Computer Physics Communications 247 (2020) 106912

Contents lists available at ScienceDirect

Computer Physics Communications

journal homepage: www.elsevier.com/locate/cpc

On architecture and performance of adaptive mesh refinement in an electrostatics Particle-In-Cell code

Matthias Frey [*], Andreas Adelmann, Uldis Locans

Paul Scherrer Institut, CH-5232 Villigen, Switzerland

ARTICLE INFO

Article history:
Received 6 June 2019
Received in revised form 6 September 2019
Accepted 9 September 2019
Available online 23 September 2019

Keywords:
Multigrid Poisson solver
Adaptive mesh refinement
Hardware portability
Particle-In-Cell
Neighbouring bunches
High intensity cyclotrons

ABSTRACT

This article presents a hardware architecture independent implementation of an adaptive mesh refinement Poisson solver that is integrated into the electrostatic Particle-In-Cell beam dynamics code OPAL. The Poisson solver is solely based on second generation TRILINOS packages to ensure the desired hardware portability. Based on the massively parallel framework AMREX, formerly known as BOXLIB, the new adaptive mesh refinement interface provides several refinement policies in order to enable precise large-scale neighbouring bunch simulations in high intensity cyclotrons. The solver is validated with a built-in multigrid solver of AMREX and a test problem with analytical solution. The parallel scalability is presented as well as an example of a neighbouring bunch simulation that covers the scale of the later anticipated physics simulation.

1. Introduction

In todays state-of-the-art beam dynamics codes the well-known Particle-In-Cell (PIC) [1] technique has become indispensable. In contrast to the direct summation, where the force on a macro particle is obtained by the superposition of the forces due to all others, PIC models discretise a domain and deposit the charge of each macro particle onto a mesh in order to evaluate Coulomb's repulsion. In combination with the efficient parallelisation of such space-charge solvers using MPI (Message Passing Interface) or accelerators such as GPU (Graphics Processing Unit) and the MIC (Many Integrated Core) architecture, e.g. in [2], large-scale simulations were enabled that are more realistic. Nevertheless, multi-bunch simulations of high intensity accelerators such as cyclotrons require fine meshes in order to resolve the non-linear effects in the evolution of the beams due to space-charge. A remedy to increase the resolution, reduce the computational effort and also memory consumption is adaptive mesh refinement (AMR) [3,4]. In the context of Vlasov–Poisson problems, AMR was applied by [5] using the Eulerian description for the coordinate and velocity space. Examples for a Lagrangian formulation are the Unified Flow Solver (UFS) framework [6] and WarpX [7].

The diversity of today's computer architectures and the fast increase of emerging high performance computing technologies have shown that it is getting more and more infeasible to design a scientific software to one specific hardware only. It is therefore obvious that recent source code developments reveal a trend towards architecture independent programming where the backend kernels exhibit the hardware-specific implementation. Examples are the second generation TRILINOS [8] packages that are built on top of the *Kokkos* library [9,10].

In this article the new AMR capability of the particle accelerator library OPAL (Object-Oriented Particle Accelerator Library) [11] using AMREX [12] is presented, as well as the built-in adaptive multigrid solver based on the algorithm in [13] and the second generation TRILINOS packages *Tpetra* [14], *Amesos2* and *Belos* [15], *MueLu* [16,17] and *Ifpack2* [18]. The new implementation was benchmarked with the Poisson multigrid solver of AMREX and the analytical example of a uniformly charged sphere.

The new AMR feature of OPAL will enable to study neighbouring bunch effects as they occur in high intensity cyclotrons due to the low turn separation in more detail. Previous investigations such as [19] for the PSI (Paul Scherrer Institut) Ring cyclotron have already shown their existence but the PIC model was limited in resolution due to the high memory needs. It is hoped that the use of AMR will reduce the memory consumption for the mesh by decreasing the resolution in regions of void while maintaining or even increasing the grid point density at locations of interest in order to resolve the neighbouring bunch interactions more precisely. In [19] was shown that the interaction of neighbouring bunches leads to an increase at the tails of a particle distribution (i.e. increase of the number of halo particles) that usually causes particle losses and therefore an activation of the machine. Thus, it is essential to quantify this effect more precisely in order to do

* Corresponding author.
E-mail addresses: matthias.frey@psi.ch (M. Frey), andreas.adelmann@psi.ch (A. Adelmann).

https://doi.org/10.1016/j.cpc.2019.106912

predictions on further machine developments with higher beam current.

Besides a short introduction to OPAL in Section 2 and AMReX in Section 3, Section 4 discusses the AMR interface in OPAL. Section 5 explains the multigrid algorithm and its implementation using Trilinos with validation in Section 6. A comparison of neighbouring bunch simulations with either AMR turned on or off is shown in Section 7. The performance of the Poisson solver is discussed in Section 8. In the last section are conclusions and outlook.

2. The OPAL library

The Object-Oriented Parallel Accelerator Library (OPAL) [11] is an electrostatic PIC (ES-PIC) beam dynamics code for large-scale particle accelerator simulations. Due to the general design its application ranges from high intensity cyclotrons to low intensity proton therapy beamlines [20] with negligible space-charge. Besides the default FFT (Fast Fourier Transform) Poisson solver for periodic and open boundary problems the built-in SAAMG (Smoothed Aggregation Algebraic Multigrid) solver enables to simulate accelerators with arbitrary beam pipe shapes [21]. The time integration relies on the second order Leapfrog, the fourth order Runge–Kutta (RK-4) or a multiple stepping Boris–Buneman method [22].

In beam dynamics the evolution of the density function $f(\mathbf{x}, \mathbf{p}, t)$ in time t of the charged particle distribution in phase space $(\mathbf{x}, \mathbf{p}) \in \mathbb{R}^6$ due to electromagnetic fields $\mathbf{E}(\mathbf{x}, t)$ and $\mathbf{B}(\mathbf{x}, t)$ is described by the Vlasov (or collisionless Boltzmann) equation

$$\begin{aligned}\frac{df(\mathbf{x},\mathbf{p},t)}{dt} &= \gamma m_0 \frac{\partial f}{\partial t} + \mathbf{p}\cdot\nabla_{\mathbf{x}} f \\ &\quad + \frac{q}{\gamma m_0^2}\left(\gamma m_0 \mathbf{E}(\mathbf{x},t) + \mathbf{p}\times\mathbf{B}(\mathbf{x},t)\right)\cdot\nabla_{\mathbf{p}} f = 0, \end{aligned} \tag{1}$$

with particle charge q and rest mass m_0. The relativistic momentum $\mathbf{p} = \gamma m_0 \mathbf{v}$ with Lorentz factor γ and particle velocity $\mathbf{v}$ is used together with the coordinate $\mathbf{x}$ to specify the state of a particle in the 6D phase space. Both, the electric and magnetic field, in Eq. (1) are a sum of an external and internal, i.e. space-charge, contribution

$$\mathbf{E}(\mathbf{x},t) = \mathbf{E}_{sc}(\mathbf{x},t) + \mathbf{E}_{ext}(\mathbf{x},t),$$
$$\mathbf{B}(\mathbf{x},t) = \mathbf{B}_{sc}(\mathbf{x},t) + \mathbf{B}_{ext}(\mathbf{x},t).$$

The external fields are given by RF-cavities and by the magnetic field of the machine. In order to evaluate the electric self-field the beam is Lorentz transformed into its rest frame where the magnetic field induced by the motion of the particles is negligible. Thus, the electric self-field is fully described by the electrostatic potential $\phi(\mathbf{x}, t)$, i.e.

$$\mathbf{E}_{sc}(\mathbf{x},t) = -\nabla\phi(\mathbf{x},t)$$

that is computed by Poisson's equation

$$\Delta\phi(\mathbf{x},t) = -\frac{\rho(\mathbf{x},t)}{\varepsilon_0},$$

with charge density ρ and vacuum permittivity ε_0. The magnetic self-field is afterwards restored by the inverse Lorentz transform. This quasi-static approximation is known as Vlasov–Poisson equation.

3. The AMReX library

The AMReX library [12] is a descendant of the parallel block-structured adaptive mesh refinement code named BoxLib. It is C++ based with an optional Fortran90 interface. Each level is distributed independently among MPI-processes in order to ensure load balancing. The owned data is located either at nodes, faces, edges or centres of cells where the latter description is used in the OPAL-AMR implementation.

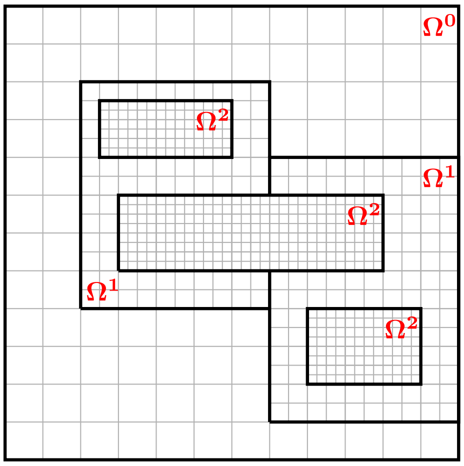


Fig. 1. Sketch of a block-structured mesh refinement of a Cartesian grid Ω^0 in 2D with AMReX. Fine levels denoted by Ω^1 and Ω^2 may span multiple coarser grids as indicated. At interfaces among grids of same level ghost cells allow exchanging data.

In order to generate a level $l+1$ each cell of the underlying coarser level l has to be marked to get refined or not according to a user-defined criterion. In electrostatic problems natural choices are for example the charge density, the potential strength or the electric field (cf. Section 4.2). Subsequent AMR levels satisfy the relation

$$h_w^{l+1} = \frac{h_w^l}{r_w} \quad \forall w \in [x, y, z], \tag{2}$$

where $r_w \in \mathbb{N}\setminus\{0\}$ is called the refinement ratio and h_w^l specifies the mesh spacing of level l in direction of w. A sketch of a refined mesh is given in Fig. 1. By definition, the coarsest level ($l = 0$) covers the full domain $\Omega = \Omega^0$ whereas a fine level is defined by patches that may overlap several coarser grids. In general, for a level $l > 0$ with n grids g_i following holds

$$\Omega^l = \left(\bigcup_{i=0}^{n-1} g_i^l\right) \subset \Omega^{l-1},$$
$$g_i^l \cap g_j^l = \emptyset \quad \forall i, j \in \{0, 1, \ldots, n-1\} \text{ and } i \neq j.$$

Although neighbouring grids are not allowed to overlap they exchange data at interfaces via ghost cells.

4. Adaptive mesh refinement in the OPAL library

In order to allow AMR and uniform mesh PIC algorithms, the interface in OPAL is implemented in a lightweight fashion where an AMR library is used as a black box. The AMR functionality is provided by concrete implementations of the abstract base class that defines common requirements on AMR libraries such as refinement strategies and mesh update functions. The actual AMR implementation is therefore hidden allowing multiple AMR dependencies.

In AMR mode the allocation of work among MPI-processes is controlled by AMReX. In contrast to OPAL where load balancing is optimised w.r.t. the macro particles, AMReX aims to achieve a uniform workload of grid operations. These two parallelisation paradigms are contradictory and cause additional MPI-communication for every PIC operation if both, grids and particles, are kept evenly distributed among the MPI-processes. In

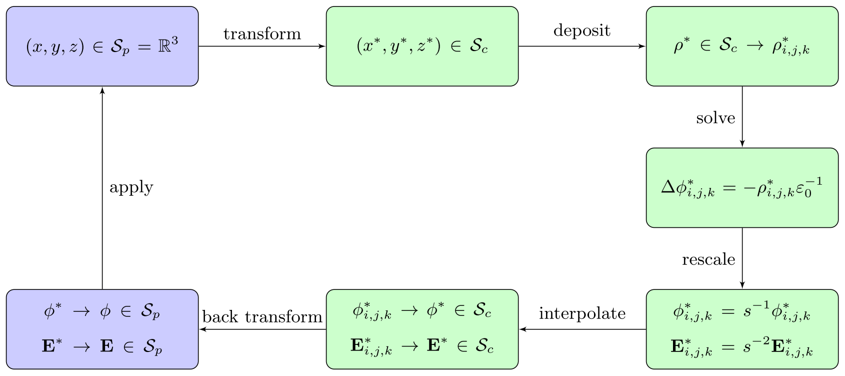

$(x, y, z) \in \mathcal{S}_p = \mathbb{R}^3$
transform
$(x^*, y^*, z^*) \in \mathcal{S}_c$
deposit
$\rho^* \in \mathcal{S}_c \to \rho^*_{i,j,k}$
solve
$\Delta\phi^*_{i,j,k} = -\rho^*_{i,j,k}\varepsilon_0^{-1}$
rescale
$\phi^*_{i,j,k} = s^{-1}\phi^*_{i,j,k}$
$\mathbf{E}^*_{i,j,k} = s^{-2}\mathbf{E}^*_{i,j,k}$
interpolate
$\phi^*_{i,j,k} \to \phi^* \in \mathcal{S}_c$
$\mathbf{E}^*_{i,j,k} \to \mathbf{E}^* \in \mathcal{S}_c$
back transform
$\phi^* \to \phi \in \mathcal{S}_p$
$\mathbf{E}^* \to \mathbf{E} \in \mathcal{S}_p$
apply

Fig. 2. Workflow of the space-charge calculation. Poisson's equation is solved in the computation domain and rescaled afterwards. All steps in particle space $\mathcal{S}_p$ and computation space $\mathcal{S}_c$ are marked in blue and green, respectively. The mapping of the particle coordinates in space $\mathcal{S}_p$ to $\mathcal{S}_c$ involves also the Lorentz transform. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

order to reduce communication effort at the expense of possible particle load imbalances the developed AMR interface distributes the particles according to their grids. For this purpose a new particle layout manager is created that stores further AMR specific attributes, i.e. the level and the grid a particle lives on.

A peculiarity of the PIC model in OPAL is the adjustment of the grid Ω^0 (cf. Fig. 1) to the particle bunch. The mesh that is co-moving with the macro particles adapts dynamically to the dimension of the bunch in rest frame, keeping the number of grid points per dimension constant, with the consequence of a constantly changing grid spacing. In longitudinal direction, i.e. the direction of travel, this change includes the correction of relativistic length contraction in laboratory frame. In AMR mode instead the macro particles are mapped to a fixed domain since the problem geometry has to be predefined in AMReX. This linear transformation includes the Lorentz transform of the particles. Adaptive mesh refinement, particle partitioning and the calculation of the electrostatic potential (cf. Section 4.1) are carried out there.

Spurious self-forces on particles close by coarse–fine grid interfaces that occur in AMR due to image charges are corrected by buffer cells as described in [23]. Another solution as depicted in [24] would be the modification of the charge deposition algorithm using a convolution of Green's function for particles near a refinement boundary.

4.1. Domain transform

In order to prevent particles leaving the predefined domain of the mesh where the AMR hierarchy is built, they are mapped into a computation space denoted by $\mathcal{S}_c$ for the evaluation of Poisson's equation, the repartition of the particles to MPI-processes and the mesh refinement. Therefore, the geometry can be kept at $\delta\mathcal{S}_c$ where δ specifies a constant box increment in percent to increase the margin of the mesh. In the co-moving frame the natural choice of the computation space is $\mathcal{S}_c = [-1, 1]^3$ since the bunch is located around the design trajectory with the reference particle at $(x, y, z) = (0, 0, 0)$. In order to consider an inhomogeneous problem domain, the box dimension of $\mathcal{S}_c$ can be adjusted by the user at the beginning. After solving Poisson's equation the electrostatic potential and the electric field have to be rescaled properly. Instead of rescaling the fields at the location of the particles, it is directly done on the grid as depicted in Fig. 2. The mapping of the particle coordinates in co-moving space $\mathcal{S}_p$ to computation space $\mathcal{S}_c$ includes also the Lorentz transform.

4.1.1. Particle coordinate

Let $\mathbf{x} = (x_0, x_1, x_2) \in \mathcal{S}_p$ be a coordinate of some particle in the particle space $\mathcal{S}_p$ and let $\mathbf{l} = (l_0, l_1, l_2) > \mathbf{0}$, then we define

$$\Gamma(\mathbf{x}, \mathbf{l}) := \max_{i=\{1,2,3\}} \left|\frac{x_i}{l_i}\right|.$$

The transform of an individual particle at position $\mathbf{x} \in \mathcal{S}_p$ into computation space $\mathbf{x}^* \in \mathcal{S}_c = [-l_0, l_0] \times [-l_1, l_1] \times [-l_2, l_2]$ is therefore given by

$$\mathbf{x}^* = \frac{\mathbf{x}}{s} \quad \text{with} \quad s = \arg\max_{\mathbf{x}\in\mathcal{S}_p} \sum_{i=0}^{N-1} \Gamma(\mathbf{x}_i, \mathbf{l}),$$

where N is the number of particles.

4.1.2. Electrostatic potential

Let $\phi \in \mathcal{S}_p$ be the electrostatic potential in particle space $\mathcal{S}_p$ and $\phi^* \in \mathcal{S}_c$ the corresponding potential value in computation space $\mathcal{S}_c$, then they relate as

$$\phi = \frac{1}{s}\phi^*. \tag{3}$$

Proof. Let the discrete charge density of N particles be described by [25, eq. 1.6]

$$\rho(\mathbf{x}) = \sum_{i=1}^{N} q_i\delta(\mathbf{x} - \mathbf{x}_i) \qquad \mathbf{x} \in \mathbb{R}^d,$$

in d dimensions and the coordinates being transformed as denoted above then

$$\rho = s^{-d}\rho^*$$

with $s > 0$ and

$$\frac{\partial}{\partial w} = \frac{\partial w^*}{\partial w}\frac{\partial}{\partial w^*} = s^{-1}\frac{\partial}{\partial w^*}$$

where $w = x_1, x_2, \ldots, x_d$. Thus,

$$\begin{aligned}
\Delta\phi &= -\frac{\rho}{\epsilon_0}\\
s^{-2}\Delta^*\phi &= -s^{-d}\frac{1}{\varepsilon_0}\rho^*\\
s^{-2}\Delta^*\phi &= s^{-d}\Delta^*\phi^*\\
\phi &= s^{2-d}\phi^*.
\end{aligned}$$

Therefore, the potential transforms in 3 dimensions as denoted in Eq. (3). In 2 dimensions the electrostatic potential remains. □

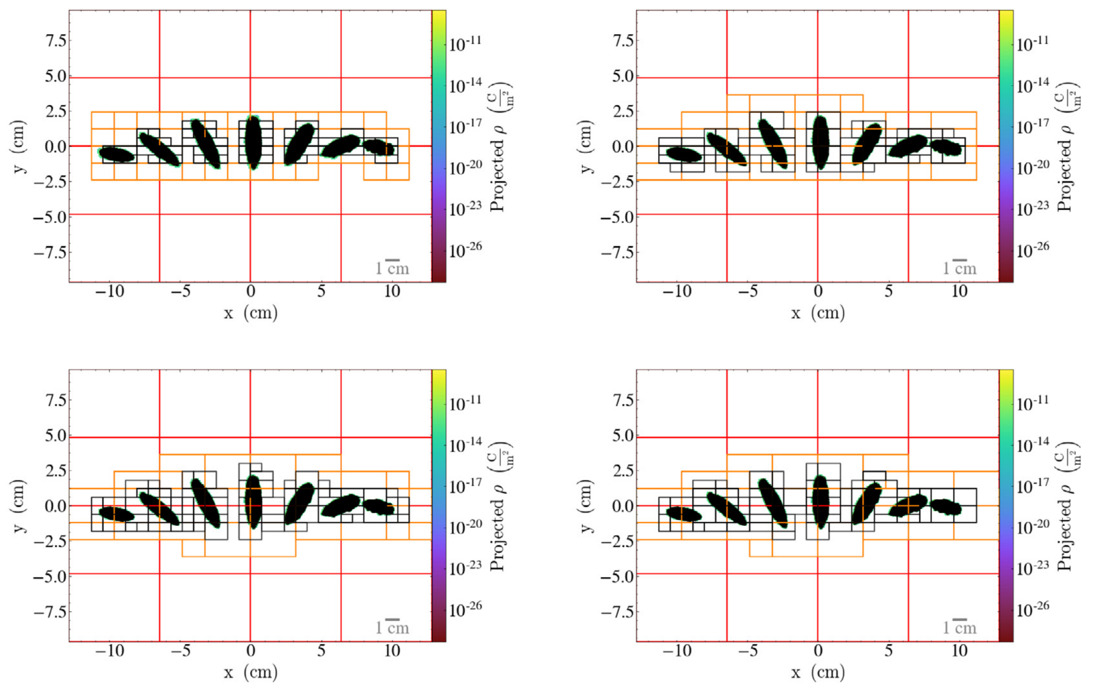


Fig. 3. Integrated projection of the charge density onto the *xy*-plane showing 7 adjacent particle bunches. Adaptive mesh refinement with charge density threshold 1×10^{-6} C/m^3 (top left), 1×10^{-7} C/m^3 (top right), 1×10^{-8} C/m^3 (bottom left), 1×10^{-9} C/m^3 (bottom right). Plotted with an own extension of the yt package [26].

4.1.3. Electric field

Let $\mathbf{E} \in \mathcal{S}_p$ be the electric field in particle space $\mathcal{S}_p$ and $\mathbf{E}^* \in \mathcal{S}_c$ the corresponding electric field vector in computation space $\mathcal{S}_c$, then they relate as

$$\mathbf{E} = \frac{1}{s^2}\mathbf{E}^*. \tag{4}$$

Proof. According to Gauss' law the electric field is the derivative of the electrostatic potential. Thus, an additional s^{-1} contributes to the transformation, therefore,

$$\mathbf{E} = s^{1-d}\mathbf{E}^*$$

that coincides with (4) in 3 dimensions. □

4.2. Adaptive mesh refinement policies

Besides the regrid function each AMR module implements the charge deposition, the particle-to-core (re-)distribution and various refinement strategies. There are currently six refinement policies available. Most refinement strategies are directly connected to particle properties since it is desirable to increase the spatial resolution at their location. Natural choices of refinement criteria are the charge density per cell, the electrostatic potential and the electric field. They are explained in more detail below. Other methods limit the minimum or maximum number of particles within a cell. The last tagging option refines cells based on the momentum of particles. While the first three methods refine the mesh based on the grid data, the latter methods use particle information directly. All methods apply a user-defined threshold λ in order to control the mesh refinement. This threshold denotes either the minimum charge density per cell

$$|\rho^l_{i,j,k}| \geq \lambda, \tag{5}$$

or a scale factor $\lambda \in [0, 1]$ in order to refine every grid cell (i, j, k) on a level l that satisfies

$$|\phi^l_{i,j,k}| \geq \lambda \max_{i,j,k} |\phi^l|$$

or

$$|E^l_{w;i,j,k}| \geq \lambda \max_{i,j,k} |E^l_w|,$$

in case of the electrostatic potential ϕ or the electric field components E_w with $w \in \{x, y, z\}$, respectively. The charge density in Eq. (5) is scaled in order to account for the domain transformation as previously mentioned and explained in detail in Section 4.1. Examples of AMR based on the charge density, potential and electric field with various thresholds are shown in Figs. 3–5, respectively.

5. Adaptive geometric multigrid

This section describes the algorithm of the adaptive geometric multigrid (AGMG) according to [13,27] and its implementation with the second generation packages of Trilinos, i.e. *Tpetra* [14], *Amesos2* and *Belos* [15], *MueLu* [16,17] and *Ifpack2* [18]. A cell-centred implementation is also presented in [28]. In opposite to previous implementations the one presented here is hardware independent thanks to the aforementioned Trilinos packages that have the *Kokkos* [9,10] library as backend. Another benefit is the convenient exchange of kernels such as smoothers (e.g. Gauss–Seidel or Jacobi) provided by *Ifpack2* or linear solvers of *Belos*, *Amesos2* and *MueLu*. The sparse matrices and vectors are instances of *Tpetra* classes.

5.1. Coarse–fine interface

AGMG is a special variant of the classical geometric multigrid since not all levels cover the full domain $\Omega = \Omega^0$ (cf. Fig. 1). At interfaces between subsequent levels $\partial\Omega^{l,l+1}$ the *elliptic matching condition* (i.e. Neumann and Dirichlet boundary condition) must be satisfied in order to ensure continuity of the solution. This condition is met by *flux differencing*

$$\mathcal{L}^l\phi(\mathbf{x}) = \sum_{d=1}^{3} \frac{f(\mathbf{x} + \frac{1}{2}h^l_d\mathbf{e}_d) - f(\mathbf{x} - \frac{1}{2}h^l_d\mathbf{e}_d)}{h^l_d} + \mathcal{O}\left((h^l_d)^2\right), \tag{6}$$

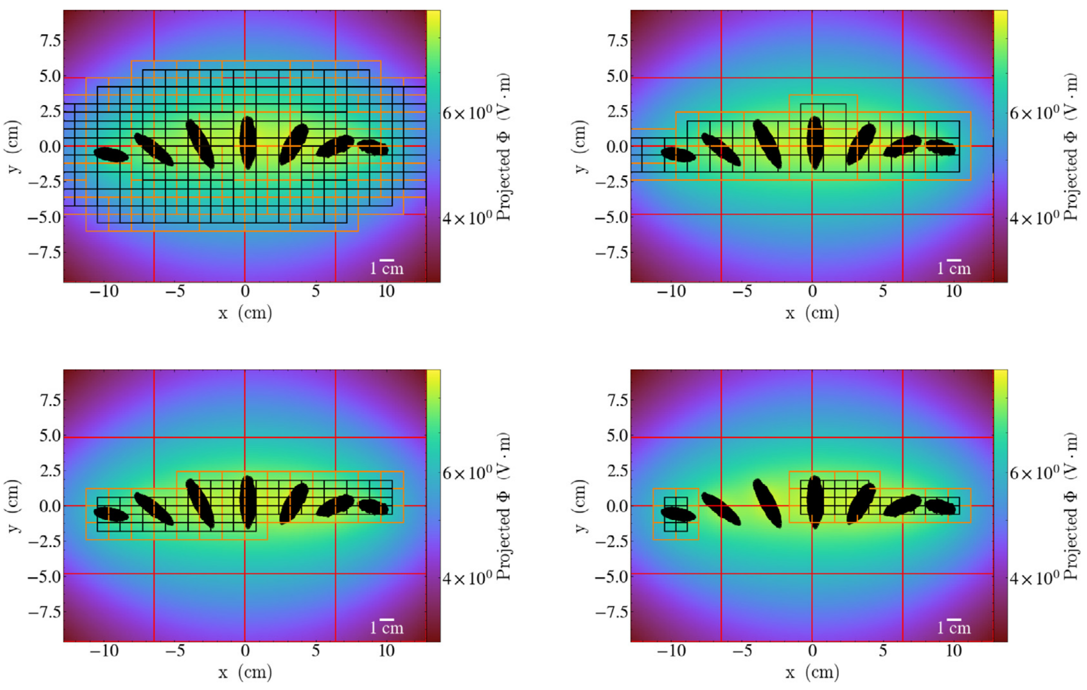


Fig. 4. Integrated projection of the electrostatic potential onto the *xy*-plane showing 7 adjacent particle bunches. Adaptive mesh refinement based on the electrostatic potential with thresholds λ from left to right and top to bottom: 0.25, 0.5, 0.75 and 0.95. Plotted with an own extension of the yt package [26].

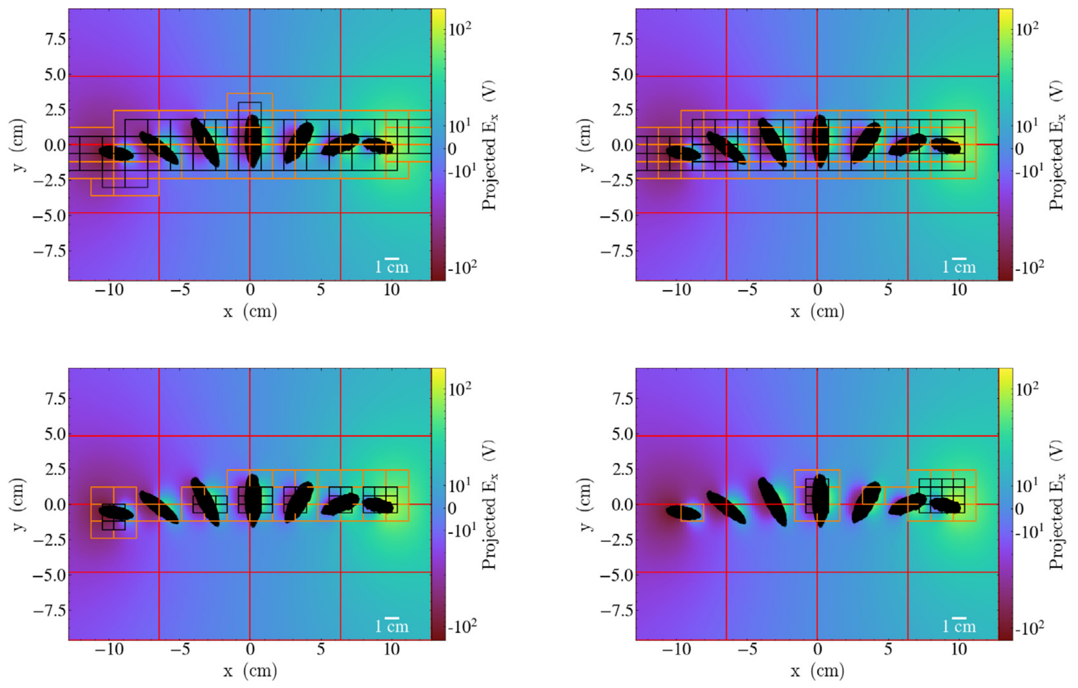


Fig. 5. Integrated projection of the electric field component E_x onto the *xy*-plane showing 7 adjacent particle bunches. Adaptive mesh refinement based on the electric field components with thresholds λ from left to right and top to bottom: 0.25, 0.5, 0.75 and 0.95. Plotted with an own extension of the yt package [26].

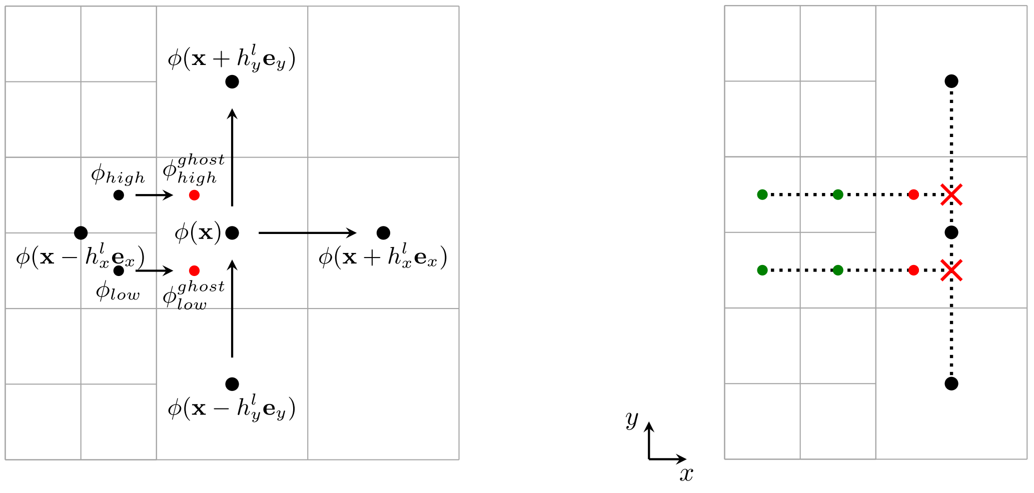
with mesh spacing h_d^l a unit vector $\mathbf{e}_d$ where either

$$\begin{aligned} f(\mathbf{x}+\frac{1}{2}h_d^l\mathbf{e}_d) &= \frac{\phi(\mathbf{x}+h_d^l\mathbf{e}_d)-\phi(\mathbf{x})}{h_d^l}, \\ f(\mathbf{x}-\frac{1}{2}h_d^l\mathbf{e}_d) &= \frac{\phi(\mathbf{x})-\phi(\mathbf{x}-h_d^l\mathbf{e}_d)}{h_d^l} \end{aligned} \tag{7}$$

on Ω^l or Eq. (8) which is given in Box I with $d^+, d^- \in \{1,2,3\}\backslash\{d\}$ and $d^+ \neq d^-$ at the interface $\partial\Omega^{l,l+1}$, i.e. the average flux across the boundary where a mesh refinement ratio (cf. Eq. (2)) of $r_d = 2\ \forall d \in \{1,2,3\}$ is assumed. In case of a cell without adjacent finer cells the flux differencing reduces to the usual second order Laplacian discretisation

$$\mathcal{L}^l\phi(\mathbf{x}) = \sum_{d=1}^{3} \frac{\phi(\mathbf{x}+h_d^l\mathbf{e}_d)-2\phi(\mathbf{x})+\phi(\mathbf{x}-h_d^l\mathbf{e}_d)}{(h_d^l)^2} + \mathcal{O}\left((h_d^l)^2\right). \tag{9}$$

An illustration of the stencil of Eq. (6) with fluxes computed either by Eq. (7) or Eq. (8) is shown in Fig. 6. In order to simplify



$\phi(\mathbf{x}+h_y^l\mathbf{e}_y)$
ϕ_{high}
ϕ_{high}^{ghost}
$\phi(\mathbf{x})$
$\phi(\mathbf{x}-h_x^l\mathbf{e}_x)$
$\phi(\mathbf{x}+h_x^l\mathbf{e}_x)$
ϕ_{low}
ϕ_{low}^{ghost}
$\phi(\mathbf{x}-h_y^l\mathbf{e}_y)$
y
x

(a) The red nodes indicate ghost cells that need to be interpolated.

(b) The red crosses specify the intermediate interpolation points using coarse cells.

Fig. 6. Illustration of flux differencing in 2D at a coarse–fine interface on the left side. In 2D the coarse–fine interface is 1D. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

$$\boxed{\begin{aligned} f(\mathbf{x}+\frac{1}{2}h_d^l\mathbf{e}_d) &= \sum_{i,j\in\{\pm\frac{1}{4}\}} \frac{\phi(\mathbf{x}+\frac{3}{4}h_d^l\mathbf{e}_d+ih_{d^+}^l\mathbf{e}_{d^+}+jh_{d^-}^l\mathbf{e}_{d^-})-\phi(\mathbf{x}+\frac{1}{4}h_d^l\mathbf{e}_d+ih_{d^+}^l\mathbf{e}_{d^+}+jh_{d^-}^l\mathbf{e}_{d^-})}{4h_d^{l+1}}, \\ f(\mathbf{x}-\frac{1}{2}h_d^l\mathbf{e}_d) &= \sum_{i,j\in\{\pm\frac{1}{4}\}} \frac{\phi(\mathbf{x}-\frac{1}{4}h_d^l\mathbf{e}_d+ih_{d^+}^l\mathbf{e}_{d^+}+jh_{d^-}^l\mathbf{e}_{d^-})-\phi(\mathbf{x}-\frac{3}{4}h_d^l\mathbf{e}_d+ih_{d^+}^l\mathbf{e}_{d^+}+jh_{d^-}^l\mathbf{e}_{d^-})}{4h_d^{l+1}} \end{aligned} \quad (8)}$$

Box I.

the representation the example is in 2D with only one coarse–fine interface on the left side. Hence, the corresponding finite difference stencil is given by

$$\mathcal{L}^l\phi(\mathbf{x}) = \frac{f(\mathbf{x}+\frac{1}{2}h_x^l\mathbf{e}_x)-f(\mathbf{x}-\frac{1}{2}h_x^l\mathbf{e}_x)}{h_x^l} + \frac{f(\mathbf{x}+\frac{1}{2}h_y^l\mathbf{e}_y)-f(\mathbf{x}-\frac{1}{2}h_y^l\mathbf{e}_y)}{h_y^l},$$

where

$$f(\mathbf{x}+\frac{1}{2}h_x^l\mathbf{e}_x) = \frac{\phi(\mathbf{x}+h_x^l\mathbf{e}_x)-\phi(\mathbf{x})}{h_x^l},$$

$$f(\mathbf{x}-\frac{1}{2}h_x^l\mathbf{e}_x) = \frac{1}{2h_x^{l+1}}\left(\phi_{high}^{ghost}-\phi_{high}+\phi_{low}^{ghost}-\phi_{low}\right),$$

$$f(\mathbf{x}+\frac{1}{2}h_y^l\mathbf{e}_y) = \frac{\phi(\mathbf{x}+h_y^l\mathbf{e}_y)-\phi(\mathbf{x})}{h_y^l},$$

$$f(\mathbf{x}-\frac{1}{2}h_y^l\mathbf{e}_y) = \frac{\phi(\mathbf{x})-\phi(\mathbf{x}-h_y^l\mathbf{e}_y)}{h_y^l}.$$

In 3D ghost cells are expressed in terms of valid coarse and fine cells where a two-step second order Lagrange interpolation in 2D

$$\phi^{interpolated}(u,v) = \sum_{i,j=0}^{2} L_i(u)L_j(v)\phi(u_i,v_j) \quad (10)$$

with

$$L_i(x) = \frac{(x-x_k)(x-x_l)}{(x_i-x_k)(x_i-x_l)} \qquad (l\neq i\neq k\neq l)$$

is performed. In 2D this corresponds to 1D Lagrange interpolations. First, the intermediate points symbolised as red crosses in Fig. 6b are computed with Eq. (10) where only non-covered coarse cells parallel to the interface are taken. Second, the fine cells normal to the boundary are used together with the intermediate locations to obtain the ghost cells with Eq. (10).

In 3D the interface is surface perpendicular to the current coarse–fine boundary. Depending on the surrounding cells this surface distinguishes nine configurations to evaluate the 2D quadratic Lagrange interpolation as shown in Fig. 7. The current location of the interface is denoted by the black dot. According to Eq. (10) nine non-refined coarse cells are required for second order interpolation denoted by the cells highlighted in red. For this purpose a surface consisting of 25 cells is checked perpendicular to the coarse–fine interface of interest. Ideally none of the surrounding coarse cells is refined such that the interpolation pattern shown in Fig. 7a is applied. The cases in Figs. 7b to 7e indicate a mesh refinement on a single side of this surface perpendicular to the coarse–fine interface. In case fine cells form a corner one of the patterns Figs. 7f to 7i is appropriate. The selection of the interpolation pattern follows a list ordered according to Fig. 7, i.e. from left to right and top to bottom. In order to simplify the evaluation of the interpolation scheme an integer value is assigned to each configuration obtained by its representation as a bit pattern (see Table 1). For this purpose all

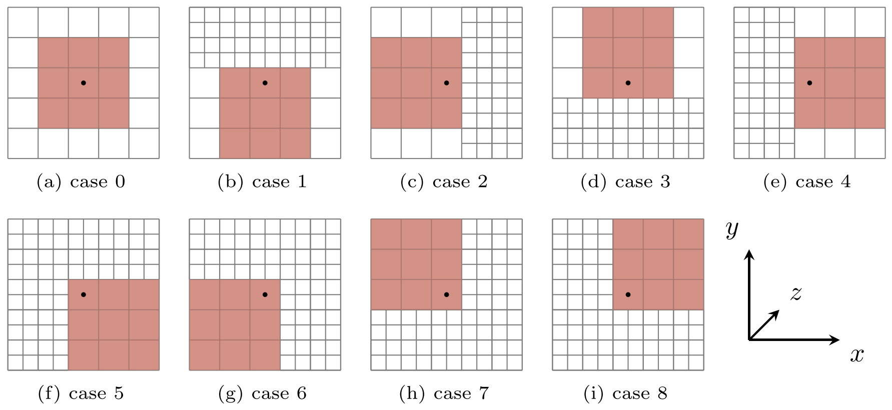


Fig. 7. All possible configurations for 2D quadratic Lagrange interpolation where the red cells are used for the interpolation. The coarse–fine interface is perpendicular to the shown cell layer (i.e. in *z*-direction). The black dot indicates the cell at the current coarse–fine-interface. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

Table 1
Bit patterns of the second order Lagrange interpolation schemes with ordering according to Fig. 7. The second column contains the corresponding number used to detect a pattern. An example of the conversion between grid and bits is indicated for the pattern on the right side with bit string highlighted in red.

20	21	22	23	24
15	16	17	18	19
10	11	12	13	14
5	6	7	8	9
0	1	2	3	4

bit pattern	unsigned long	case
0 0 0 0 0 0 1 1 1 0 0 1 1 1 0 0 1 1 1 0 0 0 0 0 0	473′536	0
0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 1 1 1 0 0 1 1 1 0	14′798	1
0 0 0 0 0 0 0 1 1 1 0 0 1 1 1 0 0 1 1 1 0 0 0 0 0	236′768	2
0 1 1 1 0 0 1 1 1 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0	15′153′152	3
0 0 0 0 0 1 1 1 0 0 1 1 1 0 0 1 1 1 0 0 0 0 0 0 0	947′072	4
0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 1 1 1 0 0 1 1 1 0 0	29′596	5
0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 1 1 1 0 0 1 1 1	7′399	6
0 0 1 1 1 0 0 1 1 1 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0	7′576′576	7
1 1 1 0 0 1 1 1 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0	30′306′304	8

Table 2
Bit patterns for 2D first order Lagrange interpolation (cf. Fig. 8). The first row highlighted in red indicates the example pattern on the right side.

6	7	8
3	4	5
0	1	2

bit pattern	unsigned long	case
0 0 0 0 1 1 0 1 1	27	0
0 1 1 0 1 1 0 0 0	216	1
1 1 0 1 1 0 0 0 0	432	2
0 0 0 1 1 0 1 1 0	54	3

25 cells are given a number denoting the position of the bits. A bit is flipped to one if the corresponding cell is not covered by fine cells. In case none of the nine patterns is applicable the interpolation order is reduced and thus one of the four first order Lagrange interpolation configurations of Fig. 8 is taken instead. The implementation follows exactly the same scheme with conversion shown in Table 2.

5.2. Boundary conditions

Assuming the beam in vacuum and neglecting any beam pipes the electrostatic potential converges to zero at infinity. In order to resemble this behaviour in *finite difference* a common approximation is the *Asymptotic Boundary Condition* (ABC) presented in [29,30] that is also denoted as *radiative* or *open* boundary condition (BC). The first order approximation ABC-1 is given by

$$\frac{\partial\phi(r)}{\partial r} + \frac{1}{r}\phi(r) = \mathcal{O}(r^{-3}). \tag{11}$$

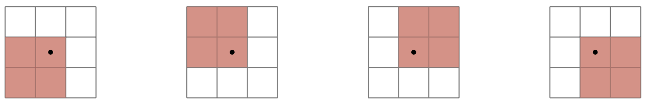
Instead to spherical coordinates a formulation in Cartesian coordinates is applied for example in [31–33]. In spherical coordinates the *n*th order approximation (ABC-*n*) is easily evaluated by

$$\left(\prod_{j=1}^{n}\left(\frac{\partial}{\partial r} + \frac{2j-1}{r}\right)\right)\phi(r) = \mathcal{O}(r^{1-2n}),$$

where the product is computed in decreasing order and $n \in \mathbb{N}$. The implementation presented in this article uses Robin boundary conditions to approximate open boundaries. The formula looks similar to Eq. (11) except that the radial derivative is replaced by a normal derivative w.r.t. the mesh boundary, i.e. [21]

$$\frac{\partial\phi}{\partial n} + \frac{1}{d}\phi = 0 \tag{12}$$

where $d > 0$ is an artificial distance. The condition is discretised using *central difference*. In addition to open BCs according to Eq. (12) the solver presented here allows to impose homogeneous Dirichlet and periodic BCs at the mesh (or physical) boundaries.



(a) case 0 (b) case 1 (c) case 2 (d) case 3

Fig. 8. All possible configurations for 2D linear Lagrange interpolation at which the red cells are used to build the Lagrange coefficients. The black dot indicates the cell at the current coarse–fine-interface. The interface is perpendicular to the shown cell layer. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

5.3. Algorithm and implementation details

Following the notation of [13,27], the full domain Ω is given by

$$\Omega = \sum_{l=0}^{l_{max}} \Omega^l - \mathcal{P}(\Omega^{l+1}),$$

where the projection $\mathcal{P}$ from level $l+1$ to level l satisfies $\mathcal{P}(\Omega^{l+1}) \subset \Omega^l$. Due to the properties of the refinement Poisson's equation is described by

$$\mathcal{L}^{comp}\phi = -\frac{\rho}{\varepsilon_0} \text{ on } \Omega$$

with composite Laplacian operator $\mathcal{L}^{comp}$ that considers only non-refined regions of each level. The full algorithm is illustrated in matrix notation in Alg. 2 to Alg. 3. It performs a V-cycle in the residual correction formulation with pre- and post-smoothing of the error. The iterative procedure stops when the l_p-norm of the residual of all levels with $p \in \{1, 2, \infty\}$ is smaller than the corresponding right-hand side norm. Since AMReX assigns the grids to cores independent of the underlying level distribution, the implementation provides special matrices, i.e. B^l_{crse} and B^l_{fine}, to handle the coarse–fine-interfaces. Thus, each AMR level stores up to ten matrices and four vectors represented by *Tpetra* objects. These are the composite Laplacian matrix A^l_{comp}, the Laplacian matrix assuming no-finer grids A^l_{nf}, the coarse boundary matrix B^l_{crse} and fine boundary matrix B^l_{fine}, the restriction and interpolation matrices R^l and I^l, respectively, the gradient matrices $\mathbf{G}^l$ and the matrix to get all uncovered cells U^l. The vectors per level are the charge density ρ^l, electrostatic potential ϕ^l, residual r^l and error e^l. Whereas the vectors span the whole level domain, some matrices only cover a subdomain or carry additional information for the coarse–fine interfaces as shown in Fig. 9. The coarse and fine boundary matrices encompass one side of the Lagrange interpolation stencil that is completed by the Laplacian matrices. In case of the finest level the composite and no-fine Laplacian matrices coincide.

The pre- and post-relaxation steps on line 8 and 16, respectively, of Alg. 3 use the algorithms provided by *Ifpack2* (e.g. Gauss–Seidel, Jacobi, etc.). The linear system of equations on the coarsest level (Alg. 3, line 20) is either solved by direct solvers available via *Amesos2* or iterative solvers of *Belos*. Furthermore, an interface to *MueLu* allows Smoothed Aggregation Algebraic Multigrid (SAAMG) as bottom solver.

Algorithm 1 Residual evaluation on the composite domain

```
Input: Level $l \geq 0$
Output: Updated residual $r^l$ on the composite domain
 1: function RESIDUAL($l$)
 2:     if $l = l_{max}$ then
 3:         $r^l \leftarrow \rho^l - A^l_{nf}\phi^l - B^l_{crse}\phi^{l-1}$
 4:     else
 5:         $r^l \leftarrow U^l\rho^l - U^l \cdot \left(A^l_{comp}\phi^l + B^l_{crse}\phi^{l-1} + B^l_{fine}\phi^{l+1}\right)$
 6:     end if
 7: end function
```

Algorithm 2 Main loop of AGMG

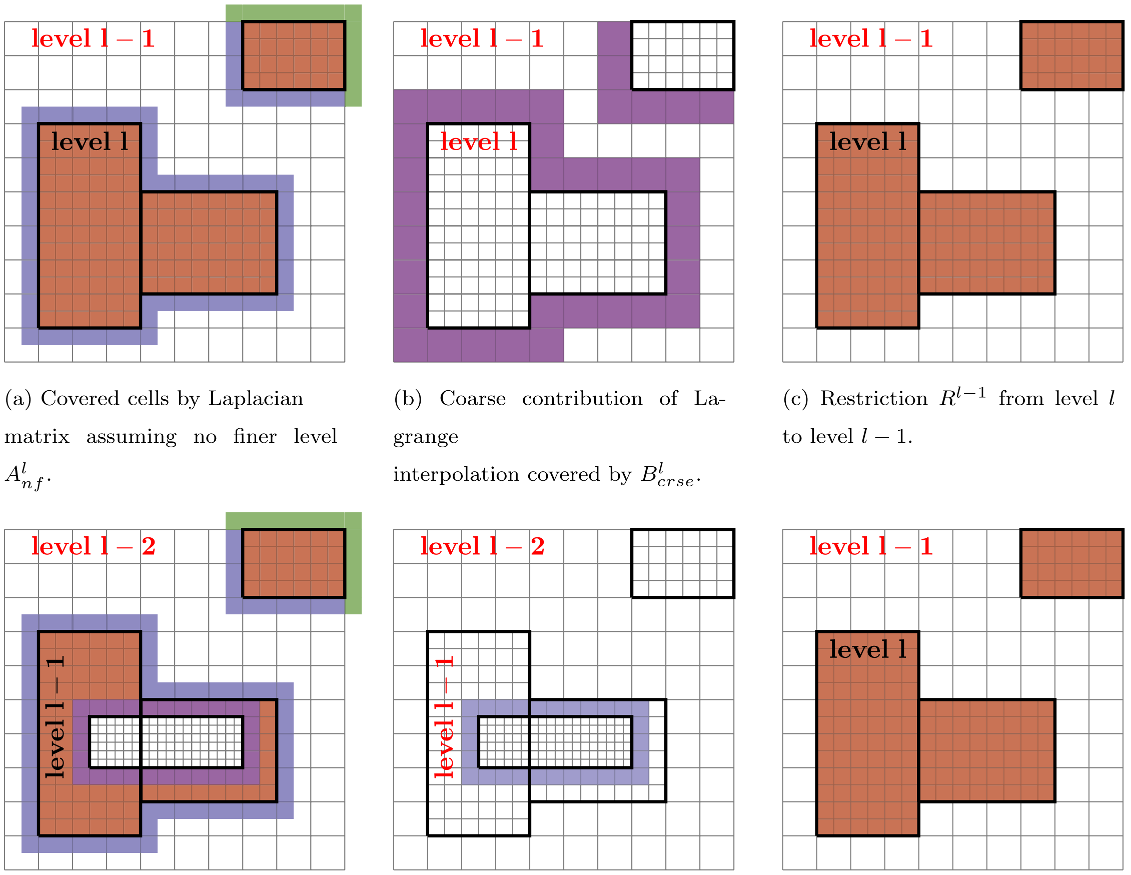
```
Input: Charge density $\rho$, electrostatic potential $\phi$, electric field $\mathbf{E}$ and finest level $l_{max}$
Output: Electrostatic potential $\phi$ and electric field $\mathbf{E}$
 1: function SOLVE($\rho, \phi, \mathbf{E}, l_{max}$)
 2:     for $l = 0$ to $l_{max}$ do
 3:         RESIDUAL($l$)                                        // Initialise residual
 4:     end for
 5:     $i \leftarrow 0$
 6:     while $i < i_{max} \wedge \exists l \in [0, l_{max}] : ||r^l||_p > \varepsilon||\rho^l||_p$ do    // $p \in \{1, 2, \infty\}$
 7:         RELAX($l_{max}$)                                      // Start of V-cycle
 8:         for $l = 0$ to $l_{max}$ do
 9:             RESIDUAL($l$)                                    // Update residual
10:         end for
11:         $i \leftarrow i + 1$
12:     end while
13:     for $l = l_{max} - 1$ to 0 do
14:         $\phi^l \leftarrow U^l\phi^l + R^l\phi^{l+1}$                              // Average down
15:     end for
16:     for $l = 0$ to $l_{max}$ do
17:         for $d = 0$ to 3 do
18:             $\mathbf{E}^l_d \leftarrow -\mathbf{G}^l_d\phi^l$                        // Evaluate electric field
19:         end for
20:     end for
21: end function
```

Algorithm 3 Residual correction V-Cycle

```
Input: Level $l \geq 0$
Output: Electrostatic potential $\phi$
 1: function RELAX($l$)
 2:     if $l = l_{max}$ then
 3:         $r^l \leftarrow \rho^l - A^l_{nf}\phi^l - B^l_{crse}\phi^{l-1}$
 4:     end if
 5:     if $l > 0$ then
 6:         $\phi^l_{save} \leftarrow \phi^l$
 7:         $e^{l-1} \leftarrow 0$
 8:         SMOOTH($e^l, r^l$)                  // Pre-smooth: Gauss–Seidel, Jacobi, ...
 9:         $\phi^l \leftarrow \phi^l + e^l$
10:         $r^{l-1} \leftarrow R^{l-1} \cdot \left(r^l - A^l_{nf}e^l - B^l_{crse}e^{l-1}\right)$     // Restrict on covered domain
11:         $r^{l-1} \leftarrow U^{l-1}\rho^{l-1} - A^{l-1}_{comp}\phi^{l-1} - B^{l-1}_{crse}\phi^{l-2} - B^{l-1}_{fine}\phi^l$ // Uncovered domain
12:         RELAX($l - 1$)
13:         $e^l \leftarrow I^l e^{l-1}$                          // Prolongation / Interpolation
14:         $r^l \leftarrow r^l - A^l_{nf}e^l - B^l_{crse}e^{l-1}$
15:         $\delta e^l \leftarrow 0$
16:         SMOOTH($\delta e^l, r^l$)                // Post-smooth: Gauss–Seidel, Jacobi, ...
17:         $e^l \leftarrow e^l + \delta e^l$
18:         $\phi^l \leftarrow \phi^l_{save} + e^l$
19:     else
20:         $Ae^0 = r^0$                            // Solve linear system of equations
21:         $\phi^0 \leftarrow \phi^0 + e^0$
22:     end if
23: end function
```

6. Poisson solver validation

The Poisson solver is validated using three different examples. First, the preservation of symmetry is tested. Second, a comparison with the analytical solution of a uniformly charged sphere in free space is shown. Although AMR is not turned on for a single-bunch simulation in the real application, it is nevertheless a good mini-app to check for any discontinuities at the coarse–fine interfaces among levels. In a third example the solver is validated by means of the built-in Poisson multi-level (ML)



(a) Covered cells by Laplacian matrix assuming no finer level A^l_{nf}.

(b) Coarse contribution of Lagrange interpolation covered by B^l_{crse}.

(c) Restriction R^{l-1} from level l to level $l-1$.

(d) Composite Laplacian matrix A^{l-1}_{comp}. $A^l_{comp} \equiv A^l_{nf}$.

(e) Fine contribution of Lagrange interpolation covered by B^{l-1}_{fine}.

(f) Prolongation I^l from level $l-1$ to level l.

Fig. 9. Cell domain occupied by matrices. Red: *Usual* cell domain; Green: Physical / mesh boundary; Blue: Fine contribution of Lagrange interpolation; Violet: Coarse contribution of Lagrange interpolation. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

solver of AMReX where 11 Gaussian-shaped bunches are placed in a chain using Dirichlet boundary conditions in the computation domain mimicking a multi-bunch simulation in high intensity cyclotrons as studied in [19]. The last two tests use the charge density to obtain the mesh refinements with threshold $\lambda = 1\,\text{fC/m}^3$ (cf. Eq. (5) in Section 4.2).

All line and projection plots are generated with an own extension of the yt package [26]. In the following, a regular PIC model with a uniform single-level mesh, i.e. without refinement, is an AMR simulation of at most level zero.

6.1. Symmetry conservation

In order to check symmetry preservation we initialise a three level problem where each level covers the centred region as shown in Fig. 10a. At each level, the grid cells are assigned to the same charge density value, starting at $1\,\text{C/m}^3$ on level zero and increasing by $0.5\,\text{C/m}^3$ on each subsequent level. Therefore, cutting a line through the centre of the domain yields a perfectly symmetric electrostatic potential and anti-symmetric electric field components mirrored at the centre. According to Fig. 10b, the symmetry is preserved with absolute errors in the order of magnitude of machine precision and thus negligible.

6.2. Uniformly charged sphere in free space

In this mini-app 10^6 particles are randomly picked within a sphere of radius $R = 5$ mm centred at origin. In order to simplify comparison to the analytical solution

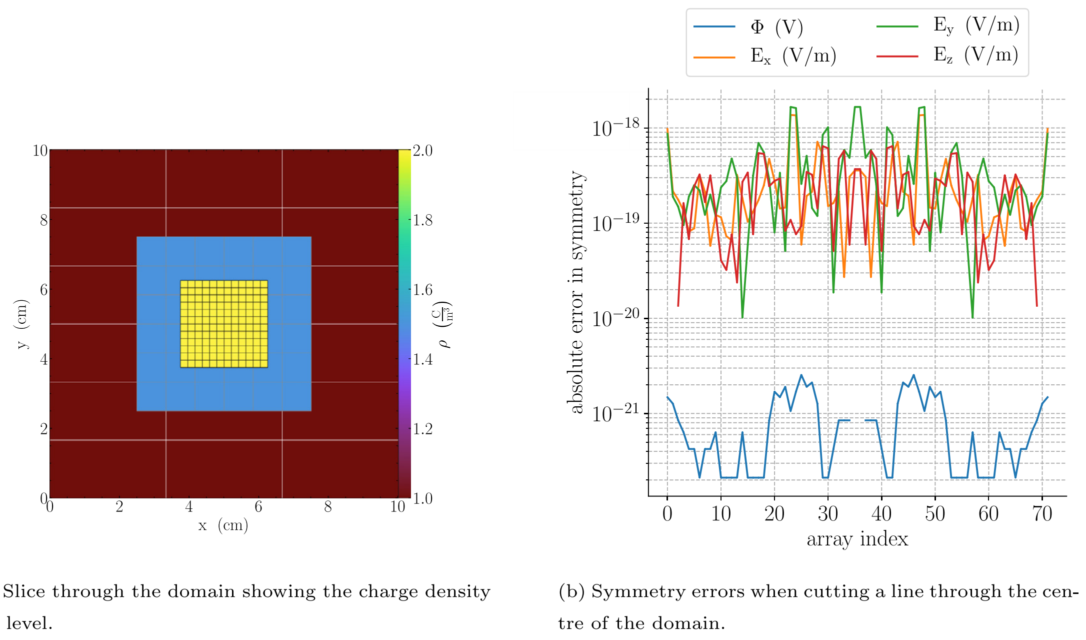
$$E(r) = \frac{Q}{4\pi\epsilon_0}\begin{cases} r^{-2}, & r > R \\ R^{-3}r, & r \le R, \end{cases}$$

$$\phi(r) = \frac{Q}{4\pi\epsilon_0}\begin{cases} r^{-1}, & r > R \\ (2R)^{-1}\cdot(3 - r^2R^{-2}), & r \le R, \end{cases}$$

each particle carries a charge of $q = 4\pi\epsilon_0 R^2 \cdot 10^{-2}$ C. Thus, the peak value of the electric field is 10^4 V/m and 75 V for the potential. The computation is performed using a base grid of 36^3 grid points and 2 refined levels. The mesh is increased by $\delta = 20\,\%$ compared to the computation domain (cf. Section 4.1). The line plots of Fig. 11 show the results for various artificial distances d of Eq. (12). The solution with distance $d = 1.7$ agrees well with the analytical solution. As expected the potential deviates at the boundaries from the analytical solution due to the numerical approximation of the open boundaries. The integrated projections onto the xy-plane of the electrostatic potential and the electric field component E_x are shown in Fig. 12.

6.3. 11 Gaussian-shaped bunches

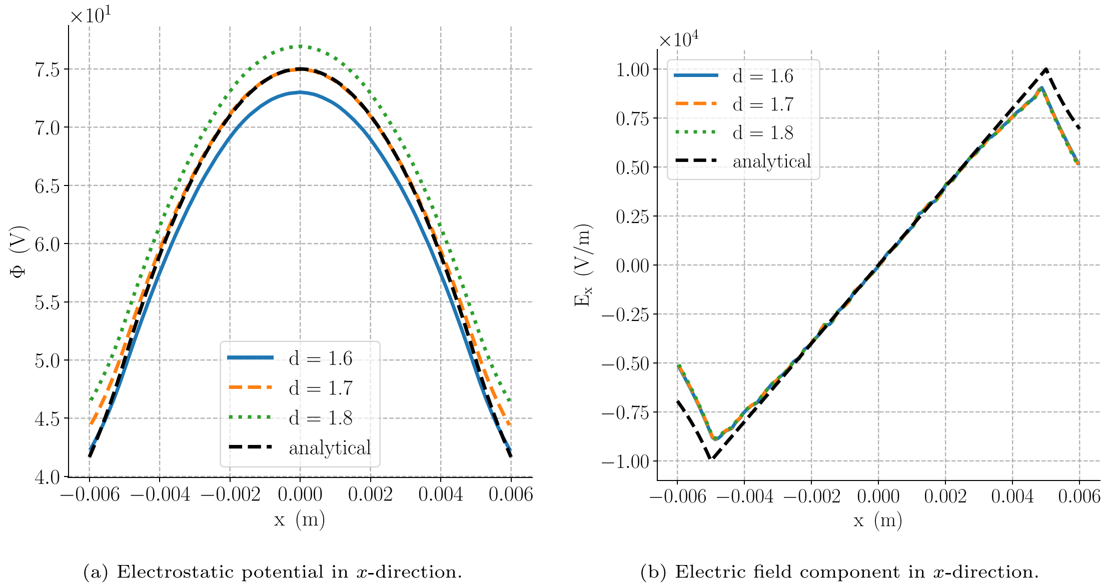
In this mini-app the newly implemented solver is compared to the Poisson solver of AMReX. Each bunch is initialised with 10^6 macro particles of charge 0.1 fC. The particles per bunch are picked using a one-dimensional Gaussian distribution per dimension with mean $\mu_y = \mu_z = 0$ m and standard deviation



(a) Slice through the domain showing the charge density per level.

(b) Symmetry errors when cutting a line through the centre of the domain.

Fig. 10. Charge density and absolute error in symmetry of electric field components and electrostatic potential. Starting at a charge density of 1 C/m^3 on level zero (full domain), it is incremented by 0.5 C/m^3 on subsequent higher levels.



(a) Electrostatic potential in x-direction.

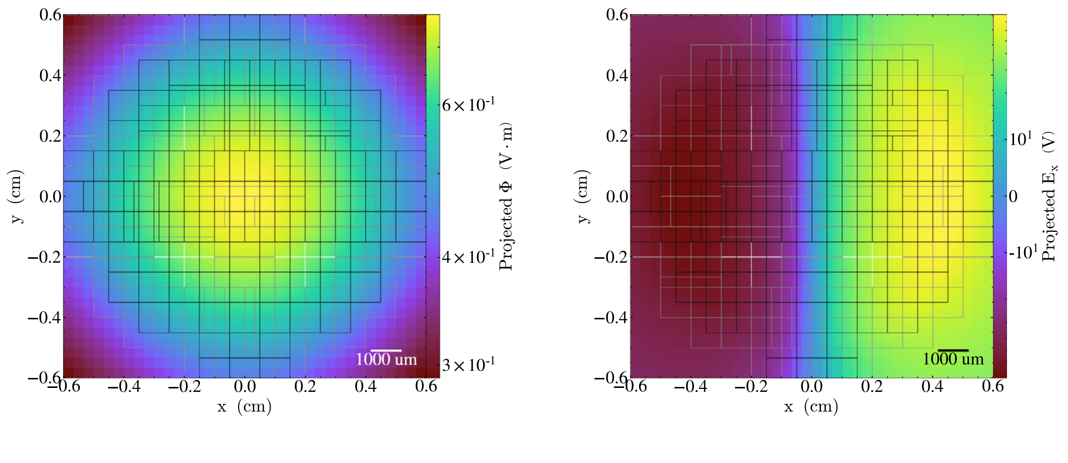
(b) Electric field component in x-direction.

Fig. 11. Comparison of the analytical and numerical solution of a uniformly charged sphere in free space with various artificial distances d of the open boundary condition (cf. Eq. (12)). The lines in (b) coincide.

$\sigma_y = \sigma_z = 5$ mm. In horizontal direction the standard deviation is $\sigma_x = 1.5$ mm with a mean shift of 4 cm to the neighbouring bunches. The problem is solved on a 144^3 base grid and 2 levels of refinement. At the mesh boundaries the Dirichlet boundary condition $\partial\phi = 0$ is imposed. The mesh is increased by $\delta = 10\,\%$ as explained in Section 4.1. As indicated by the line plots of Fig. 13 both solutions agree. The potential has a maximum absolute error 0.022 V that corresponds to a maximum relative error of 0.51%.

7. Neighbouring bunch simulation

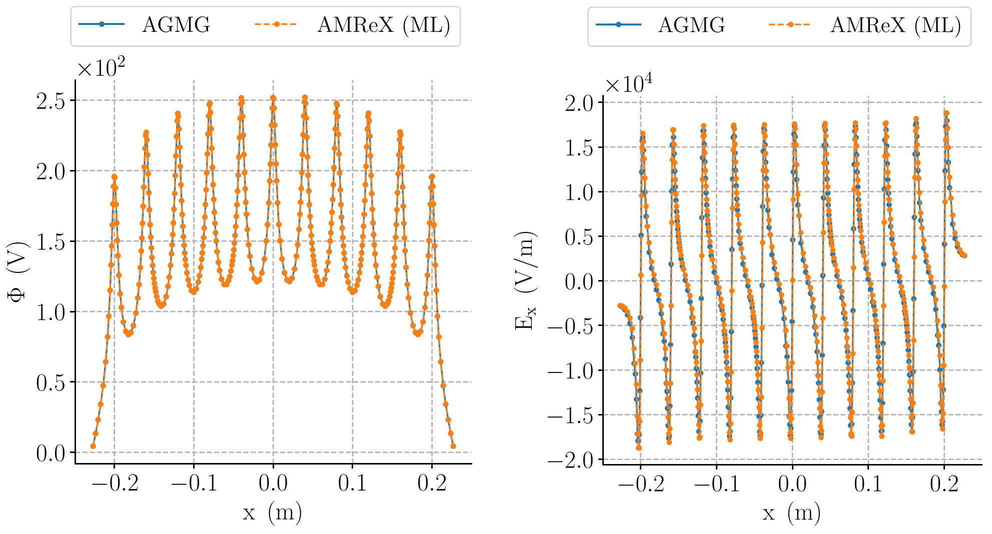
As initially stated the new AMR feature in OPAL is mainly developed to study neighbouring bunch simulations (cf. Fig. 3) in high intensity cyclotrons [19]. This type of simulation injects a new particle bunch after every turn. The computation domain therefore increases over time, resulting in a decrease in resolution in regular PIC. To overcome this issue the domain must be extremely finely discretised, at the expense of a high memory consumption and a waste of computing resources in regions without particles. In this section we illustrate the benefit of AMR over regular PIC w.r.t. memory and accuracy using a simplified model of the PSI Ring cyclotron. The simulation integrates either 5, 7, 9 or 11 neighbouring bunches each with 10^5 or 10^6 particles over one turn using 360 steps. In AMR mode the charge density per cell is used as refinement criterion (cf. Section 4.2) with cell threshold $\lambda = 1\,\mathrm{nC/m^3}$. The AMR hierarchy is updated after every tenth integration step. All simulations have an enlarged mesh of $\delta = 20\,\%$ compared to the computation domain. Poisson's equation is solved in a box with dimension $[-1, 1] \times [-0.75, 0.75] \times [-0.75, 0.75]$ to take into account the



(a) Electrostatic potential.

(b) Electric field component in x-direction.

Fig. 12. Integrated projection plots onto the *xy*-plane of the electrostatic potential and its electric field component E_x.



(a) Electrostatic potential in x-direction.

(b) Electric field component in x-direction.

Fig. 13. Line plots of the electrostatic potential and electric field of the multi-bunch test example.

inhomogeneity of the problem. At its boundaries we apply Robin BC (cf. Eq. (12)) with $d = 1.7$.

The results are compared to the single-level execution where we use the root mean square (rms) beam size, i.e.

$$\sigma_w = \sqrt{\langle w^2 \rangle} \tag{13}$$

and the beam-profile parameter [34], which is a statistical measure to determine the proportion of halo particles in a beam, i.e.

$$\xi_w = \frac{\langle w^4 \rangle}{\langle w^2 \rangle^2}, \tag{14}$$

where $\langle w^n \rangle$ denotes the nth moment of the particle distribution in coordinate $w \in \{x, y, z\}$. In Figs. 14 and 16 are the rms beam sizes and in Figs. 15 and 17 the beam-profile parameters of the centre bunch in a simulation of 5 and 11 adjacent bunches, respectively. The result of regular PIC with 512^3 grid points is compared to two AMR simulations with either 64^3 grid points on the coarsest level and three levels of refinement or 128^3 grid points on the coarsest level and two levels of refinement. All three simulations have therefore the same resolution on the finest grid. The halo parameters and rms beam sizes have an absolute error below $\mathcal{O}(10^{-5})$ compared to the regular PIC model. As observed in Fig. 18 and Table 5, however, the average resident set size (RSS), i.e. the amount of occupied physical memory, per MPI-process is on average at least four times smaller with AMR than FFT PIC. All simulations ran with 36 MPI-processes.

Besides the memory benefit, AMR reduces also the time to solution as visualised in Fig. 19. The detailed timing results of the Poisson solver and fourth order Runge–Kutta integration for 5 and 11 neighbouring bunches are shown in Tables 3 and 4. As expected, the particle integration grows in proportion to the increase in particles per bunch. The timings indicate that possible particle load imbalances do not harm the performance of the AMR PIC models significantly since the computation of the potential and electric field consume at least 87.7 % and 63.2 % in case of 10^5 and 10^6 particles per bunch, respectively. Overall, the runtime of the shown AMR configurations is at least 62.5 % shorter compared to FFT PIC.

The particle load balancing is quantified as the average number of particles per MPI-process $\langle N_p \rangle_s$ over all integration steps s

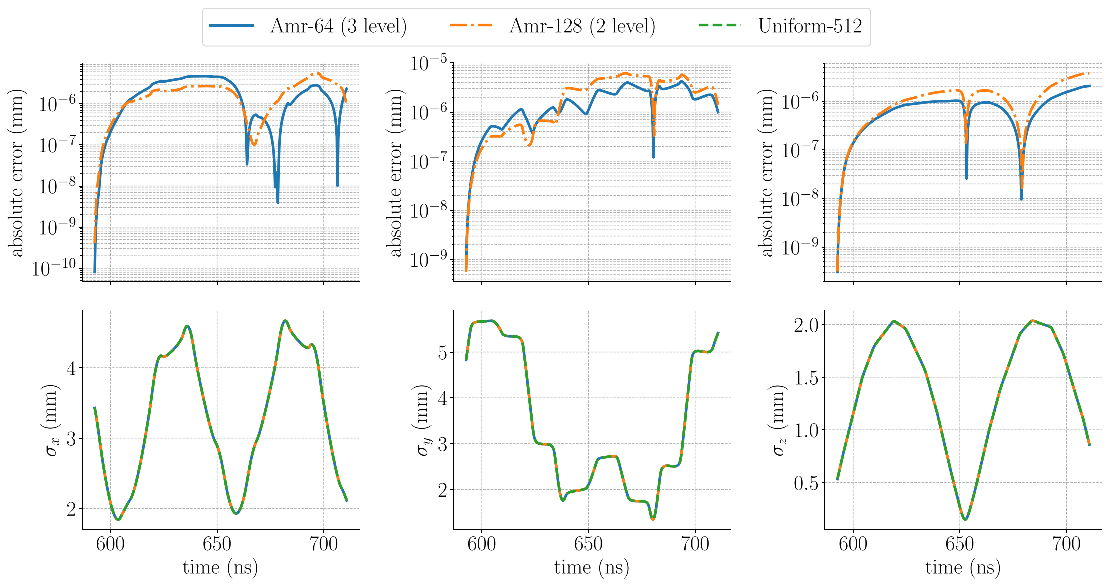


Fig. 14. Evolution of the rms beam size (cf. Eq. (13)) of the centre bunch in a simulation of 5 adjacent bunches and the absolute error of AMR models to the reference simulation with uniform mesh of 512^3 grid points (Uniform-512). On the finest level all three simulations have the same mesh resolution.

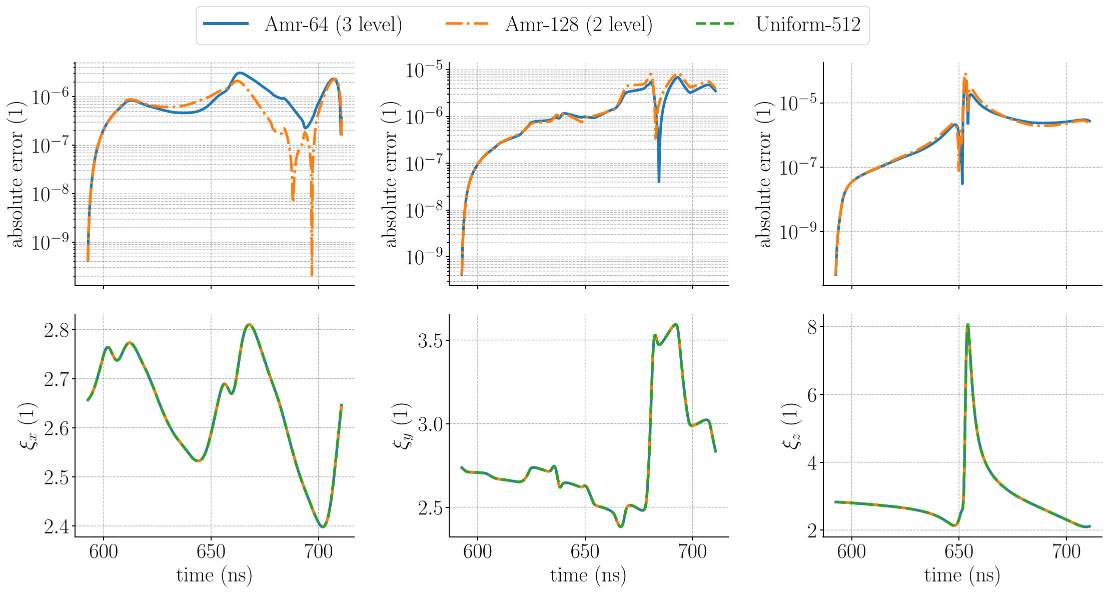


Fig. 15. Evolution of the beam-profile parameters (cf. Eq. (14)) of the centre bunch in a simulation of 5 adjacent bunches and the absolute error of AMR models to the reference simulation with uniform mesh of 512^3 grid points (Uniform-512). On the finest level all three simulations have the same mesh resolution.

Table 3
Detailed timing results (max. CPU time) of the Poisson solver and time integration with fourth order Runge–Kutta (RK-4) for 5 and 11 neighbouring bunches (nbs) of 10^5 macro particles each. The percentages are w.r.t. the total runtimes shown in the last two columns.

PIC model	Poisson timing (s)				RK-4 timing (s)				Total timing (s)	
	5 nbs		11 nbs		5 nbs		11 nbs		5 nbs	11 nbs
Amr-64	1 103	(95.3 %)	762	(87.7 %)	7.0	(0.6 %)	15.5	(1.8 %)	1 157	869
Amr-128	1 659	(96.8 %)	1296	(92.0 %)	8.5	(0.5 %)	18.5	(1.3 %)	1 714	1 409
Uniform-512	20 420	(99.7 %)	19100	(99.1 %)	33.0	(0.2 %)	76.3	(0.4 %)	20 490	19 270
FFT-512	9 325	(98.2 %)	9142	(97.8 %)	5.4	(0.1 %)	11.8	(0.1 %)	9 500	9 345

divided by the total number of particles in simulation N_t, i.e.

$$\frac{\langle N_p \rangle_s}{N_t}.$$

In the best case all MPI-processes have N_t/P_t particles during integration where P_t is the total number of processes. Figs. 20 and 21 show the number of cores that deviate from the optimum particle count within a few percent. The load balancing between 10^5 and 10^6 particles per bunch does not differ significantly. A similar observation is done in Figs. 22 and 23 where the optimal number of grid points among the MPI-processes is evaluated.

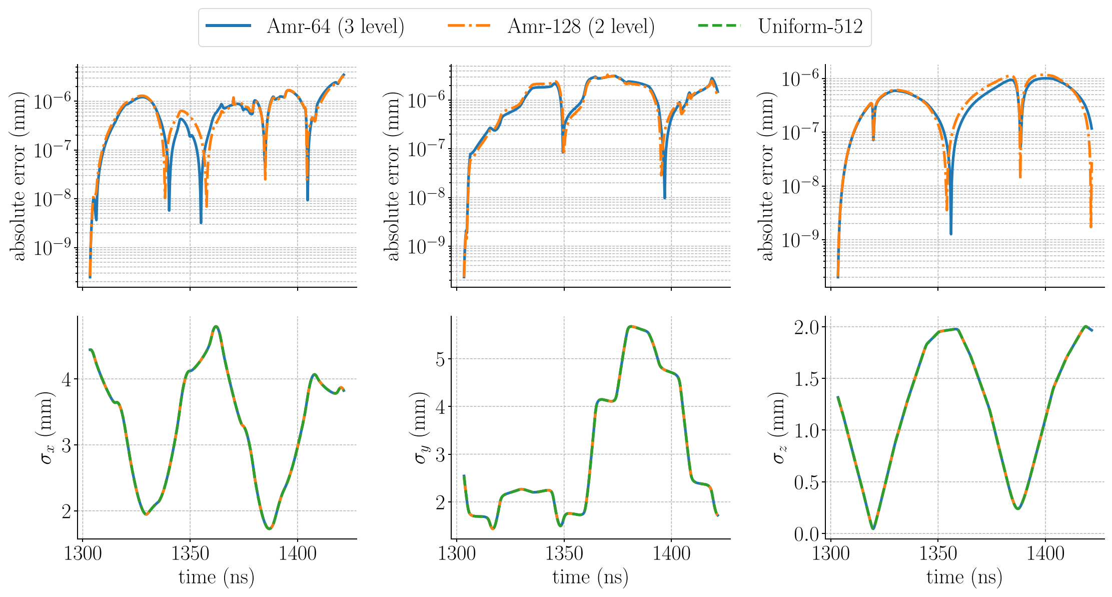


Fig. 16. Evolution of the rms beam size (cf. Eq. (13)) of the centre bunch in a simulation of 11 adjacent bunches and the absolute error of AMR models to the reference simulation with uniform mesh of 512^3 grid points (Uniform-512). On the finest level all three simulations have the same mesh resolution.

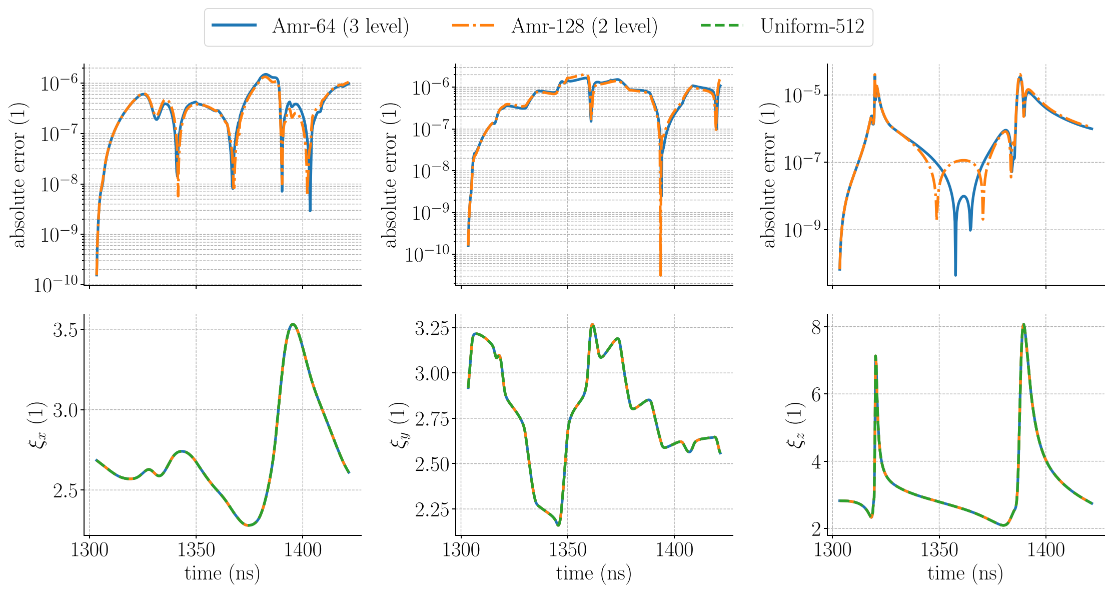


Fig. 17. Evolution of the beam-profile parameters (cf. Eq. (14)) of the centre bunch in a simulation of 11 adjacent bunches and the absolute error of AMR models to the reference simulation with uniform mesh of 512^3 grid points (Uniform-512). On the finest level all three simulations have the same mesh resolution.

Table 4
Detailed timing results (max. CPU time) of the Poisson solver and time integration with fourth order Runge–Kutta (RK-4) for 5 and 11 neighbouring bunches (nbs) of 10^6 macro particles each. The percentages are w.r.t. the total runtimes shown in the last two columns.

PIC model	Poisson timing (s)				RK-4 timing (s)				Total timing (s)	
	5 nbs		11 nbs		5 nbs		11 nbs		5 nbs	11 nbs
Amr-64	1 777	(76.9 %)	2 225	(63.9 %)	73.89	(3.2 %)	164.7	(4.7 %)	2 310	3 480
Amr-128	2 070	(78.3 %)	2 300	(63.2 %)	71.5	(2.7 %)	161.9	(4.4 %)	2 644	3 638
Uniform-512	20 750	(96.3 %)	19 240	(90.2 %)	334.3	(0.2 %)	765.8	(3.6 %)	21 540	21 340
FFT-512	8 978	(96.1 %)	9 032	(93.0 %)	52.72	(0.6 %)	118.0	(1.2 %)	9 343	9 712

8. Performance benchmark

The performance benchmark is done on the multicore partition of Piz Daint, a supercomputer at the Swiss National Supercomputing Centre (CSCS). The nodes on the multicore partition consist of two Intel Xeon E5-2695 v4 @2.10 GHz (2 × 18 cores, 64/128 GB RAM) processors [35]. The benchmark on the GPU partition of Piz Daint confirmed the hardware portability of the new solver. However, the data transfer between CPU (Central Processing Unit) and GPU as well as the launching of single

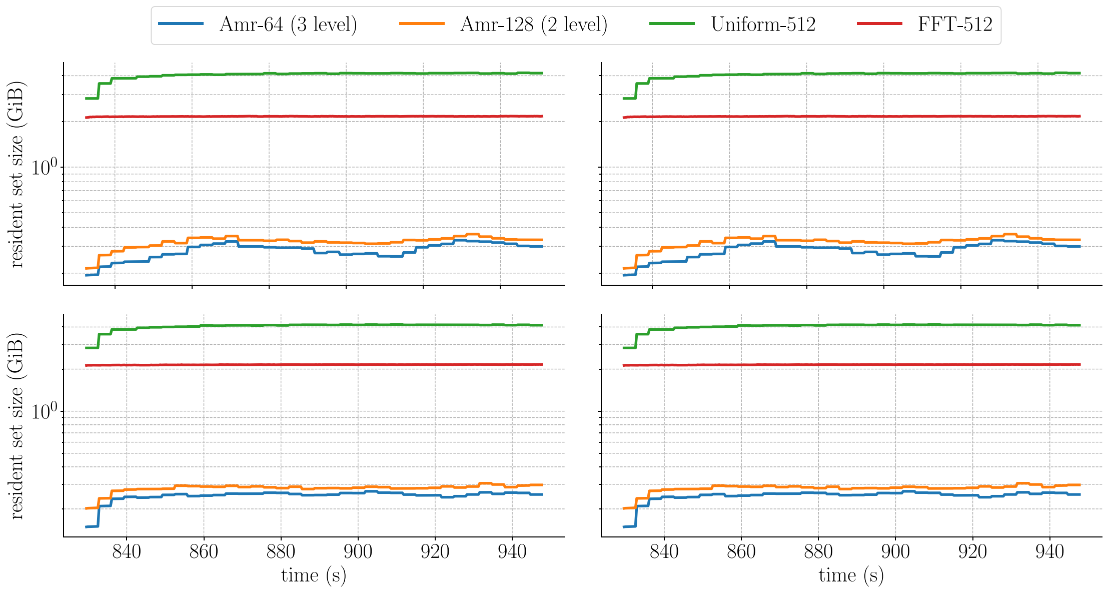


Fig. 18. Average resident set size (RSS) in Gibibyte (GiB) per MPI-process with 5 (top left), 7 (top right), 9 (bottom left) and 11 (bottom right) neighbouring bunches. Each bunch consists of 10^5 macro particles. All simulations were run with 36 MPI-processes.

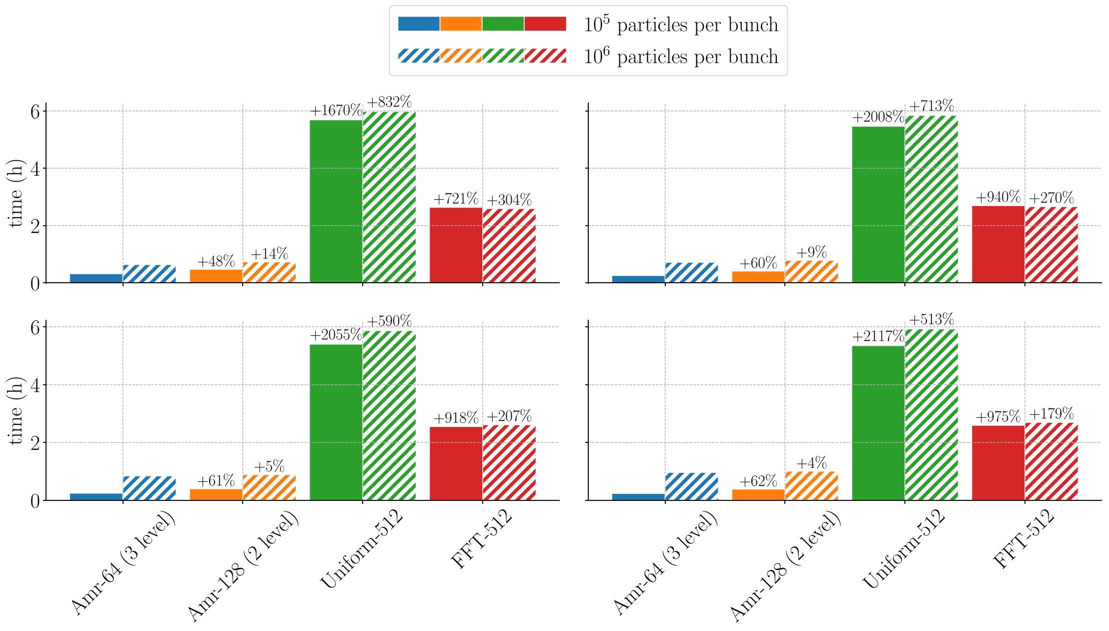


Fig. 19. Total simulation CPU time with 5 (top left), 7 (top right), 9 (bottom left) and 11 (bottom right) neighbouring bunches. A bunch consists either of 10^5 or 10^6 macro particles. All simulations were run with 36 MPI-processes. The percentages on top of the bars are w.r.t. the Amr-64 (3 level) timings.

Table 5
Average resident size (RSS) in Gibibyte (GiB) per MPI-process over all 360 integration steps with 5 or 11 neighbouring bunches (nbs) and 10^5 or 10^6 macro particles per bunch (ppb).

PIC model	Avg. RSS with 5 nbs (GiB)		Avg. RSS with 11 nbs (GiB)	
	10^5 ppb	10^6 ppb	10^5 ppb	10^6 ppb
Amr-64	0.2829	0.4386	0.2501	0.5683
Amr-128	0.3215	0.4599	0.2844	0.5900
Uniform-512	4.0524	4.1307	4.0534	4.1932
FFT-512	2.1572	2.2876	2.1757	2.3890

GPU kernels for each matrix–vector or matrix-matrix operation of *Tpetra* showed a performance bottleneck which is why the performance study presents a CPU benchmark only.

The test initialises 11 Gaussian-shaped bunches as described in Section 6.3 with 10^6 macro particles of charge 0.1 fC per bunch. The Poisson problem is solved 100 times on a three level hierarchy (two levels of refinement) with 576^3 grid points on level zero. The particles are randomly displaced within $\left[-10^{-3}, 10^{-3}\right]$ after every iteration. This represents a realistic setup for beam dynamics simulations since the particle distribution in the bunch rest frame changes only marginally from one integration time step to another. Therefore, it is not necessary to re-mesh the AMR hierarchy and thus rebuild the matrices after every time step which gives rise to computational savings. The optimal update frequency of the grids for neighbouring bunch simulations is currently unknown and is not subject in this article. Nevertheless, the computational saving is shown with two strong scalings. The

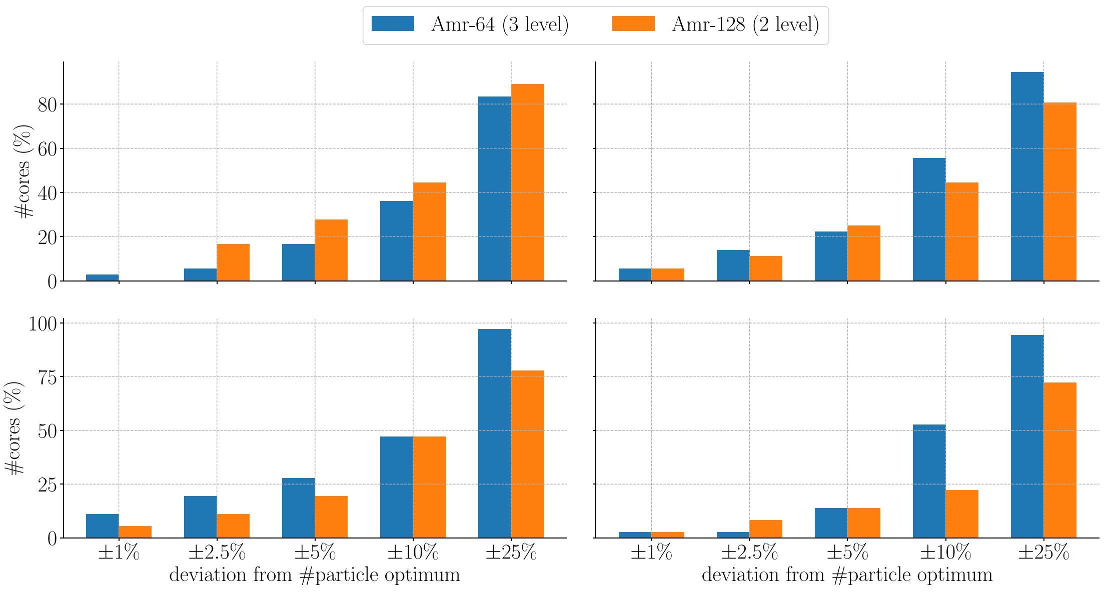


Fig. 20. Particle load balancing for 36 MPI-processes and 5 (top left), 7 (top right), 9 (bottom left) or 11 (bottom right) neighbouring bunches. Each bunch has 10^5 macro particles. The optimum is evaluated as the total number of particles divided by the number of MPI-processes.

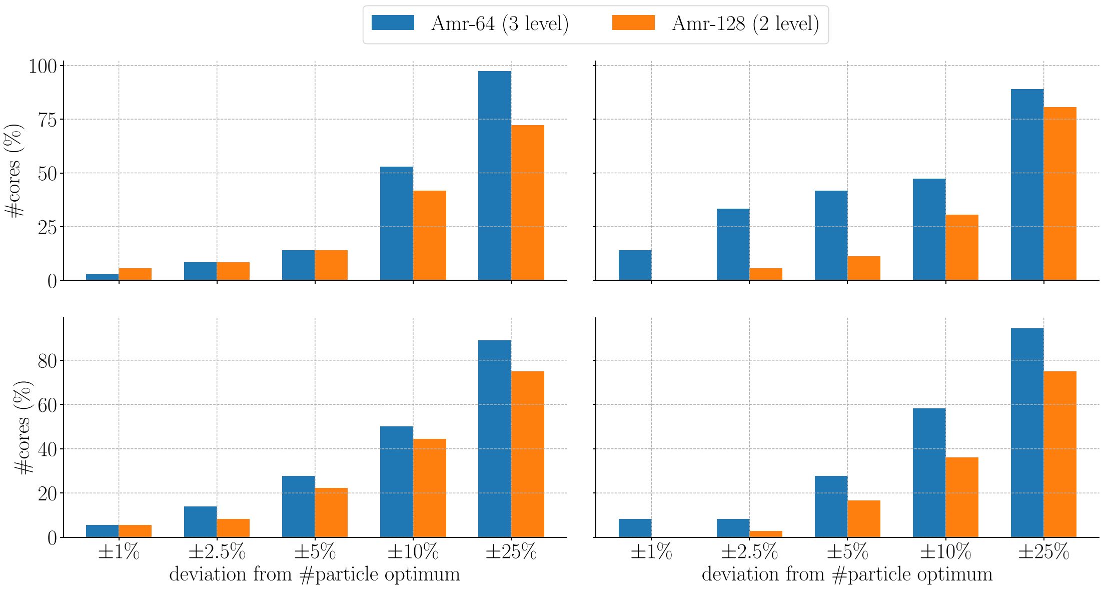


Fig. 21. Particle load balancing for 36 MPI-processes and 5 (top left), 7 (top right), 9 (bottom left) or 11 (bottom right) neighbouring bunches. Each bunch has 10^6 macro particles. The optimum is evaluated as the total number of particles divided by the number of MPI-processes.

first benchmark updates the AMR hierarchy after every computation of the electrostatic potential while the latter performs a regrid step after every tenth step. Since a constant workload per MPI-process during an upscaling that is necessary in a fair weak scaling cannot be guaranteed, the presented benchmark consists of a strong scaling only.

The blue line in Fig. 24 shows the total solver time of the 100 executions. As indicated in Table 6, the setup of the matrices (violet line), i.e. porting the AMReX mesh information to Trilinos, as well as the evaluation of the linear system of equations on the bottom level (grey line) with the algebraic multigrid solver of *MueLu* consume together more than 77 % of the time on 14 400 cores. However, the setup time can easily be reduced with a lower regrid frequency as previously mentioned. The matrix setup cost in the second timing is only 14 % of the setup cost observed by the first timing. Furthermore, the use of an algebraic multigrid solver for the linear system of equations on the bottom level is not an optimal choice. More suitable would be a geometric multigrid that keeps the structure of the problem which is planned for a future paper.

The parallel efficiency of the strong scaling of Fig. 24 is shown in Fig. 25. The efficiency of the total solve time (blue line) drops below 50 % for 120 or 160 computing nodes. In case the AMR hierarchy is updated after every solve, the efficiency is dominated by the bottom solver and the matrix setup time. However, reducing the regrid frequency shifts the dependency towards the bottom solver. For both regriding configurations we observe an increase in efficiency in case of 400 nodes. Since the maximum number of grid points per dimension on level zero is set to 24, all cores have the same amount of grid points on this level with 13 824 cores (i.e. 384 nodes) that causes the bottom solver to be more efficient.

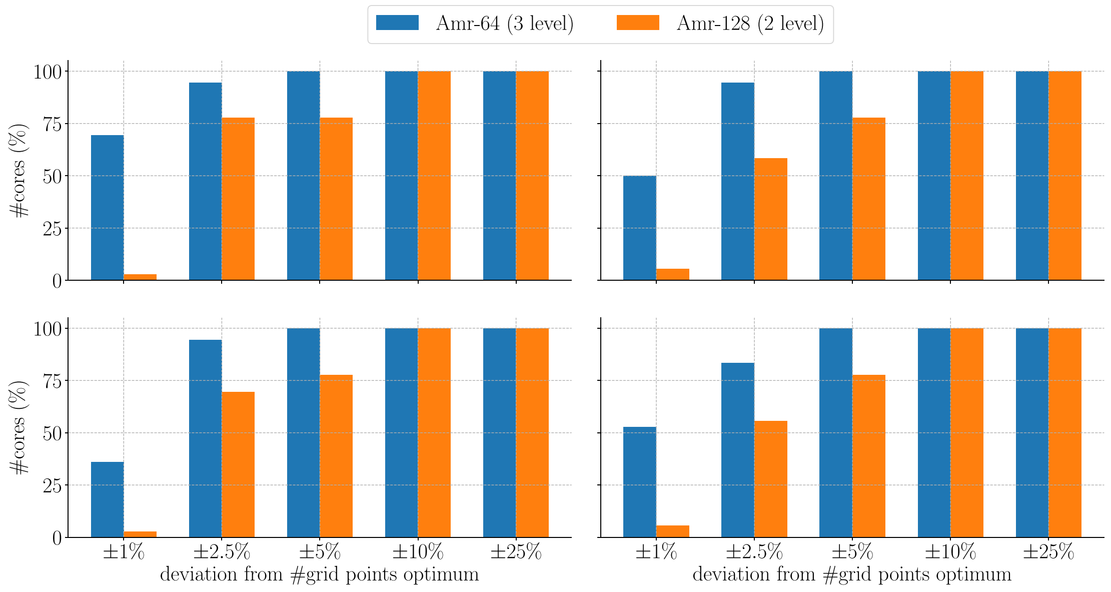


Fig. 22. Grid point load balancing for 36 MPI-processes and 5 (top left), 7 (top right), 9 (bottom left) or 11 (bottom right) neighbouring bunches. Each bunch has 10^5 macro particles. The optimum is evaluated as the total number of grid points per step divided by the number of MPI-processes.

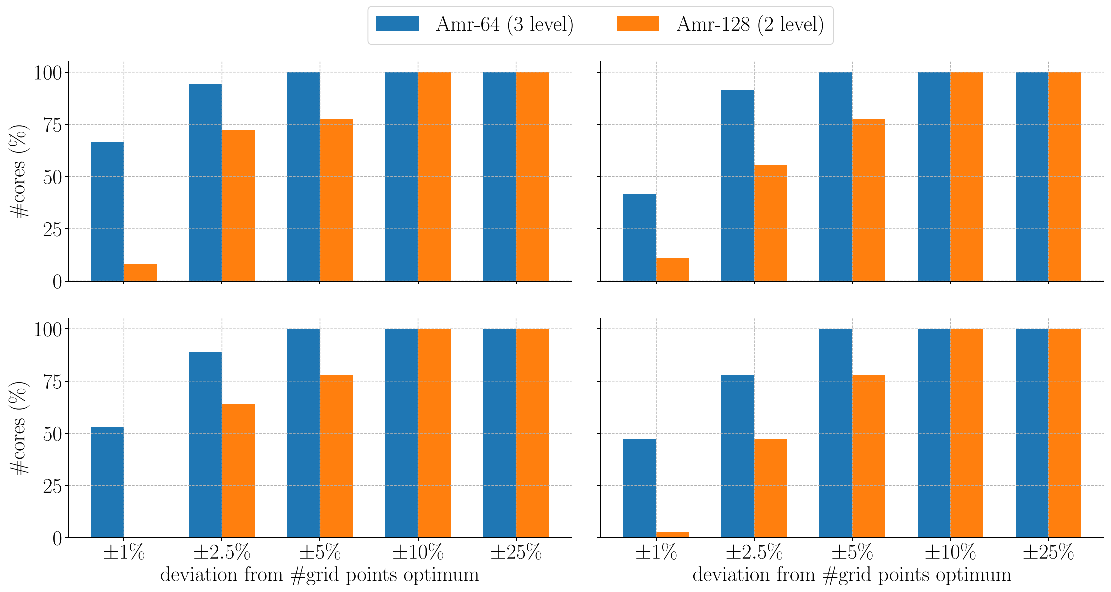


Fig. 23. Grid point load balancing for 36 MPI-processes and 5 (top left), 7 (top right), 9 (bottom left) or 11 (bottom right) neighbouring bunches. Each bunch has 10^6 macro particles. The optimum is evaluated as the total number of grid points per step divided by the number of MPI-processes.

Table 6
Summarised AGMG timings solving Poisson's equation 100 times on 14 400 cores (400 nodes). It shows the timing results of two configurations. The first updates the grids after every (100× regriding) and the second after every tenth (10× regriding) computation.

Timing	CPU avg (s) 100 × regriding	Fraction (%)	CPU avg (s) 10 × regriding	Fraction (%)
Total solve	378.72	100.00	343.66	100.00
Bottom solver	133.79	35.33	110.09	32.03
Matrix setup	159.63	42.15	22.49	6.54
Smoothing	23.23	6.13	17.59	5.12
Restriction	10.49	2.77	7.55	2.20
Bottom solver setup	2.26	0.60	1.44	0.42
Prolongation	2.09	0.55	1.60	0.46
E-field	0.67	0.18	0.61	0.18
Others	46.54	12.29	58.81	53.05

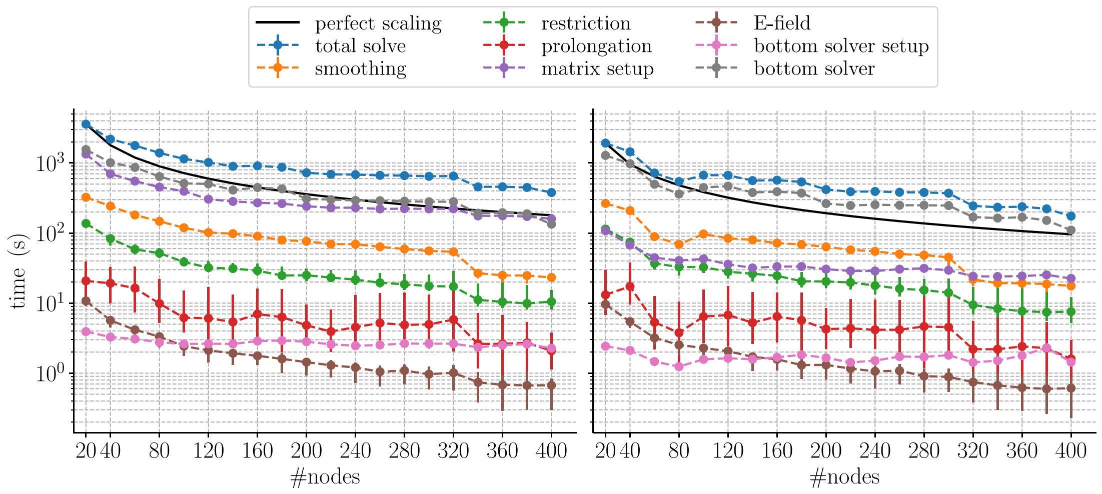


Fig. 24. Strong scaling performed on the multicore partition of Piz Daint (Cray XC40) with 36 cores per node (without hyperthreading). The perfect scaling (black line) uses the total solve time with 20 nodes as reference. Left: scaling with 100× regriding; right: scaling with 10× regriding. Each marker indicates the average CPU time per operation where the vertical line denotes the range by minimum and maximum. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

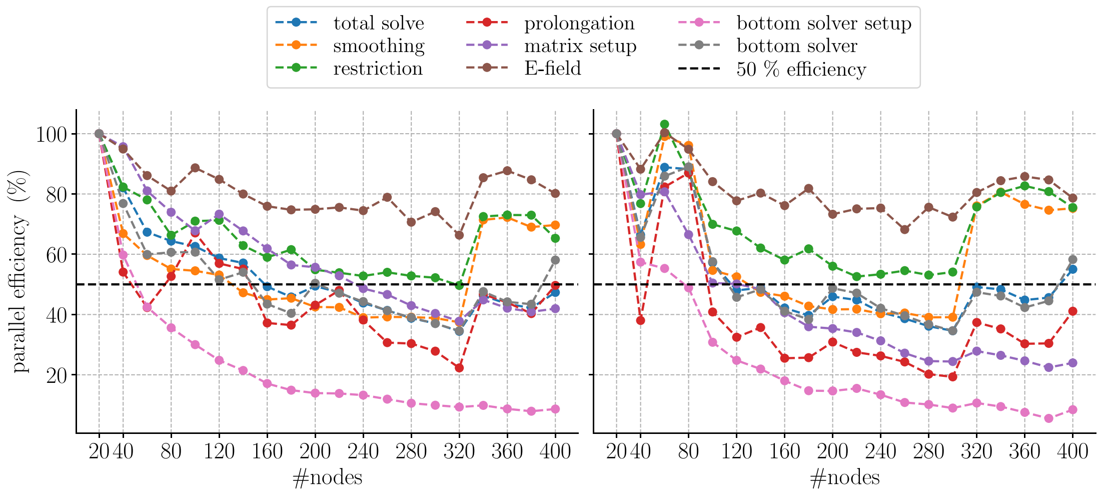


Fig. 25. Parallel efficiency. The increase of efficiency from 380 to 400 nodes is due to an optimal workload on the coarsest level with 13 824 cores. Left: efficiency with 100× regriding; right: efficiency with 10× regriding. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

9. Conclusion and outlook

In this article we presented the new adaptive mesh refinement capability of the open-source beam dynamics code OPAL which has been enhanced by AMReX. The new feature is supplemented with a hardware architecture independent implementation of a multigrid Poisson solver based on second generation Trilinos packages. Besides an artificial problem illustrating symmetry preservation and a comparison with an analytically solvable problem, the Poisson solver was validated with the built-in AMReX multi-level solver. Although the structure of the mesh is lost when going to the matrix representation, the solver shows good scalability on CPUs with a parallel efficiency between 50 % and 60 % on 14,400 cores depending on the AMR regrid frequency. The timings indicate that the matrix setup and the bottom linear system solver require 77 % of the total solver time. The former can be reduced by updating the mesh less frequently. The latter might be decreased by replacing the smoothed aggregation algebraic multigrid solver of *MueLu* with a structured aggregation procedure, a real geometric multigrid solver or a FFT solver which is subject to future research. Thanks to the hardware portability the solver runs on any backend that is supported by *Kokkos*. However, due to single kernel launches for each matrix–vector operation, the solver is not yet competitive on GPUs.

A small example of the PSI Ring cyclotron demonstrated the benefit of AMR over regular PIC models w.r.t. time to solution and memory consumption at a given accuracy. The presented benchmark shows that AMR requires about four times less memory and the time to solution is at least 62.5 % times shorter than a comparable simulation with the integrated FFT solver of OPAL. Therefore, the technique of adaptive mesh refinement will enable large-scale multi-bunch simulations in high intensity cyclotrons at higher grid resolution in order to more accurately quantify the effect of radially neighbouring bunches on halo formation and evolution. In future studies AMR might also be applied to simulations of FFA (Fixed-Field Alternating Gradient) accelerators.

Acknowledgements

Many thanks to the Center for Computational Sciences and Engineering (CCSE) at Lawrence Berkeley National Laboratory (LBNL), in particular A. S. Almgren, A. Myers and W. Zhang. The authors appreciate also the support of P. Arbenz, Dan. F. Martin, K. D. Devine and C. Siefert in the development of the multigrid solver. Furthermore, we thank the Swiss National Supercomputing Centre (CSCS) for providing the necessary computer time on Piz Daint. This project is funded by the Swiss National Science Foundation (SNSF) under contract number 200021_159936.

References

[1] R.W. Hockney, J.W. Eastwood, Computer Simulation Using Particles, Taylor & Francis, Inc., Bristol, PA, USA, 1988.
[2] A. Adelmann, U. Locans, A. Suter, Comput. Phys. Comm. 207 (2016) 83–90.
[3] M.J. Berger, J. Oliger, J. Comput. Phys. 53 (3) (1984) 484–512.
[4] M. Berger, P. Colella, J. Comput. Phys. 82 (1) (1989) 64–84.
[5] J. Hittinger, J. Banks, J. Comput. Phys. 241 (2013) 118–140.
[6] V. Kolobov, R. Arslanbekov, J. Phys. Conf. Ser. 719 (1) (2016) 012020.
[7] J.-L. Vay, A. Almgren, J. Bell, L. Ge, D. Grote, M. Hogan, O. Kononenko, R. Lehe, A. Myers, C. Ng, J. Park, R. Ryne, O. Shapoval, M. Thévenet, W. Zhang, Nucl. Instrum. Methods Phys. Res. A (2018).
[8] Trilinos, https://github.com/trilinos/Trilinos, release: 12.14.1, 2019.
[9] H.C. Edwards, C.R. Trott, D. Sunderland, J. Parallel Distrib. Comput. 74 (12) (2014) 3202–3216, https://doi.org/10.1016/j.jpdc.2014.07.003, Domain-Specific Languages and High-Level Frameworks for High-Performance Computing.
[10] H.C. Edwards, D. Sunderland, V. Porter, C. Amsler, S. Mish, Sci. Program. 20 (2) (2012) 89–114, https://doi.org/10.3233/SPR-2012-0343.
[11] A. Adelmann, P. Calvo, M. Frey, A. Gsell, U. Locans, C. Metzger-Kraus, N. Neveu, C. Rogers, S. Russell, S. Sheehy, J. Snuverink, D. Winklehner, arXiv e-prints, arXiv:1905.06654, 2019.
[12] AMReX, https://ccse.lbl.gov/AMReX, release: 18.07, 2019.
[13] D.F. Martin, An Adaptive Cell-centered Projection Method for the Incompressible Euler Equations (PhD thesis), University of California at Berkeley, 1998.
[14] C.G. Baker, M.A. Heroux, Sci. Program. 20 (2) (2012) 115–128.
[15] E. Bavier, M. Hoemmen, S. Rajamanickam, H. Thornquist, Sci. Program. 20 (2012) Issue 3.
[16] L. Berger-Vergiat, C.A. Glusa, J.J. Hu, M. Mayr, A. Prokopenko, C.M. Siefert, R.S. Tuminaro, T.A. Wiesner, MueLu User's Guide, Technical Report SAND2019-0537, Sandia National Laboratories, 2019.
[17] L. Berger-Vergiat, C.A. Glusa, J.J. Hu, M. Mayr, A. Prokopenko, C.M. Siefert, R.S. Tuminaro, T.A. Wiesner, MueLu multigrid framework, http://trilinos.org/packages/muelu, 2019.
[18] A. Prokopenko, C.M. Siefert, J.J. Hu, M. Hoemmen, A. Klinvex, Ifpack2 User's Guide 1.0, Technical Report SAND2016-5338, Sandia National Labs, 2016.
[19] J.J. Yang, A. Adelmann, M. Humbel, M. Seidel, T.J. Zhang, Phys. Rev. ST Accel. Beams 13 (2010) 064201.
[20] V. Rizzoglio, A. Adelmann, C. Baumgarten, M. Frey, A. Gerbershagen, D. Meer, J.M. Schippers, Phys. Rev. Accel. Beams 20 (2017) 124702.
[21] A. Adelmann, P. Arbenz, Y. Ineichen, J. Comput. Phys. 229 (12) (2010) 4554–4566.
[22] M. Toggweiler, A. Adelmann, P. Arbenz, J. Yang, J. Comput. Phys. 273 (2014) 255–267.
[23] J.L. Vay, D.P. Grote, R.H. Cohen, A. Friedman, Comput. Sci. Discov. 5 (1) (2012) 014019.
[24] P. Colella, P.C. Norgaard, J. Comput. Phys. 229 (4) (2010) 947–957.
[25] J.D. Jackson, Classical Electrodynamics, third ed., John Wiley & Sons, Inc., New York, 1999.
[26] M.J. Turk, B.D. Smith, J.S. Oishi, S. Skory, S.W. Skillman, T. Abel, M.L. Norman, Astrophys. J. Suppl. 192 (2011) 9.
[27] D.F. Martin, K.L. Cartwright, Solving Poisson's equation using adaptive mesh refinement, Technical Report UCB/ERL M96/66, Univ. Calif. Berkeley, 1996.
[28] A.S. Almgren, J.B. Bell, P. Colella, L.H. Howell, M.L. Welcome, J. Comput. Phys. 142 (1) (1998) 1–46.
[29] B. Alvin, T. Eli, Comm. Pure Appl. Math. 33 (6) (1980) 707–725.
[30] A. Bayliss, M. Gunzburger, E. Turkel, SIAM J. Appl. Math. 42 (2) (1982) 430–451.
[31] A. Khebir, A.B. Kouki, R. Mittra, IEEE Trans. Microw. Theory Tech. 38 (10) (1990) 1427–1432.
[32] R.K. Gordon, S.H. Fook, IEEE Trans. Microw. Theory Tech. 41 (8) (1993) 1280–1286.
[33] D. Biswas, G. Singh, R. Kumar, Phys. Plasmas 22 (9) (2015) 093119.
[34] T.P. Wangler, K.R. Crandall, Beam Halo in Proton Linac Beams, in: International Linac Conference, vol. 20, 2000.
[35] CSCS, https://www.cscs.ch/computers/piz-daint/, (visited: 8.10.18), 2018.