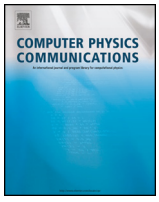
Computer Physics Communications 291 (2023) 108825

Contents lists available at ScienceDirect

Computer Physics Communications

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



Computational Physics

A GPU-parallelized interpolation-based fast multipole method for the relativistic space-charge field calculation ☆

Yi-Kai Kan [a,b,*,1], Franz X. Kärtner [a,b], Sabine Le Borne [c], Jens-Peter M. Zemke [c]

[a] Center for Free-Electron Laser Science CFEL, Deutsches Elektronen-Synchrotron DESY, Germany
[b] Department of Physics, University of Hamburg, Germany
[c] Hamburg University of Technology, Institute of Mathematics, Germany

ARTICLE INFO

Article history:
Received 16 February 2023
Received in revised form 29 April 2023
Accepted 7 June 2023
Available online 12 June 2023

Keywords:
Fast multipole method
Space-charge field calculation
Separable approximation
Admissibility condition
GPU parallelization

ABSTRACT

The fast multipole method (FMM) has received growing attention in the beam physics simulation. In this study, we formulate an interpolation-based FMM for the computation of the relativistic space-charge field. Different to the quasi-electrostatic model, our FMM is formulated in the lab-frame and can be applied without the assistance of the Lorentz transformation. In particular, we derive a modified admissibility condition which can effectively control the interpolation error of the proposed FMM. The algorithms and their GPU parallelization are discussed in detail. A package containing serial and GPU-parallelized solvers is implemented in the Julia programming language. The GPU-parallelized solver can reach a speedup of more than a hundred compared to the execution on a single CPU core.

1. Introduction

The space-charge effect is one of the most important topics in the study of beam physics. In recent years, the fast multipole method (FMM) has attracted increasing attention in the numerical modeling of the space-charge field [1–5]. Compared to the particle-in-cell (PIC) method [6–10] which has been a standard choice in the community of accelerator physics for decades, the FMM has the following advantages in the context of the study of beam physics:

1. Through the point-to-point (P2P) operation, FMMs inherently consider the point-to-point Coulomb effects, *e.g.*, disorder-induced heating and the Boersch effect which are non-negligible in the simulation for cold and dense beams [4].
2. The FMM is a gridless algorithm and can effectively handle charged particle beams with complex geometry [2].

There have been many efforts using FMM for the modeling of electrostatic Coulomb interactions [11,1,4,3]. The electrostatic model is suitable for the study of non-relativistic particle beams, *e.g.*, the simulation of ultrafast electron microscopy [1,4] and the simulation of proton dynamics in synchrotrons [11]. However, for the modeling of energetic electron beams, the consideration of the relativistic effect on the particle field may be essential. One approach to include relativistic effects is the quasi-electrostatic model [12,5] where all particles are assumed stationary in a rest-frame of the particle beam. The electrostatic field on each particle is first solved in the rest-frame and the corresponding field in the lab-frame is calculated through the Lorentz transformation [8,13]. Because of the assumption made in the quasi-electrostatic model, some adjustments are necessary to incorporate the effect of the momentum spread. For example, to handle particle beams with larger energy spread, a technique called energy-binning was proposed by binning particles in energy; the total space-charge field comes from the superposition of the source particle field evaluated in the rest-frame of each bin [14,15]. This study is the extension of our previous work on treecode [16] and consists of two parts. In the first part (Sections 2–6), we formulate an FMM for the efficient computation of the relativistic space-charge field. Our formulation is based on the barycentric Lagrange dual tree traversal

☆ The review of this paper was arranged by Prof. David W. Walker.

* Corresponding author at: Center for Free-Electron Laser Science CFEL, Deutsches Elektronen-Synchrotron DESY, Germany.
E-mail address: YKKan@lbl.gov (Y.-K. Kan).
[1] Current address: Lawrence Berkeley National Laboratory, USA.

https://doi.org/10.1016/j.cpc.2023.108825

(BLDTT) proposed in Ref. [17]. BLDTT uses barycentric Lagrange interpolation for the kernel approximation and dual tree traversal for the construction of interaction lists. Different from the quasi-electrostatic model, the proposed FMM is formulated in the lab-frame and can be applied without the use of the Lorentz transformation. We first introduce the idea of an interpolation-based FMM. After that, we formulate an interpolation-based FMM for the computation of the relativistic space-charge field. In particular, we derive a modified admissibility condition for the cluster-cluster interaction of the relativistic kernel used in the formulated FMM. The algorithms and the implementation details associated with the proposed FMM are also provided. The second part of this study (Sections 8–11) is devoted to the GPU parallelization of the proposed FMM. Different to Ref. [17] which is based on OpenACC (a directive-based programming model) [18], our parallelization is based on the CUDA programming model. We discuss the data structure and the design issues in the implementation of the GPU parallelization. A package containing serial and GPU-parallelized solvers is implemented in the Julia programming language. The performance of the parallel solver is also demonstrated.

2. The idea of FMM

In this section, we give a short overview of the interpolation-based FMM [19,17]. Consider two particle-clusters S_t and S_s. The total force-field f from the source particles in the cluster S_s applied on a target particle with index i and position $\boldsymbol{x}_i \in S_t$ through an interaction kernel $g(\cdot,\cdot)$ can be modeled as

$$f(\boldsymbol{x}_i) = \sum_{j\in\widehat{S}_s} g(\boldsymbol{x}_i, \boldsymbol{x}_j) m_j. \tag{2.1}$$

Here and in the following, $\widehat{S}_t$ and $\widehat{S}_s$ represent the index sets of the particles in S_t and S_s, respectively. The symbol m_j is the physical quantity of the source particle with index j. Although the actual meaning of m_j depends on the physics problem we investigate, without loss of generality, we call it mass throughout this section.

The idea of FMM for a fast evaluation of the summation in (2.1) is based on an approximation of the kernel function by interpolating both the target variable $\boldsymbol{x}_i$ and the source variable $\boldsymbol{x}_j$

$$\boldsymbol{g}(\boldsymbol{x}_i, \boldsymbol{x}_j) \approx \sum_{\boldsymbol{\mu}} \sum_{\boldsymbol{\nu}} \ell_{S_t,\boldsymbol{\mu}}(\boldsymbol{x}_i) \boldsymbol{g}(\boldsymbol{\xi}_{S_t,\boldsymbol{\mu}}, \boldsymbol{\xi}_{S_s,\boldsymbol{\nu}}) \ell_{S_s,\boldsymbol{\nu}}(\boldsymbol{x}_j). \tag{2.2}$$

Here, we define the bounding box Q of a cluster S as

$$Q = [a_x, b_x] \times [a_y, b_y] \times [a_z, b_z] \subset \mathbb{R}^3$$

with $a_g = \min_{i\in\widehat{S}}\{g\}$, $b_g = \max_{i\in\widehat{S}}\{g\}$ and $g \in \{x, y, z\}$. The Lagrange basis polynomials over the bounding boxes of S_t and S_s are defined as

$$\begin{aligned}
\ell_{S_t,\boldsymbol{\mu}}(\boldsymbol{x}_i) &:= \ell^x_{S_t,\mu_1}(x_i) \cdot \ell^y_{S_t,\mu_2}(y_i) \cdot \ell^z_{S_t,\mu_3}(z_i),\\
\ell_{S_s,\boldsymbol{\nu}}(\boldsymbol{x}_j) &:= \ell^x_{S_s,\nu_1}(x_j) \cdot \ell^y_{S_s,\nu_2}(y_j) \cdot \ell^z_{S_s,\nu_3}(z_j),
\end{aligned}$$

with the corresponding interpolation points

$$\boldsymbol{\xi}_{S_t\boldsymbol{\mu}} := (\xi_{S_t,\mu_1}, \xi_{S_t,\mu_2}, \xi_{S_t,\mu_3}) \quad \text{and} \quad \boldsymbol{\xi}_{S_s\boldsymbol{\nu}} := (\xi_{S_s,\nu_1}, \xi_{S_s,\nu_2}, \xi_{S_t,\nu_3}).$$

Substituting (2.2) into (2.1), we have

$$\begin{aligned}
f(\boldsymbol{x}_i) &= \sum_{j\in\widehat{S}_s} g(\boldsymbol{x}_i, \boldsymbol{x}_j) m_j\\
&\approx \sum_{\boldsymbol{\mu}} \ell_{S_t,\boldsymbol{\mu}}(\boldsymbol{x}_i) \sum_{\boldsymbol{\nu}} \boldsymbol{g}(\boldsymbol{\xi}_{S_t,\boldsymbol{\mu}}, \boldsymbol{\xi}_{S_s,\boldsymbol{\nu}}) \underbrace{\sum_{j\in\widehat{S}_s} \ell_{S_s,\boldsymbol{\nu}}(\boldsymbol{x}_j) m_j}_{=:M_{S_s,\boldsymbol{\nu}}}\\
&= \sum_{\boldsymbol{\mu}} \ell_{S_t,\boldsymbol{\mu}}(\boldsymbol{x}_i) \underbrace{\sum_{\boldsymbol{\nu}} \boldsymbol{g}(\boldsymbol{\xi}_{S_t,\boldsymbol{\mu}}, \boldsymbol{\xi}_{S_s,\boldsymbol{\nu}}) M_{S_s,\boldsymbol{\nu}}}_{=:L_{S_t,\boldsymbol{\mu}}}\\
&= \sum_{\boldsymbol{\mu}} \ell_{S_t,\boldsymbol{\mu}}(\boldsymbol{x}_i) L_{S_t,\boldsymbol{\mu}}.
\end{aligned} \tag{2.3}$$

By observing (2.3), we identify the four of FMM kernels:

- P2M (point to multipole): the micro particles in the cluster S_s are aggregated into a few macro particles and the mass of each macro particle ($M_{S_s,\boldsymbol{\nu}}$, also called multipole) can be computed by

$$M_{S_s,\boldsymbol{\nu}} := \sum_{j\in\widehat{S}_s} \ell_{S_s,\boldsymbol{\nu}}(\boldsymbol{x}_j) m_j. \tag{2.4}$$

- M2L (multipole to local): the multipoles of the source cluster are used to evaluate the force-fields acting on the macro particles ($L_{S_t,\boldsymbol{\mu}}$, also called local field) in the target cluster S_t

$$L_{S_t,\mu} := \sum_{\boldsymbol{\nu}} \boldsymbol{g}(\boldsymbol{\xi}_{S_t,\boldsymbol{\mu}}, \boldsymbol{\xi}_{S_s,\boldsymbol{\nu}}) M_{S_s,\boldsymbol{\nu}}. \tag{2.5}$$

- L2P (local to point): in the target cluster, the effective force-fields acting on the macro particles are transferred to the micro particle at $\boldsymbol{x}_i$ by

$$f(\boldsymbol{x_i}) = \sum_{\boldsymbol{\mu}} \ell_{S_t,\boldsymbol{\mu}}(\boldsymbol{x}_i) L_{S_t,\boldsymbol{\mu}}. \tag{2.6}$$

- P2P (point to point): if S_t and S_s do not fulfill an admissibility condition (ADMC, also called multipole acceptance criteria MAC in some literature [20,21]) so that (2.2) is not applicable, the force-field can be calculated directly by

$$f(\boldsymbol{x}_i) = \sum_{j \in \widehat{S}_s} g(\boldsymbol{x}_i, \boldsymbol{x}_j) m_j \quad \forall i \in \widehat{S}_t. \tag{2.7}$$

  This also applies for the case $S_t = S_s = S$, where $i,\ j \in \widehat{S}$ and $i \neq j$.

One main feature of the FMM is the consideration of the cluster-cluster interaction (M2L) through macro particles; and hence, the total number of operations for the evaluation of force-fields can be drastically reduced. In the FMM, we first partition all particles in the system into a hierarchy of clusters (cluster tree). If we directly use (2.4) to compute the multipole of each cluster, the number of operations for computing the multipoles of the whole cluster tree is

$$\text{const} \cdot N \cdot N_r \cdot \log\left(\frac{N}{N_r}\right)$$

with the assumption that the number of macro particles used for the approximation and the number of micro particles contained in the leaf cluster are both N_r. To reduce the total number of operations for computing the multipoles, we can make use of the following property of polynomial interpolation stated in Theorem 1.

Theorem 1. *If $P(x)$ is a polynomial function of degree n, we have*

$$P(x) = \sum_{k=0}^{n} P(\xi_{S,k}) \ell_{S,k}(x) \quad \forall x \in S \tag{2.8}$$

with $\ell_{S,k}(x)$ denoting the Lagrange basis for polynomials of degree $\leq n$ for the interpolation point $\xi_{S,k} \in S$.

This equality can be seen by the fundamental theorem of algebra since both LHS and RHS have the same values at the $n+1$ points $\{\xi_{S,k} \mid k = 0, \ldots, n\}$ and the RHS is a polynomial of degree n. By using Theorem 1, we can introduce two further procedures and two kernels of FMM:

- Upward Pass: a source particle cluster S is subdivided into a hierarchy of clusters of the depth κ (called cluster tree [22,23]). The multipoles of each cluster can be computed by the multipoles of its children clusters in view of

$$\begin{aligned} M_{S,\boldsymbol{\nu}} &= \sum_{j \in \widehat{S}} \ell_{S,\boldsymbol{\nu}}(\boldsymbol{x}_j) m_j \\ &= \sum_{s' \in \text{children}(S)} \sum_{j \in \widehat{s'}} \ell_{S,\boldsymbol{\nu}}(\boldsymbol{x}_j) m_j \\ &\overset{(2.8)}{=} \sum_{s' \in \text{children}(S)} \sum_{j \in \widehat{s'}} \sum_{\boldsymbol{\nu}'} \ell_{S,\boldsymbol{\nu}}(\xi_{s',\boldsymbol{\nu}'}) \ell_{s',\boldsymbol{\nu}'}(\boldsymbol{x}_j) m_j \\ &= \sum_{s' \in \text{children}(S)} \sum_{\boldsymbol{\nu}'} \ell_{S,\boldsymbol{\nu}}(\xi_{s',\boldsymbol{\nu}'}) \sum_{j \in \widehat{s'}} \ell_{s',\boldsymbol{\nu}'}(\boldsymbol{x}_j) m_j \\ &= \sum_{s' \in \text{children}(S)} \sum_{\boldsymbol{\nu}'} \ell_{S,\boldsymbol{\nu}}(\xi_{s',\boldsymbol{\nu}'}) M_{s',\boldsymbol{\nu}'}. \end{aligned} \tag{2.9}$$

  Equation (2.9) is the formula of the M2M (multipole to multipole) kernel. In FMM, the multipoles of the source cluster tree are updated by a procedure called upward pass. In the upward pass, the multipoles of the leaf clusters are first evaluated with P2M (2.4); and then, we start from the second deepest level of the cluster tree (*i.e.*, level $\kappa - 1$) and apply M2M (2.9) to compute the multipoles of each cluster in each level (level by level).

- Downward Pass: a target cluster S is subdivided into a cluster tree of the depth κ and each target particle (say particle i) will be contained in a sequence of clusters $\{S^l \mid l = 0, \ldots, \kappa\}$ from each level l with $S^{l+1} \subset S^l$ and $S^0 = S$. The force-field on the target particle i can be calculated by

$$f(\boldsymbol{x}_i) = \sum_{l=0}^{\kappa} \sum_{\boldsymbol{\mu}} \ell_{S^l,\boldsymbol{\mu}}(\boldsymbol{x}_i) L_{S^l,\boldsymbol{\mu}} = \sum_{\boldsymbol{\mu}} \mathcal{L}_{S^\kappa,\boldsymbol{\mu}} \ell_{S^\kappa,\boldsymbol{\mu}}(\boldsymbol{x}_i). \tag{2.10}$$

Here, we define the "cumulative local field" $\mathcal{L}_{S^l}$, which follows the recursive relation

$$\mathcal{L}_{S^l,\boldsymbol{\mu}} := L_{S^l,\boldsymbol{\mu}} + \sum_{\boldsymbol{\mu}'} \mathcal{L}_{S^{l-1},\boldsymbol{\mu}'} \cdot \ell_{S^{l-1},\boldsymbol{\mu}'}(\boldsymbol{\xi}_{S^l,\boldsymbol{\mu}}) \quad \text{with} \quad \mathcal{L}_{S^0,\boldsymbol{\mu}} := L_{S^0,\boldsymbol{\mu}}. \tag{2.11}$$

Equation (2.10) can be proved by using (2.8), (2.11) and mathematical induction (cf. Appendix A). Therefore, during the downward pass of FMM, we first perform L2L (2.11) to calculate the cumulative local fields of the deepest-level cluster S^κ; afterward, we transfer $\mathcal{L}_{S^\kappa,\mu}$ to the target particles contained in S^κ via L2P (2.10).

3. FMM formulation for the efficient computation of relativistic space-charge field

Consider a relativistic charged particle beam moving in z-direction. Inside the particle beam, the space-charge field from a source particle with the position $\boldsymbol{x}_j$ exerting to a target particle with the position $\boldsymbol{x}_i$ can be approximately written as [16]

$$\boldsymbol{E}(\boldsymbol{x}_i, \boldsymbol{x}_j) \approx \frac{q}{4\pi\epsilon_0} \gamma_j \boldsymbol{g}(\boldsymbol{x}, \boldsymbol{x}_j) \quad \text{and} \quad \boldsymbol{B}(\boldsymbol{x}_i, \boldsymbol{x}_j) \approx \frac{q}{4\pi\epsilon_0 c_0} \boldsymbol{p}_j \times \boldsymbol{g}(\boldsymbol{x}, \boldsymbol{x}_j), \tag{3.1}$$

with the kernel function called "relativistic kernel"

$$\boldsymbol{g}(\boldsymbol{x}_i, \boldsymbol{x}_j) := \frac{\boldsymbol{x}_i - \boldsymbol{x}_j}{\left((x_i - x_j)^2 + (y_i - y_j)^2 + \overline{\gamma}^2 (z_i - z_j)^2\right)^{3/2}}, \tag{3.2}$$

where $\gamma_j = 1/\sqrt{1 - \|\boldsymbol{\beta}_j\|_2^2}$ and $\boldsymbol{p}_j = \gamma_j \boldsymbol{\beta}_j$ are the Lorentz factor and normalized momentum of the particle, respectively, with $\boldsymbol{\beta}_j$ the particle velocity normalized to the speed of light c_0. Here, $\overline{\gamma}$ is the average Lorentz factor and can be computed by $\overline{\gamma}^2 = 1 + \overline{\boldsymbol{p}} \cdot \overline{\boldsymbol{p}}$ with the average momentum of the particle beam $\overline{\boldsymbol{p}}$. Throughout this study, we assume that all particles in the particle beam are the same type with charge q.

Given a target particle with the position $\boldsymbol{x}_i$ contained in a target cluster S_t, the space-charge field from all the particles in the cluster S_s experienced by this target particle can be computed approximately by applying (2.2) to (3.1),

$$\begin{aligned}
\sum_{j \in \widehat{S}_s} \boldsymbol{E}(\boldsymbol{x}_i, \boldsymbol{x}_j) &\approx \frac{q}{4\pi\epsilon_0} \sum_{\boldsymbol{\mu}} \sum_{\boldsymbol{\nu}} \sum_{j \in \widehat{S}_s} \ell_{S_t,\boldsymbol{\mu}}(\boldsymbol{x}_i) \ell_{S_s,\boldsymbol{\nu}}(\boldsymbol{x}_j) \gamma_j \boldsymbol{g}(\boldsymbol{\xi}_{S_t,\boldsymbol{\mu}}, \boldsymbol{\xi}_{S_s,\boldsymbol{\nu}}) \\
&= \frac{q}{4\pi\epsilon_0} \sum_{\boldsymbol{\mu}} \ell_{S_t,\boldsymbol{\mu}}(\boldsymbol{x}_i) \boldsymbol{E}_{S_t,\boldsymbol{\mu}}, \\
\sum_{j \in \widehat{S}_s} \boldsymbol{B}(\boldsymbol{x}_i, \boldsymbol{x}_j) &\approx \frac{q}{4\pi\epsilon_0 c_0} \sum_{\boldsymbol{\mu}} \sum_{\boldsymbol{\nu}} \sum_{j \in \widehat{S}_s} \ell_{S_t,\boldsymbol{\mu}}(\boldsymbol{x}_i) \ell_{S_s,\boldsymbol{\nu}}(\boldsymbol{x}_j) \boldsymbol{p}_j \times \boldsymbol{g}(\boldsymbol{\xi}_{S_t,\boldsymbol{\mu}}, \boldsymbol{\xi}_{S_s,\boldsymbol{\nu}}) \\
&= \frac{q}{4\pi\epsilon_0 c_0} \sum_{\boldsymbol{\mu}} \ell_{S_t,\boldsymbol{\mu}}(\boldsymbol{x}_i) \boldsymbol{B}_{S_t,\boldsymbol{\mu}}.
\end{aligned} \tag{3.3}$$

Here, we introduce the effective Lorentz factor and the effective momentum of a macro particle with the position $\boldsymbol{\xi}_{S_s,\boldsymbol{\nu}}$ and index $\boldsymbol{\nu}$ in the cluster S_s as

$$\gamma_{S_s,\boldsymbol{\nu}} := \sum_{j \in \widehat{S}_s} \ell_{S_s,\boldsymbol{\nu}}(\boldsymbol{x}_j) \gamma_j \quad \text{and} \quad \boldsymbol{p}_{S_s,\boldsymbol{\nu}} := \sum_{j \in \widehat{S}_s} \ell_{S_s,\boldsymbol{\nu}}(\boldsymbol{x}_j) \boldsymbol{p}_j.$$

Similarly, the effective electric and magnetic fields experienced by a macro particle with the position $\boldsymbol{\xi}_{S_t,\boldsymbol{\mu}}$ and index $\boldsymbol{\mu}$ in the target cluster S_t are defined as

$$\boldsymbol{E}_{S_t,\boldsymbol{\mu}} := \sum_{\boldsymbol{\nu}} \gamma_{S_s,\boldsymbol{\nu}} \boldsymbol{g}(\boldsymbol{\xi}_{S_t,\boldsymbol{\mu}}, \boldsymbol{\xi}_{S_s,\boldsymbol{\nu}}) \quad \text{and} \quad \boldsymbol{B}_{S_t,\boldsymbol{\mu}} := \sum_{\boldsymbol{\nu}} \boldsymbol{p}_{S_s,\boldsymbol{\nu}} \times \boldsymbol{g}(\boldsymbol{\xi}_{S_t,\boldsymbol{\mu}}, \boldsymbol{\xi}_{S_s,\boldsymbol{\nu}}).$$

4. Admissibility condition for cluster-cluster interaction of the relativistic kernel

In the previous section, we used Lagrangian interpolation to approximate the space-charge field on a target particle in a relativistic particle beam. It is also of importance to know when this approximation can be applied. To answer this question, we may investigate the interpolation error bound of the relativistic kernel

$$\|\boldsymbol{g}(\boldsymbol{x}_i, \boldsymbol{x}_j) - \widetilde{\boldsymbol{g}}(\boldsymbol{x}_i, \boldsymbol{x}_j)\|_{\infty, Q_t \times Q_s} \leq B_t + B_s. \tag{4.1}$$

Here, B_t and B_s are the interpolation error bounds with respect to the target variable $\boldsymbol{x}_i := (x_i, y_i, z_i)$ and the source variable $\boldsymbol{x}_j := (x_j, y_j, z_j)$:

$$B_t := \text{const} \cdot \sum_{k\in\{x_i,y_i,z_i\}} (b_k - a_k)^{n+1} \cdot \frac{\|\partial_k^{n+1} \boldsymbol{g}(\boldsymbol{x}_i, \boldsymbol{x}_j)\|_{\infty, Q_t\times Q_s}}{(n+1)!},$$

$$B_s := \text{const} \cdot \sum_{k\in\{x_j,y_j,z_j\}} (b_k - a_k)^{n+1} \cdot \frac{\|\partial_k^{n+1} \boldsymbol{g}(\boldsymbol{x}_i, \boldsymbol{x}_j)\|_{\infty, Q_t\times Q_s}}{(n+1)!},$$

where the bounding boxes of S_t and S_s are

$$Q_t = [a_{x_i}, b_{x_i}] \times [a_{y_i}, b_{y_i}] \times [a_{z_i}, b_{z_i}] \subset \mathbb{R}^3,$$
$$Q_s = [a_{x_j}, b_{x_j}] \times [a_{y_j}, b_{y_j}] \times [a_{z_j}, b_{z_j}] \subset \mathbb{R}^3.$$

Following the similar analysis in Ref. [16], we can derive the error bounds of B_t and B_s respectively as

$$B_t \le \text{const} \cdot \sum_{k\in\{x_i,y_i,z_i\}} s_k^{n+1}(b_k - a_k)^{n+1} \cdot \frac{1}{\|\boldsymbol{s}\circ(\boldsymbol{x}_i - \boldsymbol{x}_j)\|_\infty^{n+3}} \le \frac{\text{const}}{\overset{(1,1,\overline{\gamma})}{\text{dist}}(S_t, S_s)^2} \frac{\overset{(1,1,\overline{\gamma})}{\text{diam}}(S_t)^{n+1}}{\overset{(1,1,\overline{\gamma})}{\text{dist}}(S_t, S_s)^{n+1}},$$

$$B_s \le \text{const} \cdot \sum_{k\in\{x_j,y_j,z_j\}} s_k^{n+1}(b_k - a_k)^{n+1} \cdot \frac{1}{\|\boldsymbol{s}\circ(\boldsymbol{x}_i - \boldsymbol{x}_j)\|_\infty^{n+3}} \le \frac{\text{const}}{\overset{(1,1,\overline{\gamma})}{\text{dist}}(S_t, S_s)^2} \frac{\overset{(1,1,\overline{\gamma})}{\text{diam}}(S_s)^{n+1}}{\overset{(1,1,\overline{\gamma})}{\text{dist}}(S_t, S_s)^{n+1}},$$

where we define a stretched vector $\boldsymbol{s} := (s_x, s_y, s_z) = (1, 1, \overline{\gamma})$ with the average Lorentz factor $\overline{\gamma}$ in (3.2). The symbol $\circ : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}^n$ denotes the component-wise product of two vectors. Here, the stretched diameter of a cluster and the stretched distance between two clusters S_t and S_s are

$$\begin{aligned} \overset{(s_x,s_y,s_z)}{\text{diam}}(S) &:= \max_{\boldsymbol{x}_i,\boldsymbol{x}_j\in S} \|(s_x, s_y, s_z)\circ(\boldsymbol{x}_i - \boldsymbol{x}_j)\|_2, \\ \overset{(s_x,s_y,s_z)}{\text{dist}}(S_t, S_s) &:= \min_{\substack{\boldsymbol{x}_i\in S_t \\ \boldsymbol{x}_j\in S_s}} \|(s_x, s_y, s_z)\circ(\boldsymbol{x}_i - \boldsymbol{x}_j)\|_2. \end{aligned} \tag{4.2}$$

Therefore, the interpolation error (4.1) can be bounded by

$$\begin{aligned} \|\boldsymbol{g}(\boldsymbol{x}_i, \boldsymbol{x}_j) - \widetilde{\boldsymbol{g}}(\boldsymbol{x}_i, \boldsymbol{x}_j)\|_{\infty, Q_t\times Q_s} &\le \text{const} \cdot \frac{\overset{(1,1,\overline{\gamma})}{\text{diam}}(S_t)^{n+1} + \overset{(1,1,\overline{\gamma})}{\text{diam}}(S_s)^{n+1}}{\overset{(1,1,\overline{\gamma})}{\text{dist}}(S_t, S_s)^{n+1}} \\ &\le \text{const} \cdot \left(\frac{\overset{(1,1,\overline{\gamma})}{\text{diam}}(S_t) + \overset{(1,1,\overline{\gamma})}{\text{diam}}(S_s)}{\overset{(1,1,\overline{\gamma})}{\text{dist}}(S_t, S_s)} \right)^{n+1} \\ &\le \text{const} \cdot \left(\frac{\max\left(\overset{(1,1,\overline{\gamma})}{\text{diam}}(S_t), \overset{(1,1,\overline{\gamma})}{\text{diam}}(S_s) \right)}{\overset{(1,1,\overline{\gamma})}{\text{dist}}(S_t, S_s)} \right)^{n+1}, \end{aligned}$$

and we can define an admissibility condition for the cluster-cluster interaction of the relativistic kernel by

$$\frac{\max\left(\overset{(1,1,\overline{\gamma})}{\text{diam}}(S_t), \overset{(1,1,\overline{\gamma})}{\text{diam}}(S_s) \right)}{\overset{(1,1,\overline{\gamma})}{\text{dist}}(S_t, S_s)} < \eta \tag{4.3}$$

with some admissibility parameter $\eta \in \mathbb{R}_{>0}$ which can be chosen to control the interpolation error bound.

Besides deriving the stretched admissibility condition for the relativistic kernel, it is possible to bypass the mathematical analysis by using the Lorentz transformation. In the rest-frame of a particle beam with $\overline{\boldsymbol{p}} = 0$, the relativistic kernel $\boldsymbol{g}$ is approximately equal to the electrostatic kernel and the conventional admissibility condition can be used for controlling the interpolation error. However, as already discussed in our previous work [16], this approach can result in a larger error when a particle beam with larger momentum spread is considered because the distance of each target-source pair in the rest-frame of the particle beam is not correctly evaluated. Therefore, we will not discuss this approach in this study.

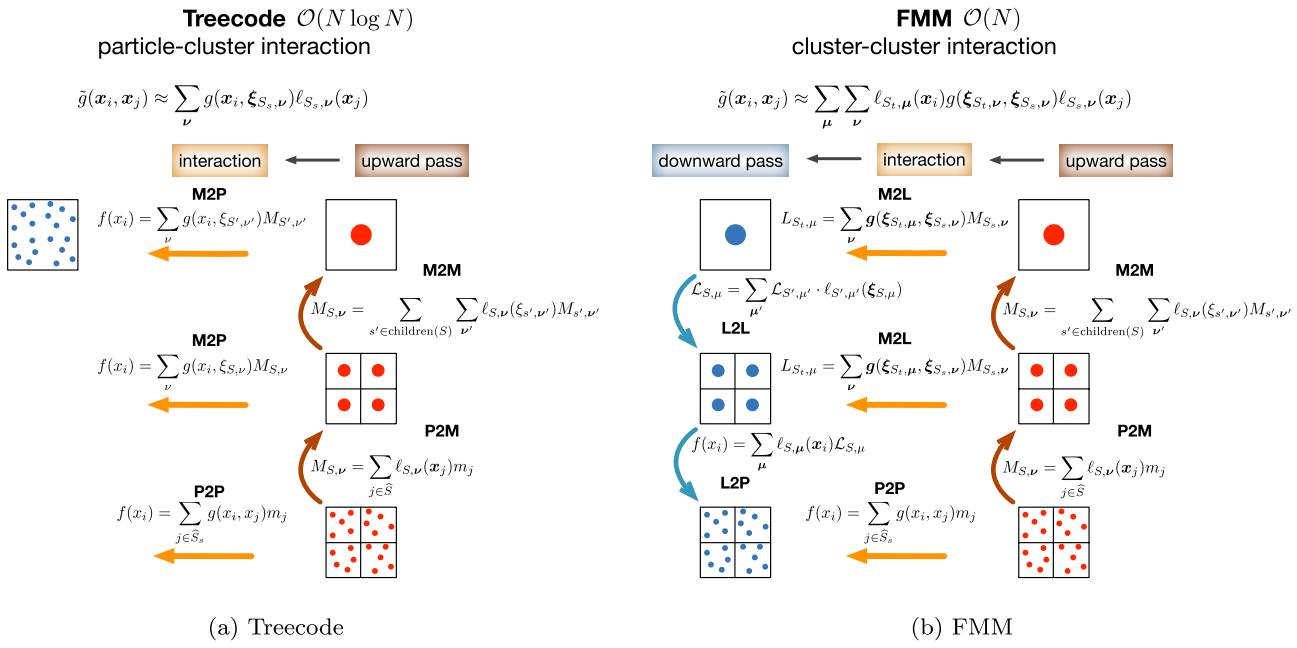


Fig. 1. A schematic comparison of treecode and FMM. The blue dots and red dots indicate the target and source particles, respectively. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

5. The procedure of FMM

With the FMM kernels introduced in Section 2 and the stretched admissibility condition for the cluster-cluster interaction of the relativistic kernel derived in Section 4, we can formulate an FMM for the calculation of relativistic space-charge field. The proposed FMM (Algorithm 5.12) can be summarized in the following procedures:

1. A cluster tree using a k-d tree with a cardinality-balanced subdivision scheme [16, Section 3] is constructed from the particles in the system. To control the interpolation error subject to the relativistic kernel (3.2) with the stretched admissibility condition (4.3), the effect of the stretch also needs to be considered in the particle cluster subdivision [16]. Hence, the cluster tree is constructed through Algorithm 5.2 with $\boldsymbol{s} = (1, 1, \overline{\gamma})$.
2. In the upward pass (Algorithm 5.5), the multipoles of each leaf cluster is computed by P2M (Algorithm 5.3) and are transferred to the multipoles of its ascendants by M2M (Algorithm 5.4).
3. A list of interaction pairs is determined dynamically by the dual tree traversal [24,25]. The corresponding pseudocode is Algorithm 5.11. In the simulation of the space-charge effect for a particle beam, the roots of the target tree S_t and the source tree S_s are an identical cluster S, *i.e.*, $S_t = S_s = S$. For the pair of clusters fulfilling the admissibility condition (Algorithm 5.8 with $\boldsymbol{s} = (1, 1, \overline{\gamma})$), the local field of the target cluster is computed by M2L (Algorithm 5.9). For the interaction pair of leaf clusters, the force-fields on the particles in the target cluster are computed by P2P (Algorithm 5.10). In our implementation, instead of using (4.2), we adapt a different definition to compute the stretched diameter and the stretched distance as illustrated respectively in Algorithm 5.6 and Algorithm 5.7 because of their simplicity in the practical implementation.
4. In the downward pass (Algorithm 5.15), the cumulative local fields of each cluster are transferred to its descendants by L2L (Algorithm 5.13) and the cumulative local fields of each leaf cluster are transferred to the force-field on its member particles by L2P (Algorithm 5.14).

A schematic comparison of treecode [20] and FMM is illustrated in Fig. 1. In the treecode (Fig. 1a), we interpolate the source variable $\boldsymbol{x}_j$ of the kernel function so that we can cluster the source particles and build up the effective masses of each cluster. The force-field of each particle in the target cluster is evaluated by each independent traversal of the source cluster tree and by investigating the particle-cluster interaction. In the FMM (Fig. 1b), we interpolate both target variable $\boldsymbol{x}_i$ and source variable $\boldsymbol{x}_j$ of the kernel function so that the target particles and source particles can be clustered and the corresponding effective force-fields and masses of each cluster can be built up. The force-field on a target particle is transferred from the effective force-fields of the clusters, which are computed by cluster-cluster interaction through the traversal of the target cluster tree and the source cluster tree simultaneously.

The pseudocodes for the algorithms are presented with the following global variables:

$\boldsymbol{E}_i$	electric field experienced by the i-th particle,
$\boldsymbol{B}_i$	magnetic field experienced by the i-th particle,
$\gamma_{S,\boldsymbol{\nu}}$	effective Lorenz factor of a macro particle with index $\boldsymbol{\nu}$ in cluster S,
$\boldsymbol{p}_{S,\boldsymbol{\nu}}$	effective momentum of a macro particle with index $\boldsymbol{\nu}$ in cluster S,
$\boldsymbol{E}_{S,\boldsymbol{\nu}}$	total electric field acting on a macro particle with index $\boldsymbol{\nu}$ in cluster S,
$\boldsymbol{B}_{S,\boldsymbol{\nu}}$	total magnetic field acting on a macro particle with index $\boldsymbol{\nu}$ in cluster S.

The algorithms described above are implemented as a solver in the Julia programming language [26]. The cluster tree constructed with Algorithm 5.2 is implemented using a pointer-based data structure.

Algorithm 5.1: Direction for the Split of Particle Cluster with Stretch.

S: particle cluster
$\boldsymbol{s}$: stretch factor
Function direction4split(S, $\boldsymbol{s}$)
  $(s_x, s_y, s_z) = \boldsymbol{s}$
  $[a_x, b_x] \times [a_y, b_y] \times [a_z, b_z] = \mathrm{bbox}(S)$
  $k = \underset{i \in \{x,y,z\}}{\operatorname{argmax}}\ s_i(b_i - a_i)$
  return k
end

Algorithm 5.2: Subdivision of Particle Cluster.

S: particle cluster
$\boldsymbol{s}$: stretch factor
N_0: maximum number of particles in the leaf node
Function subdivide(S, $\boldsymbol{s}$, N_0)
  if $|S| > N_0$ **then**
    $k = \mathrm{direction4split}(S, \boldsymbol{s})$ (Algorithm 5.1)
    $k_{\mathrm{split}} = \lfloor |S|/2 \rfloor$-th largest element of $\{k_i \mid i \in \widehat{S}\}$
    $S_1 = \{\boldsymbol{x}_i \mid k_i \leq k_{\mathrm{split}}, i \in \widehat{S}\}$
    $S_2 = \{\boldsymbol{x}_i \mid k_i > k_{\mathrm{split}}, i \in \widehat{S}\}$
    children(S) $= \{S_1, S_2\}$
    subdivide(S_1, $\boldsymbol{s}$, N_0)
    subdivide(S_2, $\boldsymbol{s}$, N_0)
  else
    children(S) $= \emptyset$
  end
end

Algorithm 5.3: P2M.

S: particle cluster
Function P2M(S)
  $\gamma_{S,\boldsymbol{v}} = \sum_{j \in \widehat{S}} \ell_{S,\boldsymbol{v}}(\boldsymbol{x}_j)\gamma_j$
  $\boldsymbol{p}_{S,\boldsymbol{v}} = \sum_{j \in \widehat{S}} \ell_{S,\boldsymbol{v}}(\boldsymbol{x}_j)\boldsymbol{p}_j$
end

Algorithm 5.4: M2M.

S: parent particle cluster
S': child particle cluster
Function M2M(S, S')
  $\gamma_{S,\boldsymbol{v}} = \sum_{\boldsymbol{v}'} \ell_{S,\boldsymbol{v}}(\xi_{S',\boldsymbol{v}'})\gamma_{S',\boldsymbol{v}'}$
  $\boldsymbol{p}_{S,\boldsymbol{v}} = \sum_{\boldsymbol{v}'} \ell_{S,\boldsymbol{v}}(\xi_{S',\boldsymbol{v}'})\gamma_{S',\boldsymbol{v}'}$
end

Algorithm 5.5: Upward Pass.

S: particle cluster
Function upwardpass(S)
  if children(S) $== \emptyset$ **then**
    P2M(S) (Algorithm 5.3)
  else
    for $s \in$ children(S) **do**
      upwardpass(s)
    end
    for $s \in$ children(S) **do**
      M2M(S, s) (Algorithm 5.4)
    end
  end
end

Algorithm 5.6: Stretched Diameter of Cluster.

S: particle cluster
$\boldsymbol{s}$: stretch factor
Function diam$(S, \boldsymbol{s})$
  $[a_x, b_x] \times [a_y, b_y] \times [a_z, b_z] = \text{bbox}(S)$
  return $\|\boldsymbol{s} \circ (\boldsymbol{a} - \boldsymbol{b})/2\|_2$
end

Algorithm 5.7: Stretched Distance between two Clusters.

S_1: particle cluster 1
S_2: particle cluster 2
$\boldsymbol{s}$: stretch factor
Function dist$(S_1, S_2, \boldsymbol{s})$
  $[a_{1,x}, b_{1,x}] \times [a_{1,y}, b_{1,y}] \times [a_{1,z}, b_{1,z}] = \text{bbox}(S_1)$
  $[a_{2,x}, b_{2,x}] \times [a_{2,y}, b_{2,y}] \times [a_{2,z}, b_{2,z}] = \text{bbox}(S_2)$
  $\boldsymbol{c}_1 = (\boldsymbol{a}_1 + \boldsymbol{b}_1)/2$
  $\boldsymbol{c}_2 = (\boldsymbol{a}_2 + \boldsymbol{b}_2)/2$
  return $\|\boldsymbol{s} \circ (\boldsymbol{c}_1 - \boldsymbol{c}_2)\|_2$
end

Algorithm 5.8: Stretched Admissibility Condition for Cluster-Cluster Interaction.

S_1: particle cluster 1
S_2: particle cluster 2
$\boldsymbol{s}$: stretch factor
η: admissibility parameter
Function admissible$(S_1, S_2, \boldsymbol{s}, \eta)$
  $r_1 = \text{diam}(S_1, \boldsymbol{s})$ (Algorithm 5.6)
  $r_2 = \text{diam}(S_2, \boldsymbol{s})$ (Algorithm 5.6)
  $d = \text{dist}(S_1, S_2, \boldsymbol{s})$ (Algorithm 5.7)
  return $\max(r_1, r_2)/d < \eta$
end

Algorithm 5.9: M2L.

S_t: target particle cluster
S_s: source particle cluster
Function M2L(S_t, S_s)
  $\boldsymbol{E}_{S_t,\boldsymbol{\mu}} = \sum_{\boldsymbol{\nu}} \gamma_{S_s,\boldsymbol{\nu}} \cdot \boldsymbol{g}(\xi_{S_t,\boldsymbol{\mu}}, \xi_{S_s,\boldsymbol{\nu}})$
  $\boldsymbol{B}_{S_t,\boldsymbol{\mu}} = \sum_{\boldsymbol{\nu}} \boldsymbol{p}_{S_s,\boldsymbol{\nu}} \times \boldsymbol{g}(\xi_{S_t,\boldsymbol{\mu}}, \xi_{S_s,\boldsymbol{\nu}})$
end

Algorithm 5.10: P2P.

S_t: target particle cluster
S_s: source particle cluster
Function P2P(S_t, S_s)
  for $i \in \hat{S}_t$ **do**
    for $j \in \hat{S}_s$ **do**
      $\boldsymbol{E}_i = \boldsymbol{E}_i + \gamma_j \boldsymbol{g}(\boldsymbol{x}_i, \boldsymbol{x}_j, \boldsymbol{p}_j)$
      $\boldsymbol{B}_i = \boldsymbol{B}_i + \boldsymbol{p}_j \times \boldsymbol{g}(\boldsymbol{x}_i, \boldsymbol{x}_j, \boldsymbol{p}_j)$
    end
  end
end

6. Results

To understand the performance of the proposed FMM, we first demonstrate a plot of elapsed time against the error for the simulations with different FMM parameters in Fig. 2. In each simulation, 1.28×10^6 particles are randomly uniformly distributed in the unit cube $[0, 1]^3$ and each particle has the same momentum $\boldsymbol{p} = (0, 0, p_0)$ with $p_0 = (\gamma^2 - 1)^{1/2}$ and $\gamma = 50$. The measured error is the maximal relative error in the electrical and magnetic fields,

$$\text{error} := \max_{\boldsymbol{f} \in \{\boldsymbol{E}, \boldsymbol{B}\}} \left(\sum_{i=1}^{N} \|\boldsymbol{f}_i^t - \boldsymbol{f}_i^b\|_2^2 / \sum_{i=i}^{N} \|\boldsymbol{f}_i^b\|_2^2 \right)^{1/2}, \tag{6.1}$$

where N is the number of particles in the system. The space-charge fields $\boldsymbol{f}_i^t$ and $\boldsymbol{f}_i^b$ experienced by the i-th particle are computed by FMM and a brute-force method Algorithm 5.10 with $S_t = S_s = S$ and $i \neq j$), respectively. We can observe that a smaller admissibility parameter η (4.3) leads to higher accuracy (smaller error) but costs more elapsed time. This is because fewer M2Ls in the coarse level are performed and each non-admissible pair of clusters in the coarse level can result in many M2Ls in the finer level or P2Ps in the leaf level.

Algorithm 5.11: Cluster-Cluster Interaction by Dual Tree Traversal.

S_t: target particle cluster
S_s: source particle cluster
$\boldsymbol{s}$: stretch factor
η: admissibility parameter
Function dualtraverseinteract($S_t, S_s, \boldsymbol{s}, \eta$)
  if children(S_t) == $\emptyset$ $\wedge$ children(S_s) == $\emptyset$ **then**
    P2P(S_t, S_s) (Algorithm 5.10)
  else
    isAdmissible = admissible(S_t, $S_s, \boldsymbol{s}, \eta$) (Algorithm 5.8)
    if isAdmissible **then**
      M2L(S_t, S_s) (Algorithm 5.9)
    else if children(S_t) == $\emptyset$ **then**
      for $s \in$ children(S_s) **do**
        dualtraverseinteract($S_t, s, \boldsymbol{s}, \eta$)
      end
    else if children(S_s) == $\emptyset$ **then**
      for $t \in$ children(S_t) **do**
        dualtraverseinteract($t, S_s, \boldsymbol{s}, \eta$)
      end
    else
      if diam($S_t, \boldsymbol{s}$) > diam($S_s, \boldsymbol{s}$) **then**
        for $t \in$ children(S_t) **do**
          dualtraverseinteract($t, S_s, \boldsymbol{s}, \eta$)
        end
      else
        for $s \in$ children(S_s) **do**
          dualtraverseinteract($S_t, s, \boldsymbol{s}, \eta$)
        end
      end
    end
  end
end

Algorithm 5.12: FMM with Stretch.

S: particle cluster
$\boldsymbol{s}$: stretch factor
N_0: maximum number of particles in the leaf node
η: admissibility parameter
Function FMM(S, $\boldsymbol{s}$, N_0, η)
  subdivide(S, $\boldsymbol{s}$, N_0) (Algorithm 5.2)
  $\boldsymbol{E}_i = \boldsymbol{0}, \boldsymbol{B}_i = \boldsymbol{0} \quad \forall i \in \widehat{S}$
  upwardpass(S) (Algorithm 5.5)
  dualtraverseinteract($S, S, \boldsymbol{s}, \eta$) (Algorithm 5.11)
  downwardpass(S) (Algorithm 5.15)
end

Algorithm 5.13: L2L.

S: parent particle cluster
S': child particle cluster
Function L2L(S', S)
  $\boldsymbol{E}_{S',\boldsymbol{\mu}'} = \boldsymbol{E}_{S',\boldsymbol{\mu}'} + \sum_{\boldsymbol{\mu}} \ell_{S,\boldsymbol{\mu}}(\xi_{S',\boldsymbol{\mu}'}) \boldsymbol{E}_{S,\boldsymbol{\mu}}$
  $\boldsymbol{B}_{S',\boldsymbol{\mu}'} = \boldsymbol{B}_{S',\boldsymbol{\mu}'} + \sum_{\boldsymbol{\mu}} \ell_{S,\boldsymbol{\mu}}(\xi_{S',\boldsymbol{\mu}'}) \boldsymbol{B}_{S,\boldsymbol{\mu}}$
end

Algorithm 5.14: L2P.

S: particle cluster
Function L2P(S)
  for $i \in \hat{S}$ **do**
    $\boldsymbol{E}_i = \boldsymbol{E}_i + \sum_{\boldsymbol{\mu}} \ell_{S,\boldsymbol{\mu}}(\boldsymbol{x}_i) \boldsymbol{E}_{S,\boldsymbol{\mu}}$
    $\boldsymbol{B}_i = \boldsymbol{B}_i + \sum_{\boldsymbol{\mu}} \ell_{S,\boldsymbol{\mu}}(\boldsymbol{x}_i) \boldsymbol{B}_{S,\boldsymbol{\mu}}$
  end
end

The usage of a higher interpolation degree n leads to a result with higher accuracy and higher elapsed time because more macro particles are used in the calculation of M2L. In Fig. 3, the elapsed time of FMM against the number of particles N is presented. We can see that our FMM approaches the theoretical complexity $\mathcal{O}(N)$ as the number of particles N becomes big enough (Fig. 3b).

We also perform code profiling on our solver and demonstrate the cumulative elapsed time of the six FMM kernels in Fig. 4. One can observe that the total elapsed time is mostly dominated by P2P and M2L; this indicates that the routines of these two kernels will be the focus when any further optimizations for the solver are considered. Besides, one can also observe a sudden jump in the value of the

Algorithm 5.15: Downward Pass.

```
S: particle cluster
Function downwardpass(S)
    if children(S) == ∅ then
        L2P(S) (Algorithm 5.14)
    else
        for s ∈ children(S) do
            L2L(s, S) (Algorithm 5.13)
        end
        for s ∈ children(S) do
            downwardpass(s)
        end
    end
end
```

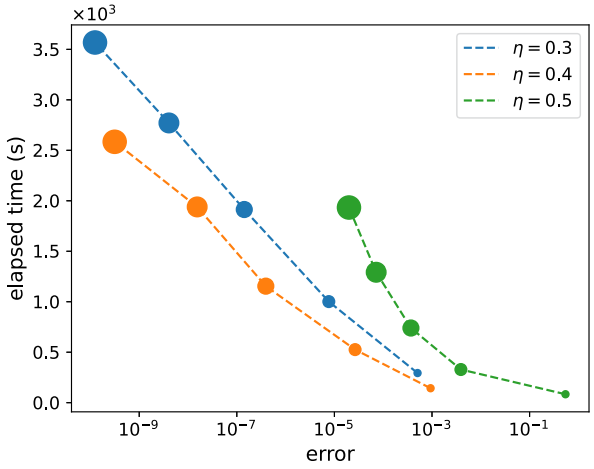
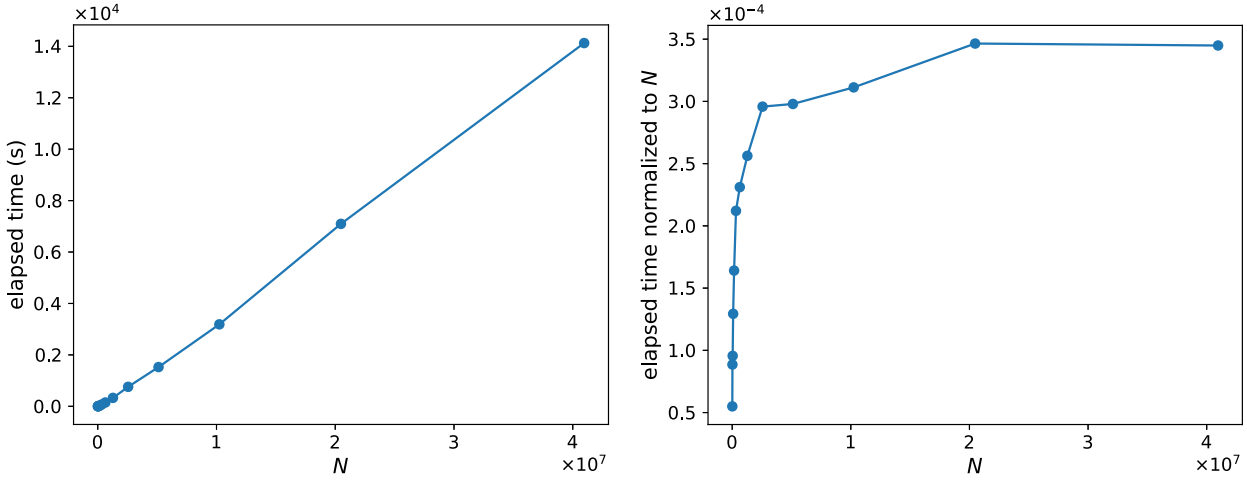


Fig. 2. A plot of elapsed time against the error (6.1) for the proposed FMM. Each line represents the result computed with an admissibility parameter $\eta = 0.3, 0.4, 0.5$. Each point in a line represents a simulation with an interpolation degree $n = 2, 4, 6, 8, 10$ and the maximum number of particles in the leaf cluster $N_0 = (n+1)^3$. The point size is associated with the value of n; data with bigger n is expressed with bigger point size.



(a) elapsed time (b) normalized elapsed time

Fig. 3. The performance of the FMM method. Fig. 3a shows the elapsed time used by FMM to evaluate the space-charge field of increasing numbers of particles N with interpolation degree $n = 4$, the maximum number of particles in the leaf cluster $N_0 = (n+1)^3$ and admissibility parameter $\eta = 0.5$. Fig. 3b shows the elapsed time normalized to N.

elapsed times for P2P and M2L at a specific number of particles N. This behavior was demonstrated and briefly explained in Ref. [27]. Here, we provide a performance model for the illustration. We consider a case where the total particle number is equal to a transition value $N = N_t^\kappa := 2^\kappa N_0$ with κ the depth of the cluster tree. If each leaf cluster interacts with at most a constant number of clusters via P2P, the total number of operation counts to perform P2P can be written as

$$W_{\mathrm{P2P}}(N) = \mathrm{const} \cdot 2^\kappa \cdot N_0^2 \quad \text{for } N = N_t^\kappa . \tag{6.2}$$

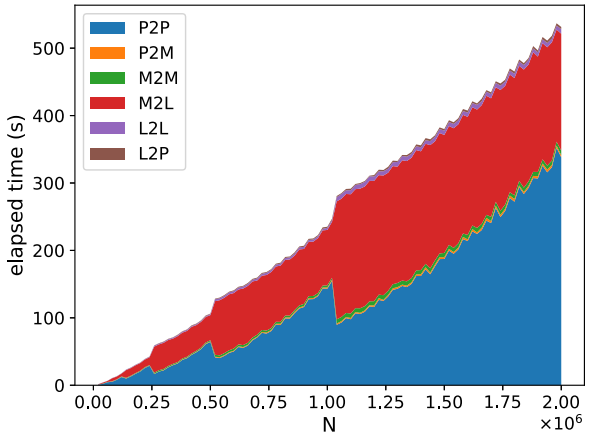


Fig. 4. Elapsed time of the six FMM kernels against the increasing number of particles N with interpolation degree $n = 4$, the maximum number of particles in the leaf cluster $N_0 = (n+1)^3$ and admissibility parameter $\eta = 0.5$.

When the number of particles N slightly increases with $\delta N \to 0$ so that $N > N_t^\kappa$, the number of particles in each leaf cluster N_f will slightly increase with $\delta N_\mathrm{f} \to 0$ so that $N_\mathrm{f} > N_0$. In this case, each leaf cluster will be subdivided into two clusters and the cluster tree will gain one more level $\kappa + 1$. Therefore, the number of leaf clusters will increase from 2^κ to $2^{\kappa+1}$ and the value of N_f reduces from N_0 to $N_0/2$. Thus, the total number of operation counts for P2P can be written as

$$W_{\mathrm{P2P}}(N) = \mathrm{const} \cdot 2^{\kappa+1} \cdot (\tfrac{N_0}{2} + \delta N_\mathrm{f})^2 \quad \text{for } N_t^{\kappa+1} \geq N > N_t^\kappa, \tag{6.3}$$

with $N_0/2 \geq \delta N_\mathrm{f}(N, N_0) > 0$. The ratio between (6.2) and (6.3) for different δN_f is

$$\frac{W_{\mathrm{P2P}}(N_t^\kappa + \delta N)}{W_{\mathrm{P2P}}(N_t^\kappa)} = \begin{cases} 1 & \text{for } \delta N_\mathrm{f} = 0, \\ \frac{1}{2} & \text{for } \delta N_\mathrm{f} \to 0, \\ 2 & \text{for } \delta N_\mathrm{f} = \frac{N_0}{2}. \end{cases}$$

Together with (6.3), we can see that W_{P2P} suddenly decreases to one half of $W_{\mathrm{P2P}}(N_t^\kappa)$ as N increases from $N = N_t^\kappa$ and then grows quadratically until it is two times bigger than $W_{\mathrm{P2P}}(N_t^\kappa)$ at $N = N_t^{\kappa+1}$. This performance model can describe the trend of elapsed time for P2P. Likewise, we can also apply a similar analysis to M2L and write down the corresponding performance model as

$$W_{\mathrm{M2L}}(N) = \begin{cases} \mathrm{const} \cdot 2^{\kappa+1} \cdot (n+1)^6 & N = N_t^{\kappa+1}, \\ \mathrm{const} \cdot 2^{\kappa+2} \cdot (n+1)^6 & N_t^{\kappa+1} \geq N > N_t^\kappa. \end{cases} \tag{6.4}$$

Here, we use the fact that a balanced cluster tree with the depth l contains 2^{l+1} total clusters and the assumption that each cluster interacts with at most a constant value of clusters through M2L. Equation (6.4) shows W_{M2L} suddenly increases to two times of $W_{\mathrm{M2L}}(N_t^\kappa)$ as N slightly increases with $\delta N \to 0$ from $N = N_t^\kappa$; and then it remains constant whenever $N_t^{\kappa+1} \geq N > N_t^\kappa$. This performance model can successfully explain the behavior of the elapsed time for M2L.

7. GPU parallelization

There have been several implementations of GPU-parallelized FMM based on the traditional scientific programming languages, like C/C++ or Fortran [28,27,29–33,17]. In this study, we use the Julia programming language [26] for the implementation. Compared to the traditional scientific programming languages, the Julia language has the advantages below:

- It was designed for high performance and provides high-level syntax suitable for scientific programming.
- Its programming paradigm of multiple dispatch makes it expressive and composable.
- It provides a built-in package manager.

Besides to reducing the development time, these features also make packages user-friendly and reusable.

As illustrated in Algorithm 5.11, our FMM is based on the dual tree traversal. The dual tree traversal could not be naively parallelized in data parallelism and might not benefit from GPUs. For one thing, the power of GPUs comes from executing multiple simple tasks through multiple threads in SIMD (single instruction, multiple data); for another thing, a single GPU core usually has weaker computing power than a single CPU core. Therefore, hybrid CPU-GPU approaches based on the creation of the interaction lists by CPU were investigated in some former works [17,34]. In this approach, the CPU first performs a dual tree traversal to generate interaction lists; and then, the GPU handles the interaction of each pair of clusters in the interaction lists. In this study, we refer to the work proposed by Wilson et al. [17,21] and discuss a GPU parallelization for our proposed FMM. The CPU-GPU execution of the proposed FMM can be summarized in the 10 steps listed in Algorithm 7.1. The H2D and D2H denote the data transfers of "host to device" and "device to host", respectively. As shown in Fig. 4, the execution of FMM spends most of the time on the interaction phases (P2P and M2L). Although the parallelization of each FMM kernel is implemented in our application, we will only focus on the implementations of P2P and M2L (*i.e.*, step 6 and step 8 in Algorithm 7.1) in later discussions.

Algorithm 7.1: The CPU-GPU Ecxecution of the Proposed Parallelized FMM.

1 CPU: generate particles information $\boldsymbol{x}_i$, $\boldsymbol{p}_i$ and allocate $\boldsymbol{E}_i$, $\boldsymbol{B}_i$
2 CPU: create cluster tree S with $\boldsymbol{x}_i$, $\boldsymbol{p}_i$
3 H2D: copy $\boldsymbol{x}_i$, $\boldsymbol{p}_i$ and S to device
4 GPU: allocate $\gamma_{S,v}$, $\boldsymbol{p}_{S,v}$, $\boldsymbol{E}_{S,v}$, $\boldsymbol{B}_{S,v}$, $\boldsymbol{E}_i$, $\boldsymbol{B}_i$
5 GPU: perform upwardpass with S, $\boldsymbol{x}_i$, $\boldsymbol{p}_i$ to compute $\gamma_{S,v}$, $\boldsymbol{p}_{S,v}$
6 CPU: perform dual tree traversal on S to build interaction lists (ITLs)
7 H2D: copy ITLs to device
8 GPU: perform P2P, M2L with ITLs to compute $\boldsymbol{E}_{S,v}$, $\boldsymbol{B}_{S,v}$
9 GPU: perform downwardpass with $\boldsymbol{E}_{S,v}$, $\boldsymbol{B}_{S,v}$ to compute $\boldsymbol{E}_i$, $\boldsymbol{B}_i$
10 D2H: copy $\boldsymbol{E}_i$, $\boldsymbol{B}_i$ to host

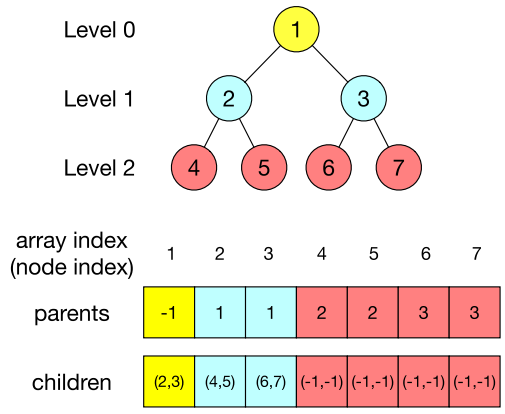


Fig. 5. An array-representation of a balanced binary tree. The node index is assigned with the breadth-first scheme.

8. Array-based tree data structure

In the implementation, it might be straightforward to express the cluster tree with a pointer-based data structure; that is, each node object (particle cluster in our case) contains data fields and a pointer, and this pointer is used to allocate the objects of children nodes. One major disadvantage of using a pointer-based tree is that the node objects are not stored in contiguous locations in the memory, and this makes the data transfer between host and device difficult. Hence, it might be beneficial to consider an array-based tree in the GPU application. Following the approach in Ref. [35], we use multiple arrays to store node objects with multiple members, one array for one member. A member of a node object with index i is located in the i-th element of the corresponding array. Besides, two additional arrays are respectively used to specify the parent index and children indices of nodes. Because our cluster tree is constructed through a k-d tree with cardinality-balanced subdivision of particles, it will be a balanced binary tree. Hence, we will narrow our following discussions to balanced binary tree.

Although there may exist several possibilities, we adapt the breadth-first scheme to assign the node index of a tree. With this index assignment scheme, the nodes in the level l have the indices $\{2^l, \ldots, 2^{(l+1)-1}\}$; and similarly, a node with the index i belongs to a level $\lfloor \log i / \log 2 \rfloor$. Here, we define that the level of cluster tree starts from 0 and the node index starts from 1. The breadth-first scheme can ensure that the member data of nodes from the same level stays contiguously in an array. This data arrangement is cache-friendly for both the upward pass (P2M and M2M) and the downward pass (L2L and L2P) where the whole member data of nodes from the same level will be accessed for the calculation. Therefore, the parent index and the children pair of indices for a node with index i are defined as

$$\text{iparent}(i) = \begin{cases} -1 & i = 1, \\ \left\lfloor \frac{i}{2} \right\rfloor & i \neq 1, \end{cases} \quad \text{and} \quad \text{ichildren}(i) = \begin{cases} (-1,-1) & \text{if leaf node}, \\ (2\cdot i, 2\cdot i+1) & \text{else}. \end{cases}$$

A schematic representation of our array-based tree is provided in Fig. 5. Since our tree is balanced (due to the cardinality-balanced subdivision scheme), we can preallocate a fixed-size array by knowing that the total number of nodes is $2^{(\kappa+1)} - 1$ with the tree-depth

$$\kappa = \begin{cases} 0 & N \leq N_0, \\ \lceil \log(N/N_0)/\log 2 \rceil & N > N_0. \end{cases}$$

It is worth noting that the resulting tree might not be balanced if other space-subdivision schemes are adapted. In this case, one may preallocate a big enough array that each node contains the children nodes of a maximum possible number. However, this causes large memory of unused nodes and leads to poor load-balancing across multiple ranks when MPI parallelization is considered [36]. One possible way to work around this issue is first creating a pointer-based tree, and an array-based tree can be allocated based on the information from that. This approach is adapted by some solvers, *e.g.*, BaryTree [37].

9. Parallelization of P2P and M2L kernels

Two lists of interaction pairs respectively for P2P and M2L, called interaction lists (ITLs), are generated by the execution of the dual tree traversal (Algorithm 9.1) in the CPU and copied into GPU. The GPU kernels respectively of P2P and M2L are launched in a way

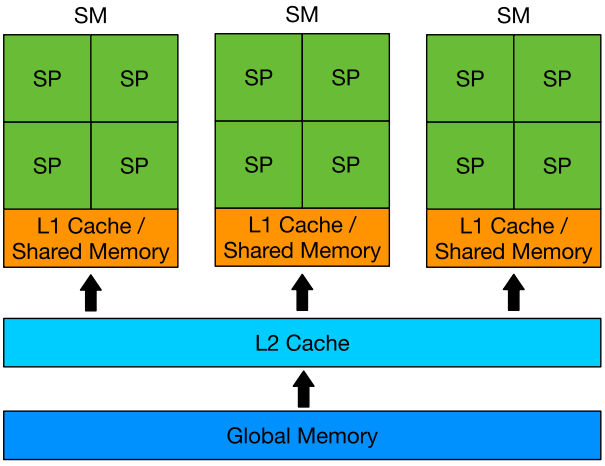
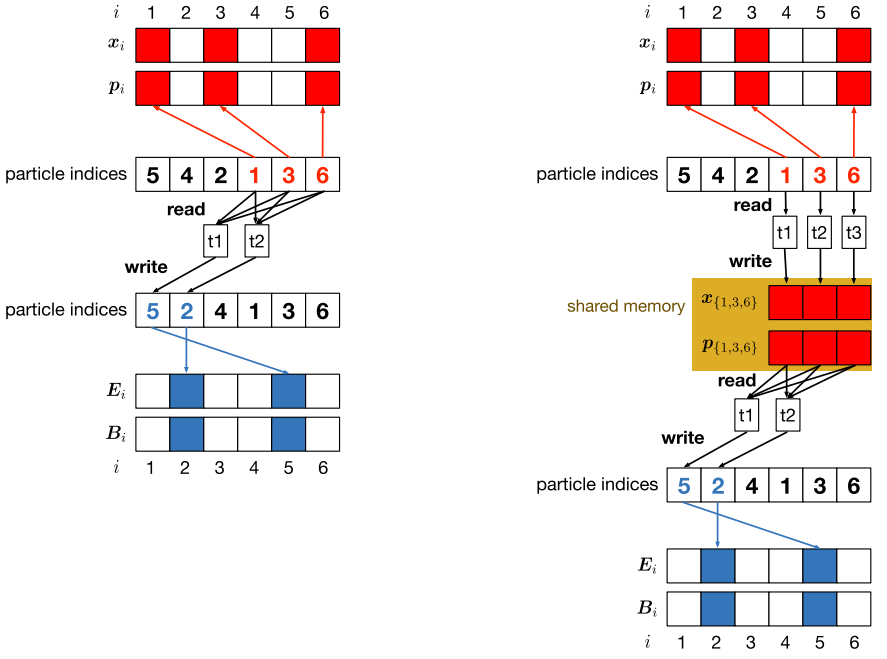


Fig. 6. Memory hierarchy of CUDA-capable GPUs. SP and SM denote streaming processor and streaming multiprocessor, respectively.



(a) P2P without shared memory (b) P2P with shared memory

Fig. 7. Implementations of P2P kernels (a) without shared memory and (b) with shared memory. The data associated with the target and source particle is colored in blue and red, respectively. The thread is denoted by a shorthand "t".

that each interaction pair is handled by a thread block. As P2P and M2L are both similar to a direct summation algorithm, their GPU implementations are straightforward; one thread in the threads block handles the evaluation of the force-field of one target micro/macro particle (P2P/M2L). The GPU parallelization of P2P is illustrated in Fig. 7. In our implementation, we use an additional array to store the indices of all particles in the system and the indices of particles from a cluster will always stay in a contiguous memory block in the array during the subdivision (cf. Appendix B). However, each member data of particles (*e.g.*, positions and momenta) from a cluster accessed through this particle-indices array does not necessarily stay contiguous in its array (Fig. 7a). Thus, a member data of source particles accessed by threads is non-contiguously distributed in an array. This can slow down the application because the data access is not cache-friendly and requires frequent access from the global memory. One way to remedy this problem is using the shared memory (Fig. 6) provided in CUDA-capable GPUs: we first load each member data of particles from a source cluster to shared memory so that the data can be accessed much faster by threads (Fig. 7b). For one thing, each member data of source particles stays in a contiguous block in the shared memory. For another, the shared memory is on-chip memory and has much lower latency than the global memory. For the M2L implementation, it is not necessary to apply the shared memory, because the member data of macro particles from a cluster is originally in a contiguous memory block. The data access is already cache-friendly so that the L1 cache in each streaming multiprocessor can be effectively used. A schematic of M2L implementation is provided in Fig. 8. Because the implementations of the other FMM kernels share large similarities with P2P or M2L, we will not go through the details of the implementations.

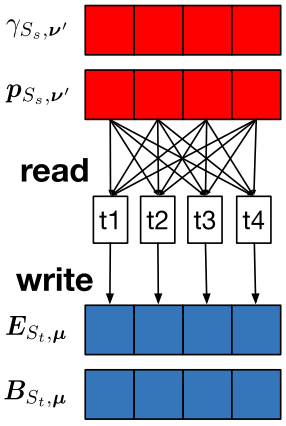


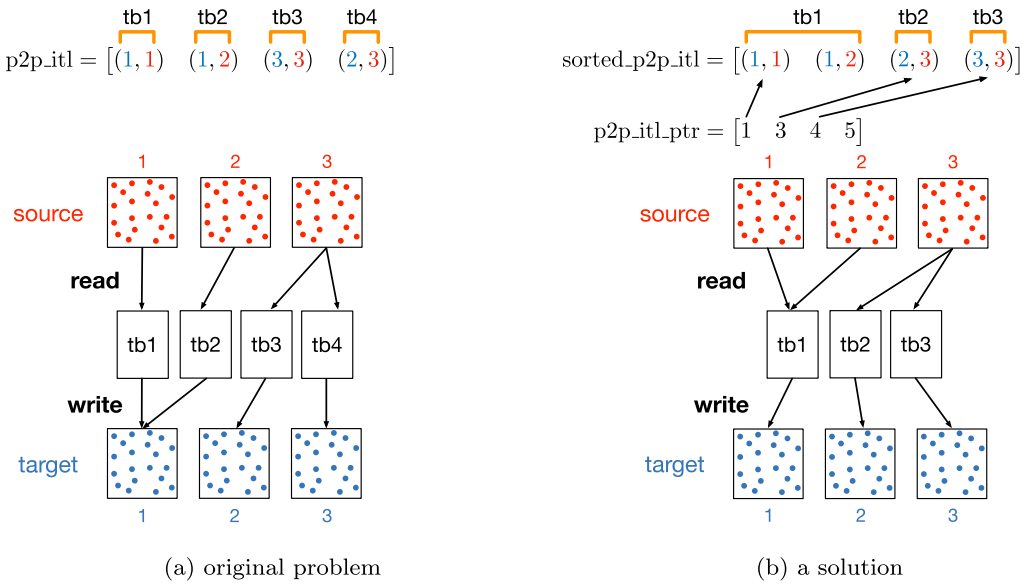
Fig. 8. An illustration of the implementation of the M2L kernel. The data associated with the macro particle of the target cluster and the source cluster are colored with blue and red, respectively. The thread is denoted by a shorthand "t".

Algorithm 9.1: Generation of Interaction Lists by Dual Tree Traversal.

S_i: particle cluster with node index i
$\boldsymbol{s}$: stretch factor
η: admissibility parameter
p2p_itl: initially empty list of interaction for P2P (global scope)
m2l_itl: initially empty list of interaction for M2L (global scope)
Function dualtraversefillitl(i, j, $\boldsymbol{s}$, η)
  if ichildren(i) == $(-1,-1)$ $\wedge$ ichildren(j) == $(-1,-1)$ **then**
    push (i, j) to p2p_itl
  else
    isAdmissible = admissible(S_i, S_j, $\boldsymbol{s}$, η) (Algorithm 5.8)
    if isAdmissible **then**
      push (i, j) to m2l_itl
    else if ichildren(i) == $(-1,-1)$ **then**
      for $k \in$ ichildren(j) **do**
        dualtraversefillitl($i, k, \boldsymbol{s}, \eta$)
      end
    else if ichildren(j) == $(-1,-1)$ **then**
      for $k \in$ ichildren(i) **do**
        dualtraversefillitl($k, j, \boldsymbol{s}, \eta$)
      end
    else
      if diam($S_i, \boldsymbol{s}$) > diam($S_j, \boldsymbol{s}$) **then**
        for $k \in$ ichildren(i) **do**
          dualtraversefillitl($k, j, \boldsymbol{s}, \eta$)
        end
      else
        for $k \in$ ichildren(j) **do**
          dualtraversefillitl($i, k, \boldsymbol{s}, \eta$)
        end
      end
    end
  end
end

10. Race conditions in P2P and M2L kernels

In CUDA applications, a GPU kernel can be launched with a grid of thread blocks and several thread blocks can be executed by streaming multiprocessors concurrently. In the execution of M2L or P2P, each pair of interaction is handled by one thread block and it is possible that several pairs of interaction with the same target index are handled by different thread blocks simultaneously. This can cause a race condition and produce an unexpected result because the corresponding memory data associated with a target cluster can be updated by the threads of different thread blocks at the same time (Fig. 9a). One common remedy for the race conditions is using CUDA's atomic operations [38], which locks a memory location so that only one exclusive thread is allowed to update the value each time. However, the atomic operations in CUDA only support some primitive types (*e.g.*, `Int32` and `Float32`) and cannot be used in our implementation, because each three-dimensional vector in the physical system (*e.g.*, position, momentum and vector field) is represented by a non-primitive and immutable type `SVector{3,T}` [39] with three elements of a parametric type `T`. Due to this immutability, we cannot apply atomic operations to change any elements of a `SVector{3,T}` object even though `T` is a primitive type (if so we can apply atomic operations to update each element of a `SVector{3,T}` object). Therefore, in our implementation, we divide pairs of interaction into groups such that each group only contains the pairs of interaction with the same target index; and during the kernel execution, each group will be handled by a thread block. To implement this, we first sort the pairs by the value of target index, which can be done efficiently with Quicksort. After that, we generate an additional array to indicate the start position of each group of pairs in the sorted ITL so that this array can be used to dispatch thread blocks to each group during the kernel execution (Fig. 9b).

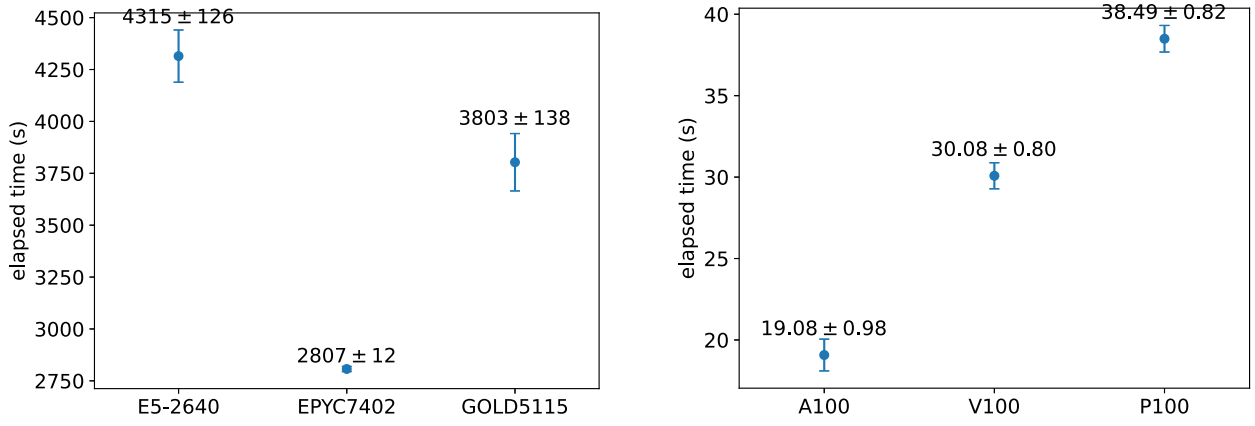


(a) original problem (b) a solution

Fig. 9. A schematic of the race condition problem in a naive parallelization of the P2P kernel (same for the M2L kernel) with ITL. Fig. 9a illustrates the original problem. Fig. 9b illustrates a solution by dividing the pairs of interaction into groups with the same target index. The thread block is denoted by a shorthand "tb".

Table 1
CPUs and GPUs used in the simulations for the performance benchmark.

CPU	GPU
INTEL XEON E5-2640V4	NVIDIA A100
AMD EPYC 7402	NVIDIA V100
INTEL XEON GOLD 5115	NVIDIA P100



(a) CPU (b) GPU

Fig. 10. Elapsed times of simulations with $N = 2.56 \times 10^7$ particles on different (a) CPUs and (b) GPUs. The simulation is performed with $\eta = 0.5$, $n = 4$ and $N_0 = (n+1)^3$. Each data point is the statistical result of 100 samples.

11. Performance

In this study, a package FMM4RBGPU.jl is written in the Julia programming language [26] with CUDA.jl [40,41]. This package provides CPU (serial) and GPU solvers for the FMM proposed in this study. The cluster tree used in this package is implemented with the array-based data structure discussed in Section 8.

To understand the performance of our GPU parallelization, we consider a simulation with $N = 2.56 \times 10^7$ particles on different CPUs and GPUs as listed in Table 1. The elapsed times of the simulations (*i.e.*, the execution of Algorithm 5.12) are demonstrated in Fig. 10. We can see that our GPU-based solver can achieve a speedup between 57 and 197 (relative to the result of a single CPU).

12. Summary

In this study, we propose an interpolation-based FMM for the computation of the relativistic space-charge field. With our proposed modified admissibility condition, the FMM can be directly evaluated in the lab-frame without the need for a Lorentz transformation. The

GPU parallelization and the pseudocode of the algorithms are also provided. In particular, we use the Julia programming language to implement the GPU-parallelized FMM. The proposed algorithms and package can be used to model the space-charge effect in the beam dynamics simulation of relativistic beams.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgement

This work was supported by DASHH (Data Science in Hamburg – HELMHOLTZ Graduate School for the Structure of Matter) with the Grant-No. HIDSS-0002 and in part by the European Research Council under the European Union Seventh Framework Programme (FP7/2007-2013) through Synergy Grant (609920). The authors acknowledge the computational resources of the Maxwell Cluster operated at Deutsches Elektronen-Synchrotron (DESY).

Appendix A. Definition of cumulative local field

Lemma 1. *Assume a target point $\boldsymbol{x}_i$ is contained in a sequence of clusters $\{S^l \mid l = 0, \dots, k\}$ of each level l with $S^{l+1} \subset S^l$ and $S^0 = S$. The total force-field of the macro particles from this sequence of clusters transferred to this target point can be calculated by*

$$f(\boldsymbol{x}_i) = \sum_{l=0}^{k} \sum_{\boldsymbol{\mu}} \ell_{S^l,\boldsymbol{\mu}}(\boldsymbol{x}_i) L_{S^l,\boldsymbol{\mu}} = \sum_{\boldsymbol{\mu}} \mathcal{L}_{S^k,\boldsymbol{\mu}} \ell_{S^k,\boldsymbol{\mu}}(\boldsymbol{x}_i),$$

where $\mathcal{L}_{S^l}$ is defined as

$$\mathcal{L}_{S^l,\boldsymbol{\mu}} := L_{S^l,\boldsymbol{\mu}} + \sum_{\boldsymbol{\mu}'} \mathcal{L}_{S^{l-1},\boldsymbol{\mu}'} \cdot \ell_{S^{l-1},\boldsymbol{\mu}'}(\boldsymbol{\xi}_{S^l,\boldsymbol{\mu}}) \quad \textit{with} \quad \mathcal{L}_{S^0,\boldsymbol{\mu}} := L_{S^0,\boldsymbol{\mu}}.$$

Proof. We prove this statement by mathematical induction. By the definition above, the statement automatically holds for the case $l = 0$. We only need to prove the statement holds for the case $l = k$ provided that it is true for $l = k - 1$. Thus, we have

$$\begin{aligned}
&\sum_{l=0}^{k} \sum_{\boldsymbol{\mu}} \ell_{S^l,\boldsymbol{\mu}}(\boldsymbol{x}_i) L_{S^l,\boldsymbol{\mu}} \\
=\ & \sum_{\boldsymbol{\mu}} \ell_{S^k,\boldsymbol{\mu}}(\boldsymbol{x}_i) L_{S^k,\boldsymbol{\mu}} + \sum_{l=0}^{k-1} \sum_{\boldsymbol{\mu}} \ell_{S^l,\boldsymbol{\mu}}(\boldsymbol{x}_i) L_{S^l,\boldsymbol{\mu}} \\
=\ & \sum_{\boldsymbol{\mu}} \ell_{S^k,\boldsymbol{\mu}}(\boldsymbol{x}_i) L_{S^k,\boldsymbol{\mu}} + \sum_{\boldsymbol{\mu}'} \mathcal{L}_{S^{k-1},\boldsymbol{\mu}'} \cdot \ell_{S^{k-1},\boldsymbol{\mu}'}(\boldsymbol{x}_i) \quad \text{(by assumption)} \\
\overset{(2.8)}{=}\ & \sum_{\boldsymbol{\mu}} \ell_{S^k,\boldsymbol{\mu}}(\boldsymbol{x}_i) L_{S^k,\boldsymbol{\mu}} + \sum_{\boldsymbol{\mu}'} \mathcal{L}_{S^{k-1},\boldsymbol{\mu}'} \sum_{\boldsymbol{\mu}} \ell_{S^{k-1},\boldsymbol{\mu}'}(\boldsymbol{\xi}_{S^k,\boldsymbol{\mu}}) \ell_{S^k,\boldsymbol{\mu}}(\boldsymbol{x}_i) \\
=\ & \sum_{\boldsymbol{\mu}} \underbrace{\left(L_{S^k,\boldsymbol{\mu}} + \sum_{\boldsymbol{\mu}'} \mathcal{L}_{S^{k-1},\boldsymbol{\mu}'} \cdot \ell_{S^{k-1},\boldsymbol{\mu}'}(\boldsymbol{\xi}_{S^k,\boldsymbol{\mu}}) \right)}_{=\mathcal{L}_{S^k,\boldsymbol{\mu}}} \ell_{S^k,\boldsymbol{\mu}}(\boldsymbol{x}_i). \quad \square
\end{aligned}$$

Appendix B. Data structure of the cluster tree

Although the FMM solvers developed in this work are written in the Julia programming language, we use C-style pseudocode to illustrate the data structure of the cluster tree. The data structure of the cluster tree can be naively designed as follows:

```
struct Cluster {
  size_t npar;
  value_type (*positions)[3]; // array of particle positions
  Cluster* children;
}
```

However, this naive implementation may require a significant amount of memory as the position of particles in each cluster is explicitly stored. For a balanced cluster tree describing an N-particles cluster, the number of particle positions to be stored is $N \log_2 N$. If we have $N = 2 \times 10^6$, a memory of roughly 1 GB will need to be allocated during the construction of the cluster tree and this could cause a performance bottleneck.

Alternatively, one may store the particle positions outside the structure and declare an external array `parindices` to store the indices of all the particles. In such a case, the data structure can be expressed as

```
size_t parindices[N]
value_type positions[N][3]
struct Cluster {
  size_t pindex_lo;
  size_t pindex_hi;
  Cluster* children;
}
```

If the elements of `parindices` are arranged in such a manner that the indices of the particles in the cluster S occupy in `parindices` contiguously from l-th (`pindex_lo`) to h-th (`pindex_hi`) location, their values in `parindices` (*i.e.*, their indices) can be expressed as

$$p_l, p_{l+1}, \ldots, p_h.$$

In the subdivision of S, we first determine the splitting coordinate direction $g \in \{x, y, z\}$ from $\mathrm{bbox}(S)$ and permute the elements in `parindices` that

$$p'_l, \ldots, p'_{\lfloor \frac{l+h}{2} \rfloor}, p'_{\lfloor \frac{l+h}{2} \rfloor + 1}, \ldots p'_h$$

and

$$g_{p'_i} \begin{cases} \leq g_{p'_{\lfloor \frac{l+h}{2} \rfloor}} & \text{if } \left\lfloor \frac{l+h}{2} \right\rfloor \geq i \geq l, \\ > g_{p'_{\lfloor \frac{l+h}{2} \rfloor}} & \text{if } \left\lfloor \frac{l+h}{2} \right\rfloor < i \leq h. \end{cases}$$

This permutation enables the objects of the children clusters `S1` and `S2` to access their belonging particle indices by:

$$\texttt{S1.pindex\_lo}=l,\ \texttt{S1.pindex\_hi}=\lfloor \tfrac{l+h}{2} \rfloor,$$
$$\texttt{S2.pindex\_lo}=\lfloor \tfrac{l+h}{2} \rfloor + 1,\ \texttt{S2.pindex\_hi}=h,$$
$$\texttt{parindicies[S1.pindex\_lo]},\ldots,\texttt{parindicies[S1.pindex\_hi]},$$
$$\texttt{parindicies[S2.pindex\_lo]},\ldots,\texttt{parindicies[S2.pindex\_hi]}.$$

In this study, the permutation is implemented by the Quickselect algorithm with the Lomuto partition scheme [42]. The complexity on average is $\mathcal{O}(N)$ and can be $\mathcal{O}(N^2)$ in the worst-case scenario.

References

[1] H. Zhang, J. Portman, Z. Tao, P. Duxbury, C.-Y. Ruan, K. Makino, M. Berz, Microsc. Microanal. 21 (S4) (2015) 224–229, https://doi.org/10.1017/S1431927615013410.
[2] H. Zhang, H. Huang, R. Li, J. Chen, L.-S. Luo, AIP Conf. Proc. 1812 (1) (2017) 050001, https://doi.org/10.1063/1.4975862.
[3] M. Langston, R. Lethin, P. Letourneau, M. Morse, J. Wei, in: Proceedings of the 12th International Particle Accelerator Conference, No. 12 in International Particle Accelerator Conference, JACoW Publishing, Geneva, Switzerland, 2021, pp. 4237–4240.
[4] M. Gordon, S. Van Der Geer, J. Maxson, Y.-K. Kim, Phys. Rev. Accel. Beams 24 (8) (2021) 084202, https://doi.org/10.1103/PhysRevAccelBeams.24.084202.
[5] S. Schmid, H. De Gersem, M. Dohlus, E. Gjonaj, in: Proceedings of 3rd North American Particle Accelerator Conference, WEPLE10, 2019.
[6] J.M. Dawson, Rev. Mod. Phys. 55 (1983) 403–447, https://doi.org/10.1103/RevModPhys.55.403.
[7] C.K. Birdsall, A.B. Langdon, Plasma Physics via Computer Simulation, CRC Press, 2018.
[8] K. Flöttmann, S. Lidia, P. Piot, Recent improvements to the ASTRA particle tracking code, Tech. rep., Lawrence Berkeley National Lab (LBNL), USA, 2003.
[9] J. Qiang, S. Lidia, R.D. Ryne, C. Limborg-Deprey, Phys. Rev. Spec. Top., Accel. Beams 9 (2006) 044204, https://doi.org/10.1103/PhysRevSTAB.9.044204.
[10] J. Qiang, Phys. Rev. Accel. Beams 21 (2018) 054201, https://doi.org/10.1103/PhysRevAccelBeams.21.054201.
[11] F.W. Jones, AIP Conf. Proc. 448 (1) (1998) 359–370, https://doi.org/10.1063/1.56759.
[12] S.A. Schmid, H.D. Gersem, E. Gjonaj, REPTIL - a Relativistic 3D Space Charge Particle Tracking Code Based on the Fast Multipole Method, 2019, unpublished.
[13] J. Qiang, Phys. Rev. Accel. Beams 20 (2017) 014203, https://doi.org/10.1103/PhysRevAccelBeams.20.014203.
[14] S. Schmid, H.D. Gersem, E. Gjonaj, in: Proceedings of the 12th International Particle Accelerator Conference, No. 12 in International Particle Accelerator Conference, JACoW Publishing, Geneva, Switzerland, 2021, pp. 4244–4246.
[15] G. Fubiani, J. Qiang, E. Esarey, W.P. Leemans, G. Dugan, Phys. Rev. Spec. Top., Accel. Beams 9 (2006) 064402, https://doi.org/10.1103/PhysRevSTAB.9.064402.
[16] Y.-K. Kan, F.X. Kärtner, S. Le Borne, J.-P.M. Zemke, Comput. Phys. Commun. 286 (2023) 108668, https://doi.org/10.1016/j.cpc.2023.108668.
[17] L. Wilson, N. Vaughn, R. Krasny, Comput. Phys. Commun. 265 (2021) 108017, https://doi.org/10.1016/j.cpc.2021.108017.
[18] S. Wienke, P. Springer, C. Terboven, D. an Mey, in: C. Kaklamanis, T. Papatheodorou, P.G. Spirakis (Eds.), Euro-Par 2012 Parallel Processing, Springer Berlin Heidelberg, Berlin, Heidelberg, 2012, pp. 859–870.
[19] W. Fong, E. Darve, J. Comput. Phys. 228 (23) (2009) 8712–8725, https://doi.org/10.1016/j.jcp.2009.08.031.
[20] L. Wang, R. Krasny, S. Tlupova, Commun. Comput. Phys. 28 (4) (2020) 1415–1436, https://doi.org/10.4208/cicp.OA-2019-0177.
[21] L. Wilson, Development and Application of Numerical Methods in Biomolecular Solvation, Ph.D. thesis, University of Michigan, 2021.
[22] S. Börm, Efficient Numerical Methods for Non-local Operators: H^2-Matrix Compression, Algorithms and Analysis, EMS Tracts in Mathematics, vol. 14, European Mathematical Society (EMS), Zürich, 2010.

[23] W. Hackbusch, Hierarchical Matrices: Algorithms and Analysis, vol. 49, Springer, 2015.
[24] A.W. Appel, SIAM J. Sci. Stat. Comput. 6 (1) (1985) 85–103, https://doi.org/10.1137/0906008.
[25] W. Dehnen, J. Comput. Phys. 179 (1) (2002) 27–42, https://doi.org/10.1006/jcph.2002.7026.
[26] J. Bezanson, A. Edelman, S. Karpinski, V.B. Shah, SIAM Rev. 59 (1) (2017) 65–98, https://doi.org/10.1137/141000671.
[27] R. Yokota, T. Narumi, R. Sakamaki, S. Kameoka, S. Obi, K. Yasuoka, Comput. Phys. Commun. 180 (11) (2009) 2066–2078, https://doi.org/10.1016/j.cpc.2009.06.009.
[28] I. Lashuk, A. Chandramowlishwaran, H. Langston, T.-A. Nguyen, R. Sampath, A. Shringarpure, R. Vuduc, L. Ying, D. Zorin, G. Biros, in: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis, 2009, pp. 1–12.
[29] R. Yokota, J.P. Bardhan, M.G. Knepley, L. Barba, T. Hamada, Comput. Phys. Commun. 182 (6) (2011) 1272–1283, https://doi.org/10.1016/j.cpc.2011.02.013.
[30] R. Yokota, L. Barba, T. Narumi, K. Yasuoka, Comput. Phys. Commun. 184 (3) (2013) 445–455, https://doi.org/10.1016/j.cpc.2012.09.011.
[31] Y. Wang, Q. Wang, X. Deng, Z. Xia, J. Yan, H. Xu, Adv. Eng. Softw. 82 (2015) 105–118, https://doi.org/10.1016/j.advengsoft.2015.01.002.
[32] R. Adelman, N.A. Gumerov, R. Duraiswami, IEEE Trans. Magn. 53 (12) (2017) 1–11, https://doi.org/10.1109/TMAG.2017.2725951.
[33] T. Takahashi, C. Cecka, W. Fong, E. Darve, Int. J. Numer. Methods Eng. 89 (1) (2012) 105–133, https://doi.org/10.1002/nme.3240.
[34] J. Liu, M. Robson, T. Quinn, M. Kulkarni, in: Proceedings of the ACM International Conference on Supercomputing, ICS '19, Association for Computing Machinery, New York, NY, USA, 2019, pp. 24–34.
[35] M. Burtscher, K. Pingali, in: W. Mei, W. Hwu (Eds.), GPU Computing Gems Emerald Edition, in: Applications of GPU Computing Series, Morgan Kaufmann, Boston, 2011, pp. 75–92.
[36] Leighton Wilson, 2022, private communication.
[37] L. Wilson, N. Vaughn, BaryTree, https://github.com/Treecodes/BaryTree, 2021.
[38] J. Cheng, M. Grossman, T. McKercher, Professional CUDA C Programming, John Wiley & Sons, 2014.
[39] A. Ferris, et al., StaticArrays, https://github.com/JuliaArrays/StaticArrays.jl, 2016.
[40] Julia Computing, et al., CUDA.jl, https://github.com/JuliaArrays/StaticArrays.jl, 2016.
[41] T. Besard, C. Foket, B.D. Sutter, IEEE Trans. Parallel Distrib. Syst. 30 (4) (2019) 827–841, https://doi.org/10.1109/tpds.2018.2872064.
[42] T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein, Introduction to Algorithms, 4th edition, MIT Press, 2022.