel implementation of the sweep operation on top of the generic task-based runtime system: PARSEC. Such an implementation targets three nested levels of parallelism: message passing, multi-threading, and vectorization. A theoretical performance model was designed to validate the approach and help the tuning of the multiple parameters involved in such an approach. The proposed parallel implementation of the Sweep achieves a sustained performance of 6.1 Tflop/s, corresponding to 33.9% of the peak performance of the targeted supercomputer. This implementation compares favorably with state-of-art solvers such as PARTISN; and it can therefore serve as a building block for a massively parallel version of the neutron transport solver DOMINO developed at EDF.**

***Keywords*-3D Sweep; Distributed computing; Hybrid parallelism; Computational model; Task-Based Programing Model**

I. INTRODUCTION

The Boltzmann Transport Equation (BTE) is a fundamental tool for physics that describes the statistical evolution of large sets of particles. In the context of nuclear power plant simulations, the solutions of the BTE are used to predict the evolution of the neutron phase-space density within nuclear cores. A phase space density $f(\vec{r},\vec{p},t)$ is a function of 7 variables (3 for space, 3 for momentum and 1 for the time) that is proportional to the number of particles close to a position $\vec{r}$, a momentum $\vec{p}$ at a given time t. A discrete approximation of such a function on a mesh is not a trivial task. Considering a low resolution Cartesian mesh containing 100 points on each axis of the phase-space, one would have to store 100^7 floating point values representing 400 TeraBytes of computer memory. Considering the size of the BTE solution, one can easily infer that its solving procedure can rapidly exhaust the capability of the largest supercomputers. A great amount of work has been done during the last century to obtain approximate BTE solutions. These efforts can be separated in two main branches: the probabilistic methods (Monte-Carlo) and the deterministic ones. Monte-Carlo methods allow to avoid the phase-space mesh problems and have been the only approaches able to deal with 3D cases until the beginning of this century. Unfortunately, probabilistic methods allow to estimate the phase-space density with an accuracy that converges slowly with the number N of pseudo-particles ($\propto 1/\sqrt{N}$). Nowadays, modern supercomputers capabilities allow to consider deterministic methods in the 3D case and open a way toward unprecedented accuracy levels for BTE approximate solutions.

In the nuclear context, the kinetic energy $E = |\vec{p}|^2/2m$ of a neutron plays a dominant role for its interaction with matter and the time dependent phase-space density $f(\vec{r},\vec{p},t)$ is replaced by the time dependent neutron flux $\psi(\vec{r},E,\vec{\Omega},t) = v\ f(\vec{r},\vec{p},t)$ where $\vec{\Omega}$ stands for the particle momentum direction and v its velocity ($v = |\vec{p}|/m$). The discrete ordinates method (S_N) consists in considering only a finite set of angular components $\psi(\vec{r},E,\vec{\Omega}_i)$ that correspond to a finite set of angles $\vec{\Omega}_i$. The association of this set with appropriate weights ω_i allows to replace angular integration by discrete summation over angular flux components:

$$\int_{\vec{\Omega}} \psi(\vec{r},E,\vec{\Omega})d\vec{\Omega} \simeq \sum_{\vec{\Omega}_i} \omega_i \psi_{\vec{\Omega}_i}(\vec{r},E).$$

The so-called Sweep algorithm is a very important part of discrete ordinates BTE solvers and usually consumes the vast majority of the computing time. This sweep algorithm consists in performing a given task on all cells of a spatial mesh following an order which partly depends on the considered angular direction. This order obeys a frequently encountered dependency graph called wavefront that is studied as a classic parallel pattern in numerous references [1], [2]. As a consequence, the optimization of the sweep parallel implementation may have applications in fields that are not related to the particle transport.

The main contribution of this work is to present the parallel performance of a 3D Cartesian sweep implementation that is optimized for three differents levels of parallelism found in current supercomputers (multi-nodes, multi-core, multi-SIMD units), and an associated simplified performance model. Regarding parallel programming models, the *hybrid* approach (MPI+threads) should be more efficient than the classical *flat* approach (MPI only). In this paper, we focus on two specific aspects of the sweep implementation:

- the use of the task-based programming model via the PARSEC framework that allows to separate the dependency graph from the data distribution;
- the use of a theoretical performance model in order to assess the performance of our implementation, compute the optimal data distribution, and compare it with the Adams et al. *flat* model.

After recalling some related work in section II, we will present the sweep algorithm and its implementation on top of PARSEC in section III. Then follow descriptions of theoretical models of the considered algorithm in section IV, and performance measurements in section V. The summary and concluding remarks are given in section VI.

II. Related Work

The BTE resolution represents a significant portion in the main applications targeted by the DOE's Advanced Simulation Computing[1], formerly known as ASCI [3]. This initiative led to numerous research activities on the resolution of BTE. An important part of these researches concerns the development of efficient parallel sweep algorithms.

In the paper [4], Koch, Baker and Alcouffe have proposed the KBA algorithm which decomposes the 3D spatial grid onto a 2D process grid. This reference algorithm in the field of S_N Cartesian transport is also used in the neutron transport code DENOVO [5], developed at ORNL[2]. Moreover, the codes UNIC [6] and PENTRAN [7] partition the global problem onto a 3D virtual grid of $S \times A \times G$ processes, where S, A and G represent respectively the number of processes allocated for the spatial, angular and energy decompositions.

There are also several works on theoretical performance modeling of the sweep operation. Kerbyson et al. in the paper [8] presented the first performance model for transport sweeps on unstructured meshes. A similar work has been carried out by Plimpton et al. as described in the paper [9]. In the paper [10], Kumar et al. explore scheduling strategies for a generalized sweep algorithm. The algorithm they presented is applicable to more general cases than radiation transport problems as it uses no geometric information about the mesh. In the paper [11], the authors presented new algorithms for the sweep operation on unstructured meshes by designing algorithms that achieve overlap of communication by computations, as well as message buffering to reduce cost associated with the latency of parallel machines which are used.

For structured meshes, Hoisie et al. [12] extend the KBA algorithm parallel performance model by taking into account both communication and computations that fit the SWEEP3D[3] MPI based application. Chaussumier in [13] studied the impact of software pipelining in the overlap of communication by computations. Recently, Adams et al. [14] proposed a generalization of the KBA algorithm on structured meshes. They split the 3D spatial domain onto a 3D process grid; they have also presented scheduling algorithms proved optimal. In the paper [15], Azmy et al. give a method for finding the best way to decompose a given problem, of fixed size, between angular and spatial decompositions. They achieved this goal using performance models.

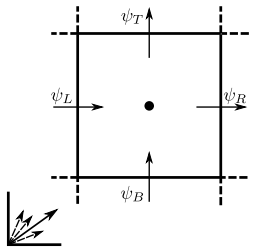
Except for the recent release of the PARTISN [16] neutron transport code developed at LANL, which features a parallel implementation of the sweep using both MPI and OpenMP, all the previous listed codes and theoretical models follow a classical one-level SPMD/MPI parallel programming model (or *flat* model). However, given that modern supercomputer architectures are becoming more and more heterogeneous (presence of accelerators inside computing nodes) and hybrid (interconnection of several nodes), it may be important to review classical parallel programming models as shown in the paper [17]. In a previous work [18], we have presented the DOMINO neutron transport solver designed for those modern architectures. We have especially showed that: 1) a good data locality dramatically improves arithmetic intensity of the sweep operation, and allows us to efficiently exploit SIMD units available inside current processors; 2) usage of the task-based programming model helped us to parallelize the sweep of DOMINO, by relying on INTEL TBB [19] library that addresses shared memory supercomputing nodes.

However, this approach does not go beyond shared memory systems. Regarding modern heterogeneous platforms, many initiatives have emerged in previous years to develop efficient runtime systems. Most of these runtime systems use a task-based paradigm to express concurrency and dependencies by employing a task dependency graph to represent the application to be executed. Without going into details, the main differences between these approaches are related to their representation of the graph of tasks, whether they manage data movements between computational resources, the extent to which they focus on task scheduling, and their capabilities to handle distributed platforms. Runtime systems such as Quark [20], STARPU [21], or STARSS [22] propose an insert task paradigm where a sequential code submits all computational tasks and the runtime discover the dependency graph at execution. This graph is a direct acyclic graph (DAG) where nodes are computational tasks and edges represent data flows and dependencies. This sequential discovery model of the DAG seems not to be the most appropriated to our regular grid solver. Indeed, the full graph submission loop might be a large overhead over the per node computational granularity targeted in this work. Another relevant framework is Charm++ [23] which is a parallel variant of the C++ language that provides sophisticated load balancing and a large number of communication

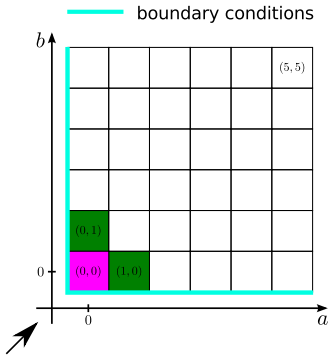
[1] http://www.lanl.gov/asc/

[2] Oak Ridge National Laboratory

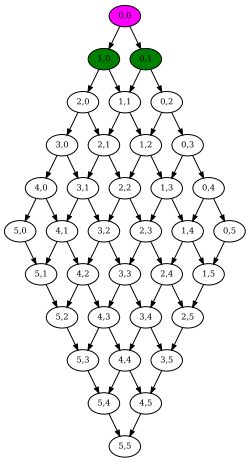
[3] http://wwwc3.lanl.gov/pal/software/sweep3d/sweep3d_readme.html



(a) 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).



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



(c) Task graph of the sweep operation over the domain.

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

optimization mechanisms. Intel CnC [24] and PARSEC [25] constructs an abridged representation of the DAG with the tasks and their dependencies with a structure agnostic to algorithmic subtleties, where all intrinsic knowledge about the complexity of the underlying algorithm is extricated, and the only constraints remaining are annotated dependencies between tasks [26].

The regularity of our problem does not justify a complex dynamic load balancing and thus leads us to consider the PARSEC framework. It is a framework intended to develop applications on distributed heterogeneous architectures. It features a generic data-flow runtime system, supporting a task-based implementation and targeting distributed hybrid systems. This framework relies on the dynamic scheduling of a direct acyclic graph (DAG) of the considered algorithm, whose nodes represent computational kernels (tasks), and edges data transfers between tasks. Thanks to an algebraic description of the task dependencies, the scheduling is completely asynchronous and fully distributed. Moreover, it takes into account user defined priorities and overlaps communications by computations. The programming model exploited in PARSEC allows to fully separate the major concerns in distributed algorithm: the kernels, the algorithm, and the data distribution. Here only one kernel is required to update angular fluxes in one quadrant (or octant in 3D) for one cell. It is vectorized over angular directions thanks to the C++ generic library Eigen [27] and has been presented in [18].

III. The sweep operation on top of PARSEC

The sweep operation is used to compute the 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 bottom-left quadrant, as shown on Figure 1a. Each cell has two incoming dependencies ψ_L and ψ_B for each angular direction. At the beginning of the computations, incoming fluxes on all left and bottom faces are known as indicated in Figure 1b. Hence, the cell $(0, 0)$ located at the bottom-left corner is the first to be processed as indicated in Figure 1c. The treatment of this cell allows to update outgoing fluxes ψ_R and ψ_T, that satisfy dependencies of the cells $(0, 1)$ and $(1, 0)$.

These dependencies on the cells processing define a sequential nature throughout the progression of the sweep operation: two adjacent cells belonging to successive diagonals cannot be processed simultaneously. Otherwise, all cells belonging to a same diagonal can be processed in parallel; moreover, 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. For some boundary conditions, processing of the four quadrants can be done concurrently. In the following, we assume that this is the case.

To use the PARSEC framework, the algorithm must be described as a DAG using the symbolic representation specific to PARSEC in a Job Data Flow (JDF) file. In this file, all tasks of the algorithm are defined by their execution space; their data placement or affinity; their input, output or in-out data-flows; and body. Each data-flow has incoming and outgoing edges connecting them to other tasks of DAG, or directly to memory accesses. The body specifies the computation of the task. This file is later compiled into a C code with a set of functions that will: submit the tasks to runtime system, check the dependencies, release the data to the following tasks, or execute the body.

Listing 1: JDF file of the 2D sweep for one quadrant

```
T(a, b)
// Execution space
a = 0 .. ncx
b = 0 .. ncy

// Parallel partitioning
: mcg(a, b)

// Parameters
RW X   <- (a != 0)   ? X T(a-1, b) : psi_x(b)
       -> (a != ncx) ? X T(a+1, b) : psi_x(b)
RW Y   <- (b != 0)   ? Y T(a, b-1) : psi_y(a)
       -> (b != ncy) ? Y T(a, b+1) : psi_y(a)

RW MCG <- mcg(a, b)
       -> mcg(a, b)

// Task Priority
; priority(a, b)

BODY
{
  computePhi ( MCG, X, Y, ... );
}
END
```

The sweep operation, either 2D or 3D, by its geometric structure, is a natural and simple candidate to this formalism. A simplified version of the 2D sweep operation, without boundary cases and one single quadrant, is given in the listing 1. Only one task `T`, described by its position in the grid (a, b), composes the algorithm. The execution space specifies that there are as many tasks as cells in the grid of size `ncx` $\times$ `ncy`. The parallel partitioning argument indicates the runtime to run the task `T(a,b)` on the node where the data `mcg`(a, b) is located. `mcg`(a, b) is here the structure holding data associated to a cell in the grid. Each task `T` has three in-out data dependencies:

- `X` and `Y` are aliases that correspond respectively to the ψ_L and ψ_B fluxes of the figure 1a. These two variables are labeled as read/write (`RW`) to indicate that the fluxes on the incoming faces are overwritten by those on the outgoing faces. `X` is the flux on the first direction. It comes from the previous task on this direction ($T(a-1, b)$), and is forwarded to the next task $T(a+1, b)$. `Y` is respectively the flux on the second direction. On the initial border of the domain, they are directly read from the main memory with the initial condition, and on the final border of the domain, they are written to the initial storage space. In our case of non reflecting nor periodic boundaries, they are directly initialized through a set of initial tasks at the beginning, and destroyed at the end.
- `MCG` represents a collection of several contiguous spatial cells on which the kernel will update the internal flux ψ in order to compute the outgoing `X` and `Y` fluxes. This object, called `MacroCell`, defines the granularity of a task. The larger is a `MacroCell`, the greater is the arithmetic intensity; but at the same time, the lower is the amount of available parallelism. Therefore, there exists an optimal `MacroCell` size that gives a maximum of performances.

The priority line allows the developer to provide a hint to the scheduler helping it to prioritize the more important tasks. Finally, The `BODY` section contains the computational task itself exploiting the parameters and flow aliases of the task to access the data.

Listing 2: Blocked data distribution of the `MacroCell` grid over the $P \times Q$ grid of processes.

```
// Rank of the process owning the MacroCell (a,b)
int rank_of ( Parameters & param, int a, int b ) {
  int lp = a / (param.ncx / param.P);
  int lq = b / (param.ncy / param.Q);

  return lq * param.P + lp;
}

// Address of the MacroCell object (a,b)
void * data_of ( Parameters & param, int a, int b,
                 DataType & mcgData ) {

  int aa = a % (param.ncx / param.P);
  int bb = b % (param.ncy / param.Q);

  return mcgData[bb][aa];
}
```

Once the algorithm is described in the PARSEC language, the runtime needs to know what is the data distribution and their location. A simple API must be implemented to provide this information to the scheduler. This is shown in the Listing 2 for the case of a 2D block distribution of `ncx` $\times$ `ncy` spatial grid over a $P \times Q$ grid of processes, as shown in Figure 2a. The `rank_of()` function is used by the scheduler to know on which process a data belongs to, but also for task mapping over the node as previously shown. Two tasks on different nodes can then be detected by the locality of the data they use, and communications are automatically generated: through direct accesses in shared memory, or MPI asynchronous communications in distributed memory. This separation between algorithm and data, specific to PARSEC, allows to quickly evaluate several data distributions for the same algorithm. The `data_of()` function returns the pointer to the `MacroCell` object when this one is local. Addresses of transferred objects are internally handled by the runtime.

IV. Theoretical Performance Models

In this section, we present a performance model of the *hybrid* 3D sweep corresponding to our implementation with PARSEC. This model is an extension of the *flat* model proposed by Adams et al. in [14]. We first introduce notations and remind basics of the *flat* model.

Let us consider the sweep over a 3D spatial domain of $N_x \times N_y \times N_z$ cells, with M angular directions per octant. The process coordinates inside the 3D grid of $N = PQR$ processes are noted (P, Q, R) (see Figure 2a). As mentioned

in previous section, cells are grouped by `MacroCell` of size $C_x \times C_y \times C_z$. We define:

- $S_x = N_x/C_x$, $S_y = N_y/C_y$, and $S_z = N_z/C_z$, the number of `MacroCell` along each axis of the spatial domain;
- $L_x = S_x/P$, $L_y = S_y/Q$, and $L_z = S_z/R$ the number of `MacroCell` along each axis on each process;
- $F = S_x + S_y + S_z - 2$, and $D = P + Q + R - 2$ respectively the number of diagonal planes (front) of `MacroCell` in the domain, and of processes per octant.

A. Flat *model*

The performance model of Adams et al. in [14] is a generalization of the KBA algorithm [4]. As opposite to the KBA algorithms which parallelizes the sweep operation over planes, it defines a *volumetric* decomposition of the spatial domain according to the x, y and z axis of the 3D domains.

Let G be the number of energy groups; A_m and A_g the numbers of angular directions and energy group per task; A_z the number of z-planes each process needs to compute before a communication step occurs. With $C_u = 1 | u \in (x, y, z)$, $N_k = L_z/A_z$ is the number of communication steps per process. Hence, each task carries the computation of angular flux for A_m directions, A_g energy groups and $L_x \times L_y \times A_z$ cells. Note that the number of cells per task can be modified through aggregation factors A_x and A_y defining number of cells per task according to x and y directions. According to Adams et al. model, the makespan of the the whole sweep is therefore:

$$T_{flat} = N_{stages}\ (T_{task} + T_{comm}), \tag{1}$$

where T_{task} is the cost of a computation step; T_{comm} the time needed by a process to communicate its outgoing angular flux to its neighboring processes after each computation step; and N_{stages} the total number of computation steps to perform the sweep and given by:

$$N_{stages} = 2\left(\left\lceil\frac{P}{2}\right\rceil - 1 + \left\lceil\frac{Q}{2}\right\rceil - 1 + N_k\left(\left\lceil\frac{R}{2}\right\rceil - 1\right)\right) + 8MGN_k/(A_m A_g). \tag{2}$$

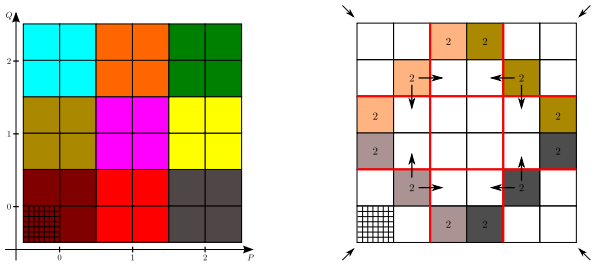
B. Hybrid *model*

The goal of this model is to extent the previous model to take into consideration the two levels of parallelism to validate the need for such an approach and provide us insight on the ideal process grid for a given problem and architecture. To achieve this goal, we need to know the new number of computation steps N_{stages}, and a new global communication time T^*_{comm}. Indeed, in this model not all stages induce a communication step anymore thanks to the shared memory accesses. Furthermore, in this model the constraint on the coupling between computation and communication steps is released, removing synchronizations. Thus, the makespan of the sweep operation in this model is given by:

$$T_{hybrid} = N_{stages}T_{task} + T^*_{comm}. \tag{3}$$

1) Number of computation steps: A computation stage is defined as the execution of a set of tasks in parallel. This set may be reduced to a singleton. The following notations are used:

- W_f is the list of all nodes having at least one task of the front f, for all quadrants/octants. A front is defined by all cells that can be executed in parallel. In our case, this corresponds to the f^{th} diagonal of each quadrant/octant.
- N_f^w is the number of tasks of the front f owned by the node w;
- N_f is the number of computation steps to process all tasks of the front f;
- N_{core} is the number of cores per node.



(a) Illustration of the blocked data distribution. `MacroCells` of similar colors belong to the same process.

(b) Simultaneous sweep of all quadrants. This shows that in this example M_2 is equal to 2.

Figure 2: Data distribution and communication pattern of the sweep for a 6×6 `MacroCell` grid, over a 3×3 process grid ($C_x = C_y = 7$; $S_x = S_y = 6$; $L_x = L_y = 2$; $F = 11$).

Figure 2 shows the evolution of the sweep for all the four quadrants of a 2D spatial grid of 6×6 `MacroCell`. The computer used in this example is a cluster comprising nine dual-core computing nodes. For this example, there are eight nodes involved to process the front $f = 2$:

$$W_2 = \{(0,0), (0,1), (0,2), (1,0), (1,2), (2,0), (2,1), (2,2)\},$$

where each node is represented by its coordinates in the processes grid. The largest number of ready tasks on a single node for this front is $\max_{w \in W_2}(N_2^w) = 2$. Therefore, the number of computation steps required to process all tasks belonging to the considered front is then $N_2 = \lceil 2/2 \rceil = 1$.

In general, the number of computation steps required to process all tasks of the front f is given by:

$$N_f = \left\lceil \max_{w \in W_f}(N_f^w)/N_{\text{core}} \right\rceil,$$

and by summing up over all fronts, we obtain an upper bound of the number of computation steps required to

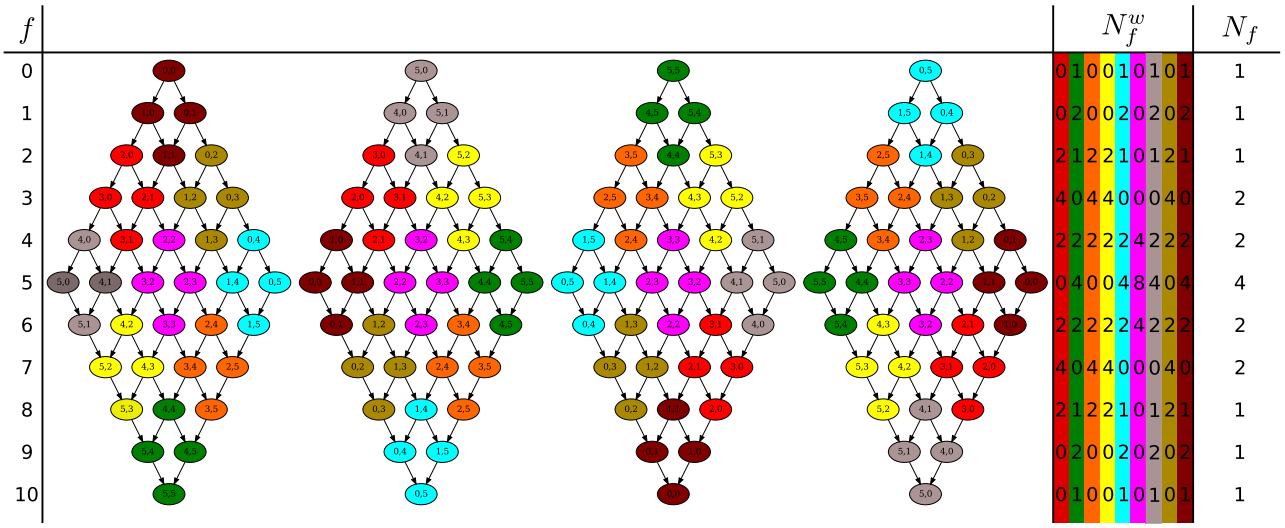


Figure 3: Verification of the computation steps formula for a concurrent sweep of all quadrants of a 6×6 `MacroCell` grid. We consider a process grid of 3×3 dual-core. Tasks of similar colors are on the same node and follow distribution on Figure 2.

process all stages of the whole sweep:

$$N_{stages} = \sum_{f=0}^{F-1} N_f. \quad (4)$$

This is an upper bound because this *hybrid* model is somewhat restrictive, as in practice one can start processing a front before the previous one is completed. Figure 3 shows a verification procedure of this formula, by counting the number of computation steps in the 2D case.

2) Number of communication steps: A communication step is the message of size of a `MacroCell` face, between two distinct nodes. For a given front f, let $M_f^w(u)|u \in (x, y, z)$ be the number of `MacroCell` faces sent by the node w towards the u direction. With PARSEC, all the communications from a single node are carried out sequentially by a dedicated thread. Thereby, we consider the sum of those communications and a part of them can be overlapped by computation threads. To take this effect into account, it is necessary to introduce an overlap rate k.

On a given front, the node performing the highest number of communications is necessarily a node having executed the highest number of tasks. Consequently, the number of communication steps, after executing all tasks of the front f, corresponds to the maximum number of communications issued by a node of this front. Hence the formula giving the communication time:

$$T^*_{comm} = (1-k) \sum_{f=0}^{F-1} \max_{w \in W_f} \left(\sum_{u \in (x,y,z)} M_f^w(u) \tau_u \right), \quad (5)$$

where τ_u $u \in (x, y, z)$ is the time to send a face of a `MacroCell` on the u direction. It is defined by the classic linear communication model [28]:

$$\tau_u = \alpha + s_u/\beta,$$

where s_u is the size of the message, and α, β are respectively the latency and bandwidth between two nodes.

3) Limitations of the hybrid *model:* The formula of computation steps (4) gives only an upper bound of the number of stages required to process the full sweep. Then, the execution performance could theoretically outperform this model.

In practice, the overlap rate should depend on the front f that is considered, since it depends on the number of computational cores that are busy, and the number of communication at each stage.

V. Performance Measurements and Validation of Models

Performance measurements were carried out on 64 nodes of the `IVANOE` supercomputer. Each node is equipped of two Intel Xeon X5670 making a total of 768 cores, and interconnected by a QDR Infiniband network. The theoretical peak performance of this cluster is of 18 Tflop/s in single precision. Experiments were carried out on the two test cases described in Table I). The `small` test case is configured to test our implementation when parallelism is insufficient for the total number of cores studied, increasing the importance of the scheduling and data distribution. The `big` test case is closer to reality cases and provides enough parallelism for the whole cluster. All experimental results represent the average performance obtained over one hundred run for each case.

		small	big
N_{dir}		288	288
Discretizations	$N_u \vert u \in (x,y,z)$	120	480
Granularity	$C_u \vert u \in (x,y,z)$	10	20
T_{task} (μs)		76.5	552.8
N_{tasks}		13824	110592
TFlops		0.012	0.796

Table I: Characteristics of the test cases used. Tflops is the number of floating point operations required for one complete sweep. We count 25 floating point operations per spatial cell per angular direction (see [18]).

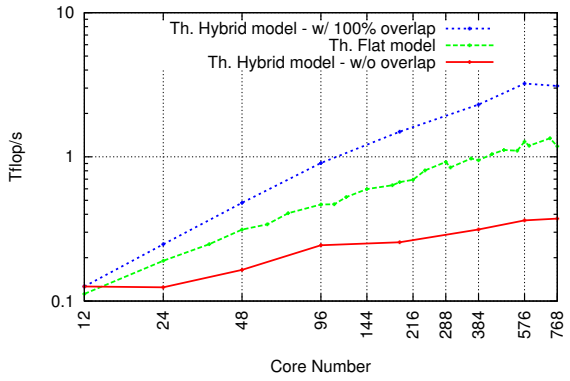


Figure 4: Comparison of *hybrid* and *flat* models using `small` test case.

A. Hybrid *versus* Flat

Figure 4 shows a comparison of the theoretical performance given by both models described in previous section on the *small* test case. One can observe that the estimated performance of the *flat* model using 768 cores is 3 times faster than for the *hybrid* model with no communication overlap. Indeed, the hybrid model induces a lot of small communications on the face of the `MacroCell` of size $C_x \times C_y \times C_z$, while the *flat* model has equivalent `MacroCell` of size $L_x \times L_y \times A_z$. On the other side, in the *hybrid* model we propose, one multi-threaded process can simultaneously perform computations and communications. Consequently, it is possible for the *hybrid* approach to outperform the *flat* one, provided that the runtime system manages to efficiently overlap communications by computations.

Figure 5 shows the experimental results of our implementation compared to the models to assess that they match the prediction model, and that it is possible to efficiently overlap the communications. In order to validate Adams et al. *flat* model on our architecture, we would have needed a hand-written MPI implementation of the sweep. We can meanwhile mimic its behavior with our PaRSEC implementation. To achieve this with PaRSEC, we used as many processes of one core as cores available to perform the experiment. In reality, PaRSEC framework runs an extra thread dedicated to communications per process. Then, for preventing the communication thread from disrupting the computation progress, we dedicated two cores per process: one for the computation, and one for the communication. Therefor, experiments were run only up to 384 computation cores. For the *hybrid* implementation, each process owns 11 computation cores, and 1 communication core. Experiments with 12 computation cores showed that the disruptions degraded the performance. The results of the Figure 5 confirm that our *hybrid* approach is more efficient than the *flat* one.

On a single node with 12 cores, we observe that the *hybrid* implementation is 1.2 times faster than the *flat* one. To justify this discrepancy, first note that although intra-node MPI communications can be done via shared memory, it remains that they conduct to a poor usage of caches and saturation of the memory buses. In contrast, these effects are much reduced when using threads in shared memory, and this may justify the observed difference. At 384 cores, the *hybrid* implementation is 1.7 times faster than the *flat* one. Indeed, as the number of communications increases with the number of cores, especially in the *flat* approach, the latter does not scale as much as the *hybrid* one.

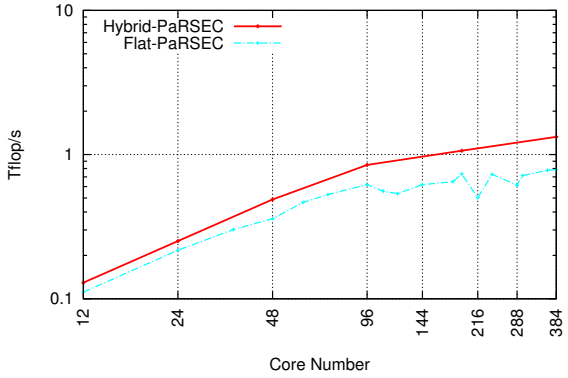
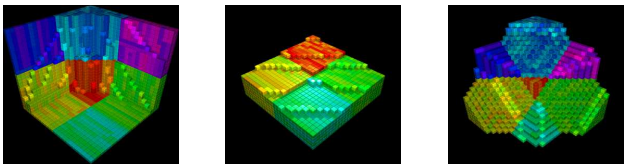


Figure 5: Comparison of the performance of *hybrid* and *flat* implementations, using the `small` test case.

B. Hybrid *model performance*

Note that PaRSEC implements several schedulers, which may impact the order of task processing and the performance. The default scheduler, Local Flat Queue (`LFQ`), favors the memory affinity by maximizing the memory reuse in following the dependencies of tasks as defined in the JDF file (Listing 1). Tasks released by a dependency are added to the local queue of the working thread. In the case of the 3D sweep, it leads to a prioritization of the sweep per columns of cells (Figure 6a). Such a scheduling is not an efficient policy for the sweep since it does not try to maximize the wave front, and thus the number of parallel tasks available. To tackle this problem, we considered the Priority Based Queue (`PBQ`) scheduler. This scheduler, similar to `LFQ`, adds ordering of the tasks in the local queue based



(a) LFQ (b) PBQ – PlaneZ (c) PBQ – Front

Figure 6: Snapshots showing the behaviour of various scheduling strategies on the sweep progress throughout the spatial domain. Data is distributed over a $2 \times 2 \times 2$ process grid, and threads belonging to the same node have similar colors.

on optional user defined priorities. We have studied two different priorities: PlaneZ and Front. The first favors tasks belonging to the same z-plane (Figure 6b), while the latter favors a progression by front (Figure 6c). The first one gives good results when the number of process over z, R, is equal to 2, while the second suits generic 3D distribution. We based our model on the front evolution, and experiments showed that the first two scheduling strategies could not match the model, then all following results are using the PBQ scheduler with the Front priorities.

For each number of cores, we want the performance model to give use the optimal data distribution (P, Q, R) that minimizes the equation (3). Figure 7 compares the experimental results against the prediction model with, or without overlap on the big test case. It shows that experiments are quite close to predictions, even if, as expected, the number of stages is overestimated by the model. Both gave $(12, 2, 2)$ as the best process grid. On the case $(24, 2, 1)$, overlap is no more possible and performance drops as predicted by the model without overlap.

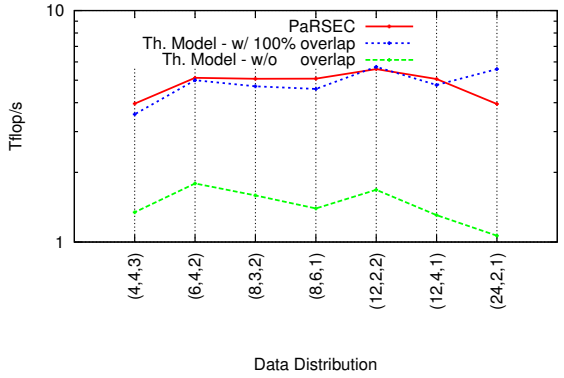


Figure 7: Sensitivity of the *hybrid* model to data distributions using 48 nodes (576 cores) and the big test case.

1) big test case: Figure 8 presents performance studies using the big test case over IVANOE supercomputer. The size of the domain provides enough tasks to keep all available cores working. Hence, one can observe a good scalability with a 68% parallel efficiency at 768 core, and an overlap ratio close to 100% that matches the model (error of 4% at 768 cores). This implementation reaches 6.1 Tflop/s, which corresponds to 33.9% of the peak performance of the supercomputer.

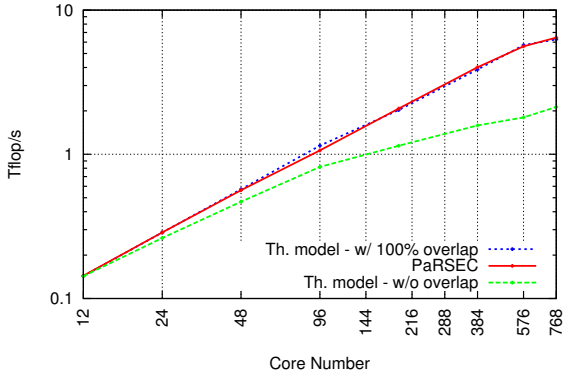


Figure 8: Strong scalability of the parallel *hybrid* PARSEC sweep on the big test case with respects to the prediction model.

2) small test case: Figure 9 compares the experimental results against the model on the small test case to illustrate the impact of lack of parallelism. In this case, communications become more important and assuming no overlap is possible leads to results far away from reality. In practice, on can observe that communication overlap is doable to a certain size where the parallelism becomes insufficient to hide it, 144 cores in this case. Adapting the average overlap ratio, k, of the equation (5) allows the model to match more correctly the results. Thus, results with 144 to 384 cores have an average overlap ratio of 90%, while the last two cases are closer to 85%. In fact, as mentioned in section IV, an extension to this model would be to automatically adapt this ratio on each front to better match the curves in every case.

The results presented in this section validated experimentally the prediction model presented in the previous section for the large test cases. This model also gives the trend of the curves for test cases with low parallelism but need to be manually tuned on the overlap ratio. Last, as for the *flat* implementation in [14], experiments showed that priorities are essential to the implementation in order to obtain similar performances on both the model and the experiments.

C. Comparison with SNAP/PARTISN

In this section, we compare the performance of the sweep implementation on top of PARSEC to the SNAP[4] code: one of the few sweep implementations freely available and similar to our code. SNAP is a proxy application for the

[4]http://www.lanl.gov/projects/feynman-center/technologies/software/snap-sn.php

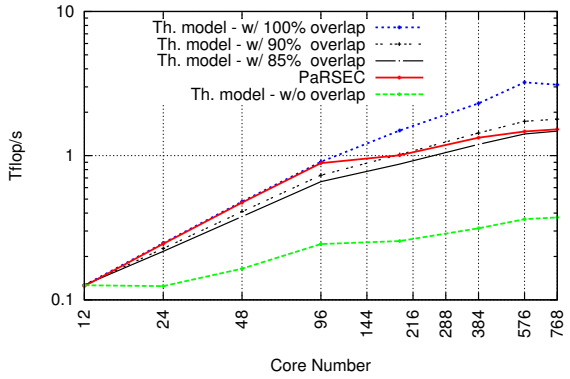


Figure 9: Strong scalability of the parallel *hybrid* PARSEC sweep on the `small` test case with respects to the prediction model.

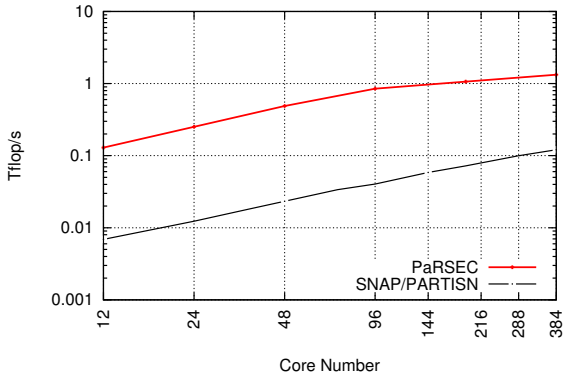


Figure 10: Comparison of the *hybrid* implementation of the sweep with the SNAP proxy application of the PARTISN code, using `small` test case.

neutron transport code PARTISN [16] developed at LANL. PARTISN solves the non-stationary Boltzmann equation over cartesian geometries using the S_N method. It comprises three levels of parallelism:

- the multi-group problem is solved by a *Block Jacobi* algorithm, parallelized in shared-memory with OPENMP;
- the spatial problem is solved in parallel with the KBA algorithm, by partitioning the spatial domain onto a 2D MPI process grid, and sweeping local cells with a second level of `OpenMP` threads;
- finally outgoing angular flux computations on the same cell are vectorized.

In the stationary case, SNAP performs the same number of floating point operations per spatial cell and per angular direction as our sweep model. However, some of its implementation details, such as computing the cell coordinates from the front index it belongs to, add more operations. On Figure 10, we present a performance comparison of our sweep implementation to the sweep portion SNAP. We consider the same number of floating point operations in both implementations, and the timing is based only on the sweep operation for the eight octants. On a single computing node having 12 cores, our *hybrid* implementation of the sweep is 18 times faster than the SNAP code. At 384 cores, the performance ratio between our implementation and the SNAP code drops to 10. Two things can explain such a difference: the SNAP code does not perform the octants concurrently as we do, and SNAP stores the intermediate ψ. This last change can divide by up to four the arithmetic intensity of the kernel. This confirms the preliminary results on shared memory systems with INTEL TBB presented in [18] against the DENOVO and PENTRAN code.

VI. CONCLUSION

High-fidelity nuclear power plant core simulations require solving the Boltzmann transport equation. In discrete ordinate methods, the most computationally demanding operation of this equation is the sweep operation. In this paper, we have presented a parallelization of this operation using the PARSEC task-based programming model framework. This framework allowed us to develop an hybrid MPI/thread application without the burden of the implementation, and with the objective of targeting heterogeneous distributed systems in the future. This implementation reaches 6.1 Tflop/s on 768 cores of a distributed memory supercomputer. This corresponds to 33.9% of its theoretical peak, and outperforms concurrent solver SNAP. We have also presented a theoretical performance model fitting this *hybrid* implementation that helped us to obtain such performance. Those results highlight the interest of the two-level approach (MPI+threads) against the classical one-level approach (MPI only). With the growing number of cores per node, this move is an essential step toward high efficiency solvers on exascale clusters. The next step will be to handle accelerators such as GPUs or INTEL XEON PHI in the sweep operation. PARSEC framework already supports those architectures and is able to offload part of the computations to them. Then, the objective is to provide efficient kernels for those architectures and scheduling strategies to help the runtime decide which tasks should be offloaded. On the other side, this implementation will be integrated in our neutron transport solver, namely DOMINO, in order to get a massively parallel solver.

ACKNOWLEDGMENT

The authors wish to thank Hugues Prisker and all the supercomputer support team at EDF who help us to carry out the scalability experiments.

REFERENCES

[1] S. MacDonald, J. Anvik, S. Bromling, J. Schaeffer, D. Szafron, and K. Tan, "From patterns to frameworks to parallel programs," *Parallel Computing*, vol. 28, no. 12, pp. 1663 – 1683, 2002. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167819102001904

[2] K. De Bosschere and M. Sawyer, *Applications, Tools and Techniques on the Road to Exascale Computing*. IOS Press, 2012, vol. 22.

[3] A. R. Larzelere *et al.*, "Creating simulation capabilities," *Computational Science & Engineering, IEEE*, vol. 5, no. 1, pp. 27–35, 1998.

[4] R. S. Baker and K. R. Koch, "An SN algorithm for the massively parallel CM200 computer," 1998.

[5] G. G. Davidson, T. M. Evans, J. J. Jarrell, and 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, May 2011.

[6] D. Kaushik, M. Smith, A. Wollaber, B. Smith, A. Siegel, and W. S. Yang, "Enabling high-fidelity neutron transport simulations on petascale architectures," in *High Performance Computing Networking, Storage and Analysis, Proceedings of the Conference on*. IEEE, 2009, pp. 1–12.

[7] 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*, vol. 2, pp. 628–633, 2011.

[8] D. J. Kerbyson, A. Hoisie, and S. D. Pautz, "Performance modeling of deterministic transport computations," in *Performance Analysis and Grid Computing*. Springer, 2004, pp. 21–39.

[9] S. J. Plimpton, B. Hendrickson, S. P. Burns, W. McLendon, and L. Rauchwerger, "Parallel sn sweeps on unstructured grids: Algorithms for prioritization, grid partitioning, and cycle detection," *Nuclear science and engineering*, vol. 150, no. 3, pp. 267–283, 2005.

[10] V. Anil Kumar, M. V. Marathe, S. Parthasarathy, A. Srinivasan, and S. Zust, "Provable algorithms for parallel generalized sweep scheduling," *Journal of Parallel and Distributed Computing*, vol. 66, no. 6, pp. 807–821, 2006.

[11] G. Colomer, R. Borrell, F. Trias, and I. Rodríguez, "Parallel algorithms for SN transport sweeps on unstructured meshes," *Journal of Computational Physics*, vol. 232, no. 1, pp. 118–135, 2013.

[12] A. Hoisie, O. Lubeck, and H. Wasserman, "Performance and scalability analysis of teraflop-scale parallel architectures using multidimensional wavefront applications," *International Journal of High Performance Computing Applications*, vol. 14, no. 4, pp. 330–346, 2000.

[13] F. Silber-Chaussumier, "Recouvrement des communications et des calculs, du matériel au logiciel," Ph.D. dissertation, Lyon, Ecole normale supérieure, 2002.

[14] M. P. Adams, M. L. Adams, W. D. Hawkins, T. Smith, L. Rauchwerger, N. M. Amato, T. S. Bailey, and R. D. Falgout, "Provably optimal parallel transport sweeps on regular grids," in *Proc. International Conference on Mathematics and Computational Methods Applied to Nuclear Science & Engineering, Idaho*, 2013.

[15] J. W. Fischer and Y. Azmy, "Comparison via parallel performance models of angular and spatial domain decompositions for solving neutral particle transport problems," *Progress in Nuclear Energy*, vol. 49, no. 1, pp. 37–60, 2007.

[16] R. S. L. A. N. L. Baker, *PARTISN on Advanced/Heterogeneous Processing Systems*, Feb 2013. [Online]. Available: http://www.osti.gov/scitech/servlets/purl/1063248

[17] I. Karlin, A. Bhatele, J. Keasler, B. L. Chamberlain, J. Cohen, Z. DeVito, R. Haque, D. Laney, E. Luke, F. Wang *et al.*, "Exploring traditional and emerging parallel programming models using a proxy application," in *Parallel & Distributed Processing, IEEE 27th International Symposium on*. IEEE, 2013, pp. 919–932.

[18] S. Moustafa, I. Dutka Malen, L. Plagne, A. Ponçot, and P. Ramet, "Shared Memory Parallelism for 3D Cartesian Discrete Ordinates Solver," *Annals of Nuclear Energy*, Sep. 2014. [Online]. Available: http://hal.inria.fr/hal-00986975

[19] Intel TBB: a C++ template library for task parallelism, "https://www.threadingbuildingblocks.org."

[20] A. Yarkhan, "Dynamic task execution on shared and distributed memory architectures," Ph.D. dissertation, University of Tennessee, 2012.

[21] C. Augonnet, S. Thibault, R. Namyst, and P.-A. Wacrenier, "StarPU: A Unified Platform for Task Scheduling on Heterogeneous Multicore Architectures," in *Proceedings of the 15th International Euro-Par Conference*. Springer, Aug. 2009. [Online]. Available: http://hal.inria.fr/inria-00384363

[22] R. M. Badia, J. R. Herrero, J. Labarta, J. M. Pérez, E. S. Quintana-Ortí, and G. Quintana-Ortí, "Parallelizing dense and banded linear algebra libraries using SMPSs," *Concurrency and Computation: Practice and Experience*, vol. 21, no. 18, pp. 2438–2456, 2009.

[23] L. V. Kalé and S. Krishnan, "CHARM++: A portable concurrent object oriented system based on C++," in *OOPSLA*, 1993, pp. 91–108.

[24] M. G. Burke, K. Knobe, R. Newton, and V. Sarkar, "The concurrent collections programming model," Rice University Houston, Tech. Rep., 2010.

[25] G. Bosilca, A. Bouteiller, A. Danalis, T. Hérault, P. Lemarinier, and J. Dongarra, "DAGuE: A generic distributed DAG engine for High Performance Computing," *Parallel Computing*, vol. 38, no. 1-2, 2012.

[26] M. Cosnard, E. Jeannot, and T. Yang, "Compact DAG representation and its symbolic scheduling," *Journal of Parallel and Distributed Computing*, vol. 64, no. 8, pp. 921–935, 2004.

[27] Eigen: a C++ template library for linear algebra, "http://eigen.tuxfamily.org/index.php?title=Main_Page."

[28] A. Clematis and A. Corana, "Modeling performance of heterogeneous parallel computing systems," *Parallel Computing*, vol. 25, no. 9, pp. 1131–1145, 1999.