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

D1.11 - M28
Applied research activities

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:	D1.11 - M28
	Lead Beneficiary:	Juelich JSC
	Contact:	Paul Gibbon
	Contact e-mail:	p.gibbon@fz-juelich.de
	Deliverable Nature:	Report
	Dissemination Level:	PU*
	Contractual Date of Delivery:	M28 01/31/2018
	Actual Date of Delivery:	05/31/2018
	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:	Applied research activities
	ID:	D1.11 - M28
	Available at:	http://www.eocoe.eu
	Software tool:	LaTeX
Authorship	Written by:	Matthieu Haefele (MdlS), Stéphane Lanteri (INRIA), Sebastian Lührs (JSC), Salvatore Filippone (Cranfield University), Corentin Roussel (MdlS), Alexis Gobé (Inria), Giorgio Giorgiani (CEA)
	Contributors:	Wolfgang Frings (JSC), Agostino Funel (ENEA), Fiorenzo Ambrosino (ENEA), Maciej Brzezniak (PSNC), Krzysztof Wadowka (PSNC), Karol Sierocinski (PSNC), Tomasz Paluszkiewicz (PSNC), Pasqua D'Ambra (CNR), Daniela di Serafino (U. Campania), Julien Bigot (MdlS), Leonardo Bautista (BSC), Kai Keller (BSC), Jonathan Viquerat (Inria), Patrick Tamain (CEA)
	Reviewed by:	Paul Gibbon (JSC)

Contents

1. Document release note

This document is the first report on software technology improvement. Some activities are already implemented and some others are still on going work. The final document D1.12 due for M36 will replace this document and contain the final status of all activities that have taken place in in the applied research context.

2. Motivation

From the outset, the EoCoE project was equipped with a diverse set of HPC expertise in WP1 designed to tackle a variety of possible performance bottlenecks in the applications from the four domain pillars. These range from state-of-the-art computer science tools for performance analysis, parallel IO etc. . . , to advanced linear algebra and other applied mathematics methods. This permits a layered approach to application tuning, starting from initial blind analysis to identify problematic code portions, then subsequently delving deeper to untertake complete refactoring of critical, compute-intensive routines. The key feature of EoCoE has been the close interaction between WP1 and the application domains WP2-WP5, enabling real-world energy applications to effectively exploit the existing European computing infrastructure and better equip them for future hardware advances. Ultimately we expect this work to expedite advances in simulations of low-carbon energy systems and technology.

This deliverable gathers the status of the long-term activities conducted within the project. By "long-term" we mean that EoCoE contributes to applied research whose results will likely have an impact well beyond the end of the project. This includes, for example, activities in advanced applied mathematics involving substantial personnel resources (up to 24PM), allowing more radical reworking of core numerical schemes in a given scientifi c application, together with comprehensive validation using real test cases. Another important aspect is the development of new generic software packages and libraries, which also consumes time and eff ort to ensure an impact within the HPC community beyond the immediate scope of the EoCoE.

3. High order finite elements for Tokam3X

Contributors	Giorgio Giorgiani (CEA Cadarache), Patrick Tamain (CEA Cadarache)

The main goal of this research is to improve the numerical efficiency of the code TOKAM-3X. This task is divided in two main parts:

- implementation of scalable linear solvers for the inversion of the vorticity operator;
- investigation of advanced numerical schemes for plasma simulations based on non-aligned discretizations.

The methodology and results obtained for the two tasks are detailed next.

4. Scalable linear solvers in TOKAM-3X

4.1 Introduction

The vorticity operator describes the evolution of the electric potential in the machine. Due to the fast dynamic associated to the electric potential, the vorticity operator is treated implicitly, giving rise to a 3D Poisson-type equation with strongly anisotropic coefficients in the parallel and perpendicular directions with respect to the magnetic field. The spatial discretization is obtained with a second-order finite difference scheme specifically designed for anisotropic problems [1], and produces a linear system which solution represents the value of the electric potential in each point of the 3D plasma domain.

For physically interesting parameters and sufficiently fine meshes, the resulting linear system is typically characterized by high condition numbers ($> 10^{10}$). This is due to essentially three facts:

- strong anisotropy: the ratio between the parallel and the perpendicular diffusion in the vorticity operator is usually considered of the order 10^5;
- non alignment with the computational mesh: due to the shape of the magnetic field lines, the computational grid is aligned along the magnetic lines in the poloidal plane, but in the toroidal direction a small non-alignment is typically present (pitch angle);
- Bohm boundary condition: the plasma wall interaction is described by the so-called Bohm boundary condition for the electric potential, which is translated mathematically by a Robin-type condition where the ratio between the Neumann part and the Dirichlet part has the magnitude of the anisotropy ratio.

The approach used so far to invert the linear system has been to use a direct solver. In particular, the parallel library PastiX, developed by INRIA Bordeaux, has been used for this task. However, the scaling laws of performing a Gaussian elimination on a discretized 3D domain ($\mathcal{O}(n^{4/3})$ fill-in scaling and $\mathcal{O}(n^2)$ flop scaling), limit on the one hand the attainable grid resolution, and on the other obliges to make an isothermal hypothesis of the plasma (that allows to perform the LU factorization of the matrix just once at the beginning of the time iterations).

In order to overcome this limitation, the direct solver should be replaced by an iterative solver. Due to the high-condition number of the matrix, dedicated solution strategies are investigated.

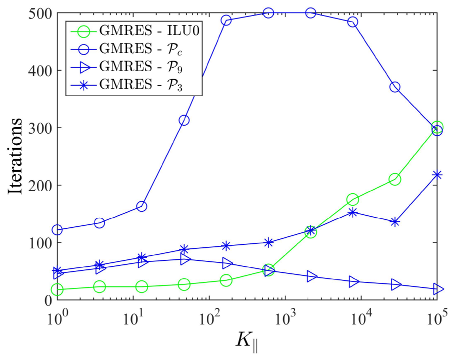


Figure 1: GMRES study: iterations to converge with different preconditioners.

4.2 Solution considered and state of the art

Three solutions were considered so far:

- **Preconditioned GMRES**: this solution relies on the implementation of a parallel GMRES iterative solver in the code TOKAM-3X. The key ingredient is the identification of a preconditioning technique, based on a physical insight of the problem, that allows to speed-up the convergence of the iterative solution. A reduced Matlab model was used to perform a parametric study of different preconditioners, that allowed to identify a suitable candidate for the introduction in the TOKAM-3X code, see Figure 1. The parallel GMRES scheme is introduced in TOKAM-3X via PastiX, with the "iterative refinement" procedure.
Preliminary tests on a serial version give promising results. More conclusive experiments will be performed with the distributed version when a minor interface problem in PastiX will be solved by the Bordeaux team.

- **AGMG solver**: AGMG implements an aggregation-based algebraic multi-grid method [2]. It is developed by the Université Libre de Bruxelles, which collaboration with the IRFM is brought in by the EoCoE network. After some preliminary tests performed on matrices saved from TOKAM-3X executions, the solver was implemented in the code and tested in a serial implementation. Results proved that AGMG could be a suitable candidate to replace PastiX for performing non-isothermal computations (hence, a new matrix is computed at each time iteration), and could also allow to attain finer grids than the one computed nowadays. However, the performance depends on the magnetic field geometry and needs to be further investigated. More specific results will be obtained with a distributed version of AGMG which is now in the implementation process in TOKAM-3X.

- **MaPHyS solver**: the EoCoE collaborative network allowed to test another iterative solver, MaPHyS, developed by INRIA Bordeaux [3]. MaPHyS is a hybrid direct/iterative solver based on domain decomposition. Prelimiary results based on saved TOKAM-3X matrices showed interesting results, but so far less promising then the ones obtained with AGMG. The implementation in TOKAM-3X is in process.

5. Investigate advanced numerical schemes for plasma simulations based on non-aligned discretizations

5.1 Introduction

Computational grids aligned with the magnetic field lines are of great interest in numerical modeling of tokamaks. From a numerical point of view, the use of aligned grids allows to reduce the pollution error in the perpendicular direction introduced by strong anisotropic equations.

However, non-aligned schemes open the path to new code capabilities that are nowadays very difficult to obtain with the aligned approach, for example

- very accurate description of the reactor chamber,
- computation extended up to the plasma center,
- possibility of computing the plasma transport in a moving equilibrium situation, for example, at start-up or during control operations.

Therefore, in order to introduce these new capabilities in the code TOKAM-3X, a non-aligned scheme is investigated. The interest was focused on a hybrid discontinuous Galerkin scheme (HDG), for its unique properties of stabilization, robustness and reduced degrees of freedom. In order to reduce the numerical diffusion introduced by the non-aligned approach, high-order polynomials are used for the interpolation of the solution.

5.2 HDG scheme and results

At first, a 2D isothermal model has been developed, with unknowns n, the plasma density, and Γ, the plasma parallel momentum. The code works on unstructured triangular grids with curved geometries, and employs polynomial interpolations of arbitrary order to approximate the solution. The code has been validated with manufactured solutions showing high-order convergence of the numerical solution and the recovery of the theoretical slopes, see Figure 2. A benchmark with the code SOLEDGE2D was then proposed, that showed a qualitative and quantitative agreement between the two codes, see Figures 3 and 4. Finally, a challenging problem with very low physical diffusivity and drift velocities was used to demonstrate the capabilities of the code to deal with very low diffusion values, and also the shock-capturing strategies used. This work allowed to publish a paper, see Ref. [4].

Taking advantage of the non-aligned grids, some computations with a moving equilibrium were performed. In the framework of the EoCoE collaboration with INRIA Sophia-Antipolis, the equilibrium code FEEQS.M was coupled to the HDG code to perform a simulation of a configuration transition from limiter to divertor. Figure 5 depicts the energy deposit on the limiter and the divertor during a slow and a fast transitions. This study produced another publication, see Ref. [5].

5.3 Future developments

The code will be extended next to consider the full set of transport equations with unknowns n, Γ, T_i and T_e (ionic and electronic temperature). This will confirm the capacity of the scheme of dealing with the non-linear parallel diffusion term, characterized by extreme values of the parallel diffusivity. In a future stage, the axisymmetric hypothesis will be dropped a 3D version will be implemented, allowing turbulent computations.

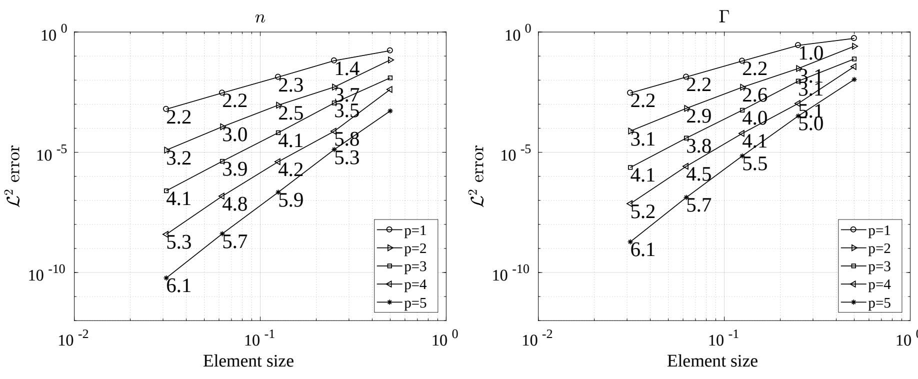
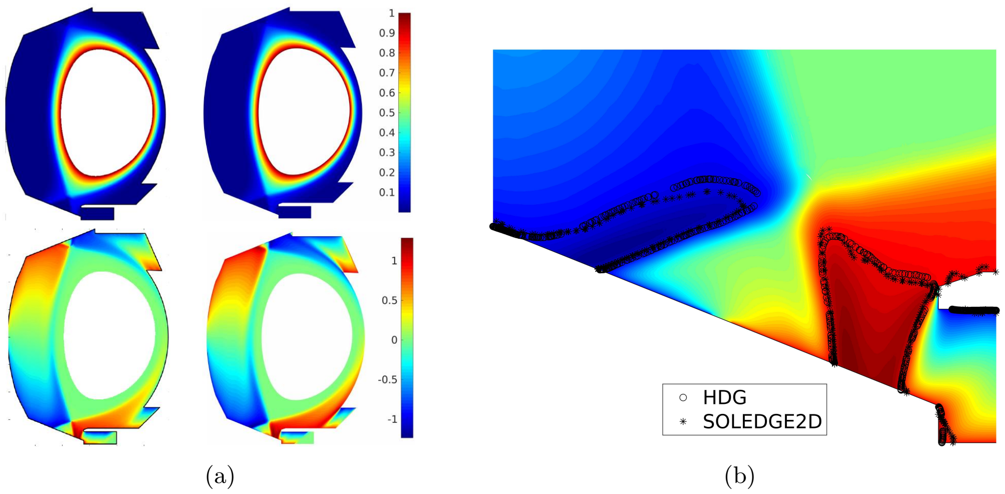


Figure 2: h-convergence tests showing the $p+1$ rate of convergence for the density (left) and parallel momentum (right). Evolution of the L^2-errors when refining meshes for 5 different polynomial degrees p.



(a) (b)

Figure 3: Solutions benchmarking in WEST. In (a) the large scale flows predicted by SOLEDGE2D (left) and the new HDG solver (right) are shown. Maps of density n (top) and parallel Mach number $M_{\|}$ (bottom). In (b) is depicted a zoom on the parallel Mach number $M_{\|}$ around the separatrix within the divertor area, with black lines showing the transition to supersonic flows predicted by the two codes.

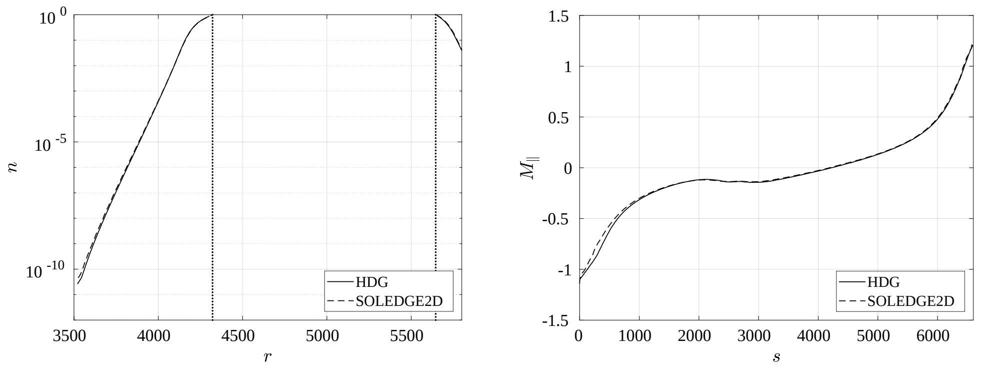


Figure 4: Solutions benchmarking in WEST. Radial density profiles at midplane (left), and parallel profiles of parallel Mach number along a magnetic field line within the SOL and close to the separatrix (right). The abscissa s defines the curvilinear coordinate along the magnetic field line. Computations are carried out with $D = \mu = 0.038$.

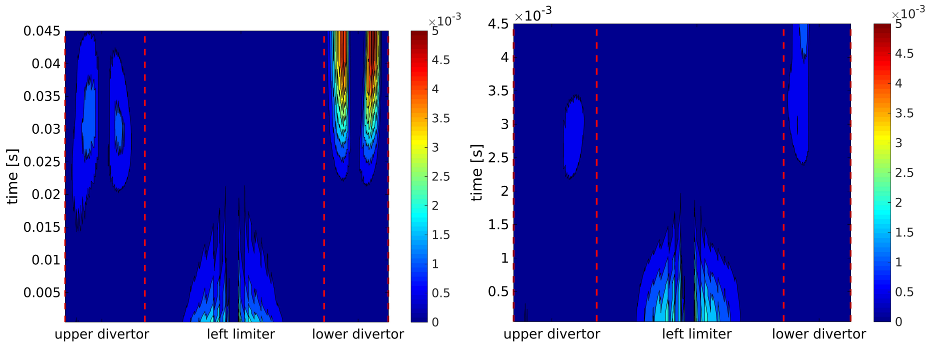


Figure 5: Evolving equilibrium: distribution of the outgoing flux of particles as a function of the time, in the slow transition simulation (left) and the fast transition simulation (right).

The coupling with the equilibrium code will be taken further to include a feedback of the transport HDG code on the equilibrium computation. This closed loop will eventually allow to obtain consistent simulation equilibrium/transport.

References

[1] S. Günter, Q. Yu, J. Krüger, and K. Lackner. Modelling of heat transport in magnetised plasmas using nonn-aligned coordinates. J. Comp. Phys., 209:354–370, 2005.

[2] Y. Notay and A. Napov. A massively parallel solver for discrete Poisson-like problems. Technical Report, 2014.

[3] S. Nakov On the design of sparse hybrid linear solvers for modern parallel architectures. PhD thesis, 2015.

[4] G. Giorgiani et al. A hybrid discontinuous Galerkin method for tokamak edge plasma simulations in global realistic geometry. Submitted, 2018.

[5] G. Giorgiani et al. A new high-order fluid solver for tokamak edge plasma transport simulations based on a magnetic-field independent discretization. Contributions to Plasma Physics, 2018, DOI: 10.1002/ctpp.201700172.

6. NanoPV

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

Context

The goal of this work is to adapt and exploit a finite element type solver from the DIOGENeS software suite [1] developed at Inria Sophia Antipolis-Méditerranée for the simulation of light absorption in complex solar cells structures involving material layers with nanoscale textured surfaces. The considered electromagnetic wave propagation solver has been adpated in order to deal accuractely and efficiently with the multiscale features of the target problem. Some specific work has also been undertaken in order to optimize the scalability of the solver. This work has been realized in interaction with researchers from the group of Urs Aeberhard at IEK-5 Photovoltaic, Forschungszentrum Julich, in relation with the objectives of workpackage WP3 of the EoCoE project.

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 method 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.

The present study is concerned with the adaptation and application of a DGTD solver for the simulation of light trapping in a multilayer solar cell structured with textured interfaces. 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.

[1]http://diogenes.inria.fr/

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 ε_∞ represents the core electrons contribution to the relative permittivity ε_r, γ 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

$$\begin{cases} \mathrm{rot}(\mathbf{H}) = \dfrac{\partial \mathbf{D}}{\partial t}, \\ \mathrm{rot}(\mathbf{E}) = -\dfrac{\partial \mathbf{B}}{\partial t}, \end{cases} \tag{1}$$

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

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

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 $\hat{\cdot}$ 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

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

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

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

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 already been considered 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 τ_i of the tetrahedral mesh writes

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

In the above system of ODEs, $\overline{\mathbf{E}}_i$ is the vector of all the degrees of freedom of $\mathbf{E}$ in τ_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].

6.1 Application to photovoltaics

We study light trapping in a silicon-based thin-film solar cell setup that consists of several randomly textured layers. The focus is on amorphous and microcrystalline silicon (a-Si:H and μc-Si:H) which belong to the family of disordered semiconductors. The main characteristics of those materials is the structural disorder, which affect in an essential way the optical and electronic properties.

Dealing with measured optical properties

Given an experimental set of points describing a permittivity function of a material, a Padé type approximation is a convenient analytical coefficient-based function to approach experimental data. The fundamental theorem of algebra allows to expand this approximation as a sum of a constant, one zero-order pole (ZOP), a set of first-order generalized poles (FOGP), and a set of second-order generalized poles (SOGP), as

$$\varepsilon_{r,g}(\omega) = \varepsilon_\infty - \frac{\sigma}{i\omega} - \sum_{l\in L_1} \frac{a_l}{i\omega - b_l} - \sum_{l\in L_2} \frac{c_l - i\omega d_l}{\omega^2 - e_l + i\omega f_l}, \tag{7}$$

where $\varepsilon_\infty, \sigma, (a_l)_{l\in L_1}, (b_l)_{l\in L_1}, (c_l)_{l\in L_2}, (d_l)_{l\in L_2}, (e_l)_{l\in L_2}, (f_l)_{l\in L_2}$ are real constants, and L_1, L_2 are non-overlapping sets of indices. The constant ε_∞ represents the permittivity at infinite frequency, and σ the conductivity. This general writing allows an important flexibility for several reasons. First, it unifies most of the common dispersion models in a single formulation. Indeed, Debye (biological tissues in the MHz regime), Drude and Drude-Lorentz (noble metals in the THz regime), retarded Drude and Drude-Lorentz (transition metals in the THz regime), but also Sellmeier's law (glass in the THz regime), are naturally included. Second, as will be shown later, it permits to fit a large range of experimental data set in a limited number of poles (thus leading to reasonable memory and CPU overheads).

In order to fit the coefficients of (7) to experimental data, various techniques can be used, such as the well-known least square method. Here, a free existing algorithm from W.L. Goffe[2] was adapted for this study. In practice, for a given model, a set of experimental data is provided to the optimization algorithm. This method demonstrated good efficiency while fitting up to 17 parameters simultaneously.

Following similar steps as for the Drude model, one derives the system of PDEs, accounting for the generalized dispersive model in time-domain

$$\left\{\begin{array}{rcl}
\dfrac{\partial \mathbf{H}}{\partial t} & = & -\nabla \times \mathbf{E}, \\
\dfrac{\partial \mathbf{E}}{\partial t} & = & \dfrac{1}{\varepsilon_\infty}\left(\nabla \times \mathbf{H} - \mathbf{J}_0 - \displaystyle\sum_{l\in L_1} \mathbf{J}_l - \sum_{l\in L_2} \mathbf{J}_l\right), \\
\mathbf{J}_0 & = & (\sigma + \displaystyle\sum_{l\in L_2} d_l)\mathbf{E}, \\
\mathbf{J}_j & = & a_l \mathbf{E} - b_l \mathbf{P}_l \quad \forall l \in L_1, \\
\dfrac{\partial \mathbf{P}_l}{\partial t} & = & \mathbf{J}_l \quad \forall l \in L_1, \\
\dfrac{\partial \mathbf{J}_l}{\partial t} & = & (c_l - d_l f_l)\mathbf{E} - f_l \mathbf{J}_l - e_l \mathbf{P}_l \quad \forall l \in L_2, \\
\dfrac{\partial \mathbf{P}_l}{\partial t} & = & d_l \mathbf{E} + \mathbf{J}_l \quad \forall l \in L_2.
\end{array}\right. \tag{8}$$

The numerical treatment of system (8) in the framework of a DGTD method is 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[3].

[2]`http://ideas.repec.org/c/wpa/wuwppr/9406001.html`
[3]http://www.meshgems.com/

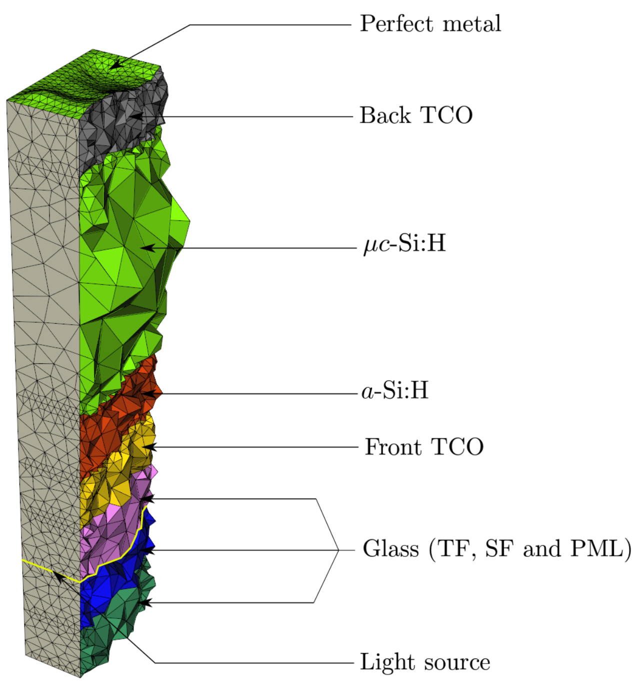


Figure 6: Geometrical model of the solar cell structure and composition of the different layers. Layer thicknesses are in the order of the wavelength of relevant sunlight.

Smoothing step

The initial layers data presented abrupt jumps in one direction of space. We started by smoothing the layers with an in-house tool (see a comparison in Fig. 7).

Building process

The building process of geoemtrical models of a cell structurr 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;
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 is not an easy task to obtain an exploitable mesh. In particular, we have to impose Periodic Boundary Conditions (PBC) that allow to simulate artificially infinite mono-directional or

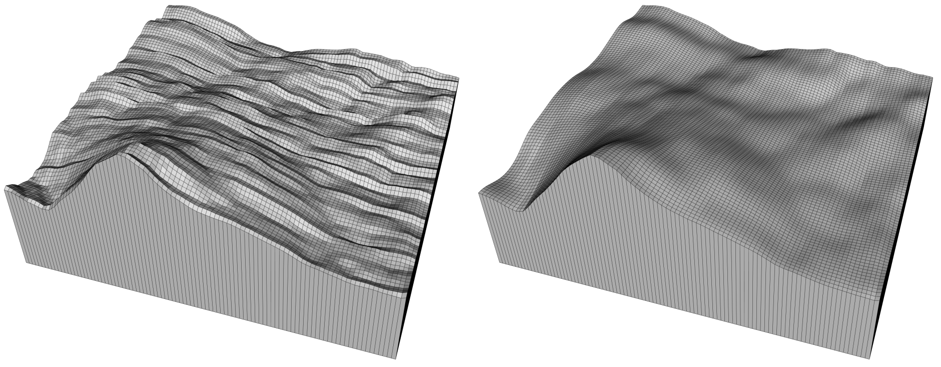

(a) Original surface. (b) Smoothed surface.

Figure 7: Smoothed and not-smoothed versions of the UcSI layer. This is a 100×100 subset of the full layer surface.

bi-directional arrays while considering only one elementary pattern. To do so, cells from a periodic boundary face are matched with their neighbors on the opposite boundary of the domain. We have two possibilities to obtained such boundaries. The first one consist of symmetrizing the mesh. The main drawback of this method is the multiplication of the domain size by 4. This is not a major issue for a small model, but for the full device this is not a feasible approach. The second option is to use a Tukey-window function on each layer to have the same 1D border and so the same lateral faces. (as done in [JLIZ15]). The disadvantage here is the loss of information on the edges of the structure as can be seen in Fig.8.

Obtained meshes

Partial views of generated meshes are shown in Fig. 8. These meshes corresponds to a $1 \times 1\,\mu\text{m}^2$ subset of the full model, which will be used for preliminary tests. The mesh in Fig. 9 is obtained for the full model using the Tukey-window approach.

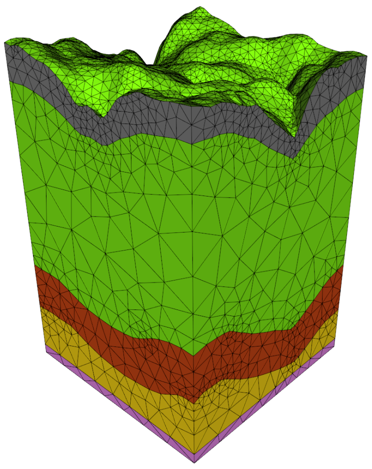
Material models

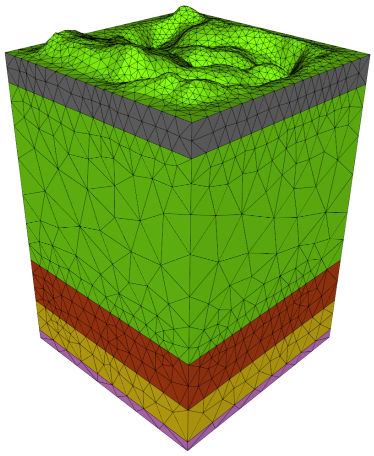
The optical properties of the different materials that constitute the considered solar cell structure have been fitted to the parameters of our generalized dispersion model, which was originally intended for metals. The obtained permittivity functions are plotted in Fig. 10. As can be seen, all materials are relatively well approximated on the range $\lambda = [300, 1300]$ nm. Regarding glass, a constant permittivity $\varepsilon_r = 2.25$ is used.

Parallelization aspects

The DGTD solver is parallelized using a classical SPMD strategy combining a partitioning of the underlying tetrahedral mesh, with a message passing programming model using the MPI standatd. The MeTiS[4] graph partitioning tool is exploited for decomposing the mesh into submeshes.

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


(a) Original model.


(b) Mirrorized version of the original model.

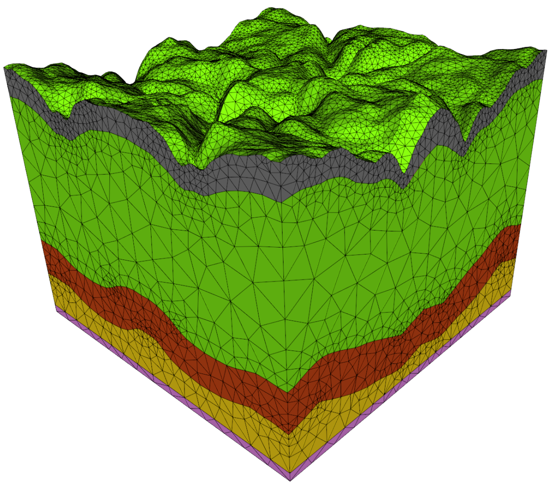

(c) Modified model after applying Tukey-window.

Figure 8: Examples of the obtained meshes. SF and PML layers are not included here.

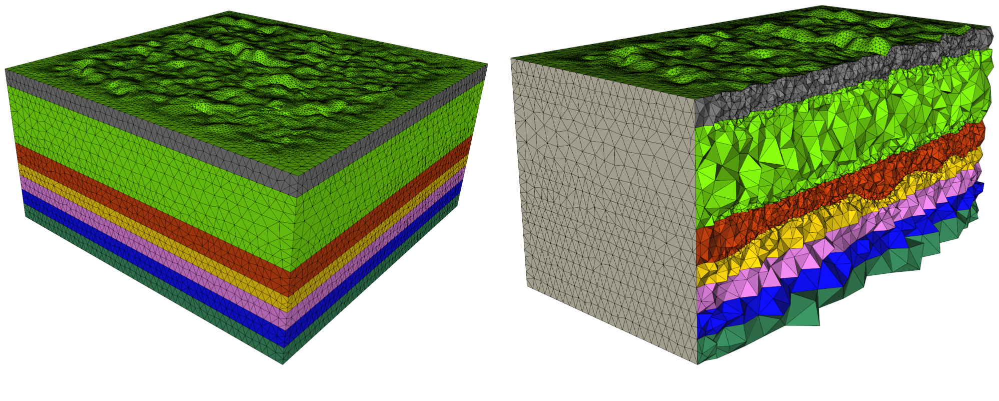
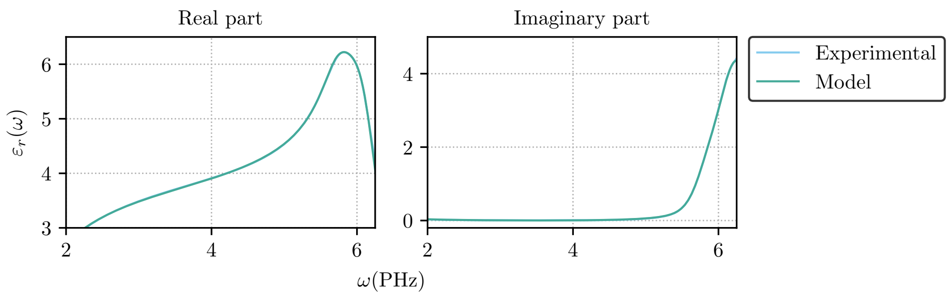
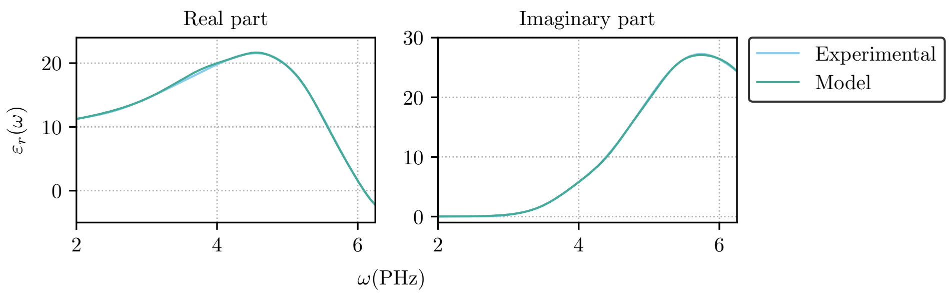


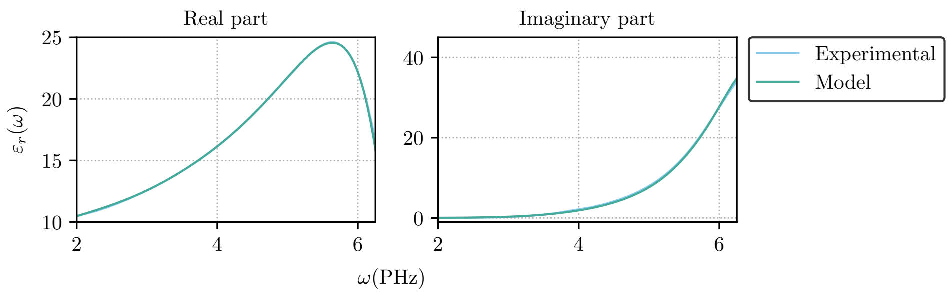
Figure 9: Examples of the obtained meshes.



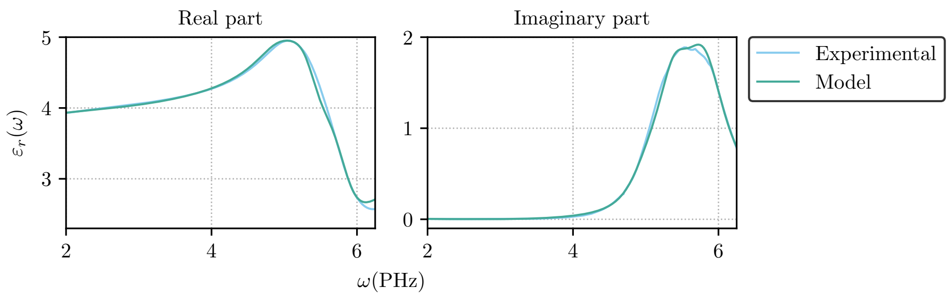
(a) Front TCO.



(b) a-Si:H.



(c) μc-Si:H.



(d) Back TCO.

Figure 10: Real and imaginary parts of the relative permittivity of front TCO, Asi-i, Ucsi-i and back TCO predicted by our dispersive model compared to experimental data.

Observable quantities

A physical quantity that is relevant for the study is the absorption in the silicon-based materials. It can be computed with a volumetric method [Viq15]. Indeed, it is possible to evaluate the ohmic losses directly inside the material. It can be shown that the power absorbed by a layer as ohmic losses is

$$\mathcal{A}_{\text{layer}}(\omega) = P_{Ohm}(\omega) = \frac{\varepsilon_0 \omega}{2} \int_{\Omega_S} \Im(\varepsilon_r(\omega)) |\widehat{\mathbf{E}}(\mathbf{r}, \omega)^2| \tag{9}$$

where Ω_S is the volume delimited by the layer. To allow for a straightforward comparison between experimental and simulation results, the external quantum efficiency (EQE) is used

$$\text{EQE}(\omega) = \frac{\mathcal{A}_{\text{layer}}(\omega)}{I_0 \cdot S_{\text{domain}}} \tag{10}$$

where I_0 is the intensity of the incident light and S_{domain} is the surface area of the domain perpendicular to the incident wave. This quantity represent the amount of the incident light which is absorbed in the layer.

Numerical and performance results

Convergence study

Numerical convergence has been assessed by considering $1 \times 1\,\mu\text{m}^2$ submodel of the solar cell structure on one hand, and the full size model on the other hand. Several tetrahedral meshes have been constructed using the procedure described in section 6.1. The characteristics of some of the meshes are summarized in Tab. 1. We plot in Fig. 11 to 12 the EQE obtained for polynomial orders ranging from 1 to 4 in the DGTD method. As can be seen, the DGTD-$\mathbb{P}_3$ and DGTD-$\mathbb{P}_4$ methods yield almost identical results when considering the submodels for the a-Si:H layer.

Mesh	Tetrahedron	$h_{\min}$ (nm)	$h_{\max}$ (nm)	$\frac{h_{\max}}{h_{\min}}$
$1 \times 1\,\mu\text{m}^2$	41387	9.9	482.6	48.7
$10 \times 10\,\mu\text{m}^2$	1151793	6.7	917.8	137.0
$10 \times 10\,\mu\text{m}^2$ homogeneous	603343	10.0	530.4	53.0

Table 1: Characteristics of the tetrahedral meshes for the $1 \times 1\,\mu\text{m}^2$ and $10 \times 10\,\mu\text{m}^2$ models.

Influence of layer surface texturing

Here we highlight the effects of texturing of the layers by comparing simulations made on the model with textured interfaces and a model with flat interfaces. In Fig. 14, we plot the EQE spectrum of the silicon-based layers for a $\mathbb{P}_4$ polynomial expansion. As can be seen, flat interfaces created cavity mode between layers which are responsible of the resonances.

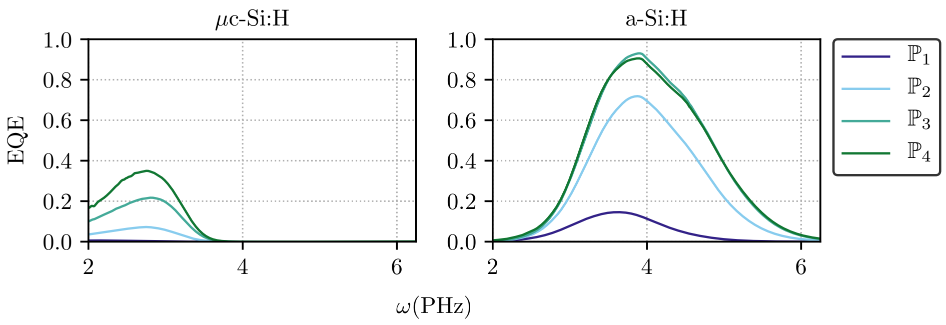


Figure 11: EQE for $1 \times 1\,\mu\text{m}^2$ submodel with polynomial orders from 1 to 4.

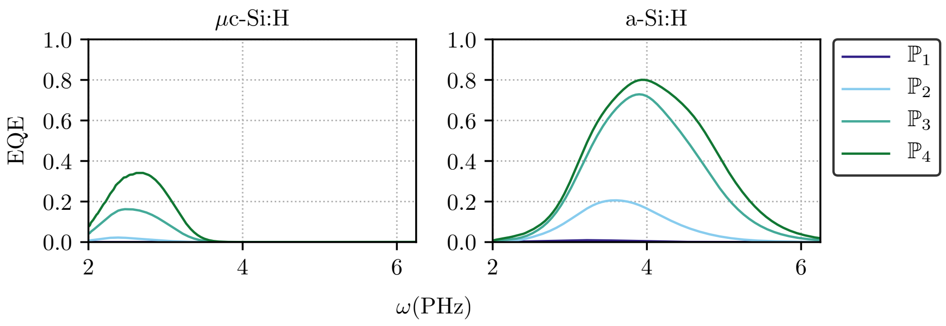


Figure 12: EQE for $10 \times 10\,\mu\text{m}^2$ model with polynomial orders from 1 to 4.

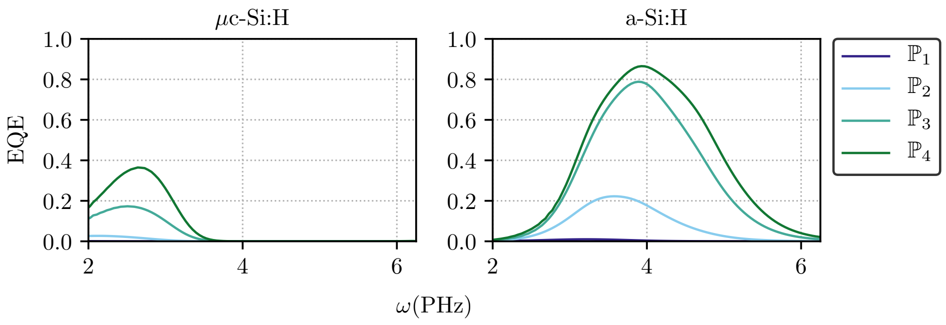


Figure 13: EQE for $10 \times 10\,\mu\text{m}^2$ homogeneous model with polynomial orders from 1 to 3.

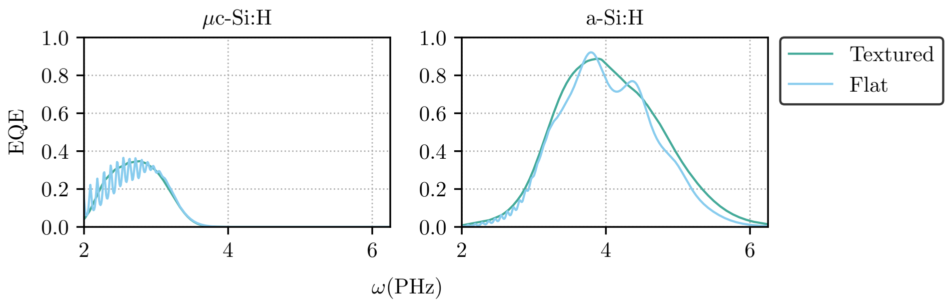


Figure 14: EQE of microcrystalline silicon (μc-Si:H) and amorphous silicon (a-Si:H) layers for textured and flat $1 \times 1\,\mu\text{m}^2$ submodel.

Comparison with FDTD simulations

A comparison with results from a FDTD simulations performed at IEK-5 is shown in Fig. 15. As can be seen, the results are in relatively good agreement.

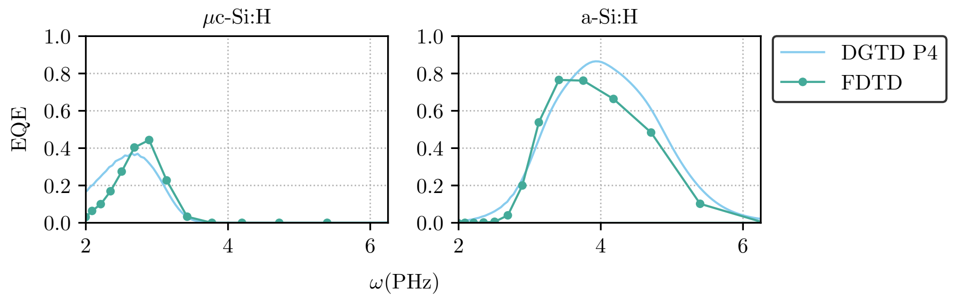


Figure 15: Comparison of EQE from simulations with the FDTD and DGTD solvers, for microcrystalline silicon (μc-Si:H) and amorphous silicon (a-Si:H) using the full model.

Shannon theorem for frequency acquisition

In order to reduce the computational overhead of computing the volumetric absorption we have exploited Shannon's theorem for sampling of frequency-dependent quantities. In fact, setting

$$\Delta t_{\text{obs}} = \frac{1}{2f_{max}},$$

allows to perfectly sample the spectrum of the $\mathcal{A}_{\text{layer}}$ operator, by evaluating Eq. 10 at certain Δt_{obs} time steps. By doing so, we can reduce the CPU time up to 400% for the full model as one can see in Tab. 2. Simulations are run on a in-house cluster system with 128 cores (16 Intel E5-2670@2.60 GHz nodes each with 8 cores).

Mesh	Order	Improved time	Original time	Gain (%)
$1 \times 1\,\mu\text{m}^2$	$\mathbb{P}_1$	12m 38s	13m 58s	10.6
	$\mathbb{P}_2$	19m 01s	23m 39s	24.4
	$\mathbb{P}_3$	46m 58s	1h 01m 08s	30.2
	$\mathbb{P}_4$	3h 25m 34s	3h 45m 06s	9.5
$10 \times 10\,\mu\text{m}^2$	$\mathbb{P}_1$	3h 02m 16s	15h 54m 14s	423.5
	$\mathbb{P}_2$	9h 29m 52s	49h 46m 51s	424.1
	$\mathbb{P}_3$	32h 43m 57s	126h 12m 00s	285.5

Table 2: CPU times with and without applying Shannon theorem.

Scalability

We have also performed a strong scalability analysis by applying the proposed DGTD solver with a tetrahedral mesh of the full solar cell model consisting of 305,265 vertices and 1,689,764 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. 3 and 4. where "Elapsed" is the elapsed time, which is used for the evalaution of the parallel speedup relatively to the first figure given for each configuration of the DGTD-$\mathbb{P}_k$ method.

In the first table, the timings include the calculation of an important observable quantity for the considered problem, which is the *volume absorption*. The computation of this quantity requires to sweep over frequencies in a target spectrum, and compute on the fly during the time evolution a discrete Fourier transform of the electrical field (i.e. for each degree of freedom of the DG approximation of the electrical field). The later computation can be performed in a fully parallel way for each element of the mesh. However, the evaluation of the volume absorption is limited to a subvolume (i.e. a layer) of the multilayer solar cell model; in the present case the aSi layer is selecetd. In our current method for partitioning the tetrahedral mesh for the SPMD parallelization of the DGTD solver, we do not take into account the computational load balancing issues raised by this localization of the computation of the volume absorption. Then, the obtained parallel speedup in Tab. 3 is suboptimal and degrades when the interpolation degree k is increased. On the contrary, when the evaluation of the volume absorption is deactivated, the parallel performances are quasi-optimal i.e. a linear speedud is observed (see Tab. 4).

Solver	# cores	Elapsed	Speedup
DGTD-$\mathbb{P}_1$	96	2681 sec	1.00 (1.0)
-	192	1365 sec	1.95 (2.0)
-	384	768 sec	3.50 (4.0)
DGTD-$\mathbb{P}_2$	96	4364 sec	1.00 (1.0)
-	192	2254 sec	1.95 (2.0)
-	384	1332 sec	3.30 (4.0)
DGTD-$\mathbb{P}_3$	192	3678 sec	1.00 (1.0)
-	384	2232 sec	1.65 (2.0)

Table 3: Strong scalability analysis of the DGTD-$\mathbb{P}_k$ solver on the Occigen system. Mesh M1 (full model) with 305,265 vertices and 1,689,764 elements. Timings for 1000 time iterations including the evaluation of the volume absorption in the aSi layer. Execution mode: 1 MPI process per core.

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

Solver	# cores	Elapsed	Speedup
DGTD-$\mathbb{P}_1$	96	584 sec	1.00 (1.0)
-	192	292 sec	2.00 (2.0)
-	384	146 sec	4.00 (4.0)
DGTD-$\mathbb{P}_2$	96	974 sec	1.00 (1.0)
-	192	490 sec	2.00 (2.0)
-	384	246 sec	3.95 (4.0)
DGTD-$\mathbb{P}_3$	192	808 sec	1.00 (1.0)
-	384	418 sec	1.95 (2.0)

Table 4: Strong scalability analysis of the DGTD-$\mathbb{P}_k$ solver on the Occigen system. Mesh M1 (full model) with 305,265 vertices and 1,689,764 elements. Timings for 1000 time iterations excluding the evaluation of the volume absorption in the aSi layer. Execution mode: 1 MPI process per core.

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.

[JLIZ15] K. Jäger, D.N.P. Linssen, O. Isabella, and M. Zeman. Ambiguities in optical simulations of nanotextured thin-film solar cells using the finite-element method. *Optics Express*, 23(19):A1060, 2015.

[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.

7. PSBLAS and MLD2P4

Contributors	Pasqua D'Ambra (National Research Council of Italy - CNR, Naples, Italy), Daniela di Serafino (University of Campania "L. Vanvitelli", Caserta, Italy), Salvatore Filippone (Cranfield University, Cranfield, UK)

7.1 Introduction

We summarize the long-term activities performed until the end of January 2018 by CNR, University of Campania "Luigi Vanvitelli" (formerly Second University of Naples) and University of Rome "Tor Vergata", within Task 2 (*Linear Algebra*) of Workpackage 1.

The activities described here were mainly motivated by the results obtained by applying the algebraic multilevel preconditioners implemented in MLD2P4 [5, 6], coupled with Krylov solvers from PSBLAS [10, 11], to linear systems made available from WP2 and WP4. Although MLD2P4 was improved during the EoCoE project, and provided satisfactory results on some EoCoE test problems, its algebraic multilevel preconditioners (see Deliverable D 1.7 on Software Technology Improvement) lost their robustness and parallel efficiency when dealing with systems arising from highly anisotropic problems.

A crucial issue was the decoupled smoothed-aggregation implemented in MLD2P4 as coarsening algorithm [19, 18]. Therefore, we directed our interest toward different algebraic coarsening algorithms. The first results along this line are described in Sections 7.2-7.3 and were also discussed in [2].

We also started some work for extending the current versions of PSBLAS and MLD2P4 with GPU plugins, specifically tailored for EoCoE applications, in order to efficiently run our solvers and preconditioners on current and future heterogeneous architectures toward exascale, including GPGPU accelerators.

7.2 Algebraic coarsening based on weighted matching

We began investigating the effectiveness, in the MLD2P4 framework, of a coarsening algorithm for symmetric positive definite (spd) matrices based on a graph matching approach. This algorithm, named *coarsening based on compatible weighted matching*, was recently proposed in [7, 8] and implemented in the C package *BootCMatch: Bootstrap AMG based on Compatible Weighted Matching* [9]. It defines a pairwise aggregation of unknowns where each pair is the result of a maximum weight matching in the matrix adjacency graph. Specifically, the aggregation scheme uses a maximum product matching in order to enhance the diagonal dominance of a matrix representing the hierarchical complement of the resulting coarse matrix, thereby improving the convergence properties of a corresponding compatible relaxation scheme. The matched nodes are aggregated to form coarse index spaces, and piecewise constant or smoothed interpolation operators are applied for the construction of a multigrid hierarchy. More aggressive coarsening can be obtained by combining multiple steps of the pairwise aggregation, which allows to reduce operator complexity of the final AMG preconditioners.

A parallel version of the maching-based coarsening algorithm was implemented in MLD2P4 by using a decoupled approach, where each parallel process performs coarsening on the part of the matrix owned by the process itself. The MLD2P4 software framework was extended in order to efficiently interface the BootCMatch functions implementing the sequential coarsening algorithm, and to combine the new functionality with the other AMG components of MLD2P4. Details on the interfacing between MLD2P4 and BootCMatch are given in [2].

Three algorithms for maximum product weighted matching were considered, all available to MLD2P4 through BootCMatch:

MC64: the algorithm implemented in the MC64 routine of the HSL library [13], which finds optimal matchings with a worst-case computational complexity $O(n(n+nnz)\log n)$, where n is the matrix dimension and nnz the number of its nonzero entries;

half-approximate: the greedy algorithm described in [17], capable of finding, with complexity $O(nnz)$, a matching whose total weight is at least half the optimal weight;

auction-type: a version of the near-optimal auction algorithm proposed in [4], implemented in the SPRAL Library as described in [14]; note that this algorithm reduces the cost of the original auction one, producing a near-optimal matching at a much lower cost than that of the (optimal) MC64.

7.3 Results on EoCoE data sets

First experiments with multilevel preconditioners using maching-based coarsening were performed on two linear systems from WP2 (*Meteorology for Energy*) and on three linear systems from WP4 (*Water for Energy*), respectively. The experiments were carried out on the yoda linux cluster, operated by the Naples Branch of the CNR Institute for High-Performace Computing and Networking. Its compute nodes consist of 2 Intel Sandy Bridge E5-2670 8-core processors and 192 GB of RAM, connected via Infiniband. Given the size and the sparsity of the linear systems, at most 64 cores, running as many parallel processes, were used; 4 cores per node were considered, according to the memory bandwidth requirements of the linear systems. PSBLAS 3.4 and MLD2P4 2.2, installed on the top of MVAPICH 2.2, were used together with a development version of BootCMatch and the version of UMFPACK available in SuiteSparse 4.5.3. The codes were compiled with the GNU 4.9.1 compiler suite.

WP2 code: Alya

The systems come from computational fluid dynamics simulations for wind farm design and management, carried out at BSC by using the HPC multi-physics simulation code Alya [20]. Specifically, the systems arise from the numerical solution of Reynolds-Averaged Navier-Stokes equations coupled with a modified $k-\varepsilon$ model. The space discretization is obtained by using stabilized finite elements, while the time integration is performed by combining a backward Euler scheme with a fractional step method, which splits the computation of the velocity and pressure fields and thus requires the solution of two linear systems at each time step. The systems considered here concern the pressure field. They have symmetric spd matrices of size 790856 and 2224476, with 20905216 and 58897774 nonzeros, respectively, and are denoted by PRESS1 and PRESS2. Their sparsity pattern is shown in Figure 16. As mentioned in Section 7.1, we did not achieve satisfactory results on these matrices by using as coarsening algorithm the classical smoothed aggregation implemented in MLD2P4.

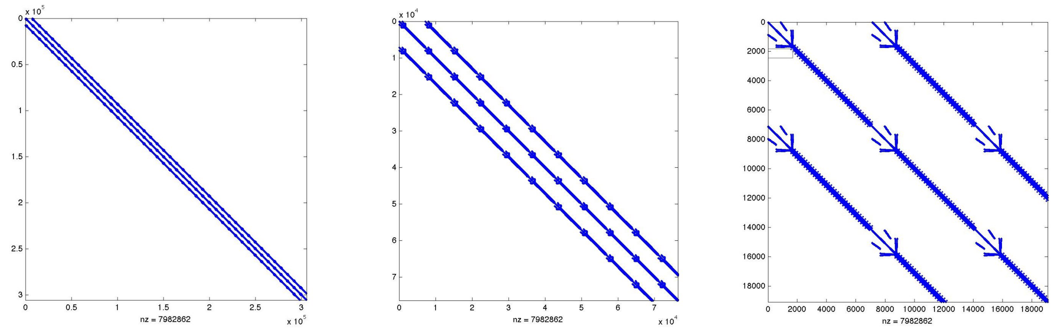

full zoom 1 zoom 2

Figure 16: Pressure matrices from wind farm simulations: sparsity pattern (full matrix and details).

PRESS1				PRESS2			
procs	iters	time	sp	procs	iters	time	sp
1	8	51.22	1.0	1	8	76.00	1.0
2	39	25.39	2.0	2	40	81.90	0.9
4	40	11.69	4.4	4	44	39.26	1.9
8	50	5.96	8.6	8	43	19.24	3.9
16	57	2.89	17.7	16	48	10.84	7.0
32	84	2.18	23.5	32	52	5.25	14.5
64	58	1.20	42.8	64	48	2.60	29.2

Table 5: Test problems PRESS1 and PRESS2: number of iterations, execution time and speedup for weighted matching based on MC64.

PRESS1 and PRESS2 were preconditioned by using a K-cycle [16] with decoupled unsmoothed double-pairwise matching-based aggregation. One block-Jacobi sweep, with ILU(0) factorization of the blocks, was applied as pre/post-smoother, and UMFPACK (`http://faculty.cse.tamu.edu/davis/suitesparse.html`) was used, through the interface provided by MLD2P4, to solve the coarsest-level system, replicated in all the processes. The PSBLAS implementation of FCG(1) [15] was chosen to solve the systems, according to the variability introduced in the preconditioner by the K-cycle. The experiments were performed using all the three matching algorithms mentioned in Section 7.2. The zero vector was used as starting guess and the preconditioned FCG iterations were stopped when the 2-norm of the residual achieved a reduction by a factor of 10^{-6}. A generalized row-block distribution of the matrices, obtained by using the Metis graph partitioner (`http://glaros.dtc.umn.edu/gkhome/metis/metis/overview`), was chosen. Among the matching algorithms, only MC64 was able to produce an effective coarsening, hence avoiding a significant increase of the number of iterations with the number of cores. Therefore, we discuss results for this case only.

In Table 5 we report the FCG iterations, the execution time (in seconds) and the speedup obtained with PRESS1 and PRESS2. The execution time includes the construction of the preconditioner and the solution of the preconditioned linear system. The preconditioned solver shows good algorithmic scalability in general; the number of iterations on a single

	MAT1			MAT2			MAT3		
np	MC64	half-app	auction	MC64	half-app	auction	MC64	half-app	auction
1	13	11	12	18	29	18	46	58	52
2	15	11	14	20	35	21	79	68	65
4	15	11	13	20	32	21	62	83	64
8	15	11	13	20	31	22	69	77	71
16	13	11	15	19	29	19	75	69	101
32	15	11	15	21	37	21	79	68	82
64	15	11	13	26	32	21	86	76	78

Table 6: Test problems MAT1, MAT2, and MAT3: number of FCG iterations for the three matching algorithms.

core is much smaller because in this case the smoother reduces to an ILU factorization. A speedup of 42.8 is achieved for PRESS1, which reduces to 29.3 for the larger matrix PRESS2; in our opinion this can be considered satisfactory, given the memory-bound nature of the computation.

WP4 code: ParFlow

The aggregation-based multilevel preconditioners were also tested on three linear systems coming from the numerical simulation of the filtration of 3D incompressible single-phase flows through anisotropic porous media, performed at the Jülich Supercomputing Centre (JSC) within WP4. The linear systems arise from the discretization of an elliptic equation with no-flow boundary conditions, modelling the pressure field, which is obtained by combining the continuity equation with Darcy's law [1]. The discretization is performed by a cell-centered finite volume scheme (two-point flux approximation) on a Cartesian grid. The systems considered here have spd matrices with size 10^6 and a classical seven-diagonal sparsity pattern, with 6940000 nonzero entries. The anisotropic permeability tensor in the elliptic equation is randomly computed from a lognormal distribution with mean 1 and three standard deviation values, i.e., 1, 2 and 3, corresponding to three systems, denoted by MAT1, MAT2 and MAT3. The systems are generated by using a Matlab code implementing the basics of reservoir simulations and can be regarded as simplified samples of systems arising in ParFlow, an integrated parallel watershed computational model for simulating surface and subsurface fluid flow, currently used at JSC.

The preconditioner used in this case differs from the previous one for the choice of the hybrid backward and forward Gauss-Seidel methods as pre-smoother and post-smoother, respectively. The remaining testing details are the same as in the previous case. The number of FCG iterations obtained using the preconditioner variants corresponding to the three matching algorithms are reported in Table 6. The corresponding execution times and speedup values are shown in Table 7.

In general, the preconditioned FCG solver shows reasonable algorithmic scalability, i.e., for all systems, the number of iterations does not vary too much with the number of processes. A larger variability in the iterations can be observed with MAT3, due to the higher anisotropy of this problem and its interaction with the decoupled aggregation

MAT1						
	MC64		half-app		auction	
np	time	sp	time	sp	time	sp
1	18.58	1.0	8.72	1.0	8.90	1.0
2	9.67	1.9	4.43	2.0	4.71	1.9
4	5.30	3.5	2.68	3.2	2.75	3.2
8	2.66	7.0	1.42	6.1	1.32	6.7
16	1.05	17.7	0.71	12.2	0.73	12.2
32	0.67	27.9	0.52	16.8	0.52	17.2
64	0.43	43.0	0.43	20.4	0.39	22.9

MAT2						
	MC64		half-app		auction	
np	time	sp	time	sp	time	sp
1	19.54	1.0	12.46	1.0	9.73	1.0
2	11.16	1.8	6.61	1.9	5.58	1.7
4	5.55	3.5	4.05	3.1	3.14	3.1
8	2.79	7.0	2.02	6.2	1.63	6.0
16	1.18	16.6	1.07	11.7	0.77	12.6
32	0.75	26.2	0.82	15.3	0.60	16.1
64	0.51	38.5	0.60	20.7	0.40	24.1

MAT3						
	MC64		half-app		auction	
np	time	sp	time	sp	time	sp
1	25.15	1.0	18.11	1.0	15.61	1.0
2	16.24	1.5	10.07	1.8	9.48	1.6
4	8.29	3.0	7.17	2.5	5.63	2.8
8	4.41	5.7	3.66	5.0	3.24	4.8
16	1.88	13.4	1.43	12.6	2.05	7.6
32	1.25	20.2	1.16	15.7	1.07	14.6
64	0.82	30.6	0.82	22.2	0.83	18.7

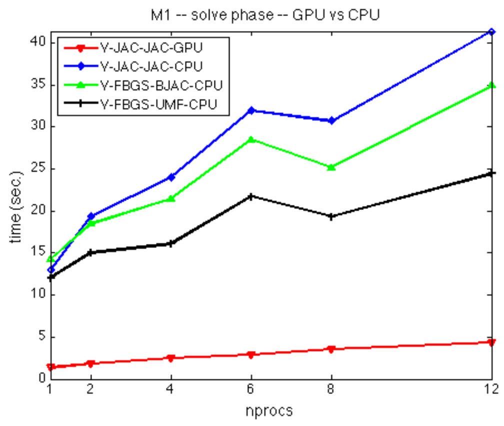
Table 7: Test problems MAT1, MAT2, and MAT3: execution time and speedup for the three matching algorithms.

strategy. None of the three matching algorithms yields preconditioners that are clearly superior in reducing the number of FCG iterations; indeed, for these systems there is no advantage in using the optimal matching algorithm implemented in MC64, since the non-optimal ones appear very competitive. The times corresponding to the half-approximation and auction algorithms are generally smaller, mainly because the time needed to build the corresponding AMG hierarchies is reduced. The speedup decreases as the anisotropy of the problem grows, because of the larger number of FCG iterations. The highest speedups are obtained with MC64, because of the larger time required by MC64 on a single core.

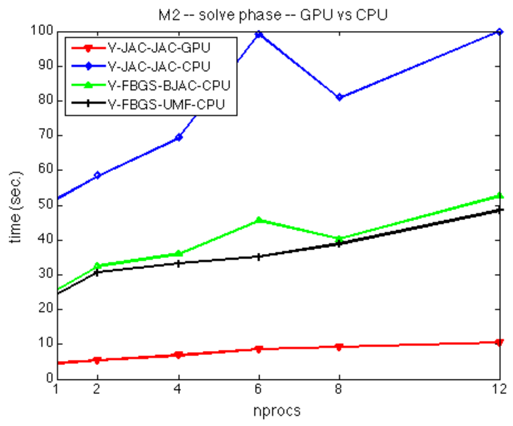
In conclusion, the results discussed so far show the potential of parallel matching-based aggregation and provide a basis for further work in this direction, such as the application of non-decoupled parallel matching algorithms.

7.4 PSBLAS and MLD2P4 GPU plugins

Two recently developed GPU plugins are now being tested and integrated. The PSBLAS plugin allows for transparent execution of linear algebra operators on NVIDIA GPGPUs, whilst the MLD2P4 plugin implements multiple approximate inverse algorithms for use in conjunction with the MLD2P4 smoothers and solvers. To use the GPU plugins it is only needed to declare and link the relevant data structures into the main application; neither the main application nor the library needs any other coding changes. Currently, the linear system solve phase only is implemented on GPUs.



MAT1



MAT2

Figure 17: GPU vs CPU execution times (secs.) on linear systems from ParFlow.

Preliminary tests to evaluate the performance on GPUs were carried out on the yoda cluster operated by ICAR-CNR, which is equipped with 2 NVIDIA K20M GPUs per node. The linear systems considered here were generated by using a parallel Fortran module, based on PSBLAS functionalities for matrix/vector data management, specifically developed to reproduce the same type of systems resulting from the aforementioned Matlab code for building test cases. This allowed us to build larger instances of the test problems. In particular, in our experiments we used instances of MAT1 and MAT2 such that number of matrix rows per process was kept equal to 4 million, achieving a system dimension of 48 million with 12 processes.

The matrices were preconditioned by using a V-cycle with decoupled smoothed aggregation. A simple point-wise Jacobi smoother and coarsest-level solver (1 and 10 sweeps, respectively) was used on GPUs. The same V-cycle with the hybrid forward/backward Gauss-Seidel smoother (1 sweep), coupled both with the distributed coarsest solver based on 10 sweeps of block-Jacobi (BJAC), with ILU(0) on the blocks, and UMFPACK on the replicated coarsest matrix, respectively, was also used on CPUs for comparison. The zero vector was used as starting guess and the preconditioned CG iterations were stopped when the 2-norm of the residual achieved a reduction by a factor of 10^{-6}. A pure row-block distribution of the matrices was applied. A strong reduction of the execution time was observed when the solve phase was run on GPUs, as shown Fig. 17.

We also began investigating the use of smoothers based on approximate inverses [3] within algebraic multilevel preconditioners on linear systems arising from ParFlow. Preconditioning by approximate inverses has a long history, and many algorithms and criteria have been proposed in the literature to build such approximations. On more traditional computing platforms, incomplete factorizations are more widely used, as they tend to be easier to build and provide better convergence properties; however they require an efficient implementation of the solution of a linear system with a sparse triangular coefficient matrix. These triangular linear systems are very difficult to solve efficiently on platforms such as GPGPUs; therefore the approximate inverses have an advantage, since they use the sparse matrix-vector product, for which very efficient kernels are available (see [12]). We run a set of tests on the matrices coming from the EoCoE applications, combining the algebraic multigrid framework we have discussed with approximate inverse smoothers, obtaining very promising performance data. The speed increase coming from the execution of the individual computational kernels on the GPU is such that a good speedup is obtained, despite an increase in the number of iterations to solution. A preliminary analysis was presented at the EoCoE workshop in the European HPC Summit 2017 in Barcelona and further experiments are in progress.

References

[1] J. E. Aarnes, T. Gimse, K.-A. Lie, *An Introduction to the Numerics of Flow in Porous Media using Matlab*, in: Geometric Modelling, Numerical Simulation, and Optimization, (G. Hasle, K.-A Lie, E. Quak, Eds.), Springer, 2007, 265–306.

[2] A. Abdullahi Hassan, P. D'Ambra, D. di Serafino, S. Filippone, *Parallel Aggregation Based on Compatible Weighted Matching for AMG*, in: Large Scale Scientific Computing (I. Lirkov and S. Margenov eds.), Lecture Notes in Computer Science, vol. 10665, Springer, 2018, 563–571.

[3] D. Bertaccini and S. Filippone, *Sparse Approximate Inverse Preconditioners on High Performance GPU Platforms*, Comput. Math. Appl., 71, 2016, 693–711.

[4] D. P. Bertsekas, *The Auction Algorithm: A Distributed Relaxation Method for the Assignment Problem*, Ann. Oper. Res., 14, 1988, 105–123.

[5] P. D'Ambra, D. di Serafino, S. Filippone, *MLD2P4: a Package of Parallel Multilevel Algebraic Domain Decomposition Preconditioners in Fortran 95*, ACM Trans. Math. Softw., 37, 2010, Art. No. 30.

[6] P. D'Ambra, D. di Serafino, S. Filippone, *MLD2P4 Rel. 2.1, User's and Reference Guide*, July, 2017. Available from `https://github.com/sfilippone/mld2p4-2/`.

[7] P. D'Ambra, P. S. Vassilevski, *Adaptive AMG with Coarsening based on Compatible Weighted Matching*, Comput. Vis. Sci., 16, 2013, 59–76.

[8] P. D'Ambra, P. S. Vassilevski, *Adaptive AMG based on Weighted Matching for Systems of Elliptic PDEs arising from Displacement and Mixed Methods*, in: Progress in Industrial Mathematics at ECMI 2014 (Russo, G., et al. Eds.), Vol. 22, Springer, 2016, 1013–1020.

[9] P. D'Ambra, S. Filippone, P. S. Vassilevski, *BootCMatch: a Software Package for Bootstrap AMG based on Graph Weighted Matching*, ACM Trans. on Math Software, 44, 2018, Art. No. 39.

[10] S. Filippone, M. Colajanni, *PSBLAS: A Library for Parallel Linear Algebra Computation on Sparse Matrices*, ACM Trans. Math. Softw., 26, 2000, 527–550.

[11] S. Filippone, A. Buttari, *Object-Oriented Techniques for Sparse Matrix Computations in Fortran 2003*, ACM Trans. on Math Software, 38, 2012, Art. No. 23.

[12] S. Filippone, V. Cardellini, D. Barbieri, A. Fanfarillo, *Sparse Matrix-Vector Multiplication on GPGPUs*, ACM Trans. Math. Softw., 43, 2016, Art. No. 30.

[13] HSL: A Collection of Fortran Codes for Large Scale Scientific Computation. Version 2013, `http://www.hsl.rl.ac.uk`.

[14] J. Hogg, J. Scott, *On the Use of Suboptimal Matchings for Scaling and Ordering Sparse Symmetric Matrices*, Numer. Linear Algebra Appl. 22, 2015, 648–663.

[15] Y. Notay, *Flexible Conjugate Gradients*, SIAM J. Sci. Comput., 22, 2000, 1444–1460.

[16] Y. Notay, *An Aggregation-based Algebraic Multigrid Method*, Electron. Trans. Numer. Anal., 37, 2010, 123–146.

[17] R. Preis, *Linear time 1/2-approximation Algorithm for Maximum Weighted Matching in General Graphs*, in: STACS 99, Lecture Notes in Computer Science 1563, Springer, 1999, 259–269.

[18] R. S. Tuminaro, C. Tong, *Parallel Smoothed Aggregation Multigrid: Aggregation Strategies on Massively Parallel Machines*, in: Proceedings of SuperComputing 2000 (J. Donnelley, Ed.), Dallas, 2000.

[19] P. Vaněk, J. Mandel and M. Brezina, *Algebraic Multigrid by Smoothed Aggregation for Second and Fourth Order Elliptic Problems*, Computing, 56, 1996, 179–196.

[20] M. Vásquez et al., *Alya: Multiphysics Engineering Simulation toward Exascale*, J. Comput. Sci. 14, 2016, 15–27.

8. IO benchmark

Contributors	Matthieu Haefele (MdlS), Sebastian Lührs (JSC), Wolfgang Frings (JSC), Agostino Funel (ENEA), Fiorenzo Ambrosino (ENEA), Maciej Brzezniak (PSNC), Krzysztof Wadowka (PSNC), Karol Sierocinski (PSNC), Tomasz Paluszkiewicz (PSNC)

Beside the computational scalability of a HPC application, its I/O behavior can massively influence the overall performance. The I/O behavior is influenced by many aspects, the implementation within the application, the file system, the network and data hardware and, as storage is normally a shared resource, other jobs running on the same cluster.

To measure the I/O behavior and to allow the test of I/O performance improvements, an I/O benchmarking activity was created as part of the EoCoE project. This task bases on three pillars:

1. Validation of the I/O behavior of the EoCoE project specific energy oriented applications to extract typical I/O pattern
2. Benchmarking of similar I/O pattern on different computing platforms using established I/O benchmarking codes
3. Interpretations and recommendations based on the benchmarking results

It is planned to present the benchmark results and the recommendations within a public available technical report.

8.1 I/O Questionnaire

All code developers which attended the performance evaluation workshops were asked to fill out a short I/O questionnaire to gather a good overview about the different I/O methods and behaviors which are currently used.

The questions are

- Used I/O libraries
- Used I/O strategy
- Typical I/O call behavior
- Do you use additional pre-processing / post-processing steps or work-flows in your production runs that are potentially I/O demanding?
- Reading type / Writing type
- Checkpointing strategy implemented; Size of a single checkpoint; Typical number of checkpoints for production runs
- Number of output files generated; Total size of files
- Is your I/O and more generally your data management strategy limited by the performance of I/O libraries and/or files system

So far 16 codes filled out the questionnaire. These are most of all codes which attended the first, second and third performance evaluation workshop.

Figure 18 and figure 19 shows two results of the questionnaire. The overall I/O library usage shows that most of the applications still use standard binary or ASCII based outputs, while higher APIs such as HDF5 or netCDF are only used by small number of applications. On the other site the datasize overview highlights a few applications which produces a large amount of data either as general output or within the check-pointing mechanism.

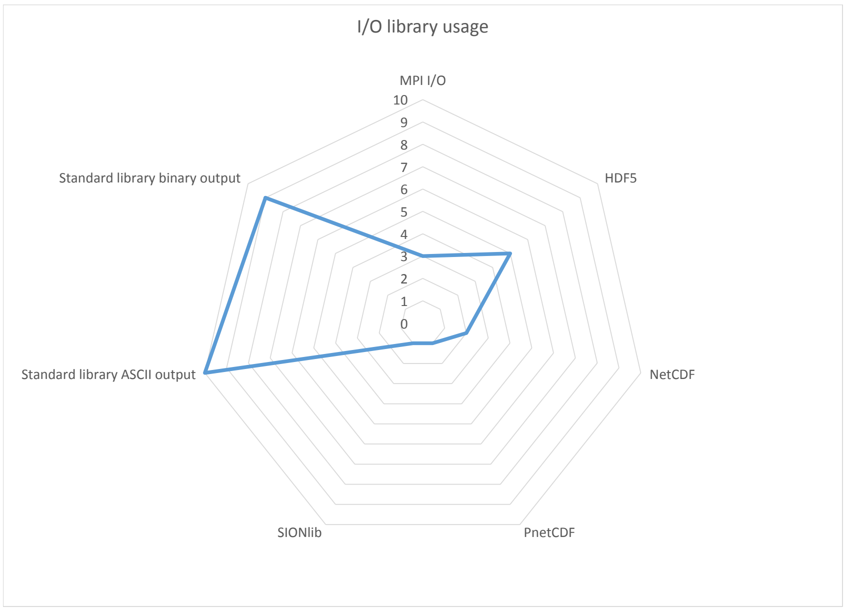


Figure 18: Distribution I/O llibrary usage of evaluated EoCoE applications.

8.2 Benchmarking

Available I/O benchmark applications are used to represent different I/O behavior on different HPC systems.

As benchmark, the popular and established HPC I/O benchmark IOR [1] was selected. This benchmark allows the direct comparison of POSIX, MPIIO, HDF5 and parallel-netcdf.

Two different configurations were specified. One to represent continuous large block I/O using different transfer sizes to analyze different buffer techniques and a direct bandwidth comparison between the different libraries. The other configuration used a striped writing approach to analyze the benefits of collective I/O operations. These two configurations together with the availability of the different APIs allows a wide variety of benchmark runs. The main idea here was to have representative benchmark cases for different types of I/O not to exactly reproduce an application I/O behavior (which is hard to match as the behavior can completely change due to different application configurations).

In addition to IOR a second benchmark with the name partest [2] was selected, which allows a direct comparison between POSIX and SIONlib I/O especially in context of check pointing.

All benchmarks are tested on different number of nodes on different HPC systems.

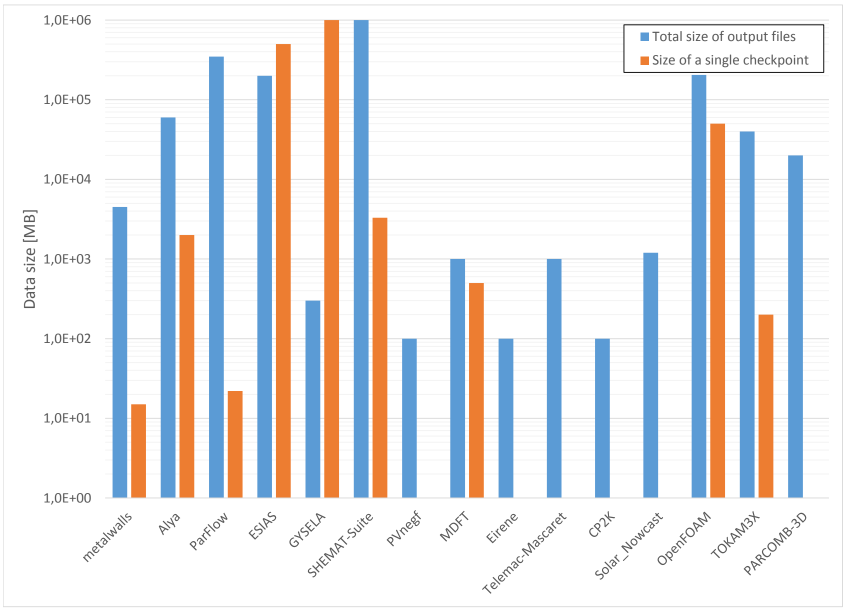


Figure 19: Overview about the total and - if available - checkpoint datasize for a typical production case of the EoCoE applications.

Three systems are used for running the benchmarks, JURECA at JSC (using the GPFS filesystem), CRESCO at ENEA (also using the GPFS filesystem) and EAGLE at PSNC (using the Lustre filesystem).

The benchmarks are executed with the help of the JUBE [3] environment, which allows a automated execution of the different benchmarking setups and system configurations.

Most of the different IOR and SIONlib benchmarks were already executed. All results are gathered in the EoCoE Gitlab to allow a final data extraction and comparison. To allow a graphical data evaluation a Python script using the matplotlib library was created.

8.3 Interpretation and technical report

The benchmarking results will allow us to create evaluation and best practice information. These results are gathered and connected to the real world applications to finally allow the code developers to test the configurations within their applications or provide hints for future code changes in context of I/O. The overall outcome of this activity will be public available in a technical report.

References

[1] https://github.com/LLNL/ior
[2] https://apps.fz-juelich.de/jsc/sionlib/docu/util page.html
[3] http://www.fz-juelich.de/jsc/jube

Acknowledgement

In case of PSNC the scientific/academic work is co-financed from financial resources for science in the years 2016-2018 granted for the realization of the international project financed by Polish Ministry of Science and Higher Education (agreement number 3543/H2020/2016/2)

In case of PSNC the scientific/academic work is co-financed from financial resources for science in the years 2016-2018 granted for the realization of the international project financed by Polish Ministry of Science and Higher Education (agreement number 3543/ H2020/2016/2)

9. Parallel Data Interface (PDI)

Contributors	Julien Bigot (Maison de la Simulation, CEA), Corentin Roussel (Maison de la Simulation, CEA), Leonardo Bautista (BSC), Kai Keller (BSC)

9.1 Context

High-performance computing (HPC) applications manipulate and store large datasets for scientific analysis, visualization purposes and/or resiliency. Multiple software libraries have been designed for interacting with the parallel file system and in some cases with intermediate storage levels. These libraries provide different level of abstraction and have been optimized for different purposes. The best I/O library for a given usage depends on multiple criteria including the purpose of the I/O, the computer architecture or the problem size. Therefore, to optimize their I/O strategies, scientists have to use multiple API's depending on the targeted execution. As a result, simulation codes contain intrusive and library dependent I/O instructions interwoven with domain instructions. We have designed a novel interface that transparently manage the I/O aspects and support multiple I/O libraries within the same execution.

9.2 Introduction

Scientific simulation codes consist of several components, such as one or several physical model implementations, post-processing code and multiple inputs and outputs. The first aspects found the basis of computational science. Inputs and outputs (I/O), on the other hand, are at the border between the concerns of computational scientists that know what data to write and for what purpose, and those who know how to write the data correctly and efficiently. I/O dedicated libraries have thus been designed to encode this technical knowledge and to provide it for the implementation in any kind of software.

The choice of a specific I/O library is however constrained by multiple aspects, such as the architecture of the supercomputer, the type and size of the data, the purpose of the write- or read-action and others. All these factors must be taken into account in order to optimize the I/O of a given application code. In order to optimize the I/O of a code, an efficient strategy is to select both the file format and the I/O library that suit best all the previous constraints. The intrusive I/O directives found inside the code affect clarity, decrease maintainability and increase development costs.

In this report, we describe the parallel data interface (PDI), a novel interface that enables users to access multiple I/O libraries through a single API. PDI is not an I/O library by itself; it only offers a unified way to access existing libraries. The API supports read- and write- operations using various I/O libraries within the same execution, and allows to switch and configure the I/O strategies without modifying the source (no re-compiling). However, it does not offer any I/O functionality on its own. It delegates the request to a dedicated library plug-in where the I/O strategy is interfaced. The range of functions and the performance of the underlying I/O libraries are not straitened

During the EoCoE project our work have focus on: **1)** Designing and implementing PDI, a new interface for accessing multiple I/O libraries without the need of modifying or recompiling the source code of the application. **2)** Adapting the GYSELA code to use either FTI/HDF5/SIONlib plain or embedded via PDI for checkpointing. We have verified the correctness of the written datasets. **3)** Performing an evaluation at scale on different supercomputers with different architectures.

In the next section we summarize the main motivations, the design, and the implementation of PDI.

9.3 Motivations

Gysela is a scientific code that models the electrostatic branch of the ion temperature gradient turbulence in tokamak efficiency [Gra15] excluding I/O and diagnostics. At the heart of Gysela is a self-consistent coupling between a 3D Poisson solver and a 5D Vlasov solver. The main data manipulated in the code from which all other values are derived is a 5D particle distribution function in phase space.

Since the complete simulation state can be derived from the 5D particle distribution function and a few scalar values (time-step, etc.) only these data are written into checkpoints. This single field usually represents in the order of one quarter of the total memory consumed. Storing it too often would represent a large overhead both in term of time and storage space. Therefore, actual results exploited by physicists, take the form of *diagnostics*: smaller arrays computed from the 5D distribution function and written to permanent storage regularly. In term of I/O these two parts of the code have different requirements.

One can further distinguish two types of checkpoints with different requirements. Preventive checkpoints, which are written during execution for fault-tolerance purpose only and job segmentation checkpoints on the other hand, which are written at the end of a job. For preventive intermediate checkpoints, one can leverage burst-buffering strategies overlapping computation and I/O, use any kind of on-disk file format or even rely on temporary storage only available for the duration of the job. For final segmentation checkpoints, however, there is no more computation to overlap with I/O. The restart can happen on a different set of nodes or even on a different machine which requires to write them to permanent storage.

Given that the best I/O strategy depends on multiple criteria (e.g., purpose of the I/O, size of the problem, hardware platform) one would like to be able to independently select the strategy to use, for each case. Choosing the best I/O strategies requires deep technical knowledge in aspects that are not the main concern of domain scientists and is thus best handled by a different person. As a matter of fact, many parallel codes have no support by I/O specialists at all and any approach that would require such a role is likely to fail.

I/O libraries claim to offer a separation of concern between application developers that use the library and library developers that encode efficient I/O strategies in the library. Nonetheless, each of them has been created to account for a specific need and provides interesting features in a specialized context.

As no single library provides an optimal choice in all possible situation, a possible solution could be to rely of multiple distinct libraries to support different strategies in the code. The library choice can then either be made at compile-time or at run-time. Compile-time choice with approaches such as `#ifdef`s in the code, means that a single library is available for each run and does not enable to use multiple libraries for different purposes during the same run (e.g., preventive versus segmentation checkpoints). The choice at run-time, with an approach such as "`switch/case`", can support mixing libraries. Without compile-time support however, this induces a hard dependency on the library that prevents compiling the code on a machine where a single of the options has not been ported yet. Thus, one would ideally have to implement both types of choices; a compile-time choice, to either depend on the library or not and a run-time choice, to select between all compiled libraries for each specific I/O.

A limitation of this approach is, however, that it increases the ratio of code, dedicated to I/O, with each new supported library or even each distinct strategy implemented using the same library. It requires cumbersome code to deal with both, run-time and compile-time choice of libraries. It leads to duplication of code, as the data to write has to be specified for each strategy implemented with distinct API's. As a result, this makes the code difficult to maintain, since multiple concerns are mixed at the same place. As the number of supported libraries grows, the maintenance cost may become unaffordable.

Since commonly none of the existing libraries can offer all the desired features at once, we propose a new interface, which permits users to enable distinct features of different libraries through one single API: the parallel data interface (PDI).

9.4 Design of the Parallel Data Interface

Acknowledging the huge amount of work that has already been done in existing libraries, we do not intend to re-develop the I/O strategies previously implemented but rather to build on top of them. PDI has therefore been designed to be a simple API that provides access to existing libraries and enables to combine them. The main goal of PDI is to separate the I/O aspects from the domain code and thus to improving the separation of concerns. PDI is a glue layer that sits in-between the implementation of these two aspects and interface both of them.

PDI has been designed in such a way that the separation of concern does not come at the expense of good properties of existing approaches. The implementation of a given I/O strategy through PDI should be as efficient and as simple (ideally less complex) than existing approaches, both from the user and I/O expert point of view. This should hold, whatever the level of complexity of the I/O strategy, from the simplest one where the implementation time is paramount, to the most complex one where the evaluation criterion is the performance on a specific hardware.

We therefore design PDI to act as a *lingua franca*, a thin layer of indirection that exposes a declarative API for the code to describe the information required for I/O and that offers the ability to implement the I/O behavior using these informations. In order to decouple both sides, we rely on a configuration file that correlates information exposed by the simulation code with that required by the I/O implementation and enables to easily select and mix the I/O strategies used for each execution. This approach brings important benefits as it improves the separation of concerns thanks to the two abstraction layers. It offers a simple API that allows a uniform code design while accessing and mixing the underlying I/O libraries.

The API limits itself to the transmission of information (required by the I/O implementation) that can only be provided during execution. Information that is known statically is expressed in the configuration file. The only elements that have to be described through the API are therefore: **1)** the buffers that contain data with their address in memory and the layout of their content, **2)** the time period along execution when these buffers are accessible either to be read or written. In addition, the API handles the transmission of control flow from the code to the library through an event system. Events are either generated explicitly by the code or generated implicitly when a buffer is made available or just before it becomes unavailable.

The data layout is often at least partially fixed, only some of its parameters vary from one execution to the other (*e.g.* the size of an array). We therefore support the description of this layout in the configuration file so as not to uselessly clutter the application code. The value of parameters that are only known during the execution can be extracted from the content of buffers exposed by the code.

Implementing an I/O strategy is done by catching the control flow in reaction to a event emitted by the simulation code and using one ore more of the exposed buffers. The name of the events and buffers to use come from the configuration file, ensuring a weak coupling between both side. Two levels of API are offered. A low level API enables to react to any event and to access the internal PDI data structures where all currently exposed buffers are stored. A higher level API enables to call user-defined functions to which specific buffers are transmitted in reaction to well specified events.

When using the low-level API, it is the responsibility of the I/O code implementation to access the configuration file to determine the events and buffers to use. This API is well suited for the development of plugins that require a somewhat complex configuration because they are intended to be reused in multiple codes. This is typically the case when interfacing I/O libraries with declarative API's close to that of PDI where options in the configuration file are enough to match the API's.

User can implement their own I/O strategies that can be interfaced with PDI. When using user-defined functions, the name of the events and buffers passed to the function are specified in the configuration file in a generic way. The function itself does neither have access to the configuration file content nor to the list of shared buffers.

This approach is less flexible but much easier to implement. It is well suited when a specific code has to be written to use a given I/O library in a given simulation code as is often the case with libraries with imperative API's. It can be used to provide additional instructions that complement but are distinct from the library features.

In order to decouple this I/O implementation code both from PDI and from the simulation code, it is defined in dedicated object files that can either be loaded statically or dynamically (a plugin system). This means that PDI does not depend on any I/O library, only its plugins do. This also simplifies changing strategy from one execution to the other as the plugins to load are specified in the configuration file.

To summarize, PDI offers a declarative API for simulation codes to expose information required by the implementation of I/O strategies. The I/O strategies are encapsulated inside plugins that access the exposed information. A week coupling mechanism enables to connect both sides through a configuration file. This can be understood as an application of aspect oriented programing (AOP) to the domain of I/O in HPC. The locations in the simulation code where events are emitted are the *joint points* of AOP. The I/O behavior encapsulated in the plugins are the *advices* of AOP. The configuration file specifies which behavior to associate at which location and constitute the *pointcuts* of AOP.

```
enum PDI_inout_t { PDI_IN=1, PDI_OUT=2, PDI_INOUT=3 };

PDI_status_t PDI_init(PC_tree_t conf, MPI_Comm *world);
PDI_status_t PDI_finalize();

PDI_errhandler_t PDI_errhandler(PDI_errhandler_t handler);

PDI_status_t PDI_event(const char *event);

PDI_status_t PDI_share(const char *name, void *data, PDI_inout_t access);
PDI_status_t PDI_access(const char *name, void **data, PDI_inout_t access);
PDI_status_t PDI_release(const char *name);
PDI_status_t PDI_reclaim(const char *name);
```

Listing 1: The PDI public API

```
PDI_status_t PDI_export(const char *name, void *data);
PDI_status_t PDI_expose(const char *name, void *data);
PDI_status_t PDI_import(const char *name, void *data);
PDI_status_t PDI_exchange(const char *name, void *data);

PDI_status_t PDI_transaction_begin(const char *name);
PDI_status_t PDI_transaction_end();
```

Listing 2: Simplified PDI API for buffer exposing

9.5 PDI Implementation

PDI is freely and publicly available[5] under a BSD license. It is written in C and offers a C API with Fortran bindings to the simulation code. This covers uses from C, C++ and Fortran, the three most widespread languages in the HPC community. We present the C flavor in this section but the Fortran binding offers the exact same interface. The plugin API is currently limited to C but bindings for other languages (*e.g.* LUA, Python) are planned.

The simulation code API contain functions to initialize and finalize the library, change the error handling behavior, emit events and expose buffers as presented in Listings 1. The initialization function takes the library configuration (a reference to the content of a YAML [BKEN09] file) and the world MPI communicator that it can modify to exclude ranks underlying libraries reserve for I/O purpose. The error handling function enables to replace the callback invoked when an error occurs. The event function takes a character string as parameter that identifies the event to emit.

The most interesting functions of this API are however the buffer sharing functions. They support sharing a buffer with PDI identified by a `name` character string and with a specified `access` direction specifying that information flows either to PDI (`PDI_OUT`, read-only share), from PDI (`PDI_IN`, write-only share) or in both directions (`PDI_INOUT`.) The `PDI_share` and `PDI_access` function start a buffer sharing section while the `PDI_release` or `PDI_reclaim` function end it. `PDI_share` is used for a buffer whose memory was previously owned by the user code while `PDI_access` is used to access a buffer previously unknown to the user code. Reciprocally, `PDI_reclaim` returns the memory responsibility to the user code while `PDI_release` releases it to PDI.

In a typical code, the buffers are however typically shared for a brief period of time between

[5]`https://gitlab.maisondelasimulation.fr/jbigot/pdi`

```
data:
  my_array: { sizes: [$N,$N], type: double }
metadata:
  N:  int  # data id and type
  it: int
plugins:
  declh5: # plug-in name
    outputs:
      # data to write, dataset and file name
      my_array: { var: array2D, file: example_$it.h5,
        # condition to write
        select: ($it>0) && ($it%10) }
```

Listing 3: Example of PDI configuration file

two access by the code. The previously introduced API requires two lines of code to do that. The API presented in Listing 2 simplifies this case. Its four first functions define a buffer sharing section that lasts during the function execution only. The functions differ in terms of access mode for the shared buffer: `share(OUT)` + `release` for `PDI_export`; `share(OUT)` + `reclaim` for `PDI_expose`; `share(IN)` + `reclaim` for `PDI_import`; and, `share(INOUT)` + `release` for `PDI_exchange`.

This API has the disadvantage that it does not enable to access multiple buffers at a time in plugins. Each buffer sharing section ends before the next one starts. The two transaction functions solve this. All sharing sections enclosed between calls to these functions have their end delayed until the the transaction ends. This effectively supports sharing of multiple buffers together. The transaction functions also emit a named event after all buffers have been shared and before their sharing section ends.

At the heart of PDI is a list of currently shared buffers. Each shared buffer has a memory address, a name, an access and memory mode and a content data type. The access mode specifies whether the buffer is accessible for reading or writing and the memory mode specifies whose responsibility it is to deallocate the buffer memory. The content data type is specified using a type system very similar to that of MPI and is extracted from the YAML configuration file.

The `data` section of the configuration file (example in Listing 3) contains an entry for each buffer, specifying its type. The type can be a scalar, array or record type. Scalar types include all the native integer and floating point of Fortran and C (including boolean or character types.) Array types are specified by a content type, a number of dimensions and a size for each dimension. They support the situation where the array is embedded in a larger buffer with the buffer size and shift specified for each dimension. Record types are specified by a list of typed and named fields with specific memory displacement based on the record address.

The types can be fully described in the YAML file, but this makes them completely static and prevents the size of arrays to change at execution for example. Any value in a type specification can therefore also be extracted from the content of an exposed buffer using a dollar syntax similar to that of bash for example. The syntax supports array indexing and record field access. For the content of a buffer to be accessible this way, it does however needs to be specified in the `metadata` section of the YAML file instead of its `data` section. When a metadata buffer is exposed, its content is cached by PDI to ensure that it can be accessed at any time including outside its sharing section.

```
main_comm = MPI_COMM_WORLD
call PDI_init(PDI_subtree, main_comm)
call PDI_transaction_begin("checkpt")
ptr_int=> N; call PDI_expose("N" ,ptr_int)
ptr_int=> iter;  call PDI_expose("it",ptr_int)
call PDI_expose("my_array",ptr_A)
call PDI_transaction_end()
call PDI_finalize()
```

Listing 4: Example of PDI API usage

The plugins to load are specified in the `plugins` section of the configuration file. Each plugin is loaded statically if linked with the application and dynamically otherwise. A plugin defines five function: an initialization function, a finalization function and three event handling functions. The event handling functions are called whenever one of the three types of PDI event occurs, just after a buffer becomes available, just before it becomes unavailable and when a named event is emitted.

The plugins can access the configuration content and the buffer repository. Configuration specific to a given plugin is typically specified under this plugin in the `plugins` section of the YAML file. The YAML file can however also contain configuration used by plugins in any section. It can for example contain additional information in a buffer description.

We currently have developed three plugins. The *FTI* plugin interfaces the declarative FTI library, the *decl'H5* plugin interfaces a declarative interface built on top of HDF5 and the *usercode* plugin supports user written code as specified in Section 9.4. A plugin interfacing a declarative version of SIONlib is also available in the repository, but most imperative libraries are best accessed through the *usercode* plugin.

Let us now present an example to show how PDI usage works in practice. Listing 4 shows the use of the Fortran API to expose to PDI two integers: `N` and `it`, and an array of dimension `N`×`N`, `my_array`. The configuration file for this example is the one presented in Listing 3.

When the `PDI_init` function is called, the configuration file is parsed and the *decl'H5* plugin is loaded. This plugin initialization function is called and analyzes its part of the configuration to identify the events to which it should react. No plugin modifies the provided MPI communicator that is therefore returned unchanged. A transaction is then started in which three buffers are exposed: `N`, `it` and `my_array`. The *decl'H5* plugin is notified of each of these events but reacts to none. The transaction is then closed that triggers a named event to which the *decl'H5* plugin does not react as well as three end of sharing section events, one for each buffer. The *decl'H5* reacts to the end of the `my_array` sharing since this buffer is identified in the configuration file. It evaluates the value of the `select` clause and if nonzero writes the buffer content in a dataset whose name is provided by the `var` value ("array2D") to a HDF5 file whose name is provided by the `file` value ("example$it.h5".)

9.6 Summary & Conclusions

The parallel Data interface, abbreviated PDI, is a novel interface that allows to separate most of the I/O concerns from the application code. PDI transparently manages the I/O aspects that are provided by external libraries or user codes. It does not impose any limitation on the underlying I/O aspects and decreases the programming effort required to perform and adapt I/O operations for different machines. Moreover we have demonstrated[6] that, thanks to PDI, scientists can use multiple I/O libraries within the same execution by simply changing a configuration file and without the need of modifying or recompiling the source code of the application.

Currently, PDI supports FTI, HDF5 libraries and a SIONlib plug-in has been finalized. Other I/O libraries (XIOS for instance) and other use cases are considered, including but not restricted to, in-situ visualization, scientific workflows.

References

[BKEN09] Oren Ben-Kiki, Clark Evans, and Ingy dot Net. *YAML Ain't Markup Language (YAML) Version 1.2, 3rd edition.* No Starch Press, 2009.

[Gra15] Virginie Grandgirard. High-Q club: Highest scaling codes on JUQUEEN – GYSELA: GYrokinetic SEmi-LAgrangian code for plasma turbulence simulations. online, March 2015.

[6]A paper has been submitted to IEEE Cluster 2017 with this work.