E-Infrastructures

H2020-EINFRA-2015-1

EINFRA-5-2015: Centres of Excellence

for computing applications

# EoCoE

Energy oriented Center of Excellence

for computing applications

Grant Agreement Number: EINFRA-676629

## D1.1

## Triggered Application Support

## Project and Deliverable Information Sheet

| | | |
|---|---|---|
| EoCoE | Project Ref: | EINFRA-676629 |
| | Project Title: | Energy oriented Centre of Excellence |
| | Project Web Site: | http://www.eocoe.eu |
| | Deliverable ID: | D1.1 |
| | Lead Beneficiary: | CEA |
| | Contact: | Matthieu Haefele |
| | Contact's e-mail: | matthieu.haefele@maisondelasimulation.fr |
| | Deliverable Nature: | Report |
| | Dissemination Level: | PU* |
| | Contractual Date of Delivery: | M16 01/31/2017 |
| | Actual Date of Delivery: | M16 01/31/2017 |
| | EC Project Officer: | Carlos Morais-Pires |

* - The dissemination level are indicated as follows: PU – Public, CO – Confidential, only for members of the consortium (including the Commission Services) CL – Classified, as referred to in Commission Decision 2991/844/EC.

## Document Control Sheet

| | | |
|---|---|---|
| Document | Title : | Triggered Application Support |
| | ID : | D1.1 |
| | Available at: | http://www.eocoe.eu |
| | Software tool: | LaTeX |
| Authorship | Written by: | Haefele (MdlS, WP1), Tamain (CEA, WP5), Lanteri (Inria, WP1), Ould-Rouis (MdlS, WP1), Lührs (JSC, WP1), Latu (CEA, WP5) |
| | Contributors: | Gobé (Inria, WP1), Lanteri (Inria, WP1), Viquerat (Inria, WP1), Tamain (CEA, WP5), Latu (CEA, WP5), Giraud (Inria, WP1), Ould-Rouis (MdlS, WP1), Owen (BSC, WP2), Houzeaux (BSC, WP2), Haefele (MdlS, WP1), Lührs (JSC, WP1), Bruckmann (RWTH, WP4), Büsing (RWTH, WP4), Niederau (RWTH, WP4), Wylie (JSC, WP1), Gimenez (BSC, WP1), Bigot (MdlS, WP1) |
| | Reviewed by: | Haefele (MdlS), Gibbon (JSC) |

## Contents

## List of Figures

## List of Tables

## 1. Overview

The present deliverable D1.1 reports on the triggered application support, i.e. applications the consortium decided to give support to when writing the proposal. The objective at that time was to select a subset of the applications present in WP2-5, but at least one from each pillar, such that the different groups of expertise in the transversal basis WP1 get activated from the very beginning of the EoCoE project. This section gives a very brief overview of each six application support activities as well as their contribution to the general project impacts. The following sections present more details for each support activity.

Section 2 describes the work performed on NanoPV (WP3). The objective is to develop a new numerical scheme based on Discontinuous Galerkin methods that solves the same scientific problems as the original NanoPV code. This activity will have an impact on the long term because designing such numerical scheme on this type of equation is almost a topic of research in applied mathematics. So only preliminary results are available in this reports. More consistent results are expected to appear in D12 and D14 The purpose of this activity was to trigger the group dedicated to applied mathematics (Task1).

Section 3 describes the two activities related to Tokam3X (WP5). This code needs to solve very frequently pretty large linear systems that are not well conditioned. The integration of PastiX, a parallel direct solver part of the consortium, improved already the situation. Some first tests with an iterative solver coupled with a preconditioner have been made and runs on production cases are expected to take place early 2017. This activity could trigger the group dedicated to linear algebra at the very beginning of the project (Task2). At the same time, a High order Discontinuous Galerkin numerical scheme has been implemented in a reduced version of Tokam3X. The objective here is to have a more flexible representation of the computational domain that would allow to model precisely the wall geometry of the studied fusion devices. Similarly to the work on NanoPV, results on these type of activities are expected on the long term.

Section 4 presents the improvement of the input mechanism in Shemat (WP4). The original input procedure was ASCII based with a format that prevented the code to scale correctly with larger problem sizes. Large speedups could be achieved on large test cases but, even if this IO optimisation has been put back in the production version of SHEMAT, this version did not run in 2016. The implementation of an ad hoc HDF5 based format could improve drastically the situation and could trigger the group dedicated on IO (Task3).

Section 5, 6 and 7 describe code optimisation activities related respectively to Alya (WP2), Metalwalls (WP3) and Gysela (WP5). These activities triggered the groups dedicated to code optimisation (Task4) and to HPC tools (Task5). Thanks to performance evaluation tools, critical regions could be spot in these applications and specific code optimisations could be implemented in each of them. On Alya, the focus has been put on refactoring the matrix assembly routines. There are still on-going activities on ALya linked to solvers integration but nothing is reported in this document. On Metalwalls, the code vectorisation improved significantly the single core performance of the code. Finally, on Gysela issues related to the usage of SMT could be spot and fixed.

Table 1 shows that optimisation activities performed on only two of the six triggered applications could fulfil the total project objectives of impact 1.1 concerning the amount of CPU hours saved with only the CPU hours granted in 2016. This deliverable reports

| Code | Lead Institute | perf. gain (%) | CPUh saved in 2016 (MCPUh) | EoCoE tools usage |
|---|---|---|---|---|
| NanoPV | INRIA | 0 | 0 | 1 |
| Tokam3X | CEA | 0 | 0 | 1 |
| Shemat | RWTH Aachen | 1000 | 0 | 0 |
| Alya | BSC | 10 | 1 | 0 |
| Metalwalls | CEA | 250 | 42 | 0 |
| Gysela | CEA | 24 | 16 | 2 |
| Total | | | 59 | 4/6 |

Table 1: Contribution of Deliverable D1.1 to impacts 1.1, 1.2, 1.3 and 3.1

also that six scientific teams have received support from EoCoE's transversal basis and thus contributes to impact 1.3. Three codes already got performance improvement above 10%. The global project target defined by impact 3.1 is 10. The table also shows that four software packages developed or improved within EoCoE have been integrated in the six triggered codes. It contributes to impact 1.2.

As a conclusion and if a single fact should remain from the contribution of this deliverable, the transversal basis of EoCoE managed to **save 59 MCPUh in computing infrastructure time** only for year 2016. The whole project target of 50 MCPUh saved is already reached with only the support given to the triggered applications, keeping in mind that these optimised applications will also run in 2017 and 2018.

## 2. NanoPV

| | |
|---|---|
| Activity type | WP1 support |
| Contributors | Alexis Gobé (Inria), Stéphane Lanteri (Inria) and Jonathan Viquerat (Inria) |

### Context

The goal of this work is to exploit a finite element type solver from the DIOGENeS software suite [1] developed at Inria Sophia Antipolis-Méditerranée for the simulation of the light absorption in silicon thin-film tandem solar cells, using both large computational domains for random roughness as well as high resolution of nanosized features at reflecting metal contacts. This particular application setting will incur some specific mathematical and algorithmic adaptations of this electromagnetic wave propagation solver in view of being able to deal accuractely and efficiently with the multiscale features of the target problem. Performance will be assessed by comparison with experiment (for accuracy) and alternative solvers (for speed/system size). This work is carried on in close interaction with researchers from IEK-5 Photovoltaic, Forschungszentrum Julich, k (Urs Aeberhard and Alexander Markus Ermes) for an application framework relevant to work pacakage WP3.

### Introduction

The numerical modeling of light interaction with nanometer scale structures generally relies on the solution of the system of time-domain Maxwell equations, possibly taking into account an appropriate physical dispersion model, such as the Drude or Drude-Lorentz models, for characterizing the material properties of metallic nanostructures at optical frequencies [Mai07]. In the computational nanophotonics literature, a large number of studies are devoted to Finite Difference Time-Domain (FDTD) type discretization methods based on Yee's scheme [Yee66]. As a matter of fact, the FDTD [TH05] method is a widely used approach for solving the systems of partial differential equations modeling nanophotonic applications. In this method, the whole computational domain is discretized using a structured (cartesian) grid. However, in spite of its flexibility and second-order accuracy in a homogeneous medium, the Yee scheme suffers from serious accuracy degradation when used to model curved objects or when treating material interfaces. During the last twenty years, numerical methods formulated on unstructured meshes have drawn a lot of attention in computational electromagnetics with the aim of dealing with irregularly shaped structures and heterogeneous media. In particular, the Discontinuous-Galerkin Time-Domain (DGTD) method has met an increased interest because these methods somehow can be seen as a crossover between Finite Element Time-Domain (FETD) methods (their accuracy depends of the order of a chosen local polynomial basis upon which the solution is represented) and Finite Volume Time-Domain (FVTD) methods (the neighboring cells are connected by numerical fluxes). Thus, DGTD methods offer a wide range of flexibility in terms of geometry (since the use of unstructured and non-conforming meshes is naturally permitted) as well as local approximation order refinement strategies, which are of useful practical interest.

In this preliminary report, we report on our efforts aiming at the adaptation and application of a DGTD solver to the simulation of light trapping in a multi-layer solar

[1] http://www-sop.inria.fr/nachos/index.php/Software/DIOGENeS

cell with surface texture. Our aim is to demonstrate the possibility and benefits (in terms of acuracy and computational efficiency) of exploiting topography conforming geometrical models based on non-uniform discretization meshes.

### DGTD solver for nanoscale light/matter interactions

The basic ingredient of our DGTD solver is a discretization method which relies on a compact stencil high order interpolation of the electromagnetic field components within each cell of an unstructured tetrahedral mesh. This piecewise polynomial numerical approximation is allowed to be discontinuous from one mesh cell to another, and the consistency of the global approximation is obtained thanks to the definition of appropriate numerical traces of the fields on a face shared by two neighboring cells. Time integration is achieved using an explicit scheme and no global mass matrix inversion is required to advance the solution at each time step. Moreover, the resulting time-domain solver is particularly well adapted to parallel computing. For the numerical treatment of dispersion models in metals, we have adopted an Auxiliary Differential Equation (ADE) technique that has already proven its effectiveness in the FDTD framework. From the mathematical point of view, this amounts to solve the time-domain Maxwell equations coupled to a system of *ordinary differential equations*. The resulting ADE-based DGTD method is detailed in [Viq15].

#### Mathematical modeling

Towards the general aim of being able to consider concrete physical situations relevant to nanophotonics, One of the most important features to take into account in the numerical treatment is physical dispersion. In the presence of an exterior electric field, the electrons of a given medium do not reach their equilibrium position instantaneously, giving rise to an electric polarization that itself influences the electric displacement. In the case of a linear homogeneous isotropic non-dispersive medium, there is a linear relation between the applied electric field and the polarization. However, for some range of frequencies (depending on the considered material), the dispersion phenomenon cannot be neglected, and the relation between the polarization and the applied electric field becomes complex. In practice, this is modeled by a frequency-dependent complex permittivity. Several such models for the characterization of the permittivity exist; they are established by considering the equation of motion of the electrons in the medium and making some simplifications. There are mainly two ways of handling the frequency dependent permittivity in the framework of time-domain simulations, both starting from models defined in the frequency domain. A first approach is to introduce the polarization vector as an unknown field through an auxiliary differential equation which is derived from the original model in the frequency domain by means of an inverse Fourier transform. This is called the *Direct Method* or *Auxiliary Differential Equation* (ADE) formulation. Let us note that while the new equations can be easily added to any time-domain Maxwell solver, the resulting set of differential equations is tied to the particular choice of dispersive model and will never act as a black box able to deal with other models. In the second approach, the electric field displacement is computed from the electric field through a time convolution integral and a given expression of the permittivity which formulation can be changed independently of the rest of the solver. This is called the *Recursive Convolution Method* (RCM).

In [Viq15], an ADE formulation has been adopted. We first considered the case of

Drude and Drude-Lorentz models, and further extended the proposed ADE-based DGTD method to be able to deal with a generalized dispersion model in which we make use of a Padé approximant to fit an experimental permittivity function. The numerical treatment of such a generalized dispersion model is also presented in [Viq15]. We outline below the main characteristics of the proposed DGTD approach in the case of the Drude model. The latter is associated to a particularly simple theory that successfully accounts for the optical and thermal properties of some metals. In this model, the metal is considered as a static lattice of positive ions immersed in a free electrons gas. In the case of the Drude model, the frequency dependent permittivity is given by $\varepsilon_r(\omega) = \varepsilon_\infty - \frac{\omega_d^2}{\omega^2 + i\omega\gamma}$, where $\varepsilon_\infty$ represents the core electrons contribution to the relative permittivity $\varepsilon_r$, $\gamma$ is a coefficient linked to the electron/ion collisions representing the friction experienced by the electrons, and $\omega_d = \sqrt{\frac{n_e e^2}{m_e \varepsilon_0}}$ ($m_e$ is the electron mass, $e$ the electronic charge and $n_e$ the electronic density) is the plasma frequency of the electrons. Considering a constant permeability and a linear homogeneous and isotropic medium, one can write the Maxwell equations as

$$\mathrm{rot}(\mathbf{H}) = \frac{\partial \mathbf{D}}{\partial t}, \qquad \mathrm{rot}(\mathbf{E}) = -\frac{\partial \mathbf{B}}{\partial t}, \tag{1}$$

along with the constitutive relations $\mathbf{D} = \varepsilon_0 \varepsilon_\infty \mathbf{E} + \mathbf{P}$ and $\mathbf{B} = \mu_0 \mathbf{H}$, which can be combined to yield

$$\mathrm{rot}(\mathbf{E}) = -\mu_0 \frac{\partial \mathbf{H}}{\partial t}, \qquad \mathrm{rot}(\mathbf{H}) = \varepsilon_0 \varepsilon_\infty \frac{\partial \mathbf{E}}{\partial t} + \frac{\partial \mathbf{P}}{\partial t}. \tag{2}$$

In the frequential domain the polarization $\mathbf{P}$ is linked to the electric field through the relation $\hat{\mathbf{P}} = -\frac{\varepsilon_0 \omega_d^2}{\omega^2 + i\gamma_d \omega} \hat{\mathbf{E}}$, where $\hat{}$ denotes the Fourier transform of the time-domain field. An inverse Fourier transform gives

$$\frac{\partial^2 \mathbf{P}}{\partial t^2} + \gamma_d \frac{\partial \mathbf{P}}{\partial t} = \varepsilon_0 \omega_d^2 \mathbf{E}. \tag{3}$$

By defining the dipolar current vector $\mathbf{J}_p = \dfrac{\partial \mathbf{P}}{\partial t}$, (2)-(3) can be rewritten as

$$\begin{gathered} \mu_0 \frac{\partial \mathbf{H}}{\partial t} = -\nabla \times \mathbf{E} \quad , \quad \varepsilon_0 \varepsilon_\infty \frac{\partial \mathbf{E}}{\partial t} = \nabla \times \mathbf{H} - \mathbf{J}_p, \\ \frac{\partial \mathbf{J}_p}{\partial t} + \gamma_d \mathbf{J}_p = \varepsilon_0 \omega_d^2 \mathbf{E}. \end{gathered} \tag{4}$$

Recalling the definitions of the impedance and light velocity in vacuum, $Z_0 = \sqrt{\mu_0/\varepsilon_0}$ and $c_0 = 1/\sqrt{\varepsilon_0 \mu_0}$, and introducing the following substitutions, $\widetilde{\mathbf{H}} = Z_0 \mathbf{H}$, $\widetilde{\mathbf{E}} = \mathbf{E}$, $\widetilde{\mathbf{J}}_p = Z_0 \mathbf{J}_p$, $\widetilde{t} = c_0 t$, $\widetilde{\gamma}_d = \gamma_d / c_0$ and $\widetilde{\omega}_d^2 = \omega_d^2 / c_0^2$, it can be shown that system (4) can be normalized to yield

$$\begin{gathered} \frac{\partial \widetilde{\mathbf{H}}}{\partial t} = -\nabla \times \widetilde{\mathbf{E}} \quad , \quad \varepsilon_\infty \frac{\partial \widetilde{\mathbf{E}}}{\partial t} = \nabla \times \widetilde{\mathbf{H}} - \widetilde{\mathbf{J}}_p, \\ \frac{\partial \widetilde{\mathbf{J}}_p}{\partial t} + \gamma_d \widetilde{\mathbf{J}}_p = \widetilde{\omega}_d^2 \widetilde{\mathbf{E}}, \end{gathered} \tag{5}$$

knowing that $\mu_0 c_0 / Z_0 = 1$ and $\varepsilon_0 c_0 Z_0 = 1$. From now on, we omit the $\widetilde{X}$ notation for the normalized variables.

### DGTD method

The DGTD method can be considered as a finite element method where the continuity constraint at an element interface is released. While it keeps almost all the advantages of the finite element method (large spectrum of applications, complex geometries, etc.), the DGTD method has other nice properties:

- It is naturally adapted to a high order approximation of the unknown field. Moreover, one may increase the degree of the approximation in the whole mesh as easily as for spectral methods but, with a DGTD method, this can also be done locally i.e. at the mesh cell level.
- When the discretization in space is coupled to an explicit time integration method, the DG method leads to a block diagonal mass matrix independently of the form of the local approximation (e.g the type of polynomial interpolation). This is a striking difference with classical, continuous FETD formulations.
- It easily handles complex meshes. The grid may be a classical conforming finite element mesh, a non-conforming one or even a hybrid mesh made of various elements (tetrahedra, prisms, hexahedra, etc.). The DGTD method has been proven to work well with highly locally refined meshes. This property makes the DGTD method more suitable to the design of a $hp$-adaptive solution strategy (i.e. where the characteristic mesh size $h$ and the interpolation degree $p$ changes locally wherever it is needed).
- It is flexible with regards to the choice of the time stepping scheme. One may combine the discontinuous Galerkin spatial discretization with any global or local explicit time integration scheme, or even implicit, provided the resulting scheme is stable.
- It is naturally adapted to parallel computing. As long as an explicit time integration scheme is used, the DGTD method is easily parallelized. Moreover, the compact nature of method is in favor of high computation to communication ratio especially when the interpolation order is increased.

As in a classical finite element framework, a discontinuous Galerkin formulation relies on a weak form of the continuous problem at hand. However, due to the discontinuity of the global approximation, this variational formulation has to be defined at the element level. Then, a degree of freedom in the design of a discontinuous Galerkin scheme stems from the approximation of the boundary integral term resulting from the application of an integration by parts to the element-wise variational form. In the spirit of finite volume methods, the approximation of this boundary integral term calls for a numerical flux function which can be based on either a centered scheme or an upwind scheme, or a blend of these two schemes.

The DGTD method has been considered rather recently as an alternative to the widely used FDTD method for simulating nanoscale light/matter interaction problems [NKSB09]-[BKN11]-[MNHB11]-[NDB12]. The main features of the DGTD method studied in [Viq15] for the numerical solution of system (5) are the following:

- It is formulated on an unstructured tetrahedral mesh;
- It can deal with linear or curvilinear elements through a classical isoparametric mapping adapted to the DG framework [VS15];
- It relies on a high order nodal (Lagrange) interpolation of the components of $\mathbf{E}$, $\mathbf{H}$ and $\mathbf{J}_p$ within a tetrahedron;
- It offers the possibility of using a fully centered [FLLP05] or a fully upwind [HW02] scheme, as well as blend of the two schemes, for the evaluation of the numerical

traces (also referred as numerical fluxes) of the $\mathbf{E}$ and $\mathbf{H}$ fields at inter-element boundaries;

- It can be coupled to either a second-order or fourth-order leap-frog (LF) time integration scheme [FL10], or to a fourth-order low-storage Runge-Kutta (LSRK) time integration scheme [CK94];
- It can rely on a Silver-Muller absorbing boundary condition or a CFS-PML technique for the artificial truncation of the computational domain.

Starting from the continuous Maxwell-Drude equations (5), the system of semi-discrete DG equations associated to an element $\tau_i$ of the tetrahedral mesh writes

$$\left\{\begin{aligned} \mathbb{M}_i \frac{d\overline{\mathbf{H}}_i}{dt} &= -\mathbb{K}_i \times \overline{\mathbf{E}}_i + \sum_{k\in\mathcal{V}_i} \mathbb{S}_{ik}\left(\overline{\mathbf{E}}_\star \times \mathbf{n}_{ik}\right), \\ \mathbb{M}_i^{\varepsilon_\infty} \frac{d\overline{\mathbf{E}}_i}{dt} &= \mathbb{K}_i \times \overline{\mathbf{H}}_i - \sum_{k\in\mathcal{V}_i} \mathbb{S}_{ik}\left(\overline{\mathbf{H}}_\star \times \mathbf{n}_{ik}\right) - \mathbb{M}_i \overline{\mathbf{J}}_i, \\ \frac{d\overline{\mathbf{J}}_i}{dt} &= \omega_d^2 \overline{\mathbf{E}}_i - \gamma_d \overline{\mathbf{J}}_i. \end{aligned}\right. \tag{6}$$

In the above system of ODEs, $\overline{\mathbf{E}}_i$ is the vector of all the degrees of freedom of $\mathbf{E}$ in $\tau_i$ (with similar definitions for $\overline{\mathbf{H}}_i$ and $\overline{\mathbf{J}}_i$), $\mathbb{M}_i$ and $\mathbb{M}_i^{\varepsilon_\infty}$ are local mass matrices, $\mathbb{K}_i$ is a local pseudo-stiffness matrix, and $\mathbb{S}_{ik}$ is a local interface matrix. Moreover, $\overline{\mathbf{E}}_\star$ and $\overline{\mathbf{H}}_\star$ are numerical traces computed using an appropriate centered or upwind scheme. All these quantities are detailed in [Viq15].

## Construction of geometrical models

The first task that we had to deal with aimed at developing a dedicated preprocessing tool for building geometrical models that can be used by the DGTD solver. Such a geometrical model consists in a fully unstructured tetrahedral mesh, which is obtained using an appropriate mesh generation tool. We use the tetrahedral mesh generator from the MeshGems suite[2].

### Smoothing step

The initial layers data presented abrupt jumps in one direction of space (maybe resulting from the imaging process ?). We started by smoothing the layers with an in-house tool (see a comparison on figure 1).

### Building process

The building process is the following :

1. Build an initial closed surface mesh made of quadrilaterals from the smoothed layers data, using a specifically developed tool;
2. Transform the quadrangular faces to triangular faces to obatin a highly refined trangular surface mesh;

[2]http://www.meshgems.com/

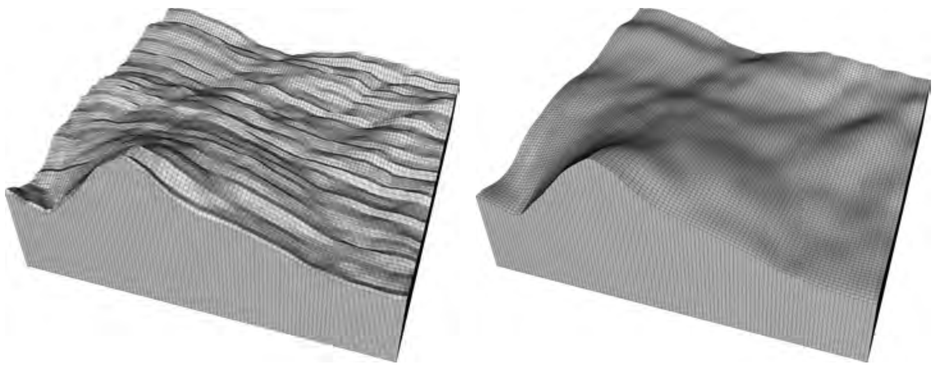

(a) Original surface. (b) Smoothed surface.

Figure 1: Smoothed and not-smoothed versions of the UcSI layer. This is a $100 \times 100$ subset of the full layer surface.

3. Build a pseudo-CAD model from the triangular surface mesh;
4. Use of a surfacic meshing tool to create a new, optimized triangular mesh from the CAD model;
5. Build a tetrahedral mesh from the optimized surface mesh.

Obviously, all these steps introduce discrepancies between the ideal model and the obtained mesh. It would be important to check, as a future step, how we can control and minimize the impact of the aforementioned steps (however, we must note here that it was not an easy task to obtain an exploitable mesh).

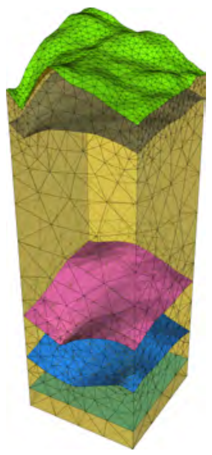
**Examples of meshes**

Partial views of generated meshes are shown on figure 2. One of these meshes corresponds to a $100 \times 100$ subset of the full model, which will be used for preliminary tests; the second and third meshes have been obtained for the full model.

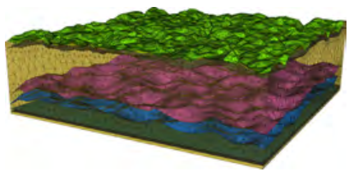
**Numerical and performance results**

We perform a strong scalability analysis by applying the proposed DGTD solver with a tetrahedral mesh of the full solar cell model consisting of 55,245 vertices and 300,426 elements. This performance analysis is conducted on the Occigen PRACE system hosted by CINES in Montpellier. Each node of this system consists of two Intel Haswell E5-2690@2.6 GHz CPU each with 12 cores. The parallel speedup is evaluated for 1000 time iterations of the DGTD-$\mathbb{P}_k$ solver using a fourth-order low-storage Runge-Kutta time scheme. Here, $\mathbb{P}_k$ denotes the set of Lagrange polynomials of order less or equal to $k$. In other words, DGTD-$\mathbb{P}_k$ refers to the case where the interpolation of the components of the $(\mathbf{E}, \mathbf{H}, \mathbf{J}_p)$ fields relies on a $k$-order polynomial within each element of the mesh. For this preliminary study, the interpolation order is uniform (i.e. is the same for all the elements of the mesh) but the DG framework allows to easily adapt locally the interpolation order [VL16]. Performance results are presented in Tab. 2 where:

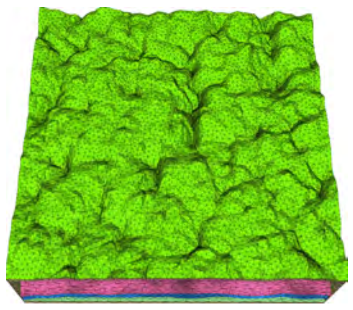
- "CPU min" et "CPU max" respectively denote the minimum and maximum values



(a) 100×100 subset of the full model.



(b) Full model.



(c) Full model.

Figure 2: Examples of the obtained meshes.

of the CPU time recorded for each process;

- "Elapsed" is the elapsed time, which is used for the evalaution of the parallel speedup;
- The parallel speedup is here evaluated relatively to the elapsed time on 24 cores.

The observed superlinear values of the parallel speedup are due to the fact that the tetrahedral mesh used for the simulations is too coarse. Then, cache effects impact the single core performances. We note that as the interpolation degree is increased a more reasonable behaviour is observed because the size of the problem to be simulated also increases although the mesh is unchanged. In all cases, the scalability is perfect (at least up to the maximum number of processing cores considered in our simulations). The difference between the minimum and maximum values of the CPU time is a result of the computational load imbalance, the later being directly related to the quality of the partitioning of the mesh (wee use the MeTiS[3] partitioning tool).

| Solver | # cores | CPU min | CPU max | Elapsed | Speedup |
|---|---|---|---|---|---|
| DGTD-$\mathbb{P}_1$ | 24 | 538 sec | 572 sec | 578 sec | 1.0 (-) |
| - | 48 | 249 sec | 263 sec | 271 sec | 2.1 (2.0) |
| - | 96 | 107 sec | 114 sec | 118 sec | 4.9 (4.0) |
| - | 192 | 45 sec | 49 sec | 47 sec | 12.3 (8.0) |
| - | 384 | 16 sec | 18 sec | 21 sec | 27.5 (16.0) |
| DGTD-$\mathbb{P}_2$ | 24 | 1395 sec | 1452 sec | 1471 sec | 1.0 (-) |
| - | 48 | 661 sec | 704 sec | 716 sec | 2.0 (2.0) |
| - | 96 | 312 sec | 331 sec | 340 sec | 4.3 (4.0) |
| - | 192 | 131 sec | 147 sec | 165 sec | 8.9 (8.0) |
| - | 384 | 59 sec | 67 sec | 74 sec | 19.8 (16.0) |
| DGTD-$\mathbb{P}_3$ | 24 | 2798 sec | 2948 sec | 2963 sec | 1.0 (-) |
| - | 48 | 1352 sec | 1456 sec | 1474 sec | 2.0 (2.0) |
| - | 96 | 663 sec | 713 sec | 727 sec | 4.1 (4.0) |
| - | 192 | 323 sec | 348 sec | 358 sec | 8.3 (8.0) |
| - | 384 | 142 sec | 161 sec | 173 sec | 17.1 (16.0) |

Table 2: Strong scalability analysis of the DGTD-$\mathbb{P}_k$ solver on the Occigen system. Mesh M1 (full model) with 55,245 vertices and 300,426 elements. Timings for 1000 time iterations. Execution mode: 1 MPI process per core.

## Conclusion and future works

In this preliminary report, we did not include physical results for simulations with the full model. Currently, on the lateral faces of the computational model we apply an absorbing boundary condition, similarly to what is done on the bottom and top surfaces. The incident plane wave is injected in the domain through the absorbing boundary (in the DG framework, this can be achieved in a simple way by adpting the numerical flux at the corresponding absorbing boundaries). However, we use the same definition of the incident plane wave everywhere, which is assumed to be a plane wave in vacuum. Moreover, we solve for the total field rather than the scattered field. Clearly, the definition of the incident wave that we use is not appropriate. In order to cure this problem we plan to investigate two possible approaches:

[3]http://glaros.dtc.umn.edu/gkhome/metis/metis/overview

- Solve for the scattered field at the expense of the evaluation of a volumic source term everywhere (this can be made clear by rewriting system (5) for the scattered fields);
- Adopt periodic boundary conditions on the lateral faces. In our setting, this will require to adapt the geometrical model and the surfacic mesh of the lateral faces so that we enforce a perfect matching of the periodic triangular faces on the two lateral surfaces along each spatial direction.

We will then be able to perform more realstic simulations involving higher resolution geometrical models with several million mesh elements, and compare the physical results with experimental data one one hand, and simulated data obatiend with a FDTD solver that is exploited by the Theoretical Nanostructure Photovoltaics group at IEK-5 Photovoltaic.

## 3. Tokam3X

| | |
|---|---|
| Activity type | Consultancy or WP1 support |
| Contributors | P. Tamain (WP5), G. Latu (WP5), L. Giraud (WP1) |

Since the start of the EoCoE project, two axes of development have been explored to improve the performances of the TOKAM3X tokamak edge plasma turbulence code. On the one hand, exhaustive studies have been carried out in sight of solving the main numerical bottle neck of the existing production version of TOKAM3X, ie, the 3D implicit solver for the vorticity operator; on the other hand, preliminary studies have started and produced first results concerning a possible porting of the whole numerical scheme towards High order Discontinuous Galerkin (HDG) methods. Both are expected to contribute to getting the code closer to being able to tackle full scale ITER simulations. Let us now detail each of these axes. Concerning the 3D implicit solver for vorticity operator, studies have focused on the possibility of replacing the current direct solver by a more scalable iterative method. A systematic study of preconditioner – iterative scheme options has been conducted (Fig. 1) and led to the identification of a promising preconditioner – iterative solver couple (the so-called "P9" preconditioner associated with a GMRES solver) showing a good balance of convergence rate and memory consumption. In collaboration with INRIA Bordeaux, the PASTIX solver used in the TOKAM3X code has been modified to include an implementation of a parallel GMRES solver with flexible pre-conditioner. Tests in large scale simulations are planned for the beginning of 2017. Note also that this study highlighted that the condition number of the matrix to invert could get extremely large (1014 or more) in some conditions of physical interest. In such cases, given the numerical precision of floating point operations, the solution found by iterative or even direct solvers can be very different from the analytical solution of the problem. To circumvent this issue, we have modified the equations of the physical model by adding electron inertia which was neglected before. This modification prevents the condition number of the matrix from diverging (gain of 103 or more on the condition number), making the system easier to solve with iterative methods and the solution more precise.

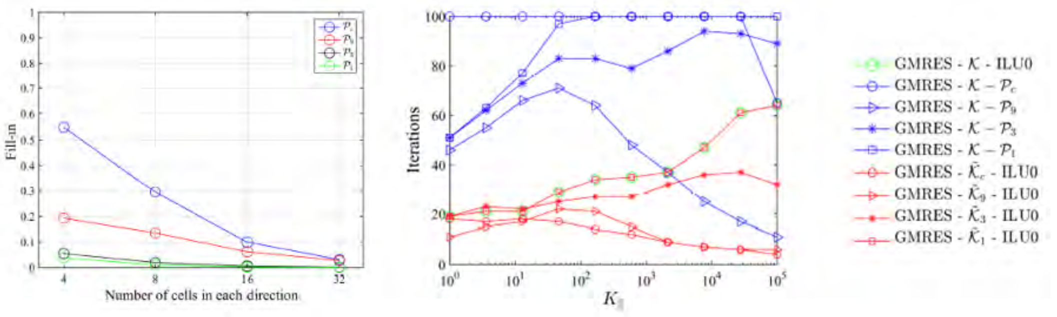


Figure 3: Left: memory consumption of the LU factorization of 4 different preconditioners relative to the memory consumption of the LU factorization of the total matrix. Right: number of iterations before convergence (100 = not converging) as a function of the parallel conductivity for different preconditioner-solver couples. Discretization mesh = 32x32x32, tolerance for convergence = 10-4 and 100 iterations maximum.

In parallel, studies have started on the usage of HDG methods. Adopting an HDG scheme implies a complete change of the numerical scheme of the code with the promise of considerably increasing its geometrical flexibility and allowing high-order spatial discretization. Preliminary studies have been carried out on a reduced version of the TOKAM3X model, including solely the parallel dynamics of the plasma in an isothermal assumption.

Fig. 2 shows an application of the HDG version of the code to a simulation in the WEST geometry. The HDG scheme allows one to model precisely the wall's geometry among which the fine baffle structures at the bottom of the machine. It also makes possible the extension of the simulation domain to the center of the machine, which is particularly difficult with structured mesh algorithms and resolves questions about the boundary conditions one should use at the inner boundary of the domain. This work has been carried out in collaboration with the M2P2 laboratory (CNRS, Aix-Marseille University). Future work will focus on the implementation of heat transport equations whose properties (stiffness, anisotropy) will be a decisive test for the potentiality of HDG schemes to treat the whole TOKAM3X model in sight of ITER cases.

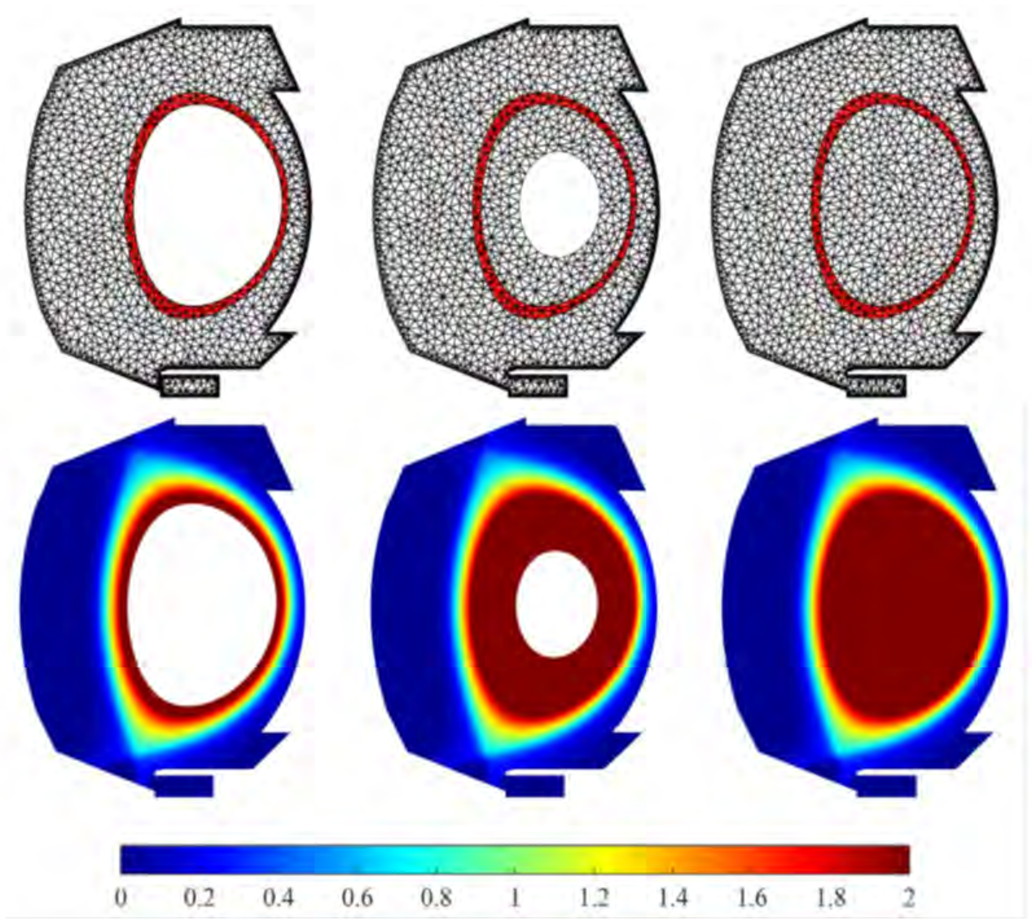


Figure 4: Top: mesh used in 3 different HDG simulations with fixed particle source location (the red-shaded area) extending the domain progressively towards the center of the machine. Bottom: corresponding equilibria obtained on the electron density field.

# 4. Shemat

| | |
|---|---|
| Activity type | WP1 support |
| Contributors | Sebastian Lührs (JSC, WP1), Johanna Bruckmann (RWTH, WP4), Henrik Büsing (RWTH, WP4), Jan Niederau (RWTH, WP4) |

## 4.1 Overview

**Implementation of HDF5 as a parallel I/O input format for SHEMAT-Suite**

The target of this support activity was the optimization of the I/O behavior of SHEMAT-Suite by adding new HDF5[4] capabilities for the input parsing process.

The existing established input format of SHEMAT-Suite allows the usage of a mix of ASCII and HDF5 (in a preliminary version) input files. The HDF5 files are referenced in the ASCII file. Large datasets can either be provided directly in the ASCII file or via the separate HDF5 file. The existing HDF5 capabilities weren't used quite often and could only be used for a subset of input parameters. So far the HDF5 input files are generated by SHEMAT-Suite itself. So these files could only be used to allow re-usage of datasets in a secondary run (e.g. to restart with checkpoint data). New datasets from scratch could only be defined via the ASCII format.

The handling of the ASCII file format could be rather slow if all different input variables are written to a single input file (which is a common input case). Instead it is also possible to distribute the data over multiple files. In the case of using a single input file and a file size larger then 100 MB the input file read duration of SHEMAT-Suite could take multiple minutes.

To allow a more flexible HDF5 input format (without using SHEMAT-Suite for conversion), parallel I/O capabilities and to avoid the long input file handling of large ASCII input files, the code developers asked for application support by WP1 to implement a new HDF5 based input strategy.

## 4.2 Structure

To allow an easy implementation of the new input format, two new parts within SHEMAT-Suite were implemented as part of this support activity:

- A conversion script which efficiently converts the existing input format to the new input format. By using this intermediate script solution, all preprocessing steps can stay unchanged and are not affected by this implementation.
- Adding new HDF5 capabilities to SHEMAT-Suite to allow parsing of the new input files.

The output behavior of SHEMAT-Suite was not changed.

The old ASCII based input format contains several different variables to parametrize SHEMAT-Suite. In addition this format also supports different ways to represent these data (e. g. by using Fortran based repeated values like `10 * 2.3` to reuse the same value ten times). The different variables and the different formats must be interpreted by the

[4]https://support.hdfgroup.org/HDF5

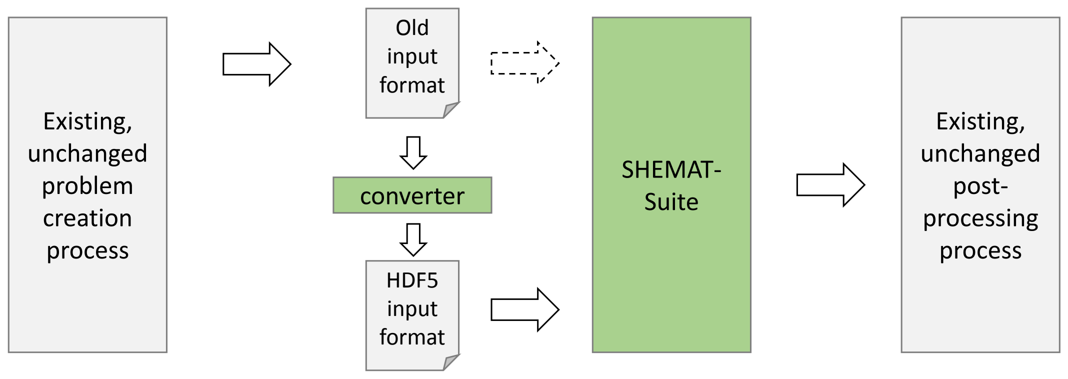

Existing, unchanged problem creation process
Old input format
converter
HDF5 input format
SHEMAT-Suite
Existing, unchanged post-processing process

Figure 5: Updated SHEMAT-Suite I/O workflow layout. The parts which were added/changed in the activity are marked in green.

conversion script. The new implementation focuses on the larger variables in context of data size, smaller scalar values are currently not converted and stay unchanged in the old input file. The parsing process automatically change between the old and the new input format if a variable is found which isn't converted so far. This also allows easy addition of new variables in the future, because these could be added to the old input format and don't need to be directly added into the conversion process. On the other site additional variables can be converted one after each other. Not all variables together has to be converted to still allow program execution. Until now nearly thirty variables were already moved from the old to the new format.

## 4.3 Implementation

The existing ASCII format uses header lines to mark the different variables in the input file.

As an example the following lines describe the general grid structure:

```
# grid
362 287 72

# delx
362*10.

# dely
287*10.

# delz
72*10.
```

These examples are rather short entries because they use the repeat feature of Fortran to avoid repeating multiple values. Other entries might contain several MB of data. Due to the existing parsing implementation of SHEMAT-Suite, the input file is searched for each individual header line starting at the beginning of the file. This process can extremely slow down the parsing process if header entries do not exist (e.g. if they are optional), because the whole file has to be scanned multiple times.

The new conversion script is written in Python using the h5py[5] HDF5 Python bindings. Because it reads the original input file, the existing parsing process of SHEMAT-Suite has to be reimplemented to support as many input features as possible. To avoid the same header searching bottleneck, the file is only scanned once to mark all header lines. After this, all existing headers can easily be reached by jumping directly to the specific file position.

HDF5 does not support some of the features which were present in the old ASCII format (like the Fortran repeat syntax). Such values are now automatically converted to a general format using a fixed structure. Repeated values are now also stored directly in the file, which increases the file size, but also allows to use a file more easily within other applications.

The conversion script will produce the following HDF5 structure to store the grid layout:

```
GROUP "grid" {
      DATASET "delx" {
         DATATYPE  H5T_IEEE_F64LE
         DATASPACE  SIMPLE { ( 362 ) / ( 362 ) }
      }
      DATASET "dely" {
         DATATYPE  H5T_IEEE_F64LE
         DATASPACE  SIMPLE { ( 287 ) / ( 287 ) }
      }
      DATASET "delz" {
         DATATYPE  H5T_IEEE_F64LE
         DATASPACE  SIMPLE { ( 72 ) / ( 72 ) }
      }
   }
```

Once a variable is extracted, it is deleted from the ASCII file to only keep the unconverted values. In addition a new entry is added to the ASCII file automatically pointing to the new HDF5 data file. The ASCII file itself still remains the main entry point for SHEMAT-Suite. Depending on the presence of the HDF5 file link entry, the new or old parsing process is triggered.

Within SHEMAT-Suite a new HDF5 parsing interface was implemented. This interface wraps the most common reader functions and some additional help routines. The new HDF5 data file is opened once in the beginning and is kept in a global reachable file handle until all input datasets are loaded. The data layout (dimension, type and structure) in the HDF5 file was selected based on the SHEMAT-Suite internal data representation to avoid any conversion process.

In addition to the performance improvements and the direct HDF5 conversion capabilities, the new input format also allows the usage of distributed I/O calls calls in future implementations of SHEMAT Suite. Instead of reading global datasets with each individual processor, the usage of distributed I/O calls can help to avoid memory scalability

[5]http://www.h5py.org/

problems once the application will be executed on larger scales.

### 4.4 Results

To validate the benefit of the new input parsing process, the input parsing time was measured using different filesizes (of the original ASCII main input file) and different number of cores on the JURECA [6] system. The comparison is done between the original ASCII based input file and the new converted input file. The time for the conversion process itself is included in the new timings.

Figure 6 shows a small input file, were the old and the new input format show nearly the same read duration and scaling behavior.

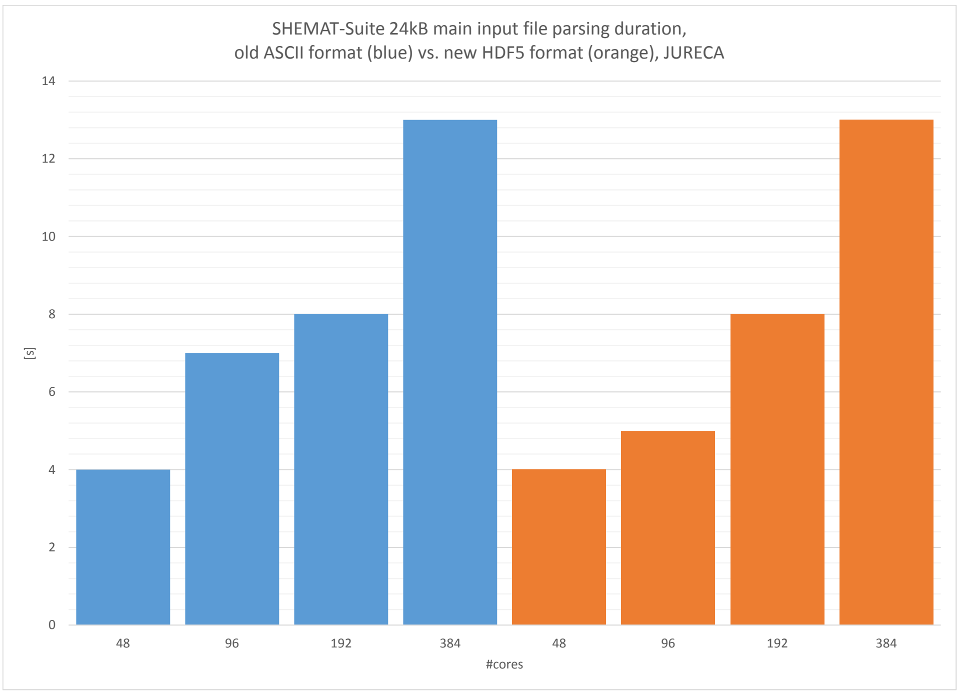


Figure 6: SHEMAT-Suite input parsing time on JURECA using different number of cores and a 24kB main input file. The old ASCII format is marked in blue, the new HDF5 format is marked in orange.

Figure 7 shows a second input case, using a larger ASCII input file. The ASCII parsing process is already much slower in comparison to the new format due to the header line handling which is mentioned before.

Figure 8 shows an (in context of SHEMAT-Suite) large input case, storing more than 200MB within the single main SHEMAT-Suite ASCII input file. The parsing of this file takes nearly 45 minutes in serial by using the old parsing process. This process could be significantly improved and reduced to less then a minute.

As shown in the figures, the new input format could speed up the parsing process. Additional variables could easily be added to the conversion process to allow future sustainability.

[6]http://www.fz-juelich.de/ias/jsc/EN/Expertise/Supercomputers/JURECA/JURECA_node.html

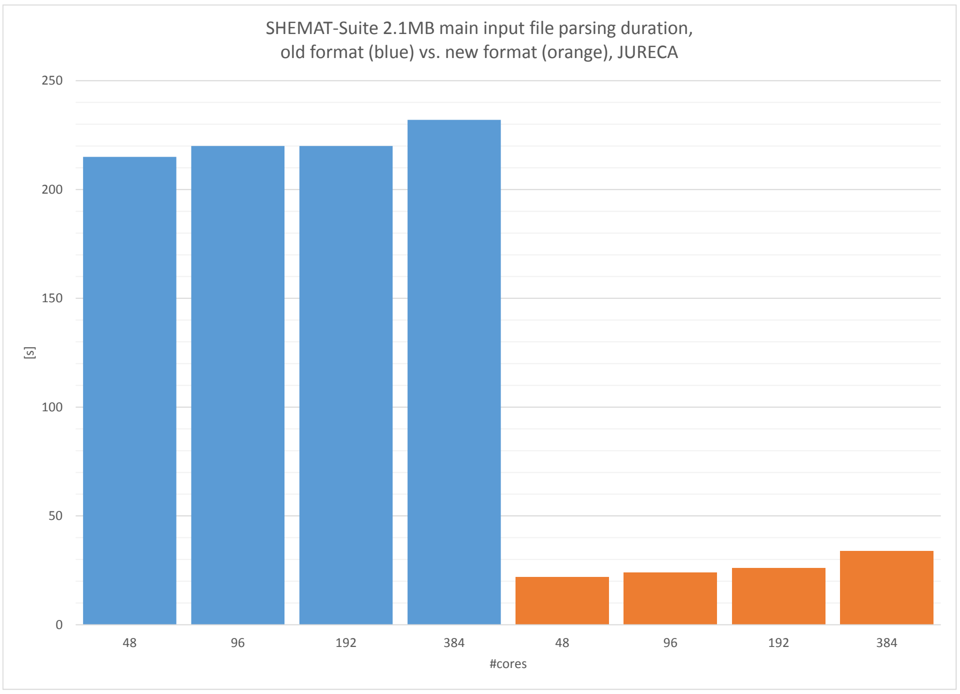


Figure 7: SHEMAT-Suite input parsing time on JURECA using different number of cores and a 2MB main input file. The old ASCII format is marked in blue, the new HDF5 format is marked in orange.

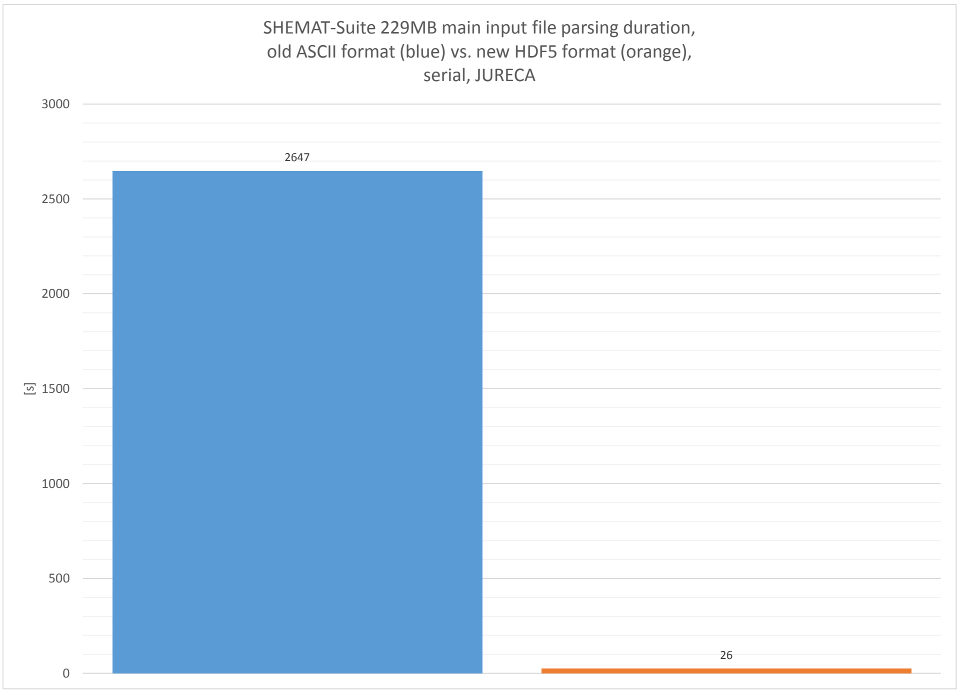


Figure 8: Serial SHEMAT-Suite input parsing time on JURECA using a 229MB main input file. The old ASCII format is marked in blue, the new HDF5 format is marked in orange.

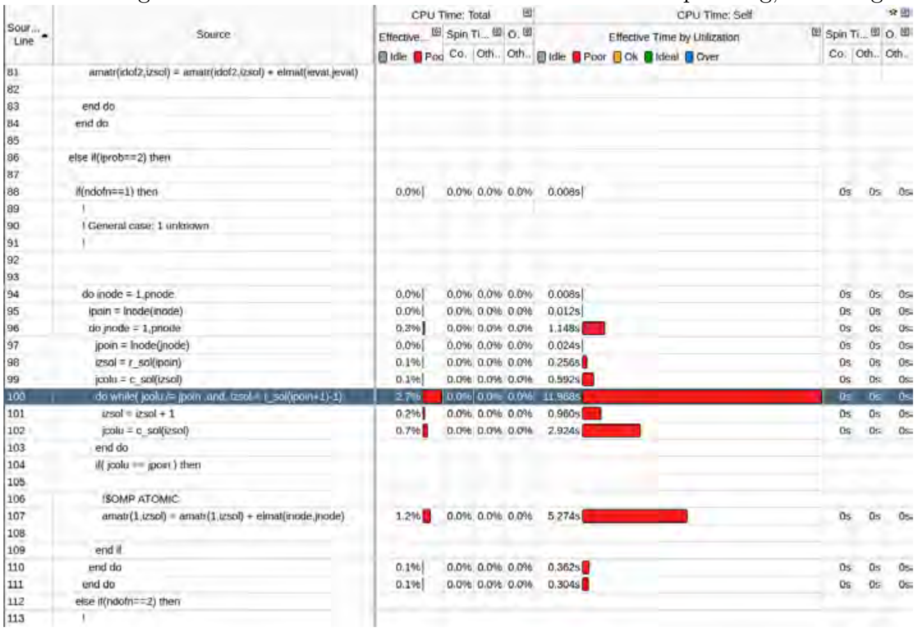


Figure 9: Part of *csrase* subroutine instruction level profiling, unveiling a costly loop

| Sour... Line | Source | CPU Time: Total | | | | CPU Time: Self | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | Effective... | Spin Ti... | | O. | Effective Time by Utilization | Spin Ti... | | O. |
| | | 🟫 Idle 🟥 Poo | Co. | Oth.. | Oth.. | 🟫 Idle 🟥 Poor 🟧 Ok 🟩 Ideal 🟦 Over | Co. | Oth.. | Oth.. |
| 81 | amatr(idof2,izsol) = amatr(idof2,izsol) + elmat(ievat,jevat) | | | | | | | | |
| 82 | | | | | | | | | |
| 83 | end do | | | | | | | | |
| 84 | end do | | | | | | | | |
| 85 | | | | | | | | | |
| 86 | else if(iprob==2) then | | | | | | | | |
| 87 | | | | | | | | | |
| 88 | if(ndofn==1) then | 0.0% | 0.0% | 0.0% | 0.0% | 0.008s | 0s | 0s | 0s |
| 89 | ! | | | | | | | | |
| 90 | ! General case: 1 unknown | | | | | | | | |
| 91 | ! | | | | | | | | |
| 92 | | | | | | | | | |
| 93 | | | | | | | | | |
| 94 | do inode = 1,pnode | 0.0% | 0.0% | 0.0% | 0.0% | 0.008s | 0s | 0s | 0s |
| 95 | ipoin = lnode(inode) | 0.0% | 0.0% | 0.0% | 0.0% | 0.012s | 0s | 0s | 0s |
| 96 | do jnode = 1,pnode | 0.3% | 0.0% | 0.0% | 0.0% | 1.148s | 0s | 0s | 0s |
| 97 | jpoin = lnode(jnode) | 0.0% | 0.0% | 0.0% | 0.0% | 0.024s | 0s | 0s | 0s |
| 98 | izsol = r_sol(ipoin) | 0.1% | 0.0% | 0.0% | 0.0% | 0.256s | 0s | 0s | 0s |
| 99 | jcolu = c_sol(izsol) | 0.1% | 0.0% | 0.0% | 0.0% | 0.592s | 0s | 0s | 0s |
| 100 | do while( jcolu /= jpoin .and. izsol /= r_sol(ipoin+1)-1) | 2.7% | 0.0% | 0.0% | 0.0% | 11.968s | 0s | 0s | 0s |
| 101 | izsol = izsol + 1 | 0.2% | 0.0% | 0.0% | 0.0% | 0.960s | 0s | 0s | 0s |
| 102 | jcolu = c_sol(izsol) | 0.7% | 0.0% | 0.0% | 0.0% | 2.924s | 0s | 0s | 0s |
| 103 | end do | | | | | | | | |
| 104 | if( jcolu == jpoin ) then | | | | | | | | |
| 105 | | | | | | | | | |
| 106 | !$OMP ATOMIC | | | | | | | | |
| 107 | amatr(1,izsol) = amatr(1,izsol) + elmat(inode,jnode) | 1.2% | 0.0% | 0.0% | 0.0% | 5.274s | 0s | 0s | 0s |
| 108 | | | | | | | | | |
| 109 | end if | | | | | | | | |
| 110 | end do | 0.1% | 0.0% | 0.0% | 0.0% | 0.362s | 0s | 0s | 0s |
| 111 | end do | 0.1% | 0.0% | 0.0% | 0.0% | 0.304s | 0s | 0s | 0s |
| 112 | else if(ndofn==2) then | | | | | | | | |
| 113 | ! | | | | | | | | |

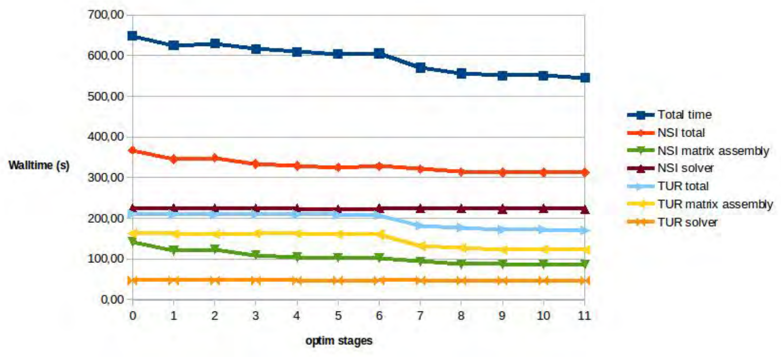
## 5. Alya

| | |
|---|---|
| Activity type | WP1 support |
| Contributors | Y. Ould-Rouis (WP1), H. Owen (WP2), G. Houzeaux (WP2) |

Core level optimization

1. detection of loop level pathologies using VTune.
2. loops reordering : in order to take profit of data locality in the cache by contiguous accesses.
3. Refactoring of potential redundant calculations.
4. Helping vectorization : by data restructuring and getting rid of dependencies inside inner loops.
   example : in *nsi_elmmat*, building elauu matrix as 9 distinct arrays that are assembled together at the end allowed an 11% improvement of this routine, in addition to the 15% obtained with the 2 previous steps.
5. Memory padding (or data structure alignment). tested only on some local array variables.
6. eliminating while/conditional loops.
7. Element to sparse matrix pre-mapping : This step was motivated by the observation that *csrase* subroutine contains a very costly while loop (2.7% of total CPU time) [figure 9]. This subroutine is used in turbul matrix assembly, to copy the coefficients calculated by element into a larger sparse matrix, and the while loop finds for each coefficient of indices (inode, jnode) in a local element's matrix the right index 'izsol' in the CSR matrix. Same is done other way around, and in other parts/modules.
   The idea is to calculate this relation once in a pre-processing step.

Figure 10: ALYA NSI+TUR perf evolution - 1Melem, 30 timesteps, 1 node (16 processes) on MareNostrum



- advantages : avoid a costly redundant while loop.
- neutral : it won't solve the problem of indirections when writing in the sparse matrix.
- negative : an estimated memory cost of 8*8*integer_size for each element. With long integers (8 bytes), this means 50Mbytes for each 100 000 elements. This is reasonable when compared to the total memory footprint.

<u>Results</u>

The core level optimization has been successful in securing 10 to 13% gain in total, depending on the hardware. The figure 10 shows the evolution of the performance for the successive versions of Alya. The stage 0 shows the original times. Stages 1 to 6 show the results of steps 2 to 6 described above, applied to the NASTIN matrix assembly. Stages 7 to 11 are the results of steps 2 to 6 applied to the TURBUL module combined with other efforts.

The final results on Jureca are as follows :

- NASTIN matrix assembly : 35% improvement.
- TURBUL matrix assembly : 22% improvement.
- Total : 10% global improvement.

## 6. Metalwalls

| | |
|---|---|
| Activity type | Consultancy (WP1 support request on-going) |
| Contributors | Haefele M. (Marin Lafleche A. if request accepted) |

From a computer science point of view, the application follows a two steps algorithm: i) to compute the charges density within the electrodes according to the atom locations, an iterative process that requires to reach a convergence, ii) to compute the position of each atom according to the charge density. More than 90% of the total runtime is spent in the charge density computation, so this is definitely a target for the code optimization process.

In term of parallelisation, the application is pure MPI and no data is distributed across the ranks. This means that each rank has all the whole data. Only the computations are distributed such that each rank computes the interaction between a set of pairs of atoms. An MPI Allreduce sums up all these contributions and sends the result back to all MPI ranks.

Three optimizations have been implemented in Metalwalls during this application support activity.

### 6.1 Memory footprint reduction

At several places in the code, the information if the interaction between two specific atoms has to be taken into account is needed. As this information depends only on the type of system simulated, in the original version, it was computed once during the code intialisation and stored. But the amount of memory required to store this information grows with $N^2$, $N$ being the number of atoms in the simulation.

The optimisation that has been implemented suppresses completely the need for this $N^2$ memory by recomputing this information each time it is required from existing information of size $N$. Now the memory footprint of the application scales linearly with the number of atoms and enables to treat larger systems. From the restitution time point of view, this optimisation had only a moderate impact as the time spent in this part of the code was not that important. Unfortunately, we could not measure the impact of this single optimisation as it has been done in conjunction with the code vectorisation.

### 6.2 Vectorisation

As mentioned in the performance report, the vectorization of the code could be the source of potential improvements. A careful examination of the compiler log could identify the internal loops that the compiler could not vectorize.

For instance, Fig 11 shows a kernel not vectorized by the compiler. The *if* statement introduces an issue: the iteration $j = i$ executes different code than $j = i - 1$ and $j = i + 1$. The compiler can simply not transform this code into a Single Instruction Multiple Data (SIMD) version. As a consequence, the whole $j$ loop is not vectorized. By examining the code in the *if* and *else* branches, one can notice that the purpose of this construct is to save the evaluations of an error function, an exponential function and some multiplications. This optimisation has been likely implemented at a time where scalar processors did not have vector units. Nowadays, this construct prevents the compiler from introducing vector instructions. By removing the *if* part and keeping only the *else* part, a speedup of 2.5 could

```
do i = ibeg_w , iend_w
  vsumzk0=0.0d0
  do j = nummove+1,num
    if (i==j) then
      vsumzk0=vsumzk0+q(j)*sqrpieta
    else
      zij=z(i)-z(j)
      zijsq=zij*zij
      rerf = erf(eta*zij)
      vsumzk0=vsumzk0+q(j)*((sqrpieta*exp(-etasq*zijsq))+&
        (pi*zij*rerf))
    end if
  enddo
  cgpot(i)=cgpot(i)-vsumzk0
enddo
```

Figure 11: Kernel not vectorized by the compiler

be obtained on this single kernel.

Other code modifications enabled the compiler vectorization and now, thanks to Intel Advisor, we could check that all the kernels in the high computing intensity part of Metalwalls are vectorized by the compiler.

### 6.3 Cache blocking

During the porting of Metalwalls on Intel Xeon Phi KNL architecture, we observed larger run times than expected for some routines and especially the *cgwallrecipE* routine. After an examination with the memory analyser of VTune, it turned out that these routines were almost compute bound on Xeon architectures and became memory bound on KNL. A careful examination of the source code revealed that several large arrays were accessed within the same kernel. These large arrays were still fitting in the L3 cache of Xeon processors but, as there is no L3 cache on KNL, these arrays could not fit into L2 cache. As a consequence, the kernel triggered a large amount of memory transfer and, despite the considerably large memory bandwidth of KNL's MCDRAM, the execution time of this kernel on KNL was larger by a factor of 8.

A cache blocking mechanism has been implemented on this kernel. Instead of performing computations on the total size of the arrays, computations are performed only on a subset such that the sum of all these subsets fit into the cache. The implementation was not completely trivial as a reduction of some of these arrays was performed inside the kernel and used directly in the kernel. The kernel had to be split in two and an intermediate data structure that accumulates the partial reductions had to be introduced. The overhead in memory of this data structure is negligible and we could recover very good performance on this specific routine.

### 6.4 Results

Metwalls' performance has been improved by a factor 2.5. This work has been merged in the production version of the code in January 2016. 28 MCPUh have been used for Metalwalls on various Tier-0 and French Tier-1 machines during 2016. So performing the same numerical experiments would have required 70 MCPUh without the code optimisations. This represents a saving of 42 MCPUh for the year 2016.

Metalwalls has also been tested on Intel Xeon Phi KNL processors available at CINES[7]. It turns out that on 4KNL nodes, the code is faster by 40% compared to 4 Intel Haswell nodes. The fact that the code requires a very small amount of memory and presents high complexity algorithms helps considerably in achieving these good results. These preliminary results are very encouraging.

[7]`https://www.cines.fr/`

## 7. Gysela

| | |
|---|---|
| Activity type | Consultancy or WP1 support |
| Contributors | Brian Wylie (WP1, Germany), Judit Gimenez (WP1, Spain), Guillaume Latu (France), Julien Bigot (WP1, France) |

### 7.1 Direct benefits of SMT

To evaluate SMT, we choose a domain size of $N_r \times N_\theta \times N_\varphi \times N_{v_\parallel} \times N_\mu = 512 \times 256 \times 128 \times 60 \times 32$ in this section. Due to GYSELA internal implementation choices, we are constrained to choose, inside each MPI process, a number of threads as a power of two. Let us remark, that the application performance increases by avoiding very small power of two (*i.e.* 1, 2). Haswell node that we target are made of 24 cores. That is the reason why we choose to set 8 threads per MPI process for the runs shown hereafter. This configuration will allow us to compare easily an execution with or without SMT activated.

In the following, the deployment with 3 MPI processes per node (one compute node, 24 threads, 1 thread per core) is checked against a deployment with 6 MPI processes per node (one compute node, 48 threads, 2 threads per core, SMT used). Strong scaling experiments are conducted with or without SMT, timing measurements are shown in Table 3. Let us assume that processes inside each node is numbered with an index $n$ going from $0$ to $2$ without SMT, and $n = 0$ to $5$ whenever SMT is activated. For process $n$, threads are pinned to cores in this way: logical cores id from $8n$ up to $8n + 7$.

| **Number of nodes/cores** | **Exec. time (1 th/core)** | **Exec. time (2 th/core)** | **Benefit of SMT** |
|---|---|---|---|
| 22/ 512 | 1369s | 1035s | **-24%** |
| 43/1024 | 706s | 528s | **-25%** |
| 86/2048 | 365s | 287s | **-21%** |
| 172/4096 | 198s | 143s | **-28%** |

Table 3: Time measurements for a strong scaling experiment with SMT activated or deactivated, and gains due to SMT

The different lines show successive doubling of the number of cores used. The first column gives the CPU resources involved. The second and third columns highlight the execution time of mini runs comprising 8 time steps (excluding initialization and output writings): using 1 thread per core (without SMT), or using 2 threads per core (with SMT support). The last column points out the reduction of the run time due to SMT comparing the two previous columns. As a result, the simultaneous multi-threading with 2 threads per core gives a benefit of 21% up to 28% over the standard execution time (deployment with one thread per core). While an improvement is expected with SMT, as already reported for other applications this speedup is quite high for a HPC application.

Within Paraver, we observe that for each intensive computation kernel the number of instructions per cycle (IPC) cumulated over the 2 threads on one core with SMT is always higher than the IPC obtained with one thread per core without SMT. For these kernels, the cumulated IPC is comprised between 1.4 and 4 for two threads per core with SMT, whereas it is in the range of 0.9 up to 2.8 with one thread per core without SMT. These IPC numbers should be compared to the number of micro-operations achievable per cycle, 4 on Haswell. Thus, we use a quite large fraction of available micro-operation slots. Two factors explain the boost in performance with SMT. First, SMT hides some cycles wastes due to data dependencies and long latency operation (*e.g* memory accesses).

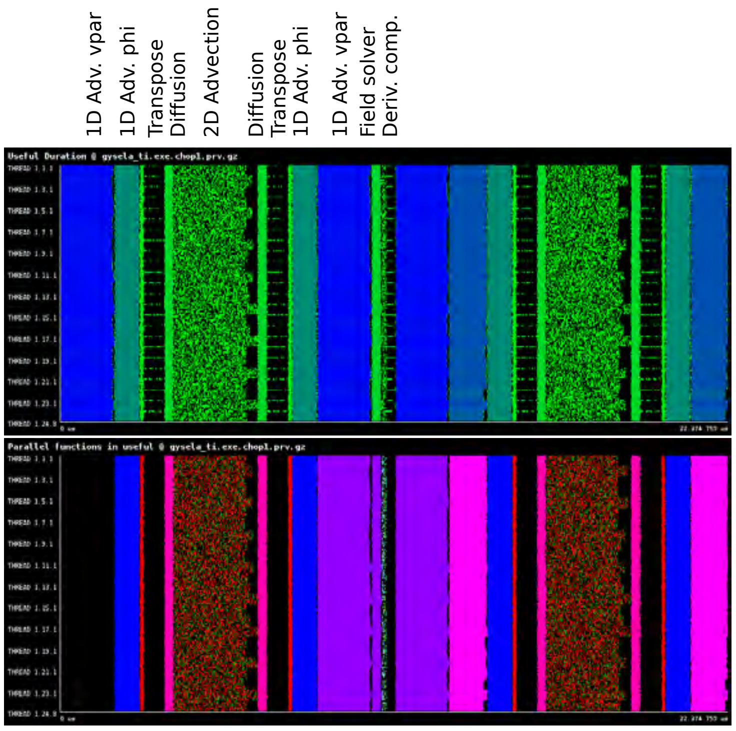


Figure 12: Snippet of a run with 2 threads per core (SMT), Top: Paraver useful duration plot, Bottom: Parallel functions plot

Second, SMT enables to better fill available execution units. It provides a remedy against the fact that, within a cycle, some issue slots are often unused.

## 7.2 Optimizations to increase SMT gain

The Paraver tool gives us the opportunity to have a view of OpenMP and MPI behaviors at a very fine scale. The visual rendering informs rapidly the user of an unusual layout and therefore hints to look on some regions with unexpected patterns. On the Fig. 12 is plotted a snippet of the timeline of a small run with SMT (2 threads per core, 24 MPI processes, 8 threads per MPI process, meaning 4 nodes hosting 48 threads within each node). We can extract the following information:

1. The 2D advection kernel (first computationally intensive part of the code) is surprisingly full of small black holes.

2. There are several synchronizations during this timeline between MPI processes that are noticeable. As several moderate load imbalances are also visible, a performance penalty can be induced by these synchronizations. See for example 2D advection and Transpose steps (Useful duration plots), there is much black color at the end of these steps. This is due to final MPI barriers. Nevertheless the impact is relatively low in this reduced test case because the tool reported a parallel efficiency of 97% over the entire application indicating that only 3% of the iteration time is spend on the MPI and OpenMP parallel runtimes. The impact is stronger on larger cases, because load imbalance is larger.

3. The transpose steps show a lot of black regions (threads remaining idle). At the end of the phase, all the ranks are synchronized by the MPI_Barrier. Checking

the hardware counters indicate the problem is related with a different IPC where the fast processes are getting twice the IPC of the delayed ones. This behavior illustrates well that SMT introduces heterogeneity of the hardware that should be handled by the application even if the load is well balanced between threads.

4. At the end of 2D advection step, a serrated form is noticeable. All the processes that straddle two different sockets are slowed down a little bit.

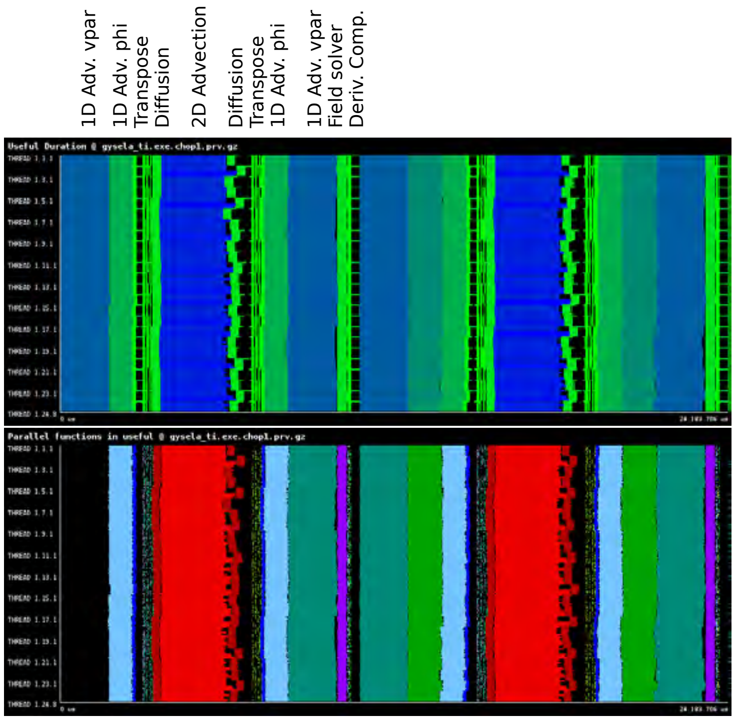


Figure 13: Snippet of a run with 2 threads per core (SMT), after optimizations are done, Top: Paraver useful duration plot, Bottom: Parallel functions plot

These inputs from the Paraver visualization helped us to determine some code transformations to make better use of unoccupied computational resources. The key point was to point out the cause of the problem, the improvements were not so difficult to put into place. The upgrade are described in the following list. The Table 4 and Fig. 13 exhibits associated measurements.

1. The 2D advection kernel is composed of OpenMP regions. There is mainly an alternation of two distinct OpenMP kernels. The first one fills the input buffers to prepare the computation of 2D spline coefficients for a set of $N$ poloidal planes (corresponding to different $\varphi, v_{\parallel}$ couples). The second kernel computes the spline coefficients for the same $N$ poloidal planes and performs the advection itself that encompasses an interpolation operator. Yet, there is no reason for having two separate OpenMP regions encapsulated in two different routines, apart from historical ones. Thus, we decided to merge these OpenMP regions in a single large one. This modification avoids the overheads due to entering and leaving the OpenMP regions multiple times. Also the implicit synchronization at the beginning and end of each parallel region are removed. Thus, avoiding synchronization leads to a better load balance by counteracting the imbalance originating mainly from the SMT effects.

2. Some years ago, with homogeneous computing units and resources, the workload in GYSELA was very balanced between MPI processes and inside them, between threads. Thus, even if some global MPI barriers were present within several routines, they induced negligible extra costs because every task was executed synchronously with the others. In latest hardware, there is heterogeneity coming from cache hierarchy, SMT, NUMA effects or even Turbo boost. The penalty due to MPI barriers is now a key issue, and thread idle time is visible on the plot. We removed several useless MPI barriers. As a result, we see for example in Fig. 13 that, now, diffusion is sandwiched between the transpose step and the 2D advection, without global synchronization.

3. The transpose step is compounded of three sub-steps: copy of data into send buffers, MPI non-blocking send/receive calls with a final wait on pending communications, copy of receive buffers into target distribution function. On the Fig. 12, it is worth noticing that only the first thread of each MPI process is working, *i.e.* only the master thread is performing a useful work. To improve this, we added OpenMP directives to parallelize all the buffers' copies. This modification increases the extracted memory bandwidth and the thread occupancy. On the Fig. 13, the bottom plot shows that the transpose step is now partly parallelized with OpenMP.

Thanks to these upgrades, there is much less black (idle time) in Fig. 13 compared to Fig. 12. Still, MPI communications induce idle time for some threads in the transpose step and in the field solver. This can not be avoided within the current assumptions done in GYSELA. Table 4 also illustrates the achieved gain in term of elapsed time. If one compares to Table 3, the timings are reduced with one or two threads per core. Comparing one against two threads per core, the SMT gain is still greater than 20% (almost the same statement as before optimization). Now, if we cumulate the gain resulting from SMT and from the optimizations, we end up with a net benefit on execution time of 32% up to 38% depending on the number of nodes.

| **Number of nodes/cores** | **Exec. time (1 th/core)** | **Exec. time (2 th/core)** | **Benefit of SMT** | **Benefit vs. Table 1** |
|---|---|---|---|---|
| 22/ 512 | 1266s | 931s | **-26%** | **-32%** |
| 43/1024 | 631s | 474s | **-25%** | **-33%** |
| 86/2048 | 320s | 239s | **-25%** | **-34%** |
| 172/4096 | 164s | 124s | **-25%** | **-38%** |

Table 4: Time measurements and gains achieved after optimizations that remove some synchronizations and some OpenMP overheads

Much of this work has been publised in a paper `https://doi.org/10.1145/2929908.2929912` entitled *Benefits of SMT and of Parallel Transpose Algorithm for the Large-Scale GYSELA Application.*

In addition to these upgrades, we have modified the code to remove the constraint of having a power of two concerning the number of threads within a MPI process. This has been a bit of work to modify some algorithms, but the reward is that we can now avoid MPI processes that straddle two different sockets. Typically we now put 2 MPI processes per socket and 2 threads per core for production runs. Avoiding the straddling allows us for an extra saving of 5% on the total execution time.

To conclude, the use of SMT has decrease run times by 24%, whereas additional optimization done in the framework of this application support brought an additional 16%

of reduction. Then, this optimization work consitutes a strong benefit for the user of Gysela application. Modifications has been included in the production code in january of 2016. The amount of core-hours consumed in 2016 on machines that have Hyperthreading/SMT activated was 40 millions for Gysela code. One can estimate that this application support has saved at least 6.4 millions of core-hours in 2016 and direct SMT use has saved 9.6 millions of core-hours.

### 7.3 Other improvements

**Portability of performance with static auto-tuning.** Within a single HPC application, multiple aims concerning the source code should be targeted at once: performance, portability (including portability of performance), maintainability and readability. These are very difficult to handle simultaneously. A solution is to overhaul some computation intensive parts of the code in introducing well defined kernels. BOAST (Bringing Optimization Through Automatic Source-to-Source Transformations, developed by INRIA project-team CORSE, part of WP1) is a metaprogramming framework to produce portable and efficient computing kernels. It offers an embedded domain specific language (using ruby langage) to describe the kernels and their possible optimization. It also supplies a complete runtime to compile, run, benchmark, and check the validity of the generated kernels. BOAST has been applied to some of the most computation intensive kernels of Gysela: 1D and 2D advection kernels. It permitted to gain speedup from 1.9$\times$ up to 5.7$\times$ (depending on the machine) on the 2D advection kernel which is a computation intensive of the code. Furthermore, BOAST is able to generate AVX-512 instructions on INTEL Xeon Phi KNL in order to get high performance on this architecture. A specific point to take into account with this approach is to handle the integration of Ruby code within the production/legacy code. This optimization work saves in average 8% of computation time over the total execution time, integration into production code will be carried out soon. This activity was part of the CEMRACS school where WP1 and WP5 people have met (`http://smai.emath.fr/cemracs/cemracs16`). A proceeding paper has been submitted that describes this auto-tuning approach for the GYSELA application.

**Portability of performance with dynamic auto-tuning.** Another option for performance portability is to use auto-tuning at runtime. Compared with static auto-tuning, this dynamic approach incurs more overhead at runtime but it is able to leverage information that only becomes available at execution. The result of these different compromises is that the dynamic approach makes sense at a coarser grain than the static approach and that is therefore interesting to combine both. In the case of Gysela, we have implemented this approach based on the StarPU runtime developed in Inria project-team STORM (related to WP1 contribution). The whole 2D advection of which the kernel optimized with BOAST is a part has been ported to use the native StarPU API for parallelization instead of OpenMP. This new approach makes it possible to express parallelism at a grain that would be complex to express in the previously used OpenMP fork-join model and thus improve cache usage. StarPU should also improve performance portability by letting execution choices be made in the StarPU scheduling plug-in rather than in the application code. The scheduling plug-in can take into account informations about the available hardware and can even be changed for different executions. Some preliminary evaluations on the Poincare cluster have demonstrated a 15% speedup on a realistically sized case compared to the version using OpenMP fork-join. This optimization can not be included in production code for the moment. This work has also been described in the proceeding paper of the CEMRACS school.

## References

[BKN11] K. Busch, M. König, and J. Niegemann. Discontinuous Galerkin methods in nanophotonics. *Laser and Photonics Reviews*, 5:1–37, 2011.

[CK94] M.H. Carpenter and C.A. Kennedy. Fourth-order $2n$-storage Runge-Kutta schemes. Technical report, NASA Technical Memorandum MM-109112, 1994.

[FL10] H. Fahs and S. Lanteri. A high-order non-conforming discontinuous Galerkin method for time-domain electromagnetics. *J. Comp. Appl. Math.*, 234:1088–1096, 2010.

[FLLP05] L. Fezoui, S. Lanteri, S. Lohrengel, and S. Piperno. Convergence and stability of a discontinuous Galerkin time-domain method for the 3D heterogeneous Maxwell equations on unstructured meshes. *ESAIM: Math. Model. Numer. Anal.*, 39(6):1149–1176, 2005.

[HW02] J.S. Hesthaven and T. Warburton. Nodal high-order methods on unstructured grids. I. Time-domain solution of Maxwell's equations. *J. Comput. Phys.*, 181(1):186–221, 2002.

[Mai07] S.A. Maier. *Plasmonics - Fundamentals and applications.* Springer, 2007.

[MNHB11] C. Matysseka, J. Niegemann, W. Hergertb, and K. Busch. Computing electron energy loss spectra with the Discontinuous Galerkin Time-Domain method. *Photonics Nanostruct.*, 9(4):367–373, 2011.

[NDB12] J. Niegemann, R. Diehl, and K. Busch. Efficient low-storage Runge-Kutta schemes with optimized stability regions. *J. Comput. Phys.*, 231(2):364–372, 2012.

[NKSB09] J. Niegemann, M. König, K. Stannigel, and K. Busch. Higher-order time-domain methods for the analysis of nano-photonic systems. *Photonics Nanostruct.*, 7:2–11, 2009.

[TH05] A. Taflove and S.C. Hagness. *Computational electrodynamics: the finite-difference time-domain method - 3rd ed.* Artech House Publishers, 2005.

[Viq15] J. Viquerat. *Simulation of electromagnetic waves propagation in nano-optics with a high-order discontinuous Galerkin time-domain method.* PhD thesis, University of Nice-Sophia Antipolis, 2015. https://tel.archives-ouvertes.fr/tel-01272010.

[VL16] J. Viquerat and S. Lanteri. Simulation of near-field plasmonic interactions with a local approximation order discontinuous Galerkin time-domain method. *Photonics and Nanostructures - Fundamentals and Applications*, 18:43–58, 2016.

[VS15] J. Viquerat and C. Scheid. A 3D curvilinear discontinuous Galerkin time-domain solver for nanoscale light–matter interactions. *J. Comp. Appl. Math.*, 289:37–50, 2015.

[Yee66] K.S. Yee. Numerical solution of initial boundary value problems involving Maxwell's equations in isotropic media. *IEEE Trans. Antennas and Propag.*, 14(3):302–307, 1966.