Efficient Algorithms for the Fast Computation of Space Charge Effects Caused by Charged Particles in Particle Accelerators

Dissertation
zur
Erlangung des akademischen Grades
Doktor-Ingenieur (Dr.-Ing.)
der Fakultät für Informatik und Elektrotechnik
der Universität Rostock

vorgelegt von

Dawei Zheng, geboren am 25.09.1985 in Jilin, VR China
aus Rostock

Rostock, 02.09.2016

urn:nbn:de:gbv:28-diss2018-0013-9

Tag der Einreichung: 02.09.2016
Tag der Verteidigung: 16.12.2016

Gutachter

Prof. Dr. Ursula van Rienen, Universität Rostock
Dr. Ji Qiang, Lawrence Berkeley National Laboratory
Prof. Dr. Daniel Potts, TU Chemnitz

By pursuing his own interest he frequently promotes that of the society more effectually than when he really intends to promote it.

——An Inquiry into the Nature and Causes of the Wealth of Nations

Adam Smith

Abstract

The calculation of the solution to Poisson's equation regarding the space charge force is still a challenging task in beam dynamics simulations and therefore further improvement is necessary. The partial differential equations lead to different numerical methods and treatments. Efficiency and accuracy are the two key aspects in the numerical computation. In this dissertation, we firstly summarize a type of classical fast Poisson solver, Green's function-Fast Fourier transform routine in beam dynamics simulations. This includes the Hockney and Eastwood's convolution routine, the integrated Green's function method, etc. These solvers have been used for decades in the research community. However, the efficiency of this commonly used Poisson solver can still be further improved. The improvement for the commonly used Poisson solver is separated into three parts: the calculation of Green's function integral values is replaced by the efficient integrated Green's function integral values; the discrete Fourier transform calculation of the real even symmetric extension of the integrated Green's function values is replaced by the discrete cosine transform of the efficient integrated Green's function values; the explicitly zero-padded fast Fourier transform is replaced by the implicitly zero-padded fast Fourier transform for charge density. In addition, the state-of-the-art high performance computing technology is utilized for the further improvement of efficiency. These technologies include: OpenMP API for multi-thread CPU parallelization; OpenMP+CUDA for CPU and GPU heterogeneous parallelization; MPI for parallelization in supercomputers; MPI+OpenMP for parallelization in advanced supercomputers. The examples demonstrate that the resulting improvement regarding the efficiency is significant. The successful routine is programmed and integrated into the simulation packages MOEVE-PIC for the implementation in workstations and IMPACT for the implementation in supercomputers, respectively. The simulation results are matched with the results of the commonly used Poisson solver in order to demonstrate the accuracy performance. In total, the new Poisson solver routine preserves the advantages of fast computation and high accuracy. The efficiency for both sides of algorithm and technology is promising and attractive. In conclusion, this novel Poisson solver provides a fast routine for high performance calculation of the space charge effect in the simulation of beam dynamics in accelerators.

Zusammenfassung

Das Berechnen der Lösungen für die Poisson-Gleichung bezogen auf die Raumladung stellt immer noch eine große Herausforderung für dynamische Simulationen dar und erfordert weiterhin stetige Verbesserungen. Die partiellen Differentialgleichungen führen zu verschiedenen numerischen Methoden und Verfahren. In numerischen Berechnungen sind die Leistungsfähigkeit und Genauigkeit die beiden wesentlichen Aspekte. Diese Dissertation beschäftigt sich mit den klassischen schnellen Lösungensverfahren für die Poisson-Gleichnung und der Greenschen Schnellen Fourier-Transformation in Simulationen der Strahldynamik. Diese bezieht unter anderem die Konvolutionsroutine von Hockney und Eastwood und die integrierte Greensche Funktionsmethode mit den Lösungen, die seit Jahrzehnten in der Forschung benutzt werden. Für die derzeit angewendete Lösungensverfahren für die Poisson-Gleichnung besteht immer noch Entwicklungsspielraum hinsichtlich der Leistungsfähigkeit. Die Verbesserung der Lösung ist in drei Schritte unterteilt. Die Berechnung der Greenschen Funktionsintegralwerte wird durch die effizienteren Greenschen Funktionsintegralwerte ersetzt. Die diskrete Berechnung der Fourier-Transformation der reellen symmetrischen Erweiterung der integrierten Greenschen Funktionswerte wird durch die diskrete Kosinustransformation der effizienten integrierten Greenschen Funktionswerte ersetzt. Die explizite schnelle Faltung der Fourier-Transformation wird durch die implizite schnelle Faltung der Fourier-Transformation für die die Ladungsdichte ersetzt. Zudem kommt für die weitere Verbesserung der Leistungsfähigkeit die modernste Hochleistungscomputertechnologie zum Einsatz. Die Technologie besteht aus einer OpenMP API für mehrsträngige CPU Parallelisierung; OpenMP+CUDA für die heterogene Parallelisierung von CPU und GPU; MPI für die Parallelisierung in Hochleistungsrechnern; MPI+OpenMP für die Parallelisierung in weiterentwickelten Großrechnern. Die Beispiele belegen, dass die resultierende Verbesserung der Leistungsfähigkeit groß ist. Die erfolgreiche Routine wird entsprechend programmiert und in die Simulationspakete MOEVE-PIC für den Einsatz in Arbeitsplatzrechnern und in IMPACT für den Einsatz in Hochleistungsrechnern integriert. Die Simulationsergebnisse werden mit den Ergebnissen der allgemein üblichen Lösungsverfahren für die Poisson-Gleichung abgeglichen, um die Leistungsfähigkeit im Detail darzustellen. Insgesamt werden in den neuen Routinen für die Lösung der Poisson-Gleichung die Vorteile der schnellen Verarbeitung und der hohen Genauigkeit beibehalten. Die Leistungsfähigkeit für den Algorithmus als auch die Technologie ist vielversprechend. Zusammenfassend lässt sich sagen, dass die neuen Lösungsansätze eine schnelle Routine für präzise Berechnungen des Raumladungseffektes in Strahldynamiksimulationen für Beschleuniger darstellt.

Danksagungen

Diese Arbeit wurde von der “The China Scholarship Council” (CSC) unterstützt.

Ich kann ganz gewiss sagen, dass meine Promotionszeit in Deutschland die großartigste Zeit in meinem Leben bisher ist. Ich habe so viel gelernt, darunter ist das wichtigste Selbstständigkeit; nicht nur die Selbstständigkeit im wissenschaftlichen Arbeiten, sondern auch die Selbstständigkeit im Leben. Die Ideologie, die Denkart und die Sozialordnung in Deutschland haben mich außerdem stark beeinflusst. Für die so viele Erlebnisse, Leute und Dinge in den vergangenen Jahre möchte ich meinen Dank und Erkenntlichkeit ausdrücken.

Vom heutigen Standpunkt aus freue ich mich sehr, dass ich mein Promotionsprojekt fertig gestellt habe, obwohl es ein völlig neues Thema für mich war, als ich vor fünf Jahren damit begonnen habe und ich zunächst viel Angst vor diesem neuen Thema hatte. Ich hätte diesen Erfolg nicht ohne die Hilfe der folgenden Menschen erreichen können,

auf der Arbeitsseite:

- Prof. Dr. Ursula van Rienen als meine Doktormutter für die viele Unterstützung und die Chance, die sie mir gegeben hat.
- Dr. Gisela Pöplau für die Unterstützung in der ersten Phase der Promotionszeit.
- Dr. Ji Qiang für die Unterstützung meines Aufenthaltes am LBNL.
- Prof. Dr. Daniel Potts für die Übernahme der externen Gutachter.
- meine promovierten (Ex)Kollegen aus der Arbeitsgruppe: Dr. Dirk Hecht, Dr. Thomas Flisgen, Dr. Tomasz Galek, Dr. Revathi Appali, Dr. Christian Bahls, Dr. Aleksandar Markovik, Dr. Christian Schmidt.
- meine (Ex)Kollegen aus der Arbeitsgruppe: Johann Heller, Franziska Reimann, Shahnam Gorgi Zadeh, Korinna Brackebusch, Kai Papke, Ulf Zimmermann, Kiran Kumar Sriperumbudur, Mirjana Holst, Emanuele Brentegani, Prasanth Babu Ganta, Ahmed Masood.
- Henrik Bönner, Ulrike Heeder, Craig Meulen.
- Dr. Kwok Ko, Dr. Stephan Frank, Dr. Mikhail Krasilnikov, Dr. Ming Zhou, Dr. Ji Li, Frederik Kesting, Renhao Xie, Huang Tien Tran, Mahesh Dhone.
- Kathrin Krebs, Maja Gudat, Dr. Bernhard Himmel, Petra Gefken.

auf der privaten Seite:

- Jan Füsting und seiner Familie.

- Henning Stubbe und Clara Waldmann.
- Tao Xie und seiner Familie.
- Friederike Kunz und ihre Familie.
- Gisela Pöplau und ihre Familie.
- Freunde: Jie Chen, Ji Shen, Yujie Quan, Xiaoqi Ni, Lipeng Wu, Chengliang Lin, Nan Li, Xing Wang, Chen Su, Fengxue Qi, Jianyin Mai, Renhao Xie, Tom Niedzwiedz, Amy Meehan, Dan Poppa.

Einen großen Dank an meine Mutter.

Thesis Statements

of the dissertation

Efficient Algorithms for the Fast Computation of Space Charge Effects Caused by Charged Particles in Particle Accelerators

by Dawei Zheng

1. The multi-particle system inside the particle accelerator can be described with the help of Vlasov-Maxwell equations. The dynamics of the particle distribution leads to the hard mathematical problem —— the N-body problem of identical particles.

2. The particle-in-cell model, a particle-mesh method that treats the particles by approximating the quantity on a mesh, is chosen as the computational method for beam dynamics simulation.

3. The calculation of space charge effects is influenced by the beam itself, surrounding conditions, and time evolution. There are various numerical methods and the corresponding efficient optimization is demanded.

4. The Poisson solvers in MOEVE-PIC software package are further enhanced for GF-FFT methods.

5. Green's function (GF) and the integrated Green's function (IGF) methods are optimized by efficient IGF methods, i.e. CIGF, RIGF, and CRIGF. The efficiency is achieved while the accuracy does not decline.

6. The CIGF fits far-bunch space charge calculations and the RIGF matches near-bunch space charge calculations.

7. The trivial FFT convolution routine is easily implemented as Hockney and Eastwood created their own successful efficient routine for computing.

8. A novel discrete convolution with implicitly zero-padded FFT is investigated for the improvement of the convolution routine.

9. The calculation of the IGF values is replaced by the efficient IGF integral values.

10. The discrete Fourier transform calculation of the real even symmetric extension of the IGF values is replaced by the discrete cosine transform of the efficient IGF values.

11. The explicitly zero-padded fast Fourier transform is replaced by the implicitly zero-padded fast Fourier transform for charge density.

12. The real to complex FFT implementation can further save nearly half of the time consumption of the novel fast 3D convolution.

13. Among others, major errors resulting from the PIC model are the density fluctuation, the coupling between the macro particles and the mesh. Regarding the GF-FFT method, numerical errors are induced by numerical integration.

14. The OpenMP routine of shared-memory parallelization of the novel Poisson solver is programmed and examined by increasing CPU thread numbers.

15. A heterogeneous parallelization of CPU+GPU relying on an OpenMP+CUDA API implementation for workstations is provided. The limitations of CUDA API are the reasons for this heterogeneous routine rather than a pure GPU parallel routine.

16. The corresponding parallel Poisson solver for supercomputers is further programmed in Fortran and integrated into IMPACT software package at LBNL. An MPI+OpenMP parallel routine for future advanced supercomputers is attempted as well.

17. Efficiency studies as weak scaling, strong scaling, assignment of processors, different compilers, and FFT libraries are given for the implementation on a supercomputer.

18. An ideal uniform charged sphere bunch, an ideal uniform charged ellipsoid bunch, and an ideal Gaussian distributed charged bunch are introduced for the numerical verification, validation and convergence studies.

19. For applications, the validations are done by comparing the novel Poisson solver with a commonly used Poisson solver. Two beam dynamic simulations through two different simulation codes, MOEVE-PIC and IMPACT, are given.

20. The improvements in efficiency regarding both the algorithm and the HPC technologies for the implementations of both workstation and supercomputer are significant.

List of Acronyms

API application program interface

ASTRA A Space Charge Tracking Algorithm

BBGKY Bogolyubov-Born-Green-Kirkwood-Yvon

Bi-CGStab bi-conjugate-gradient stabilized

CG conjugate-gradient

CIC Cloud In Cell

CIGF cutting integrated Green's function

CPU Central Processing Unit

CPS commonly used Poisson solver

CRIGF cutting reduced integrated Green's function

CUDA Compute Unified Device Architecture

cuFFT CUDA Fast Fourier Transform (FFT) product

DCT (type-I) discrete cosine transform

DCT-II type-II discrete cosine transform

DESY Deutsches Elektronen-Synchrotron

DFT discrete Fourier transform

DST (type-I) discrete sine transform

DST-II type-II discrete sine transform

EPS novel efficient Poisson solver

E.M. electromagnetic

ESS European Spallation source

FDM finite difference method

FEL free-electron laser

FFT fast Fourier transform

FFTW Fastest Fourier Transform in the West

GF Green's function

GPU Graphical Processing Unit

HPC high performance computing

HandE Hockney and Eastwood

IDCT (type-I) inverse discrete cosine transform

IDFT inverse discrete Fourier transform

IGF integrated Green's function

ILC International Linear Collider

IMPACT Integrated Map and Particle ACcelerator Tracking Code

KNL Knights Landing

LBNL Lawrence Berkeley National Laboratory

LHC Large Hadron Collider

MIC Many Integrated Core

MOEVE Multigrid for non-equidistant grids to solve Poisson's equation

MPI Message Passing Interface

NERSC National Energy Research Scientific Computing Center

NGP Nearest Grid Point

NUMA Non-Uniform Memory Access

OPAL Object Oriented Parallel Accelerator Library

OpenMP Open Multi-Processing

OpenCL Open Computing Language

PDEs partial differential equations

PIC Particle-in-Cell

PIConGPU A Many-GPGPU Particle-in-Cell Code

PITZ Photo Injector Test Facility at DESY, Location Zeuthen

RAM random-access memory

RESE real even symmetric extension

RF radio-frequency

RIGF reduced integrated Green's function

r.h.s right-hand side

rms root mean square

SIMD single instruction multiple data

SMP symmetric multiprocessing

SOR successive over-relaxation

TLP thread-level parallelism

TSC Triangular Shaped Cloud

XFEL X-ray free-electron laser

Contents

List of Figures

List of Tables

1 Introduction

Particle accelerators have a history of almost 100 years. The breakthrough started with Rutherford's scattering experiment involving alpha particles on a gold foil [78] [79]. Originally the alpha particles came from natural radioactive sources. Gamow predicted that perhaps a source of lower energy particles than the natural radioactive sources would be sufficient. For the physical studies, researchers showed interest in the manufactured particle sources, and the first accelerators were built around the 1930s. The practical machines were the Van de Graff generator [85] [86], the Cockcroft-Walton generator [20], and the first cyclotron by Lawrence [51] [50]. Some theoretical input was provided by Ising [47], Wideröe [94] [95] and Alvarez. Since then, further concepts and results have appeared, such as colliding beams, synchronous acceleration with phase stability, among others. Accelerator physics studies have been an essential ingredient to overcome the practical limits of the modern accelerators, e.g. beam dynamics, beam cooling, collective beam instability, space charge effects. In the meantime, more acceleration technologies have been exploited, e.g. the radio-frequency (RF) power sources, high acceleration gradients, wakefield acceleration. Today's accelerators are pursuing higher energy, higher luminosity, and higher brightness beams. Accelerators for high energy physics are operating or built, such as the Large Hadron Collider (LHC) [21], a ring-like machine at CERN in Switzerland or the International Linear Collider (ILC) [84], a linear accelerator scheduled in Japan. In applications, other types of accelerators are more popular; synchrotron light sources and spallation sources are used for material studies. New generations of brilliant light sources and spallation sources have been designed and are under construction now, such as the European X-ray free-electron laser (XFEL) [98] at Deutsches Elektronen-Synchrotron (DESY) in Hamburg[1] and the European Spallation source (ESS) [29] in Lund, Sweden. The types of today's accelerators are diverse. Further accelerators are used for different purposes, such as cancer therapy and rocket motors.

This dissertation starts with the introduction of the basic beam dynamics from physical and mathematical viewpoints. This involves Maxwell's equations, Hamiltonian dynamics, and Einstein's special theory of relativity, which are introduced at the beginning. In particular, the space charge effects as the interesting field of this dissertation are explained with respect to different numerical methods.

Chapter 3 then discusses the basic Green's function (GF) method with free-space boundary conditions. The electrostatics potential φ is obtained as the convolution of the charge density and GF within a computational domain. The mid-point integral rule is chosen for the numerical convolution calculation. Based on the primitive func-

[1] This machine is partly built in Schleswig-Holstein.

tion of GF, the integrated Green's function (IGF) provides a further approach for the discrete convolution. However, the time consumption of the numerical convolution is conspicuous. A couple of efficient GFs are introduced.

Replacing the trivial fast Fourier transform (FFT) convolution routine, Chapter 4 reviews the multidimensional discrete Fourier transform (DFT) and the commonly used classical method by Hockney and Eastwood. Furthermore, a novel efficient convolution routine is presented for 1D, 2D and 3D situations. Vector extensions and DFTs are both considered for the optimization of the GF-related calculations. Furthermore, the implicitly zero-padded FFTs for the charge density are applied. The combination of real to complex FFTs with the novel discrete convolution is developed as well. Finally, an error study is carried out in Section 4.5.

Chapter 5 approaches various parallel routines. First, the OpenMP application program interface (API) speeds up the computation relating to the number of threads in usage for shared memory parallel routine. Second, an effort on CPU+GPU heterogeneous parallelization is carried out by means of an OpenMP+Compute Unified Device Architecture (CUDA) routine. Third, the Message Passing Interface (MPI) routine for the distributed shared memory parallelization in supercomputers for high performance computing (HPC) implementation is designed. For state-of-the-art supercomputers, the MPI+OpenMP parallel routine is further developed.

Chapter 6 verifies the aforementioned novel efficient methods with examples and applications. First, three examples are introduced for the numerical verification. Second, the efficient IGF integrals, cutting integrated Green's function (CIGF) and reduced integrated Green's function (RIGF) are verified through these test examples. Additionally, the novel fast convolution routine combined with the efficient IGF integral is studied to determine its accuracy. For applications, the validations are done by comparing the novel Poisson solver and a commonly used Poisson solver. Both serial and parallel routines are considered. The parallel routines are studied with different architectures.

2 Fundamentals of the mathematical model and space charge effects

A beam, composed of bunches of charged particles, is accelerated and manipulated by electromagnetic (E.M.) fields, which are generated from particle sources and maintained by cavities and magnet components inside the particle accelerator. Other facilities, e.g. devices to detect beam motion, vacuum systems to attain excellent beam lifetime, undulators and wigglers to produce high brilliance photon beams, targets for producing secondary beam, are supplemental options with different purposes [52].

This chapter starts with the introduction of the basis of beam dynamics from a physical and mathematical view. This involves Maxwell's equations, Hamiltonian dynamics, and Einstein's special theory of relativity, which are introduced in Section 2.1.

In particular, the space charge effects as a focal issue of this dissertation are explained with different respects to numerical methods in Section 2.2.

Section 2.3 introduces the objectives and research goals of this dissertation.

2.1 Beam dynamics inside particle accelerators

The physics inside particle accelerators is complex and complicated. Many topics are subject to deep study or even beyond our current knowledge level, e.g. the N-body problem arising in beam dynamics, finding of new particles inside the accelerator [1], or investigating the beginning of universe from the evolution of particles [75].

Beam dynamics study is the foundation for all accelerator-related studies, because all experiments rely on certain types of beams. For the precise operation to obtain specific beams, we need to theoretically understand the beam enough in advance.

The beam is guided by external magnetic fields, accelerated by electric fields, and travels relativistically in the accelerator system. The physical fundamentals involve Maxwell's equations, particle motions, and relativity theory, among others.

2.1.1 Maxwell's equations

The E.M. fields' propagation is described by Maxwell's equations [59], whose integral form is:

$$\text{Gauss's law:} \quad \oiint_{\partial\Omega} \mathbf{D}(\mathbf{r},t)\cdot \mathrm{d}\mathbf{S} = \iiint_{\Omega} \rho(\mathbf{r},t)\mathrm{d}V, \tag{2.1}$$

$$\text{Gauss's law for magnetism:} \quad \oiint_{\partial\Omega} \mathbf{B}(\mathbf{r},t)\cdot \mathrm{d}\mathbf{S} = 0, \tag{2.2}$$

$$\text{Faraday's law of induction:} \quad \oint_{\partial\Sigma} \mathbf{E}(\mathbf{r},t)\cdot \mathrm{d}\mathbf{l} = -\frac{\partial}{\partial t}\iint_{\Sigma} \mathbf{B}(\mathbf{r},t)\cdot \mathrm{d}\mathbf{S}, \tag{2.3}$$

$$\text{Ampere's law:} \quad \oint_{\partial\Sigma} \mathbf{H}(\mathbf{r},t)\cdot \mathrm{d}\mathbf{l} = \iint_{\Sigma} \left(\frac{\partial}{\partial t}\mathbf{D}(\mathbf{r},t) + \mathbf{J}(\mathbf{r},t)\right)\cdot \mathrm{d}\mathbf{S}, \tag{2.4}$$

where $\mathbf{D}(\mathbf{r},t)$ is defined as the electric flux density, $\rho(\mathbf{r},t)$ as the electric charge density, $\mathbf{B}(\mathbf{r},t)$ as the magnetic flux density, $\mathbf{E}(\mathbf{r},t)$ as the electric field strength, $\mathbf{H}(\mathbf{r},t)$ as the magnetic field strength, and $\mathbf{J}(\mathbf{r},t)$ as the electric current density. The spatial position is $\mathbf{r}$, and the time variable is t. The domain covering the charged area is denoted as Ω, and the closed boundary of Ω is the surface $\partial\Omega$. If a surface Σ is considered such that the magnetic flux passes through it, then $\partial\Sigma$ is defined as the closed boundary of Σ.

The differential form of Maxwell's equations is:

$$\nabla\cdot\mathbf{D}(\mathbf{r},t) = \rho(\mathbf{r},t), \tag{2.5}$$

$$\nabla\cdot\mathbf{B}(\mathbf{r},t) = 0, \tag{2.6}$$

$$\nabla\times\mathbf{E}(\mathbf{r},t) = -\frac{\partial}{\partial t}\mathbf{B}(\mathbf{r},t), \tag{2.7}$$

$$\nabla\times\mathbf{H}(\mathbf{r},t) = \frac{\partial}{\partial t}\mathbf{D}(\mathbf{r},t) + \mathbf{J}(\mathbf{r},t), \tag{2.8}$$

A single charged particle with charge q experiences the external electric fields $\mathbf{E}$ and magnetic flux densities $\mathbf{B}$ inside the particle accelerator. These E.M. fields are described by Maxwell's equations.

For an electron inside an accelerator, the electric field acts on the electron in the anti-direction of the electric field. The magnetic fields force the electron in the direction which is perpendicular to both $\mathbf{B}$ and $\mathbf{v}$ (the velocity of the electron). The $\mathbf{E}$ acceleration and $\mathbf{B}$ bending are usually separated inside an accelerator. Electric fields can be used for acceleration of particles while magnetic fields can serve to bend particles and as optical lenses. In combination of the E.M. fields with motion, the Lorentz force $\mathbf{F}$ is obtained by $\mathbf{E}$ and $\mathbf{B}$ as:

$$\mathbf{F} = q(\mathbf{E} + \mathbf{v} \times \mathbf{B}). \tag{2.9}$$

The RF cavities provide the major accelerating $\mathbf{E}$ fields, whereas the magnetic components guide the particles. A dipole is used to provide the magnetic flux density $\mathbf{B}$ for guiding the beam. In particular, a circular accelerator uses dipole magnets as the essential elements in which there is no electric field $\mathbf{E}$ for the acceleration of the charged particle, the magnetic flux density $\mathbf{B}$ is perpendicular to the velocity $\mathbf{v}$ in the vertical direction, and the bending radius r of the circular motion can be determined in a simple form as:

$$r = \frac{m\|\mathbf{v}\|}{q\|\mathbf{B}\|}, \tag{2.10}$$

where m is the mass of the particle.

Different components provide different forms of E.M. fields in Eq. (2.9). Combined quadrupoles named as "FODO" cells are used for focusing and defocusing the beam. The sextupoles can be used to cancel some of the chromatic aberration (chromaticity) from quadrupoles [96].

2.1.2 Multi-particle system in Hamiltonian dynamics

In comparison to an analysis of a single reference particle, multi-particle systems show the real dynamic situation inside the particle accelerator. The E.M. interactions among the enormous number of particles play an important role for the dynamics of the multi-particle system. Hence, the charge density $\rho(\mathbf{r}, t)$ in Eq. (2.1) feels not only the external driven E.M. fields $\mathbf{E}_{\text{driven}}$, but also the interactive E.M. fields $\mathbf{E}_{\text{interactive}}$ of the charged particles themselves.

For the multi-particle system study of an electron beam, phase space variables $(\mathbf{r}, \mathbf{p})$ are defined, where $\mathbf{r}$ represents position and $\mathbf{p}$ denotes momentum. The charge density results from a more fundamental quantity in physics: the particle distribution function $f(\mathbf{r}, \mathbf{p}, t)$. The connection is linked by the following two formulas:

$$\rho(\mathbf{r}, t) = n(\mathbf{r}, t)e, \tag{2.11}$$

where $n(\mathbf{r}, t)$ is particle density, e is the unit charge, and

$$n(\mathbf{r}, t) = \int f(\mathbf{r}, \mathbf{p}, t)\mathrm{d}\mathbf{p}. \tag{2.12}$$

The dynamics of the charge density is derived from the dynamics of the particle distribution, which is a hard problem named the N-body problem of identical particles. The problem has not been solved in mathematics so far [25]. Hamiltonian dynamics is a way to describe the N-body problem by Hamiltonian function $\mathcal{H}(X)$ through Hamiltonian equation:

$$\dot{\mathbf{r}}_l = \partial_{\mathbf{p}_l}\mathcal{H}, \quad \dot{\mathbf{p}}_l = -\partial_{\mathbf{r}_l}\mathcal{H}, \tag{2.13}$$

where l defines the lth particles of the N-particle system $X = (x_1, \cdots, x_N)$. $f(\mathbf{r}, \mathbf{p}, t)$ is also denoted as $f_N(X; t)$ for the N-particle system.

The evolution of the N-particle system is given by Liouville's equation under Hamilton dynamics. The Bogolyubov-Born-Green-Kirkwood-Yvon (BBGKY) hierarchy is a series of coupled equations transformed from Liouville's equation. However, the coupled equations are not easier to solve, because a $f_s(X; t)$ (s-particle distribution function) is determined by $f_{s+1}(X; t)$ ($(s+1)$-particle distribution function) in the whole particle interaction system. Assumptions and simplifications have to be declared in order to solve the equations. Truncation of the BBGKY hierarchy is a common way to achieve an initial simplification: we assume that all particles are independent. For the two-particle distribution function we have

$$f_2(x_1, x_2; t) = f(x_1; t)f(x_2; t). \tag{2.14}$$

By assuming a conservative system (i.e. the system is free of damping or diffusion effects due to external sources) and applying the relevant formula derivations and theories, the Vlasov equation leading to the description of multi-particle beam dynamics can be derived as:

$$\partial_t f + \frac{\mathbf{p}}{m} \cdot \partial_{\mathbf{r}} f - \mathbf{F} \cdot \partial_{\mathbf{p}} f = 0. \tag{2.15}$$

The force $\mathbf{F}$ represents the sum of the external force and the interactive force. Still, the acting force $\mathbf{F}$ mainly represents the E.M. force on the charged particles. We apply the Lorentz force:

$$\mathbf{F} = q(\mathbf{E}_{\text{driven}} + \mathbf{E}_{\text{interactive}} + \frac{\mathbf{p}}{m} \times \mathbf{B}), \tag{2.16}$$

where the $\mathbf{E}_{\text{driven}}$ is the electric field provided by the RF cavity, and the $\mathbf{E}_{\text{interactive}}$ is the electric field provided by the beam-self field of the particle ensemble, which is described by Poisson's equation:

$$\nabla \cdot \mathbf{E}_{\text{interactive}} = \int f(\mathbf{r}, \mathbf{p}, t)\mathrm{d}\mathbf{p}. \tag{2.17}$$

The equations of the Vlasov-Maxwell system, (2.1)-(2.4), (2.15), (2.16) and (2.17), are useful tools to determine the evolution of a multi-particle system under the influence of the forces depending on the physical parameters of the system through

differentiable functions. The Vlasov equation can also be enhanced by adding perturbation terms and damping terms in order to describe a real system.

Additionally, the real system of physical events is complex: certain processes with forces caused by purely statistical process are involved, e.g. by the quantized emission of synchrotron radiation photons, by collisions with other particles within the same bunch, or residual gas atoms. Statistical processes are considered and lead to Fokker-Planck's equation [76] which is identical to the Vlasov equation excluding the statistical excitation terms.

The bunch distribution contains a great number of particles, which makes the direct analysis of interactions among particles difficult. Furthermore, the motion of charged particles finally reaches relativistic velocity inside the particle accelerator. Based on the theory of relativity, the analysis becomes more complicated. Instead of the theoretical analysis, numerical models are used to manage most studies. Solving for the field $\mathbf{E}_{\text{interactive}}$ precisely with large number N directly leads to methods which calculate with mesh-free (particle-particle) models, e.g. the tree-based method, and the fast multiple method [36] [37]. However, mesh-related methods are more common in the research community. The Particle-in-Cell (PIC) model is the most common tool for studying and is introduced in Section 2.1.3, and further discussed in Section 2.2.

Space charge effects are defined as the influence on the particle's motion caused by the E.M. fields of the charged particles themselves. They are responsible for many phenomena in beam dynamics: the betatron tune shift, the synchrotron tune shift, energy loss, energy spread, and instabilities. However, there are still many behaviors which are not fully understood, e.g. the beam motion after the photoinjector in free-electron laser (FEL), and beam-cloud interaction. All these applications need the space charge force calculation.

Some beam parameters are introduced in the following, and used in Chapter 6 for comparisons.

Beam size: the scaling parameter to describe the beam scope. The root mean square (rms) beam size used in simulation is defined as:

$$\text{RMS}.x = \sqrt{\langle x^2 \rangle}, \tag{2.18}$$

where

$$\langle x^2 \rangle = \frac{\Sigma x_l^2}{N} - \left(\frac{\Sigma x_l}{N} \right)^2 . \tag{2.19}$$

For the N particle distribution, $\Sigma x_l = \Sigma_{i=1}^{N} x_l$ is performed as the sum of the N particles.

The beam size should be as small as possible in order to maximize the electron density especially along the linac and undulator. However, if the beam size is too small, diffraction effects will appear. In practice, an optimum beam size will be realized by inserting quadrupoles as the FODO channel to balance the effects. To study the optimum beam size, numerical simulations are required to determine the best parameter for the FODO channel [96].

Beam emittance: the area of a particle beam in phase space, for both transverse phase space and longitudinal phase space. It is a statistical definition, and the normalized rms emittance (e.g. $(x, \mathbf{p}_x)$ phase space) for numerical beam studies is presented in the following [34] [33]:

$$\varepsilon_{\mathrm{N,rms}} = \frac{1}{m_0 c}\sqrt{\langle x^2\rangle\langle \mathbf{p}_x^2\rangle - \langle x\mathbf{p}_x\rangle^2}, \tag{2.20}$$

where

$$\langle \mathbf{p}_x^2\rangle = \frac{\Sigma \mathbf{p}_{x,l}^2}{N} - \left(\frac{\Sigma \mathbf{p}_{x,i}}{N}\right)^2, \tag{2.21}$$

$$\langle x\mathbf{p}_x\rangle = \frac{\Sigma x_l \mathbf{p}_{x,l}}{N} - \frac{\Sigma x_l \Sigma \mathbf{p}_{x,k}}{N^2}. \tag{2.22}$$

In the design of accelerators, a low emittance is preferred since most of the particles are confined into a small area that suits the beam chamber and magnets in the accelerating system. The low emittance guarantees obtaining high luminosity for the collider accelerator, and high brilliance for the light source accelerator. Numerical simulations can predict the emittance growth in acceleration, and further influence of luminosity or brilliance [96].

2.1.3 Particle-in-cell model

Functional analysis and resolutions are difficult and varied in calculations, thus the distribution function for the particles is not straightforward in numerical computations. Replacing the distribution function, the discrete charge density is taken for numerical computing.

In particular, replacing the direct solution of Vlasov-Maxwell equations with the PIC model is a proper choice. It is a particle-mesh method that treats the particles by approximating the quantity on a mesh. This dissertation specifically deals with the PIC model using equations of motion. In computation, the differential operators are replaced by the finite difference formulas, e.g. Laplacian is approximated on the mesh as the Laplace matrix. All the calculations of force, field, potential are computed on the mesh. In contrast, the calculations of particle assigning, mesh interpolating, and particle movement are also computed for each particle.

For the purpose of computational efficiency, the charged particles are replaced by macro particles in a bunch which usually contains a large number of particles (for instance $10^6 \sim 10^{13}$) concentrated around one synchronous particle. The dynamic motions are discretized by the time step δt. The data of the bunch particles are initialized on the laboratory frame. The velocity of the charged particle becomes nearly relativistic, the Lorentz transform between the laboratory frame and the rest frame is necessary for a couple of physical quantities, e.g. the position and the E.M. fields. Because the calculation of E.M. fields by Poisson's equation should be done in the rest frame.

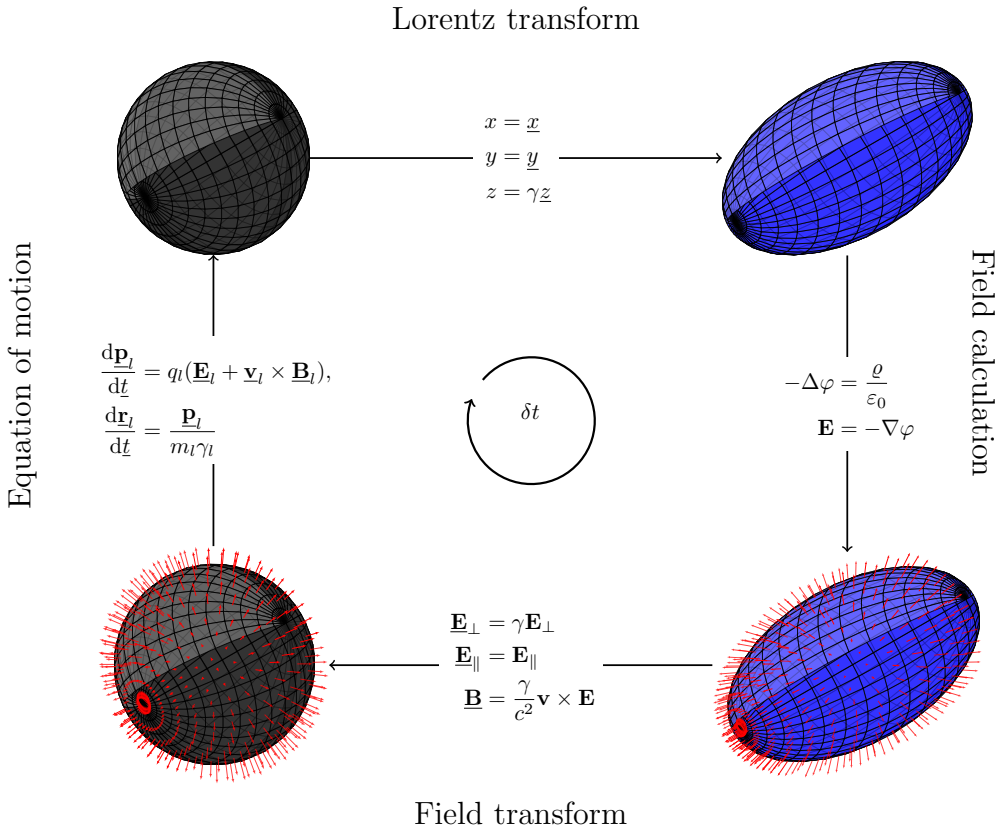


Figure 2.1: A schematic model of bunch motion with transformation between the laboratory frame (left) and the rest frame (right).

The procedures are summarized in Figure 2.1, whereby four procedural steps are repeated in a cycle for time step δt until a certain time limit or distance limit is reached. Here, the Cartesian coordinates and a Cartesian mesh is used in the algorithm.

Firstly, the routine loads bunch information and transforms the bunch position from the laboratory frame to the rest frame by a Lorentz transform (see Figure 2.1 from left top to right top):

$$\begin{aligned} x &= \underline{x}, \\ y &= \underline{y}, \\ z &= \gamma\underline{z}. \end{aligned} \tag{2.23}$$

Here, the z direction is the bunch's travel direction and the coordinates $(\underline{x}, \underline{y}, \underline{z})$ have been transformed to (x, y, z) with the Lorentz factor $\gamma = 1/\sqrt{1-\beta^2}$, where β is defined by $\beta = \underline{\mathbf{v}}_{\underline{z}}/c$ with the velocity in z direction $\underline{\mathbf{v}}_{\underline{z}}$ and the speed of light c.

Secondly, we solve the E.M. fields in the rest frame. As a start, the charge of each macro particle is assigned to the mesh. A cubic domain, Ω, over $[0, L_x] \times [0, L_y] \times$

$[0, L_z]$, is taken to cover the charged particles. L_x, L_y, L_z are the lengths in each direction. The above $L_x \times L_y \times L_z$ domain is discretized to $N_x \times N_y \times N_z$ mesh points within each direction. By equidistant discretization, the step sizes are formed as: $h_x = \frac{L_x}{N_x - 1}$, $h_y = \frac{L_y}{N_y - 1}$, $h_z = \frac{L_z}{N_z - 1}$. A non-equidistant discretization can allow denser meshing at the position of the particle distribution and a coarser meshing elsewhere. Thus, the parameter particles-per-cell is balanced in comparison with equidistant discretization.

For each particle, the charge is assigned to the neighboring grid points (Figure 2.2) with special assignment functions for different schemes, e.g. the Nearest Grid Point (NGP) scheme, Cloud In Cell (CIC) scheme, and Triangular Shaped Cloud (TSC) scheme [44].

The NGP scheme is the simplest charge assignment of the PIC model. As the name suggests, the charge of a given particle is assigned to its nearest mesh point. Furthermore, the force value from the nearest mesh point is taken as the particle's force. The NGP scheme, whose obtained interparticle forces are discontinuous in value, is not often used in practice.

The CIC scheme gives a better approximation and much smoother force than the NGP scheme. Instead of assigning to the nearest grid point, the charge of a particle is assigned to the neighboring grid points which belong to the weighting cell. For an M dimensional problem, the number of neighboring points in a cell is 2^M. Because of the particles being displaced with respect to the mesh, the interparticle forces are continuous in value, but discontinous in their first derivative (see Figure 2.2).

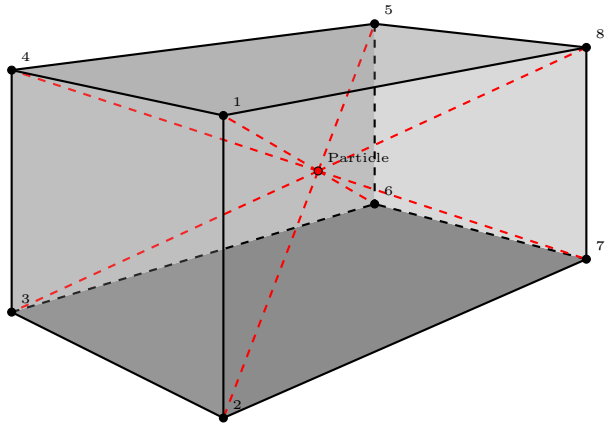


Figure 2.2: A schematic plot of the particle assignment of CIC scheme. A fraction of its charge is assigned to each of the eight neighboring grid points of a particle at some position inside the grid cell.

TSC is an even more accurate scheme. It uses the piece-wise quadratic function with more grid points for assigning charge of a particle. For an M dimensional problem, the number of neighboring points in a cell is 3^M. This results in continuity of value (first order) and first derivative (second order).

Introductions to these charge assignment methods with different orders and smoothness can be found in [44] and [9]. For the applications in this dissertation, the CIC scheme of the 3D PIC model is chosen for numerical calculations.

The introduced mesh provides a lot of convenience in the numerical calculations of electric field. As introduced before, the particles affect each other due to the space charge forces generated by the particles themselves at the beginning sections of the accelerator. The electric field $\mathbf{E}_{\text{interactive}}$ has to be computed through Poisson's equation (for each time step δt). Efficiency is the main advantage of the PIC model as Poisson's equation is discretized on the mesh for various discrete numerical methods. In contrast, it is more time-consuming and complicated to solve Poisson's equation with the particle distribution function.

The E.M. fields are computed on the mesh rather than on the involved particles. The computation of the electric field caused by space charges directly connects to the electrostatic potential φ based on the formula between $\varphi(x, y, z)$ and $\mathbf{E}(x, y, z)$:

$$\mathbf{E}(x, y, z) = -\nabla\varphi(x, y, z). \tag{2.24}$$

φ is the solution of Poisson's equation,

$$-\Delta\varphi(x, y, z) = \frac{\rho(x, y, z)}{\varepsilon_0}, \text{ in } \Omega \subset \mathbb{R}^3, \tag{2.25}$$

with the Laplace operator Δ, the charge density ρ, the permittivity in vacuum ε_0, and the considered domain Ω.

Thirdly, the updated E.M. fields have to be transformed back from the rest frame to the laboratory frame (see Figure 2.1 from right bottom to left bottom):

$$\underline{\mathbf{E}}_{\perp} = \gamma\mathbf{E}_{\perp}, \tag{2.26}$$

$$\underline{\mathbf{E}}_{\parallel} = \mathbf{E}_{\parallel}, \tag{2.27}$$

$$\underline{\mathbf{B}} = \frac{\gamma}{c^2}\mathbf{v} \times \mathbf{E}, \tag{2.28}$$

where $\perp$ stands for the transverse directions and $\parallel$ for the longitudinal direction.

The transformed fields are further interpolated from the mesh to each particle l as $\underline{\mathbf{E}}_l$ and $\underline{\mathbf{B}}_l$. The method of the field interpolation corresponds to the applied charge assignment scheme. Additionally, the driven E.M. fields should be added up if the bunch travels in the RF cavity, or other E.M. components.

Fourthly, instead of solving the Vlasov equation for the particle distribution in the Hamiltonian system, we solve the equation of motion for particles. The individual particle is moved by the relativistic equations of motion for particle l. The equations in the laboratory frame are given as:

$$\frac{\mathrm{d}\underline{\mathbf{p}}_l}{\mathrm{d}\underline{t}} = q_l(\underline{\mathbf{E}}_l + \underline{\mathbf{v}}_l \times \underline{\mathbf{B}}_l), \tag{2.29}$$

$$\frac{\mathrm{d}\underline{\mathbf{r}}_l}{\mathrm{d}\underline{t}} = \frac{\underline{\mathbf{p}}_l}{\underline{m}_l\gamma_l}, \tag{2.30}$$

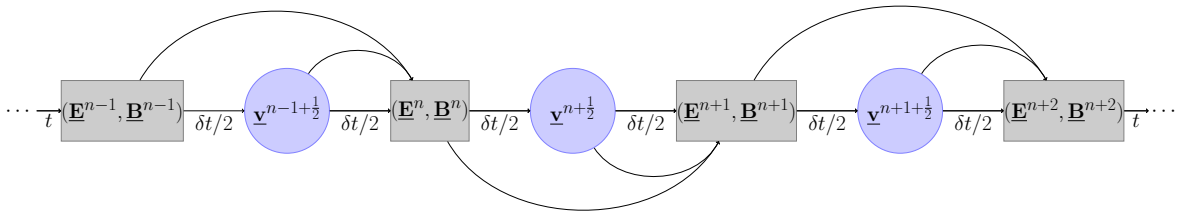


Figure 2.3: A schematic plot of leap-frog algorithm.

where $\underline{\mathbf{p}}_l$ denotes the momentum and $\underline{m}_l$ the rest mass. Furthermore, $\underline{\mathbf{r}}_l$ refers to the position and $\underline{\mathbf{v}}_l$ to the velocity with $\beta_l = \underline{\mathbf{v}}_{\underline{z},l}/c, \gamma_l = 1/\sqrt{1-\beta_l^2}$ for particle l.

For the time integration of the ordinary differential equation, we can use either the Runge-Kutta or the leap-frog scheme.

The leap-frog scheme is well-used in updating E.M. fields with time integration calculations. The stability can be determined by Courant-Friederichs-Lewy stability condition [53]:

$$\delta t < \delta t_{max}, \tag{2.31}$$

where

$$\delta t_{max} = \frac{1}{c} \frac{1}{\sqrt{(\frac{1}{\Delta x})^2 + (\frac{1}{\Delta y})^2 + (\frac{1}{\Delta z})^2}}, \tag{2.32}$$

Δx, Δy and Δz are step sizes in space.

The time step δt is proceeded by $\delta t/2$ for the discrete time integration. The leap-frog algorithm forwards the $(\mathbf{E}, \mathbf{B})$ and $\mathbf{v}$ with different half time steps as shown in Figure 2.3 and [9]. It is an explicit scheme and can be implemented very efficiently on modern computer architecture, therefore we choose the leap-frog algorithm for the time integration for applications in this dissertation.

2.2 Mesh-related calculation of space charge effects

For the E.M. fields generated by the beam in the multi-particle system, regardless of the space charge effects, which are mainly induced from the self-bunch, there are also sources from the previous bunches called wakefields [18], [89], [92]. Generally, the E.M. fields induced by the two types of bunches are named as collective effects which are always studied for the beam instability in accelerator design and operation.

Additionally, space charge effects are classified by direct space charge and image space charge. The direct space charge comes from the Coulomb interactions inside the charged particle bunch. The image space charge represents the "virtual" charge from the metallic surface screen of the vacuum chamber, acceleration cavity, or other surroundings.

Statically, the space charge effects depend on some facts of the beam and surroundings: the charge of the particle bunch, the geometry of the beam, and the vacuum chamber's shape. For instance, the space charge force acting on the bunch is large if the charge of the bunch is massive; the different distributions of particles inside a bunch make the acting forces different; the variety of circular, ellipse and rectangular shapes of the vacuum chamber reflects assorted equi-potential planes for the image charge, etc.

Dynamically, the numerical calculation methods, image charge and other conditions in space charge effects calculation can be treated differently in different regimes inside the particle accelerator. The image charge generated from the surface of the cathode of a photo cathode electron gun should be under consideration for the space charge effects. After a short distance, the direct space charge is the major source of space charge effects while the image charge from the metallic surface screen is limited. Here, it is reasonable to neglect the image charge in a simulation since the space charge force obtained from the image charge is very low. Since the beam is concentrated in the center of the vacuum chamber in transverse directions, the size of the beam is far smaller compared to the vacuum chamber size in most application cases. For instance, the superconducting TESLA cavity [6] has a radius in centimeters while the transverse beam size is in the millimeter range. Furthermore, the space charge effects affect the motion of the bunch in the low energy regime of particle accelerator, and are neglected for high energy bunches in most cases.

As mentioned, the space charge effects act in the form of the electric field $\mathbf{E}$, which is related to the electrostatic potential φ by Eq. (2.24).

For a volume of continuous charge density ρ,

$$\varphi(\mathbf{r}) = \frac{1}{4\pi\varepsilon_0} \int \frac{\rho(\mathbf{r}')}{\iota} \mathrm{d}\mathbf{r}', \tag{2.33}$$

where ι is the distance from charge to a position $\mathbf{r}$. The density $\rho(\mathbf{r}')$ corresponds to the local charged particles per volume unit. In the next chapters, $1/\iota$ is understood as GF. In fact, the whole dissertation discusses the numerical extensions of this form.

From the view of all charged particles, the computation of the electrostatic potential connects to the solution of Poisson's equation by using Gauss's Law for the surface of a considered domain Ω:

$$-\Delta\varphi(\mathbf{r}) = \frac{\rho(\mathbf{r})}{\varepsilon_0}, \text{ in } \Omega, \tag{2.34}$$

where the charge density is $\rho(\mathbf{r})$.

The above two equations (2.33) and (2.34), relating to the potential φ, are essentially the same in physics. In principle, each is the inverse problem of the other (see Appendix A). In computation, they bring the mesh-related numerical study into the PIC model. Eqs. (2.33) and (2.34) can share the same mesh, but may lead to different numerical approaches, shown as integral and differential forms. In the following we will discuss more details about different aspects of the mesh-related methods.

2.2.1 Dimensions

For the spatial dimension, a couple of options are available to choose in simulations, e.g. 2D, 2.5D and 3D. A 2D simulation is usually described by a slice in the transverse directions, or a slice in the longitudinal direction. A 2.5D simulation is defined by a number of transverse 2D slices along the longitudinal direction. Both the 2D and the 2.5D case simplify the real physics, which may lead to a loss of information in the study of the beam dynamics. In 3D, the real spatial situation inside the accelerator is taken into account. This dissertation studies the 3D case.

2.2.2 Computational domain and boundary conditions

As mentioned, the currently used vacuum chamber's radius is still much larger than the transverse lengths of the bunch, and the image space charge can reasonably be neglected. Furthermore, in the bunch tracking related simulation it is appropriate to consider a domain which is only slightly larger than necessary to cover the bunch as only the E.M. fields in the regime of the bunch itself are useful for the particles of the bunch. The bunch shape is various and dynamic, the considered domain can be set as a cube, cylinder, or other regular shape relating to the further numerical methods and computational efficiency. In most simulation codes, a cubic domain is the usual choice since the dynamic bunch shape will not match any regular domain. There is no evidence to show the cubic domain is less accurate than other choices, and it is probably the most efficient in computation. If the considered domain just covers the surrounding bunch, we define this calculation as near-bunch domain calculation.

In contrast, the aperture of the RF cavity may be small for some accelerators in order to achieve higher geometrical shunt impedance of the fundamental mode, and (or) save costs. In some cases, the E.M. fields from the surface of the RF cavity to the bunch may need to be considered for certain purposes. In these cases, the effects of the image space charge on the surface screen are not negligible. A computational domain which fits the shape of the inner geometry of the RF cavity should be considered for reasons of accuracy. In this case, a cylinder (or a pseudo cylinder in a cube) may be preferred. We refer to this calculation as far-bunch domain calculation.

There is a large area of vacuum included if the considered domain fits the aperture of the RF cavity. The considered domain is difficult to fit the vacuum chamber in reality due to the transverse radius of the chamber varying with the cavity's shape in longitudinal direction. For further numerical calculations, the discretized charge density (r.h.s.) values contain numerous zeros in the implementation, and the mesh-particle assignment is proceeded in a rough way compared to near-bunch domain. The right-hand side (r.h.s) function is discontinuous, and loses some detailed information of the discretized charge density. These properties may call for extra effort to reflect the real E.M. fields.

Other types of considered domains e.g. irregular domains are not popular in beam dynamics simulations.

For a single bunch simulation, the longitudinal boundary condition is mostly free-

space. The transverse boundary conditions are Dirichlet boundary conditions. Periodic boundary conditions with symmetric extensions in transverse directions may be better to fit the real physics situations for screen symmetry. In some cases, this can be chosen for approximation. In the near-bunch domain simulation, the corresponding boundary conditions are set to be free-space in all three directions.

This dissertation studies the cubic domain with free-space boundary conditions for the near-bunch domain simulation.

2.2.3 Coordinate settings

The coordinate settings should match the considered domain: Cartesian coordinates fit the cubic domain, cylindrical coordinates suit the cylindrical domain, and the elliptic cylindrical coordinates match the elliptic shape in transverse. Other types of coordinates may not be suitable to cover the geometry of the domain.

In cylindrical coordinates, the coordinates (r, θ, z), or (radial, azimuthal, vertical), are mostly chosen through a number of different cylindrical notations. In Cartesian coordinates, (x, y, z) are chosen as usual.

This dissertation studies Cartesian coordinates with equidistant discretization. The non-equidistant discretization is also an option for reflecting a solution, but may need preprocessing and/or postprocessing for the results. For the GF-FFT method discussed in this dissertation, the equidistant mesh points can be directly proceeded by FFTs in comparison with non-equidistant discretization.

2.2.4 Classical numerical methods for Poisson's equation

The numerical solution of Poisson's equation is not difficult to obtain, even in an irregular domain. The real issue is achieving efficiency with reasonable accuracy in practice. Because the charge density is updated during every time step, the Poisson solver has to be repeated within each time step. In the beam dynamics simulation, the Poisson solver generally dominates the time consumption, and this can obstruct the whole simulation.

In Cartesian coordinates (x, y, z), the Laplace operator is expressed as:

$$\Delta = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2}. \tag{2.35}$$

If the Taylor series is used for deriving the discrete formulation, the popular finite difference method (FDM) is obtained and easily implemented in computers. There are forward difference, backward difference, and central difference formulas. The most commonly used one is a seven points central difference for the 3D Laplace operator and reads as:

$$\frac{-2\varphi_{i,j,k} + \varphi_{i-1,j,k} + \varphi_{i+1,j,k}}{h_x^2} + \frac{-2\varphi_{i,j,k} + \varphi_{i,j-1,k} + \varphi_{i,j+1,k}}{h_y^2} + \frac{-2\varphi_{i,j,k} + \varphi_{i,j,k-1} + \varphi_{i,j,k+1}}{h_z^2}.$$

The discrete Laplace operator can also be expressed by stencil matrices using more neighbor points for a higher order solution [43]. Finally, all these various discretizations result in a linear system. The discretized matrix is usually sparse, symmetric and positive if the computational domain and boundary conditions are chosen properly. There are dozens of options to solve the linear system derived from Poisson's equation, for instance, the direct solvers which utilize the matrix decomposition, e.g. Cholesky decomposition, band Cholesky decomposition; the classical iterative methods, e.g. the Jacobi method, the Gauss-Seidel method, and the successive over-relaxation (SOR) method; Krylov subspace methods [80], e.g. the conjugate-gradient (CG) method, the bi-conjugate-gradient stabilized (Bi-CGStab) method; the spectral-DFT (FFT) direct method; the cycle reduction method; FACR method [82]; the domain decomposition method [81]; and multigrid methods.

For the discrete linear system derived from Eq. (2.34), the multigrid method is mostly preferred for its high speed convergence compared to other numerical methods. Possible contributions are [30], [7], [14], [15].

For the discrete integration form derived from Eq. (2.33), the GF-FFT method is the classical numerical method used in beam dynamics studies. However, the GF values for different computational domains and boundary conditions are difficult to obtain. Thus the boundary conditions are limited to be free-space boundary conditions. This dissertation focuses on the further efficiency improvement of this method.

From the numerical solution side for mesh-related methods, alternatives for the FDM are possible, e.g. the finite element method [19], [83], and the finite integration technique [90], [91], [88].

2.2.5 Parallel calculation of space charge effects

HPC has played an important role for modeling, simulation, and data analysis in the design, optimization, and operation of current and future accelerators.

The parallel calculation enhances modeling and simulation results to make them much closer to a realistic scenario. For accuracy, we can use the real number of particles instead of macro particles, and finer mesh instead of coarse mesh. For efficiency, the computational time is shortened by increasing process numbers.

A parallel routine has to be programmed differently from the serial routine for HPC. The program does not only focus on the implementation of the algorithm but also on the speed-up with a multi-processes pattern.

For beam dynamics simulations, it is now increasingly popular to use the HPC architecture in all phases of accelerator design and study. Some parallel algorithms on space charge effects are found in [35], [99], [2], [73], [28]. Integrated Map and Particle ACcelerator Tracking Code (IMPACT) is a parallel PIC code suite for modeling high intensity, high brightness beams in RF proton linacs, electron linacs and photoinjectors [69] [74]. Object Oriented Parallel Accelerator Library (OPAL) is an open-source package for general particle accelerator simulations including 3D space charge, short range wake fields and particle matter interaction [3], [4], [61]. A Space

Charge Tracking Algorithm (ASTRA)-parallel ([32], [62]) is also free of charge for non-commercial and non-military use. It covers particle injecting, bunch tracking with 3D (or 2D) space charge calculations and wake fields. A Many-GPGPU Particle-in-Cell Code (PIConGPU) [16] is specifically programmed for the GPU-level architecture. All packages, and others, meet the challenge of computing the space charge effects efficiently.

As the third pillar of science, computational simulation with HPC certainly deserves to be a key direction of the future numerical beam dynamics study. This dissertation discusses different routines of a novel GF-FFT algorithm for different HPC architectures.

2.3 Objectives and research goals

The more advanced the technologies of particle accelerators are, the more precise are the requirements on the quality of the beam. Several influences on the dynamics of the particle beam can severely decrease the performance of the accelerator facility. Thus, a detailed study of beam dynamics is very important in all phases of the establishment of a new machine: design, commissioning, and operation. In order to meet the needs of the next-generation accelerator facilities, the numerical methods for the computation of beam dynamics issues have to be further developed.

The detailed investigation of space charge related beam dynamics is a crucial issue for the design and operation process of an accelerator facility. The repulsive Coulomb field caused by space charge effects of a beam can generate a defocussing force that reduces the external focusing. Yet, almost all low energy synchrotrons suffer space charge induced emittance growth. The electron gun for the next generation light source is also affected by space charge effects, especially from the densely charged bunch [40].

G. Pöplau has developed and implemented fast methods for space charge computations in 3D: the main field of activity is adaptive multigrid methods [66], [63], [64], [67], [65] for Poisson's equation. These algorithms - Poisson solvers - are available with the software package Multigrid for non-equidistant grids to solve Poisson's equation (MOEVE), which is under the regents of the university of Rostock. MOEVE has been further enhanced by A. Markovik into the MOEVE-PIC software package. This tracking code is now applied to the simulation of the interaction of a positron beam with an electron cloud [54], [55], [56], [57], [58] .

Although the multigrid approach is more flexible with respect to boundary conditions and adaptive discretizations, the GF-FFT technique achieves better simulation results for free-space boundary conditions. The GF-FFT method [44] is correct but not very efficient since it needs many more grid points in the large aspect-ratio direction in order to resolve the variation of the GF. A further development was made in [70], [71] with the introduction of the IGF method. This approach allows the computation of fields of very long or very short bunches correctly. However, the computations are complex and time-consuming, which requires further improvement.

The resulting aim of this PhD work follows as:

- The numerical GF-FFT convolution routine formed by Hockney and Eastwood should be integrated into the MOEVE-PIC package.
- The further IGF method for the GF-FFT convolution routine should be studied and also integrated into the MOEVE-PIC package.
- In order to optimize the efficiency of the IGF method, a couple of efficient GF methods should be researched.
- The alternative approaches of the GF-FFT convolution routine should be further developed.
- The corresponding parallelization of the achieved efficient algorithms should be programmed and carried out for different computer architectures.

The FFT Poisson solver with the IGF found by J. Qiang [71] [72] proofed to achieve correct simulation results for very long or short bunches. When the Poisson solver is repeated again and again, more attention needs to be paid to the elapsed time of the Poisson solver. Even though the FFT implementation is optimized for modern computers, there is always demand for faster implementation times. Further, the computation of the IGF increases the time consumption while increasing the accuracy of very long or short bunches. This approach has to be optimized in efficiency for 3D space charge computations. The IGF-FFT routine is manually separated into three main parts of implementation: the GF values computation, and the DFT (inverse discrete Fourier transform (IDFT)) of the extension vector of GF vector, the DFT (IDFT) of the zero-padded charge density. The efficiency improvements here consider each of these three parts separately. The algorithms are programmed in C language and further integrated into the MOEVE-PIC package. The code is tested in a workstation with a 3.7 GHz Intel Xeon(R) CPU and a Quadro K4000 NVIDIA GPU. Most Poisson solvers from other simulation codes would benefit from the algorithm optimization.

The efficiency of simulation is so important that more and more beam simulations are implemented in multi-core CPU and supercomputers with parallel routines. The outcome of the optimization task should therefore also be simultaneously programmed in parallel routine, e.g. with OpenMP API for shared memory multi-core CPU parallelization, with the CUDA library for GPU parallelization, with the MPI library for distributed memory parallelization in supercomputers, and the combination of the above frames, e.g. OpenMP+CUDA, MPI+OpenMP. The MOEVE-PIC software package is further enhanced by the OpenMP API for the implementation of a workstation. For the implementation in supercomputers, the parallel routine has been carried out in IMPACT at Lawrence Berkeley National Laboratory (LBNL) in the USA. As the GF-FFT routine is the essential Poisson solver for nearly all beam dynamics simulation codes, the optimizations from both algorithm and technology will positively influence other parallel simulation codes in the research community.

3 Numerical studies of Green's functions with free-space boundary conditions in electrostatics

Mathematically, GF is an integral kernel function solving differential equations, such as the partial differential equations (PDEs) with certain boundary conditions. Methods utilizing the GF are widely used in the areas of mathematics, physics, and engineering. In electrostatics, determining the electrostatic potential φ poses a fundamental problem: solving Poisson's equation as Eq. (2.34). The electrostatic potential φ connects the essential electric field $\mathbf{E}$ by Eq. (2.24).

For a single point charge, the corresponding fundamental potential is known as GF. Additional GFs are also known as the integral kernel to solve PDEs for various domain shapes and boundary conditions. The free-space boundaries express the decay condition as $|\mathbf{r}| \to \infty$:

$$|G(\mathbf{r})| \to 0.$$

φ can be obtained as the convolution of the charge density and GF within the computational domain. The result of numerical convolution is often obtained by the mid-point integral rule. The details can be found in Section 3.1.

In some cases, the mid-point rule is not sufficient for a large longitudinal-to-transverse ratio domain. A further integral for the discrete convolution is studied, which is based on the primitive function of GF. For this part, Section 3.2 provides more information.

However, the time consumption of the upgraded numerical convolution is significant. A selection of efficient versions of GF are introduced in Section 3.3.

Finally, the trivial FFT convolution routine is presented in Section 3.4, which is improved within the scope of this dissertation as described in Chapter 4.

Some of the results described in this chapter have already been published in own publications [102] [100].

3.1 Green's function

3.1.1 Green's function

Poisson's equation with free-space boundaries is commonly used in simulations of beam dynamics. In reality, the free-space boundary conditions do not fit the accelerator's chamber, but insofar as the bunch concentrates near the center of the chamber the free-space boundary conditions are a valid approximation in simulations. This fact has also been accepted by the research community as most simulations use the free-space boundary conditions as well.

GF in free-space with Cartesian coordinates is presented as:

$$G(x, x', y, y', z, z') = \frac{1}{\sqrt{(x-x')^2 + (y-y')^2 + (z-z')^2}}, \tag{3.1}$$

which is given by the reciprocal distance between two points in the free space. In some cases, in particular for the calculation side, the GF can be described directly as the distance's function. Similarly, treat the Eq. (3.1) by the coordinate translation, substituting $w - w'$ by w for w in $\{x, y, z\}$ and thus use $(0, 0, 0)$ instead of w', i.e.

$$G(x, x', y, y', z, z') = G(x - x', y - y', z - z', 0, 0, 0)$$

Therefore, the GF can also be simplified as

$$G(\mathbf{r}) = \frac{1}{\mathbf{r}}, \text{ or}$$
$$G(x, y, z) = \frac{1}{\sqrt{x^2 + y^2 + z^2}}. \tag{3.2}$$

This simplified formula is often used in discrete computation since the GF is always pre-calculated in practice.

Using GF, the solution of Poisson's equation in $\mathbf{R}^3$, i.e. the continuous electrostatic potential φ, reads as [44] [71]:

$$\varphi(x, y, z) = \frac{1}{4\pi\varepsilon_0} \cdot \iiint \rho(x', y', z') G(x, x', y, y', z, z') \mathrm{d}x' \mathrm{d}y' \mathrm{d}z'. \tag{3.3}$$

$G(x, x', y, y', z, z')$ is the integral kernel, which is convoluted with the charge density $\rho(x', y', z')$. In practice, the continuous solution is not always the case. ρ is expressed in discrete grid points as obtained in the last chapter. Although it can be approximated by continuous or basic functions, the integral cannot be solved directly or easily by the way of analysis. In total, the discrete computation methods are more competitive in various aspects, e.g. feasibility, execution time.

The cubic computational domain Ω, which is obtained from the last chapter, is already discretized by N_x, N_y and N_z steps, respectively, in each coordinate direction with equidistant step sizes h_x, h_y, h_z. The integral in Eq. (3.3) on the domain Ω is

discretized on the same grid as introduced in Section 2.1.3 for the charge distribution $\rho(x_{i'}, y_{j'}, z_{k'})$. Then, the discrete integral formula is given by

$$\varphi(x_i, y_j, z_k) \approx \frac{1}{4\pi\varepsilon_0} \cdot \sum_{i'=0}^{N_x-1} \sum_{j'=0}^{N_y-1} \sum_{k'=0}^{N_z-1} \rho(x_{i'}, y_{j'}, z_{k'}) \tilde{G}(x_i, x_{i'}, y_j, y_{j'}, z_k, z_{k'}), \tag{3.4}$$

where the grid points (x_i, y_j, z_k) are the center points of each integral. The integral cell is equal to the individual grid cells with side lengths h_x, h_y and h_z. Thus, the integral over one grid cell reads as:

$$\tilde{G}(x_i, x_{i'}, y_j, y_{j'}, z_k, z_{k'}) = \int_{x_i-h_x/2}^{x_i+h_x/2} \int_{y_j-h_y/2}^{y_j+h_y/2} \int_{z_k-h_z/2}^{z_k+h_z/2} G(x_i, x', y_j, y', z_k, z') \mathrm{d}x' \mathrm{d}y' \mathrm{d}z'. \tag{3.5}$$

The $\tilde{G}(x_i, x_{i'}, y_j, y_{j'}, z_k, z_{k'})$ is demonstrated as shown in Figure 3.1: the points colored in red are the discretized grid points (x_i, y_j, z_k). The grid cell, which is expressed for the numerical evaluation of the integrals in Eq.(3.5), is actually edged by the lines colored in blue around each red central grid point.

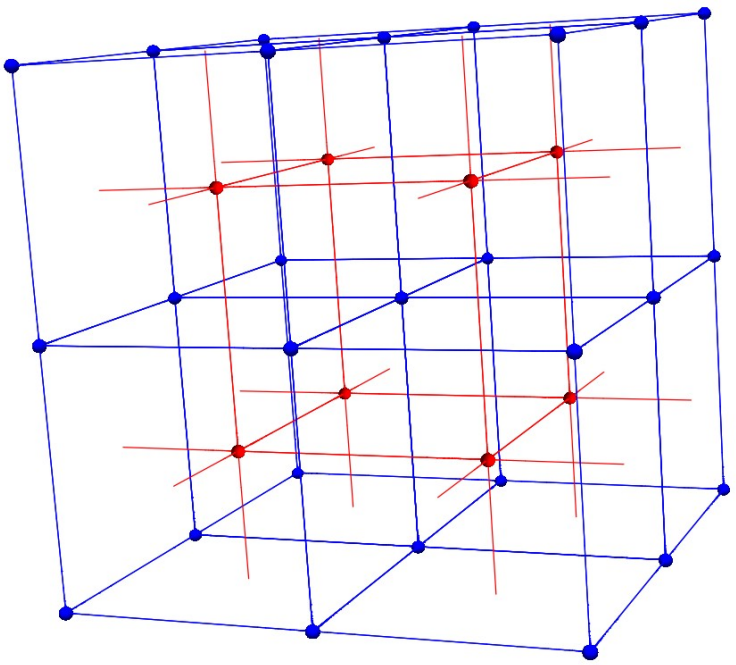


Figure 3.1: A schematic plot of the grid arrangement used for the numerical integrals in Eq. (3.4). Source: Thomas Flisgen [31].

Substitute $w - w'$ by w for w in x, y, z in Eq. (3.5),

$$\tilde{G}(x_i, y_j, z_k) = \int_{x_i-h_x/2}^{x_i+h_x/2} \int_{y_j-h_y/2}^{y_j+h_y/2} \int_{z_k-h_z/2}^{z_k+h_z/2} G(x', y', z') \mathrm{d}x' \mathrm{d}y' \mathrm{d}z' \tag{3.6}$$

Applying the midpoint rule to the numerical integral, the GF integral is obtained:

$$\tilde{G}_{\mathrm{GF}}(x_i, y_j, z_k) = h_x h_y h_z G(x_i, y_j, z_k) = \frac{h_x h_y h_z}{\sqrt{x_i^2 + y_j^2 + z_k^2}}. \tag{3.7}$$

Since the charge density ρ in Eq. (3.3) cannot be obtained as a function in practice, the results of the numerical convolution are always obtained by the approximation Eq. (3.4). For the integral of GF, the numerical midpoint integral rule of $\tilde{G}$ in Eq. (3.5) is sufficient for most applications. Theoretically, the approximation of the midpoint integral reaches an accuracy of $O(h^2)$. However, as Poisson's equation is solved after the Lorenz transform, the considered domain is stretched by the Lorenz factor γ. When the bunch of particles reaches a higher energy, the accuracy of the numerical integral Eq. (3.7) is challenged. The improvement of the numerical integral accuracy would directly focus on the accuracy improvement of $\tilde{G}$. There are methods available to improve the accuracy of the integral. First, the high order accuracy numerical integrals can be used to replace the midpoint rule, e.g. Simpson's rule, Simpson's 3/8 rule, and other types of Newton-Cotes quadrature rule. Second, the numerical integral can also be directly solved by the first fundamental theorem of calculus for high dimensions. However, the antiderivative function's form is not straightforward to express manually in practice. The next section will discuss the topic in details.

3.2 Integrated Green's function

If the primitive function of $G(x, y, z)$ is defined as $\mathrm{IGF}(x, y, z)$, the values of $\tilde{G}(x_i, y_j, z_k)$ in Eq. (3.5) can be easily calculated by summing up the eight terms of the surrounding IGF function values of (x_i, y_j, z_k), which is shown as $\tilde{G}_{\mathrm{IGF}}$:

$$\begin{aligned}
\tilde{G}_{\mathrm{IGF}}(x_i, y_j, z_k) &= \int_{x_i - h_x/2}^{x_i + h_x/2} \int_{y_j - h_y/2}^{y_j + h_y/2} \int_{z_k - h_z/2}^{z_k + h_z/2} G(x', y', z') \mathrm{d}x' \mathrm{d}y' \mathrm{d}z' \\
&= \mathrm{IGF}(x_i + \frac{h_x}{2}, y_j + \frac{h_y}{2}, z_k + \frac{h_z}{2}) - \mathrm{IGF}(x_i + \frac{h_x}{2}, y_j + \frac{h_y}{2}, z_k - \frac{h_z}{2}) \\
&- \mathrm{IGF}(x_i + \frac{h_x}{2}, y_j - \frac{h_y}{2}, z_k + \frac{h_z}{2}) - \mathrm{IGF}(x_i - \frac{h_x}{2}, y_j + \frac{h_y}{2}, z_k + \frac{h_z}{2}) \\
&+ \mathrm{IGF}(x_i - \frac{h_x}{2}, y_j - \frac{h_y}{2}, z_k + \frac{h_z}{2}) + \mathrm{IGF}(x_i + \frac{h_x}{2}, y_j - \frac{h_y}{2}, z_k - \frac{h_z}{2}) \\
&+ \mathrm{IGF}(x_i - \frac{h_x}{2}, y_j + \frac{h_y}{2}, z_k - \frac{h_z}{2}) - \mathrm{IGF}(x_i - \frac{h_x}{2}, y_j - \frac{h_y}{2}, z_k - \frac{h_z}{2}).
\end{aligned} \tag{3.8}$$

The IGF integral has definitely a better performance than the GF integral in the numerical integral calculation of (3.5). It calculates the integral precisely for each grid cell.

The primitive function (antiderivative) of Eq. (3.1) can be expressed by Wolfram *Mathematica*'s integration tool. The IGF(x, y, z), is demonstrated as:

$$
\begin{aligned}
&\mathrm{IGF}(x,y,z) \doteq \iiint \frac{1}{\sqrt{x^2+y^2+z^2}}\mathrm{d}x\mathrm{d}y\mathrm{d}z \qquad (3.9)\\
&= yz\ln\left(x+\sqrt{x^2+y^2+z^2}\right)+xz\ln\left(y+\sqrt{x^2+y^2+z^2}\right)\\
&+xy\ln\left(z+\sqrt{x^2+y^2+z^2}\right)\\
&-\frac{1}{4}iz^2\ln\left(\frac{-8ix^2+8(y-iz)z-8ix\sqrt{x^2+y^2+z^3}}{x^2(y-iz)z^2}\right)\\
&+\frac{1}{4}iz^2\ln\left(\frac{8ix^2+8(y+iz)z+8ix\sqrt{x^2+y^2+z^3}}{x^2(y+iz)z^2}\right)\\
&+\frac{1}{4}ix^2\ln\left(\frac{8\left(x^2-ixy+z\left(z+\sqrt{x^2+y^2+z^2}\right)\right)}{x^2(x-iy)z^2}\right)\\
&-\frac{1}{4}ix^2\ln\left(\frac{8\left(x^2+ixy+z\left(z+\sqrt{x^2+y^2+z^2}\right)\right)}{x^2(x+iy)z^2}\right)\\
&-\frac{1}{4}iy^2\ln\left(\frac{2xy-2i\left(y^2+z\left(z+\sqrt{x^2+y^2+z^2}\right)\right)}{y(x-iy)z^2}\right)\\
&+\frac{1}{4}iy^2\ln\left(\frac{2xy+2i\left(y^2+z\left(z+\sqrt{x^2+y^2+z^2}\right)\right)}{y(x+iy)z^2}\right).
\end{aligned}
$$

A simpler form of (3.9) from [72] is shown as follows:

$$
\begin{aligned}
&\mathrm{IGF}(x,y,z) \doteq \iiint \frac{1}{\sqrt{x^2+y^2+z^2}}\mathrm{d}x\mathrm{d}y\mathrm{d}z \qquad (3.10)\\
&= -\frac{z^2}{2}\arctan\left(\frac{xy}{z\sqrt{x^2+y^2+z^2}}\right)-\frac{y^2}{2}\arctan\left(\frac{xz}{y\sqrt{x^2+y^2+z^2}}\right)\\
&-\frac{x^2}{2}\arctan\left(\frac{yz}{x\sqrt{x^2+y^2+z^2}}\right)+yz\ln(x+\sqrt{x^2+y^2+z^2})\\
&+xz\ln(y+\sqrt{x^2+y^2+z^2})+xy\ln(z+\sqrt{x^2+y^2+z^2}).
\end{aligned}
$$

From the view of physics, the IGF and GF have the same physical meaning. Regardless of the IGF and GF, the accuracy of the potential form (3.4) is limited to the discrete charge density ρ.

3.3 Efficient integrated Green's function

3.3.1 Reduced integrated Green's function

Comparison of integrated Green's function and Green's function

The IGF and GF integral routines show significantly different results in some specific cases, i.e. where the computational domain has a large longitudinal-to-transverse ratio. This is the situation which arises when the bunch has been accelerated to a few MeV. Then the bunch size is extremely long after the Lorentz transform to the rest framework for the design of next-generation accelerators.

The simulation results differ from each other, which suggests it may be useful to conduct a comparative approach of the two GF integrals. The comparison focuses on two aspects: the aspect of accuracy, which relates to the relative difference of the numerical integrals; and the aspect of efficiency, which relates to the execution time.

Comparison of accuracy

Define the tilde GF integral error from tilde IGF integral as:

$$\delta\tilde{G}_{\mathrm{GF}}(x_i, y_j, z_k) = \tilde{G}_{\mathrm{GF}}(x_i, y_j, z_k) - \tilde{G}_{\mathrm{IGF}}(x_i, y_j, z_k),$$

comparatively define the relative error between the GF and IGF integrals as:

$$\eta_G(x_i, y_j, z_k) = \left| \frac{\delta\tilde{G}_{\mathrm{GF}}(x_i, y_j, z_k)}{\tilde{G}_{\mathrm{IGF}}(x_i, y_j, z_k)} \right|.$$

To confront the specific cases, a rectangular computational domain with a large aspect ratio $L_x : L_y : L_z = 1 : 1 : 30$ is chosen. The grid is generated in equidistant points with step sizes (h_x, h_y, h_z) with grid number $(64, 64, 64)$ in each axis. The number of grid points for GF values is $65 \times 65 \times 65 = 274,625$ (The GF grid is organized by one more point than the discrete $\rho(x_i, y_j, z_k)$ in each axis for the further fast convolution calculation).

In Figure 3.2, the local relative errors $\eta_G(x_i, y_j, z_k)$ are plotted along index k (also z, the longitudinal direction), which means each column in the Figure corresponds to one slice of index k. The slice is a 2D cutting plane of the 3D domain along z direction and identified by index k. The trend of the plot is obvious: the local relative errors decrease sharply with increasing value of k. For the first several 2D slices along k, the local relative errors vary strongly and variously. With increasing k, the errors within a slice coincide more and more, and the errors of further slices converge to zero. The strong decreasing property of GF indicates this fact principally.

On the other hand, the numerical midpoint integral's relative error can be bounded by the following error analysis as:

$$\left|\delta\tilde{G}_{\mathrm{GF}}(x_i, y_j, z_k)\right| \leq \frac{h_x^3 h_y^3 h_z^3}{24^3} K_x K_y K_z, \tag{3.11}$$

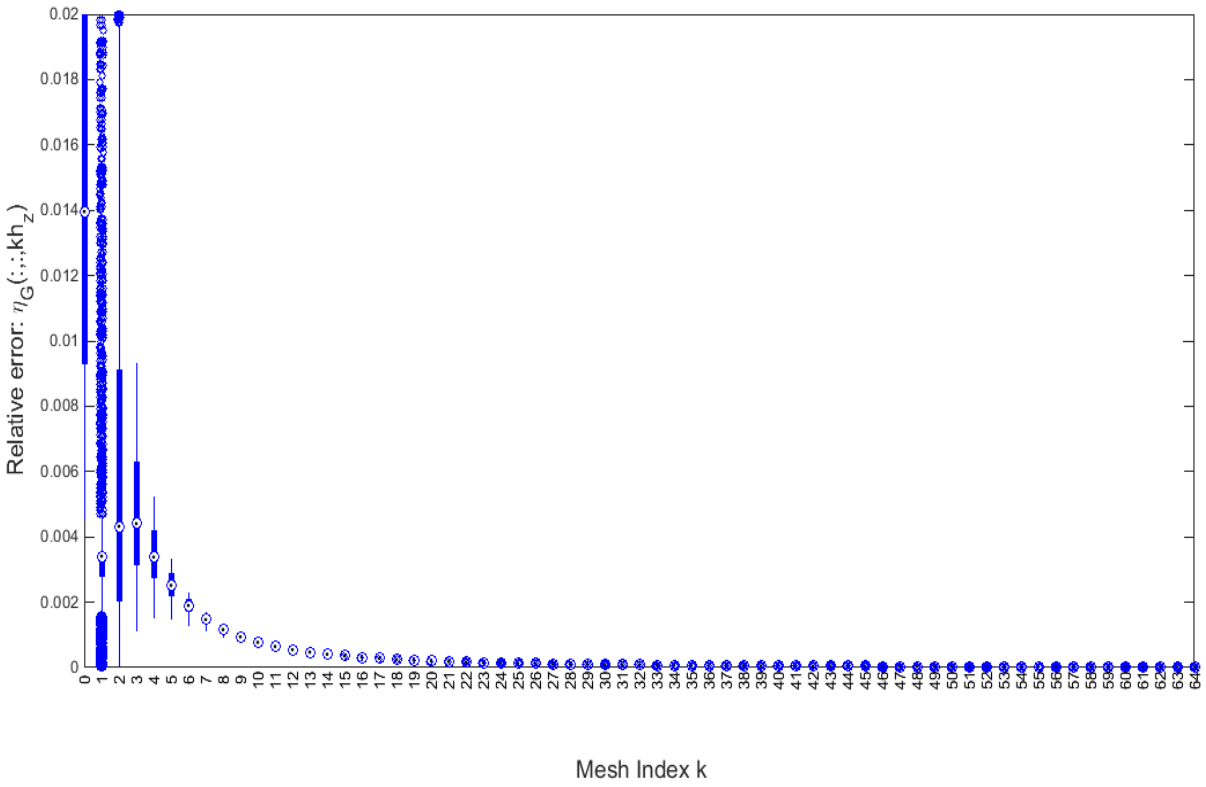


Figure 3.2: The local relative error of the GF integral.

where K_w is the largest value of $|G''_w(x,y,z)|$, and $w \in [w_l - \frac{h_w}{2}, w_l + \frac{h_w}{2}]$ for w in $\{x,y,z\}$ and l in $\{i,j,k\}$. K_w has a stronger decreasing trend than $\tilde{G}(x,y,z)$ if we check the derivatives. Aiming for a smaller $\eta_G(x_i,y_j,z_k)$, the grid number N_z along the large aspect ratio direction can be set as a large number in order to obtain a finer grid in z direction and smaller step size h_z. However, the calculation time increases proportionally with the increasing N_z. The high efficiency would be a challenge for this finer grid assignment.

The local relative error exponentially decreases with increasing distance from the origin, while the error varies also with the relative aspect ratio.

From a different point of view, the diagonal line in the first slice along k direction is considered as shown in Figure 3.3 (left). As the longitudinal-to-transverse ratio increases from 1 to 10, 30, and 100, the $\eta_G(x_i,y_j,z_k)$ values exhibit an increasing relative error along the diagonal line of the traverse plane in Figure 3.3 (right), i.e. the larger the aspect ratio is the worse the integral errors are.

Figures 3.2 and 3.3 reflect the reason why the standard GF integral solves Poisson's equation less accurately when large aspect ratios are concerned. As a global explanation for both approaches shown in the figures, the midpoint rule GF integrals near the origin point are affected by the large ratio scaling. Far from a certain distance to the origin, the numerical midpoint integral $\tilde{G}_{\mathrm{GF}}$ shows acceptable local relative

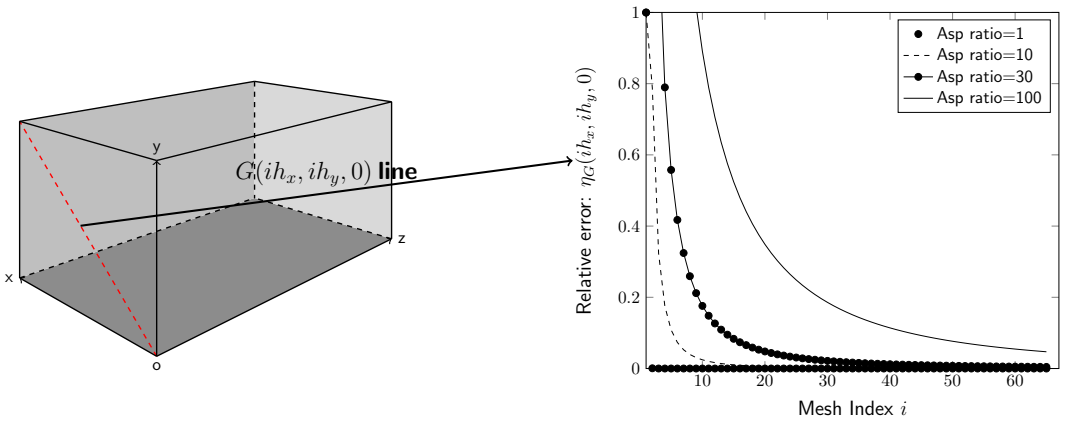


Figure 3.3: The local relative error of standard GF integrals, $\eta_G(ih_x, ih_y, 0)$ with different aspect ratios.

errors with $\tilde{G}_{\mathrm{IGF}}$.

Comparison of execution time

In comparison to $\tilde{G}(x_i, y_j, z_k)$ value calculations the IGF integral $\tilde{G}_{\mathrm{IGF}}(x_i, y_j, z_k)$ calls for eight terms in Eq.(3.8), and each term is calculated by the complicated formula in Eq.(3.10). In practice, the values from Eq.(3.10) can be computed once and stored in advance. Then the eight-terms summation in Eq.(3.8) is simply substituted by the value for each term rather than computing eight times. In contrast, the numerical GF integral $\tilde{G}_{\mathrm{GF}}(x_i, y_j, z_k)$ has only one simple term in Eq.(3.10). The complexity of the numerical GF integral $\tilde{G}_{\mathrm{GF}}(x_i, y_j, z_k)$ is much lower than the IGF integral $\tilde{G}_{\mathrm{IGF}}(x_i, y_j, z_k)$.

Table 3.1: Comparison of execution time for the two GF integrals $\tilde{G}$

N+1	$\tilde{G}_{\mathrm{GF}}$ execution Time	$\tilde{G}_{\mathrm{IGF}}$ execution Time
33	1.7576e-03 s	1.3171e-02 s
65	6.8170e-03 s	7.2431e-02 s
129	5.2252e-02 s	5.2918e-01 s
257	4.0518e-01 s	4.0992e+00 s

It can be demonstrated straightforwardly that there is an execution time difference between the two integrals. As shown in Table 3.1, the execution time[1] of IGF integrals is more than a dozen times that of the GF integrals for $N = N_x = N_y = N_z$. This

[1]The execution time for all comparisons is recorded by the average time if we do not distinguish it in this thesis.

reflects the fact that the IGF integral leads to high time-consuming terms while it is a simple term for the GF integral.

By comparing both accuracy and efficiency aspects, the midpoint rule for $\tilde{G}_{\mathrm{GF}}(x_i, y_j, z_k)$ integrals above the grid cells shows poor performance at the front parts of the large aspect ratio direction compared to $\tilde{G}_{\mathrm{IGF}}(x_i, y_j, z_k)$. This fact is determined by the distance from the origin and aspect ratio. In contrast, the execution time of $\tilde{G}_{\mathrm{IGF}}(x_i, y_j, z_k)$ integral is an order of magnitude higher than that of $\tilde{G}_{\mathrm{GF}}(x_i, y_j, z_k)$ integrals.

Reduced integrated Green's function

A numerical integral routine, which combines both the accuracy and efficiency properties from GF and IGF integrals, respectively, is an ultimate method in practice. The RIGF routine is a hybrid way to implement the plan. The trick starts by having the IGF integral inside RIGF takes the responsibility of the accuracy aspect while the GF integral inside RIGF takes charge of the efficiency aspect. Additional integer parameters (R_x, R_y, R_z) determine the separation of $\tilde{G}$ values between the IGF portion (the near-origin parts) to the GF portion (the rest parts inside the computational domain) (see Figure 3.4 blue line between Ω_{IGF} and Ω_{GF}). These (R_x, R_y, R_z) parameters scale the balance between the accuracy and efficiency demands.

Then the new RIGF integral reads as follows:

$$\tilde{G}_{\mathrm{RIGF}}(x_i, y_j, z_k) = \begin{cases} \tilde{G}_{\mathrm{IGF}}(x_i, y_j, z_k), & (0,0,0) \leq (i,j,k) < (R_x, R_y, R_z); \\ \tilde{G}_{\mathrm{GF}}(x_i, y_j, z_k), & \text{otherwise;} \end{cases} \tag{3.12}$$

Based on the two key aspects, two general strategies are carried out to choose these parameters R_w for w in $\{x, y, z\}$. For the large scaling domain size, the parameters R_w are reasonable to be limited in certain axes. For the long bunch after Lorenz transform in longitudinal direction, the R_w is not needed for w in $\{x, y\}$. To demonstrate this clearly, the next two strategies are considered in this situation.

Time determination strategy:

Defining a splitting parameter s_z which respects the ratio between IGF portion and the whole domain in z direction. R_z can be automatically defined as:

$$R_z = [\frac{N_z + 1}{s_z}]. \tag{3.13}$$

The splitting parameter s_z not only separates the space, but also scales the computational time. The final RIGF computational time t_{RIGF} is linear between the IGF computational time t_{IGF} and the GF computational time t_{GF} shown as:

$$t_{\mathrm{RIGF}} = \frac{R_z}{N_z + 1}(t_{\mathrm{IGF}} - t_{\mathrm{GF}}) + t_{\mathrm{GF}}. \tag{3.14}$$

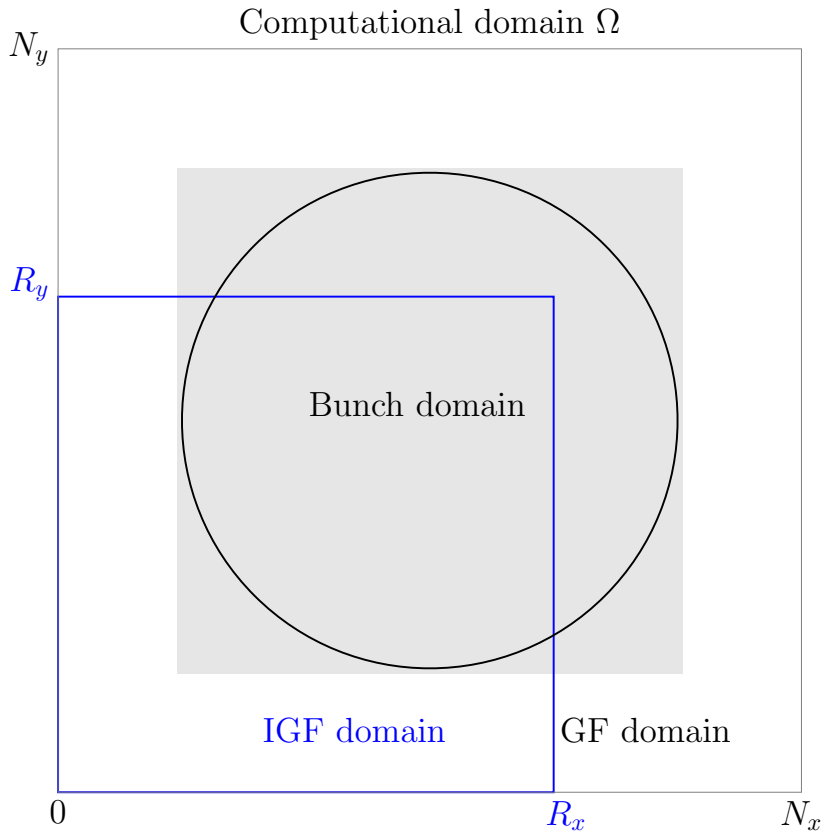


Figure 3.4: A schematic plot of RIGF domain for Cartesian coordinates.

The time determination strategy has a good estimation of execution time with the information of t_{IGF}, t_{GF}. In some cases, it could be directly used as the bunch ratio is not large.

Adapted relative error determination strategy:

The final solver's computational error would definitely be influenced by the numerical RIGF integral error which exactly matches the GF integral portion inside RIGF. The magnitude of the numerical integral's relative error is referenced to have a global estimation of the final solver's relative error in this strategy.

The strategy is demonstrated as follows:

First, $f(N_z) = 1/\log_2 N_z$ is chosen as the reference function which compares with the $\tilde{G}_{\mathrm{IGF}}$'s decreasing trend by

$$\|\tilde{G}_{\mathrm{IGF}\ k-1} - \tilde{G}_{\mathrm{IGF}\ k}\|/\tilde{G}_{\mathrm{IGF}\ k-1} < f(N_z),$$

where $\tilde{G}_{\mathrm{IGF}}$ means IGF integration is used for the $\tilde{G}$ calculation. For the determination strategy, the referenced indices (i, j, k) are chosen as $(0, 0, k)$, thus can be set as $\tilde{G}_{\mathrm{IGF}\ k}$ simply and k increases from 0.

In this step, a stationary area is determined by the inequality because the $\tilde{G}_{IGF}$ is decreasing tremendously with respect to the beginning part. $1/\log(z)$ exhibits similar behavior in practice and is a comparable function regarding the error magnitude estimation.

The first k which satisfies the inequality would be recorded as k_{stable}.

Second, the proper R_z is chosen from the k, which determines the further accuracy tolerance by

$$\delta\tilde{G}_k/\delta\tilde{G}_{k_{\text{stable}}} \leq 10^{-s},$$

where $\delta\tilde{G}_k = \|\tilde{G}_{\text{IGF }k} - \tilde{G}_{\text{GF }k}\|$, where $\tilde{G}_{\text{GF}}$ means the GF integration is used for the $\tilde{G}$ calculation. s is the magnitude determination (the accuracy control) factor, which is an integer and greater than 0. The first k that satisfies the inequality in this step is recorded as R_z.

These parameters have to be determined individually for different problems under examination. The validation for the relative error determination strategy will be discussed in Chapter 6.

3.3.2 Cutting integrated Green's function

The above GF integrals are generally used for bunch tracking simulations. This means the computation domain just covers the bunch space and only the self-field inside the domain is calculated. This kind of simulation is denoted as near-bunch domain calculation. In contrast, the external electric field outside the bunch can also be needed. The computational domain would be larger than the bunch which is known as far-bunch domain calculation. In some cases the far-bunch domain calculation is considered, e.g. the interaction between bunch and cloud simulation. The GF-kind integral methods have also a potential to make far-bunch domain simulation. Unsurprisingly, even the same bunch tracking routines in two different domain settings lead to two different results. The following efficient GF integrals of CIGF concentrate on the far-bunch domain calculation.

As presented in Figure 3.5, the bunch domain Ω_{Bunch} (surrounding cuboid) is supposed to be located at the subdomain $[x_b, x_t] \times [y_b, y_t] \times [z_b, z_t]$ of the grid (corresponding to $[N_{xb}, N_{xt}] \times [N_{yb}, N_{yt}] \times [N_{zb}, N_{zt}]$ in mesh) which is in the center of the computational domain Ω, the ratio between bunch domain and the complete computational domain is $1 : \alpha_w$ for w in x, y, z directions, respectively.

Taking the summation form of the discretized convolution in Eq. (3.4) there will be a large area in which the charge density parts $\rho(x_{i'}, y_{j'}, z_{k'})$ are zeros. That means if we use the $\tilde{G}_{\text{IGF}}$ for the far-bunch domain calculation, a portion of the $\tilde{G}_{\text{IGF}}$ values is over-calculated and, in principle, unnecessary. The form of Eq. (3.4) without zero terms is shown in Eq. (3.15). The corresponding portion of $\tilde{G}(x_i, x_{i'}, y_j, y_{j'}, z_k, z_{k'})$ is automatically waived for calculations as well.

$$\varphi(x_i, y_j, z_k) \approx \frac{1}{4\pi\varepsilon_0} \cdot \sum_{i'=N_{xb}}^{N_{xt}} \sum_{j'=N_{yb}}^{N_{yt}} \sum_{k'=N_{zb}}^{N_{zt}} \rho(x_{i'}, y_{j'}, z_{k'}) \tilde{G}(x_i, x_{i'}, y_j, y_{j'}, z_k, z_{k'}). \quad (3.15)$$

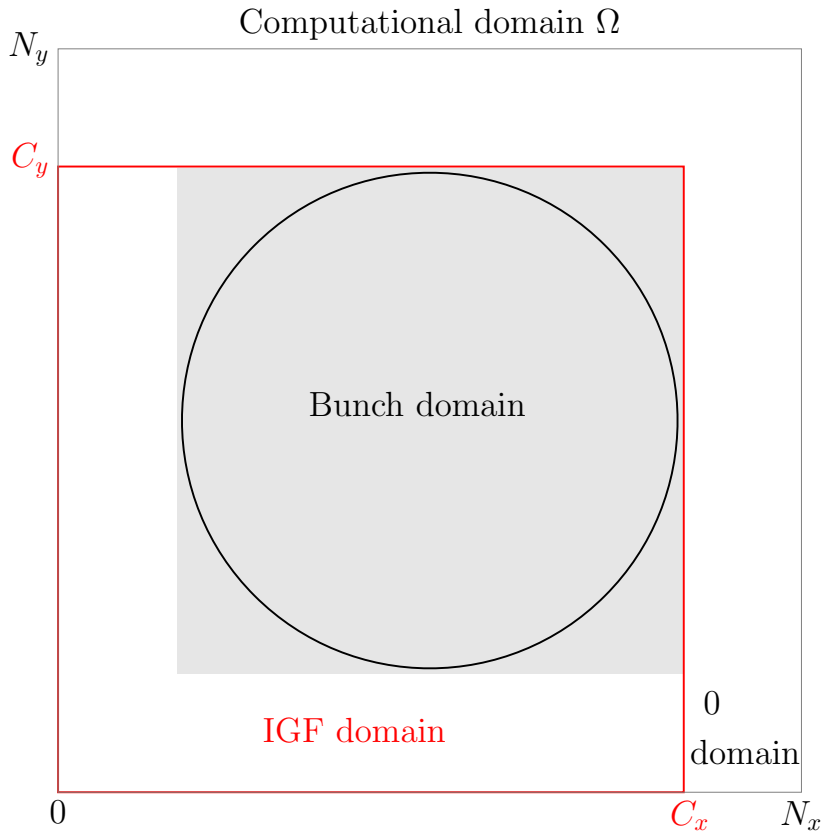


Figure 3.5: A schematic plot of CIGF domain for Cartesian coordinates.

As the $\tilde{G}$ values are practically computed in advance, the waived calculation portion of $\tilde{G}$ is actually outside of the domain bounded by the red line (Figure 3.5), but inside the computational domain. Although the calculations are waived, a value should be filled in each of these grid points for $\tilde{G}$. In practice, the values are set as zero and the domain is named as Ω_0. This is the numerical demand for the next fast Fourier convolution step in Section 3.4.

The GF value $\tilde{G}$, outside of the subdomain bounded by the red line, does not contribute to the discretized convolution at all. The red line is determined by the bunch-domain ratio, the ratio between the Green's function domain and the total computational domain, defined as $(1 + \alpha_w/2) : \alpha_w$.

Naturally, we define the CIGF as $\tilde{G}_{\mathrm{CIGF}}(x_i, y_j, z_k)$:

$$\tilde{G}_{\mathrm{CIGF}}(x_i, y_j, z_k) = \begin{cases} \tilde{G}_{\mathrm{IGF}}(x_i, y_j, z_k), & (0,0,0) \leq (i,j,k) < (C_x, C_y, C_z); \\ 0, & \text{otherwise;} \end{cases}$$

where C_w is determined by the domain-bunch ratio $\alpha_w = L_w\mathrm{Domain}/L_w\mathrm{Bunch}$ ($\alpha_w > 1$), L_w is the length for w in $\{x, y, z\}$ and $C_w = [N_w(1 + \alpha_w)/2\alpha_w]$. The large area with zero charge density in Ω_0 guarantees the CIGF is as accurate as IGF, and highly efficient in $\tilde{G}$ integrals.

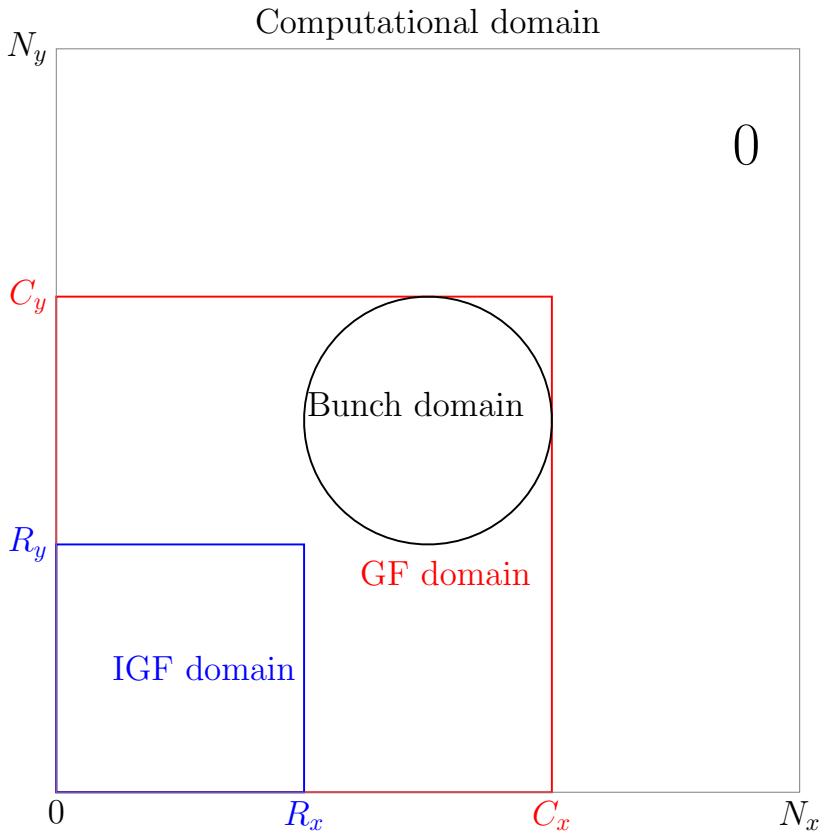


Figure 3.6: A schematic plot of CRIGF domain for Cartesian coordinates.

3.3.3 Cutting reduced integrated Green's function

For far-bunch domain space charge simulation, the CIGF integral is efficient and does not waste calculations. When the near-bunch domain simulation takes place, the CIGF is not valid anymore. However, the RIGF can always be implemented. In total, we have the following cutting reduced integrated Green's function (CRIGF) integral:

CRIGF integral

The combination of RIGF and CIGF as the CRIGF should be more efficient than pure CIGF for the same problem,

$$\tilde{G}_{\mathrm{CRIGF}}(x_i, y_j, z_k) = \begin{cases} \tilde{G}_{\mathrm{IGF}}(x_i, y_j, z_k), & (0,0,0) \leq (i,j,k) < (R_x, R_y, R_z); \\ \tilde{G}_{\mathrm{GF}}(x_i, y_j, z_k), & (R_x, R_y, R_z) \leq (i,j,k) < (C_x, C_y, C_z); \\ 0, & \text{otherwise}; \end{cases}$$

where (C_x, C_y, C_z) and (R_x, R_y, R_z) are chosen as above.

In Table 3.2, the different GF integrals are compared in the respect of executed time in practice.

Table 3.2: Comparison of different GF integrals' elapsed time with increasing grid resolution.

N+1	GF Time	IGF Time	RIGF Time	CIGF Time	CRIGF Time
33	1.7576e-03 s	1.3171e-02 s	2.5239e-03 s	2.4517e-03 s	6.2260e-04 s
65	6.8170e-03 s	7.2431e-02 s	1.3374e-02 s	1.6735e-02 s	3.5395e-03 s
129	5.2252e-02 s	5.2918e-01 s	7.8222e-02 s	1.1753e-01 s	2.4910e-02 s
257	4.0518e-01 s	4.0992e+00 s	5.3560e-01 s	9.4162e-01 s	1.9471e-01 s

The accuracy respect of different GF integrals connecting to relative errors of Poisson solvers is not shown in this chapter. The verification of RIGF integral and CIGF integral (CRIGF integral is then automatically included) will be discussed in Chapter 6.

3.4 The trivial FFT convolution routine with padding zeros

The summation of Eq. (3.4) is highly time-consuming when the complexity reaches numbers as high as $N_x^2 N_y^2 N_z^2$. However, based on the Fourier convolution theory, the discrete convolution in the space domain equals the point-wise multiplication in the Fourier domain, i.e. using 3D DFT $\mathfrak{F}$ and convolution theory, the extended potential expression is:

$$[\varphi_{ex}]_{i,j,k} = \frac{1}{4\pi\varepsilon_0}\mathfrak{F}^{-1}\{[\mathfrak{F}\tilde{G}_{ex}]_{i,j,k} \cdot [\mathfrak{F}\rho_{ex}]_{i,j,k}\}_{2N_x,2N_y,2N_z}. \tag{3.16}$$

Here, the domain ω is doubled in each axis, i.e. the computational domain should be $2N_x \times 2N_y \times 2N_z$ rather than the original $N_x \times N_y \times N_z$ due to the implementation of a cyclic convolution. The cyclic convolution is a result from the naturally periodic property of DFT. A direct DFT will force both the charge density $\rho(x_i, y_j, z_k)$ and the tilde GF $\tilde{G}(x_i, y_j, z_k)$ to be periodic. The obtained solution would be meaningless, since neither $\rho(x_i, y_j, z_k)$ nor $\tilde{G}(x_i, y_j, z_k)$ is periodic in reality. The $N_x \times N_y \times N_z$ size results in the following problem: if we perform the cyclic convolution directly on $\tilde{G}(x_i, y_j, z_k)$ and $\rho(x_i, y_j, z_k)$ of the original $N_x \times N_y \times N_z$ domain, wrong $\tilde{G}(x_i, y_j, z_k)$, $\rho(x_i, y_j, z_k)$ values will automatically be defined as periodic functions along each axis. To overcome these issues, both $\tilde{G}(x_i, y_j, z_k)$ and $\rho(x_i, y_j, z_k)$ are extended as $\tilde{G}_{ex}(x_i, y_j, z_k)$ and $\rho_{ex}(x_i, y_j, z_k)$. The extended potential is denoted as $\phi_{ex}(x_i, y_j, z_k)$.

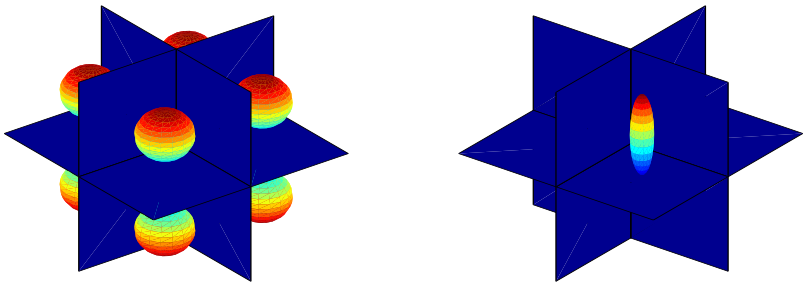


Figure 3.7: The extension of GF (left side) and the extension of charge density (right side).

For the extensions: the charge density $\rho_{ex}(x_i, y_j, z_k)$ is padded with zeros in all extension grid points, the tilde GF $\tilde{G}(x_i, y_j, z_k)$ is extended to be symmetric. The details are demonstrated in the following routine and Figure 3.7. The original potential at each grid point equals the first N_x, N_y, N_z values of the extended potential expression on each axis.

As the cyclic convolution is executed through DFT, the computation complexity can be reduced due to FFT's implementation. However, the double-sized extension of data in each axis can not be avoided.

Routine: the trivial FFT convolution routine

1. Green's function calculation and extension (Left side of Figure 3.7):

1.1. Calculate each tilde GF value $\tilde{G}(x_i, y_j, z_k)$ by using the formula from either GF, IGF, or CRIGF integrals.

1.2. Extend the tilde GF values as:

$$\tilde{G}_{ex}(x_i, y_j, z_k) =$$
$$\begin{cases} \tilde{G}(x_i, y_j, z_k), & 0 \le i \le N_x; 0 \le j \le N_y; 0 \le k \le N_z, \\ \tilde{G}(x_{2N_x-i}, y_j, z_k), & N_x < i \le 2N_x - 1; 0 \le j \le N_y; 0 \le k \le N_z, \\ \tilde{G}(x_i, y_{2N_y-j}, z_k), & 0 \le i \le N_x; N_y < j \le 2N_y - 1; 0 \le k \le N_z, \\ \tilde{G}(x_i, y_j, z_{2N_z-k}), & 0 \le i \le N_x; 0 \le j \le N_y; N_z < k \le 2N_z - 1, \\ \tilde{G}(x_{2N_x-i}, y_{2N_y-j}, z_k), & N_x < i \le 2N_x - 1; N_y < j \le 2N_y - 1; 0 \le k \le N_z, \\ \tilde{G}(x_{2N_x-i}, y_j, z_{2N_z-k}), & N_x < i \le 2N_x - 1; 0 \le j \le N_y; N_z \le k \le 2N_z - 1, \\ \tilde{G}(x_i, y_{2N_y-j}, z_{2N_z-k}), & 0 \le i \le N_x; N_y < j \le 2N_y - 1; N_z \le k \le 2N_z - 1, \\ \tilde{G}(x_{2N_x-i}, y_{2N_y-j}, z_{2N_z-k}), & N_x < i \le 2N_x - 1; N_y < j \le 2N_y - 1; N_z < k \le 2N_z - 1, \end{cases}$$

2. Extend the charge density $\rho(x_i, y_j, z_k)$ (right side of Figure 3.7):

$$\rho_{ex}(x_i, y_j, z_k) = \begin{cases} \rho(x_i, y_j, z_k), & 0 \le i \le N_x - 1; 0 \le j \le N_y - 1; 0 \le k \le N_z - 1, \\ 0, & N_x \le i \le 2N_x - 1; 0 \le j \le N_y - 1; 0 \le k \le N_z - 1, \\ 0, & 0 \le i \le N_x - 1; N_y \le j \le 2N_y - 1; 0 \le k \le N_z - 1, \\ 0, & 0 \le i \le N_x - 1; 0 \le j \le N_y - 1; N_z \le k \le 2N_z - 1, \\ 0, & N_x \le i \le 2N_x - 1; N_y \le j \le 2N_y - 1; 0 \le k \le N_z - 1, \\ 0, & N_x \le i \le 2N_x - 1; 0 \le j \le N_y - 1; N_z \le k \le 2N_z - 1, \\ 0, & 0 \le i \le N_x - 1; N_y \le j \le 2N_y - 1; N_z \le k \le 2N_z - 1, \\ 0, & N_x \le i \le 2N_x - 1; N_y \le j \le 2N_y - 1; N_z \le k \le 2N_z - 1, \end{cases}$$

3. Obtain $\mathfrak{F}\tilde{G}_{ex}(x_i, y_j, z_k)$: 3D FFT of $\tilde{G}_{ex}(x_i, y_j, z_k)$.

4. Obtain $\mathfrak{F}\rho_{ex}(x_i, y_j, z_k)$: 3D FFT of $\tilde{\rho}_{ex}(x_i, y_j, z_k)$.

5. Multiply $\mathfrak{F}\rho_{ex}(x_i, y_j, z_k)$ and $\mathfrak{F}\tilde{G}_{ex}(x_i, y_j, z_k)$ by corresponding indices. The results are stored to $\mathfrak{F}\rho_{ex}(x_i, y_j, z_k)$.

6. Obtain the potential $\phi_{ex}(x_i, y_j, z_k)$: 3D inverse FFT of $\mathfrak{F}\rho_{ex}(x_i, y_j, z_k)$.

7. The potential $\phi(x_i, y_j, z_k)$ is obtained by cutting the original domain Ω from the extended domain of $\phi_{ex}(x_i, y_j, z_k)$.

The trivial FFT convolution routine is straightforwardly obtained by the Fourier convolution theorem. For implementation, the routine is still not effective or efficient enough. In the next chapter, this trivial routine will be further improved with respect to less storage requirement and less time consumption. The historic line will be linked for a clear demonstration.

In some simulation applications this trivial routine is still in use. For example for the GPU version simulation [24] [77] which is due to the simple programming, and highly optimized FFT library provided by hardware vendors.

4 A novel discrete convolution with implicitly zero-padded FFT

Since FFT accelerates the numerical computation within an acceptable amount of time, Fourier transform has been widely used for convolution computations. The trivial FFT convolution routine, as shown in Section 3.4, is the basic and the simplest routine for the numerical convolution implementation. However, this routine treats the extensions and the multidimensional DFT in a rough way. Many unnecessary operations are counted inside the computation, which means that potential efficiency improvements are available to perform. Also, the Poisson solver is executed for tens of thousands of times, so that a few percentage points of execution time equal a few hours in real time. Efficient numerical methods are needed and the pursuit of that is endless.

In Section 4.1, multidimensional DFT and FFT are represented. They are essential for the optimization of a convolution routine for either explicitly zero-padded FFT, as done by Hockney and Eastwood, or the implicitly zero-padded FFT as used in the novel discrete convolution.

In Section 4.2, the classical method by Hockney and Eastwood, which has been widely used for a couple of decades, is reviewed.

Furthermore, the novel efficient convolution routine is presented in Section 4.3 for 1D, 2D, and 3D situations. Both the extension and DFT are considered for $\tilde{G}$'s optimization. On the other hand, the implicitly zero-padded FFTs for ρ are implied.

The combination of real to complex FFTs with the novel discrete convolution is considered for further efficiency improvement in Section 4.4.

Finally, the error study of the novel numerical convolution is given in Section 4.5. A detailed numerical convergence approach is presented in Section 6.3.1 in Chapter 6.

Some of the results described in this chapter have already been published in own publication [101].

4.1 Multidimensional discrete Fourier transform

DFT is probably the most widely used algorithm in both scientific and engineering fields. However, it was ignored for a long time due to the high complexity of the matrix-vector multiplication operation. As early as 1805, Gauss had already presented the algorithm in his works. The formulas published by Gauss are expressed as well as the forms almost a hundred years later [41]. After 1965, when Cooley and Tukey published their global renowned article [23] on FFT, computational tools based on FFT spread over most areas of the scientific world, including the space charge calculation in beam dynamics.

Before introducing further optimization steps for the convolution routine in Section 3.4, the basic ideas behind FFT and multidimensional FFT routines have to be explained.

4.1.1 The discrete Fourier transform

The DFT of a n-sized vector $f = [f_0, f_1, f_2, \ldots, f_{n-1}]^T$ is, in principle, a matrix-vector product on the vector, i.e.

$$\hat{f} = \mathrm{DFT}_n(f).$$

Or the full-matrix form:

$$\begin{pmatrix} \hat{f}_0 \\ \hat{f}_1 \\ \hat{f}_2 \\ \vdots \\ \hat{f}_k \\ \vdots \\ \hat{f}_{n-1} \end{pmatrix} = \begin{pmatrix} \omega_n^{0\cdot 0} & \omega_n^{0\cdot 1} & \omega_n^{0\cdot 2} & \cdots & \omega_n^{0\cdot l} & \cdots & \omega_n^{0\cdot n-1} \\ \omega_n^{1\cdot 0} & \omega_n^{1\cdot 1} & \omega_n^{1\cdot 2} & \cdots & \omega_n^{1\cdot l} & \cdots & \omega_n^{1\cdot n-1} \\ \omega_n^{2\cdot 0} & \omega_n^{2\cdot 1} & \omega_n^{2\cdot 2} & \cdots & \omega_n^{2\cdot l} & \cdots & \omega_n^{2\cdot n-1} \\ \vdots & \vdots & \vdots & & \vdots & & \vdots \\ \omega_n^{k\cdot 0} & \omega_n^{k\cdot 1} & \omega_n^{k\cdot 2} & \cdots & \omega_n^{k\cdot l} & \cdots & \omega_n^{k\cdot n-1} \\ \vdots & \vdots & \vdots & & \vdots & & \vdots \\ \omega_n^{n-1\cdot 0} & \omega_n^{n-1\cdot 1} & \omega_n^{n-1\cdot 2} & \cdots & \omega_n^{n-1\cdot l} & \cdots & \omega_n^{n-1\cdot n-1} \end{pmatrix} \begin{pmatrix} f_0 \\ f_1 \\ f_2 \\ \vdots \\ f_l \\ \vdots \\ f_{n-1} \end{pmatrix}$$

whereby

$$\omega_n = \cos\left(\frac{2\pi}{n}\right) - \mathrm{i}\sin\left(\frac{2\pi}{n}\right) = \exp\left(-\frac{2\pi}{n}\mathrm{i}\right),$$

and $\omega_n^n = 1$ which means ω_n is an nth root of unity.

The one-line form of DFT is prescribed as

$$\hat{f}(\omega_n^k) = \sum_{l=0}^{n-1} \omega_n^{kl} f_l, \text{ for } k = 0, 1, 2 \ldots, n-1. \tag{4.1}$$

It is easy to confirm that the DFT matrix $\mathfrak{F}$ is symmetric, and the inverse procedure of DFT, known as IDFT, $\mathfrak{F}^{-1}$ equals to the conjugate transpose of $\mathfrak{F}$ regardless a constant factor as:

$$\mathfrak{F}_n^{-1} = \frac{1}{n}\mathfrak{F}_n^*.$$

The corresponding one-line form of IDFT is obtained as:

$$f_l = \frac{1}{n}\sum_{k=0}^{n-1} \bar{\omega}_n^{lk} \hat{f}_k, \text{ for } l = 0, 1, 2 \ldots, n-1. \tag{4.2}$$

where the $\bar{\omega}_n$ is the complex conjugate value of ω_n.

4.1.2 The fast Fourier transform

To divide a n-sized DFT into a tree of smaller and smaller DFTs is the general idea of Cooley-Tukey's FFT algorithm. As an instruction, the following theorem summarizes the basic idea of FFT in the radix-2 splitting situation. The routine is continued until the new ns are primes (2 for this case).

Theorem 1 (Radix-2 splitting)**.** *If $n = 2m$ and*

$$\Omega_m = diag(1, \omega_n, \ldots, \omega_n^{m-1}),$$

then

$$\mathfrak{F}_n \Pi_n = \begin{bmatrix} \mathfrak{F}_m & \Omega_m \mathfrak{F}_m \\ \mathfrak{F}_m & -\Omega_m \mathfrak{F}_m \end{bmatrix} = \begin{bmatrix} I_m & \Omega_m \\ I_m & -\Omega_m \end{bmatrix} \begin{bmatrix} \mathfrak{F}_m & 0 \\ 0 & \mathfrak{F}_m \end{bmatrix},$$

where $\omega_n = \exp(-2\pi i/n)$ and Π_n is the permutation matrix, which groups the even-indexed columns of $\mathfrak{F}_m$ firstly and odd-indexed columns afterwards.

The proof of this theorem is provided in [87] (page 12). The key aspect of the proof is substituting the following formulas Eq. (4.3) into the Fourier matrix and simplifying it.

$$\omega_n^2 = \omega_m, \text{ and } w_n^m = -1 \tag{4.3}$$

In many cases, n cannot be divided by 2, but by other primes. Comparably, the radix splitting in other primes is still successful for FFT, which can be found in [87] (chapter 2).

Computer science assumes a major role inside FFT's widely used applications. Several software packages are specified for the implementation of FFT, e.g. Fastest Fourier Transform in the West (FFTW) package developed by Matteo Frigo and Steven G. Johson at MIT, and FFTPACK by Paul N. Swarztrauber at National Center for Atmospheric Research, US. The FFT's efficient implementation is so important that the lower bound on the complexity of FFT algorithm has been questioned and is still unsolved in the theoretical aspect [48], while different computing architectures are carried out and optimized for the FFT's speed-up in comparison with the common CPU's implementation, e.g. the implementation of FFT in GPU platform.

4.1.3 The multidimensional DFTs and FFTs

As the 3D problem is a typical situation in beam dynamics simulation, we choose the 3D DFT as an instruction. The DFT can also be denoted as FFT in some cases when we do not need to distinguish the two in this dissertation.

The multidimensional FFTs are obtained by serially implementing the 1D FFT along all directions as performed in Algorithm 1. However, there are other issues which need to be considered, e.g. the transportation within 3D vectors and memory traffic problems.

Algorithm 1 3D FFT

Input: $f(0:N_x-1, 0:N_y-1, 0:N_z-1)$
Output: $f(0:N_x-1, 0:N_y-1, 0:N_z-1)$

```
1: function 3D DFT
2:     for j = 0 → N_y − 1 do
3:         for k = 0 → N_z − 1 do
4:             f(:, j, k) ← FFT[f(:, j, k)]
5:         end for
6:     end for
7:     for i = 0 → N_x − 1 do
8:         for k = 0 → N_z − 1 do
9:             f(i, :, k) ← FFT[f(i, :, k)]
10:        end for
11:    end for
12:    for i = 0 → N_x − 1 do
13:        for j = 0 → N_y − 1 do
14:            f(i, j, :) ← FFT[f(i, j, :)]
15:        end for
16:    end for
17: end function
```

In Section 3.4, the 3D FFT of extended GF data and ρ are briefly obtained as Algorithm 1. However, the FFT size with the extended vectors is exponentially larger than the original problem's size. So it has to be reluctantly accepted that the full 3D FFTs for both $\tilde{G}_{ex}$, and ρ_{ex} are applied in the routine, especially when only the potential in the original domain is needed. As a start of the optimization, the 3D FFT of ρ_{ex} can be optimized due to the property of the zero-padded extension. Hockney and Eastwood have presented the algorithm in their book [44], and it is also presented here in Section 4.2.

The multidimensional DFTs are sometimes presented via the Kronecker product of matrix and vectorization of multidimensional arrays for analysis purpose and instructions. ([87] section 3.4).

4.2 The commonly used convolution routines

4.2.1 The routine from Hockney and Eastwood

The fast convolution routine by Hockney and Eastwood has been used widely for a couple of decades. In their routine, a full structure of 3D FFT is organized to be pruned. The pruned 3D FFT succeeds because seven eighths of ρ_{ex} are zeros after padding zeros for the charge density ρ. It is implemented based on the idea of splitting the zero portions in the original structure of 3D FFT. The detailed algorithm is demonstrated in Algorithm 2.

In addition, the saving of memory storage in the routine is planned as well. The storage required in Hockney and Eastwood's routine is for two active data $2N_x \times N_y \times N_z$ (f_{ex} in Algorithm 2) and $2N_x \times 2N_y \times 2N_z$ ($3DFFTGreenFunExt$ in Algorithm 2), plus a temporary plane of $2N_y \times N_z$ points ($Temp2D$ in Algorithm 2), and a temporary vector of $2N_z$ points ($Temp1D$ in Algorithm 2). The storage savings are competitive to the trivial extended ρ_{ex}, which is eight-fold of ρ.

The routine includes two major types of functions: $PaddingZeroInW[\,]$ for W in $\{X, Y, Z\}$, and 1D $FFT[\,]$ ($IFFT[\,]$) functions in each direction.

As an instruction for $PaddingZeroInW[\,]$ in 1D: if a vector $f = [f_0, f_1, f_2, \ldots, f_{n-1}]^T$, the $PaddingZero$ of f is

$$f = [f_0, f_1, f_2, \ldots, f_{n-1}, 0, 0, 0, \ldots, 0]_{2n}^T,$$

where $[\,]_{2n}^T$ means the transpose of a $2n$-sized vector. All three usages of padding zero functions can be understood as the above instruction in different directions.

For the involved $FFT[\,]$, the implementation can be effected by the *for*-loops. Based on the storage settings, the numbers of batch FFTs in each direction are different: $(N_x - 1) \times (N_y - 1)$ FFTs in x axis, $N_z - 1$ FFTs in y axis, and 1 FFT in z axis. The storage saving of a temporary vector of $2N_z$ points for the z direction is not worthwhile compared to the FFT efficiency in the z direction in most cases.

From a modern perspective of efficient FFT calculations, the FFT parts for the z direction in Algorithm 2's routine can be modified to be implemented in batch or parallel. Nowadays, the batch FFTs are fully optimized by different kinds of FFT packages. The FFTW package provides both batch plans and shared-memory parallel ways for efficient FFT implementation. Similarly, the CUDA FFT library contributes another efficient FFT implementation for GPU implementation. Moreover, both strategies based on the batch FFTs are efficient in practice, which is well included in software packages. This realization differs from the one Hockney and Eastwood introduced in their work a couple of decades ago.

The Hockney and Eastwood routine waives the FFTs of zero vectors inside Eq. (3.16) of the Fourier convolution theory. A reasonable efficiency improvement is achieved. However, Hockney and Eastwood do not optimize the FFT for the GF vector, even though they notice the symmetric property after the 3D FFT because they use the symmetric property for the purpose of storage savings.

Algorithm 2 Fast 3D Convolution by Hockney and Eastwood

Input: $f(0:N_x-1,0:N_y-1,0:N_z-1)$, $3DFFTGreenFunExt(0:2N_x-1,0:2N_y-1,0:2N_z-1)$
Output: $u(0:N_x-1,0:N_y-1,0:N_z-1)$

1: **function** CONVOLUTION3D_HANDE
2: $f_{ex}(0:2N_x-1,0:N_y-1,0:N_z-1) \leftarrow PaddingZeroInX[f(0:N_x-1,0:N_y-1,0:N_z-1)]$
3: **for** $j=0 \rightarrow N_y-1$ **do**
4: **for** $k=0 \rightarrow N_z-1$ **do**
5: $f_{ex}(:,j,k) \leftarrow FFT[f_{ex}(:,j,k)]$
6: **end for**
7: **end for**
8: **for** $i=0 \rightarrow 2N_x-1$ **do**
9: $Temp2D(0:2N_y-1,0:N_z-1) \leftarrow PaddingZeroInY[f_{ex}(i,0:N_y-1,0:N_z-1)]$
10: **for** $k=0 \rightarrow N_z-1$ **do**
11: $Temp2D(:,k) \leftarrow FFT[Temp2D(:,k)]$
12: **end for**
13: **for** $j=0 \rightarrow 2N_y-1$ **do**
14: $Temp1D(0:2N_z-1) \leftarrow PaddingZeroInZ[Temp2D(j,0:N_z-1)]$
15: $Temp1D \leftarrow FFT[Temp1D]$
16: $Temp1D \leftarrow Temp1D(:) * 3DFFTGreenFunExt(i,j,:)$
17: $Temp1D \leftarrow IFFT[Temp1D]$
18: $Temp2D(j,0:N_z-1) \leftarrow Temp1D(0:N_z-1)$
19: **end for**
20: **for** $k=0 \rightarrow N_z-1$ **do**
21: $Temp2D(:,k) \leftarrow IFFT[Temp2D(:,k)]$
22: **end for**
23: $f_{ex}(i,:,k) \leftarrow Temp2D(0:Ny,k)$
24: **end for**
25: **for** $j=0 \rightarrow N_y-1$ **do**
26: **for** $k=0 \rightarrow N_z-1$ **do**
27: $f_{ex}(:,j,k) \leftarrow IFFT[f_{ex}(:,j,k)]$
28: **end for**
29: **end for**
30: $u(0:N_x-1,0:N_y-1,0:N_z-1) \leftarrow f_{ex}(0:N_x-1,:,:)$
31: **end function**

In the next section, the efficiency optimization of FFT will be further studied for both ρ_{ex} and $\tilde{G}_{ex}$.

4.3 A novel fast 3D convolution routine without explicit zero padding

Recently, Bowmann and Roberts have shown an efficient dealiased convolution algorithm without the expense of conventional zero padding [12]. He uses a routine to calculate the implicitly zero-padded FFT convolution rather than the explicitly zero-padded FFT convolution. The core finding is to leave out a bit reversal stage in FFT in order to reduce both the time and the storage consumption. In general, the speed-up is as high as 2 compared to the explicitly zero-padded pruned-FFT convolution. The precision of the implicitly zero-padded FFT convolution is identical to that of the explicitly zero-padded convolution. This kind of convolution is used in the study of turbulence. However, the input data's extension and convolution in Bowmann's study is different from the convolution we mentioned in the Poisson solver for space charge calculation. This is because the zero-padded $\tilde{G}_{ex}$ is replaced by a different extension, defined in Def.1.

Definition 1. *Suppose a vector $g = [g_0, g_1, \ldots, g_n]^T$, the real even symmetric extension (RESE) of g is given as g_{ex} which is expressed as:*

$$g_{ex} = [g_0, g_1, \ldots, g_n, g_{n-1}, g_{n-2}, \ldots, g_1]^T_{2n}.$$

The RESE of a multidimensional vector is obtained by applying 1D RESE in each dimension.

The goal to produce the convolution without the last bit reversal stage must be developed differently in our convolution. The efficiency improvement needs to be tested for our new routine and the novel fast convolution routine will be compared with the classical routine from Hockney and Eastwood.

4.3.1 Optimization of the 3D FFT with extension for GF values

The optimization of the FFT for $\tilde{G}_{ex}$ starts with a real to real Fourier transform: (type-I) discrete cosine transform (DCT) and the specific symmetric extension of $\tilde{G}$. There are eight different types of DCTs in total. In this dissertation, DCT refers to the first type if not specific. Elsewhere, type-II discrete cosine transform (DCT-II) is used in the field of signal and image processing, for example, but this is usually a small size transform.

Definition 2. *Suppose a vector $g = [g_0, g_1, \ldots, g_n]^T$, define DCT of g as y, which is expressed by:*

$$\begin{aligned} &y = DCT_n(g), \ \textit{i.e.} \\ &y_k = \frac{g_0}{2} + \sum_{j=1}^{m-1} \cos(\frac{kj\pi}{m}) g_j + \frac{(-1)^k g_m}{2}, \ \textit{for } k = 0, 1, \ldots, n. \end{aligned} \tag{4.4}$$

The DCT has a strong connection with the FFT of roughly double length with RESE. Theorem 2 presents the connection in 1D form as follows:

Theorem 2. *Suppose a vector $g = [g_0, g_1, \dots g_n]^T$, the RESE of g is g_{ex}. If we define the vectors*

$$y_{ex} = \frac{1}{2} DFT_{2n}(g_{ex}),$$

and

$$y = DCT_n(g),$$

then y_{ex} is the RESE of y.

The DCT can be achieved by applying the same size of the FFT with some additional operations, as introduced in [87] (section 4.4). The complexity is on the same level as FFT, $2.5n \log n$. The idea behind the fast computation is also due to the RESE property. In total, Theorem 2 shows us the capability to reduce the "double-sized" $2n$ FFT with RESE by exchanging the RESE after the DCT. Regardless of the computation of RESE, the complexity reduces from $5n \log 2n$ to $2.5n \log n$.

The 3D DCT is achieved similar to the routine of the 3D FFT by implementing 1D DCT along each dimension for x, y, z directions. The conclusion for 3D, Theorem 3, is similar to the 1D Theorem:

Theorem 3. *Suppose a 3D vector g, and the 3D RESE of g is g_{ex}. If the 3D vectors y_{ex} and y,*

$$y_{ex} = \frac{1}{8} \mathfrak{F} g_{ex}$$

and

$$y = \mathfrak{D}(g),$$

where $\mathfrak{F}$ is the 3D Fourier transform and $\mathfrak{D}$ is the 3D DCT, then y_{ex} is the 3D RESE of y.

Or:

Corollary 1. *The vector obtained by the DFT of a RESE vector is RESE.*

For the further usage of $\mathfrak{F}\tilde{G}_{ex}$, the extension can be expressed implicitly from $\mathfrak{D}(G)$ with real even symmetry, rather than with extension in storage. The corresponding index is as shown:

$$\mathfrak{F}\tilde{G}_{ex}(i,j,k) = \mathfrak{D}(\tilde{G})(I,J,K), \tag{4.5}$$

where

$$\begin{aligned}
I &= \begin{cases} i, & i \in [0, N_x] \\ 2N_x - i & i \in [N_x + 1, 2N_x - 1] \end{cases}, \\
J &= \begin{cases} j, & j \in [0, N_y] \\ 2N_y - j & j \in [N_y + 1, 2N_y - 1] \end{cases}, \\
K &= \begin{cases} k, & k \in [0, N_z] \\ 2N_z - k & k \in [N_z + 1, 2N_z - 1] \end{cases}.
\end{aligned} \tag{4.6}$$

Some more conclusions regarding to RESE, DCT, (type-I) inverse discrete cosine transform (IDCT), DFT and IDFT are shown in Theorem 4 and Corollary 2.

Theorem 4. *Apart from a constant factor, DCT is the same as IDCT.*

Corollary 2. *Apart from a constant factor, DFT and IDFT of real-even symmetric vectors are the same.*

Resulting from Corollary 2, we notice that both DFT and IDFT for $\tilde{G}_{ex}$ can be used for Hockney and Eastwood (HandE)'s routine, apart from a constant factor.

Optimization results of the 3D FFT with RESE for GF values:

Based on Theorem 2 and 3, the improvement of the 3D DCT with implicit RESE for GF values is significant for both storage and efficiency. As the explicit RESE of $\tilde{G}$ values is waived, the storage stays the same size as $\tilde{G}$ rather than increasing to roughly 8 times its size. Secondly, the 3D $2N_x \times 2N_y \times 2N_z$ FFT is replaced by 3D $(N_x+1) \times (N_y+1) \times (N_z+1)$ DCT. The improvement of complexity and efficiency is apparent: the complexity of DCT is $2.5n \log n$ while the complexity of FFT is $5n \log 2n$ for the 1D problem. In contrast, the complexity and the efficiency improvements of DCT versus FFT for the 3D problem are exponential. A direct comparison of CPU elapsed time can be found in Table 4.1.

Table 4.1: Comparison of the elapsed time of 3D DCT and FFT transforms for GFs with increasing grids resolution.

N+1	3D DCT	$2N$	3D FFT
33	9.9720e-04 s	64	2.3370e-03 s
65	3.6033e-03 s	128	3.7920e-02 s
129	1.4419e-02 s	256	1.4211e+00 s
257	6.2829e-02 s	512	1.4005e+01 s

4.3.2 Implicitly zero-padded convolution in 1D

Suppose m-sized $\hat{f}_{ex}$ is obtained by padding with zeros to n-sized $\hat{f}$, where $m = 2n$. The backward DFT of $\hat{f}_{ex}$ reads as:

$$f_{ex}(k) = \sum_{l=0}^{m-1} \exp(i\frac{2\pi}{m}kl)\hat{f}_{ex}(l) = \sum_{l=0}^{m-1} \bar{\omega}_m^{kl}\hat{f}_{ex}(l), \text{ for } k = 0, 1, 2, \ldots, m-1, \quad (4.7)$$

where ω_m is defined by $\exp(-2\pi \mathrm{i}/m)$.

As $\hat{f}_{ex}(l)$ has the zero portion structure, i.e. $\hat{f}_{ex}(l) = 0$ with $l = n, \cdots, m-1$, the multiply operations are waived due to the fact that any number multiplied by zero equals zero.

By separation between even and odd indices of f_{ex}, and consideration of the properties of Fourier coefficients: $\bar{\omega}_{2n}^{2k} = \bar{\omega}_n^k$ and $\bar{\omega}_m^m = 1$:

$$\begin{aligned} f_{ex}(2k) &= \sum_{l=0}^{n-1} \bar{\omega}_{2n}^{2kl} \hat{f}_{ex}(l) = \sum_{l=0}^{n-1} \bar{\omega}_n^k \hat{f}_{ex}(l) = \sum_{l=0}^{n-1} \bar{\omega}_n^k \hat{f}(l), \\ f_{ex}(2k+1) &= \sum_{l=0}^{n-1} \bar{\omega}_{2n}^{(2k+1)l} \hat{f}_{ex}(l) = \sum_{l=0}^{n-1} \bar{\omega}_n^k \bar{\omega}_{2n}^l \hat{f}_{ex}(l) = \sum_{l=0}^{n-1} \bar{\omega}_n^k \left(\bar{\omega}_{2n}^l \hat{f}(l) \right), \\ & \quad \text{for } k = 0, 1, 2 \ldots n-1. \end{aligned} \tag{4.8}$$

From Eq. (4.8), we achieve a backward FFT with padding zeros implicitly, the notation of this new type FFT is called fftpadBackward. Algorithm 3 shows the corresponding routine for implementation.

Algorithm 3 fftpadBackward

Input: $f(0 : n-1)$
Output: $f(0 : n-1)$, $u(0 : n-1)$

```
1: function fftpadBackward
2:     for l = 0 → n − 1 do
3:         u(l) ← ω̄_{2n}^l f(l);
4:     end for
5:     f(:) ← IFFT[f(:)];
6:     u(:) ← IFFT[u(:)];
7: end function
```

In contrast, the scaled forward DFT of f_{ex} can be proceeded inversely: starting with the original DFT of f_{ex},

$$\hat{f}_{ex}(l) = \frac{1}{2n} \sum_{k=0}^{m-1} \omega_m^{lk} f_{ex}(k), \text{ for } l = 0, 1, 2 \ldots, m-1. \tag{4.9}$$

The summation is split into even and odd index divisions of k and substituted m by $2n$:

$$\hat{f}_{ex}(l) = \frac{1}{2n} \left(\sum_{k=0}^{n-1} \omega_n^{lk} f_{ex}(2k) + \omega_{2n}^l \sum_{k=0}^{n-1} \omega_n^{lk} f_{ex}(2k+1) \right), \text{ for } l = 0, 1, 2 \ldots, m-1. \tag{4.10}$$

From Eq. (4.10), the $\hat{f}$ is automatically obtained by forgoing the zero padding portion $(l = n, n+1, n+2, \ldots, m-1)$:

$$\hat{f}(l) = \frac{1}{2n} \left(\sum_{k=0}^{n-1} \omega_n^{lk} f_{ex}(2k) + \omega_{2n}^l \sum_{k=0}^{n-1} \omega_n^{lk} f_{ex}(2k+1) \right), \text{ for } l = 0, 1, 2 \ldots, n-1. \tag{4.11}$$

Eq. (4.11) provides the corresponding forward FFT of f_{ex} by the name fftpadForward, to obtain $\hat{f}_{ex}$, which is the padding zero extension of $\hat{f}$. Algorithm 4 presents the exact implementation routine.[1]

Algorithm 4 fftpadForward

Input: $f(0:n-1)$, $u(0:n-1)$
Output: $f(0:n-1)$
1: **function** fftpadForward
2: $f(:) \leftarrow FFT[f(:)]$;
3: $u(:) \leftarrow FFT[u(:)]$;
4: **for** $l = 0 \rightarrow n-1$ **do**
5: $f(l) \leftarrow f(l) + \omega_{2n}^{l} u(l)$;
6: **end for**
7: **end function**

Finally, we achieve the FFT without a bit reversal stage in the memory operation (see [12] as well) for the 1D implicitly zero-padded convolution, they are proceeded by fftpadBackward and fftpadForward as in [12].

The different FFT routines indicate a different convolutin routine, since the dimension is reduced to be n from $2n$-sized f_{ex} (zero-padded charge density), but two vectors f and u. In the meanwhile, the GF vector g is neither extended to be real even symmetric nor FFT of that extended vector, but is operated by the DCT routine. The 1D convolution routine has to be reorganized to fit the optimizations. The indices of g will be re-indexed to match the multiplication with both f and u, by the implicit RESE and the splitting indices of f_{ex} (f and u), as explained in function $Multiply1D$ (Algorithm 5):

Algorithm 5 Multiply1D

Input: $f(0:n-1)$,$u(0:n-1)$, $g(0:n)$
Output: $f(0:n-1)$,$u(0:n-1)$,
1: **function** $Multiply1D$
2: **for** $i = 0 \rightarrow n-1$ **do**
3: **if** $i < n/2$ **then**
4: $I_e = 2i$, $I_o = 2i+1$;
5: **else**
6: $I_e = 2(n-i)$, $I_o = 2(n-i)-1$;
7: **end if**
8: $f(i) = f(i) * g(I_e), \quad u(i) = u(i) * g(I_o)$;
9: **end for**
10: **end function**

[1] The fftpadBackward and fftpadForward routines may have a factor difference in comparison with the extended backward FFT and forward FFT, because of the different scaling strategies in practice.

Algorithm 6 presents a straightforward routine of the implicitly zero-padded convolution in 1D. For two input vectors: n-sized f (represents charge density), $n+1$-sized g (represents GF after DCT), and one output n-sized f (represents potential after the convolution). The convolution routine is unique compared to other applications such as Bowmann's convolution routine for turbulence simulation, since the GF vector is extended in the real even symmetric way rather than by padding zeros. 1D fftpadBackward is proceeded to replace the original FFT of f_{ex}. The function $Multiply1D$ multiplies f with even indices of g and u with odd indices of g in frequency domain. Finally, proceeding the fftpadForward with f and u, and the result of 1D convolution is achieved.

Algorithm 6 Implicitly zero-padded convolution in 1D

Input: $f(0:n-1)$, $g(0:n)$
Output: $f(0:n-1)$
1: **function** CONVOLUTION1D
2: $[u, f] \leftarrow$ fftpadBackward$[f]$;
3: $[f(:), u(:)] \leftarrow Multiply1D[f(:), u(:), g(:)]$;
4: $[f] \leftarrow$ fftpadForward$[f, u]$;
5: **end function**

4.3.3 Implicitly zero-padded convolution in 2D

The basic idea of the 2D implicitly zero-padded convolution routine is similar to the former 1D situation in Section 4.3.2. Before introducing the 2D convolution routine, the 2DfftpadBackward and 2DfftpadForward functions are given.

The 2DfftpadBackward is demonstrated in Algorithm 7: $f(0:N_j-1, 0:N_k-1)$ as the 2D input vectors, and four output vectors $f(0:N_j-1, 0:N_k-1)$, $f_{eo}(0:N_j-1, 0:N_k-1)$,$f_{oe}(0:N_j-1, 0:N_k-1)$,$f_{oe}(0:N_j-1, 0:N_k-1)$. These four resulting vectors are obtained serially by 1D fftpadBackwards along both directions. First, the fftpadBackwards along j direction derive $f_{oe}(:,:)$ and refresh $f(:,:)$. Second, twice fftpadBackwards along k direction of the two, derive $f_{oo}(:,:)$ and refresh $f_{oe}(:,:)$, and derive $f_{eo}(:,:)$ and refresh $f(:,:)$, separately. In Figure 4.1, a sketch of the 2DfftpadBackward transform is presented. The horizontal red color lines indicate the fftpadBackward along x direction, and the vertical blue color lines show the fftpadBackward along y direction.

In contrast, Algorithm 7 and Figure 4.2 show the inverse transform of 2DfftpadBackward: 2DfftpadForward. The notations are the same while the processing is reversed, and the fftpadBackward is replaced by fftpadForward.

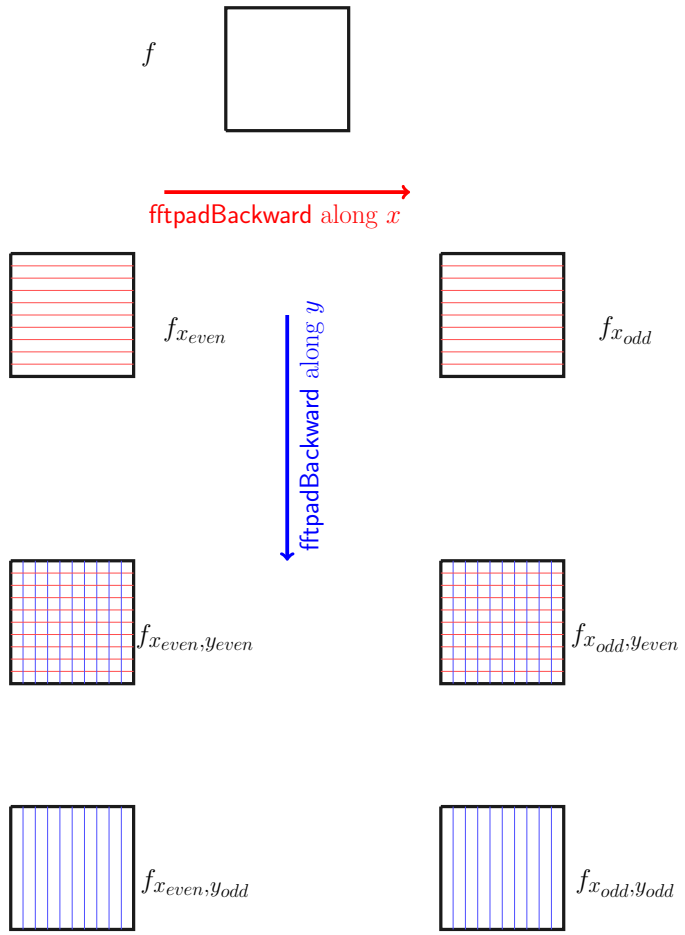


Figure 4.1: A sketch of the 2DfftpadBackward transform

Algorithm 7 2DfftpadBackward

Input: $f(0:N_j-1,0:N_k-1)$
Output: $f(0:N_j-1,0:N_k-1)$, $f_{eo}(0:N_j-1,0:N_k-1)$,$f_{oe}(0:N_j-1,0:N_k-1)$,$f_{oe}(0:N_j-1,0:N_k-1)$

1: **function** 2DfftpadBackward
2: **for** $k=0 \rightarrow N_k-1$ **do**
3: $[f(:,k), f_{oe}(:,k)] \leftarrow$ fftpadBackward$[f(:,k)]$;
4: **end for**
5: **for** $j=0 \rightarrow N_j-1$ **do**
6: $[f(j,:), f_{eo}(j,:)] \leftarrow$ fftpadBackward$[f(j,:)]$;
7: $[f_{oe}(j,:), f_{oo}(j,:)] \leftarrow$ fftpadBackward$[f_{oe}(j,:)]$;
8: **end for**
9: **end function**

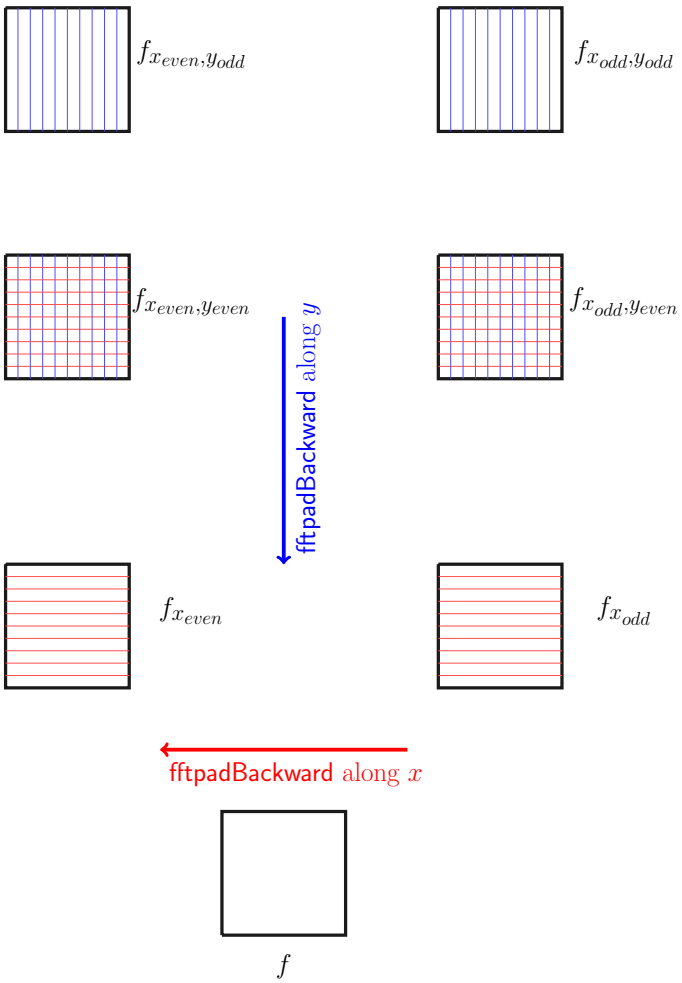


Figure 4.2: A sketch of the 2DfftpadForward transform

Algorithm 8 2DfftpadForward

Input: $f(0 : N_j - 1, 0 : N_k - 1)$, $f_{eo}(0 : N_j - 1, 0 : N_k - 1)$,$f_{oe}(0 : N_j - 1, 0 : N_k - 1)$,$f_{oe}(0 : N_j - 1, 0 : N_k - 1)$
Output: $f(0 : N_j - 1, 0 : N_k - 1)$

```
1: function 2DfftpadForward
2:     for j = 0 → N_j − 1 do
3:         [f(j,:)] ← fftpadForward[f(j,:), f_eo(j,:)];
4:         [f_oe(j,:)] ← fftpadForward[f_oe(j,:), f_oo(j,:)] ;
5:     end for
6:     for k = 0 → N_k − 1 do
7:         [f(:,k)] ← fftpadForward[f(:,k), f_oe(:,k)];
8:     end for
9: end function
```

The 2D implicitly zero-padded convolution starts with two inputs of data: $f(0 :$

$N_j - 1, 0 : N_k - 1)$ and $g(0 : N_j, 0 : N_k)$. The procedure is the following: first, take a 2DfftpadBackward for $f(:,:)$; second, conduct the piece-wise multiplication with $g(:,:)$ by $Multiply2D$ for the obtained results in 2D Fourier domain; third, use the 2DfftpadForward to obtain the output f. The pseudo code is shown in Algorithm 9.

Algorithm 9 Implicitly zero-padded convolution in 2D

Input: $f(0 : N_j - 1, 0 : N_k - 1)$, $g(0 : N_j, 0 : N_k)$
Output: $f(0 : N_j - 1, 0 : N_k - 1)$

function CONVOLUTION2D
    $[f(:,:), f_{eo}(:,:), f_{oe}(:,:), f_{oo}(:,:)] \leftarrow$ 2DfftpadBackward$[f(:,:)]$;
    $[f(:,:), f_{eo}(:,:), f_{oe}(:,:), f_{oo}(:,:)] \leftarrow Multiply2D[f(:,:), f_{eo}(:,:), f_{oe}(:,:), f_{oo}(:,:), g(:,:)]$;
    $[f(:,:)] \leftarrow$ 2DfftpadForward$[f(:,:), f_{eo}(:,:), f_{oe}(:,:), f_{oo}(:,:)]$;
end function

The element-by-element multiplication $Multiply$ in 2D is formalized in Algorithm 10. The function renews the four vectors $f(:,:)$, $f_{eo}(:,:)$, $f_{oe}(:,:)$, and $f_{oo}(:,:)$ with $g(:,:)$ in Fourier domain. The $g(:,:)$ is extended implicitly by RESE with reindexing.

Algorithm 10 Multiply2D

Input: $f(:,:)$, $f_{eo}(:,:)$, $f_{oe}(:,:)$, $f_{oo}(:,:)$, $g(:,:)$
Output: $f(:,:)$, $f_{eo}(:,:)$, $f_{oe}(:,:)$, $f_{oo}(:,:)$

function $Multiply2D$
    for $k = 0 \rightarrow N_k - 1$ **do**
        for $j = 0 \rightarrow N_j - 1$ **do**
            if $k < N_k/2$ **then**
                $K_e = 2k$, $K_o = 2k + 1$;
            else
                $K_e = 2(N_k - k)$, $K_o = 2(N_k - k) - 1$;
            end if
            if $j < N_j/2$ **then**
                $J_e = 2j$, $J_o = 2j + 1$;
            else
                $J_e = 2(N_j - j)$, $J_o = 2(N_j - j) - 1$;
            end if
            $f(j,k) = f(j,k) * g(J_e, K_e)$, $f_{eo}(j,k) = f_{eo}(j,k) * g(J_e, K_o)$,
            $f_{oe}(j,k) = f_{oe}(j,k) * g(J_o, K_e)$, $f_{oo}(j,k) = f_{oo}(j,k) * g(J_o, K_o)$;
        end for
    end for
end function

Moreover, the 2D implicitly zero-padded convolution routine is different from the structure of HandE's routine. There are other options for the convolution routine,

e.g. it can also be proceeded by the recurrence of 1D implicitly convolution, which is similar as the HandE's routine's recurrence structure. As mentioned before, the FFTs can be implemented in batch and parallel. For the future parallel routine, this structure is well organized for FFTs and the element-by-element multiplication computations, which is beneficial for the modern advanced computer architectures.

The 2D convolution algorithm can be used to solve the 2D space charge calculation as well as recurrence parts in the 3D convolution algorithm.

4.3.4 Implicitly zero-padded convolution in 3D

The 3D implicitly zero-padded convolution routine is demonstrated in Algorithm 11: the two inputs $f(:,:,:)$ and $g(:,:,:)$ are sized by $N_i \times N_j \times N_k$ and $(N_i+1) \times (N_j+1) \times (N_k+1)$, respectively. To start with, the fftpadBackward transfers $f(:,:,:)$ along i direction to obtain $f(:,:,:)$ and $f_o(:,:,:)$ over the other two dimensions y and z. Next, the 2D slices, $f(i,:,:)$ and $f_o(i,:,:)$, are ordered by index i from 0 to N_i-1 for the 2D convolution routine $CONVOLUTION2D$ is introduced in Section 4.3.3.

Algorithm 11 Implicitly zero-padded convolution in 3D

Input: $f(0:N_i-1, 0:N_j-1, 0:N_k-1)$, $g(0:N_i, 0:N_j, 0:N_k)$
Output: $f(0:N_i-1, 0:N_j-1, 0:N_k-1)$

1: **function** CONVOLUTION3D
2: **for** $j = 0 \rightarrow N_j - 1$ **do**
3: **for** $k = 0 \rightarrow N_k - 1$ **do**
4: $[f(:,j,k), f_o(:,j,k)] \leftarrow$ fftpadBackward$[f(:,j,k)]$;
5: **end for**
6: **end for**
7: **for** $i = 0 \rightarrow N_i - 1$ **do**
8: $[g_e(0:N_j-1, 0:N_k-1), g_o(0:N_j-1, 0:N_k-1)] = Set2Dg[g(:,:,:), i]$;
9: $f(i,:,:) = CONVOLUTION2D(f(i,:,:), g_e(:,:))$;
10: $f_o(i,:,:) = CONVOLUTION2D(f_o(i,:,:), g_o(:,:))$;
11: **end for**
12: **for** $j = 0 \rightarrow N_j - 1$ **do**
13: **for** $k = 0 \rightarrow N_k - 1$ **do**
14: $f(:,j,k) \leftarrow$ fftpadForward$[f(:,j,k), f_o(:,j,k)]$;
15: **end for**
16: **end for**
17: **end function**

Within the i loop, GF values $g(:,:,:)$ is extended to $g_e(:,:)$ and $g_o(:,:)$ in 2D for matching the i index with $f(i,:,:)$ and $f_o(i,:,:)$, the work is done by function $Set2Dg$ in Algorithm 12. After the two 2D implicitly zero-padded convolutions for both $f(i,:,:)$ and $f_o(j,:,:)$ with the corresponding $g_e(:,:)$ and $g_o(:,:)$ are finished for all is, $f(:,:,:)$ and $f_o(:,:,:)$ are transferred back by fftpadBackward to refresh f which is the result of the 3D convolution.

Algorithm 12 Set2Dg

Input: $g(:,:,:)$, i
Output: $g_e(:,:)$, $g_o(:,:)$
1: **function** $Set2Dg$
2: **if** $i < N_i/2$ **then**
3: $I_e = 2i$, $I_o = 2i + 1$;
4: **else**
5: $I_e = 2(N_i - i)$, $I_o = 2(N_i - i) - 1$;
6: **end if**
7: $g_e(0 : N_j - 1, 0 : N_k - 1) = g(I_e, :, :)$, $g_o(0 : N_j - 1, 0 : N_k - 1) = g(I_o, :, :)$;
8: **end function**

Optimization results of the 3D explicitly zero-padded convolution:
The optimization shows the benefits for both the computational complexity and memory assignment. In the HandE's routine, the FFTs of the zero data portions are skipped. The padding of zero data is implied implicitly, and other small computations are also skipped for the sake of computational complexity. In contrast, the memory assignment is probably the most apparent reason. Because the data transport inside the multidimensional FFT consumes time. Two 3D $N_x \times N_y \times N_z$ sized vectors $f(:,:,:)$ $f_o(:,:,:)$ and $(N_x+1) \times (N_y+1) \times (N_z+1)$ sized g, plus six 2D slices $f_{ee}(:,:)$, $f_{eo}(:,:)$, $f_{oe}(:,:)$, $f_{oo}(:,:)$, $g_e(:,:)$, and $g_o(:,:)$. In comparison with HandE's routine, the total memory is smaller mainly because g is not extended explicitly. On the other hand, all separated data is smaller in size but more in quantity. This memory assignment is more efficient than a single large data for data transport. The smaller separated data is also good for the parallel implementation with distributed memory. A direct comparison of CPU elapsed time can be found in Table 4.2.

Table 4.2: Elapsed time comparison of 3D explicitly zero-padded convolution and implicitly zero-padded convolution with increasing grids resolution.

N	3D explicitly zero-padded	3D implicitly zero-padded
32	0.011504 s	0.006787 s
64	0.180757 s	0.065217 s
128	1.576133 s	0.671114 s
256	14.21496 s	10.01754 s

There are other possible options for implementation, but the routine presented here is considered as the option which balances the computation between efficiency and memory storage.

4.4 The real to complex FFT implementation for the novel fast 3D convolution

Compared to general complex data, the Fourier transform for real data is specific. The vacancy of the imaginary portions, taken together with data symmetry, provides further efficient FFT and the fast trigonometric transforms. This section considers the case where the input data f is pure real vector, which is considered as a generic complex vector in the last sections.

4.4.1 The real to complex FFT for explicitly zero-padded data in i dimension

Theorem 5. *Suppose two n-sized real vectors $f = [f_0, f_1, \ldots, f_{n-1}]^T$, $g = [g_0, g_1, \ldots, g_{n-1}]^T$. If the complex vector $h = \mathfrak{F}_n(f + \mathrm{i}g)$, then*

$$\mathfrak{F}_n f = [(I_n + T_n)Real(h) + \mathrm{i}(I_n - T_n)Imag(h)],$$

$$\mathfrak{F}_n g = [(I_n + T_n)Imag(h) - \mathrm{i}(I_n - T_n)Real(h)],$$

where the T_n is the n-by-n identity matrix with the last $n-1$ columns arranged in reverse order.

Theorem 5 shows that the DFT of two real vectors can be obtained simultaneously by combining the two vectors as a same-sized complex vector. Thus a $2n$-sized real vector can be split into two n-sized real vectors (by odd-even index in reality), the sub-DFTs of the two real vectors are processed as Theorem 5. The final DFT of the original $2n$-sized real vector is obtained automatically based on Theorem 1. Moreover, when the elements of input data f are purely real numbers, the DFT output satisfies the "Hermitian" redundancy as shown in the following Theorem 6 (see also [87]).

Theorem 6. *Suppose a real vector $f = [f_0, f_1, \ldots, f_{n-1}]^T$, if*

$$h = \mathfrak{F}_n(f),$$

then h is conjugate even symmetric, h_{n-i} is the conjugate of h_n, i.e. $h_{n-i} = \bar{h}_i$, $1 \leq i < \lfloor \frac{n}{2} \rfloor$.

Thus, the DFT of any subvector of a real vector is conjugate even. This property cooperates with the last butterfly[2] so it can be simplified to be a conjugate-even butterfly to avoid redundant computation. The exact real to complex FFT is explained in [87] (Section 4.3). The improvement is a rough factor of two in both speed and memory usage for 1D case, technologically. From the practical side, if the input is a $2n$-sized real vector, the output vector size after the real to complex FFT is $n + 1$.

[2]The diagram of the data-flow of FFT in the radix-2 case is similar to the shape of a butterfly. The Cooley-Turkey FFT computation, in particular with data flow, is often referred to as a "butterfly" [93].

The real to complex FFT cannot be implied for all three directions. After the FFT in the first direction, the output vector is already a complex vector. Therefore we only use the real to complex FFT and the corresponding complex to real FFT in i direction. The routines are shown in Algorithm 13 and Algorithm 14.

Algorithm 13 The real to complex FFT in i dimension

Input: $f_{ex}(0:2N_i-1, 0:N_j-1, 0:N_k-1)$
Output: $f_{cmplx}(0:N_i, 0:N_j-1, 0:N_k-1)$
1: **function** FFTR2CINI
2: **for** $j = 0 \rightarrow N_y - 1$ **do**
3: **for** $k = 0 \rightarrow N_z - 1$ **do**
4: $f_{cmplx}(0:N_i, j, k) \leftarrow FFTr2c[f_{ex}(:, j, k)]$
5: **end for**
6: **end for**
7: **end function**

Algorithm 14 The complex to real FFT in i dimension

Input: $f_{cmplx}(0:N_i, 0:N_j-1, 0:N_k-1)$
Output: $f_{ex}(0:2N_i-1, 0:N_j-1, 0:N_k-1)$
1: **function** FFTC2RINI
2: **for** $j = 0 \rightarrow N_y - 1$ **do**
3: **for** $k = 0 \rightarrow N_z - 1$ **do**
4: $f_{ex}(0:2N_i-1, j, k) \leftarrow FFTc2r[f_{cmplx}(:, j, k)]$
5: **end for**
6: **end for**
7: **end function**

4.4.2 The fast routine with real to complex FFT for 3D convolution

For the other two directions, CONVOLUTION2D routine is implied whose algorithm is the same as mentioned in Section 4.3.3. The exact algorithm is shown in Algorithm 15 named as CONVOLUTION3D_rFFT.

The explicitly zero padded r2c FFT is efficient. Firstly, it is implemented with explicitly zero padding and approximately half of the computations of a normal FFT, compared to two complex FFTs as the implicit zero padded FFT way. Secondly, the implicitly zero padded FFTs for the other directions are also significantly improved because one CONVOLUTION2D is waived inside the loop for i in Algorithm 15, which reduces the computation size to a half (approximately) for the other two directions. Furthermore, the g ($\tilde{G}$) is not required to extend with odd and even indices as g_e, g_o (as in Algorithm 11). The indices of f_{cmplx} and g match perfectly.

Algorithm 15 Implicitly zero-padded convolution in 3D with real FFT

Input: $f(0:N_i-1,0:N_j-1,0:N_k-1)$, $g(0:N_i,0:N_j,0:N_k)$
Output: $f(0:N_i-1,0:N_j-1,0:N_k-1)$

1: **function** CONVOLUTION3D_RFFT
2: $f_{ex}(0:2N_x-1,0:N_y-1,0:N_z-1) \leftarrow PaddingZeroInX[f(0:N_x-1,0:N_y-1,0:N_z-1)]$
3: $f_{cmplx}(0:N_i,0:N_j-1,0:N_k-1) \leftarrow FFTR2CinI[f_{ex}(0:2N_x-1,0:N_y-1,0:N_z-1)]$
4: **for** $i=0 \rightarrow N_i$ **do**
5: $f_{cmplx}(i,:,:) = CONVOLUTION2D(f_{cmplx}(i,:,:), g(i,:,:))$;
6: **end for**
7: $f_{ex}(0:2N_x-1,0:N_y-1,0:N_z-1) \leftarrow FFTC2RinI[f_{cmplx}(0:N_i,0:N_j-1,0:N_k-1)]$
8: $u(0:N_x-1,0:N_y-1,0:N_z-1) \leftarrow f_{ex}(0:N_x-1,:,:)$
9: **end function**

Four different implementations of the GF-FFT method for Poisson's equation have been introduced so far. The results are compared in Table 4.3. The HandE's routine is faster than the trivial 3D FFT routine for a large problem size. In contrast, it is slower for a small problem size, which is possible because of the optimization of the 3D FFT within the FFTW library. Furthermore, the speed-up of the c2c FC routine (implicitly zero-padded convolution with FFT of complex data) is significant: the factor of the c2c FC routine to the HandE routine is around 2-4. The r2c FC (implicitly zero-padded convolution with FFT of real data) routine can further reduce the execution time by nearly a half of the value for the c2c FC routine.

Table 4.3: Comparison of different Poisson solvers' execution time with increasing grids resolution.

N_w	Trivial 3D FFT	HandE	c2c FC	r2c FC
32	1.5073e-02 s	1.7862e-02 s	8.2003e-03 s	5.2606e-03 s
64	1.7020e-01 s	2.3329e-01 s	7.3771e-02 s	4.6923e-02 s
128	4.7494e+00 s	3.2693e+00 s	7.7315e-01 s	4.1733e-01 s
256	4.5813e+01 s	2.9780e+01 s	1.0685e+01 s	6.0078e+00 s

4.5 Error classification of Green's function methods

In recent times, simulation has gained an essential importance alongside theory and experiments. As a starting point, simulation results are verified by physics theory and

experiment. Furthermore, simulation can predict results of theory and experiments which are beyond the limitations of reality. However, simulation does face a challenge because its results are never one hundred percent accurate. Sometimes the issue is even much more complicated because the derived physical model, the numerical errors or the involved factors are unknown.

Therefore, it is necessary to carry out an error analysis for the methods used. Optimally, this should be done before implementing them. For the PIC model, there are various error sources. The discussion about the propagation of error with time steps is beyond the topic of this dissertation. The contributions [49] and [45] study the numerical noise in PIC with time propagation. Here, the discussion of error is limited to one time step in the dissertation. The round-off errors resulting from computers' number digit limits are also ignored in the discussion.

4.5.1 Errors resulting from the PIC model

In the PIC model, the error sources are classified into two main sources and designated as charge-assignment and force-interpolation.

First, the fluctuation of feature density, which is connected to the coupling of particles, macro particles, and the mesh. The real number of charged particles is replaced by a smaller number of macro particles, which preserves the charge-mass ratio in simulation [38]. But as a source of errors, it is natural to increase this number of macro particles, bringing it closer to the real number of particles and reducing the error. However, the computing time increases dramatically as well. Second, errors occur due to the coupling between the macro particles and the mesh, which involves the usage of interpolation and deposition algorithms for the particles on the mesh. This occurs for both charge-assignment and force-interpolation procedures. Third, the short range effect of particles inside a mesh cell is automatically ignored. To increase the mesh number can resolve these issues, but the entire operation is still limited by the time consumption.

In total, the PIC model is an approximation methodology, rather than the real physical model. The error can not be waived due to the coupling between particles and the mesh. However, this error can be reduced by increasing the number of macro particles and mesh numbers to an acceptable level with the help of different applications.

4.5.2 Errors regarding the GF-FFT method in the Poisson solver

The discrete charge density is fluctuated by spatial grid effects and its discontinuous property. This fact affects the stability of the algorithm for the Poisson solver and requires attention.

Errors induced by numerical integration

The numerical integration error for the discrete convolution is the straightforward trigger. The numerical error due to the numerical integration algorithm is the main source for errors besides the model error.

Regardless of the discrete charge density, there is no additional discretization error for the IGF method, as it calculates analytical integration. While for both the GF method and the RIGF method, the numerical integral error should be considered. From a straight point of view: the "$1/r$" potential rule shows us: IGF and GF are coming closer as the "R_z distance" increases. The GF method induces a singular matrix at the "zero point" in calculation. For a local point (x_i, y_j, z_k), all error terms where $|k - k'| \leq R_z$ are zero for RIGF. The accurate potentials around (x_i, y_j, z_k) by means of the IGF method are determined by the distance R_z. Outside of this "R_z distance", the less accurate potentials are calculated by the GF method.

We define

$$\delta\varphi_(x_i, y_j, z_k) = \varphi_{RIGF}(x_i, y_j, z_k) - \varphi_{IGF}(x_i, y_j, z_k), \tag{4.12}$$

and suppose $\delta\tilde{G} = \tilde{G}_{RIGF} - \tilde{G}_{IGF}$. $0 < |\delta\tilde{G}| \leq \eta_{R_z}$, where η_{R_z} is the upper bound of $|\delta\tilde{G}|$.

The local potential error $\delta\varphi_{ex}(x_i, y_j, z_k)$ is obtained as:

$$\delta\varphi(x_i, y_j, z_k) = \frac{1}{4\pi\varepsilon_0} \sum_{i'=0}^{N_x-1} \sum_{j'=0}^{N_y-1} \sum_{k'=0}^{N_z-1} \rho_{ex}(x_{i'}, y_{j'}, z_{k'}) \delta\tilde{G}_{ex}(x_i, x_{i'}, y_j, y_{j'}, z_k, z_{k'}). \tag{4.13}$$

If $k \leq R_z$, $\delta\tilde{G}(i, j, k) = 0$ for the RIGF method.

Further, define $A_z = \mathbb{N}[0, k - R_z - 2] \cup \mathbb{N}[k - R_z - 1, 2N_z - 1]$, where $\mathbb{N}[a, b] = \{n \in \mathbb{N},\ a \leq n \leq b\}$. By the Cauchy-Schwarz inequality:

$$|\delta\varphi(x_i, y_j, z_k)| \leq \frac{1}{4\pi\varepsilon_0} \sqrt{\sum_{i'=1}^{N_x} \sum_{j'=1}^{N_y} \sum_{k' \in A_z} |\rho(x_{i'}, y_{j'}, z_{k'})|^2} \cdot \sqrt{\sum_{i'=1}^{N_x} \sum_{j'=1}^{N_y} \sum_{k'=R_z+1}^{N_z} \left|\delta\tilde{G}(x_{i'}, y_{j'}, z_{k'})\right|^2} \tag{4.14}$$

Since the RIGF method integrates with the mid-point rule, the bound of $\delta\tilde{G}_{GF}(x_i, y_j, z_k)$ can be derived from the common 1D error analysis, as:

$$\left|\delta\tilde{G}_{GF}(x_i, y_j, z_k)\right| \leq \frac{h_x^3 h_y^3 h_z^3}{24^3} K_x K_y K_z, \tag{4.15}$$

where K_x is the largest value of $\partial^2 G(x, y, z)/\partial x^2$ with $x \in [x_i - \frac{h_x}{2}, x_i + \frac{h_x}{2}]$. The numbers K_y, K_z are defined in the same way.

Substitute Eq. (4.15) to Eq. (4.14):

$$|\delta\varphi(x_i, y_j, z_k)| \leq \frac{1}{4\pi\varepsilon_0}\sqrt{\sum_{i'=1}^{N_x}\sum_{j'=1}^{N_y}\sum_{k'\in A_z}|\rho(x_{i'}, y_{j'}, z_{k'})|^2}\cdot \frac{|h_x^3 h_y^3 h_z^3|}{24^3}\sqrt{\sum_{i'=1}^{N_x}\sum_{j'=1}^{N_y}\sum_{k'=R_z+1}^{N_z}K_{i'}^2 K_{j'}^2 K_{k'}^2} \tag{4.16}$$

where $K_{i'}$, $K_{j'}$, $K_{k'}$ represents K_x, K_y, K_z in each grid cell. This leads to the bound of the potential error, which is determined by stepsizes in three directions, two summation terms of $|\rho(x_{i'}, y_{j'}, z_{k'})|^2$ and $K_{i'}^2 K_{j'}^2 K_{k'}^2$ which are dominated by R_z.

From the inequality (4.16), the error is also stable if the two summation terms are stable. For the term regarding $\tilde{G}$: the singular and strong decreasing portion property of the RIGF is excluded, the remaining term is smooth and stable. The term of discrete charge density is handled by the charge assignment scheme connected to Section 4.5.1. The summation should not vary much for different schemes. In total, the errors induced by numerical integration are stable and not sensitive when ρ and $\delta\tilde{G}$ are smooth. The $\delta\tilde{G}$ is, of course, smooth as the RIGF is specific in optimizing the $\tilde{G}$ function.

5 High performance computing studies for fast Poisson solvers

The novel efficient Poisson solver introduced in the previous chapters will be further developed in order to fit the modern HPC patterns. The parallel programs are managed by OpenMP, CUDA, and Open MPI. These programming APIs are specific for different levels of parallelizations and hardwares. The combination of these programming APIs is also available.

The HPC platforms include the multi-core CPU workstation, heterogeneous architecture of CPU and GPU, and supercomputers (and clusters).

Section 5.1 introduces the shared memory parallel routine for the multi-core CPU workstation. The OpenMP API is used to speed up the computation with the number of CPU threads in usage.

Section 5.2 attempts a heterogeneous parallelization of CPU+GPU for workstations. The parallel routine relies on an OpenMP+CUDA API implementation. The limitations of CUDA API are also the reasons for the heterogeneous parallelization rather than a pure GPU parallel routine.

Section 5.3 deals with the distributed memory parallel routine for supercomputers and clusters by applying MPI implementation. In pursuit of the highest speed-up possible, the limiting factor is the data transport between processors.

For the state-of-the-art supercomputers, both shared memory and distributed parallelizations are used. The MPI+OpenMP parallel routine is further developed in Section 5.4. An attempt for the future CPU+Many Integrated Core (MIC) architecture is prepared as well.

Some of the results described in this chapter have already been published in own publications [103] [104].

5.1 The shared memory parallel routine with multiprocessors

5.1.1 Shared-memory parallelization

Nowadays, almost all CPU products provided by the major CPU manufactures (i.e. Intel and AMD) are the multi-processor type in PC and server markets. An architecture called symmetric multiprocessing (SMP) is constructed by two or more identical processors through a shared memory. The memory in the SMP architecture is accessed equally for every processor as shown in Figure 5.1.

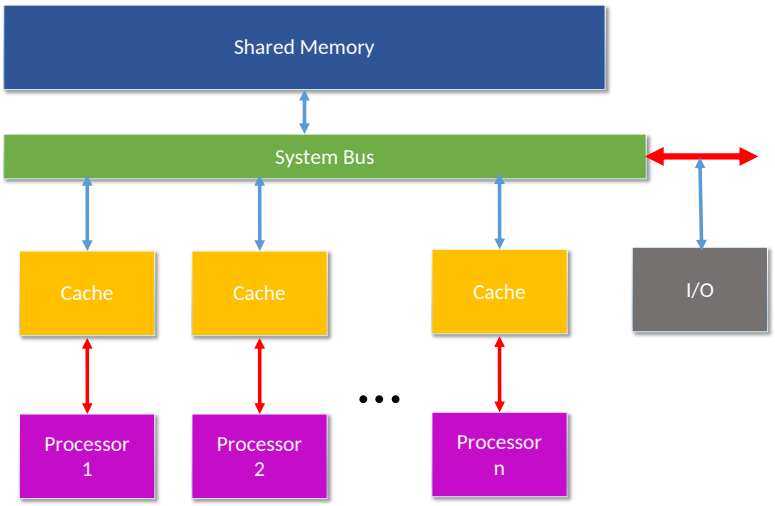


Figure 5.1: A schematic plot of the symmetric multiprocessing architecture, adapted from Ferruccio Zulian [105].

In contrast, there is the Non-Uniform Memory Access (NUMA) architecture which can be composed of more processors, e.g. 24 or 32. These processors are accessed in allocations differently to SMP. Processors in the same chunk access the same local shared memory. When a processor accesses the non-local memory in a different chunk, it has to use a different access latency, i.e. intersocket. This access is slower than the local memory access and leads to imbalance problems between the data transport and computing (see Figure 5.2).

All these kinds of CPUs have the capability of implementing programs in parallel by a thread-level parallelism (TLP). These threads correspond to the cores inside processors in CPU. One practical way to operate the parallelization is the OpenMP API.

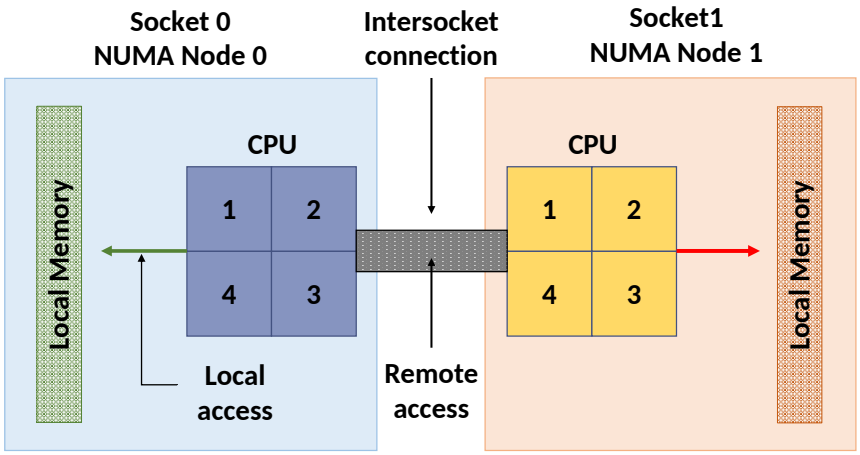


Figure 5.2: A schematic plot of the NUMA architecture, adapted from Frank Denneman [5].

The OpenMP API

The OpenMP API is defined and maintained via collaborative work from the hardware and software industry, and academia. The current version, OpenMP 4.5 specification, was released in November 2015. It is one of the most popular parallel programming models and still shows a promising future. It is a single standard rather than consisting of several manufacture-specific standards. It is also relatively easy to start with the original serial code by adding pre-processor compiler directive, e.g. *#pragma omp* in C/C++ language. Furthermore, the OpenMP committee is going to extend the directive language to support GPU and MIC accelerators, hence OpenMP will provide more parallel capabilities.

The OpenMP API's parallel pattern is based on the fork-join model formed in 1963 [22]. An OpenMP program starts sequentially with one thread, which branches off in a parallel region with other threads. Once the parallel region's execution is finished, the program is merged ("join") to one thread again and executed further with the sequential program. The "fork-join" procedures are proceeded in the program until the program finishes its operation.

Most parallel constructs in the OpenMP API are scheduled as compiler directives:

```
#pragma omp <construct> [clause1 clause2 ...]
```

whereby the clauses provide additional information for construct. For example:

```
#pragma omp parallel
{
....\\ parallel region
}
```

All threads in the CPU execute the parallel region simultaneously.

The clauses are supplementary and limitative for the construct, e.g.,

```
#pragma omp for[clause1 clause.2..]
for-loops
```

The *for*-loops are implemented by a number of threads. Due to the application's demand and thread safety, some variables need to be protected. Inside a *for*-loop, the variable is maintained by a thread, it should be protected against reuse by other threads. In this case, the available clause options are:

```
private(variable list),
firstprivate(variable list),
lastprivate(variable list),
...
```

To add the OpenMP API into codes is simple. All common open source compilers such as gcc, intel or clang include the OpenMP API automatically. In order to implement a program, the OpenMP flag (e.g. *-fopenmp* for gcc compiler, *-openmp* for intel compiler) should be attached during compiling.

Another technology for shared-memory parallelization in HPC is known as vectorization, or single instruction multiple data (SIMD). It is usually performed on arrays, i.e. a single machine instruction operates on multiple array elements that can certainly improve the CPU efficiency. OpenMP API provides the support for SIMD since OpenMP 4.0 is released.

The combination for both multi-threads and vectorization would be promising. OpenMP tries to work on the shared-memory parallelization standard. Therefore, the knowledge of OpenMP is rather complex and dynamic. Please refer to [10] for details on the OpenMP API implementation.

5.1.2 The OpenMP routine of the novel Poisson solver

The novel Poisson solver introduced in Section 4.3 has been further parallelized via the OpenMP API. Parts of the codes have been slightly changed for the purpose of the parallel routine. For an efficiency study, three major parallel portions are compared with the serial routine to check the speed-up by varying CPU thread numbers.

The parallelization of the efficient IGF calculations

The efficient IGF values are calculated with three level nested *for*-loops. The construct is *parallel for* and the corresponding clauses are *private* and *collapse*. The simplified code reads as:

```
#pragma omp parallel for private(m) collapse(3)
for (k=0;k<N[2];k++){
    for (j=0;j<N[1];j++){
        for (i=0;i<N[0];i++){
                    m=i+j*N[0]+k*N[0]*N[1];
            G_EffIntG[m]=EffIGF_Fun(i*h[0],j*h[1],k*h[2]);
}}}
```

The parallelization results of GF calculations are shown in Figure 5.3. The blue-colored curve is the ideal speed-up line while the black curve is the real implemented time trend. The closer the black curve and the blue curve are, the better the parallel routine performs.

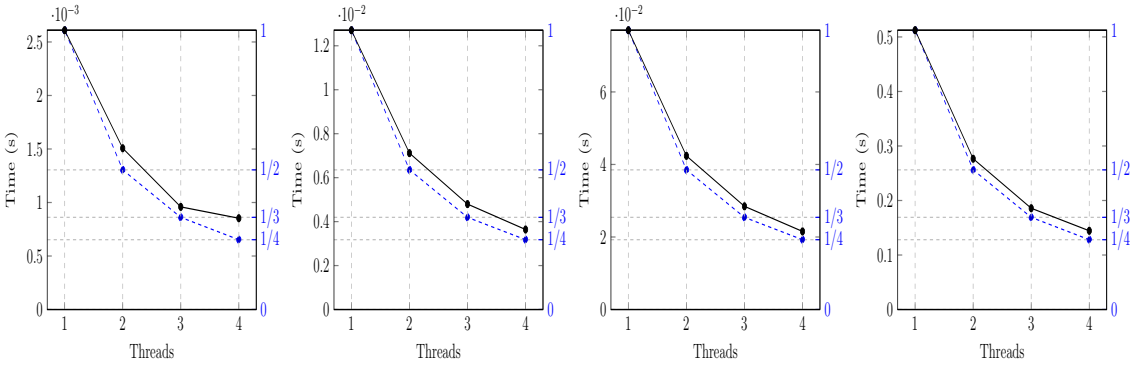


Figure 5.3: The parallelization of GF calculations with increasing thread numbers from 1 to 4 for different mesh numbers 33^3, 65^3, 129^3 and 257^3 (left to right), respectively.

The parallelization of FFTs and DCTs in the same direction

The novel Poisson solver relies heavily on FFT-related transforms: FFTs and DCTs. All these transforms are handled by the FFTW package. For parallel implementation, FFTW also provides its multi-thread version FFTs with OpenMP (or Pthread, another TLP API). The implementation is as follows:

First, the multi-thread routine should be compiled for the FFTW's configuration in order to enable the OpenMP property.

```
./configure --enable-openmp
```

Second, the following flags should be added in the makefile for compiling before the executable file is generated.

```
-lm -fopenmp -lfftw3 -lfftw3_omp
```

Third, before making FFTW's FFT plans, we add the following functions in advance. These functions initialize the multi-thread environment for FFTW and set the FFTW plans for the system's maximum number of threads, which is obtained by the OpenMP self-functions *omp_get_max_threads()*.

```
/*initialize multi-threads*/
fftw_init_threads();
/*make FFTW plan for omp_get_max_threads() threads*/
fftw_plan_with_nthreads(omp_get_max_threads());
```

The execute function stays the same as the original serial function as:

```
fftw_execute(fftw_plan);
```

Similar to Figure 5.3, Figure 5.4 shows the speed-up of FFTs inside the novel Poisson solver (DCTs are analogous). The speed-up is weaker than that of the GF calculations, but still comparable with the number of CPU threads used.

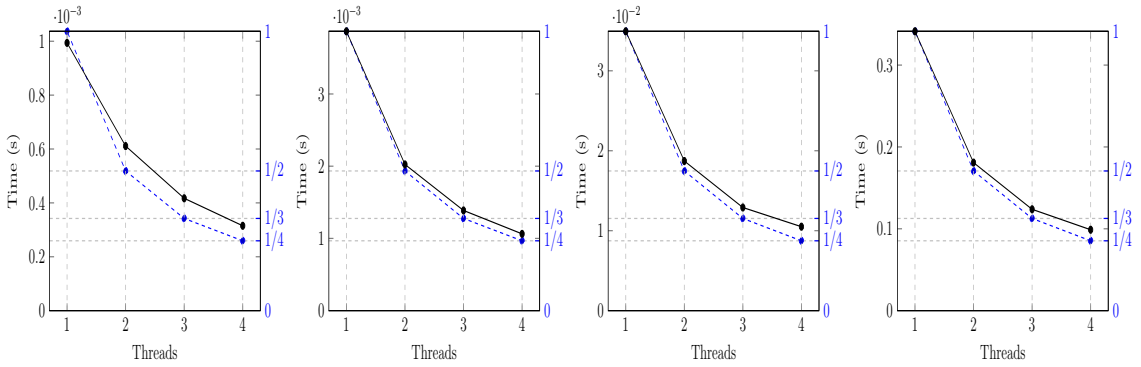


Figure 5.4: The parallelization of FFT with increasing thread numbers from 1 to 4 for different mesh numbers 32^3, 64^3, 128^3 and 256^3 (left to right), respectively.

The parallelization of element-by-element multiplications in Fourier space

The parallelization implementation of Algorithm 10 in Section 4.3.3 is simplified as follows:

```
#pragma omp parallel for private (ke,ko,Gk,je,jo,Gj,Gm_yeze,Gm_yoze,
    Gm_yezo,Gm_yozo,m) collapse(2)
for (k = 0; k <N2; k++){
  for (j = 0; j < N1; j++){
  /*set indices for f, f_eo, f_oe, foo*/
  m= ;ke= ;ko= ;je= ;jo= ;
  /*set indices for g*/
  Gk= ;Gj= ;Gm_yeze= ;Gm_yoze= ;Gm_yezo= ;Gm_yozo= ;
  /*complex multiplication in frequency domain*/
     f[m]=f[m]*g[Gm_yeze];      f_eo[m]=f_eo[m]*g[Gm_yezo];
     f_oe[m]=f_oe[m]*g[Gm_yoze];  f_oo[m]=f_oo[m]*g[Gm_yozo];
     }}
```

The speed-up of parallel element-by-element multiplications is comparable to the ideal case as shown in Figure 5.5.

Total efficiency improvement of the novel Poisson solver

The speed-up of the whole novel Poisson solver is compared with the ideal speed-up trend as shown in Figure 5.6. If the problem size is small, the speed-up is not significant. This may result from the overhead of the OpenMP API causing the major time consumption.

5.2 An effort on CPU+GPU heterogeneous parallelization

Since 2006, the GPUs have evolved in the HPC field, especially by the efficient manipulation for large blocked data. For instance, NVIDIA releases its CUDA platform

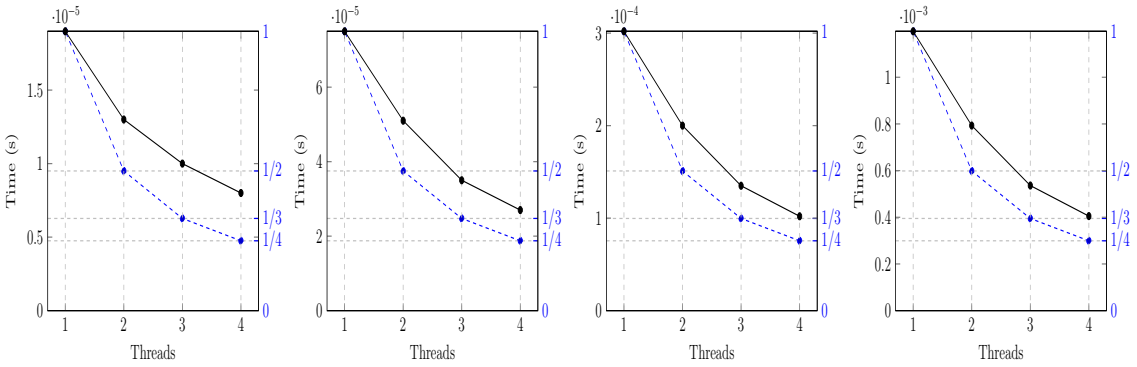


Figure 5.5: The parallelization of element-by-element multiplications with increasing thread numbers from 1 to 4 for different mesh numbers 32^3, 64^3, 128^3 and 256^3 (left to right), respectively.

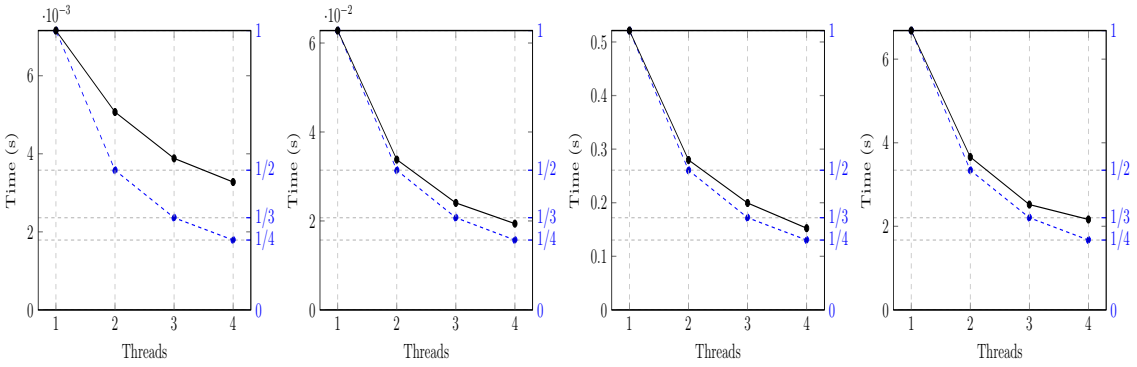


Figure 5.6: The parallelization of the novel Poisson solver with increasing thread numbers from 1 to 4 for different mesh numbers 32^3, 64^3, 128^3 and 256^3 (left to right), respectively.

for its GPU computing framework. Other manufacturers construct their own structures individually, e.g. DirectCompute by DirectX.API. Generally, an open standard known as Open Computing Language (OpenCL) [46] is published for cross-platform implementations. As mentioned, OpenMP has its own plan for the parallel implementation of the GPU together with CPUs.

A CPU has a small number of processors, usually 2 or 4 for PCs (up to 24 or 32 for workstations) while a GPU holds a high amount of CUDA cores. The CUDA cores have a lower computing capability than the CPU processors. Secondly, memory management is an advantage of GPU accelerators, e.g. scattered reads, shared memory, unified memory, and fast bandwidth in the GPU. These advantages will benefit highly parallel algorithms and large dataset computation.

On the other hand, data transformation between host (CPU random-access memory (RAM)) and device (GPU RAM) may affect performance, thus unmassive data computation may be not worthwhile. The thread control should fit with the hardware, e.g. the utilized thread number is 32 or the fold numbers of 32 depends on different GPU settings.

The IEEE Standard for Floating-Point Arithmetic (IEEE 754) is implemented

differently by vendors. In combination with other complex reasons, the computational results from CPU and GPU are slightly different.

Nevertheless, using GPU accelerators is attractive regarding the speed-up side, and has become a trend in the HPC field. Particularly, parts of the industry standard CPU-only libraries have been replaced by the GPU-accelerated libraries. For example, the FFT libraries are highly relied on in space charge calculation algorithms. The CUDA Fast Fourier Transform (FFT) product (cuFFT) library provided in CUDA by NVIDIA is specific to match the aim. The cuFFT reports that it is up to 10× faster than CPU [60]. The promising results provide an opportunity for a GPU implementation of the aforementioned Poisson solvers.

5.2.1 Technical limitation for the novel Poisson solver on a pure GPU platform

A full GPU version of the novel Poisson solver is not available for implementation so far. The cuFFT library does not support the real-to-real FFTs, such as 3D fast DCT. Since cuFFT includes highly-optimized algorithms and functions within CUDA structure, the fast DCT should be managed by cuFFT. Especially, a customized DCT routine can be far beyond the performance of its FFT peers in cuFFT. This trigger interrupts a full GPU Poisson solver.

Regarding the CPU and GPU's precisions of operations, the same sequence of operations may not have the same results due to complex reasons. For instance, the GPU may rearrange the operations, e.g. multiply-add operation for the multi-core implementations yielding different numeric results. Finally, IEEE 754 standard does not define the precision of most math functions. The CPU implementation chooses extended precision for the intermediate calculations, while the GPU does not. The IGF values involved with specific math functions are therefore achieved with different levels of precision in different implementations of GPU and CPU.

A pure GPU implementation of the novel Poisson solver is currently not possible. However, this cloud may actually have a silver lining, since a heterogeneous parallel implementation turns out to be a good choice.

Both CPUs and GPUs have strong but specific capabilities in computing. For the algorithm of the Poisson solver, the FFT performed by the cuFFT library in GPU is faster than the peer library in CPU. However, a CPU accurately calculates the IGF values and the fast DCT with FFTW. Coincidentally, the size of 3D DCT is approximately 1/8 of the size of 3D FFT: "small for slower, large for faster". These advantages and disadvantages of CPU and GPU are compensable. For the implementation, the routine is ideally separated into two portions, which are independent and simultaneous calculations. The facts given above show that the Poisson solver is capable and at least theoretically matches a CPU+GPU heterogeneous routine.

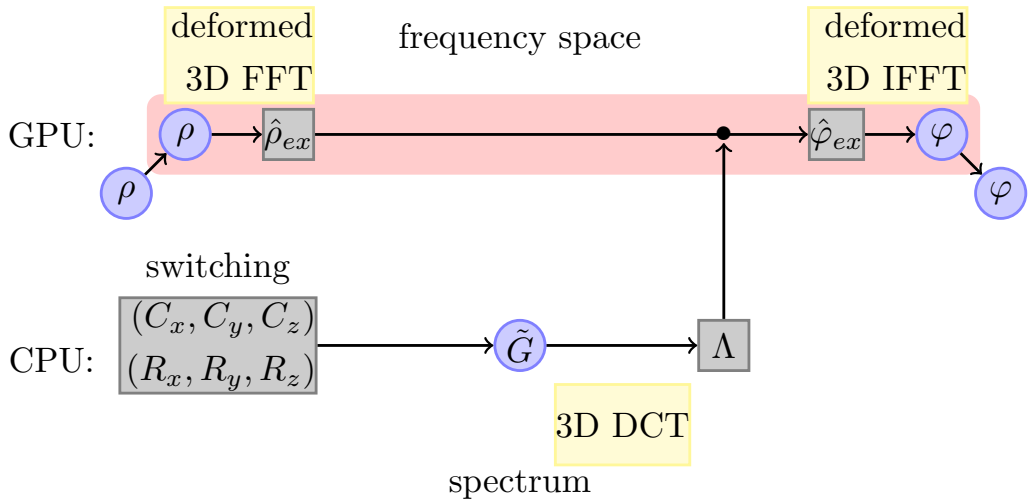


Figure 5.7: A schematic plot of the heterogeneous Poisson solver routine with CPU and GPU separation.

5.2.2 CPU+GPU heterogeneous parallel implementation

In this section, the algorithm used in a single workstation or PC with a GPU accelerator is considered. For the HPC programming of the inter-cross CPU and GPU routines, the OpenMP+CUDA framework is considered.

In details, the routine is controlled by CPU OpenMP threads. At the fork point of the Poisson solver, a CPU thread divides a GPU portion and other CPU threads still belong to the CPU portion, as shown in Figure 5.7. The aim of the separation is to fully accelerate the calculation with CPU+GPU. With an ideal synchronization, both of GPUs and CPUs can be sufficiently exploited.

There are a couple of options in OpenMP API to make such separation, e.g. with construct *sections* to make two sections, with construct *single*, or *master* to control the GPU portion.

First, with *#pragma omp parallel sections*, each portion fits a *omp section*. There is a thread-synchronization, when both sections finish the executions at the end. The construct is as follows:

```
#pragma omp parallel sections
{
#pragma omp section
{/*GPU section*/
}
#pragma omp section
{/*CPU section*/
}
}
```

If the workstation has only two processors, the construct *sections* would be sufficient

since only two sections are needed. Frequently, the calculation in GPU section is faster than the calculation in CPU section due to the fast cuFFT implementation in GPU. Inside the CPU section, the efficient IGF calculation costs more execution time than the 3D DCT of it. In the best case the CPU section and GPU section should be finished at nearly the same time. This leads to the performance balance between GPU and CPU capability.

Second, if there are more processors provided, this balance can be optimized by giving additional OpenMP threads to the CPU portion. The OpenMP function *omp_set_nested()* is applied to enable the nested environment.

```
omp_set_nested(1);
#pragma omp parallel num_threads(2)
{
if (omp_get_thread_num() == 0){
/*GPU portion with master thread*/
}
else{
/*CPU portion with other threads*/
}
}
```

Moreover, the two portions deal with different tasks.

GPU portion:

Inside the GPU portion, the CUDA programs are differently depending on the declaration of CUDA functions as follows:

```
/*Executed on the device, callable from the device only.*/
__device__ float DeviceFunction()
/*Executed on the device, callable from the host
with execution configuration <<<Dg,Db,Ns>>> */
__global__ void KernelFunction<<<Dg,Db,Ns>>>()
/* Executed on the host, callable from the host only.*/
__host__ float HostFunction()
```

The host means CPU while the device means GPU.

An OpenMP thread copies the charge density data ρ from the host RAM (of the computer) to the device (memory of the graphic card) using *CudaMemcpy* function which is provided by CUDA.

Padding zeros and processing the deformed DFT for the extended vectors are done in GPU. The functions are managed by self-defined *__device__* padding zero functions, and cuFFT library. The deformed 3D DFT can be implemented by either HandE's routine or the implicit zero-padded convolution routine. We use HandE's routine for an attempt study in this dissertation.

After synchronization, the spectrum Λ ($\tilde{G}$ after 3D DCT) is copied to the device by *CudaMemcpy* function again.

A function *__global__ void ComplexPointwiseMultiply()*, together with *__device__ inline cufftDoubleComplex ComplexScale()* function, are programmed for the complex multiplication of $\hat{\rho}_{ex}$ and Λ to obtain $\hat{\varphi}_{ex}$.

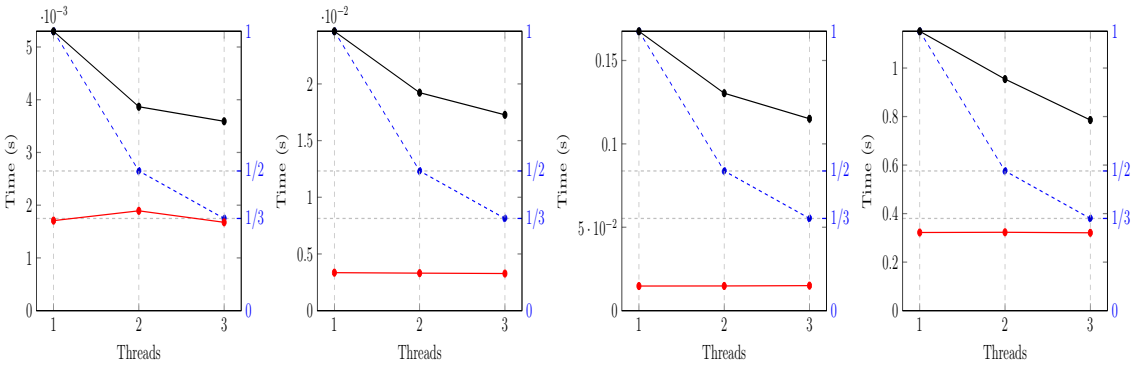


Figure 5.8: The CPU portion (black curve) and the GPU portion (red curve) calculations with increasing thread numbers from 1 to 3 for different mesh numbers 32^3, 64^3, 128^3 and 256^3 (left to right), respectively.

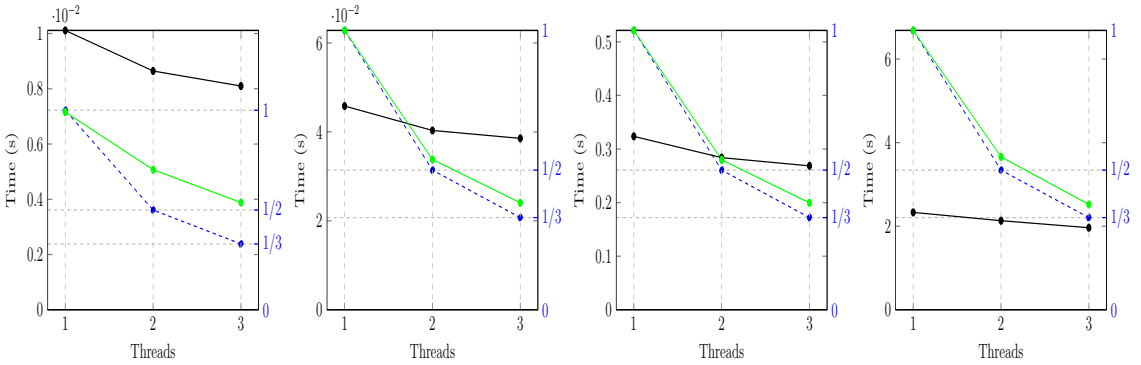


Figure 5.9: The hybrid CUDA+OpenMP Poisson solver (black curve) and the pure OpenMP Poisson solver (green curve) respectively. The mesh numbers are 32^3, 64^3, 128^3 and 256^3 (left to right), respectively.

The inverse transform procedure in HandE's routine is applied: 3D IDFT and cutting the original portion from $\hat{\varphi}_{ex}$ as φ. The final electrical potential φ is copied back to the host RAM by the *CudaMemcpy* function. Further calculations can be processed afterwards.

CPU portion:

OpenMP threads are responsible for the CPU portion in efficient IGF calculation and the following 3D DCT. Here, the efficient IGF becomes the critical factor in the efficiency improvement. Whereas the improvement for the whole Poisson solver in the pure serial CPU routine due to the efficient IGF is in the order of 15-25% (see [102]), in the parallel portions the efficient IGF leads to a much more significant improvement. The CPU program is the same as introduced in Chapter 3 and Chapter 4. After that, the synchronization between GPU and CPU portions is performed as mentioned above.

This heterogeneous parallel implementation can sufficiently exploit both CPU and

GPU by manually adding OpenMP threads for the CPU portion. In Figure 5.8, the execution time of the CPU portion is given with different threads. In comparison of the synchronization time, the execution time of the GPU portion is also plotted. As shown in the figure, the GPU portion is much faster than the CPU portion.

In Figure 5.9, the speed-up of the OpenMP+CUDA solver to the pure OpenMP parallel Poisson solver is compared. The total CPU threads for both solvers are 4. The hybrid solver (black curve) is not always faster than the pure OpenMP parallel solver (green curve). As shown, the efficiency can be achieved when the problem size is large (256^3).

This CPU+GPU routine is only an immediate parallel routine since there are some concerns in the CPU+GPU heterogeneous routine as well.

First, the data transport between the host and the device memory does not benefit the computation but still counts into executed time. Second, the overhead for the OpenMP API's preparations and operations consume some percentage points of the total execution time. Third, the load balance between the two portions of CPU and GPU may hang up one portion to wait for the other portion inside the computation. Fourth, this heterogeneous parallel routine is limited by the lack of full real-to-real FFT support in cuFFT. The explicit zero padded convolution routine by HandE is used for the current heterogeneous routine. Once the full real-to-real FFT is available in cuFFT, a full novel GPU Poisson solver is foreseen to be done by the author.

The competition between CPU and GPU has no end and the heterogeneous parallel routine has its own way to benefit from the procedure.

5.3 The MPI routine in supercomputer for the novel Poisson solver

A supercomputer is a high-level computer with an extremely strong computational capability compared to a normal computer. The development and deployment of HPC systems lead to an essential power to the scientific discovery, the economic competitiveness, and the modernization of industry. The cutting-edge HPC core technology has been making a great capability and competition inside the field: HPC, nowadays, is experiencing a revolutionary phase, where the parallelism on shared memory processors is benefitting from serious attention, e.g. the MIC architecture introduced by Intel and the CUDA platform defined by NVIDIA for GPUs are two strong competitors among the technology competition. Apart from that, international initiatives are also focussing on this potential, e.g. the European Union Horizon 2020 e-Infrastructures that build centers of excellence for computing applications, and the U.S. White House announced the "Advancing U.S. Leadership in High-Performance Computing" strategy in 2015.

The HPC complex field encompasses not only the "hard" device, e.g. presuming higher floating-point operations per second, but also the "soft" ability, e.g. the advanced numerical technologies in simulations of various computational problems.

In this section, the novel efficient Poisson solver will be further developed to fit the supercomputer's architecture with the MPI library. The parallel routine is programmed in Fortran 90 language, the parallellization is managed by Open MPI which is an open source project for HPC implementation. The source code is developed and maintained by a consortium of academic, research and industry partners.

For executing the parallel routine, the resources and facility of National Energy Research Scientific Computing Center (NERSC) at LBNL, are utilized. One of NERSC's newest supercomputers, Edison, named in honor of Thomas Edison, has a peak performance of 2.57 petaflops/sec, 133,824 compute cores for running scientific applications, 357 terabytes of memory, and 7.56 petabytes of online disk storage with a peak I/O bandwidth of 168 gigabytes (GB) per second. The parallel implementation for this chapter is materialized and tested on the Edison supercomputer.

5.3.1 The improved parallel Poisson solver

The improved parallel Poisson solver's routine

The parallel routine needs to apply a constant number of processors from the batch system. A batch system is a resource manager and scheduler of a supercomputer. Then the constant processors utilizing in the solver, numbered as np, are mapped onto a 2D $np_{Row} \times np_{Col}$ process grid. For this purpose, we need to set a new communicator of the topology information with 2D Cartesian grid by replacing the original communicator of the 1D topology information. The Open MPI function *MPI_cart_create* is applied. By using the MPI function *MPI_cart_coords*, each process owns a unique ID (RowID, ColID) as the element of a matrix, and $np = np_{Row} \cdot np_{Col}$.

The charge density $\rho(i, j, k)$ is distributed onto the process grid by filling (j, k)s into the (RowID,ColID)s as $(j_{\text{local}}, k_{\text{local}})$s equally. Only the is are free to be utilized by the local processor. Since the indices of j, k are distributed along the grid, the elements ordered by the two directions can not be used directly due to the separation of memory in different processors. The indices (j, k) can be obtained by Eq. (5.1) and Eq. (5.2),

$$j = \sum_{\text{ID}_j=0}^{\text{RowID}-1} N_{\text{ID}_j} + j_{\text{local}}, \tag{5.1}$$

$$k = \sum_{\text{ID}_k=0}^{\text{ColID}-1} N_{\text{ID}_k} + k_{\text{local}}, \tag{5.2}$$

where $(N_{\text{ID}_j}, N_{\text{ID}_k})$ are the local lengths of $(j_{\text{local}}, k_{\text{local}})$s at process $(\text{ID}_j, \text{ID}_k)$.

As presented in Figure 5.10, $\rho(i, j, k)$s are distributed in the 2D grid processes by the order (i, j, k) (i(green), j(red), k(blue)) in each process.

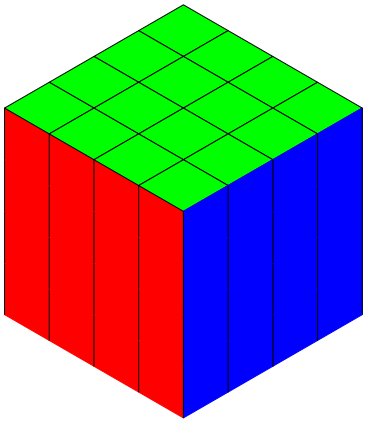


Figure 5.10: A schematic plot of the 3D charge density data distributed to a 2D Cartesian grid of processes.

Inside the improved parallel Poisson solver's routine, the multidimensional FFT becomes more challenging than the serial routine. Unlike the shared-memory computer programming (normal computers), the transport of data is a serious problem in parallel programming (supercomputers). The FFTs in different directions for the 3D situation are linked by the self-programmed transpose functions. The specified transpose function rearranges the 3D vector by exchanging the order of directions, e.g. ijk to jik or kji for each time. Communications and transports of data are a special challenge for supercomputers. This problem requires intensive concern. To use the MPI function *MPI_alltoall* is one option for the multidimensional transpose. The time consumption of the data transpose can reach a higher percentage of the total CPU time compared to the pure computing time of the multidimensional Fourier transform.

The following improved parallel Poisson solver is based on the serial routine in Section 4.4.1 in Chapter 4.

The MPI parallel programming routine is organized by four parts:

Part 1. The efficient IGF calculation and the related 3D DCT (Figure 5.11)

The GF values $G(:,:,:)$ calculated by the efficient IGF methods are distributed in the 2D grid processes by the order (k, j, i) (k(blue), j(red), i(green)) in each process. This is different from the order of the stored ρ ((i, j, k)). The reason for these index settings is the synchronization of the calculations, it is better to have the same computation quantity in each processor. There will be no load balance problems in this case. In contrast, in case of (i, j, k) order, the processors computing GF values would wait for the processors computing the IGF values due to the different complexity for the two groups. One way to treat this issue is to calculate the efficient IGF along the k direction. In this case each processor can finish the tasks at approximately the

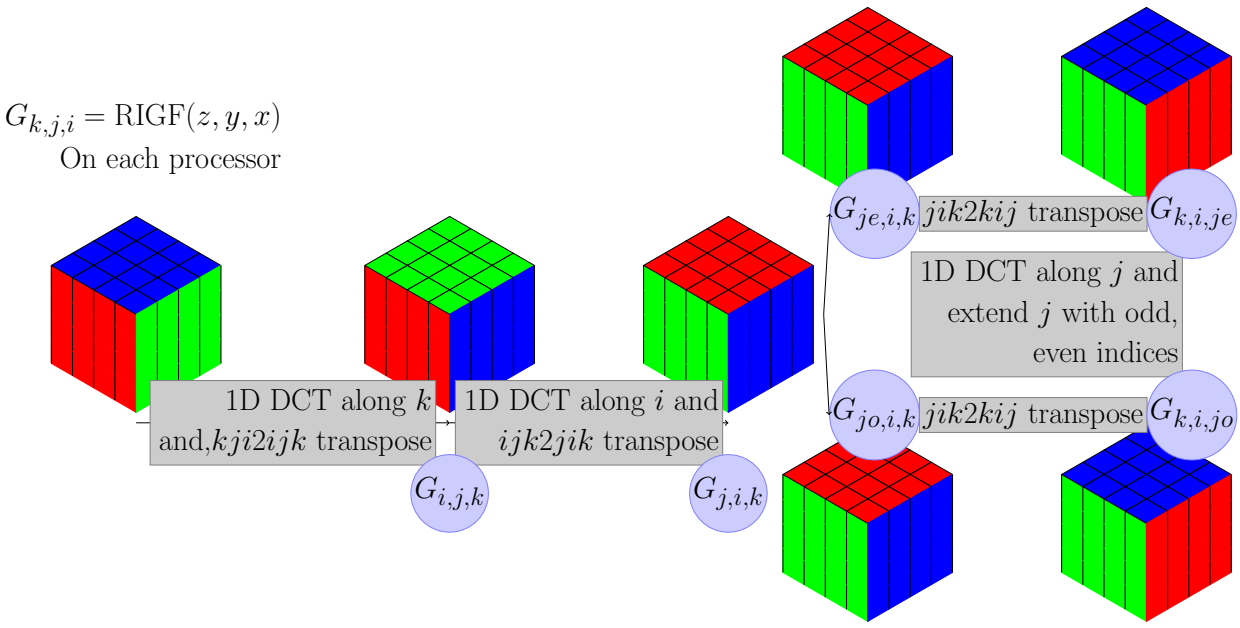


Figure 5.11: The efficient IGF calculation and the related 3D DCT.

same time rather than compute the efficient IGF along the other two directions, i.e. i, j directions. This is a special change for the MPI routine.

The implementation of the 3D DCT of G is processed by serially executing 1D DCTs along each direction in the 3D space. For the non-shared memory parallel routine, the 1D DCTs of the i, j cannot be set directly among all the processors. The work is obtained by taking the transpose functions of the index order, i.e. $kji2ijk$, and $ijk2jik$. At the beginning, we implement the 1D DCTs along k and take the $kji2ijk$ transpose, then implement the 1D DCTs along i and take the $ijk2jik$ transpose. In the end, the 1D DCTs along j are implemented to finish the 3D DCT of G. A special change for the parallel is: the results are extended in j direction as $G_{je,i,k}$ and $G_{jo,i,k}$ by classifying the even and odd index of $G_{j,i,k}$ along j direction in order to avoid the data transfer and communications in multiplication in Fourier space. Furthermore, the (j, i, k) order is transposed to the order of (k, i, j) for the further multiplication in **Part 3**.

Part 2. The 3D backward deformed FFT routine (Figure 5.12)

ρ is directly padded with the same size zeros in i direction as $\mathrm{Ex}\rho_{i,j,k}$. The following deformed 3D FFTs is constructed.

First, we act the $R2C$ FFTs to the real vectors $\mathrm{Ex}\rho_{i,j,k}$ in order to obtain the complex vectors $C\rho_{i,j,k}$. The transpose function of index between i and j is proceeded as $ijk2jik$ function. Second, the FFTPADBACKWARDS along j is organized from $C\rho_{i,j,k}$ to $C\rho_{k,i,je}$ and $C\rho_{k,i,jo}$. The transpose function of index between j and k is then followed by $jik2kij$ function twice, for both $C\rho_{k,i,je}$ and $C\rho_{k,i,jo}$. Third, the

FFTPADBACKWARDS along k is scheduled from $C\rho_{k,i,je}$ to $C\rho_{ke,i,je}$ and $C\rho_{ko,i,je}$, and from $C\rho_{k,i,jo}$ to $C\rho_{ke,i,jo}$ and $C\rho_{ko,i,jo}$, respectively. The four vectors $C\rho_{ke,i,je}$, $C\rho_{ko,i,je}$, $C\rho_{ke,i,jo}$, and $C\rho_{ko,i,jo}$ are prepared for the further multiplication in **Part 3**.

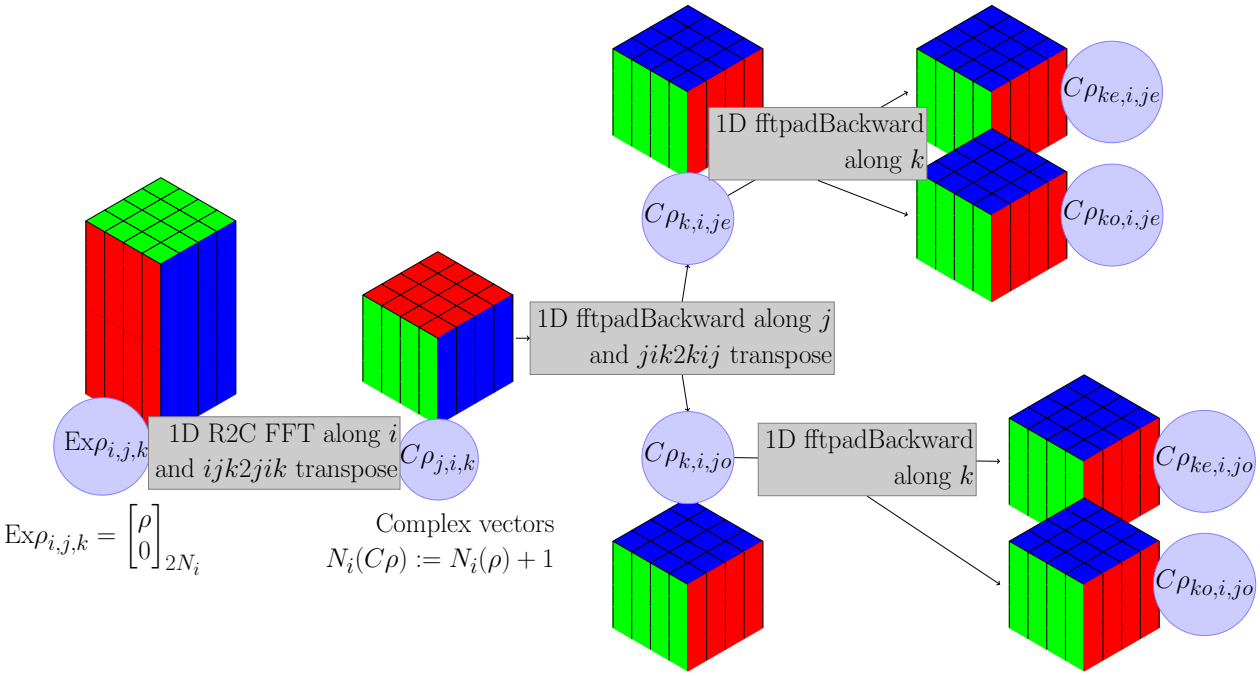


Figure 5.12: The 3D backward deformed FFT routine.

Part 3. Multiplication in Fourier space (Figure 5.13)

The results from **Part 1** and **Part 2** are multiplied element by element. $G_{k,i,je}$ and $G_{k,i,jo}$ need to be extended by the implicit RESE. The detailed multiplication is updated by the following rule in Eq. (5.3),

$$\begin{aligned} C\rho_{ke,i,je} &= \mathrm{RESE}(G_{k,i,je})_{k\ \mathrm{even}} * C\rho_{ke,i,je}, \\ C\rho_{ko,i,je} &= \mathrm{RESE}(G_{k,i,je})_{k\ \mathrm{odd}} * C\rho_{ko,i,je}, \\ C\rho_{ke,i,jo} &= \mathrm{RESE}(G_{k,i,jo})_{k\ \mathrm{even}} * C\rho_{ke,i,jo}, \\ C\rho_{ko,i,jo} &= \mathrm{RESE}(G_{k,i,jo})_{k\ \mathrm{odd}} * C\rho_{ko,i,jo}, \end{aligned} \tag{5.3}$$

where the two vectors obtained in **Part 1** are $G_{k,i,je}$ and $G_{k,i,jo}$. The four vectors obtained in **Part 2** are $C\rho_{ke,i,je}$, $C\rho_{ko,i,je}$, $C\rho_{ke,i,jo}$, and $C\rho_{ko,i,jo}$ (see also in Figure 5.13). The "even" and "odd" at the subscript indicate the even and odd indices of a vector.

Part 4. The 3D forward deformed FFT routine part (Figure 5.14)

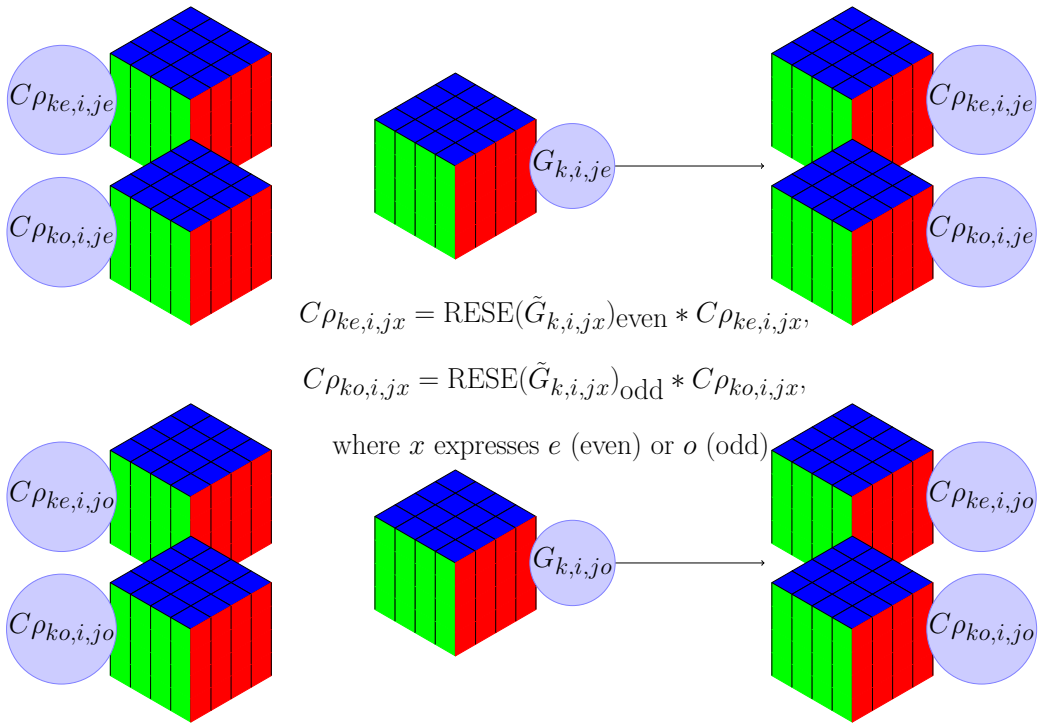


Figure 5.13: Multiplication in spectral domain.

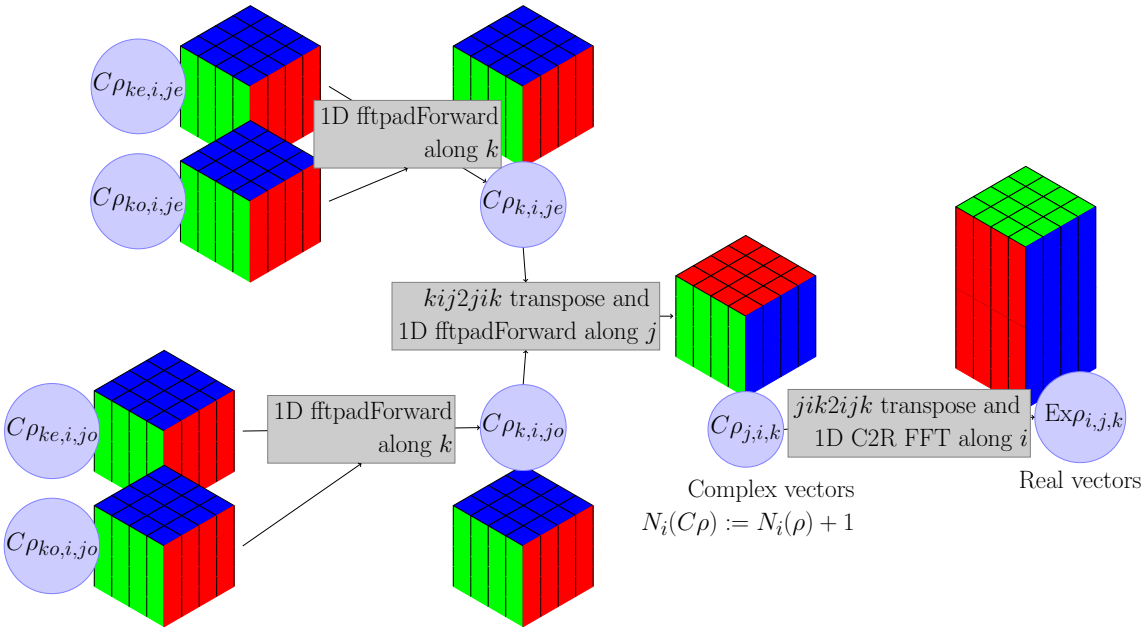


Figure 5.14: The 3D forward deformed FFT routine.

The inverse procedure of **Part 2** is performed. Among the four updated vectors, $C\rho_{ke,i,je}$, $C\rho_{ko,i,je}$ are given as the inputs to function FFTPADFORWARDS along k

direction to obtain $C\rho_{k,i,je}$. The same routine for $C\rho_{ke,i,jo}$ and $C\rho_{ko,i,jo}$ to obtain $C\rho_{k,i,jo}$. The two renewed vectors $C\rho_{k,i,je}$, $C\rho_{k,i,jo}$ are transposed by $kij2jik$ to make the vectors with full indices along j. Then the 1D FFTPADFORKWARDS along j becomes available to gain $C\rho_{j,i,k}$. For the last i direction, the $jik2ijk$ transpose function is implied to update $C\rho_{i,j,k}$, and the further $C2R$ FFTs along i are executed. The obtained real vector $\mathrm{Ex}\rho_{i,j,k}$ is $2N_i$-sized in i dimension. The front N_i parts of the $\mathrm{Ex}\rho_{i,j,k}$ is the expected solution of Poisson's equation.

5.3.2 Study of the improved parallel Poisson solver

For the parallel routine of the Poisson solver, the implementation needs additional settings regarding the execution, e.g. different compiling plans, different assignment of processor numbers, and the process grid.

The weak scaling and strong scaling study of the solver:

For the parallel performance, an ideal efficiency is achieved if the computational speed is preserved when we raise the number of processes and increase the size of the problem. However, in practice, the results do not match the ideal results. For the parallel performance of an application, some measurements have to be studied. In general, there are two scaling studies to check whether a problem is cpu-bound or memory-bound: weak scaling and strong scaling.

Weak scaling

If the problem size is defined as $N_xN_yN_z$ and the process number is defined as np, we keep the ratio of the two, $N_xN_yN_z/np$, to be constant with increasing the process number for the weak scaling study. Thus, the measurements of the execution time are compared with increasing both problem sizes and processors. The study is scheduled to have large communication patterns for a large number of processors. A good algorithm can keep the ratio as stable as a constant in performance. In this case, the algorithm can be implemented for a large problem without interruptions, i.e. overloads, communications.

If t_1 is defined as the execution time for a unit task finished by one processor, t_{np} as the execution time for np units task finished by np processors, the weak scaling efficiency is calculated by

$$\frac{t_1}{t_{np}} \times 100\%.$$

The weak scaling time scales and efficiency of the improved parallel Poisson solver are shown in Figure 5.15. The run time increases rather than remaining at an ideal constant, which may be because of the heavy usage of the global communication patterns. In total, the improved parallel Poisson solver does not perform very well for a large problem size and process number, which may be due to the 3D FFT and global transposes inside the algorithm.

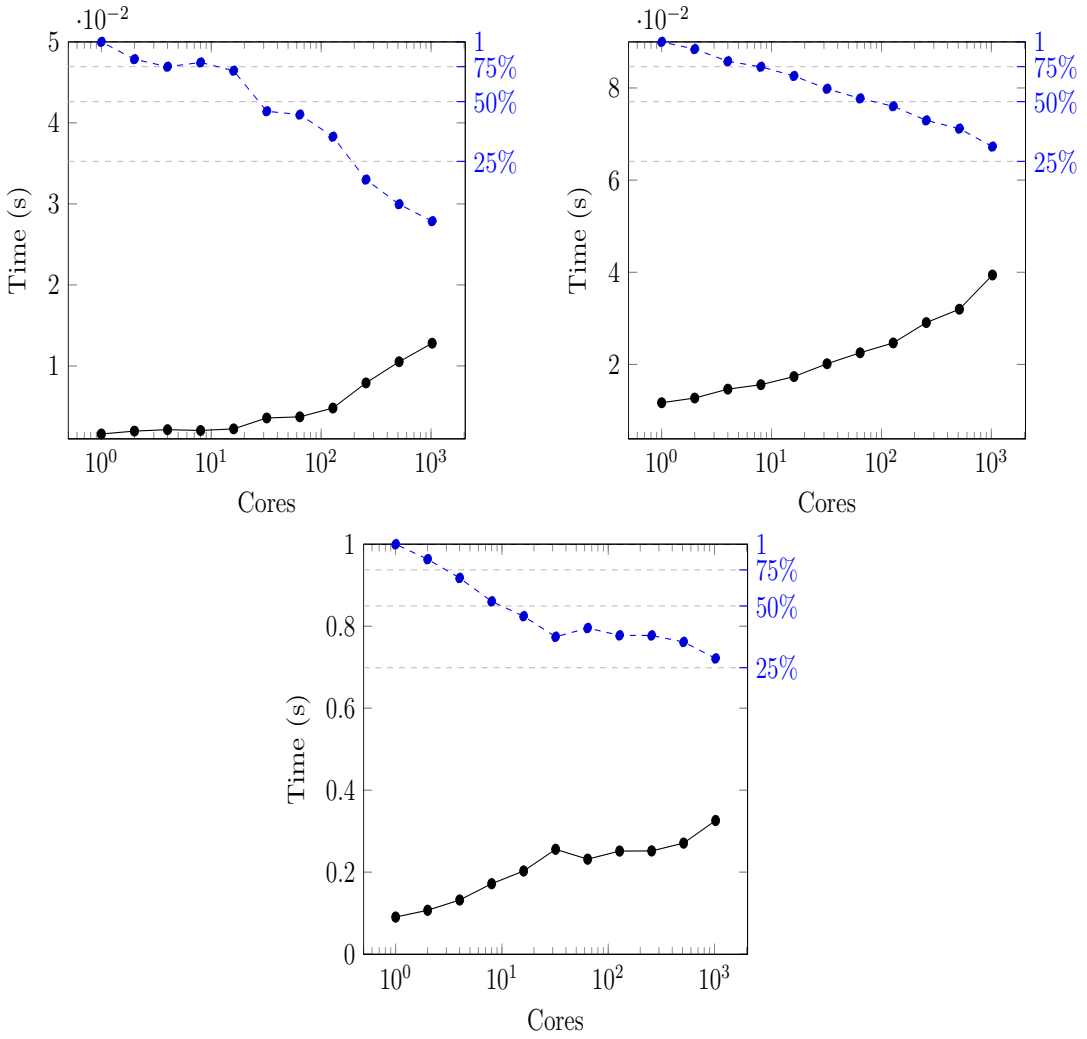


Figure 5.15: The weak scaling study: fixing the ratio between the size of the problem and the cores in order to keep it constant. The starting points are 16^3/core (top left), 32^3/core (top right), and 64^3/core (bottom) respectively. The blue curve is the weak scaling efficiency while the black curve is the execution time t_{np} for the corresponding number of cores.

Strong scaling

We increase the process numbers for the constant problem size for strong scaling (The problem size, $N_xN_yN_z$, stays fixed while the process number, np, is increased). The execution time is measured with the process numbers. In strong scaling, it is ideal to scale linearly, but difficult to stay on the linear property. The limitation point will be a reasonable choice between time and parallel overhead.

If t_1 is execution time for 1 core of a $N_xN_yN_z$-sized task while t_{np} is the execution

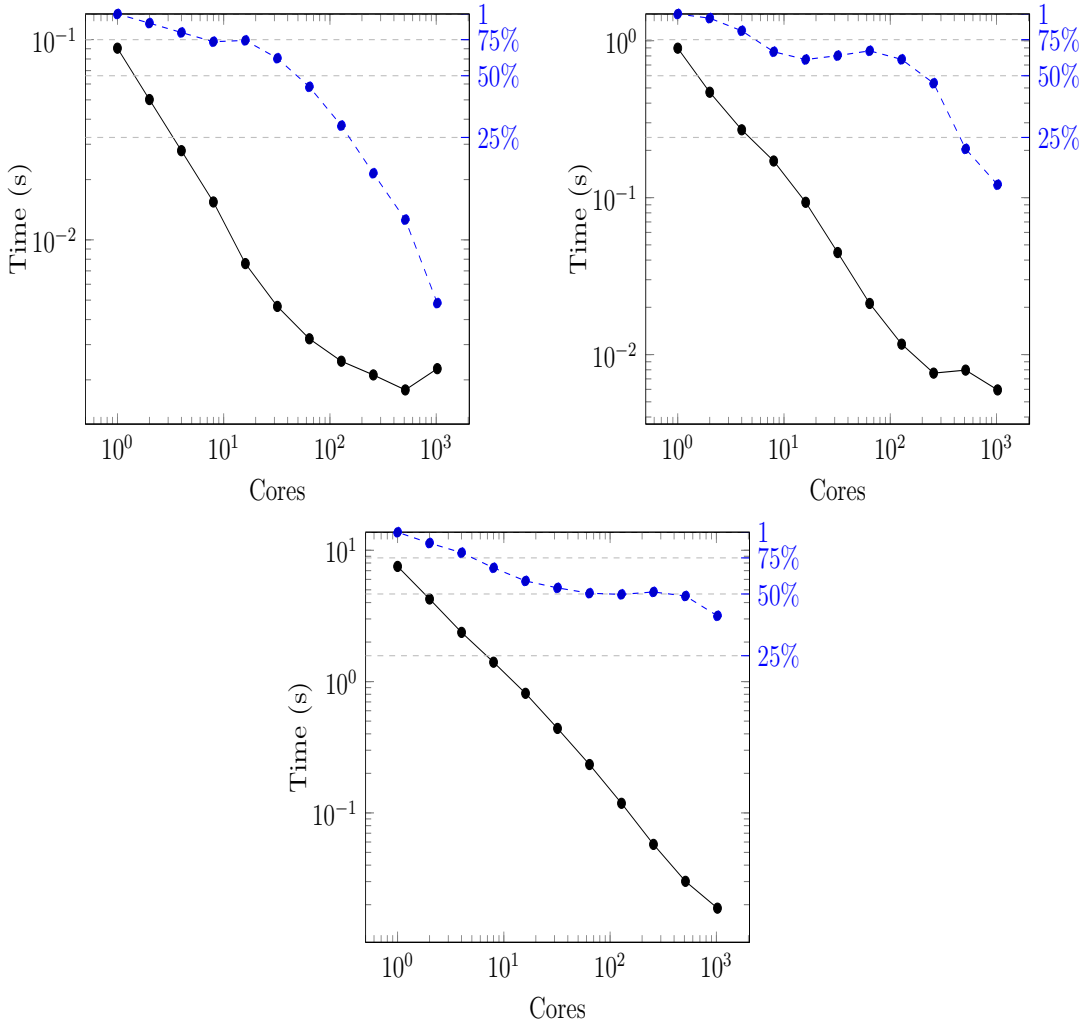


Figure 5.16: The strong scaling study: fixing the size of the problem, 64^3(top left), 128^3 (top right) and 256^3 (bottom), while increasing the number of cores in processing. The blue curve is the weak scaling efficiency while the black curve is the execution time t_{np} for the corresponding number of cores.

time for np cores of the same size task. The strong scaling efficiency is given by

$$\frac{t_1}{np \times t_{np}} \times 100\%.$$

The strong scaling figures are plotted in Figure 5.16.

The strong scaling time scales and efficiency of the improved parallel Poisson solver are shown in Figure 5.16. The run time decreases linearly until the number of cores becomes large. For problem with larger number sizes, the performance improves.

In total, the performance of the improved parallel Poisson solver shows a better performance in strong scaling than in weak scaling due to the heavy utilization in communications and transposes.

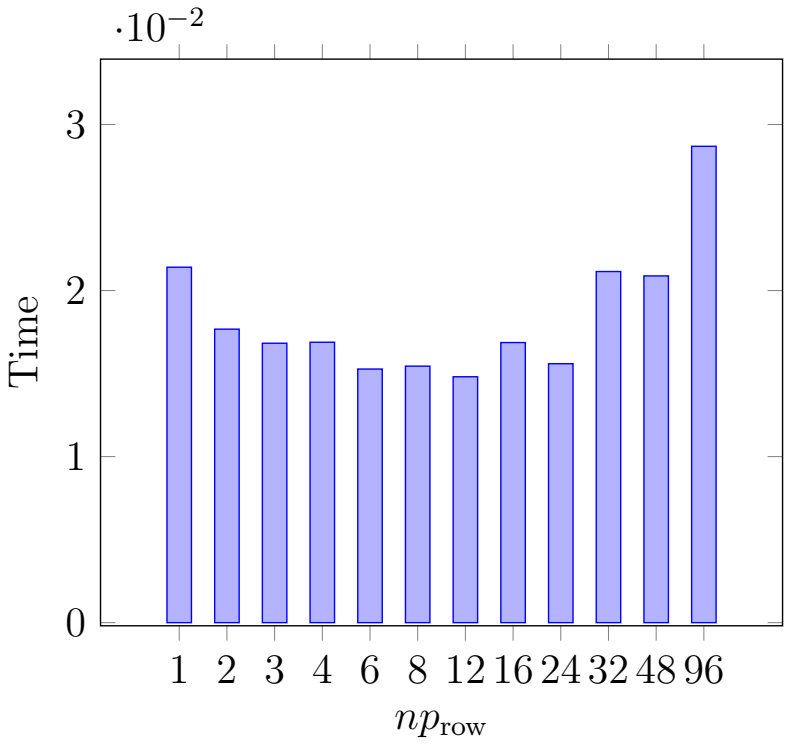


Figure 5.17: CPU time for different assignment of processors in the process grid.

Different assignment of processors within the process grid:

The np processors can be assigned by the *MPI_cart_coords* function with different row-column combinations. Some processors are within the same node while others are not.

The variation of the process grid for the same np processors can lead to different execution time.

As an explanation in practice: a total of 96 processors are applied for the implementation. By $np = np_{Row} \times np_{Col}$, the np_{Row} can vary from 1, 2, 3, 4, 6, 8, 12, 16, 24, 32, 48, and 96. Figure 5.17 shows the CPU time for the above np_{Row}s. We can see that if np_{Row} is roughly $\sqrt{np}$, the computing time is shorter than the other situations. On the other hand, np_{Row} differs from np_{Col} when we switch the two values, especially $np_{Row} = 1$ costs much less time than $np_{Col} = 1$ shown in Figure 5.17. The reason for this case is that Fortran uses the column-major order method for arranging multidimensional arrays in memory. $np_{Row} = 1$ provides the column-memory accesses across np columns while $np_{Col} = 1$ means the row-memory accesses across np rows. The memory accesses in row and column directions are different for data transport. The column access in memory is much more efficient than the row access in memory. This conducts the data transport differently with arrangement in vector transpose as can be seen in Figure 5.17.

Different compiling plans:

A compiler transforms the programming language known as source language (high-level programming language), e.g. C, Fortran, into the binary form target language (low-level programming language). With converting source code into an executable program, the program can be written from machine-independent source programs. Therefore, the compiler can compile the same source program for different specific machines.

For large applications in HPC, the compilers may not be complete enough to handle the programs. In Edison, there are a couple of compilers identified by vendors, e.g. Intel, Cray and GNU. For each individual compiler, different optimization strategies can be applied that can make various differences in performance. These techniques have to be pre-studied before generating the final execution program for simulation.

By conducting dozens of attempts, we confirmed that the Intel compilers have a better performance than others for the improved parallel Poisson solver implementation in execution time. Thus, the Intel compilers are chosen for computation in this dissertation.

5.4 The MPI + OpenMP parallel routine

As mentioned before, there are different types of shared-memory, e.g. holding by the cores within a computing node, using the MIC architecture as the co-processor for the CPU, and cooperating with GPU structure. The first option meets the bottleneck of the CPU manufacturing. The other two options are probably choices for the future HPC industry improvement. The competitions are still going on: up to the latest (November 2015) TOP 500 supercomputer list [39], 104 systems (90 on July 2015) are using the new accelerating technology for shared-memory parallel computations, 66 of these use GPUs, and there are 27 systems with MIC co-processors, 4 systems use a combination of the two.

From the programming side, the MPI+OpenMP routine would be a promising combination. The OpenMP API has been widely used for a long period. For Intel's MIC architecture, it is the official and solely supported environment that programs should be considered for the hybrid parallel routine.

5.4.1 The MPI+OpenMP routine

In this section, the hybrid parallel routine is scheduled for the existing code without unnecessary reprogramming scripts. This is organized by attaching the OpenMP API scripts in the code directly.

Inside a computational node involving the parallel computation, the OpenMP co-operates with the Open MPI process. The cooperation's performance varies with different CPU architecture and memory allocation. For instance, the computational node in Edison is the NUMA node containing 24 cores separated by two portions

connected through intersocket. NERSC announced that the best choice per NUMA node can be 1-4 MPI processes with 12-3 OpenMP threads for each node in Edison.

Fortran uses different compiler directives as C:

```
1 !$omp <construct> [clause1 clause2 ...]
```

For the hybrid routine, the OpenMP parallel parts are mainly involved with "do-loops". Not only the GF value calculations and the element-by-element multiplications are computed inside the "do-loops", but also the FFTs provided by FFTPACK are managed by "do-loops". In fact, the "do-loops" are maintained by two "loop-controllers" in the hybrid parallel routine: Open MPI task and OpenMP threads.

The "do-loops" inside each MPI process are further parallelized with OpenMP API. The parallelization is similar to the aforementioned OpenMP routine as:

- The parallelization of the efficient IGF calculations
- The parallelization of FFTs and DCTs
- The parallelization of multiplication in Fourier domain
- The parallelization of the data transpose

Substituting OpenMP threads for MPI parallelism is an excellent strategy for a type of modern supercomputers. The newest supercomputer system, named Cori, in NERSC will have the second generation of the Intel Xeon Phi co-processor products called the Knights Landing (KNL) MIC architecture which has a strong potential in the HPC application.

Benefiting from the Intel's widely available CPU market, the co-processors can easily be made to co-operate with CPUs in different strategies. Two major strategies are: the offload mode and the MPI mode. The offload mode involves localized changes to a program and generally used for finer grained parallelization. In contrast, the MPI mode requires scattered changes in a program and often used for coarse grained parallelization.

As an attempt and preparation, the future MPI+OpenMP routine should differ from the attached OpenMP routine under the goals of reducing the need for replicated data transport and adding vector parallelism. In detail, we use efficient batch FFT implementation and reduce the matrix transpose times among processes.

First, the FFTs are computed by the FFTW library rather than the former FFTPACK library. The SIMD FFTs from FFTW library are available for higher speed-up as shown in Figure. 5.18 .

Second, the data transposes should be avoided as much as it can and the processing of a transpose should be more efficient. For the first scheme, an attempt is the replacement of the 2D coordinate of processes to a 1D coordinate in the program pattern. Each process stores the 2D slices and the ordering 2D slices construct the whole 3D structure by combining all 2D slices, thus the transpose for the first 2D is maintained in the same process. For the second scheme, the hot problem of

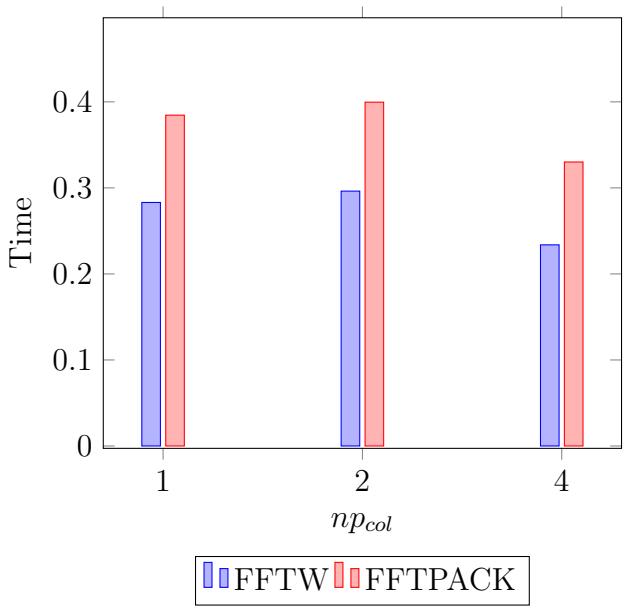


Figure 5.18: Comparison of FFTs in FFTW library and FFTPACK with $np = 4$, $N_w = 128$

efficient global communication across processes in HPC field is met. The simultaneous implementation and SIMD should also be considered for the data transpose.

Furthermore, every language is a type of column-major or row-major language, e.g. Fortran is a column-major language. The operations within dimensions of multidimensional data are different based on the difference of row-column management. In the author's view, if the data operations between column and row are hugely different, the control of the data is not ideal and calls for improving. The operations of the column direction and the row direction should be balanced as much as possible for either language. If the current capability is beyond the need, new technologies such as MIC structure should improve the multi-operation capability for such memory control.

6 Verification with examples, applications and discussions

The efficient Poisson solver has been introduced by the theoretical method (Chapter 3, Chapter 4), and further programmed in practice by the state-of-the-art HPC technologies (Chapter 5). Further validations and applications are presented in this chapter for the proposed novel efficient approach.

First, three examples are introduced for the numerical verification in Section 6.1: an ideal uniform charged sphere bunch, an ideal uniform charged ellipsoid bunch, and an ideal Gaussian distributed charged bunch.

Second, the efficient IGF integrals (CIGF and RIGF) are verified through these test examples.

Additionally, the novel fast convolution routine combining with the efficient IGF integral is studied for accuracy. The convergence study of the novel Poisson solver is presented by refining the discretization.

For applications, the validations are done by comparing the novel Poisson solver and a commonly used Poisson solver. Two beam dynamic simulations through two different simulation codes, MOEVE-PIC and IMPACT, are given.

On the efficiency improvement side, the execution times are studied for both workstation and supercomputer with regard to all of the new technologies: pure single CPU routine for a workstation, OpenMP shared-memory parallelization, the hybird parallel routine with OpenMP+CUDA for a workstation, pure Open MPI routine for a supercomputer, and the hybrid Open MPI+OpenMP routine for a new architecture of supercomputer.

6.1 Verification examples

For all the examples we use the notations, η_φ and $\eta_{\mathbf{E}}$ to measure the relative errors between the simulation results and analytic solutions, which we used as follows:

$$\begin{aligned}\eta_\varphi(i,j,k) : &= \frac{|\varphi_{i,j,k} - \varphi_{\text{true}_{i,j,k}}|}{\max_{i,j,k} |\varphi_{\text{true}_{i,j,k}}|}, \text{ and } \|\eta_\varphi\|_{\text{inf}} := \max_{i,j,k}(\eta_\varphi(i,j,k)),\\ \eta_{\mathbf{E}}(i,j,k) : &= \frac{||\mathbf{E}_{i,j,k} - \mathbf{E}_{\text{true}_{i,j,k}}||_2}{\max_{i,j,k} ||\mathbf{E}_{\text{true}_{i,j,k}}||_2}, \text{ and } \|\eta_{\mathbf{E}}\|_{\text{inf}} := \max_{i,j,k}(\eta_{\mathbf{E}}(i,j,k)).\end{aligned}$$

Additionally, we optionally study two other norms of relative errors for the potential:

$$\|\eta_\varphi\|_2 := \sqrt{\sum_{i,j,k} \eta_\varphi(i,j,k)^2}/N_p,$$
$$\|\eta_\varphi\|_1 := \sum_{i,j,k} |\eta_\varphi(i,j,k)|/N_p.$$

Here, the notations are, $\eta_\varphi(i,j,k)$, $\varphi_{i,j,k}$ and $\varphi_{\text{true}_{i,j,k}}$ as the relative error of the potential at index (i,j,k), the computed potential at index (i,j,k) and the true potential for the same index, respectively. In the same way, we have the notation for the electric field: $\eta_{\mathbf{E}}(i,j,k)$, $\mathbf{E}_{i,j,k}$ and $\mathbf{E}_{\text{true}_{i,j,k}}$.

This section contains three different verified examples for Poisson's equation based on different distributions of charge density $\rho(x,y,z)$ (or $\rho(i,j,k)$ in discretized situation), i.e. uniform sphere shape, uniform ellipsoid shape, and with Gaussian distribution.

Test Case 1: ideal uniform charged sphere bunch

In this case, Poisson's equation is described as:

$$-\Delta\varphi = \tfrac{\varrho}{\varepsilon_0} = \begin{cases} \frac{3}{4} \cdot \frac{Q}{\pi\varepsilon_0 R^3} & \text{for } \|r\|_2 \le R, \\ 0 & \text{otherwise.} \end{cases} \tag{6.1}$$

Solution:

$$\varphi(r) = \begin{cases} \frac{Q}{4\pi\varepsilon_0 R} \cdot (\frac{3}{2} - \frac{r^2}{2R^2}) & \text{for } r \le R, \\ \frac{Q}{4\pi\varepsilon_0 r} & \text{otherwise.} \end{cases} \tag{6.2}$$

The parameters are referenced from the table:

Signs	Notations	Values
Q	charge	-1 nC
R	radius of the sphere	2.2 mm
r	distance	$\sqrt{x^2+y^2+z^2}$
Ω	computational domain	$[-4.4, 4.4]^3$ mm^3
ε_0	permittivity in vacuum	8.85419×10^{-12} F·m^{-1}

Test Case 2: an ideal uniform charged ellipsoid bunch [26] [17].

In this case, Poisson's equation is described as:

$$-\Delta\varphi = \frac{\varrho}{\varepsilon_0} = \begin{cases} \frac{3}{4}\cdot\frac{Q}{\pi\varepsilon_0 abc} & \text{for } \frac{x^2}{a^2}+\frac{y^2}{b^2}+\frac{z^2}{c^2}\le 1, \\ 0 & \text{otherwise.} \end{cases} \tag{6.3}$$

Solution:

$$\varphi(r) = \begin{cases} \frac{3Q}{16\pi\varepsilon_0}\cdot(-Ax^2-By^2-Cz^2+D) & \text{for } \frac{x^2}{a^2}+\frac{y^2}{b^2}+\frac{z^2}{c^2}\le 1, \\ \frac{3Q}{16\pi\varepsilon_0}\cdot(-A_\lambda x^2-B_\lambda y^2-C_\lambda z^2+D_\lambda) & \text{otherwise.} \end{cases} \tag{6.4}$$

The parameters for the analytical solution are expressed as:

$$\begin{aligned} A &= \int_0^\infty \frac{ds}{(a^2+s)\sqrt{(a^2+s)(b^2+s)(c^2+s)}}, \\ B &= \int_0^\infty \frac{ds}{(b^2+s)\sqrt{(a^2+s)(b^2+s)(c^2+s)}}, \\ C &= \int_0^\infty \frac{ds}{(c^2+s)\sqrt{(a^2+s)(b^2+s)(c^2+s)}}, \\ D &= \int_0^\infty \frac{ds}{\sqrt{(a^2+s)(b^2+s)(c^2+s)}}, \\ A_\lambda &= \int_\lambda^\infty \frac{ds}{(a^2+s)\sqrt{(a^2+s)(b^2+s)(c^2+s)}}, \\ B_\lambda &= \int_\lambda^\infty \frac{ds}{(b^2+s)\sqrt{(a^2+s)(b^2+s)(c^2+s)}}, \\ C_\lambda &= \int_\lambda^\infty \frac{ds}{(c^2+s)\sqrt{(a^2+s)(b^2+s)(c^2+s)}}, \\ D_\lambda &= \int_\lambda^\infty \frac{ds}{\sqrt{(a^2+s)(b^2+s)(c^2+s)}}, \\ \text{where } \lambda &= \text{the greatest root of } f(s)=0, \text{and} \\ f(s) &= \frac{x^2}{a^2+s}+\frac{y^2}{b^2+s}+\frac{z^2}{c^2+s}-1. \end{aligned}$$

A "cigar" shape ideal ellipsoid bunch is chosen as the exact verified example: assume $a = b$, and $c > a$. The above parameters are calculated by *Mathematica* [97], and

read as follows:

$$
\begin{aligned}
A = B &= \frac{c}{a^2\left(c^2-a^2\right)} - \frac{\operatorname{arccosh}\left(\frac{c}{a}\right)}{\left(c^2-a^2\right)^{3/2}},\\
C &= \frac{2\left(c\sqrt{c^2-a^2}\operatorname{arccosh}\left(\frac{c}{a}\right) - a^2 + c^2\right)}{c\left(a^2-c^2\right)^2},\\
D &= \frac{2\cosh^{-1}\left(\frac{c}{a}\right)}{\sqrt{c^2-a^2}},\\
A_\lambda = B_\lambda &= \frac{-2\sqrt{\left(c^2-a^2\right)\left(c^2+\lambda\right)} - \frac{\left(a^2+\lambda\right)\left(\log\left(\sqrt{-\frac{c^2+\lambda}{a^2-c^2}}-1\right)\right)}{\sqrt{-\frac{c^2+\lambda}{a^2-c^2}}+1}}{2\left(c^2-a^2\right)^{3/2}\left(a^2+\lambda\right)},\\
C_\lambda &= \frac{1}{\left(a^2-c^2\right)^2\sqrt{c^2+\lambda}}\left(-\sqrt{-\left(a^2-c^2\right)\left(c^2+\lambda\right)}\log\left(\sqrt{-\frac{c^2+\lambda}{a^2-c^2}}-1\right)\right.\\
&\quad \left.+2a^2-2c^2+\sqrt{-\left(a^2-c^2\right)\left(c^2+\lambda\right)}\left(\log\left(\sqrt{-\frac{c^2+\lambda}{a^2-c^2}}+1\right)\right)\right),\\
D_\lambda &= \frac{\log\left(\sqrt{-\frac{c^2+\lambda}{a^2-c^2}}+1\right)}{\sqrt{c^2-a^2}\left(\sqrt{-\frac{c^2+\lambda}{a^2-c^2}}-1\right)},\\
\text{where } \lambda &= \frac{1}{2}\left(\sqrt{\left(-a^2-c^2+x^2+y^2+z^2\right)^2+4\left(-a^2c^2+a^2z^2+c^2x^2+c^2y^2\right)}\right.\\
&\quad \left.-a^2-c^2+x^2+y^2+z^2\right).
\end{aligned}
$$

The bunch parameters are referenced from the table:

Signs	Notations		Values
Q	charge		-1 nC
$a(b)$, c	ellipsoid size	$[-2.2, 2.2]$, $30[-2.2, 2.2]$	mm
Ω	computational domain	$[-4.4, 4.4]^2 \times 30[-4.4, 4.4]$	mm^3
ε_0	permittivity in vacuum	8.85419×10^{-12}	Fm^{-1}

Test Case 3: an ideal Gaussian distributed charged bunch.
In this case, Poisson's equation is described as:

$$-\Delta\varphi = \frac{\varrho}{\varepsilon_0} = \frac{Q}{\varepsilon_0\sigma^3\sqrt{2\pi}^3}\exp^{-\frac{r^2}{2\sigma^2}}, \tag{6.5}$$

Solution:

$$\varphi(r) = \frac{1}{4\pi\varepsilon_0}\frac{Q}{r}\mathrm{er}f(\frac{r}{\sqrt{2}\sigma}). \tag{6.6}$$

The parameters are referenced from the table:

Signs	Notations	Values
Q	charge	-1 nC
σ	standard deviation	0.000875
Ω	computational domain	$[-4.4, 4.4]^3$ mm^3
r	distance	$\sqrt{x^2+y^2+z^2}$
erf	(Gauss) error function	
ε_0	permittivity in vacuum	8.85419×10^{-12} F·m^{-1}

6.2 Numerical verification of the efficient IGF integrals for the Poisson solver

In Chapter 3, the efficient IGF integrals are given, analyzed and verified in a theoretical way. The numerical verification is presented in the form of numerical error studies of the corresponding Poisson solver. The efficient IGF integral, CRIGF, is tested by two sub-verifications: verification of CIGF integral, and verification of RIGF integral. The separated verification makes clear how the integrals influence the final result of the Poisson solver. The discrete convolution of the Poisson solver uses the classical HandE's routine.

6.2.1 Numerical verification of the CIGF integral

The CIGF integral is verified through Test Case 1, which the bunch parameters are set as:

Signs	Notations	Values
Q	charge	-1 nC
R	radius of the sphere	2.2 mm
Ω	computational domain	$[-2.2 \times 5, 2.2 \times 5]^3$ mm^3

Also the domain-bunch ratio α is 5, i.e. $\alpha_x = \alpha_y = \alpha_z = 5$. $C_w = [(1+\alpha_w)/2\alpha_w N_w]$ for w in $\{x, y, z\}$.

Table 6.1: Comparison of the relative errors of IGF and CIGF by $\| \|_{\text{inf}}$

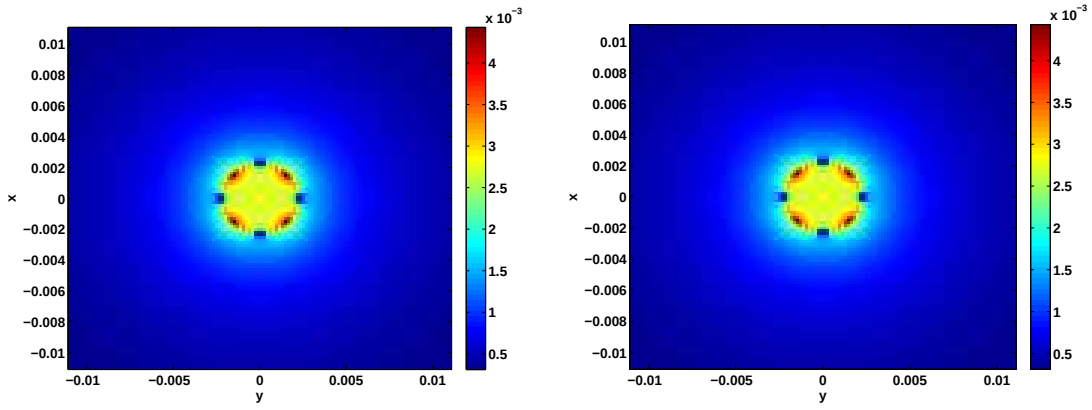
N	IGF $\|\eta_\varphi\|_{\text{inf}}$	IGF $\|\eta_{\mathbf{E}}\|_{\text{inf}}$	CIGF $\|\eta_\varphi\|_{\text{inf}}$	CIGF $\|\eta_{\mathbf{E}}\|_{\text{inf}}$
32	0.0587	0.0619	0.0587	0.0619
64	0.0130	0.0334	0.0130	0.0334
128	0.0045	0.0290	0.0045	0.0290

We further check the other two norms of relative errors:

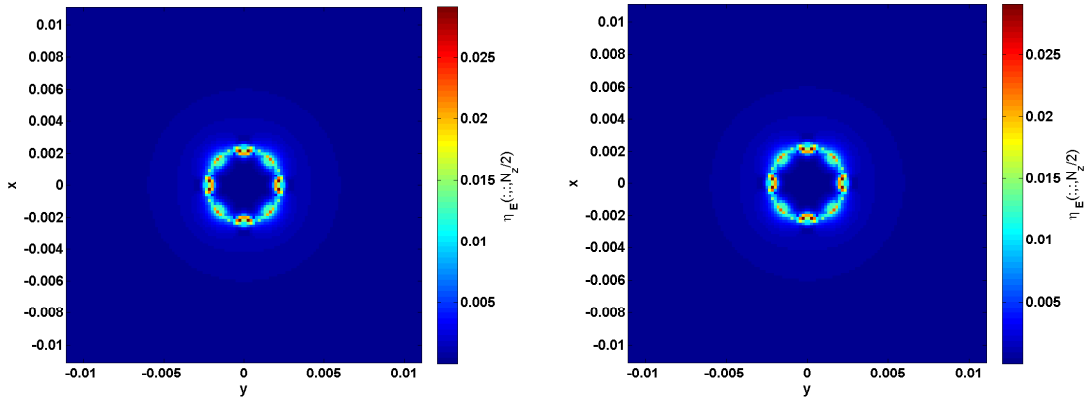
Table 6.2: Comparison of the relative errors of IGF and CIGF by $\| \ \|_1$ and $\| \ \|_2$

N	IGF $\|\eta_\varphi\|_1$	IGF $\|\eta_\varphi\|_2$	CIGF $\|\eta_\varphi\|_1$	CIGF $\|\eta_\varphi\|_2$
32	1.367e-02	8.336e-05	1.367e-02	8.336e-05
64	2.448e-03	5.273e-06	2.448e-03	5.273e-06
128	5.160e-04	4.004e-07	5.160e-04	4.004e-07

In addition, four figures are given to visualize the relative errors η_φ (top) and $\eta_{\mathbf{E}}$ (bottom). The 2D figures are plotted by cutting the slice along the middle of the computational domain in z direction.



CIGF. $\eta_\varphi(:,:,\frac{N_z}{2})$ IGF. $\eta_\varphi(:,:,\frac{N_z}{2})$



CIGF. $\eta_{\mathbf{E}}(:,:,\frac{N_z}{2})$ IGF. $\eta_{\mathbf{E}}(:,:,\frac{N_z}{2})$

Figure 6.1: Comparison of CIGF and IGF by $\eta_\varphi(:,:,:)$ and $\eta_{\mathbf{E}}(:,:,:)$ for an ideal uniform sphere beam.

The norms of the numerical errors of the Poisson solver induced by IGF and CIGF integrals are the same. A detailed comparison of the two was also done element by element and it suggests the numerical errors are the same.

6.2.2 Numerical verification of the RIGF integral

The RIGF integral is verified through Test Case 2, where some of the parameters are reset as shown in the following table.

Signs	Notations		Values
Q	charge		-1 nC
Ω	computational domain	$[-4.4, 4.4]^2 \times \beta[-4.4, 4.4]$	mm^3
$a(b)$, c	ellipsoid size	$[-2.2, 2.2]$, $\beta[-2.2, 2.2]$	mm

In the table, β is expressed as the longitudinal-to-transverse ratio of the ellipsoid bunch. The value is set as 10, 30 and 100 to compare different scaling results. The extreme conditions of large β may present a bunch in the rest frame after a Lorentz transform.

In Figure 6.2, the $\|\eta_\varphi\|_{\text{inf}}$ values are plotted when the R_z varies from 0 to 32 ($N_w = 64$). The remaining half of the points are not included in the figures since the line's trend is foreseen: smooth, stable and convergent. The plotted errors show the numerical solution is not sensitive to shift of different GF integrals as described in Section 3.3. Although GF is a strong decreasing function, the switching does not break the continuity property. This is because the accuracy shifting of GF integral values is bounded by the determined strategies. There is a slight drop in the error values for the initial values of R_z when β is large (e.g. 100). After that the errors recover again and converge to IGF integral's result. This behavior is not fully understood now, but the provided R_z determination strategies always override these points in practice.

For the adapted relative error determination strategy: $s = 1$ (see Section 3.3.1) is chosen for the switching between GF and IGF for the high accuracy purpose. The parameter R_z chosen by the adapted algorithm appears to be constant for the three different βs, i.e. $R_z = 13$.

For the time (distance) error determination strategy: $R_z = [\frac{N_z+1}{s_z}]$, $s_z = 4$ to 8 ($R_z = 16$ to 8) is enough to reflect a reasonable simulation result, as shown in Figure 6.2.

In total, the numerical verification of the efficient IGF integrals shows a reliable result: the CIGF integral fully agrees with the IGF integral on the solution of Poisson's equation; on the RIGF integral side, the solution providing by the RIGF integral stably converges to the reference solution with IGF integral. The adapted algorithm for choosing the parameter R_z is reliable and fast while the time (distance) determination strategy is simple and direct.

The CRIGF integral, as the combination integral, is verified by the above two subverifications. The efficient IGF integrals are integrated into the novel Poisson solver, and further verified by comparing with the commonly used Poisson solver in the next section.

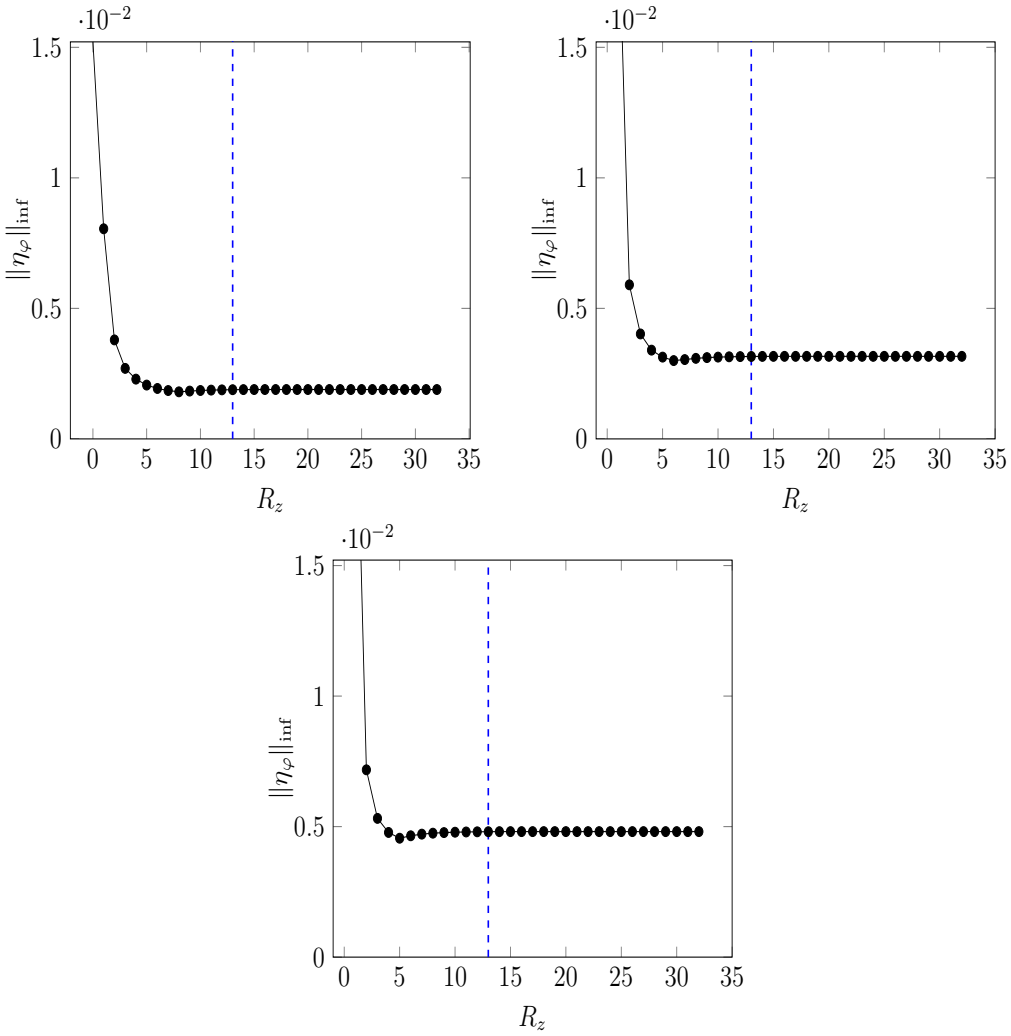


Figure 6.2: The RIGF integral study: fixing the mesh numbers $N_w = 64$ for Test Case 2, while $\beta = 10$(top left), $\beta = 30$ (top right) and $\beta = 100$ (bottom).

6.3 Verification of the novel Poisson solver

In Chapter 4, the novel discrete convolution routine is presented. The routine theoretically does not differ from the classical Poisson solvers, either the trivial FFT routine or the HandE's routine. The routine combined with the aforementioned efficient IGF integrals is verified in two aspects: the error convergence trend (the aforementioned three different error norms are all compared) with finer grids, and the local errors plotted on the cutting planes in comparison with HandE's routine.

All three test cases are tested: Test Cases 1 and 2 contrast the length scaling while Test Cases 1 and 3 differ with regard to the continuous property of the r.h.s function.

6.3.1 Convergence study of the novel Poisson solver

For Test Cases 1 and 2: the domain-bunch-ratio is 1. The domain size is reset in the following table as:

Signs	Notations		Values
Q	charge		-1 nC
$\Omega_{\text{Case 1}}$	computational domain	$[-2.2, 2.2]^2 \times [-2.2, 2.2]$	mm^3
$\Omega_{\text{Case 2}}$	computational domain	$[-2.2, 2.2]^2 \times 30[-2.2, 2.2]$	mm^3

For Test Case 3, the parameters remain the same as their initial values in Section 6.1.

The potential convergence of the novel Poisson solver is expanded by means of increasing the mesh number N_w for the constant domain. N_w is from 8 to 344 with the step value 8. In total, 43 points are plotted in each sub-figure of Figure 6.3. The plots are in log-log axes: $N_x N_y N_z$ (total mesh number) is in x axis while the magnitude of relative errors is in y axis.

For Test Cases 1 and 2, they show a similar behavior: the convergence with increasing mesh number N_w does succeed. However, the convergence is not as stable nor "continuous" as the Gaussian distribution of Test Case 3.

For Test Case 3, the plots of all three norms show a stable decreasing linear line in the log-log axes.

As an aspect of explanations: the computational domain sizes for both Test Cases 1 and 3 are the same. It means that the only difference between the two is the charge density, it is a continuous function for Test Case 3 (Gaussian bunch), but a flat-top function as Eq. (6.1) for Test Case 1 (charged sphere bunch). The poor property of the r.h.s function reflects also a weak property in sensitivity of the computed results, a wider convergent band region, and the convergence slope is lower. Furthermore, the figures of Test Cases 1 and 2 show that the convergence does not break from the sensitivity. The three norms used for testing cases show the same trend of convergence.

We cannot expect a better solution if the r.h.s function has a poor property in comparison with the situation where the r.h.s function has a good property. This is true for all numerical methods as the refining discretized grid actually represents the original function's property. The classical HandE's routine also has the same behavior for Test cases 1 and 2.

A recommendation to be drawn from the convergence study is: the convergence study reveals the advice of keeping a constant spatial discretization during a bunch tracking. Additionally, some more numerical tools may be an asset.

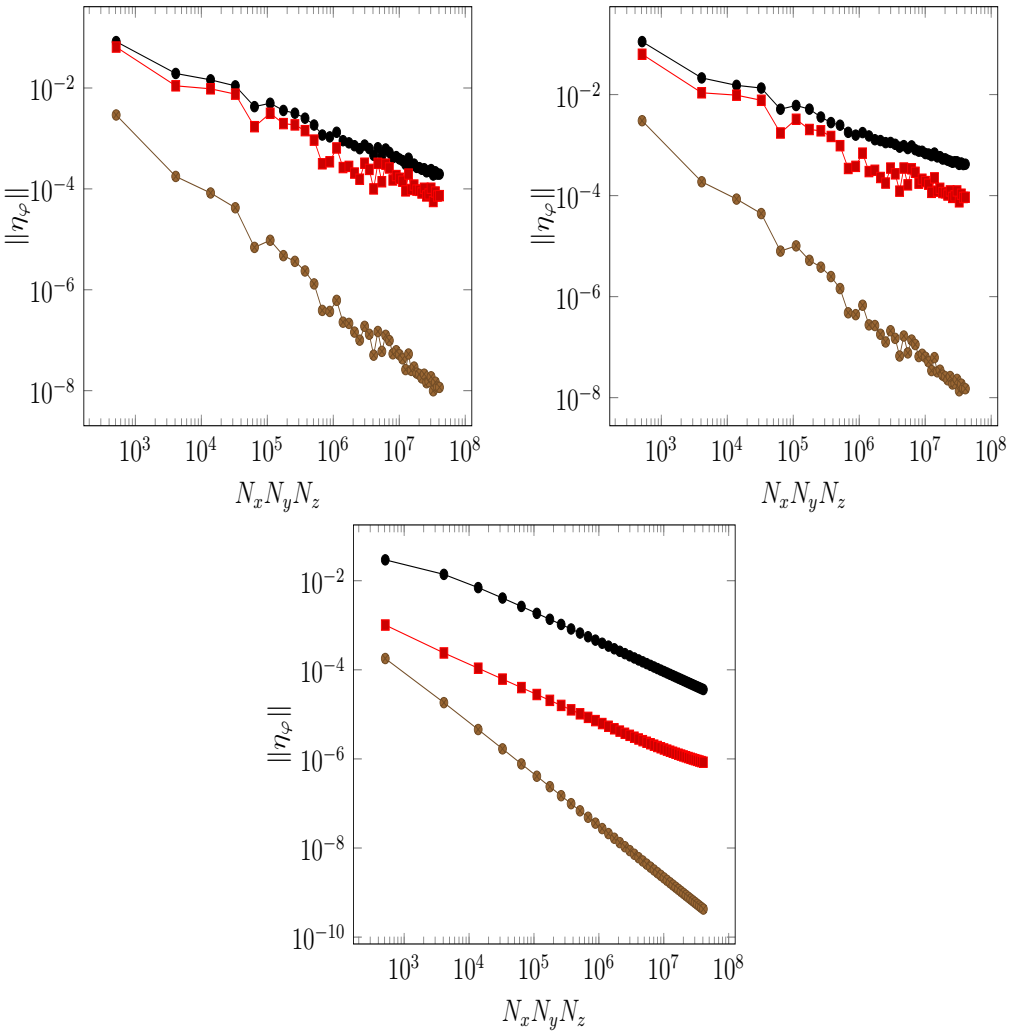


Figure 6.3: Convergence study for the novel Poisson solver with increasing mesh number N_w: Test Case 1 (top left), Test Case 2 (top right), Test Case 3 (bottom). The $\|\eta_\varphi\|$ is differed by : $\|\eta_\varphi\|_{\text{inf}}$ (black curve), $\|\eta_\varphi\|_1$ (red curve) and $\|\eta_\varphi\|_2$ (brown curve).

6.3.2 Numerical verification of the novel efficient method

The HandE's routine is chosen as the comparable convolution routine; both GF and IGF integrals are applied to show the bounds of the efficient GF integrals. The comparisons are for both aspects of potentials and electric field.

All three test cases are studied for the numerical verification while only Test Case 2 is included in this section. The other two test cases are given in Appendix C in the same way.

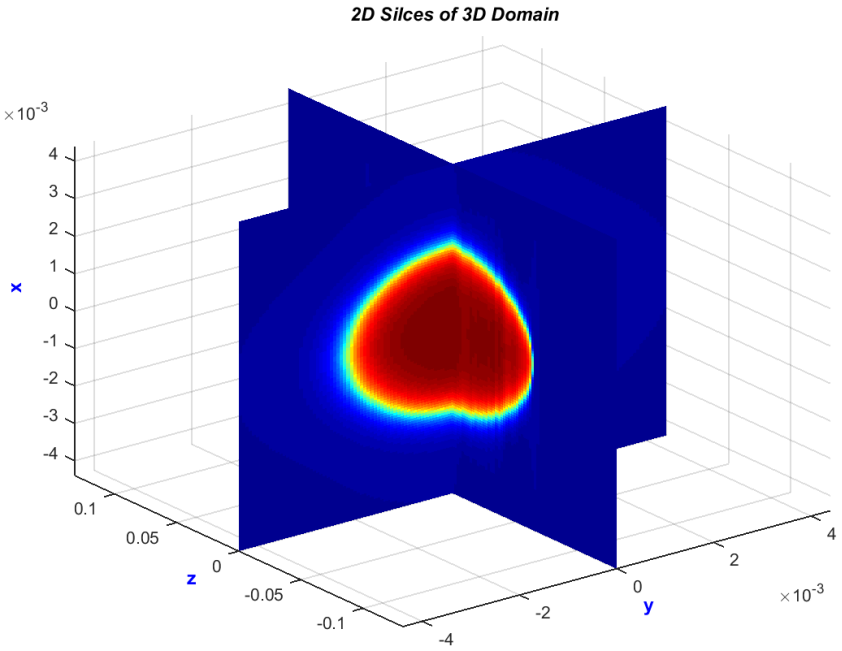


Figure 6.4: Comparison of $\eta_\varphi(i,j,k)$ of Test Case 2: at plane $(:, N_y/2, :)$ (left view) and $(:, :, N_z/2)$ (right view) inside the 3D computational domain for the GF integral.

Simulate an ideal uniform charged ellipsoid bunch

The bunch parameters of Test Case 2 are the same as their initial values in Section 6.1.

Two cutting planes, $(:, N_y/2, :)$ plane and $(:, :, N_z/2)$ plane, are chosen to show the error study. The positions of the two planes inside the 3D computational domain are plotted in Figure 6.4. Furthermore, Figure 6.5 and Figure 6.6 present the η_φ values and $\eta_{\mathbf{E}}$ values by plotting the two cutting planes directly: left column for $(:, N_y/2, :)$ planes and right column for $(:, :, N_z/2)$ planes. The different GF integrals are recognized as GF integrals (top row), efficient IGF integrals (middle row), IGF integrals (bottom row).

For Test Case 2, η_φ (also $\eta_{\mathbf{E}}$) of GF integrals is not plotted with the same color scaling as the other two integrals since the magnitude of GF integrals differs from the other two integrals. The efficient IGF and IGF integrals present nearly the same figures.

The large η_φ values locate around the center of the bunch area, whereas the large $\eta_{\mathbf{E}}$ values locate in an area around this boundary of the bunch. In comparison, the GF integrals solve larger errors of potential within the area of the bunch, whereas the GF integrals solve larger errors of the electric field around the boundary of the bunch.

The local errors from both figures show that the novel Poisson solver with efficient IGF integral agrees with the classical Poisson solver with IGF integral.

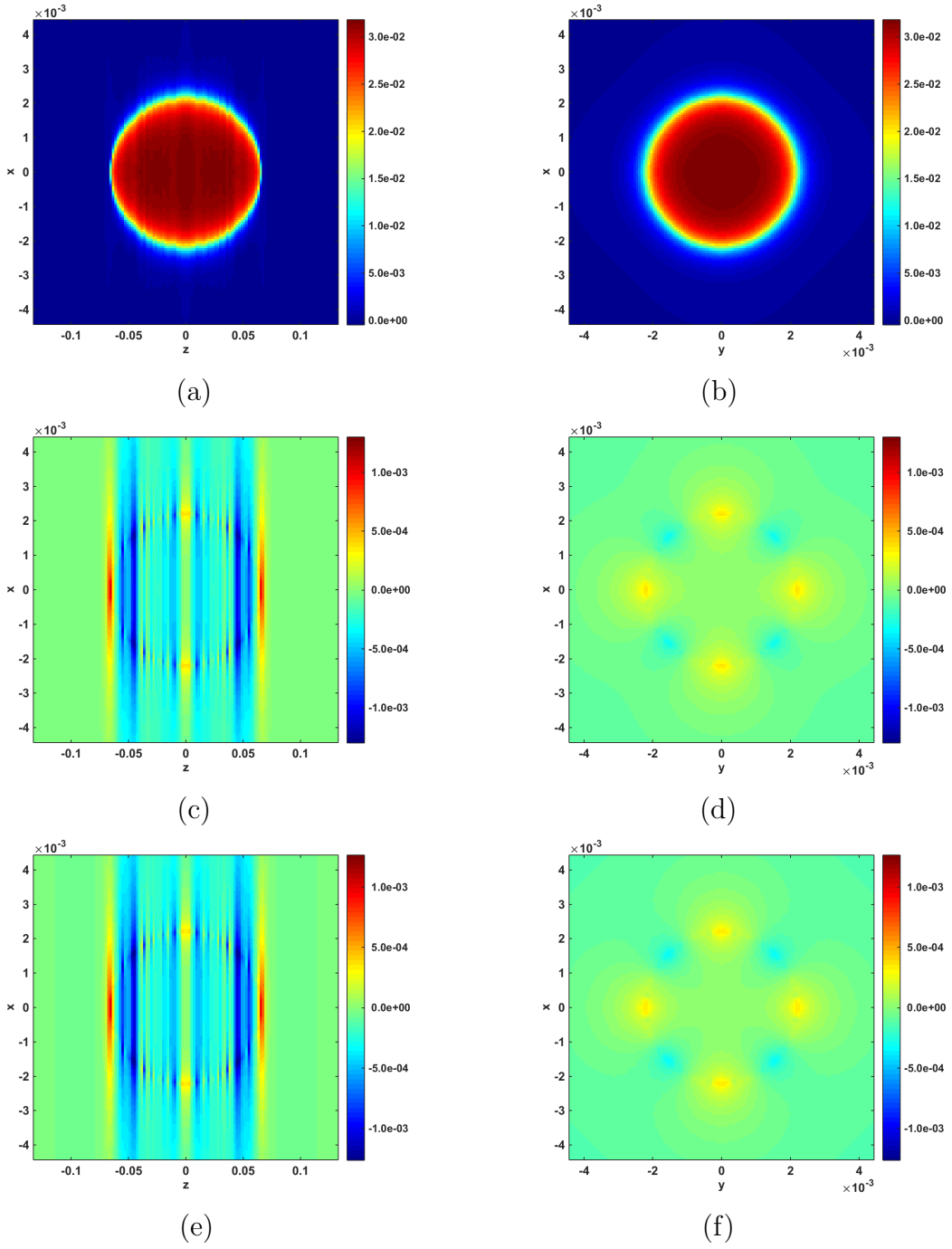


Figure 6.5: Comparison of $\eta_\varphi(i,j,k)$ of Test Case 2: at plane $(:, N_y/2, :)$ (left) and $(:, :, N_z/2)$ (right) for the GF integral (a) (b), efficient IGF (c) (d), IGF (e) (f).

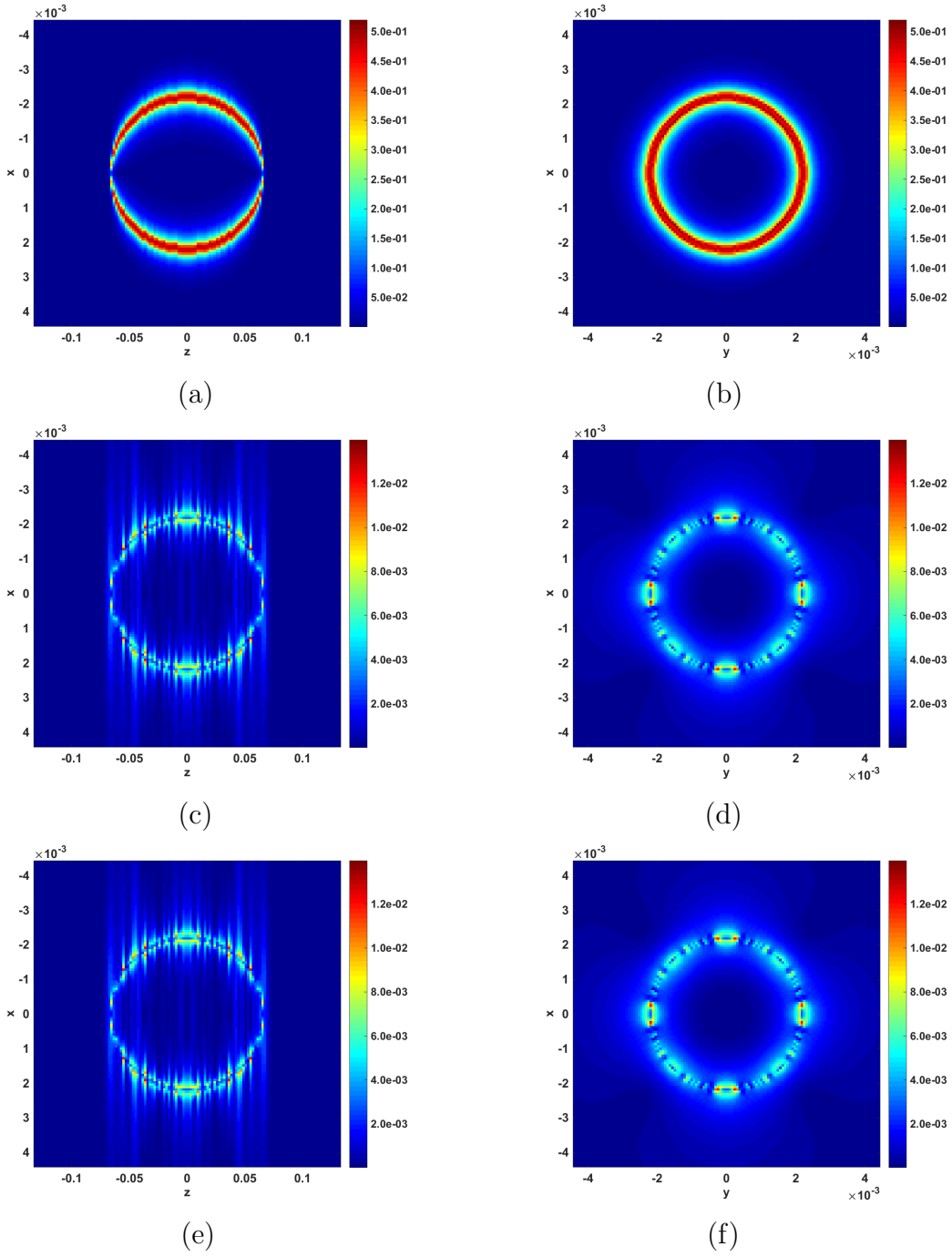


Figure 6.6: Comparison of $\eta_{\mathbf{E}}(i,j,k)$ of Test Case 2: at plane $(:, N_y/2, :)$ (left) and $(:, :, N_z/2)$ (right) for the GF integral (a) (b), efficient IGF (c) (d), IGF (e) (f).

6.4 Applications

The novel Poisson solver is already integrated into the MOEVE-PIC and IMPACT software packages. The validation is further considered for the application of beam dynamics simulations. The verifications in the preceding sections were all successfully done within the Poisson solver itself. However, the Poisson solver is repeated hundreds of thousand times with each discretized time step in beam dynamics simulations. The simulation combining the Poisson solver and other calculations in the PIC module is complex. In particular, potential and electric field resulting from the Poisson solver are only the intermediate quantities for beam dynamics study in the accelerator community. Bunch size and emittance are the parameters for the measurement in experiment and analysis in theory. The simulation code computes these parameters for comparisons with experiment and theory.

Two sub-tasks are implemented for two software packages, respectively: the serial implementation in MOEVE-PIC (Section 6.4.1); the parallel implementation in IMPACT (Section 6.4.2). By comparing the accelerator parameters, i.e. bunch size and emittance, the goal of agreement between the novel Poisson solver and a commonly used Poisson solver is achieved for the two sub-tasks.

6.4.1 Tracking a bunch in a beam line with MOEVE-PIC

MOEVE-PIC software package originally applies multigrid method for Poisson solver in bunch tracking simulations, electron cloud, and ion cloud effects studies. Now, a couple of FFT Poisson solvers have been integrated into the package. The integrated solvers combined with the existed PIC module provide a comparable result.

An electron Gaussian bunch whose parameter profile is listed in Table 6.3 is chosen as the application example.

Table 6.3: Initial parameters of the tracked bunch for MOEVE-PIC.

Bunch parameters	
number of marco particles	20,000
beam energy	0.800 MeV
beam charge	-1.000 nC
normalized emittance	1.000 π mrad mm
bunch length	22.5 mm
rms bunch radius	0.75 mm

The time step δt is set as 1 ps and 1000 total steps are applied. Without any external electric field, the bunch is tracked inside a beam line. The transverse emittance and the rms bunch size of the tracked bunch are plotted in Figure 6.7.

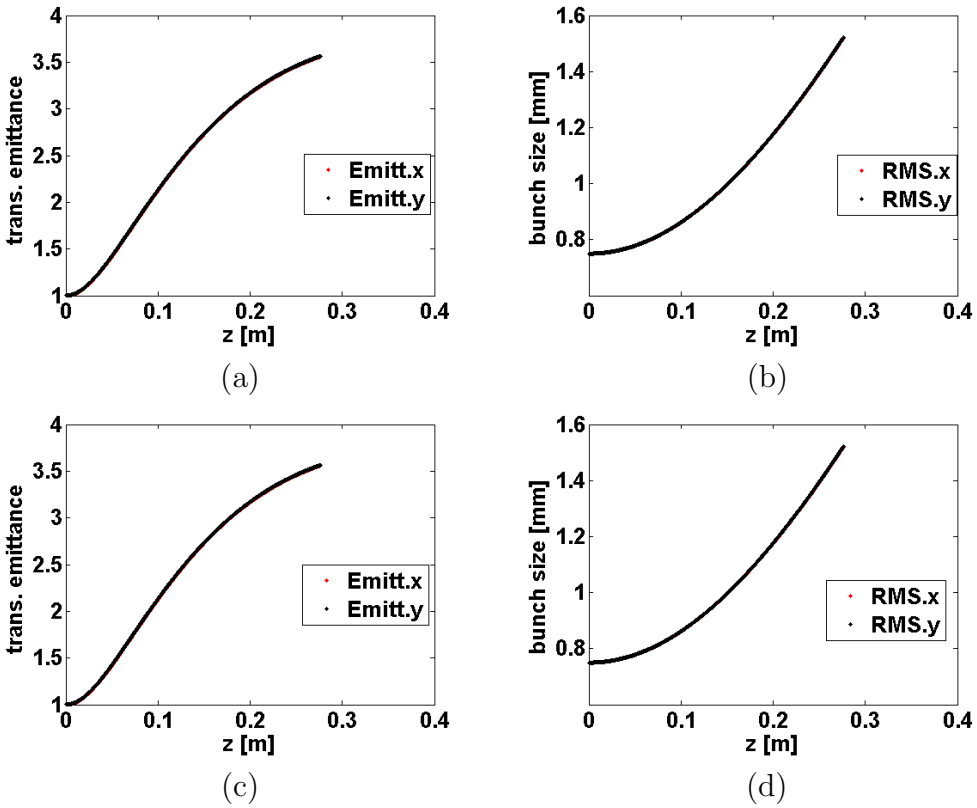


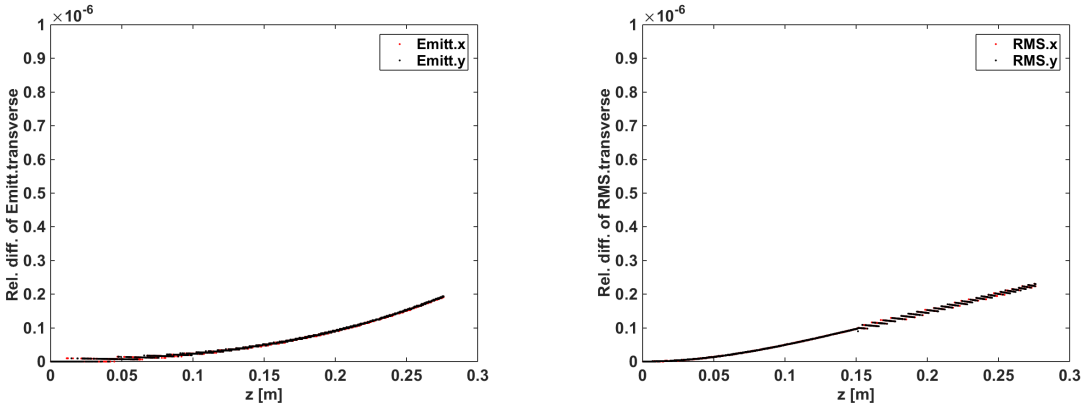
Figure 6.7: Comparison of emittance growth (left) and bunch size (right) increase for MOEVE-PIC example: efficient IGF together with novel fast convolution routine (a) (b), IGF together with HandE's routine(c) (d).

The relative difference study is presented in Figure 6.8. As shown, both relative differences of emittance and rms bunch size have a slight increase. However, the relative differences for both are on the magnitude of 10^{-6}, which is very small. In next section, the relative difference is studied with an external magnetic field for a detailed simulation.

6.4.2 Compare simulation results within IMPACT

The novel Poisson solver has also been integrated into the IMPACT package for parallel simulations. More simulation results can be compared with different accelerator settings. The comparison is against the HandE's routine with IGF integrals, but in parallel.

A virtual accelerator is set for the purpose of this comparison: a long bunch (Bunch profile is in Table 6.4) with uniform distribution in transverse direction and Gaussian distribution in longitudinal direction is generated by an RF electron gun. The starting point of the comparison is a short distance (2 meters) after the electron gun. The bunch is tracked for a certain distance (10 meters) in simulation without extra electric field but with focusing and defocusing magnetic field. The whole simulation



(a) (b)

Figure 6.8: Comparison of the relative difference (rel. diff.) of transverse emittance (Emmit.transverse) (left) and rms bunch size (RMS.transverse) (right) between efficient IGF and IGF integrals for a bunch tacking example in MOEVE-PIC.

procedure corresponds to the electron gun studies for the accelerators such as Free Electron Lasers.

Some critical simulation parameters are $\delta t = 1$ps, distance=10m, $N_w = 64$. Specific for the parallel implementation: $np = 64$, and the processors' coordinates are $np_{\text{col}} = 8$, $np_{\text{row}} = 8$. For the whole simulation, the only difference is the Poisson solvers.

Table 6.4: Initial parameters of the tracked bunch for IMPACT.

Bunch parameters	
number of marco particles	160,000
beam energy	0.511 MeV
beam charge	-1.000 nC
longitudinal bunch length	2.82 mm
transverse bunch length (x, y)	$(1.66, 1.66)$ mm

In Figure 6.9, the rms bunch size is plotted for both solvers: the red lines represent the existing FFT Poisson solver in IMPACT (named as IGF in the plots); the green lines represent the parallelized novel Poisson solver (named as RIGF in the plots). In Figure 6.11, the same notations are chosen as in Figure 6.9. The rms emittance is plotted for both solvers. From the two figures, the agreements of bunch size and emittance for both Poisson solvers are clear: they match very precisely.

In addition, the relative difference study for IMPACT is presented in Figure 6.10 and Figure 6.12. The trend of the relative difference changes with different sections of the accelerator for both emittance and rms bunch size, since the external focusing and defocusing magnetic field is added. The magnitude of relative difference is in the order

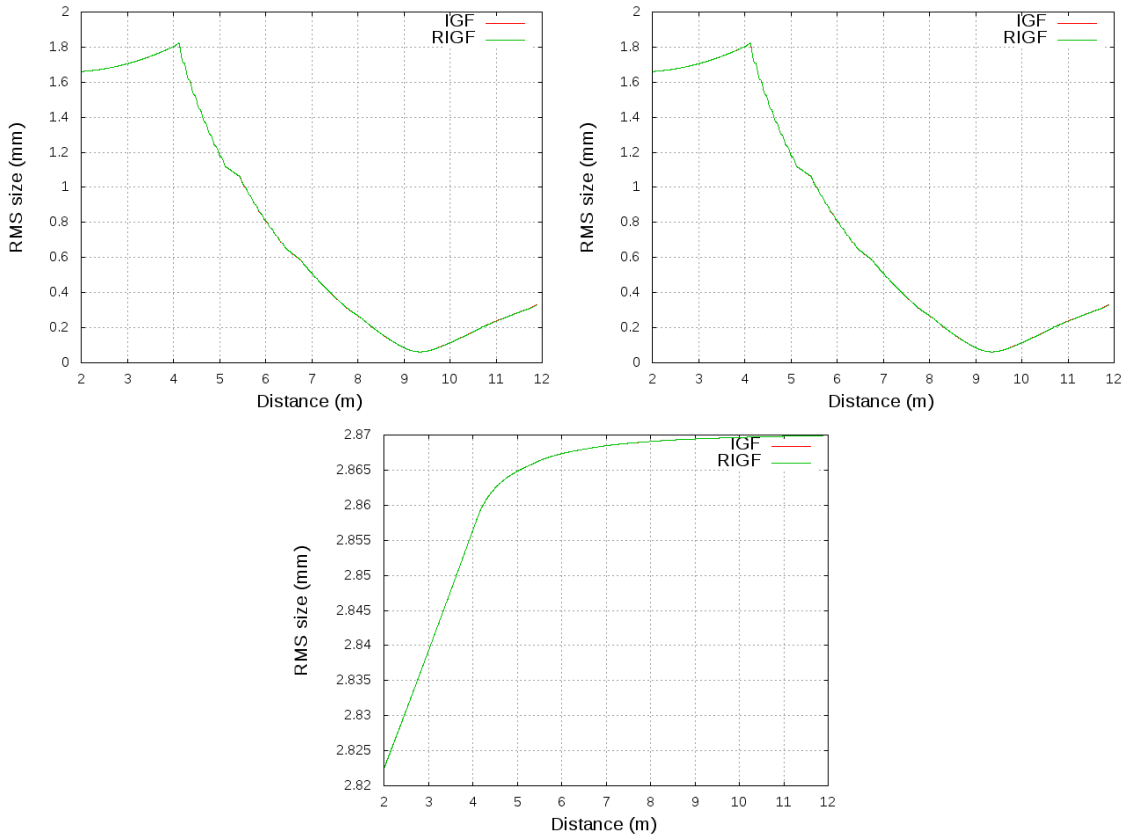


Figure 6.9: A schematic plot of a comparison of the original IGF solver in IMPACT with the novel RIGF solver implemented in IMPACT. Compared is the rms bunch size for both x direction (top left), y direction (top right), and z direction (bottom).

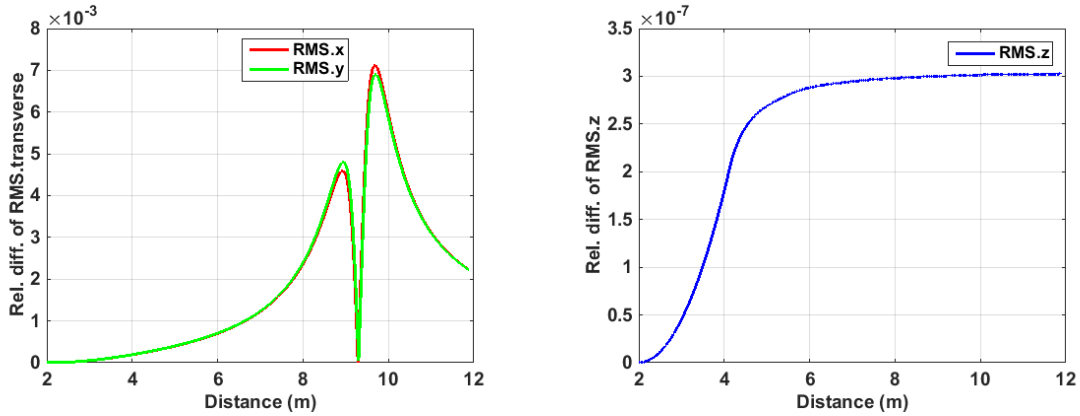


Figure 6.10: Comparison of the relative difference (rel. diff.) of the rms bunch size: RMS.transverse (left) and RMS.z (right) between RIGF and IGF integrals for a bunch tacking example in IMPACT.

of 10^{-3} in the transverse direction, whereas it is lower in the longitudinal direction. The magnitude differs to the MOEVE-PIC studies. This may be because the tracking

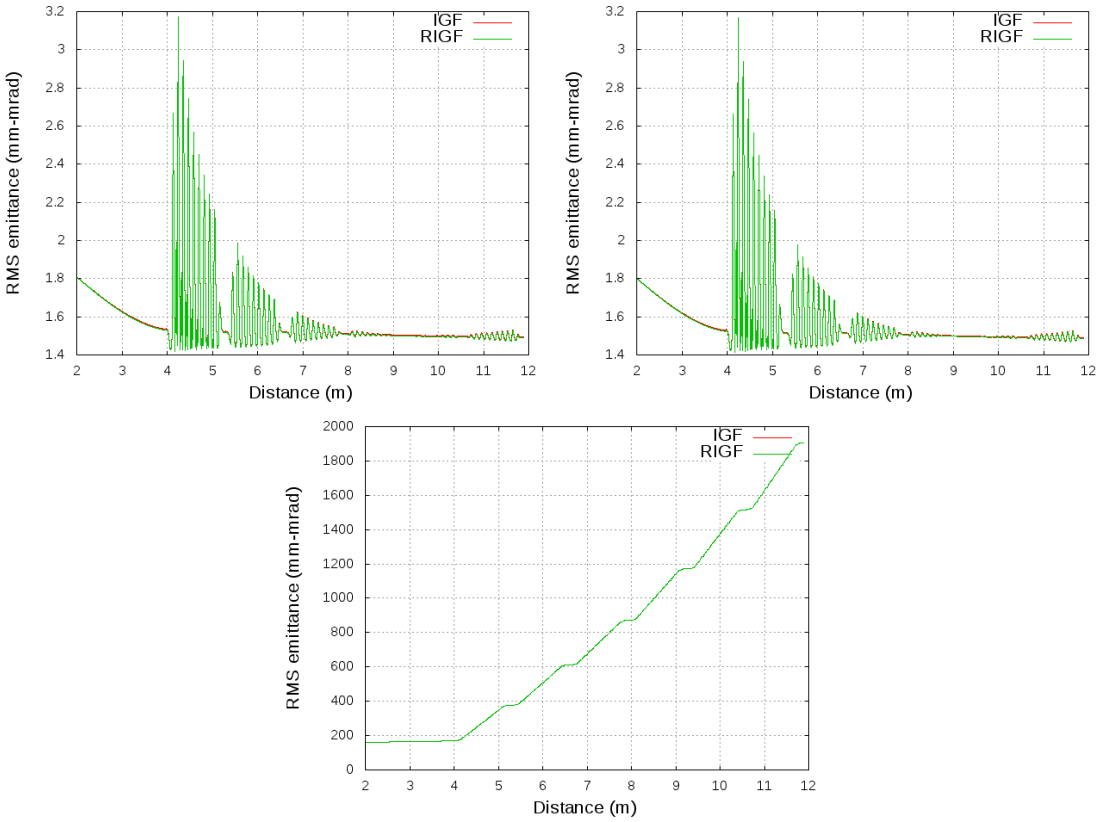


Figure 6.11: A schematic plot of a comparison of the original IGF solver in IMPACT with the novel RIGF solver implemented in IMPACT. Compared is the rms emittance for both x direction (top left), y direction (top right), and z direction (bottom).

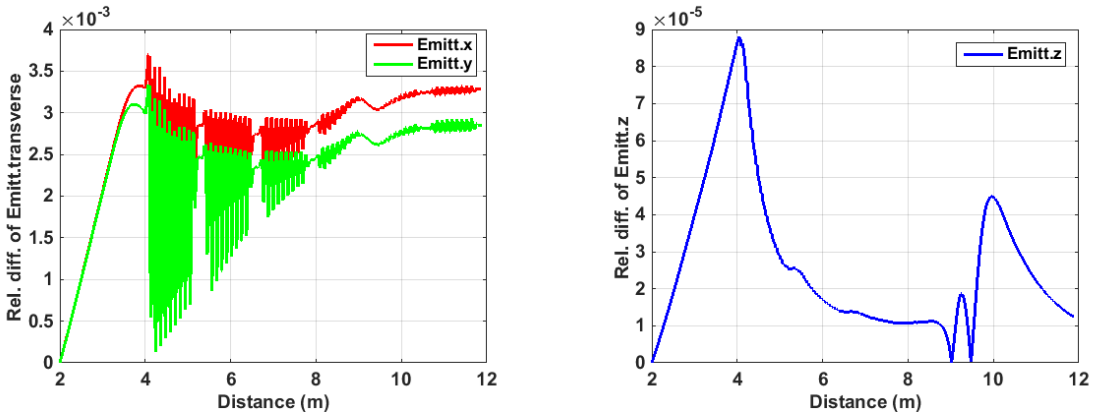


Figure 6.12: Comparison of the relative difference (rel. diff.) of the bunch emittance: Emitt.transverse (left) and Emitt.z (right) between RIGF and IGF integrals for a bunch tacking example in IMPACT.

distance in MOEVE-PIC is too short for the relative difference to reach the magnitude of the example in IMPACT. Other factors may influence the difference as well, e.g.

different mesh numbers, bunch parameters, domain size and other calculations inside PIC.

Again, both relative differences of emittance and rms bunch size have a slight increase in a drift area. However, the relative differences for both are at a low magnitude. When the external E.M. fields are added for real accelerator simulation, the magnitude varies with different sections but stays at a low magnitude.

In fact, the errors introduced by the PIC model are the dominant part of the total numerical error. Compared to that, the relatively small difference between the two IGF methods is hardly of any relevance. Boonpornprasert [11] shows the comparison between a simulation and an experiment in Photo Injector Test Facility at DESY, Location Zeuthen (PITZ). This difference between simulation and experiment is actually still not fully understood in the research community.

6.5 Efficiency improvement results of the novel Poisson solver

After completing the verification and validation of the novel Poisson solver, this section presents the efficiency improvement, which can be considered as a core result of this dissertation.

The Poisson solver's efficiency improvements are summarized in two parts: the theoretical algorithm improvement, and the computing technology improvement.

The two Poisson solvers are abbreviated as commonly used Poisson solver (CPS) and novel efficient Poisson solver (EPS) in this section. Both solvers utilize the real to complex FFT for the first dimension of the 3D vectors.

The theoretical algorithm improvement

The efficiency improvement of the Poisson solver is separated into three parts as introduced in Chapter 3 and Chapter 4:

- The calculation of the IGF integral values is replaced by the efficient IGF integral values (with blue color and noted as IGF integrals in figures).
- The FFT calculation of RESE of the IGF values is replaced by the DCT of the efficient IGF values (with red color and noted as IGF transforms in figures).
- The explicitly zero-padded FFT is replaced by the implicitly zero-padded FFT for the charge density (with yellow color and noted as ρ transforms in figures).

The results of the theoretical algorithm improvement are presented in Section 6.5.1 in a single CPU simulation, as shown in Figure 6.13 and Table 6.5. First, the percentage points of ρ transforms increase significantly for both CPS and EPS with increasing grid resolution. Also, ρ transforms dominate the total time consumption for fine meshes. The GF integrals are the second largest time consumption for CPS, similar to the GF transforms for EPS.

The speed-up of EPS compared to CPS is around 3 to 5. In particular, the speed-up for both IGF integrals and IGF transform is around 4 to 8, the speed-up for ρ transform is around 2. The time evolution with mesh numbers are plotted in log-log scale and recorded in Figure 6.13 and Table 6.5.

The presented results of execution time in this section may differ from the results in the former chapters, in particular for a problem of small size, e.g. 32^3. This is because the execution time is recorded by the average execution time in former chapters, whereas the execution time is measured for a single implementation in this section. In order to obtain a real trend with grid difference, the solver is decided to be executed for a single implementation with increasing grid resolution managed by Bash[1] scripts under the same computing condition. Furthermore, the overhead of the external APIs such as OpenMP API, is also a key factor.

[1]Bash is a system shell and command language written by Brian Fox used in Unix-like system.

The computing technology improvement

The computing technology improvement is divided into different sub-sections due to different HPC technologies, e.g. OpenMP and OpenMP+CUDA for a workstation, MPI and MPI+OpenMP for supercomputers.

1. The simulation results of OpenMP parallel solver compared to EPS in a workstation is presented in Section 6.5.2. In total, four CPU threads are used for the OpenMP parallel execution. As shown in Figure 6.15, the speed-up of OpenMP parallel solver compared to EPS can reach as high as 3. However, the acceleration of computation is not obvious for a small size problem, e.g. 32^3, 64^3 since the overhead of OpenMP API costs percentage of time as well.

2. The simulation results of OpenMP+CUDA parallel solver compared to EPS in a workstation are presented in Section 6.5.3. In total, four CPU threads are used for the OpenMP parallel execution, three threads of those are used for the CPU portion while one thread masters the GPU portion. As shown in Figure 6.15, the speed-up of OpenMP+CUDA parallel solver compared to EPS can reach as high as 3. As the OpenMP parallel routine, the acceleration of OpenMP+CUDA is weak for a small mesh, e.g. 32^3, since the overhead of both OpenMP and CUDA APIs consumes the major execution time.

Regarding the combination of the theoretical algorithm improvement and the computing technology improvement, the total speed-up results for a workstation are shown in Figure 6.16. Compared to CPS, the final speed-up with parallel technology can reach as high as 10. Still, a small size computation may limit the performance of the parallel solver. As the speed-up results are measured for a single execution in order to reflect the real trend with grid difference. In real simulation, the speed-up results can be variously higher as introduced in the former chapters when the Poisson solver is repeated, which is similar to the situation recorded in the former chapters. If the size of problem is large, the hybrid OpenMP+CUDA may show the best performance as shown.

For supercomputers, the results of speed-up are reported in Section 6.5.4 and Section 6.5.5.

6.5.1 Single CPU simulation results in a workstation

Table 6.5: Comparison of CPS and EPS with increasing grid resolution.

N	CPS	EPS	N	CPS	EPS
32^3	0.020377 s	0.008351 s	160^3	4.276374 s	1.249857 s
64^3	0.178336 s	0.064438 s	192^3	7.773760 s	2.413308 s
96^3	0.761309 s	0.276435 s	224^3	12.256783 s	3.776264 s
128^3	2.314242 s	0.518175 s	256^3	21.739702 s	6.774378 s

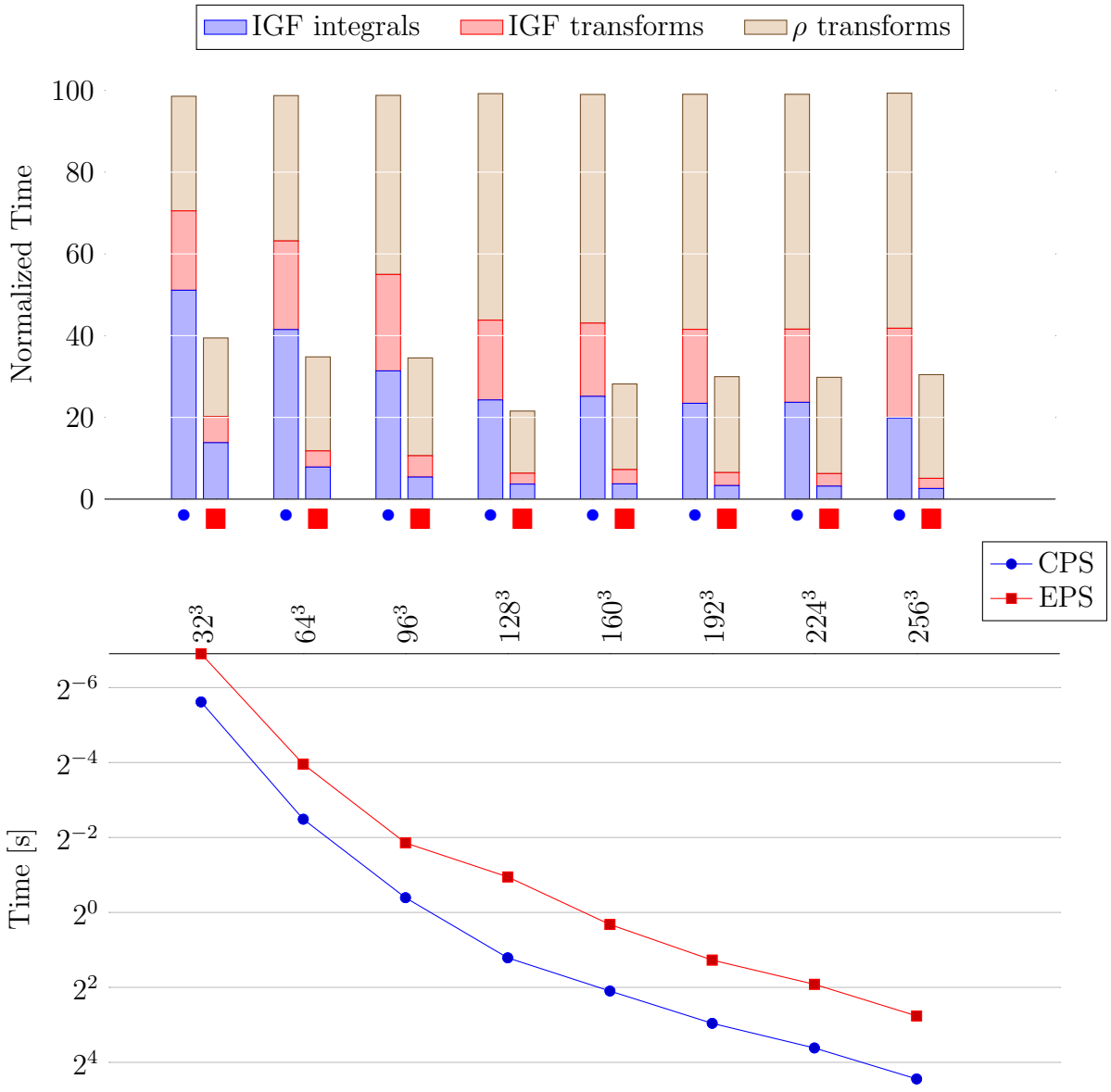


Figure 6.13: Comparison of CPS and EPS with increasing grid resolution.

6.5.2 OpenMP simulation results in a workstation

Table 6.6: Comparison of EPS and OpenMP parallel solver with increasing grid resolution.

N	EPS	OpenMP	N	EPS	OpenMP
32	0.008351 s	0.007070 s	160	1.249857 s	0.428335 s
64	0.064438 s	0.056496 s	192	2.413308 s	0.820205 s
96	0.276435 s	0.139966 s	224	3.776264 s	1.283981 s
128	0.518175 s	0.185006 s	256	6.774378 s	3.165152 s

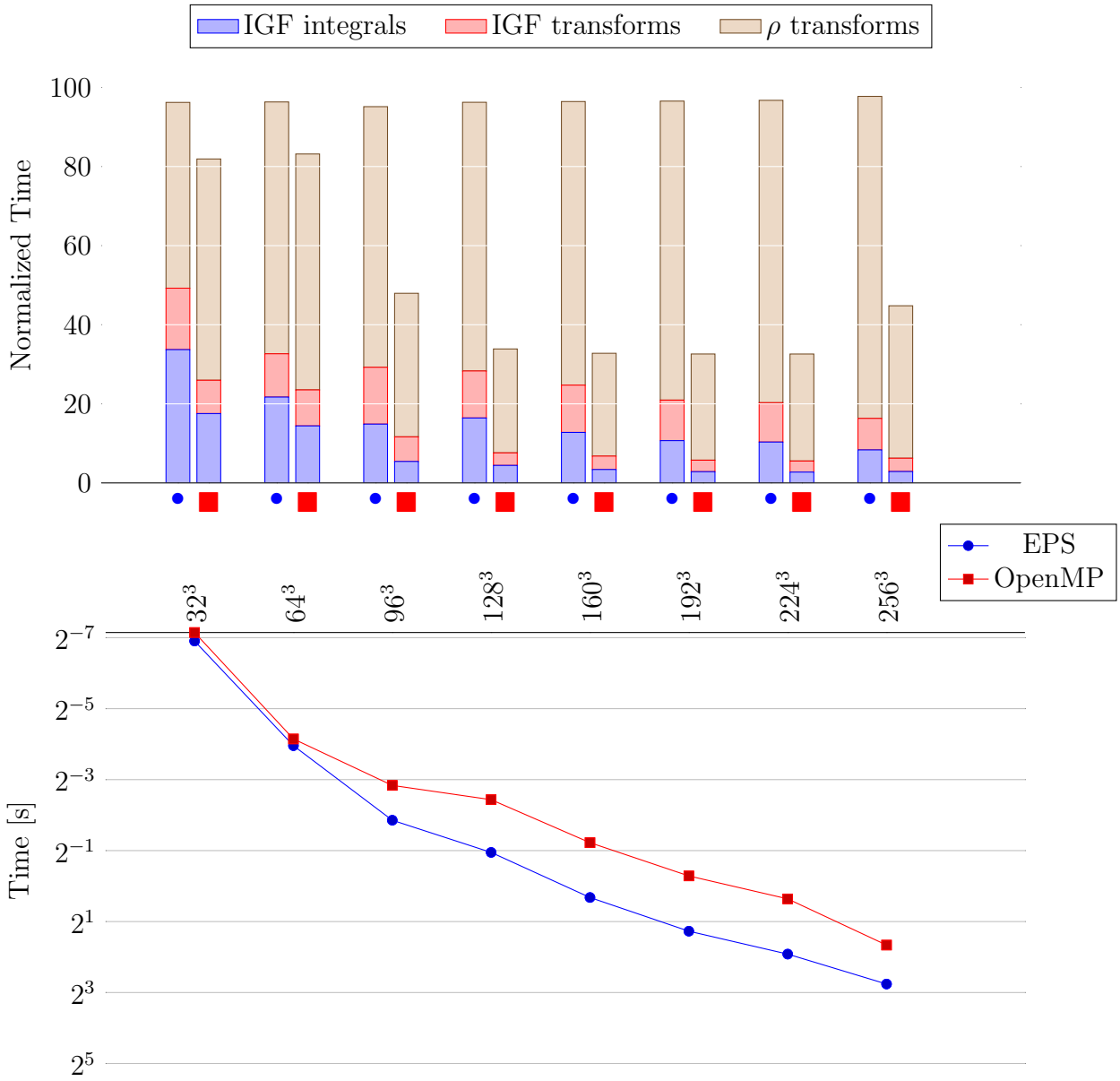


Figure 6.14: Comparison of EPS and OpenMP parallel solver with increasing grid resolution.

6.5.3 OpenMP+CUDA simulation results in a workstation

Table 6.7: Comparison of EPS and OpenMP+CUDA parallel solver with increasing grid resolution.

N	EPS	OpenMP+CUDA	N	EPS	OpenMP+CUDA
32	0.008351 s	0.008100 s	160	1.249857 s	0.547934 s
64	0.064438 s	0.038541 s	192	2.413308 s	0.926579 s
96	0.276435 s	0.132028 s	224	3.776264 s	1.450704 s
128	0.518175 s	0.268601 s	256	6.774378 s	1.964245 s

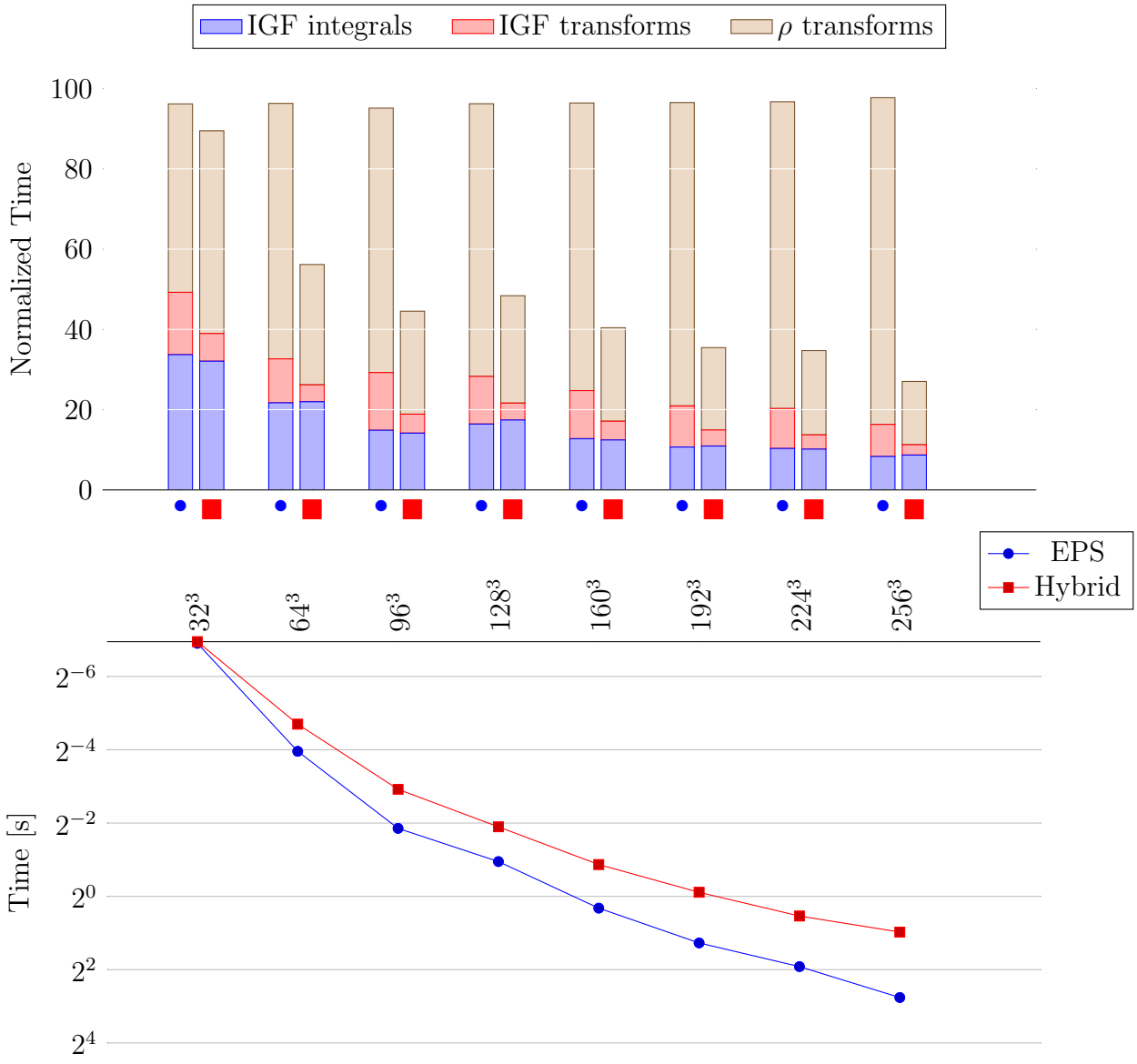


Figure 6.15: Comparison of EPS and OpenMP+CUDA (Hybrid in the legend) parallel solver with increasing grid resolution.

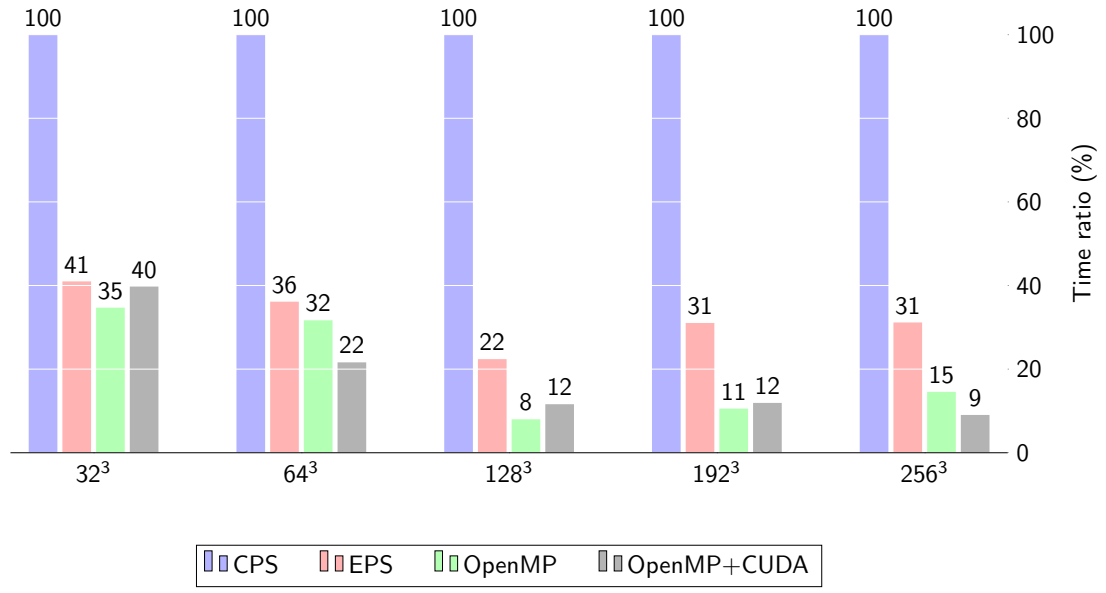


Figure 6.16: Comparison of relative computation time of different solvers and parallel routines.

6.5.4 Open MPI simulation results in a supercomputer

We compare the efficiency improvement of the new Poisson solver. As shown in Table 6.8 and Figure 6.17, the speed-up is as high as 2, which is significant. For small size problems with a large number of processes, the speed-up is not obvious. The reason for this may be that the data transports occupy a high percentage of the whole time consumption for both solvers.

Table 6.8: Comparison of parallel CPS and parallel EPS with Open MPI in Edison supercomputer

N	np	t_{CPS}	t_{EPS}	N	np	t_{CPS}	t_{EPS}
64^3	64	21.44749 s	11.39442 s	192^3	64	476.69889 s	308.23102 s
64^3	256	7.91761 s	6.89414 s	192^3	256	81.87869 s	46.75058 s
64^3	1024	14.53673 s	17.47832 s	192^3	1024	46.75058 s	30.34015 s
128^3	64	138.71456 s	78.66420 s	256^3	64	1139.38282 s	759.27792 s
128^3	256	41.37721 s	30.73737 s	256^3	256	311.81897 s	214.20530 s
128^3	1024	15.44938 s	14.00897 s	256^3	1024	142.09763 s	63.00095 s

Even though the speed-up of the parallel EPS routine in a supercomputer is not as high as that of the serial routine in a workstation, the speed-up is still obvious. The possible reasons read as follows:

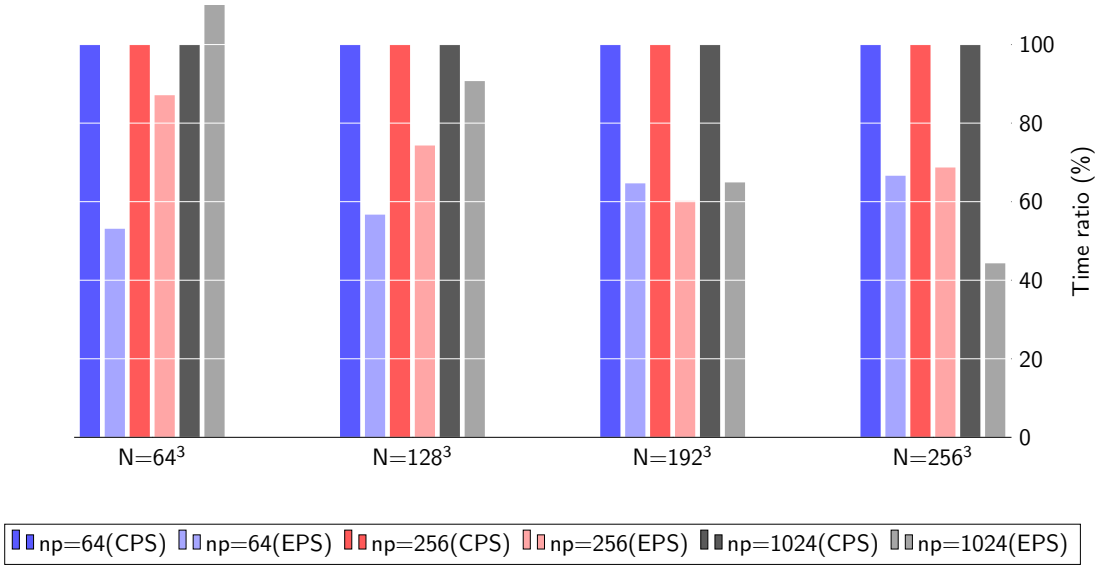


Figure 6.17: Comparison of parallel CPS and parallel EPS with Open MPI in Edison supercomputer. In each case the respective CPS computing time determined the 100%.

- The MPI overhead consumes a high percentage of the total time consumption.
- For a small problem size, the speed-up decreases when we increase the process numbers beyond a certain number.
- Due to the overhead of MPI has a balance with the improvement of the computation, the speed-up increases then decreases with refining the mesh.

The efficient numerical method for parallel routines shows promising results. We experienced a speed-up as high as 2 only from the improvement of algorithm. This speed-up due to algorithm improvements is almost free of charge, which compares very favourably to the speed-up of between 2 and 3 which can be observed moving from Hopper (the former main supercomputer at NERSC) to Edison, which involves hardware costs of dozens of millions of US dollars.

6.5.5 Open MPI+OpenMP simulation results in a supercomputer

As shown in Figure 6.18, the processor number np in use is fixed as 96 (as in Section 5.3.2) for this study. The MPI processes are 96, 48, 32, 16 and 12 while the OpenMP thread number varies from 1, 2, 3, 4 and 6 per MPI process. The execution time for different OpenMP threads per process in use (for two memory affinity situations: Numa_node and depth) is plotted. The mesh numbers $N_{x,y,z} = 512$ (left) and $N_{x,y,z} = 256$ (right) are studied. As shown, the results agree with NERSC's announcement that each MPI process handles 3 OpenMP threads may execute faster

than other number of OpenMP threads. For the two memory allocation modes, Numa_node and depth modes are also similar in performance. However, the hybrid Open MPI+OpenMP routine performs no better than the pure Open MPI routine with the same number of processors in use.

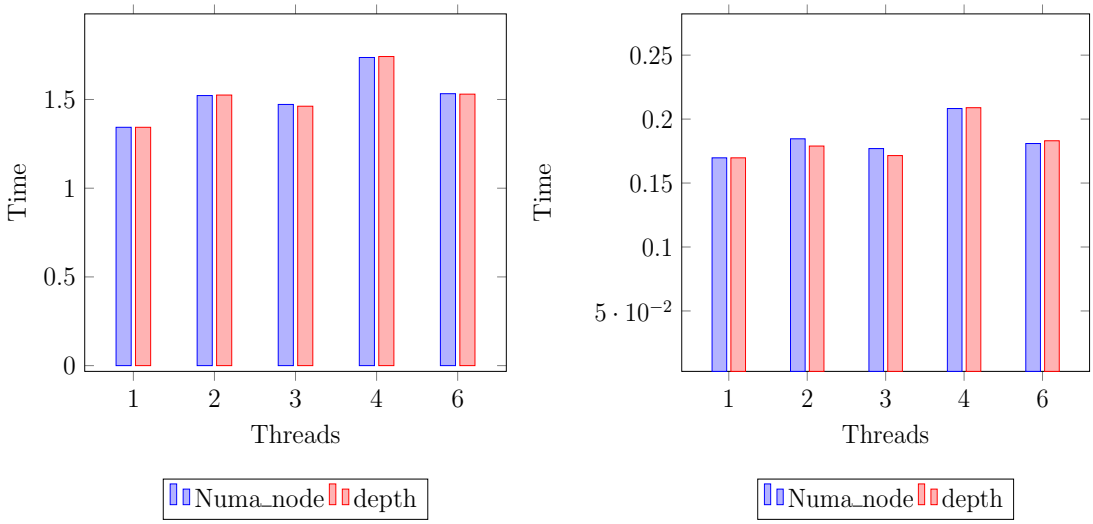


Figure 6.18: Execution time for different OpenMP threads per MPI process in use.

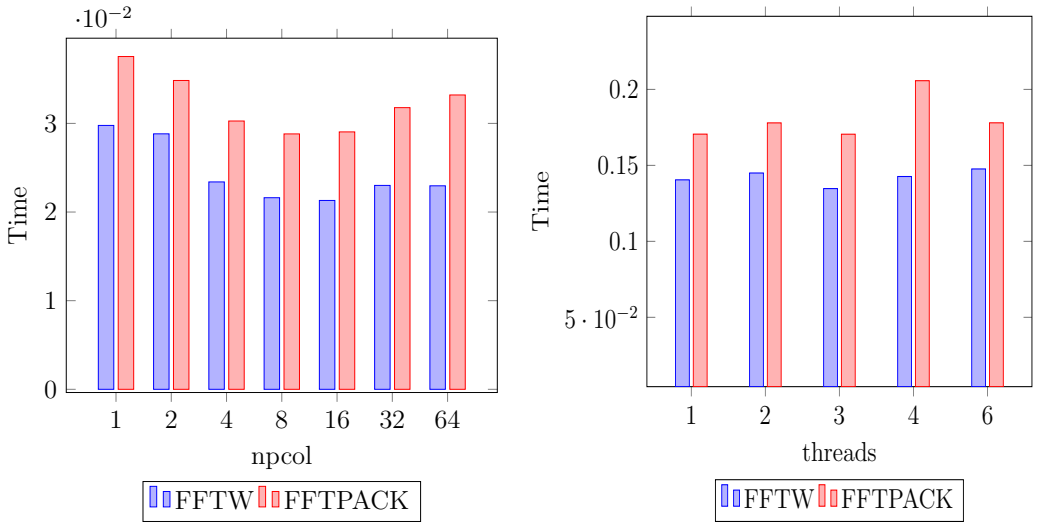


Figure 6.19: Comparison of Poisson solvers with FFTW and FFTPACK with Open MPI+OpenMP in Edison supercomputer.

The overhead of OpenMP API is a key reason for the unsuccessful improvement of the hybrid parallel routine. Adding more OpenMP threads directly does not scale the performance. The memory allocation and management are still the trigger for the shared-memory parallelization.

As an attempt for advanced supercomputers, the FFTs are further computed by the FFTW library. Figure 6.19 shows the results. On the left side, $np = 64$ and

$N_w = 128$, whereas $np = 96$ and $N_w = 256$ on the right side. FFTW improves 20%-25% efficiency of the whole Poisson solver in comparison to FFTPACK. The improvement of FFTW for the whole PIC simulation may be marginal though.

The communications and the 3D FFTs within data transposes are the targets for the further efficient implementation for the internal OpenMP shared-memory parallelization.

6.6 Discussions

6.6.1 Conclusions

In this dissertation, the numerical calculation of space charge effects in beam dynamics simulation is discussed. The mathematical model for the space charge effects leads us to solve Poisson's equation numerically with high accuracy and high efficiency. Among all possible numerical methods, the GF-FFT method is the most essential and commonly used.

We have introduced different 3D efficient integrated Green's functions for the GF-FFT Poisson solver, i.e. RIGF, CIGF and CRIGF strategies. These strategies rely on the bunch size scaling and the domain-bunch ratio. The new efficient integrated Green's functions are well suited for the original problems and speed up the calculations almost without losing accuracy for the results compared to the original IGF method. So we suggest using the efficient IGF methods rather than the original IGF in order to speed up calculations. Independently and at a later date, a similar efficient IGF method is also researched by Dohlus and Henning [27]. The idea is very similar, but they express their efficient IGF with a smoothing function and a smoothing factor.

We have also presented further optimization of the discrete convolution routine. Firstly, the double-sized extension in each direction with FFT is replaced by a fundamental DCT of the GF values. This is proven by means of a mathematical theory. Secondly, the commonly used numerical convolution routine by Hockney and Eastwood is also replaced by a novel implicitly zero padded convolution. The subsequent element-by-element multiplications are shifted to adopt the novel routine too. The accuracy is not influenced by the novel routine, but the calculation time is reduced significantly.

Since our aim is to pursue an efficient and accurate Poisson solver for the application in beam dynamics simulations, the aforementioned novel methods (routines) are parallelized for different computation architectures. Shared-memory parallelization using OpenMP API, heterogeneous parallelization of CPU+GPU, distributed memory parallelization using MPI for supercomputers, and MPI+OpenMP parallelization for advanced supercomputers are given. The parallel routines' performances with different CPU threads, processes and scalings are studied as well.

The Poisson solver performs well in example verifications and applications. Three verification examples are taken for the comparison with HandE's routine together

with IGF method. Convergence studies for the efficient IGF with varying parameters are shown, as well as the error convergences with finer meshes. The feasibility in current scientific reality is demonstrated in applications such as tracking a single bunch in a drift space in MOEVE-PIC, and simulating a bunch after the electron gun in IMPACT for a virtual accelerator.

In conclusion, the novel improved Poisson solver proves to be more efficient than the commonly used routine with no (or only very little) loss of accuracy in the calculation of space charge for beam dynamics simulations. The technique improvements are also researched with the state-of-the-art parallelization from workstations to advanced supercomputers in practice.

6.6.2 Outlook

Finally, we briefly remark on some key issues for the future research topics that may interest us and others in the related fields.

Higher accuracy Poisson solver

In the numerical convolution between charge density and Green's function, the accuracy corresponds to the mid-point rule of numerical integration. The integrated Green's function can obtain a better accuracy solution, but not an essential improvement. Simulating with finer meshes definitely can enable obtaining the solution with higher accuracy, but only by sacrificing the execution time.

A numerical method to achieve a higher accuracy solution is needed [68] [42]. This may cost slightly more in terms of calculation, but not as much as refining the discretization. On the other hand, the higher accuracy at some local areas is also welcome to decrease the influence of some discontinuous properties of the r.h.s function (charge density with the surrounding zero area) in simulations.

Further optimization of parallel routine

While the HPC industry grows overwhelmingly important in scientific research, state-of-the-art parallel technologies are developing substantially. The parallel strategies lead to new hardware, and programming routines. All existing codes have to be adapted: parts of the codes have to be optimized for the new technologies. The next generation HPC facilities will reach exaFLOPS (10^{18} floating point operations per second). The beam dynamics simulation codes have to be enhanced to fit the new environment in order to reach the peak.

Firstly, the common pure MPI routine will be replaced by the MPI+OpenMP programming routine, and (or) the MPI+CUDA programming routine. The next generation supercomputers probably contain accelerators such as MICs and GPUs. The hybrid programming has to be optimized to match the hardware's updates.

Secondly, better weak and strong scaling optimizations for massive cores and large meshes have to be achieved. Some barriers have to be cleared, such as the other form

of global data transport and communications in multidimension FFTs.

New types of fast Poisson solver

The data transport forms a barrier for the 3D vector transpose inside the FFTs for the implementations in supercomputers. A new type of fast Poisson solver which suits the supercomputer's architecture deserves further research. The multigrid solver may provide a good performance while the (free-space) boundary conditions have to be approximated in advance. The FACR method [82] may show some advantages as well, whose routine may match the modern supercomputers avoiding the specific memory-bound data transport in FFTs.

Insight for other fields

The convolution form of integrating Green's function with charge density shares the same calculation routine with other physics formulas, e.g. in the calculation of gravitational potential, and the calculations for fluid (or air) turbulence. The implicitly zero-padded FFT strategy was originally invented for the turbulence studies. As presented in this dissertation, the efficient GF calculations and implicit RESE with DCT are worth referencing in other fields. The cross-subject numerical method can benefit all the related fields, and deserves broader research.

A Green's function, inverse operator, convolution theory, DFTs, and Kronecker product

A.1 Green's function and inverse operator

In principle, Green's function is the inverse operator for a given differential operator L, which has been shown in [8]. In Theorem 1, Green's function and inverse operator for the invertible situation has been extended to generalized Green's function and generalized inverse operator $L^\dagger$ (Moore-Penrose inverse [8]), respectively.

Theorem 1. *[8] For a differential operator L, there is a kernel*

$$G(s,t), \quad a \leq s,\ t \leq b,$$

such that, for all $y \in R(L)$,

$$(L^\dagger y)(s) = \int_a^b G(s,t)y(t)dt,\ a \leq s \leq b,$$

then $G(s,t)$ is called the Green's function of L in the case of L is invertible ($L^\dagger$ is the inverse operator of L); or the generalized Green's function for the not invertible L ($L^\dagger$ is the generalized inverse operator of L). For any $y \in R(L)$, the unique solution of

$$Lx = y$$

is given by

$$x(s) = \int_a^b G(s,t)y(t)dt, a \leq s \leq b.$$

A.2 1D FFT convolution theorem

Theorem 2. *[13] Let f and g be two 1D vectors, the convolution of the two is defined as $f * g$. Let $\mathfrak{F}$ denote the Fourier transform matrix, and $\mathfrak{F}^{-1}$ denote the inverse Fourier transform matrix. Then, the following equations Eq. (A.1) Eq. (A.2) succeed.*

$$\mathfrak{F}^{-1}(f * g) = \mathfrak{F}^{-1}(g) \cdot \mathfrak{F}^{-1}(f), \tag{A.1}$$

and

$$\mathfrak{F}(f * g) = \mathfrak{F}(g) \cdot \mathfrak{F}(f). \tag{A.2}$$

A.3 Discrete FFT convolution in multidimension

As shown from Theorem 2, the point-wise multiplication in 1D can be dealt with in either Fourier space or inverse Fourier space. In multidimension convolution, the DFT is individually implemented for each direction. Thus either of two forms can be used for each dimension, and the combinations along directions are possible.

Theorem 3. *Let f and g be two 3D vectors, the convolution of the two is defined as $f * g$. Let $\mathfrak{F}$ denote the Fourier transform matrix for 1D vector, and $\mathfrak{F}^{-1}$ denote the inverse Fourier transform matrix. The following equation succeed:*

$$\mathfrak{F}_z^{sz} \otimes \mathfrak{F}_y^{sy} \otimes \mathfrak{F}_x^{sx}(vec(f * g)) = \mathfrak{F}_z^{sz} \otimes \mathfrak{F}_y^{sy} \otimes \mathfrak{F}_x^{sx}(vec(g)) \cdot \mathfrak{F}_z^{sz} \otimes \mathfrak{F}_y^{sy} \otimes \mathfrak{F}_x^{sx}(vec(f)). \quad \text{(A.3)}$$

where vec() is the vector operation of 3D vector, $\otimes$ is Kronecker product (see next section), and sw can be taken as 1 (Fourier transform), or -1 (inverse Fourier transform).

A.4 Kronecker product

If F is an $m \times n$ matrix and G is a $p \times q$ matrix, then the Kronecker product $F \otimes G$ is the $mp \times nq$ block matrix reads as:

$$F \otimes G = \begin{pmatrix} F_{1,1}G & F_{1,2}G & \dots & F_{1,n}G \\ F_{2,1}G & F_{2,2}G & \ddots & \vdots \\ \vdots & \ddots & \ddots & F_{m-1,n}G \\ F_{m,1}G & \dots & F_{m,n-1}G & F_{m,n}G \end{pmatrix}$$

B FFT-based Poisson solver with various boundary conditions

B.1 The fast spectral Poisson Solver in 1D

The basic real trigonometric transforms are essential to derive the fast 1D spectral Poisson solver.

Real trigonometric transform:

The (type-I) discrete sine transform (DST):

$$y(1:m-1) = \mathrm{DST}(x(1:m-1)),$$

$$y_k = \sum_{j=1}^{m-1} \sin(\frac{kj\pi}{m}) x_j.$$

The DCT:

$$y(0:m) = \mathrm{DCT}(x(0:m)),$$
$$y_k = \frac{x_0}{2} + \sum_{j=1}^{m-1} \cos(\frac{kj\pi}{m}) x_j + \frac{(-1)^k x_m}{2}.$$

The type-II discrete sine transform (DST-II):

$$y(1:m) = \text{DST-II}(x(1:m)),$$

$$y_k = \sum_{j=1}^{m} \sin(\frac{k(2j-1)\pi}{2m}) x_j.$$

The DCT-II:

$$y(0:m-1) = \text{DCT-II}(x(0:m-1)),$$
$$y_k = \sum_{j=0}^{m-1} \cos(\frac{k(2j+1)\pi}{2m}) x_j.$$

The complexity of these four real trigonometric transforms is expressed as $O(m^2)$. These trigonometric transforms can be obtained through Table B.1 and DFT by the way of the following relation:

$$\mathrm{DFT}_m = C_m - iS_m,$$

where

$$[C_m]_{kj} = \cos(\frac{2\pi kj}{m}),\ [S_m]_{kj} = \sin(\frac{2\pi kj}{m}).$$

As known, the calculation of DFT, C_m, S_m can be accelerated by FFT. Therefore the complexity of the real trigonometric transform can reduce to $O(5m \log_2 m)$ when m is the power of two, even reduce to $O(2.5m \log_2 m)$ if a real FFT is performed. The detailed introduction about the trigonometric transform and part of the following 1D fast Poisson solver can be found in [87].

Table B.1: Matrix specification of trigonometric transforms.

Transform	Matrix Specification
DST	$y(1:m-1) = S_{2m}(1:m-1, 1:m-1)\ x(1:m-1)$
DCT	$y(0:m) = C_{2m}(0:m, 0:m)\ [x_0/2 \mid x(1:m-1) \mid x_m/2]^T$
DST-II	$y(1:m) = S_{4m}(1:m, 1:2:2m-1)\ x(1:m)$
DCT-II	$y(0:m-1) = C_{4m}(0:m-1, 1:2:2m-1)\ x(0:m-1)$

$*$ m is the power of two, the exact calculation can be different and accelerated by different specific routines.

Poisson's equation in 1D reads as:

$$-\frac{\mathrm{d}^2U(x)}{\mathrm{d}x^2} = F(x),\ a \le x \le b. \tag{B.1}$$

with five possible boundary conditions:

$$\begin{aligned}
&\text{Dirichlet-Dirichlet:} && U(a) = \alpha, U(b) = \beta,\\
&\text{Dirichlet-Neumann:} && U(a) = \alpha, U'(b) = \beta,\\
&\text{Neumann-Dirichlet:} && U'(a) = \alpha, U(b) = \beta,\\
&\text{Neumann-Neumann:} && U'(a) = \alpha, U'(b) = \beta,\\
&\text{Periodic:} && U(a) = U(b).
\end{aligned}$$

The second order central-difference is performed to discretize the 1D continuous Poisson's equation as:

$$\frac{u(x_{l-1}) - 2u(x_l) + u(x_{l+1})}{h^2} = f_l,$$

$a = x_0 < \cdots < x_l < \cdots < x_n = b$. In the matrix form:

$$T_m u = f. \tag{B.2}$$

Theorem 1 (Spectral Decomposition, C. Van Loan, 1992)**.** *If we define the discrete Laplace matrices for the above five boundary conditions as L, there exists an invertible matrix V for each boundary condition, such that*

$$V^{-1}LV = \Lambda,$$

here Λ is the spectral matrix, which is diagonal.

$$\begin{array}{lllll}
\textit{D-D:} & \textit{V=DST,} & \Lambda_{jj} = 4\sin^2(\frac{j\pi}{2n}), & j = 1 : n-1. \\
\textit{D-N:} & \textit{V=DST-II,} & \Lambda_{jj} = 4\sin^2(\frac{2j-1}{4n}\pi), & j = 1 : n. \\
\textit{N-D:} & \textit{V=DCT-II,} & \Lambda_{jj} = 4\sin^2(\frac{2j-1}{4n}\pi), & j = 1 : n. \\
\textit{N-N:} & \textit{V=DCT,} & \Lambda_{jj} = 4\sin^2(\frac{j\pi}{2n}), & j = 0 : n. \\
\textit{P:} & \textit{V=IDFT,} & \Lambda_{jj} = 4\sin^2(\frac{j\pi}{2n}), & j = 0 : n-1.
\end{array}$$

where D stands for Dirichlet, N stands for Neumann and P stands for periodic for the boundary conditions.

By Theorem 1, the spectral Poisson solver sketch can be directly demonstrated as the following statement into two classes:

Class 1: L is a full rank matrix. We solve the linear system $Lu = f$ with the factorization of L as:

$$\begin{aligned}
Lu &= f \\
VV^{-1}LVV^{-1}u &= f \\
\Lambda V^{-1}u &= V^{-1}f \\
V^{-1}u &= \Lambda^{-1}V^{-1}f \\
\hat{u} &= \Lambda^{-1}\hat{f} \\
u &= V\Lambda^{-1}V^{-1}f.
\end{aligned}$$

The result of the inverse trigonometric transform V^{-1} of f and u signed as $\hat{f}$ and $\hat{u}$. The D-D, D-N, N-D boundary conditions belong to Class 1.

Class 2: L is a rank deficient matrix. In this case, Λ is not invertible which suppresses the solving procedure as: $\Lambda_{0,0} = 0$

$$\Lambda^{-1}\hat{f} \nrightarrow \hat{u}.$$

where,

$$\hat{f} = V^{-1}f, \quad \hat{u} = V^{-1}u.$$

The N-N and periodic boundary conditions belong to Class 2.

For N-N boundary conditions, we see that the eigenvector corresponding to $\Lambda_{0,0}$ is fulfilled with elements 1 inside the matrix form of DCT. If we define the $\Lambda_{0,0}^{-1} = 0$ instead of the infinity in Λ^{-1}, we could solve the problem further like Class 1. The derived solution is only a constant distance shift from the true solution.

For periodic boundary conditions, we solve it in the N-N way, only define the $\Lambda_{0,0}^{-1} = 0$ instead of the infinity at $\Lambda_{0,0}^{-1}$, we may not achieve the demanded solution since this automatically periodic setting does not match with reality during the simulation. In practice, we can extend the r.h.s vector f_n to:

$$f_{2n} = \begin{pmatrix} f(0:n) \\ -f(n-1:1) \end{pmatrix} \text{or} \begin{pmatrix} f(0:n) \\ 0 \end{pmatrix} \text{in } L_{2n}^{(P)} u_{2n} = f_{2n},$$

in order to meet the symmetry property in real situations.

B.2 3D fast spectral Poisson Solver for various mixed boundary conditions

Poisson's equation in 3D reads as:

$$-\Delta U(x,y,z) = F(x,y,z). \tag{B.3}$$

We consider a cubic domain in x, y, z axes with various mixed boundary conditions in each axis as: $BC\_X_{up}(x,y,z)$, $BC\_X_{down}(x,y,z)$, $BC\_Y_{up}(x,y,z)$, $BC\_Y_{down}(x,y,z)$, $BC\_Z_{up}(x,y,z)$, $BC\_Z_{down}(x,y,z)$, six surface functions for the proper boundary conditions, which may be D-D, D-N, N-D, N-N, periodic for each axis.

Suppose we discretize the computational domain with N_x, N_y, N_z mesh points, equilibrium for each axis, the stepsizes are h_x, h_y, h_z, respectively. As in the 1D situation, we use the 3D spectral decomposition by the Kronecker product form to express the discrete Poisson's equation:

$$(\frac{1}{h_x^2}(I_{N_z} \otimes I_{N_y} \otimes L_{N_x}) + \frac{1}{h_y^2}(I_{N_z} \otimes L_{N_y} \otimes I_{N_x}) + \frac{1}{h_z^2}(L_{N_z} \otimes I_{N_y} \otimes I_{N_x}))u_{bc} = f_{bc}$$

f_{bc} is the r.h.s function, which has been updated by the six surface boundary condition functions in the same way as the 1D situation.

The linear system of equations can be simply written as:

$$(I \otimes I \otimes \frac{1}{h_x^2}\Lambda_x + I \otimes \frac{1}{h_y^2}\Lambda_y \otimes I + \frac{1}{h_z^2}\Lambda_z \otimes I \otimes I) \cdot (V_z^{-1} \otimes V_y^{-1} \otimes V_x^{-1})u_{bc} = (V_z^{-1} \otimes V_y^{-1} \otimes V_x^{-1})f_{bc},$$

where I is the unit matrix, $\Lambda_{x,y,z}$ are the eigenvalue matrices. $(V_z^{-1} \otimes V_y^{-1} \otimes V_x^{-1})$ is the Fourier-type transforms in 3D. In the "Fourier space", denoted as frequency space, we have:

$$\hat{u}_{bc\,i,j,k} = \frac{1}{\frac{\Lambda_{ii}}{h_x^2} + \frac{\Lambda_{jj}}{h_y^2} + \frac{\Lambda_{kk}}{h_z^2}} \hat{f}_{bc\,i,j,k}.$$

Therefore the solution reads as:

$$u_{bc\,i,j,k} = (V_z \otimes V_y \otimes V_x)\hat{u}_{bc\,i,j,k}.$$

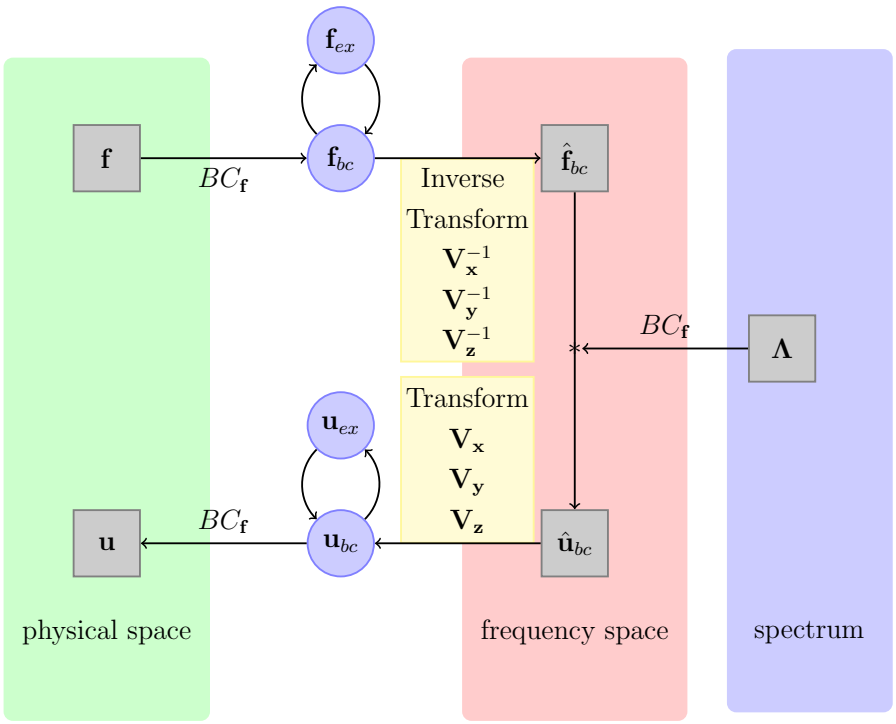


Figure B.1: A sketch of the fast spectral Poisson solver.

The discrete 3D r.h.s. vector f is transformed by boundary conditions BC_f to f_{bc}, which can be extended to f_{ex} as a new f_{bc} if necessary (for periodic boundary conditions). Then inverse transforms V_x^{-1}, V_y^{-1}, V_z^{-1} are performed in each dimension to obtain $\hat{f}_{bc}$ in frequency space. The frequency solution $\hat{u}_{bc}$ is derived by eigenvalue matrix Λ with the corresponding BC_f. The final solution u is achieved by the transforms V_x, V_y, V_z in each dimension from frequency space to physical space and the boundary conditions.

This Poisson solver provides a total of 5^3 possible boundary conditions which may reflect the fact of the space charge effect closer. The Poisson solver utilizes a direct strategy to resolve the solution. This means the error only derives from the discrete representation and round-off error. This solver provides more opportunity to study image charge as well as simulation diagnostics.

C Further numerical verification of the novel efficient method by examples

C.1 Simulate an ideal uniform charged sphere bunch

The bunch parameters of Test Case 1 are the same as their initial values in Section 6.1. The η_φ values (Figure C.1), and $\eta_{\mathbf{E}}$ values (Figure C.2) are plotted in two cutting planes of the computational domain: $(:, N_y/2, :)$ planes (left column), $(:, :, N_z/2)$ planes (right column). The different GF integrals are recognized as GF integrals (top row), efficient IGF integrals (middle row), IGF integrals (bottom row).

For Test Case 1, η_φ (also $\eta_{\mathbf{E}}$) is plotted with the same color scaling for comparison among GF, efficient IGF and IGF integrals. As expected, the difference between efficient IGF and IGF integrals disappears almost entirely, as shown in the figures.

The GF integrals solve larger errors of potential in the bunch area. Outside the bunch area, η_φ agrees among the three different integrals. In contrast to $\eta_{\mathbf{E}}$, the GF integrals represent slightly weaker electric fields at the boundary of the bunch.

C.2 Simulate a Gaussian distributed charged bunch

The bunch parameters of Test Case 3 are the same as their initial values in Section 6.1. The η_φ values (Figure C.3), and $\eta_{\mathbf{E}}$ values (Figure C.4) are plotted in two cutting planes of the computational domain: $(:, N_y/2, :)$ planes (left column), $(:, :, N_z/2)$ planes (right column). The different GF integrals are recognized as GF integrals (top row), efficient IGF integrals (middle row), IGF integrals (bottom row).

For Test Case 3, η_φ (also $\eta_{\mathbf{E}}$) cannot be plotted with the same color scaling for GF integrals since the magnitude of GF integrals differs to the other two integrals. As expected, the figures between efficient IGF and IGF integrals are nearly the same.

The GF integrals solve larger errors of potential for the whole domain. The trend of η_φ (as well as $\eta_{\mathbf{E}}$) is similar for all integrals from the figures. However, the two IGF methods achieve an improvement of one magnitude. The largest η_φ values locate in the center of the bunch area, whereas the largest $\eta_{\mathbf{E}}$ values locate in a circular area around this center.

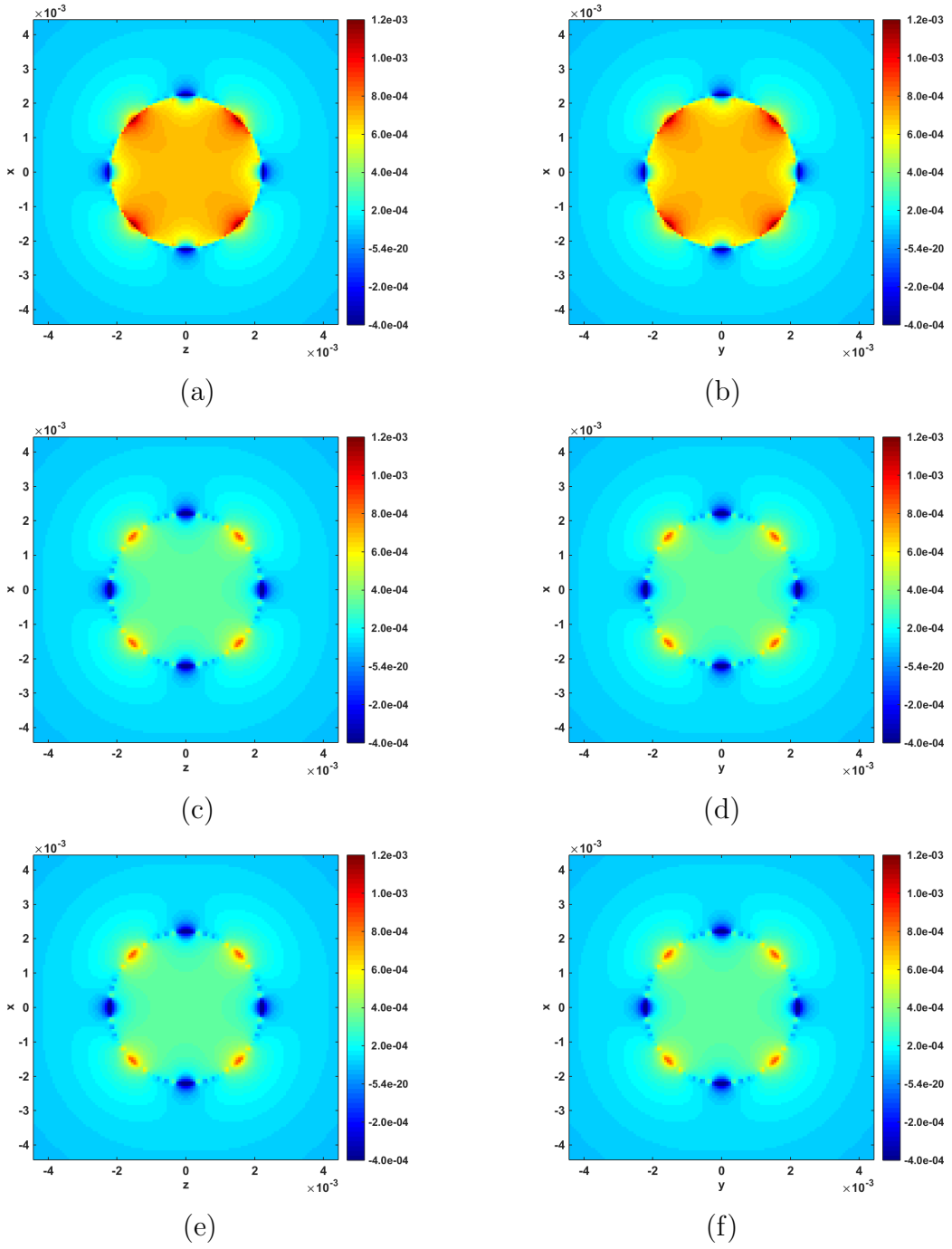


Figure C.1: Comparison of $\eta_\varphi(i,j,k)$ of Test Case 1: at plane $(:, N_y/2, :)$ (left) and $(:, :, N_z/2)$ (right) for the GF integral (a) (b), efficient IGF (c) (d), IGF (e) (f).

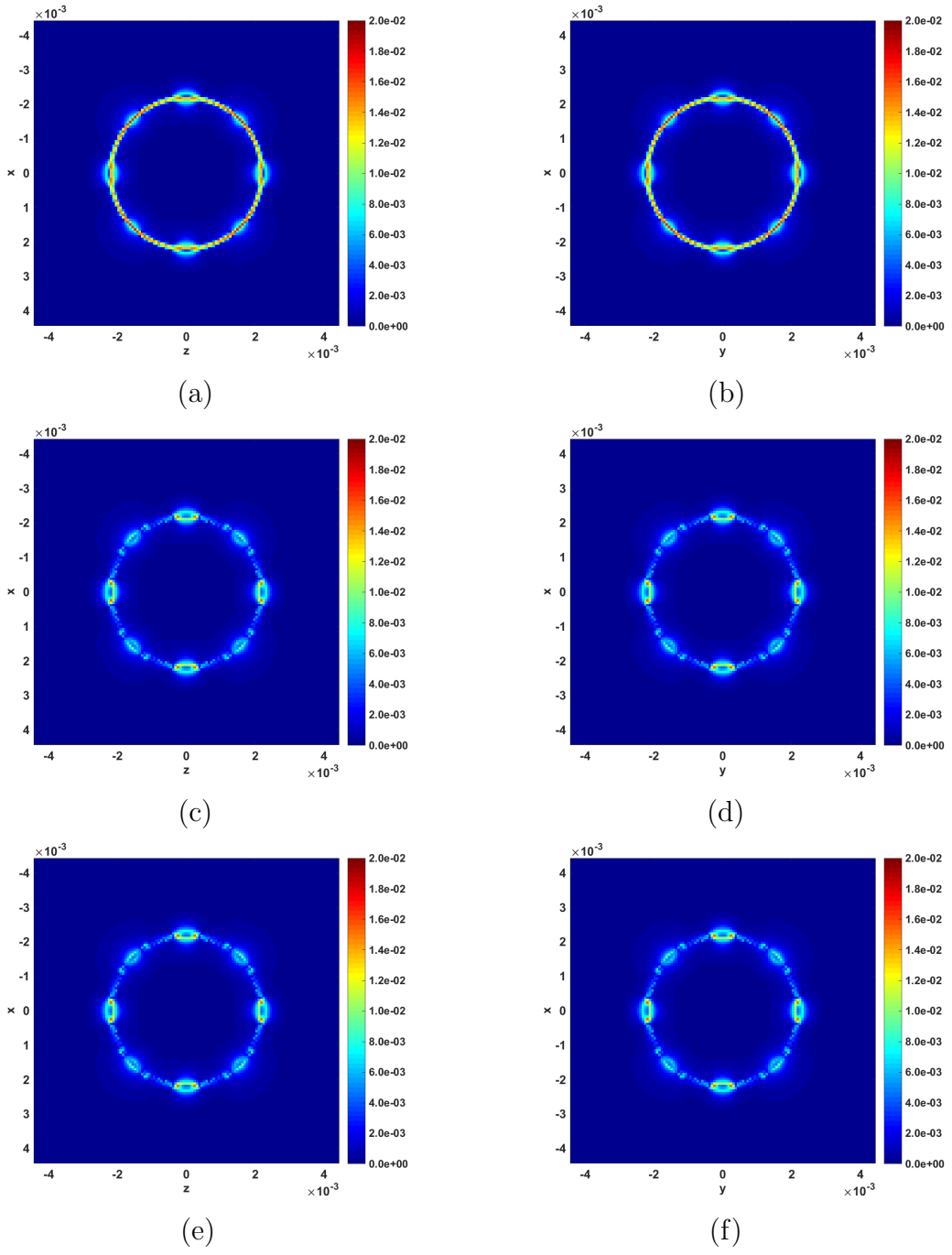


Figure C.2: Comparison of $\eta_{\mathbf{E}}(i,j,k)$ of Test Case 1: at plane $(:, N_y/2, :)$ (left) and $(:, :, N_z/2)$ (right) for the GF integral (a) (b), efficient IGF (c) (d), IGF (e) (f).

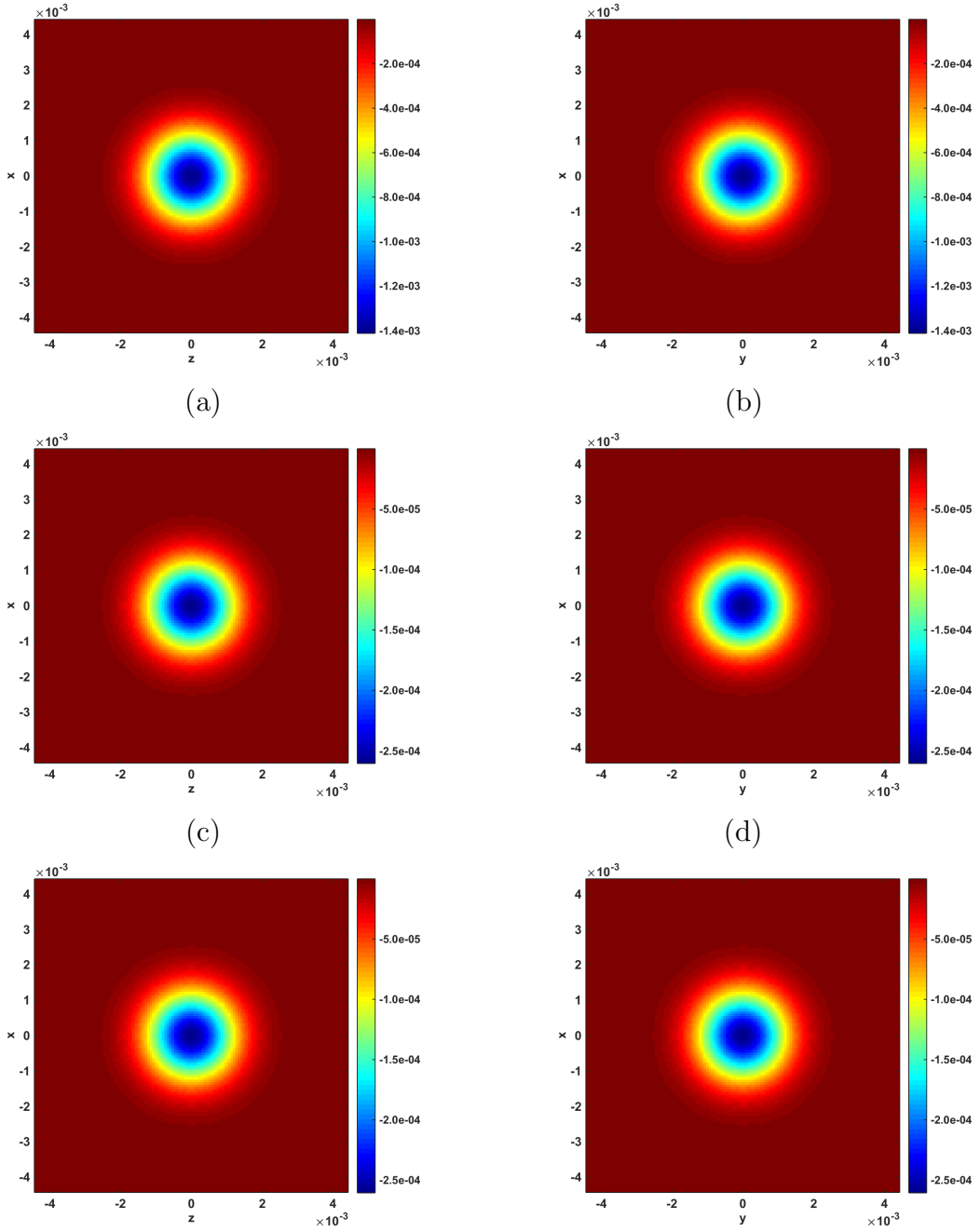


Figure C.3: Comparison of $\eta_\varphi(i,j,k)$ of Test Case 3: at plane $(:, N_y/2, :)$ (left) and $(:, :, N_z/2)$ (right) for the GF integral (a) (b), efficient IGF (c) (d), IGF (e) (f). The error is very small for all three methods. Yet, the two IGF methods still achieve an improvement of one magnitude.

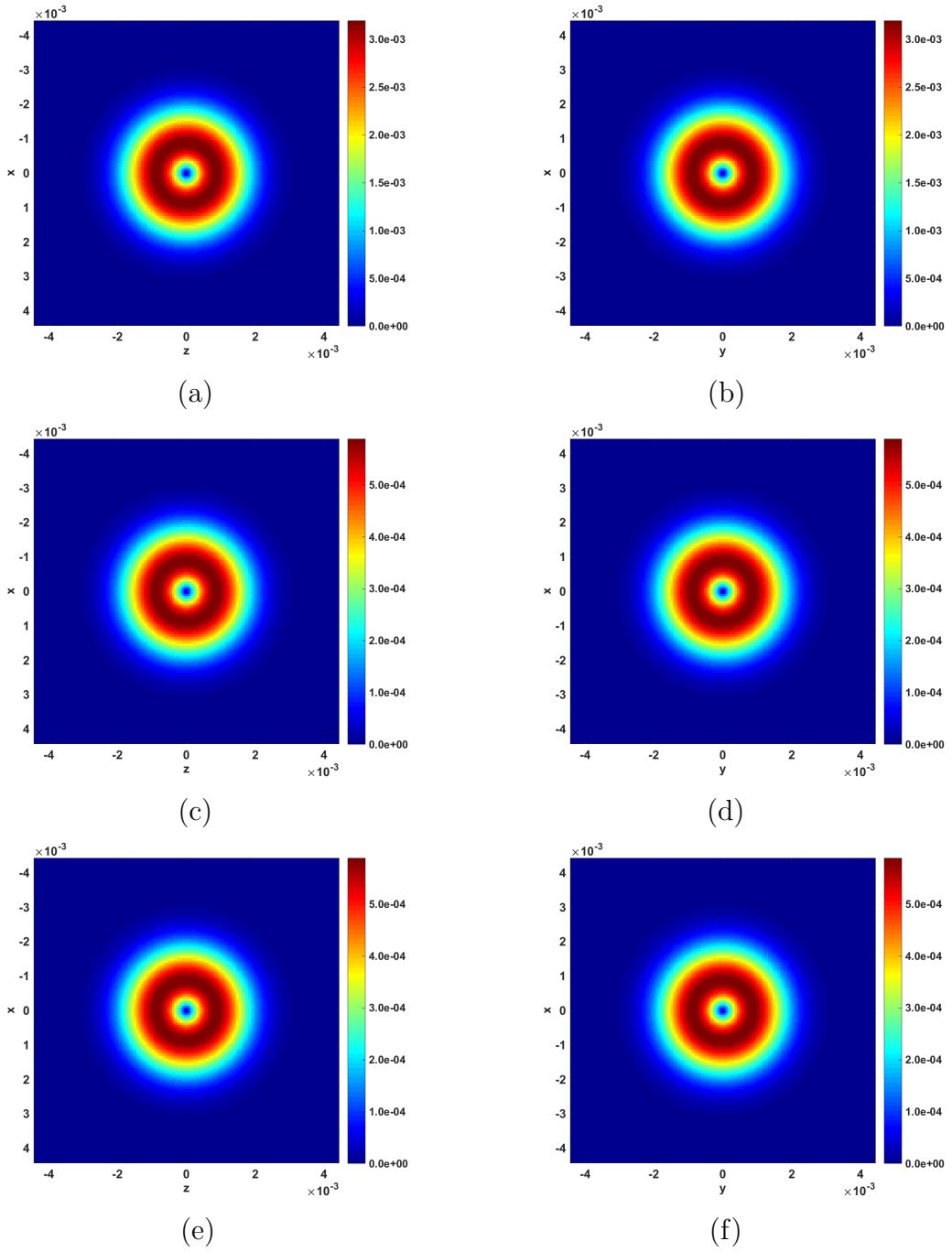


Figure C.4: Comparison of $\eta_{\mathbf{E}}(i,j,k)$ of Test Case 3: at plane $(:, N_y/2, :)$ (left) and $(:, :, N_z/2)$ (right) for the GF integral (a) (b), efficient IGF (c) (d), IGF (e) (f). The error is very small for all three methods. Yet, the two IGF methods still achieve an improvement of one magnitude.

Bibliography

[1] G. Aad and et al. Combined Measurement of the Higgs Boson Mass in pp Collisions at $\sqrt{s} = 7$ and 8 TeV with the ATLAS and CMS Experiments. *Phys. Rev. Lett.*, 114:191803, May 2015.

[2] A. Adelmann, P. Arbenz, and Y. Ineichen. A fast parallel Poisson solver on irregular domains applied to beam dynamics simulations. *Journal of Computational Physics*, 229(12):4554 – 4566, 2010.

[3] A. Adelmann and et al. OPAL, PSI, Villigen, 2016.

[4] A. Adelmann, Y. Ineichen, C. Kraus, T. Schietinger, S. Russell, and J. Yang. The Object Oriented Parallel Accelerator Library (OPAL). In *Particle accelerator. Proceedings, 23rd Conference, PAC'09, Vancouver, Canada, May 4-8, 2009*, page FR5PFP065, 2010.

[5] B. Athawes. A possible NUMA architecture highlighting local and remote access.

[6] B. Aune, R. Bandelmann, and et al. Superconducting TESLA cavities. *Phys. Rev. ST Accel. Beams*, 3:092001, Sep 2000.

[7] N. S. Bakhvalov. On the convergence of a relaxation method with natural constraints on the elliptic operator. *USSR Computational Mathematics and Mathematical Physics*, 6(5):101 – 135, 1966.

[8] A. Ben-Israel and T. N. E. Greville. *Generalized Inverses: Theory and Applications*. CMS Books in Mathematics. Springer, 2003.

[9] C. K. Birdsall and A. B. Langdon. *Plasma physics via computer simulation*. Taylor and Francis Group, 1985.

[10] OpenMP Architecture Review Board. The OpenMP API specification for parallel programming, 2016.

[11] P. Boonpornprasert. First Characterization of 4 nC Electron Beams for THz Studies at PITZ. *DPG-Frühjahrstagung*, AKBP 3.5, 2016.

[12] J. C. Bowman and M. Roberts. Efficient Dealiased Convolutions without Padding. *SIAM Journal on Scientific Computing*, 33(1):386–406, 2011.

[13] R. Bracewell. *The Fourier Transform and Its Applications*. McGraw-Hill, New York, USA, 1999.

[14] A. Brandt. Multi-level adaptive solutions to boundary-value problems. *Mathematics of Computation*, 31(138):333 – 390, 1977.

[15] W. L. Briggs, V. E. Henson, and S. F. McCormick. *A Multigrid Tutorial: Second Edition*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2000.

[16] H. Burau, R. Widera, W. Honig, G. Juckeland, A. Debus, T. Kluge, U. Schramm, T. E. Cowan, R. Sauerbrey, and M. Bussmann. PIConGPU: A Fully Relativistic Particle-in-Cell Code for a GPU Cluster. *IEEE Transactions on Plasma Science*, 38(10):2831–2839, Oct 2010.

[17] W. Cai. Potential Field of a Uniformly Charged Ellipsoid. *private article with open access*, 2007.

[18] A. W. Chao. *Physics of Collective Beam Instabilities in High Energy Accelerators*. John Wiley & Sons, Inc., 1993.

[19] P. G. Ciarlet. *The Finite Element Method for Elliptic Problems*. Classics in Applied Mathematics. Society for Industrial and Applied Mathematics, 2002.

[20] J. D. Cockcroft and E. T. S. Walton. Experiments with high velocity positive ions. *Proceedings of the Royal Society of London. Series A, Containing Papers of a Mathematical and Physical Character*, 129(811):477–489, 1930.

[21] Large Hadron Collider. European organization for nuclear research.

[22] M. E. Conway. A multiprocessor system design. In *Proceedings of the November 12-14, 1963, Fall Joint Computer Conference*, AFIPS '63 (Fall), pages 139–146, New York, NY, USA, 1963. ACM.

[23] J. W. Cooley and J. W. Tukey. An algorithm for the machine calculation of complex fourier series. *Mathematics of computation*, 19(90):297–301, 1965.

[24] V. K. Decyk and T. V. Singh. Adaptable particle-in-cell algorithms for graphical processing units. *Computer Physics Communications*, 182(3):641 – 648, 2011.

[25] F. Diacu. The solution of the n-body problem. *The Mathematical Intelligencer*, 18(3):66–70, 1996.

[26] K. O. Dimon. *Foundations of Potential Theory*. Springer-Verlag, Berlin, 1929.

[27] M. Dohlus and Ch. Henning. Periodic Poisson model for beam dynamics simulation. *Phys. Rev. Accel. Beams*, 19:034401, Mar 2016.

[28] N. Dugan, L. Genovese, and S. Goedecker. A customized 3D GPU Poisson solver for free boundary conditions. *Computer Physics Communications*, 184(8):1815 – 1820, 2013.

[29] ESS. European spallation source.

[30] R. P. Fedorenko. The speed of convergence of one iterative process. *USSR Computational Mathematics and Mathematical Physics*, 4(3):227 – 235, 1964.

[31] T. Flisgen. Plots of dual grids. private communication. 2014.

[32] K. Flöttmann. ASTRA, DESY, Hamburg, 2000.

[33] K. Flöttmann. Erratum: Some basic features of the beam emittance [phys. rev. st accel. beams **6** , 034202 (2003)]. *Phys. Rev. ST Accel. Beams*, 6:079901, Jul 2003.

[34] K. Flöttmann. Some basic features of the beam emittance. *Phys. Rev. ST Accel. Beams*, 6:034202, Mar 2003.

[35] V. Fuka. Poisfft – a free parallel fast poisson solver. *Applied Mathematics and Computation*, 267:356 – 364, 2015. The Fourth European Seminar on Computing (ESCO 2014).

[36] L. Greengard and V. Rokhlin. A fast algorithm for particle simulations. *Journal of computational physics*, 73(2):325–348, 1987.

[37] L. Greengard and V. Rokhlin. A new version of the fast multipole method for the Laplace equation in three dimensions. *Acta numerica*, 6:229–269, 1997.

[38] Y. N. Grigoryev and et al. *Numerical "Particle-In-Cell" Methods: Theory and Applications*. DE GRUYTER, Utrecht; Boston, 2002.

[39] ISC Group. Top 500 list, 2015.

[40] C. Gulliford, A. Bartnik, I. Bazarov, B. Dunham, and L. Cultrera. Demonstration of cathode emittance dominated high bunch charge beams in a DC gun-based photoinjector. *Applied Physics Letters*, 106(9), 2015.

[41] M. T. Heideman, D. H. Johnson, and C. S. Burrus. Gauss and the history of the fast fourier transform. *Archive for history of exact sciences*, 34(3):265–277, 1985.

[42] M. M. Hejlesen, J. T. Rasmussen, P. Chatelain, and J. H. Walther. A high order solver for the unbounded poisson equation. *Journal of Computational Physics*, 252:458–467, 2013.

[43] P. Henrici. Poisson's equation in a hypercube: Discrete fourier methods, eigenfunction expansions, padé approximation to eigenvalues. *Studies in Numerical Analysis, MAA Studies in Mathematics, G.H. Golub (Ed.)*, (24):371–411, 1984.

[44] R. W. Hockney and J. W. Eastwood. *Computer simulation using particles*. CRC Press, 2010.

[45] I. Hofmann and O. Boine-Frankenheim. Grid dependent noise and entropy growth in anisotropic 3d particle-in-cell simulation of high intensity beams. *Phys. Rev. ST Accel. Beams*, 17:124201, Dec 2014.

[46] L. Howes. *The OpenCL Specification Version: 2.1 Document Revision: 23*. Khronos OpenCL Working Group, 2015.

[47] E. Ising. Beitrag zur theorie des ferromagnetismus. *Zeitschrift für Physik A Hadrons and Nuclei*, 31(1):253–258, 1925.

[48] S. G. Johnson and M. Frigo. Implementing FFTs in practice. In C. Sidney Burrus, editor, *Fast Fourier Transforms*, chapter 11. Connexions, Rice University, Houston TX, September 2008.

[49] F. Kesting and G. Franchetti. Propagation of numerical noise in particle-in-cell tracking. *Phys. Rev. ST Accel. Beams*, 18:114201, Nov 2015.

[50] E. O. Lawrence. The evolution of the cyclotron. *Nobel Lecture*, 1951.

[51] E. O. Lawrence, N. E. Edlefsen, and G. N. Lewis. *On the production of high speed protons*. publisher not identified, 1930.

[52] S. Y. Lee. *Accelerator Physics*. World Scientific, Pub., 2011.

[53] A. Markoviḱ. *Simulation of the interaction of positively charged beams and electron clouds*. PhD. thesis, University of Rostock, Rostock, Germany, 2013.

[54] A. Markoviḱ, G. Pöplau, and U. van Rienen. Numerical computation of space-charge fields of electron bunches in a beam pipe of elliptical shape. *TESLA-Report 2005-21*, 2005.

[55] A. Markoviḱ, G. Pöplau, and U. van Rienen. The 3D Space Charge Field Solver MOEVE and the 2D Bassetti-Erskine Formula in the Context of Beam - E-cloud Interaction Simulations. *Proceedings of the* 11*th European Particle Accelerator Conference*, 2008.

[56] A. Markoviḱ, G. Pöplau, and U. van Rienen. 3D PIC computation of a transversal tune shift caused by an electron cloud in a positron storage ring. *Proceedings of the* 1*st International Particle Accelerator Conference*, 2010.

[57] A. Markoviḱ, G. Pöplau, U. van Rienen, and K. Flöttmann. Simulation of 3D Space-charge Fields of Bunches in a Beam Pipe of Elliptical Shape. *Proceedings of the* 10*th European Particle Accelerator Conference*, 2006.

[58] A. Markoviḱ, G. Pöplau, U. van Rienen, and R. Wanzenberg. Tracking Code with 3D Space Charge Calculations Taking into Account the Elliptical Shape of the Beam Pipe. *Proceedings of the* 9*th International Computational Accelerator Physics Conference*, 2006.

[59] J. C. Maxwell. A Dynamical Theory of the Electromagnetic Field. *Philosophical Transactions of the Royal Society of London*, 155:459–512, 1865.

[60] H. Merz. CUFFT 1.1 / 2.0 vs FFTW 3.1.2 (x86_64) vs FFTW 3.2 (Cell) comparison, http://www.sharcnet.ca/∼merz/cuda, 2016.

[61] C. Metzger-Kraus, M. Abo-Bakr, A. Adelmann, and B. Kuske. Latest Improvements of OPAL. In *Proceedings, 7th International Particle Accelerator Conference (IPAC 2016): Busan, Korea, May 8-13, 2016*, page WEPOY034, 2016.

[62] S. Meykopff. Parallelized ASTRA, DESY, Hamburg, 2014.

[63] G. Pöplau and U. van Rienen. A Self-Adaptive Multigrid Technique for 3-D Space Charge Calculations. *IEEE Transactions on Magnetics*, 44(6):1242–1245, June 2008.

[64] G. Pöplau and U. van Rienen. Efficient 3D Space Charge Calculations with Adaptive Discretization Based on Multigrid. *Proceedings of the* 1*st International Particle Accelerator Conference*, pages 1832–1834, 2010.

[65] G. Pöplau, U. van Rienen, and K. Flöttmann. New 3D space charge routines in the tracking code Astra. *Proceedings of the* 9*th International Particle Accelerator Conference*, pages 135–138, 2006.

[66] G. Pöplau, U. van Rienen, B. van der Geer, and M. de Loos. Multigrid Algorithms for the Fast Calculation of Space-Charge Effects in Accelerator Design. *IEEE Trans. Magn.*, 40(2):714–717, March 2004.

[67] G. Pöplau, U. van Rienen, S.B. van der Geer, and M.J. de Loos. A multigrid based 3D space-charge routine in the tracking code GPT. *Proceedings of the* 7*th International Computational Accelerator Physics Conference*, (175):281–288, 2005.

[68] J. Qiang. A high-order fast method for computing convolution integral with smooth kernel. *Computer Physics Communications*, 181(2):313–316, 2010.

[69] J. Qiang and et al. IMPACT, LBNL, Berkeley, 2016.

[70] J. Qiang, M. A. Furman, and R. D. Ryne. A parallel particle-in-cell model for beam-beam interaction in high energy ring colliders. *J. Comput. Phys.*, 198(1):278–294, July 2004.

[71] J. Qiang, S. Lidia, R. Ryne, and C. Limborg-Deprey. Three-dimensional quasistatic model for high brightness beam dynamics simulation. *Phys. Rev. ST Accel. Beams*, 9(4):044204, 2006.

[72] J. Qiang, S. Lidia, R. Ryne, and C. Limborg-Deprey. Erratum: Three-dimensional quasistatic model for high brightness beam dynamics simulation [phys. rev. ST accel. beams 9, 044204 (2006)]. *Phys. Rev. ST Accel. Beams*, 10(12):129901, December 2007.

[73] J. Qiang and R. D. Ryne. Parallel 3D Poisson solver for a charged beam in a conducting pipe. *Computer physics communications*, 138(1):18–28, 2001.

[74] J. Qiang, R. D. Ryne, S. Habib, and V. Decyk. An Object-Oriented Parallel Particle-in-Cell Code for Beam Dynamics Simulation in Linear Accelerators. *Journal of Computational Physics*, 163(2):434 – 451, 2000.

[75] Press release. LHC experiments bring new insight into primordial universe. November 2010.

[76] H. Risken and T. Frank. *The Fokker-Planck Equation: Methods of Solution and Applications*. Springer Series in Synergetics. Springer Berlin Heidelberg, 1996.

[77] D. Rossinelli, M. Bergdorf, G. Cottet, and P. Koumoutsakos. GPU accelerated simulations of bluff body flows using vortex particle methods. *Journal of Computational Physics*, 229(9):3316 – 3333, 2010.

[78] E. Rutherford. Lxxix. the scattering of α and β particles by matter and the structure of the atom. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 21(125):669–688, 1911.

[79] E. Rutherford and J. M. Nuttall. Lvii. scattering of α particles by gases. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 26(154):702–712, 1913.

[80] Y. Saad. *Iterative Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematics, second edition, 2003.

[81] B. F. Smith. Domain decomposition algorithms for the partial differential equations of linear elasticity. Technical report, Courant Institute„ 1990.

[82] P. N. Swarztrauber. Symmetric FFTs. *Math. Comp.*, 47(12):323 – 346, 1986.

[83] The finite element method: Its basis and fundamentals. In O.C. Zienkiewicz and et al., editors, *The Finite Element Method: its Basis and Fundamentals (Seventh Edition)*. Butterworth-Heinemann, Oxford, seventh edition edition, 2013.

[84] Ties Behnke, et al. (Eds.). The International Linear Collider Technical Design Report Volume 1: Executive Summary. Technical report, 2013.

[85] R. J. Van De Graaff. A 1,500,000 volt electrostatic generator. *Phys. Rev*, 38:1919–1920, 1931.

[86] R. J. Van De Graaff, K. T. Compton, and L. C. Van Atta. The electrostatic production of high voltage for nuclear investigations. *Physical Review*, 43(3):149, 1933.

[87] C. van Loan. *Computational frameworks for the fast Fourier transform*, volume 10, Frontiers in Applied Mathematics. SIAM, 1992.

[88] U. van Rienen. Finite integration technique on triangular grids revisited. *International Journal of Numerical Modelling: Electronic Networks, Devices and Fields*, 12(1-2):107–128, 1999.

[89] U. van Rienen. *Numerical Methods in Computational Electrodynamics*, volume 12. Springer, 2001.

[90] T. Weiland. On the Numerical Solution of Maxwell's Equations and Applications in the Field of Accelerator Physics. *Part. Accel.*, 15:245–292, 1984.

[91] T. Weiland. *Finite Integration Method and Discrete Electromagnetism*, pages 183–198. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003.

[92] T. Weiland and R. Wanzenberg. *Frontiers of Particle Beams: Intensity Limitations: Proceedings of a Topical Course Held by the Joint US-CERN School on Particle Accelerators at Hilton Head Island, South Carolina, USA 7–14 November 1990*. Springer Berlin Heidelberg, Berlin, Heidelberg, 1992.

[93] C. J. Weinstein. Quantization Effects in Digital Filters. Technical report, 1969.

[94] R. Wideröe. Improved injection system for magnetic induction accelerators, December 12 1950. US Patent 2,533,859.

[95] R. Wideröe. Collimator for beams of high-velocity electrons, January 4 1966. US Patent 3,227,880.

[96] H. Wiedemann. *Particle accelerator physics; 3rd ed.* Springer, Berlin, 2007.

[97] Wolfram. Mathematica Oxfordshire UK, 1987.

[98] XFEL. Deutsches Elektronen-Synchrotron.

[99] J. Xu, P. N. Ostroumov, and J. Nolen. A parallel 3D Poisson solver for space charge simulation in cylindrical coordinates. *Computer Physics Communications*, 178(4):290–300, 2008.

[100] D. Zheng, A. Markoviḱ, G. Pöplau, and U. van Rienen. Study of a Fast Convolution Method for Solving the Space Charge Fields of Charged Particle Bunches. *Proc. of 5th International Particle Accelerator Conference (IPAC 2014), Dresden, Germany*, pages 418–421, 2014.

[101] D. Zheng, G. Pöplau, and U. van Rienen. Efficiency optimization of a fast Poisson solver in beam dynamics simulation. *Computer Physics Communications*, 198:82–96, 2016.

[102] D. Zheng, G. Pöplau, and U. van Rienen. On several Green's function methods for fast poisson solver in free space. *Scientific Computing in Electrical Engineering (2014), Springer, Mathematics in Industry*, 23:91–99, 2016.

[103] D. Zheng and U. van Rienen. A Fast Poisson Solver for 3-D Space Charge Calculations in a CPU+GPU Heterogeneous Routine. *IEEE Transactions on Magnetics*, 52(3):1–4, 2016.

[104] D. Zheng, U. van Rienen, and J. Qiang. An improved parallel Poisson solver for space charge calculation in beam dynamics simulation. *Proc. of 12th International Computaional Accelerator Physics Conference (ICAP 2015), Shanghai, China*, 2015.

[105] F. Zulian. Diagram of a symmetric multiprocessing system.

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Dissertation mit dem Titel "Efficient Algorithms for the Fast Computation of Space Charge Effects Caused by Charged Particles in Particle Accelerators" selbstständig und ohne fremde Hilfe und nur unter Verwendung der von mir angegebenen Quellen und Hilfsmittel verfasst habe.

Rostock, 2. September 2016

Dawei Zheng

Curriculum Vitae

PERSONAL DATA

Dawei Zheng, male, born 25.09.1985 in Changchun, Jilin, P. R. China

EDUCATION

08/2001 - 06/2004 High school
Changchun experimental high school, Changchun, China

09/2004 - 07/2008 Bachelor Degree in Mathematics and applied Mathematics
School of Mathematics and Statistics,
Lanzhou University, Lanzhou, China

09/2008 - 06/2011 Master Degree in Computational Mathematics
School of Mathematics and Statistics,
Lanzhou University, Lanzhou, China

Since 10/2011 PhD. student in Electrical Engineering
Institut für Allgemeine Elektrotechnik,
The University of Rostock, Rostock

PROFESSIONAL

01/2015 - 07/2015 Short-term research scholar in Beam Physics
Accelerator Technology and Applied Physics Division,
Lawrence Berkeley National Laboratory, Berkeley, US