**Università degli Studi di Padova**
centro interdipartimentale "Centro Ricerche Fusione"

**Universidade de Lisboa**
Instituto Superior Técnico (IST)

**Università degli Studi di Napoli Federico II**

JOINT RESEARCH DOCTORATE
"Fusion Science and Engineering"

ERASMUS MUNDUS INTERNATIONAL DOCTORAL COLLEGE CYCLE XXIX
"Fusion Science and Engineering"

# GPGPU application in fusion science

| | |
|---|---|
| **Coordinator** | Paolo Bettini |
| **Supervisors** | Gabriele Manduchi |
| | Bernardo Carvalho |
| **Co-Promoter** | Horácio Fernandes |
| **PhD student** | Tautvydas Jeronimas Maceina |

Maceina

Padova, January 2017

JOINT Doctorate and NETWORK in Fusion Science and Engineering

Network Partners:

1. Instituto Superior Técnico (IST) Lisboa, Portugal;
2. Università degli studi di Padova, Italy;
3. Ludwig Maximilians University Munich, Germany.

In collaboration with:

1. Consorzio RFX, Italy;
2. IPP Garching, Germany.

# Contents

# Abstract

GPGPUs have firmly earned their reputation in HPC (**H**igh **P**erformance **C**omputing) as hardware for massively parallel computation. However their application in fusion science is quite marginal and not considered a mainstream approach to numerical problems. Computation advances have increased immensely over the last decade and continue to accelerate. GPGPU boards were always an alternative and exotic approach to problem solving and scientific programming, which was cultivated only by enthusiasts and specialized programmers. Today it is about 10 years, since the first fully programmable GPUs appeared on the market. And due to exponential growth in processing power over the years GPGPUs are not the alternative choice any more, but they became the main choice for big problem solving. Originally developed for and dominating in fields such as image and media processing, image rendering, video encoding/decoding, image scaling, stereo vision and pattern recognition GPGPUs are also becoming mainstream computation platforms in scientific fields such as signal processing, physics, finance and biology.

This PhD contains solutions and approaches to two relevant problems for fusion and plasma science using GPGPU processing. First problem belongs to the realms of plasma and accelerator physics. I will present number of plasma simulations built on a PIC (**P**article **I**n **C**ell) method such as plasma sheath simulation, electron beam simulation, negative ion beam simulation and space charge compensation simulation. Second problem belongs to the realms of tomography and real-time control. I will present number of simulated tomographic plasma reconstructions of Fourier-Bessel type and their analysis all in real-time oriented approach, i.e. GPGPU based implementations are integrated into MARTe environment. MARTe is a framework for real-time application developed at JET (**J**oint **E**uropean **T**orus) and used in several european fusion labs.

These two sets of problems represent a complete spectrum of GPGPU operation capabilities. PIC based problems are large complex simulations operated as batch processes, which do not have a time constraint and operate on huge amounts of memory. While tomographic plasma reconstructions are online (real-time) processes, which have a strict latency/time constraints suggested by the time scales of real-time control and operate on relatively small amounts of memory. Such a variety of problems covers a very broad range of disciplines and fields of science: such as plasma physics, NBI (**N**eutral **B**eam **I**njector) physics, tokamak physics, parallel computing, iterative/direct matrix solvers, PIC method, tomography and so on. PhD thesis also includes an extended performance analysis of Nvidia GPU cards considering the applicability to the real-time control and real-time performance.

In order to approach the aforementioned problems I as a PhD candidate had to gain knowledge in those relevant fields and build a vast range of practical skills such as: parallel/sequential CPU programming, GPU programming, MARTe programming, MatLab programming, IDL programming and Python programming.

# Sommario

Le GPGPU sono ormai affermate come hardware per il calcolo parallelo nel mondo HPC (**H**igh **P**erformance **C**omputing). Le applicazioni nel campo della scienza fusionistica sono tuttavia piuttosto marginali e non rappresentano il principale approccio alla soluzione di problemi numerici. I progressi nell'ambito computazionale sono stati enormi nell'ultimo decennio e continuano tutt'ora: le schede grafiche sono da sempre state una strada esotica per la soluzione di problemi di calcolo scientifico, prevalentemente percorsa da programmatori specializzati. A dieci anni di distanza dalla comparsa sul mercato della prima GPU completamente programmabile, grazie ad una crescita esponenziale della potenza di calcolo, queste schede rappresentano la principale alternativa per la soluzione di problemi computazionali importanti. Sviluppate in origine per l'utilizzo in campi quali il processamento di immagini, rendering, codifica/decodifica video, riconoscimento di pattern, le GPGPU stanno diventando le principali piattaforme di calcolo in ambiti scientifici quali fisica, finanza e biologia.

Questa tesi di dottorato affronta, utilizzando il calcolo su GPGPU, due importanti questioni legate alla scienza della fusione e del plasma. Il primo problema riguarda l'ambito della fisica del plasma e degli acceleratori. Presento un modello PIC (**P**article **I**n **C**ell) per simulare plasmi non magnetizzati, dimostrando la sua validità rispetto alla formazione dello strato di Debye, e la sua applicazione alla formazione di un fascio di ioni negativi e alla compensazione di carica spaziale. Il secondo problema è nell'ambito della tomografia e del controllo real-time. Presento diversi casi di ricostruzione tomografica simulata di tipo Fourier-Bessel, e la loro analisi con approccio real-time, con implementazione GPGPU integrata in ambiente MARTe, un framework siluppato a JET (**J**oint **E**uropean **T**orus) e usato in diversi laboratori europei.

Questi due problemi offrono un panorama completo delle capacità delle GPGPU. I codici PIC permettono simulazioni vaste, che durano molti giorni, ed impiegano enorme quantità di memoria. Le ricostruzioni tomogratiche sono operazioni da risolvere in tempo reale, usano poca memoria, con limiti sulla latenza dell'implementazione. I problemi studiati coprono diverse discipline e campi della scienza: fisica del plasma, fisica degli NBI (**N**eutral **B**eam **I**njector), fisica dei tokamak, parallel computing, solutori diretti ed iterativi, metodo PIC, tomografia e così via. La tesi include una analisi estesa delle performance delle schede Nvidia GPU, considerando l'applicabilità al controllo real-time.

Come candidato, per affrontare i problemi presentati ho acquisito competenze in questi ambiti, e raggiunto buona abilità nella programamzione seriale e parallela, applicata anche a GPGPU, e abilità specifiche legate ai framework MARTe, MatLab, IDL e Python.

# Part I
# Introduction

## 1 General overview on fusion science

The global energy demand is growing rapidly due to increasing world population and rising energy consumption per capita. The present global energy system relies heavily on fossil fuels, which supply almost 80% of the world's energy demand [1]. Current situation cannot last forever due to a simple fact, that natural recourses are finite. Another big reason is environmental issue. Fossil fuels and their products are driving Earth's ecological system to a disaster. It is expected, that soon more and more governmental restriction will be empowered upon fossil related industries. The current use of fossil fuels is facing many challenges and will be forced to reduce due to concerns about atmospheric pollution and resource depletion [2]. Renewable energy sources such as solar and wind power exist. Though at the moment they are not considered to be a real solution for super-high energy demands of modern industries. Price for renewable energy sources are too high and yield of energy production is low. Nuclear fission has played so far a very important role in the world's energy supply. However fission energy production has been compromised with horrific accidents such as Chernobyl (26 April 1986) and Fukushima (11 March 2011) and general public became very concerned about safety of nuclear fission plants. Historically the single biggest problem for fission was the disposal of radioactive waste, however distrust and lack of public endorsement became the other big issue for fission to successfully continue after Fukushima disaster.

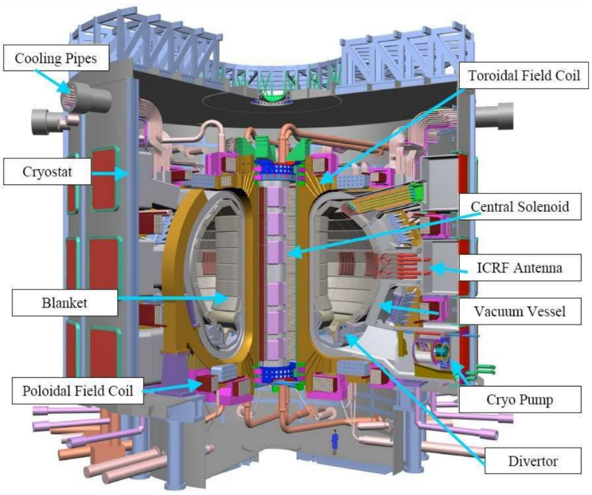
Nuclear fusion is considered to be a sound alternative source of energy because of several advantages, such as virtually limitless fuel supply, absence of greenhouse emission, suitability for large-scale electricity production and low levels of radioactive waste [3]. It seems, that fusion remains the only reasonable option for world's nearest development step. It is the only technological solution to yield huge amounts of energy with minute environmental pollution and with inherent safety against massive disasters. However fusion field comprises of many technological problems to be solved. It is in the state of though and complicated development process. That's why world still has to wait for a few decades for its commercial application.

The main goal of fusion is to harvest the energy, which is produced in the nuclear reaction of light nuclei fusing into one heavier nucleus. Fusion reactions, which would yield positive energy outcome, are only possible with lighter nuclei than Fe, because Fe has the greatest binding energy per nucleon among all elements. Heavier than Fe elements can only be in concern for fission reactions. The energy yielded in fusion reaction is contained in the mass difference between reactants and products of a fusion reaction, which is equivalent to energy emitted according to Einstein's relation $\Delta E = \Delta mc^2$. For light nuclei such as hydrogen isotopes energy per nucleon is very high. From all known reactions the most feasible one is the fusion of deuterium and tritium, which has the largest cross-section, i. e. highest probability to interact in collision:

$$^2_1\mathrm{D} +^3_1 \mathrm{T} \longrightarrow^4_2 \mathrm{He}\ (3.54\,\mathrm{MeV}) +^1_0 \mathrm{n}\ (14.05\,\mathrm{MeV}).$$

Deuterium is naturally abundant element due to fact, that the water of earth contains almost all of the earth hydrogen resources and 0.015% of it is deuterium isotope. However, tritium is not found naturally in Earth, thus it has to be transmuted from lithium (Li) and lithium is quite short in supply comparing to deuterium [4]. Disadvantages such as tritium handling and 80% energy conversion to neutrons are inherent properties of fusion technology. It is known to exist the fusion of two deuterium nuclei. In fact this reaction would be way more convenient than D-T reaction, because no tritium breeding and much less tritium handling would be necessary. However, this D-D reaction requires 30 times better plasma confinement. The two hydrogen isotopes fuse only when the kinetic collision energy is enough to overcome the Coulomb potential between the particles. This corresponds to temperatures of 100 million of Kelvins. It is clear, that none of known materials is able to sustain such a hot plasma, thus the plasma must be confined in other ways. The most advanced confinement concept known so far is called a *tokamak*. Tokamak is a torus shaped vessel, with a plasma confined inside by superposition of toroidal and poloidal magnetic field lines. These superposition lines follow the toroidal direction and simultaneously twist in the poloidal surface. Charged particles of plasma are gyrating around this superposition field line due to Lorentz force and thus are "contained" in the arbitrary volume determined by the radius of gyration. Nevertheless a part of confined plasma particles escapes the magnetic confinement and immediately collides with the wall, where it erodes/interacts the with armor materials such as tungsten,

beryllium or carbon.
So far fusion energy has been produced successfully in JET (**J**oint **E**uropean **T**orus) with 16.1 MW peak power output at efficiency value of 0.64 (produced power/input power) [5] and in TFTR (**T**okamak **F**usion **T**est **R**eactor) with 11 MW peak power output at efficiency value of 0.27 [6]. These experiments show the feasibility and potential of fusion, but any commercial realization is still far away. Next step in fusion development is ITER (**I**nternational **T**hermonuclear **E**xperimental **R**eactor) [7] (Fig. 1). ITER will have to demonstrate an inductive operation mode with efficiency value ≥10 and to sustain operation for 300—500 s. It will be working as a testing ground for plasma facing materials later to be used in DEMO (a DEMOnstration power plant). ITER is meant to be the last final step before building DEMO, which will be the first fusion power plant in operation.



| Parameter | Value |
|---|---|
| Produced fusion power | 500 MW |
| fusion power/heating power $Q$ | 10 |
| Plasma inductive burn time | 400 s |
| Plasma major radius $R$ | 6.2 m |
| Plasma minor radius $r$ | 2.0 m |
| Plasma current $I_\mathrm{p}$ | 15 MA |
| Safety factor $q$ | 3 |
| Toroidal magnetic field $B$ | 5.3 T |
| Electron density $n_\mathrm{e}$ | $10^{20}$ m$^3$ |
| Temperature $T$ | 20 keV |
| Energy confinement $\tau_E$ | 3.7 s |
| Neutral Beam Injector | 33 MW |
| Electron cyclotron antenna | 20 MW (170 GHz) |
| Ion cyclotron antenna | 20 MW (50 MHz) |
| Plasma type | deuterium-tritium |
| Plasma volume | 837 m$^3$ |

Figure 1: ITER

## 2 Plasma heating

In order to start a fusion reaction the plasma must reach temperatures higher than 10 keV. Up to date there exist 3 types of plasma heating.

**Ohmic heating** The "transformer principle". In a toroidal experiment the plasma current is driven by induced voltage of transformer action, the plasma being the secondary loop. If the plasma current is not varying with time, then the resistance of the plasma is $\frac{V_\phi}{I_\phi}$, where $V_\phi$ and $I_\phi$ are the toroidal loop voltage and plasma current respectively. The plasma resistivity is important, because it determines the ohmic heating input to the plasma. For a fully ionized plasma the resistivity is called *Spitzer resistivity* (Eq. 1) [8]:

$$\eta \approx \frac{\pi Z e^2 \sqrt{m_\mathrm{e}}}{(4\pi\varepsilon_0)^2 (k_\mathrm{B} T_\mathrm{e})^{\frac{3}{2}}} \ln \Lambda \tag{1}$$

where $T_\mathrm{e}$ is the electron temperature, $m_e$ is the electron mass, $\ln \Lambda$ is so-called Coulomb logarithm, $Z$ is ionization level. The resistivity decreases as $T_\mathrm{e}$ is rising, which makes ohmic heating inefficient and fail, when plasma electrons reach high temperatures. Ohmic heating usually exist in the early ramp-up phase during tokamak discharge, it raises the toroidal current $I_\phi$ and kick-starts the tokamak operation. It is the oldest plasma heating method.

**RF heating** The "microwave principle". The energy is transfered to the tokamak by powerful antennas using electromagnetic waves with frequencies corresponding to the resonant frequencies in the magnetized plasma. ECRH (**E**lectron **C**yclotron **R**esonance **H**eating) and ICRH (**I**on **C**yclotron **R**esonance **H**eating) are methods to exploit gyration resonances in electrons and ions respectivelly.

By targeting those frequencies and their harmonics the energy transfer is achieved:

$$\omega_{\mathrm{ce}} = \frac{eB}{m_{\mathrm{e}}} \qquad \omega_{\mathrm{ci}} = \frac{ZeB}{m_{\mathrm{i}}}, \tag{2}$$

where $B$ is magnetic field strength, $Z$ is ionization level, $m_{\mathrm{e}}$, $m_{\mathrm{i}}$ are electron and ion mases respectively. There exists another heating mehod called LHRH (**L**ower **H**ybrid **R**esonance **H**eating). It is targeting the oscillation of $\vec{E} \times \vec{B}$ drift waves and is called "hybrid", because its expression is derived from two cyclotron frequencies $\omega_{\mathrm{ce}}$ and $\omega_{\mathrm{ci}}$:

$$\omega_{\mathrm{LH}} = \frac{1}{\frac{1}{\omega_{\mathrm{pi}}^2 + \omega_{\mathrm{ci}}^2} + \frac{1}{\omega_{\mathrm{ce}}\omega_{\mathrm{ci}}}}, \tag{3}$$

where $\omega_{\mathrm{pi}} = \sqrt{\frac{n_{\mathrm{i}} Z^2 n_{\mathrm{i}}^2}{m_{\mathrm{i}} \varepsilon_0}}$ is ion plasma frequency. All the resonance frequencies are solutions of a dispersion relation for a cold magnetized plasma approximation and satisfy inequality:

$$\omega_{\mathrm{ci}} \ll \omega_{\mathrm{LH}} \ll \omega_{\mathrm{ce}} \approx \omega_{\mathrm{pe}}, \tag{4}$$

where $\omega_{\mathrm{pe}} = \sqrt{\frac{n_{\mathrm{e}} n_{\mathrm{e}}^2}{m_{\mathrm{e}} \varepsilon_0}}$ is electron plasma frequency.

**NBI heating** The "particle accelerator principle". NBI (**N**eutral **B**eam **I**njector) is a linear particle accelerator with an additional section of neutralization. Neutralization is necessary to achieve injection of energetic particles into plasma volume, since otherwise charged particle beams would be scattered and deviated by the strong magnetic fields at the plasma edge. By penetrating plasma the neutral beam is gradually ionized via collisions following one of the reactions [9] (Eq. 5, Fig. 2) and then thermalized with the background plasma. In this way energy and momentum is transfered from the beam to the plasma.

$$\begin{array}{lll}
\text{Charge exchange} & \mathrm{D} + \mathrm{D}^+ & \longrightarrow \mathrm{D}^+ + \mathrm{D} \\
\text{Ionization by ions} & \mathrm{D} + \mathrm{D}^+ & \longrightarrow \mathrm{D}^+ + \mathrm{D}^+ + \mathrm{e} \\
\text{Ionization by impurities} & \mathrm{D} + \mathrm{Z} & \longrightarrow \mathrm{D}^+ + \mathrm{Z} + \mathrm{e} \\
\text{Impurity charge exchange} & \mathrm{D} + \mathrm{Z}^+ & \longrightarrow \mathrm{D}^+ + \mathrm{Z} \\
\text{Ionization by electrons} & \mathrm{D} + \mathrm{e} & \longrightarrow \mathrm{D}^+ + 2\mathrm{e}
\end{array} \tag{5}$$

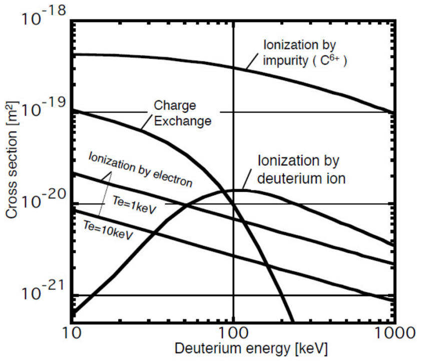


Figure 2: Beam energy dependence of charge exchange, ion, impurity and electron ionization cross-sections

Charge exchange is the dominant process in the low energy regime (<45 keV) and ionization by deuteruim ions is dominant in the high energy regime (>45 keV). NBI is also known to faciliate L–H transition in tokamaks.

# 3 Neutral beam injection systems

In order to produce a deuterium and tritium reaction (D+T→4He+n) one has to raise plasma temperature to the range of 10–30 keV. However to achieve effective heating the NBI beam must be much more energetic than that, typically in the range of 500 keV – 1 MeV. In analogy to NBI this corresponds to warming up cold water by pouring in hot water. There are two versions of NBIs: p-NBI and n-NBI. p-NBI uses positively charged particles and n-NBI uses negatively charged particles for acceleration and neutralization. First NBIs were p-NBIs constructed for ORMAK (**O**ak **R**idge Toka**mak**) [10] and ATC (**A**diabatic **T**oroidal **C**ompressor) [11] in 1974. First n-NBI was constructed for JT-60U in 1996 [12]. Improving accelerator technology allowed to increase beam energy, however it was discovered that positive ions loose neutralization efficiency at higher energies (Fig. 3) [13]. Therefore n-NBI concept was more useful than p-NBI and therefore chosen for ITER.

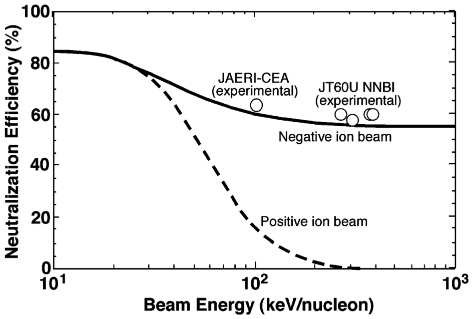


Figure 3: Neutralization efficiency of the ions

ITER NBI has to produce a beam of negatively ionized deuterium $D^-$ at 1 MeV producing 40 A current of 200 A/m$^2$ density and sustain it for 1000 s. The apparatus is constructed of beam formation sections such as ion source, accelerator, neutralizer, RID (**R**esidual **I**on **D**ump, calorimeter and peripheral modules such as cryo-pumps, vessels, fast shutter, duct, magnetic shielding, and residual magnetic field compensating coils (Fig. 4).

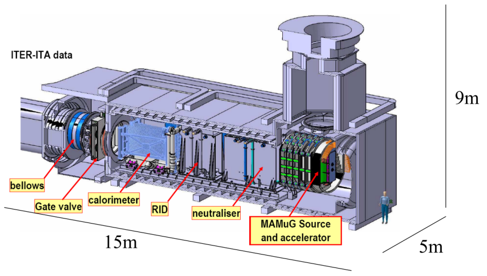


Figure 4: ITER NBI

MAMuG (**M**ulti-**A**perture **Mu**lti-**G**rid) is the five-stage electrostatic accelerator section, which was designed and tested at MTF (**Me**V **T**est **F**acility) in Japan [14]. Voltage holding capability and beam optics of the negative ion beam were studied and tested to achieve the stable acceleration of negative ion beam up to 1 MeV. Relatively recently MITICA (**M**egavolt **ITER** **I**njector **C**oncept **A**dvancement) was proposed as the upgrade for ITER NBI system, which has to produce a 55 A beam of D− and sustain it for 3600 s. Currently MITICA is under construction in Padova (Italy) at PRIMA (**P**adua **R**esearch on **ITER** **M**egavolt **A**ccelerator) facility under supervision of Consorzio RFX. As part of the PRIMA project another ITER-related experiment is being built simultaneously: SPIDER (**S**ource for the **P**roduction of **I**ons of **D**euterium **E**xtracted from an **R**F plasma). SPIDER is an ITER-scale radio-frequency negative ion source capable to produce plasma discharges of $H^-$ or $D^-$ at 15–30 eV. Together MITICA and SPIDER complete the experimental base for the design of final ITER HNB (**H**eating **N**eutral **B**eam) system.

## 3.1 Negative ion production

n-NBIs operation heavility depends on production of negative ions in the plasma. Production of negative ions are achieved in several ways:

**Volume production** Volume production occurs in a two step process. First a high-energy electron (≈10 eV) vibrationally excites the hydrogen molecule:

$$H_2 + e_{\text{fast}} \rightarrow H_2^* + e_{\text{slow}} \tag{6}$$

Later low-energy electron (≈2 eV) dissaciotively attaches itself to the vibrationally excited hydrogen molecule:

$$\mathrm{H}_2^* + \mathrm{e}_{\mathrm{slow}} \rightarrow \mathrm{H}^- + \mathrm{H} \tag{7}$$

Negative ions easily are destroyed by collisions with electrons with energies greater than 2 eV. Therefore ion source is divided into two regions: *driver* and *extractor*. In the driver the main plasma heating occurs (Eq. 6) and in the extractor the colder plasma produces negative ions (Eq. 7), which are then extracted for a beam formation. The plasma cooling can be obtained both by expansion of the source volume in the extraction region and by a transverse magnetic filter field, which reflects back the high energy electrons [15].

**Surface production** Surface production of negative ions is realized on the electrodes and structures that are in contact with the plasma. The energetic H and $\mathrm{H}^+$ ions are converted into $\mathrm{H}^-$ ions by surface interaction. For that reason materials with low work function are used to enhance production of negative ions. Cesium (Cs) is a metal with low work function $\phi < 2$ eV, therefore interaction of energy above 2 eV generates a negative ion. Research on negative ion surface production was started by Dudinikov [16] in 1973.

### 3.2 Beam formation

Negative ion beams are realized by three electrostatic grids: PG (**P**roduction **G**rid), EG (**E**xtraction **G**rid) and AG (**A**cceleration **G**rid) (Fig. 74). Negative ions are produced in volume and by PG surface. Then they are extracted by EG. By having constant magnetic field **B** present in the EG section co-extracted electrons are filtered from the beam. Magnetic field has also effect on electrostatic lensing and can cause divergence of negative ion beam. However some $\mathrm{H}^-$ ions has much higher divergence than average and form a *halo* beam ring around the main beam. The geometry of the PG (a circular hole in the plane) causes plasma to form a *meniscus*. Meniscus is a self-organized equipotential surface in 3D corresponding to the zero electric field. It is plasmas natural response to compensate the imbalance in electric field imposed by PG. Approaching meniscus from plasma bulk there is a pre-sheath region of plasma, which then becomes the sheath and finally meniscus itself. Produced by PG, extracted and formed by EG the beam goes to the acceleration phase in AG.

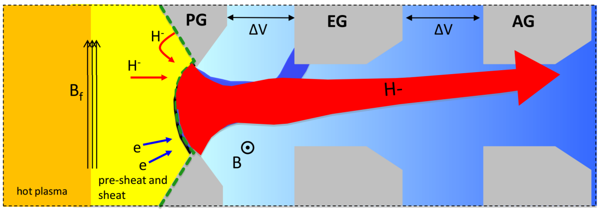


Figure 5: Schematics of extraction grids in an NBI accelerator section

# 4 Motivation of this thesis

Over the last 50 years fusion has seen some sustained yet gradual progress. Even though commercial fusion is several decades away, there are promising developments regarding the *triple product*. *Triple product* is the multiplication of three most fundamental tokamak plasma parameters: plasma density $n$, plasma temperature $T$ and confinement time $\tau_E$. Triple product determines how close to ignition plasma is:

$$nT\tau_E > 5 \cdot 10^{21} \left[\frac{\mathrm{keV}\dot{s}}{\mathrm{m}^3}\right] \tag{8}$$

Satisfying the Eq. 8 puts fusion plasma into ignition state, i. e. plasma, which is energetically self-sustainable. There will be no need of additional external heating, because the energy produced in the thermo-nuclear reactions will be equal or more than the energy needed to start the reaction. ITER is expected to ignite plasma. Since the 1960s, the fusion triple product has almost doubled every two years, a rate of increase comparable with the well-known progress in computer technology known as Moore's Law (Fig. 6). ITER is expected to upscale the triple product into ignition.

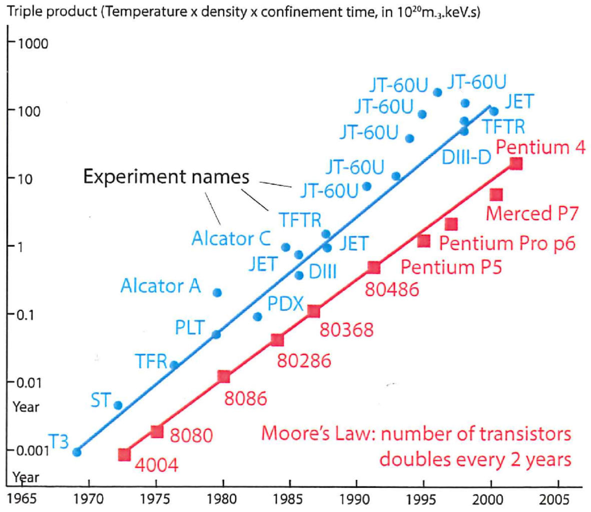


Figure 6: Triple product vs Moore's law

The fact, that fusion techonology is evolving exponentially, predicts great breakthroughs and makes it the technology of the future. The same can be said about computer technology, which has already demonstrated its capabilities of transforming the world. The motivation of the thesis is to present innovative solutions to the problems, which are one way or another linked to the progress of fusion technology, using GPGPU hardware. GPGPU technology is the latest manifestation of Moore's law in computer technology. By using cutting-edge GPGPU technology one strives to move closer to the next breakthrough in fusion evolution. This thesis will demonstrate, how solutions to relevant fusion topics, such as NBI physics and plasma tomography, can reach new quality with the help of GPGPU processing. For example NBI-related simulations are all performed in 3D, meanwhile in contrast the most of the current NBI-physics simulations in the scientific community are only in 2D. This already allows to avoid error-prone interpretations of 3D plasma physics based only on 2D simulation results. Another example is the real-time plasma tomography, where image resolution was always severely limited by real-time processing capabilities and absence of parallelism. GPGPU processing significantly pushes these limitations. Both of the aforementioned examples indicate, that GPGPU processing has plenty to offer to the fusion science, which brings a lot of optimism and motivation for this thesis.

Part II

# General Purpose GPU

## 5 History of computing

### 5.0.1 Moore's law

For the last half-century, computers have been doubling in performance and capacity every couple of years. This remarkable, continuous, exponential growth in computing performance has resulted in an increase by a factor of over 100 per decade and more than a million in the last 40 years. For example, the raw performance of a 1970's supercomputer is now available in a typical modern cell phone. This uninterrupted exponential growth is nowadays called *Moore's law* and was named after Gordon Moore (Intel co-founder), who in 1965 predicted that the number of transistors in integrated circuits would double every 18 months (Fig. 7). Due to incredible veracity of the prediction throughtout the years *Moore's law* became a societal expecation for increased technology performance and a drive for industries to sustain the exponential growth. Even though *Moore's law* was expected to fail many times in history, industries found new ways to sustain it and the trend still is present in 2016.

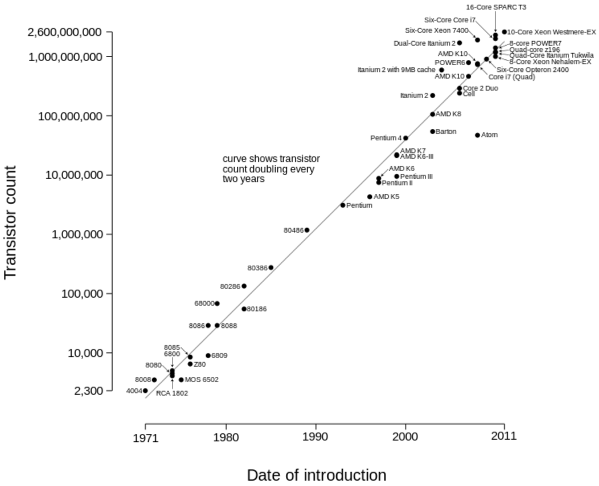


Figure 7: Number of transistors per CPU increasing at a geometric progression with time

### 5.0.2 Outlook of modern CPU

CPU (**C**entral **P**rocessing **U**nit) is the central electronic circuitry within a computer that carries out the instructions of a computer program by performing the basic arithmetic, logical, control and input/output operations specified by the instructions. For the most part of 90's and 00's systems based on a single CPU, e.g. such as Intel Pentium, Intel Xeon, AMD Opteron and AMD Athlon, were dominating consumer markets and drove rapid performance increases and cost reductions in computer applications. These processors enabled desktop systems to perform at range of several GFLOPS (**G**iga **FLO**ating-**P**oint operations per **S**econd) and cluster servers at hundreds of GFLOPS. Supported by *Moore's law* the relentless drive for performance improvement encouraged the software to evolve and sophisticate rapidly, which in turn demanded even more CPU performace creating a positive feedback for the computer industry. During the drive, most software developers have relied on the advances in hardware to increase the speed of their applications, i.e. the same software simply runs faster as each new generation of processors is introduced. However such a drive could not continue forever and reached its natural ceiling around 2003. The improvements to solely increase number of instructions per second in single-processor slowed down. Due to fundamental limits of power efficiency the integrated circuits built on CMOS (**C**omplementary **M**etal **O**xide **S**emiconductor) technology are no longer able to support increasing energy consumption and to manage heat dissipation. Since CMOS technology is virtually omnipresent in all computer components (microprocessors, microcontrollers, RAM, data converters, data transceivers and digital logic circuits), it became an obstacle to increase performance of components by increasing the clock frequency of the circuits. A CMOS transistor is a field-effect transistor, having a metal gate electrode placed on top of an oxide insulate, which is in turn is on top of a semiconductor material. Power density of CMOS circuit is given by:

$$P = NCV^2f \tag{9}$$

where $N$ – number of CMOS gates per unit area, $C$ – capacitive load per CMOS gate, $V$ – supply voltage, $f$ – clock frequency. Miniaturization of circuits naturally means higher $N$ and higher $C$, therefore $V$ and $f$ must be compromised in order to keep power density $P$ under a certain threshold value, where air cooling is still effective. In practice 130 W is considered a borderline for modern CPU to be cooled with air and approaching 130 W requires massive heat sinks and local fans. In reality reductions in transistor size still continue apace and much more transistors are being packed onto chips, which means that *Moore's law* will be supported in the near future. However a significant shift happened in design architecture of processors. In 2001 first dual-core processor (POWER4) was released by IBM. In May 2005 Intel and AMD released their first dual-core processors. Since then processors started acquiring more cores and today up to 24 physical cores can be present in a processor (Inter Xeon E7-8890 v4). Yet high number of cores does not univocally mean scaled performance, but it does play significant role in achieving optimal performance for the particular purpose of a processor. Fig. 8 clearly indicates the shift in processor industry, that is seen as a plateau in clock frequency and power consumption and a decreased slope in relative performance, while *Moore's law* still holding strong.

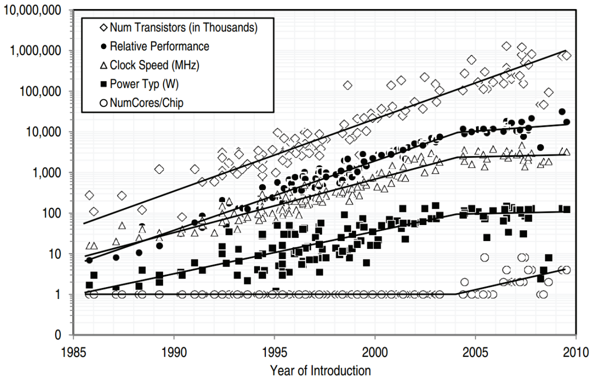


Figure 8: Number of transistors, frequency, power, performance and number of cores over the years [17]

### 5.0.3 Evolution of programming model

The change from single-core to multicore processors required a concomitant change in the software programming model. In order to use multicore processors efficiently, applications must be built using a *parallel programming model*, which divides a program into parts that are then executed in parallel. However in 00's much software today was written according to a *sequential programming model*, and applications written this way cannot easily be sped up by using multicore processors. The applications software that will continue to enjoy performance improvement with each new generation of microprocessors are the parallel programs, in which multiple threads of execution cooperate to complete the work faster. The incentive to switch from sequential to parallel program development is sometimes reffered as the *concurrency revolution*. Anyhow the practice of parallel programming is by no means new. The HPC (**H**igh **P**erformance **C**omputing) community has been developing parallel programs for decades. These programs run on large-scale computers and expensive clusters. Yet only a small fraction of public could practice parallel programming due to limited access to such systems. Now that all new processors are multicore systems, the number of applications that must be developed as parallel programs has increased dramatically. However there is fundamental limitation for code speedup, when upgrading sequential codes to parallel. Any parallel processing system has a theoretical limitation on speedup called

Amdahl's law (Fig. 9) [18], which cannot be surpassed by any means.

$$S(p,n) = \frac{1}{(1-p)+\frac{p}{n}}, \tag{10}$$

where $S(p,n)$ is the speedup measure, $p$ is the portion of parallelizable code, $n$ is the number of processing units (cores, threads). The larger the parallelizable code portion $p$ is, the higher the theoretical speedup is $S$. If the the application has little of parallelizable code $p$, even high number $n$ of processing units will not benefit the speedup. Amdahl's law suggests, that different problems have different levels of possible speedup and in order to benefit from parallelization some problems has (if possible) to be represented and "recast" in a more parallelizable formulations.

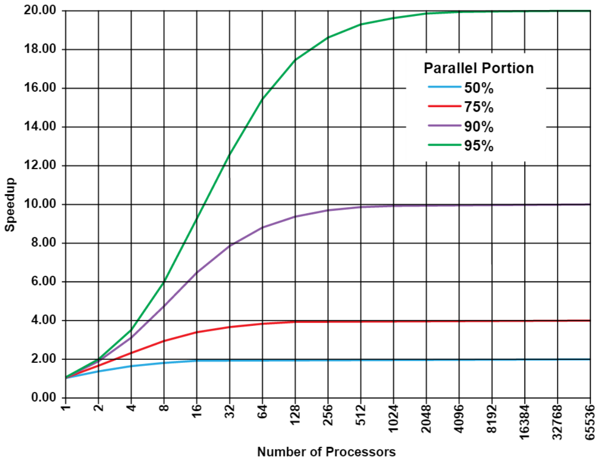


Figure 9: Amdahl's law

## 5.1 Emerging parallel computing technologies of today

### 5.1.1 GPGPU

GPGPU (**G**eneral **P**urpose **G**raphics **P**rocessing **U**nit) is relatively new device, which evolved from powerful GPU gaming cards in 2006. It is a device based on GPU architecture, but it is used for general computing instead of graphics (a dedicated GPGPU does not have a monitor port). Provided rapid evolution GPGPUs have firmly entered supercomputing domain. There are now a couple leading GPU supercomputers: Titan (USA, 3rd in TOP500), Piz Daint (Switzerland, 8th in TOP500)[19]. Usually GPGPUs are equiped with a bit slower clock speed than gaming GPUs, but instead they tend to have more processing cores and bigger (several tens of GB) RAM memory. GPGPUs are not stand-alone systems. They work in coordination with board CPU, which has to copy data and offload operations to GPGPU. Therefore GPGPU is considered an accelerator device (Fig. 10, 11).

Figure 10: Nvidia Tesla K40

Figure 11: FirePro S9300x2

### 5.1.2 Intel Xeon Phi

Intel Xeon Phi is new kind of device in computing community since 2012. It is a PCIe (**P**eripheral **C**omponent **I**nterconnect **E**xpress) board, which consists of up to 72 Intel cores connected in a MIC (**M**any **I**ntegrated **C**ores) architecture. It can be operated in two modes: *native* and *offload*. *Native* mode runs as independent a mult-core processor, i.e. coprocessor mode. *Offload* mode runs as dependent device, which can receive offloaded operations and data and execute them separately from motherboard CPU, i.e. accelerator mode. This device is thought to be the primary competitor for GPGPUs in HPC domain.

Figure 12: Intel Xeon Phi

### 5.1.3 ARM processors

ARM processor (Fig. 5.1.3) is a processor of RISC (**R**educed **I**ncstruction **S**et **C**omputing) architecture. A RISC-based design means processors require fewer transistors than typical CISC (**C**omplex **I**nstruction **S**et **C**omputing) processors (Intel, AMD). The architecture provides low cost, low heat and low power consumption, which is particularly suitable for use in portable devices, such as laptops, tablets, notepad computers and other embedded systems. This makes them the most widely used 32-bit microprocessor family in the world (Fig. 14). Evolution of ARM processors brings more and more computation power to the portable devices inluding a parallel computation. The new 64-bit ARM processors are expected to compete heavily with Intel and AMD in portable device market.

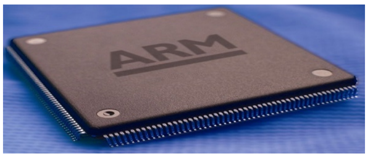


Figure 13: ARM processor

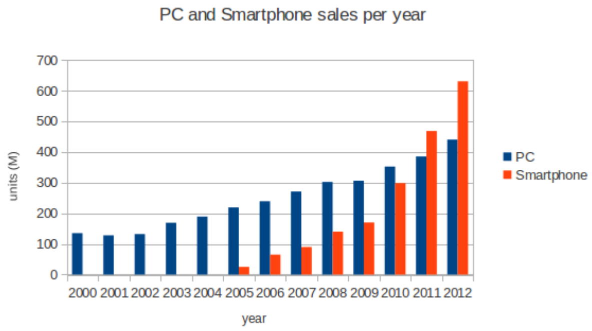


Figure 14: PC and Smarphone sales per year [20]

# 6 GPGPU

## 6.1 Birth of GPGPU

Not long after first multicore processor was released by IBM (Tab. 15) the race for performace and throughput was taken over by programable GPUs from Nvidia and ATI. Programable GPUs evolved from 3D graphic cards, which were being mainly designed and developed for gaming industry. They were programable with OpenGL and DirectX, but didn't have a dedicated API (**A**pplication **P**rogramming **I**nterface) for general purpose computing. First GPGPUs appeared in 2006 with the releases of Nvidia GeForce 8800 series and ATI Radeon X1900XTX series. Main purpose of these cards was gaming, nonetheless they had first dedicated APIs for general purpose computing: CUDA (**C**ompute **U**nified **D**evice **A**rchitecture) and FireStream respectively. Table 15 gives perspective of a timeline of events in computing industry, which preceded the first GPGPU release.

| Year | Month | Company | Model | Comment |
|---|---|---|---|---|
| 1999 | August | Nvidia | GeForce 256 | first GPU from Nvidia |
| 2001 | October | IBM | Power4 | first ever dual-core CPU |
| 2002 | August | ATI | Radeon 9000 | first GPU from ATI |
| 2005 | April | AMD | Athlon 64 X2 | first dual-core CPU from AMD |
| 2005 | May | Intel | Pentium D | first dual-core CPU from Intel |
| 2006 | January | ATI | Radeon X1900 | first FireStream version released |
| 2006 | November | Nvidia | GeForce 8800 | first CUDA version released |
| 2007 | May | Nvidia | Tesla C870 | first dedicated GPGPU |

Figure 15: The timeline of significant hardware releases marking evolution towards first GPGPU

This era of fierce competition between rival companies (Intel vs AMD in CPU industry and Nvidia vs ATI in GPU industry) ignited a very rapid development of domestic computer capabilities and it is clearly marked with concurrent hardware and software releases by rival companies. Driven by the same insatiable relentless market demand for performance (that drove CPU industry too) the programmable GPUs throughout years has evolved into a highly parallel, multithreaded, multicore monster processors with tremendous computational horsepower and very high memory bandwidth (Fig. 17).

## 6.2 GPU architecture

A modern GPU architechture is built around an array of SMs (**S**treaming **M**ultiprocessors) (Fig. 16). Each SM consists of an array of SPs (**S**treaming **P**rocessors), that share control logic and instruction cache among themselves. SMs are organized into sub-arrays, where the number of SMs in a sub-array can vary from one generation of GPUs to another generation. Concept of **S**treaming **M**ultiprocessor can be broken down to its parent concepts of *stream processing* and *multiprocessing*. *Stream processing* is a computational concept, where computation takes place automatically based on data flows and propagation of change. *Multiprocessing* is the use of two or more processing units within a single computer system. In general *stream processing* and *multiprocessing* refer to SIMD (**S**ingle **I**nstruction **M**ultiple **D**ata) and MIMD (**M**ultiple **I**nstruction **M**ultiple **D**ata) processing paradigms respectively. Therefore architecture of a GPU is a cross-design of architectures of both SIMD and MIMD origins.

SIMD model was first introduced by Cray machines[21], which were first supercomputers ever and they were equiped with VPUs (**V**ector **P**rocessing **U**nits) [22]. Nowaways Cray computers acquired the name of *vector machines* and SIMD is used to describe *streaming architectures*, where not one but several operations can be applied to the data vector. Modern GPU achitecture has close relation with both *vector processors* (Cray architecture) and *stream processors* (von Neumann architecture). Yet the true ancestors of GPUs are 3D graphics cards, which evolved from geometry and rasterization engines of the Silicon Graphics (SGI) workstations of the 1990s [23]. SGI workstations featured four heavily pipelined geometry and raster engines in order to perform massive amount of identical operations that exist within graphics applications, consisting of vertex transformations and shading, followed by scan-conversion with texture mapping interpolation. Both Cray computers and SGI machines employed the SIMD model, which is very close to what GPUs use today [24]. However a feature, that was completely lacking in these early SGI platforms, was the ability to freely program SIMD processors. All one could do was only set the parameters used in the graphics processing operations, such as lighting model, depth-buffer mode and so on. To enable this programmability, SGI founded the OpenGL graphics language which is still in use today, now accompanied by its rival DirectX from Microsoft.

The fact that GPU contains not one but an array of SMs makes it also a MIMD system. During operation SMs are heavily threaded as a MIMD system and each SM performs its part of workload as a SIMD-like system. It is not exactly a SIMD process, because data is NOT organized into registers of SSE (**S**treaming **SIMD** **E**xtentions) vectors (as in vectorizable CPUs). Instead execution and branching behaviour is specified by SIMT (**S**ingle **I**nstruction **M**ultiple **T**hread) model [25]. In SIMT all threads process data in their own registers. There is much less burden on the programmers in terms of data collection and data packing. In contrast with SIMD vector operations, SIMT enables programmers to

write thread-level parallel code for independent, scalar threads, as well as data-parallel code for coordinated threads. In SIMT threads are launched in groups of 32 called *warps*. Substantial performance improvements can be achieved by taking care that calculation does not follow different execution paths (i.e. there is no *thread divergence*) within a *warp*. Otherwise threads in a *warp* are serialized and performance is severely degraded.
Nowadays GPUs come with several gigabytes of GDDR (**G**raphics **D**ouble **D**ata **R**ate) DRAM (**D**ynamic **R**andom **A**ccess **M**emory). These GDDR DRAMs differ from the system DRAMs on the CPU motherboard in that they are essentially the frame buffer memory that is used for graphics. For graphics applications, they hold video images and texture information for 3D rendering, but for computing they function as very-high-bandwidth, off-chip memory, though with somewhat more latency than typical system memory. For massively parallel applications the higher bandwidth makes up for the longer latency.

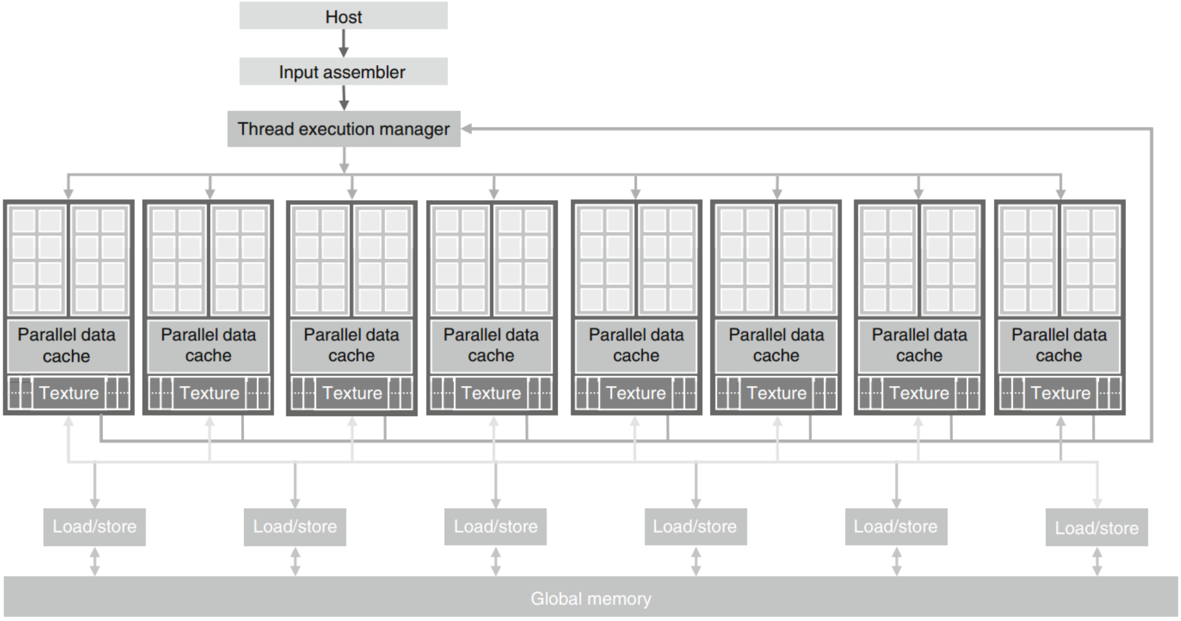


Figure 16: Architecture of a GPU: 16 SMs × 8 SPs (SMs are organized in groups of 2, that share cache memory)

## 6.3 GPU vs CPU

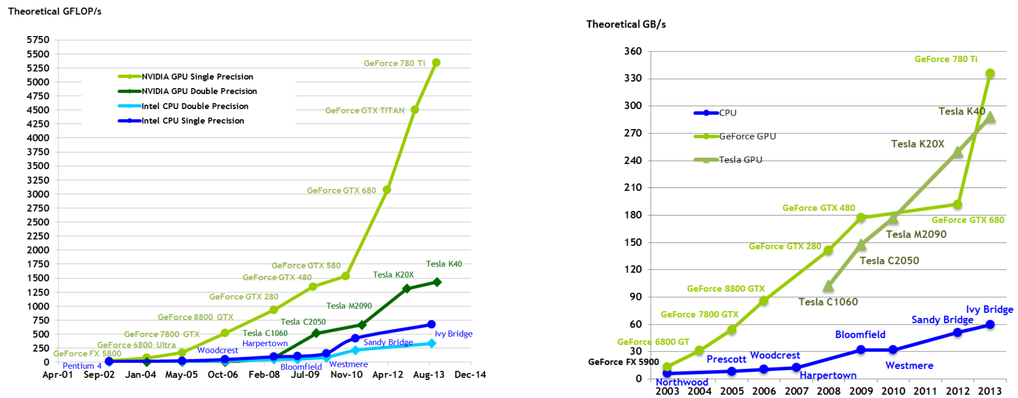
GPUs lead CPUs by a lot in terms of FLOPS (**FLO**ating-**P**oint **O**perations **p**er **S**econd) and MB (**M**emory **B**andwidth)(Fig. 17). One might ask why there is such a large performance gap between GPUs and multicore CPUs. The answer lies in the differences in the fundamental design philosophies between the two types of processors.

### 6.3.1 Task parallelism

The design of a CPU is optimized for sequential code performance. It makes use of sophisticated control logic to allow instructions from a single thread of execution to execute in parallel or even out of their sequential order while maintaining the appearance of sequential execution. More importantly, large cache memories are provided to reduce the instruction and data access latencies of large complex applications. Neither control logic nor cache memories contribute to the peak calculation speed [26]. Multicore CPUs are perfect to implement codes, which employ *task parallelism.*

### 6.3.2 Data parallelism

GPU is specialized for compute-intensive, highly parallel computation and therefore designed such that more transistors are devoted to data processing rather than data caching and flow control. The design philosophy of the GPUs was shaped by the fast growing video game industry, which exerted tremendous



(a) **FLO**ating-**P**oint **O**perations **p**er **S**econd (FLOPS)

(b) **M**emory **B**andwidth (MB)

Figure 17: The performance comparison between modern GPU and CPU [25]

economic pressure for the ability to perform a massive number of floating-point calculations per video frame in advanced games. This demand motivated the GPU designers to look for ways to maximize the chip area and power dedicated to floating-point calculations. The curent solution is to optimize the execution throughput by having massive numbers of threads. While executing a large number of inependent threads, the hardware tries to hide long-latency memory accesses by grouping them, therefore reducing the control logic required for each execution thread. Small cache memories are provided to help control the bandwidth requirements, so that multiple threads that access the same memory data do not need to all go to the DRAM. As a result, much more chip area is dedicated to the floating-point calculations. The architecture of GPU is suitable for solving problems with built-in *data parallelism.* In Fig. 18 the dedicated chip area is depicted in relative manner, which indicates the main architectural difference between CPU and GPU.

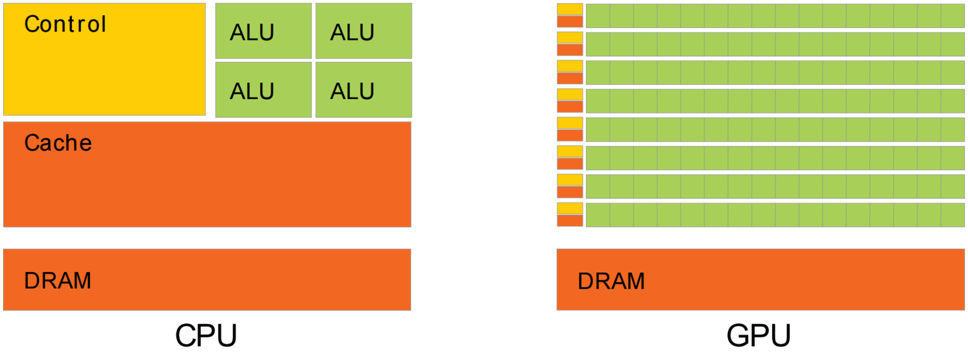


Figure 18: The GPU devotes more transistors to ALU (**A**rithmetic **L**ogic **U**nit) [25] compared to Control and Cache units

It should be clear now that GPUs are designed as numeric computing engines, and they will not perform well on some tasks on which CPUs are designed to perform well. Therefore one should expect that most effcient approach is to use both CPUs and GPUs, executing the sequential parts on the CPU and numerically intensive parts on the GPUs. This is why programming models such as CUDA, OpenCL and and general heterogeneous programming are aiming to support joint CPU/GPU execution of an application.

# 7 Programming GPUs

## 7.1 CUDA

CUDA (**C**ompute **U**nified **D**evice **A**rchitecture) is a programming framework developed by NVIDIA which allows programmers to easily use Nvidia GPUs for general purpose computation [27]. The CUDA programming language is an extension to the well-known C language suit. As a programming language it introduced an abstraction layer in order to hide the hardware layer. This layer of abstraction simplifies the code development for programmers and makes the code portable with respect to different GPUs. CUDA abstraction consists of three key elements: thread hierarchy, memory hierarchy and barrier synchronization, which are introduced as a minimal set of language extensions in order to maintain low learning curve for programmers familiar with standard C programming. CUDA allows applications to easily scale their parallelism to the full occupancy of SMs throughtout all variety of Nvidia hardware: GeForce, Quadro and Tesla board families (Fig. 19).

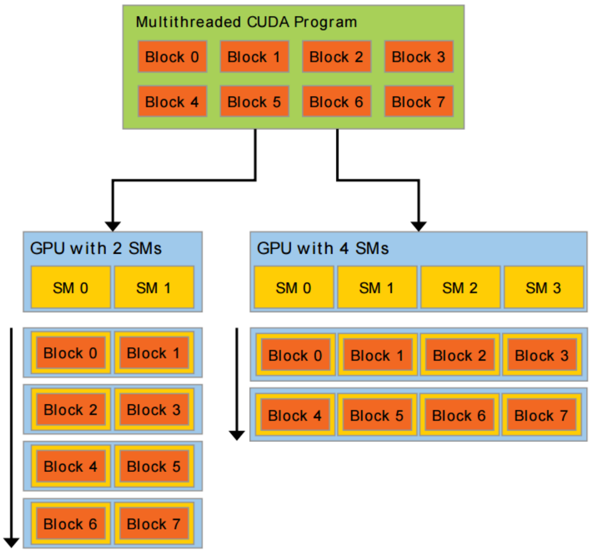


Figure 19: Executable code is scheduled accross SMs in blocks, which run independently. Therefore a GPU with more multiprocessors will automatically execute faster than a GPU with fewer SMs

### 7.1.1 Thread hierarcy and barrier synchronization

CUDA application is an extented C/C++ application, which consists of a sequential CPU code (host code) and parallel GPU code (device code). CPU (host) takes control of execution flow and initiates parallel execution on the GPU side (device). The GPU code is organized into *kernels*. A *kernel* is a SPDM (**S**ingle **P**rogram **M**ultiple **D**ata) computation executed by a large number of *threads* running in parallel. To efficiently manage large *thread* population, the SM employs a SIMT model, which executes *threads* in groups of 32 called *warps* [28]. Programatically *threads* are organized into *blocks* and *blocks* are organized into a *grid* (Fig. 20). *Thread*, *block* and *grid* are the abstractions of CUDA model, which can be thought of as virtualized scalar processor, virtualized multiprocessor and virutalized array of multiprocessors respectively. *Threads* can be indexed with 1D, 2D or 3D identifiers within a single *block* and *blocks* can be indexed with 1D or 2D identifiers within a *grid*. Therefore any *thread* can be unquivocally indentified by its thread ID and its block ID. Each *block* in a *grid* has same number of *threads* and same organization of *threads*. Synchronization of *threads* is feasible only within a single *block* (but NOT among several *blocks*) by barrier operation `__syncthreads()` and global synchronization among all *blocks* is reached only with *kernel* completion. Maximum number of *threads* in a *block* is 1024 (recent boards) and maximum number of *blocks* is virtually infinite, because *blocks* are indepedendently scheduled on SMs until execution request is completed.

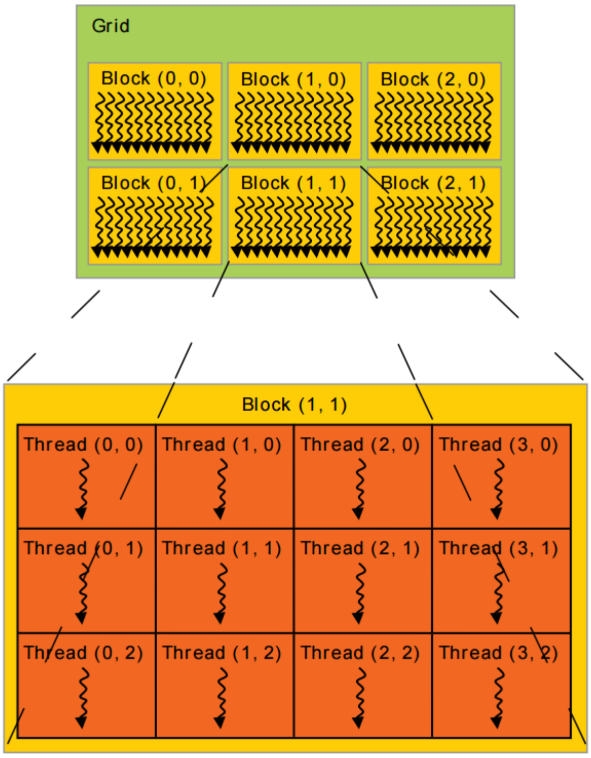


Figure 20: Grid of thread blocks [25]

### 7.1.2 Memory hierarcy

Under CUDA programming model the GPU has 4 types of memory: register memory, shared memory, global memory and constant memory (Tab 1, Fig. 21). Both register memory and shared memory reside on chip, thus are the fastest in access. Global memory is the bank usually of several gigabytes, which resides off chip and is the slowest in access. Each of them has a special purpose in the GPU. Register memory (also called private or local) is the memory available to each thread and cannot be accessed by any other thread. It stores variables, which are specific to that thread, such as thread ID, block ID and any other private variables significant for the thread. Shared memory is designed to store data that can be shared among threads belonging to the same block. It enables thread communication and is extremely useful in calculations based on mathematical reduction, such as matrix operations. Global memory is accessible by all threads and host. It is used to copy data from/to host and usually stores the user data of input and output of the application. Since global and shared memories have difference in speed of access by an order of magnitude or more, many levels of optimization can be achieved by careful tuning of parameters such as amount of shared memory, number of threads per block and number of streams. For example maximizing accesses of shared memory over the global memory leads to significant performance gains. Specific in its use constant memory is designed to store texture data for graphics rendering, where one must assure complete accesibility for all threads in all grids and still maintain low access latency. Usually variables in constant memory are meant to last the whole application execution and are read-only. Physically it is a combination of an off-chip global memory, which is frequently cashed for effcient access.

| Memory type | Location | Size | Accesibility | Comment |
|---|---|---|---|---|
| **Register** | on-chip | 16kB | thread | stores tread ID, block ID and user-defined private variables, very fast, L1 cache |
| **Shared** | on-chip | 48kB | block | enables data sharing and communication between threads within a thread block, very fast, L2 cache |
| **Global** | off-chip | several gigabytes | grid, host | main memory bank, slow, GPU RAM |
| **Constant** | on/off-chip | 64kB | all grids | optimized for texture storage for graphics rendering, read-only |

Table 1: GPU memory types of CUDA programming model

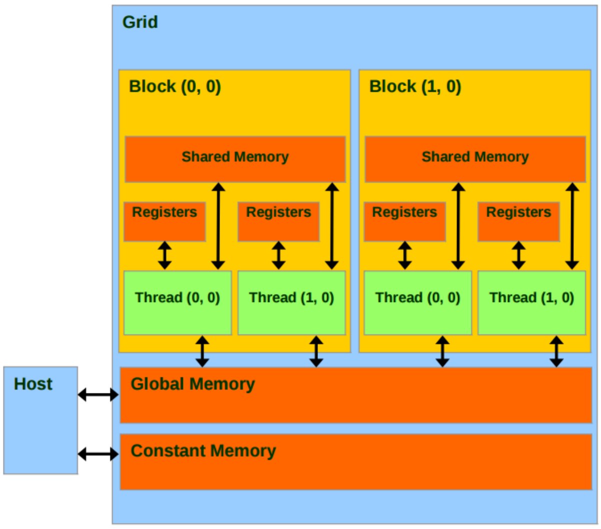


Figure 21: GPU memory hierarchy under CUDA programming model

### 7.1.3 Global memory operation modes

Global memory (GPU RAM) can be operated in several different ways. The mode of operation is determined by the type of allocation: pageable, page-locked, write-combined and directy-mapped (Tab. 2). Pageable allocation is the default type of allocation and the most commonly used. It is the type of allocation that is freely managed by the system, i.e. it can be moved and reallocated by internal system processes, which are transparent to the user. Page-locked allocation does not allow the system to page the memory, also called “pinned” allocation, i.e. it is fixed to a physical memory address that does not change. In most systems page-locked allocation results in speed up of applications, since system does not perform paging on that memory and data can copied faster via DMA (**D**irect **M**emory **A**ccess) circuit without involvement of CPU. Write-combined allocation is special type of allocation, which is a good option for one-way transfer from CPU to GPU. Directly mapped allocation is a type of allocation that allows tying CPU RAM and GPU RAM directly. In this way the programmer does not have to manually code the memory transfer from host to GPU anymore. System will manage copying automatically, i.e. whatever gets written in CPU RAM will automatically appear on GPU RAM.

| Operation mode | Comment |
|---|---|
| **Pageable** | Default type of allocation, freely moved and paged by internal system processes |
| **Page-locked** | “pinned”, expected to be faster than pageable mode, accessible via DMA |
| **Write-combined** | Special type, good for one-way transfer from CPU to GPU |
| **Directly-mapped** | Mirrors CPU RAM to GPU RAM, system manages copying automatically, keen to build up the latency due to multiple calls of copy operations of small size instead of one big transfer |

Table 2: GPU global memory operation modes

# 8 Other GPU programming models

## 8.1 OpenCL

OpenCL (**O**pen **C**omputing **L**anguage) is an open programming framework developed by Khronos Group [29], whose main goal is to provide unified programming platform across all modern computing devices: CPUs, GPUs and DSPs. OpenCL is aiming to establish itself as standart programming model for heterogeneous parallel processing systems. because todays's computers often include highly multicore CPUs, GPUs and other types of processor in one system. Even homogenous parallel processing systems are often underexploited due to intricate specificity related to the hardware, platform and vendor. OpenCL is trying to provide an solution to that and moreover offer an uniform approach to the heterogeneous parallel processing. The challenge of having uniform heterogeneous programming model accross all the hardware is immense, primarily because traditional programming approaches of each hardware are very different among themselves, independently evolved and proven by time. CPU-based parallel programming models typically assume a shared memory space and do not encompass vectorized processing. GPGPU-based progamming models are built around complex memory hierarchies and vectorization and are very platform-, vendor- and hardware-specific. All these non-uniformities build limitations for a developer to access full computational potential of heterogeneous systems. Nowadays heterogenous systems are omnipresent in all classes of devices: handheld, portable devices (phones, tablets, laptops), desktop computers and HPC computers. OpenCL abstraction consists of layers: platform layer, execution layer, memory layer.

### 8.1.1 Platform layer

Platform layer defines a *host* and *compute device*, which consists of *compute units*, which consists of *processing elements* (Fig. 22). An OpenCL application is implemented as both *host* code and *device* kernel code. The host code portion of an OpenCL runs according to the models native to the host platform. The host code invokes the kernel code execution to OpenCL devices. OpenCL devices can be systems of completely different origin and architectures, however the same kernel code has to run on all those systems, which means kernel code has to have a separate binary representations for each system, which means kernel code has to be compiled separately for each system. To adress this issue a developer can exploit a distinguised feature of OpenCL: *online compilation.* Device code can be compiled during run-time of host program and launched for each device as different binaries. This is option is the default one, requires minimum effort of the developer and guaranties maximum portability of the source code. However developer can choose *offline compilation* for the device code, which requires to have executables (binaries) already compiled into object files before application launch. It can be useful when the programmer is sure that standard OpenCL *online compilation* is underexploiting the hardware and it can be fully exloited by obtaining binaries from hardware-targeted external compilation. *Offline compilation* also tends to run faster, thus it is more suitable for real-time processing. However if the application is intended to be used on various platforms (various *compute devices*), multiple kernel binaries must be included, thus increasing the size of the executable file.

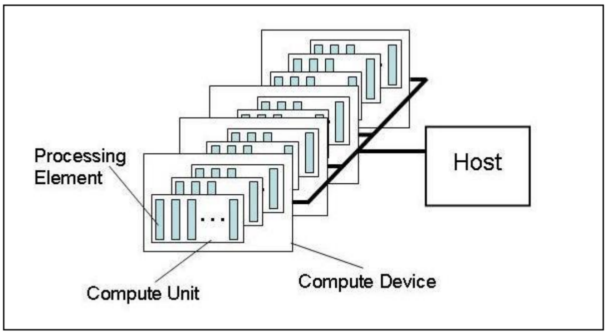


Figure 22: Platform model: hierarchy and organization of OpenCL hardware

#### 8.1.2 Execution layer

Execution layer is defined in terms of two distinct units of execution: *kernels* and a *host program.* Functions from the OpenCL API enable the host to interact with a device through a command-queue. In addition to commands submitted from the host command-queue, a kernel running on a device can enqueue commands to a device-side command queue. This results in *child kernels* enqueued by a kernel executing on a device (*parent kernel*). Each command-queue is associated with a single device. Regardless of whether the command-queue resides on the host or a device, each command passes through six states (Tab. 3, Fig. 23).

| State | Description |
|---|---|
| **Queued** | The command is enqueued to a command-queue. A command may reside in the queue until it is flushed either explicitly or implicitly by some other command |
| **Submitted** | The command is flushed from the command-queue and submitted for execution on the device. Once flushed from the command-queue, a command will execute after any prerequisites for execution are met |
| **Ready** | All prerequisites constraining execution of a command have been met. The command is placed in a device work-pool from which it is scheduled for execution |
| **Running** | Execution of the command starts. One or more work-groups associated with the command start to execute |
| **Ended** | Execution of a command ends. All of the work-groups associated with that command have finished their execution. Work-groups associated with the parent kernel are visible to other units of execution, but not the work-groups associated with the child kernels. Writing data to global memory happens |
| **Complete** | The command and also its child commands have finished execution |

Table 3: GPU global memory operation modes

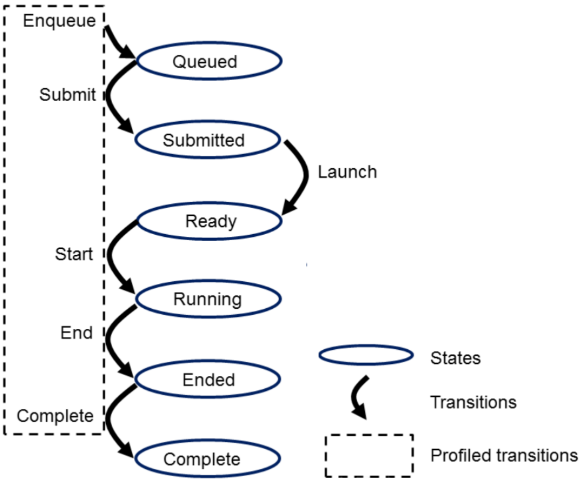


Figure 23: Execution model: states and transitions between states

A submitted kernel defines an *index space* and then executes for each index from the work-pool. Each of these executions is called a *work-item*. *Work-items* are grouped into *work-groups*. *Work-groups* are identified by their *work-group ID* are can be divided into *sub-groups*, which provide an additional level of control over execution. *Work-items* are identified by their *global ID* and *local ID*. *Global ID* is a 1D, 2D or 3D identifier, which points to an index coordinate in the*index space* and *local ID* is a 1D, 2D or 3D identifier, which points to an index coordinate within *work-group* (Fig. 24). There is a clear analogy between OpenCL execution model and CUDA thread hierarcy, where *work-item*, *work-group*, *work-pool* correspond to *thread*, *block*, *grid* respectively. Nevertheless there is a slight difference in managing *index space* between OpenCL and CUDA. In OpenCL programmer specifies only the size of the *index space* and OpenCL manages automatically the size of the *work-group* and allows *work-groups* to be of different size, while in CUDA a programmer decides the size of the *block* and the number of *threads* within the *block*. Size of the *block* remains fixed throughout *kernel* execution. In this sense CUDA is more robust and OpenCL is more programming-friendly. OpenCL reduces the programming burden by managing size of the *work-group* automatically, however finding the right size of the CUDA *block* can be a matter of fine-tuning and improved algorithm mapping to the hardware.

Synchronization in OpenCL is quite equivalent to synchronization in CUDA. Synchronization in OpenCL is provided by barrier operations, which may only synchronize *work-items* in the same *work-group*. There is no way to synchronize among different *work-groups* of a kernel instance. Synchronization within a *work-group* is obtained by barrier operations `barrier(CLK_LOCAL_MEM_FENCE)` or `barrier(CLK_GLOBAL_MEM_FENCE)`, which force to flush the data stored in variables to local or global memory respectively. Global synchronization among all *work-groups* is reached only by a kernel completion.

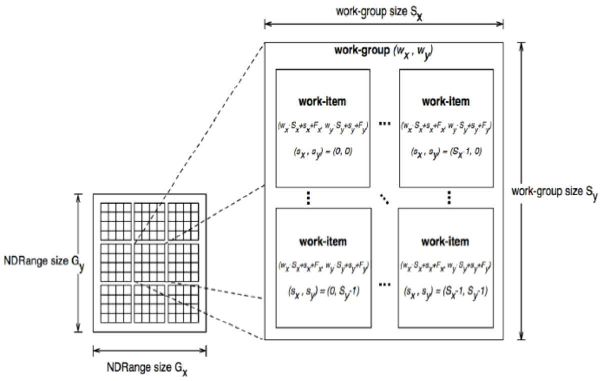

work-group size $S_x$
work-group $(w_x, w_y)$
work-item
$(w_x \cdot S_x+s_x+F_x, w_y \cdot S_y+s_y+F_y)$
$(s_x, s_y) = (0, 0)$
work-item
$(w_x \cdot S_x+s_x+F_x, w_y \cdot S_y+s_y+F_y)$
$(s_x, s_y) = (S_x-1, 0)$
work-item
$(w_x \cdot S_x+s_x+F_x, w_y \cdot S_y+s_y+F_y)$
$(s_x, s_y) = (0, S_y-1)$
work-item
$(w_x \cdot S_x+s_x+F_x, w_y \cdot S_y+s_y+F_y)$
$(s_x, s_y) = (S_x-1, S_y-1)$
work-group size $S_y$
NDRange size $G_y$
NDRange size $G_x$

Figure 24: An example of an 2D *index space* partitioning to *work-groups* and *work-items*, where $(g_x, g_y)$ is the *global ID*, $(s_x, s_y)$ is the *local ID*, $(w_x, w_y)$ is the *work-group ID*, $(G_x, G_y)$ is the size of the *index space*, $(S_x, S_y)$ is the size of *work-group* and $(F_x, F_y)$ is *global ID* offset assuming the case of *work-groups* having the same size $S_x \times S_y$ and offset $(F_x, F_y)$ is equal to zero $(0, 0)$

### 8.1.3 Memory layer

Under OpenCL programming model the abstract device has 4 types of memory: private memory, local memory, constant memory and global memory (Tab. 4, Fig. 25). Memory organization and hierarcy of OpenCL resembles a lot the CUDA memory model. However memory types in OpenCL are defined less as a hardware memory (like in CUDA), but more of a logical structure, due to the fact that OpenCL memory layer has to be applicable to a wide variety of devices, not only GPUs. The four named address spaces available to a device are logically disjoint meaning they do not overlap, however certain implementations may choose to let these disjoint named address spaces share physical memory. Another eminent feature of OpenCL is SVM (**S**hared **V**irtual **M**emory). SVM makes memory addresses to be meaningful among the host and all of the devices within a context. It logically extends a portion of the global memory into the host address space giving *work-items* access to the host address space. SVM must be supported by hardware of the OpenCL device. It enables efficient data sharing among OpenCL devices and host.

| Memory type | Accesibility | Comment |
|---|---|---|
| **Private** | work-item, single device | stores work-item ID, work-group ID and user-defined private variables |
| **Local** | work-group,single device | enables data sharing and communication between work-items within a work-group |
| **Constant** | work-pool, all devices | a region of global memory that remains constant during the execution of a kernel-instance, host allocates and initializes memory objects placed into constant memory, read-only from the kernel side |
| **Global** | work-pool, host, all devices | main memory storage, all-round read/write access accesibility, may be cached depending on the capabilities of the device |

Table 4: Device memory types of OpenCL programming model

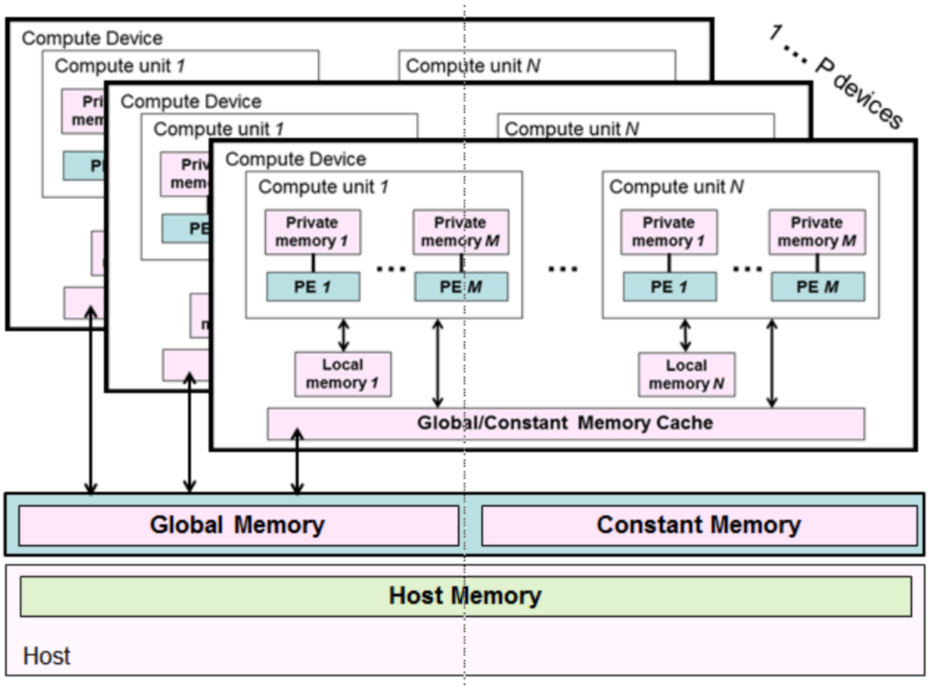


Figure 25: Memory layer under OpenCL programming model

## 8.2 OpenMP

OpenMP (**O**pen **M**ulti-**P**rocessing) is an API that supports multiplatform shared memory multiprocessing programming in C, C++, and Fortran for on most platforms, processor architectures and operating systems. It consists of a set of compiler directives, library routines and environment variables that influence run-time behavior. OpenMP is managed by the nonprofit technology consortium OpenMP Architecture Review Board (OpenMP ARB), jointly defined by a group of major computer hardware and software vendors such as: AMD, IBM, Intel, Cray, HP, Fujitsu, Nvidia, NEC, Red Hat, Texas Instruments, Oracle Corporation and more. The goal of OpenMP is to provide a portable, scalable model that gives programmers a simple and flexible interface for developing parallel applications for platforms ranging from the standard desktop computer to the supercomputer. OpenMP was first released in 1997 and is well-known to easily upgrade existing sequential codes and allowing incremental parallelisation. The latest upgrade OpenMP 4.0 in July 2013 introduced support for accelerator devices including GPUs.

### 8.2.1 Execution model

OpenMP manages threading based on a Fork and Join model (Fig. 26) [30]. All OpenMP programs begin as a single process called *master thread*. The *master thread* executes sequentially until a parallel region is encountered. At this point the master thread "forks" into a number of parallel *worker threads*. The instructions in the parallel region are then executed by the team of *worker threads*. At the end of the parallel region, the threads synchronise and join to become the single *master thread* again.

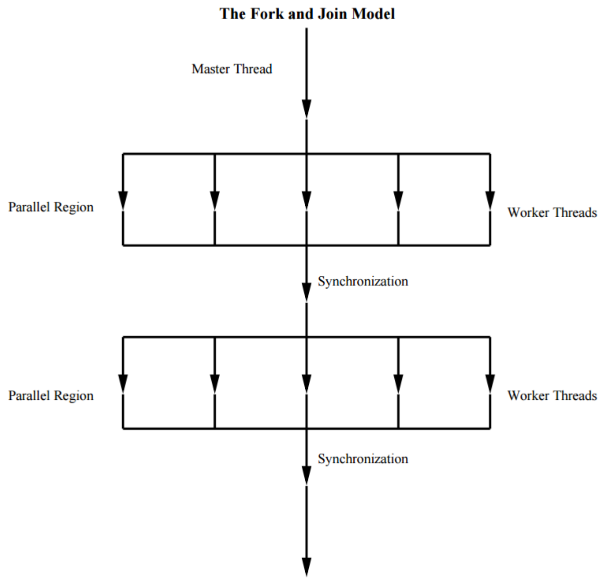


Figure 26: The Fork and Join Model

### 8.2.2 Memory model

The OpenMP API is based on shared memory model. All OpenMP threads have access to *memory* to store and retrieve variables. Each thread is allowed to have its own *temporary view* of the *memory.* The *temporary view of memory* represents any kind of intervening storage, such as machine registers, cache, other local storage, between the thread and the *memory.* The *temporary view of memory* allows the thread to cache variables and thereby to avoid going to *memory* for every reference to a variable. Beside *memory* and *temporary view* each thread also has access to its *threadprivate memory*, which cannot be shared or accessed by other threads.

# 9 Assessment of General Purpose GPU systems in real-time control

The recent advance of GPU technology is offering great prospects in computation. However, the penetration of the GPU technology in real-time control has been somewhat limited due to two main reasons:

1. Control algorithms for real-time applications involving highly parallel computation are not very common in practical applications

2. The excellent performance in computation of GPUs is paid for a penalty in memory transfer. As a consequence, GPU applications for real-time controls suffer from an often unacceptable latency

The code was designed to test latency and jitter in dense matrix-vector multiplications and memory transfer in order to mimic a large state-space based control algorithm. Result were analyzed to see where GPU computation excels and where it falls behind in order to give useful hints to designers facing the option of using either a multi-threaded, multicore CPU application or a GPU.

## 9.1 Introduction

General Purpose GPUs have firmly earned their reputation in High Performing Computation as hardware for massively parallel computation. Yet GPU application in Real-Time problem solving is just starting to develop. GPUs are currently widely used in modelling and simulation and recently are gaining attention for real-time applications. The control of complex phenomena driving the behavior of the plasma during a discharge may require sophisticated algorithm with high requirements in computation and therefore not feasible in practice until few years ago. GPUs in real-time application are being considered, but often the performance gain in computation is paid for the overhead in data transfer. At any cycle, in fact, the control system must acquire data from sensor and send the results of the control computation to actuators. For this reason, determining whether there could be a gain using GPUs in real-time application is not straightforward, even if the implemented algorithm can be parallelized. The aim of this work is to highlight the potentials and limits of the usage of GPUs in real-time applications. In particular, it will be shown that a common use case in realtime control can be considered the “border line” between classes of applications where the use of GPUs improves performances and those where other approaches, such as multicore CPUs yield better results.

## 9.2 Practical consideration of using GPUs in real-time applications

GPU accelerators work in conjunction with a CPU and therefore execution of the GPU kernels must be supervised by a program in the real-time application. Since the supervising program is running within a real-time context, it has to adhere to the common rules for real-time programming. Using a realtime framework represents a preferable solution over coding from scratch every application. In our applications, MARTe (**M**ultithreaded **A**pplication **R**eal-**T**ime **e**xecutor) has been used [31]. MARTe is a C++ framework that provides a development environment for the design and deployment of real-time applications. The kernel of MARTe comprises a set of data-driven independent blocks, connected via a shared bus. A MARTe application is designed by configuring and connecting a series of blocks named GAMs (**G**eneric **A**pplication **M**odule). Using MARTe, the developer can concentrate only on the computation involved in the real-time application, i.e. develop or reuse application-specific GAMs, being all the required data plumbing carried out by the framework in a very efficient way. GPU computation will therefore be included in the operation carried out by one or more GAMs used in the application. The integration is however not straightforward. Firstly MARTe does not yet provide an interface to access GPU resources and secondly CUDA code can only be compiled with a proprietary compiler `nvcc` from Nvidia. However these issues can be overcome by external compilation. CUDA commands have been wrapped into C++ functions and compiled externally with `nvcc` to an object file. Then the object is linked to a MARTe GAM during compilation of the GAM. In such a way GAM is able to call wrapper functions, which contain CUDA code that was compiled externally by `nvcc`. GAM instances carrying out GPU functionality are then connected within MARTe to form the required data-logical network carrying out the real-time application.

## 9.3 An example of Real-Time GPU application

There are a number of real-time domains where GPUs may be applied. GPU can efficiently carry out digital signal processing operations and matrix operations of large size. These operations, coupled with other GPU-efficient algorithms, can be used in medical imaging, video processing and data processing. As member of fusion community hereby one presents a real-time application developed for plasma control purposes on the basis of GPU computation. The application runs an algorithm of plasma tomography on ISTTOK tokamak [32] (Fig. 27). Signals are produced by 3 detectors with linear arrays of 16 sensors each sensitive to the intensity of plasma emission. Each cycle a signal of 48 channels (3x16) arrives to the data acquisition system.

DAR (**D**ata **A**cquisition and **R**econstruction) real-time system is based on ATCA (**A**dvanced **T**elecommunications **C**omputing **A**rchitecture) technology. DAR at ISTTOK uses two ATCA modules, one is the acquisition board and the other is an interface from a standard PC motherboard to the ATCA bus via the PCI-Express port. The GPU is installed on the PC motherboard in an adjacent PCI-Express port.

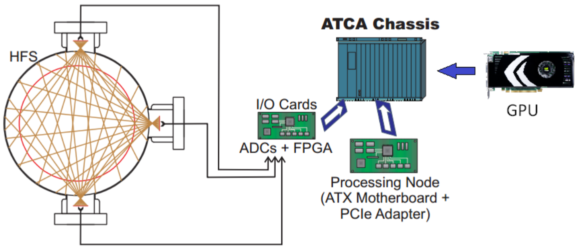


Figure 27: Tomography data acquisition ATCA system installed in ISTTOK

The signal $\vec{f}$ is received by the MARTe framework and immediately copied to the GPU, where it is multiplied with so-called pseudo-inverse contribution matrix $\mathbf{C}^{+}$ yielding tomographic coefficient vector $\vec{a}$:

$$\vec{a}_{[6]} = \mathbf{C}^{+}_{[6,48]} \cdot \vec{f}_{[48]} \tag{11}$$

Then the so-called *synthetic* signals are calculated by the original un-inversed contribution matrix $\mathbf{C}$:

$$\tilde{\vec{f}}_{[48]} = \mathbf{C}_{[48,6]} \cdot \vec{a}_{[6]} \tag{12}$$

The goal is to choose the best-fitting subset of coefficient vector $\vec{a}$. Selection is performed by calculating statistical parameter $\chi^2$ and then finding the minimum of $\chi^2$:

$$\chi^2 = \sum_{i}^{S} \left( \frac{f_i - \tilde{f}_i}{\sigma} \right)^2 \tag{13}$$

where $S$ is the number of sensors in the system and $\sigma$ is the standart deviation of the signal, which relates to noise level in the system. According to minimum of $\chi^2$ a corresponding subset of $\vec{a}$ is selected for the reconstruction. The selected subset $\vec{a}^{*}$ of $\vec{a}$ is then multiplied with so-called basis matrix $\mathbf{B}$ , which results in a reconstruction image $\vec{g}$. Image $\vec{g}$ then is copied back to the host.

$$\vec{g}_{[100,100]} = \mathbf{B}_{[100,100,6]} \cdot \vec{a}^{*}_{[6]} \tag{14}$$

Contribution matrix $\mathbf{C}$, its pseudo-inverse $\mathbf{C}^{+}$ and basis matrix $\mathbf{B}$ are pre-loaded to GPU before the real-time phase and do not need updating during execution of the algorithm. Indexed squared brackets indicate the sizes and dimensionality of matrices and vectors. Considering execution time of the whole cycle, i.e. input data transfer to GPU memory, computation and transfer of the result from the GPU memory, the main bottleneck is represented by memory transfer. This fact is evident in Fig. 28, where the computation and memory transfer times are shown at different sizes of the output reconstruction image $\vec{g}$.

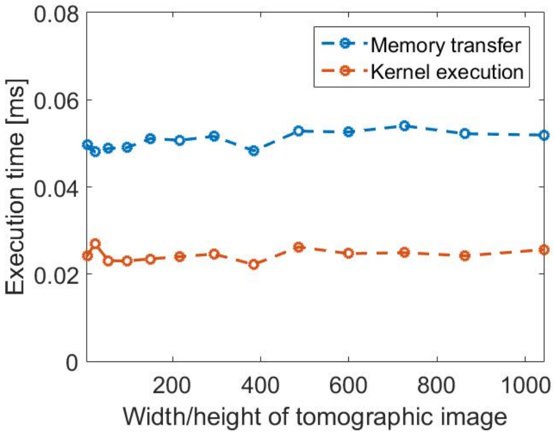


Figure 28: Demonstration of inherent bottleneck in real-time reconstruction algorithm, where overall computation performs faster than 2 memory transfers (host→GPU, GPU→host)

This application is an example of a common use case in real-time control. Input data is collected and transferred to GPU memory, where some kind of parallel algorithm is executed and finally result data copied back from GPU memory to CPU memory to be sent. Data transfer must be performed in both directions at every control cycle, thus affecting system performance. Intuitively, applications involving more complex computation, efficiently carried out by the GPU provided the algorithm can be effectively parallelized, are more likely to perform better than in CPU systems. However defining classes of computation where GPU solutions perform better than CPU ones is not straightforward. For this purpose, a particular computation, that is, matrix-vector multiplication has been considered and its performance compared with single and multithreaded CPU solutions. It turns out that for this kind of computation and for very large matrix and vector dimensions, GPU performance is comparable to multicore and multithreaded CPU performance. This fact provides a rule of thumb in the evaluation of the most appropriate architectural choice in real-time applications, i.e. application involving more complex computation can be considered possible candidates for GPU implementation, provided the amount of data exchanges to/from the GPU memory is not increased. Moreover, matrix-vector multiplication occurs very often in MIMO (**M**ultiple **I**nputs **M**ultiple **O**utputs) controls, e.g. in state-space based control and therefore represents a good example for performance analysis.
Basically GPGPGU is a fit computational platform for following requirements/options:

- Control period ranging from 1 to 100 ms
- Input and output vectors of small size, i.e. up to few thousands of input samples. The input signals are typically read from physical sensors whose number typically range from some tens (e.g. the electromagnetic probes in a fusion device) to some thousands (e.g. thermocouples in a large experimental device). Output signals are sent to actuators and are likely to be in a smaller number than the input sensors
- A possibly large set of read-only parameters that are transferred once in GPU memory (e.g. matrices used in the computation of the state space)
- A set of data maintained in the GPU memory from cycle to cycle, computed at cycle $N$ and used at cycle $N + 1$

Another class of real-time applications is the elaboration of image frames acquired by camera devices. Image based diagnostics are becoming more and more common in many fusion experiments, where they are used to complement the physical parameters derived from other diagnostics. The use case can be described as follows:

- Frame acquisition rate ranging from 5 Hz to 50 Hz
- Data input size ranging from 0.5 to 10 MB, depending on camera resolution and pixel depth. Larger input sizes occur in case GPU computation involves frames from more than one camera
- A set of read-only parameters that are transferred once in GPU memory and that are used for online analysis. The dimension of this set may be very large (e.g. neural network parameters used in feature detection)
- Depending on the application, a possibly large set of data computed at cycle N and used at cycle N+1 (in case image elaboration relies also on the past history)
- Wide range in the data output size depending on the nature of the computation (feature extraction or image elaboration)

In the following, two sample applications mimicking the above use cases are presented, i.e. 1) dense matrix-vector multiplication and 2) Sobel image filtering. Rather than in absolute performance numbers, that heavily depend on the memory access patterns specific to the given computation, we are interested in the evaluation of the jitter in overall computation time using different strategies for data transfer to and from GPU memory.

## 9.4 Performance analysis and comparison

### 9.4.1 Matrix-Vector multiplication

Matrix-vector multiplication in the following form:

$$\begin{bmatrix} A_{1,1} & \cdots & A_{1,n} \\ \vdots & \ddots & \vdots \\ A_{m,1} & \cdots & A_{m,n} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ \vdots \\ b_n \end{bmatrix} \tag{15}$$

can be easily parallelized by considering that every row-column multiplication, involving $n^2$ sums and multiplications can be performed in parallel for every element of the output vector. Fig. 29a shows the distribution of the execution time assuming an input and output vector of $10^4$ float components and without considering any optimization in memory usage, i.e. letting all data reside in Global memory. CUDA events have been used to register the execution duration. The first optimization using Texture memory to keep the values of matrix $A$ leads to a dramatic reduction in overall time and, more importantly, of its jitter that is smaller than 50 [$\mu$s] (Fig. 29b). A further, slight, improvement (but not in jitter) is achieved by pre-loading the input vector $\mathbf{x}$ from Global memory into Shared memory as it is accessed multiple times during row-column multiplication. Therefore the appropriate transfer method represents an important factor in overall performance. The other graphs in Fig. 29 refer to different configurations for memory buffer allocation.

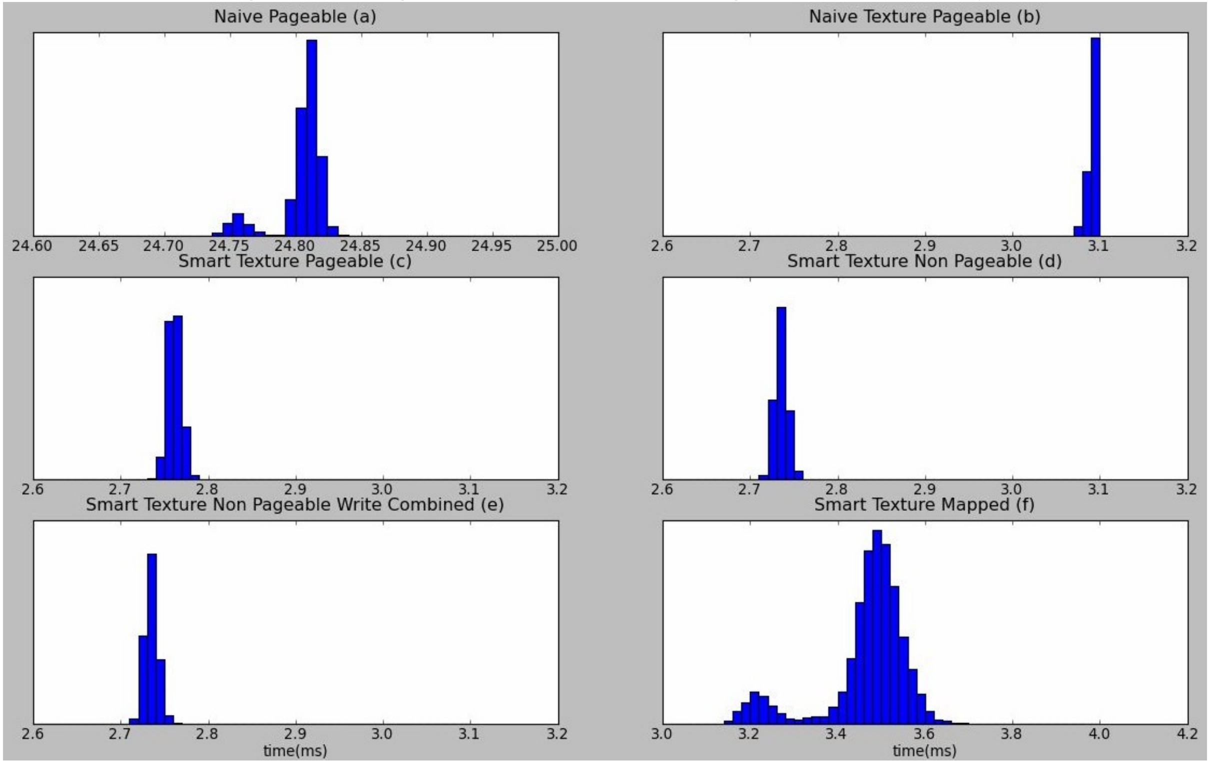


Figure 29: Matrix-vector multiplication latency including memory transfer:
(a) naïve implementation using Global memory
(b) same implementation, but using Texture memory for matrix A
(c) using Texture memory for matrix A and Shared memory for input vector
(d) same as (c), but using non-pageable memory buffer
(e) same as (d) but using write combined memory for input buffer
(f) same algorithm as (c) but using directly mapped memory buffers

Fig. 29c refers to the default usage of pageable allocation for memory transfer to/from the GPU. From Fig. 29cde it can be seen that no evident improvement in using page-locked and write-combined, probably due to the small amount of transferred data. Conversely, direct mapping shown in Fig. 29f performs the worst both in average time and jitter and thus should not be used for real-time applications. No difference in performance has been observed using multiple streamed memory transfers for input and outputs in the same control cycle. In general it is impossible to stream computation for a given control cycle, so that it overlaps with input data transfer for the same cycle (for example in this application the whole input vector must be in GPU memory before computation). However if it was possible to allow a latency larger that the system period, then one could overlap computation for cycle $N$ with data readout for cycle $N+1$ in a pipeline organization, making a more efficient usage of memory and computation resources. Depending on the nature of the computation it may be possible to stream output memory transfer so that it can be performed in parallel with GPU computation.

### 9.4.2 Comparison with CPU architectures

As already discussed, GPU-based real-time applications usually have an inherent memory bottleneck. Anyway GPU kernels alone offer great processing speeds because of their massively parallel nature. Memory bottlenecks must also be considered in CPUs (and sometimes they really limit the computing power of the processor), but their negative effects, often limited by the system caches, are in general less dramatic than in GPUs. The performance of the same matrix-vector multiplication using single-threaded and multiple-threaded (using 6 cores) computation is reported in Fig. 30a and b respectively. For the considered vector size, the GPU performance outperforms the multi-threaded CPU both in average time and jitter. The performance turns out to be comparable in average time and jitter for an input vector of around 400 components, thus providing a rule of thumb indication on the applicability of GPU solutions depending on the vector size for this class of applications.

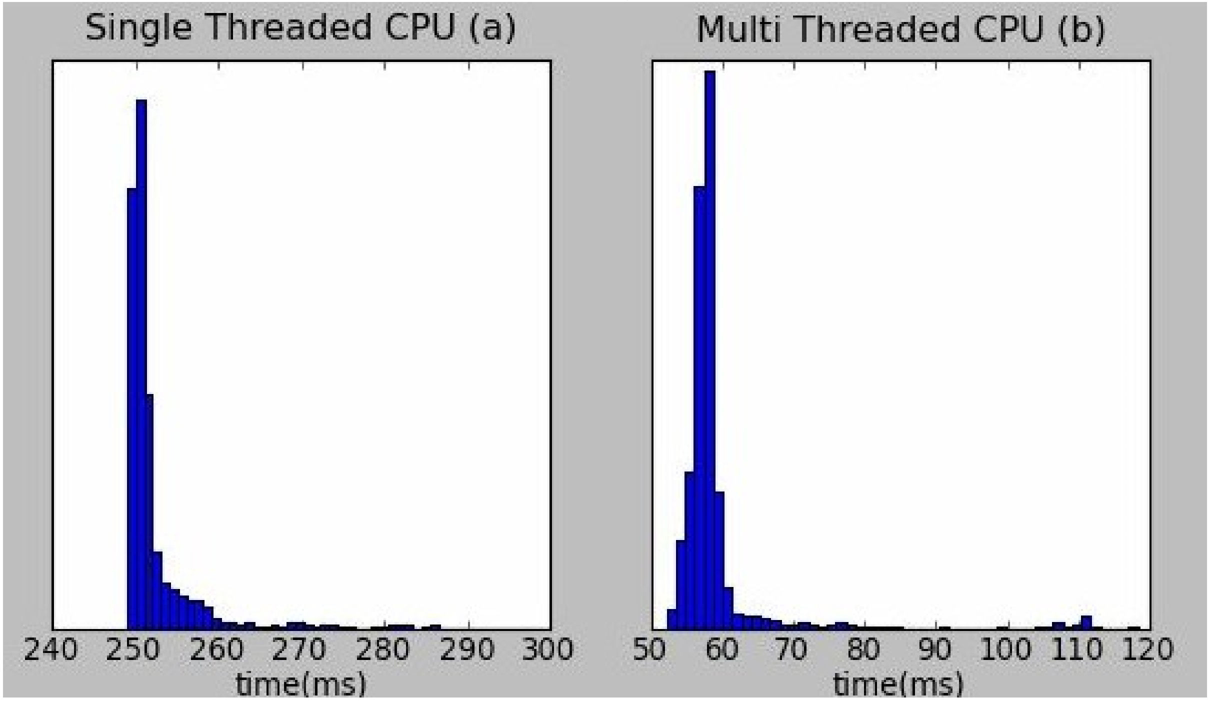


Figure 30: Matrix-vector multiplication duration on CPU:
(a) single-threaded
(b) multi-threaded

A server with Intel Xeon E5-2637 v2 was used to perform CPU tests. It is a processor with 8 physical cores and 5 Mb L2 cache running at base frequency of 3 GHz.

### 9.4.3 Sobel image filtering

The second application performs *Sobel filtering* for the computation of $X$ and $Y$ gradients in edge detection algorithms. The performance measures refer to a 1000 × 1000 image with a pixel depth of 16 bits and are shown in Fig. 31. Unlike the previous application, involving a much smaller data exchange, the advantages in using non-pageable memory buffer is evident (Fig. 31a vs Fig. 31bc). Also in this application, direct mapping performs much worse both in average time and jitter.

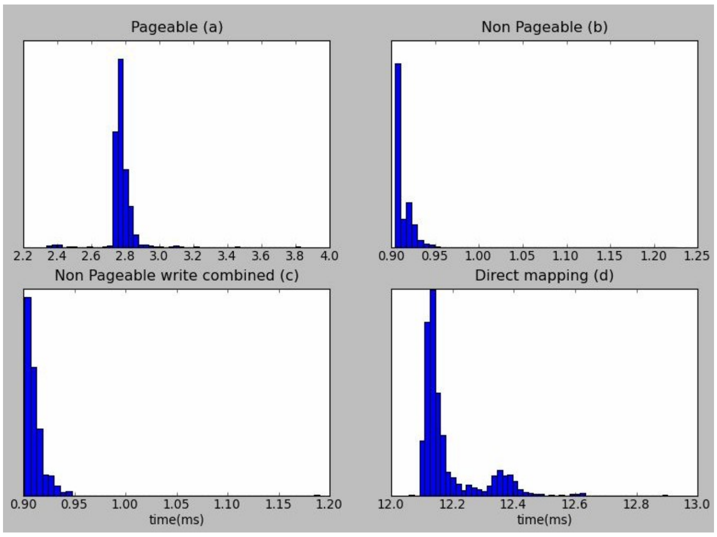


Figure 31: *Sobel filtering* latency including memory transfer:
(a) pageable buffers
(b) non-pageable buffers
(c) non-pageable, write-combined buffers
(d) direct memory mapping

*Sobel filtering* carries out a localized access pattern, because, for every pixel, only the neighbor pixels are considered in gradient computation. It is a well-known fact that GPU performance can dramatically change for different memory access patterns. Fig. 32 shows the measured performance, when *Sobel filtering* is replaced by an algorithm spanning the corresponding image column for every pixel. Unsurprisingly, the average computation and transfer time is much larger, but, interestingly, the jitter has not changed. The measured jitter is much less than 1 ms, therefore making GPUs a very promising candidate for camera-based diagnostics.

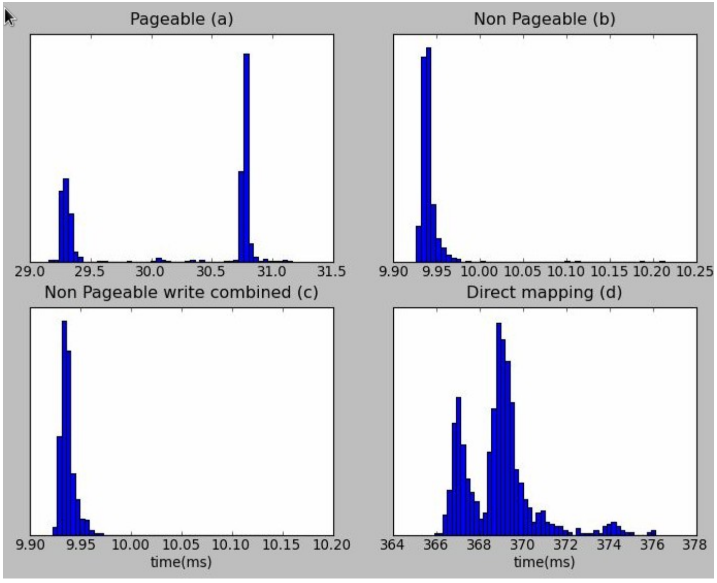


Figure 32: Same parameters of Fig. 31 using an algorithm, that
spans the corresponding image column for every pixel

It is worth noting that GPU performance in real-time applications can be definitely improved by transferring data directly from input devices into CPU memory without involving CPU memory. For example, in NVIDIA devices, GPU Direct RDMA has been introduced since Kepler class GPUs and CUDA 5.0. Using standard features of PCI Express and under some configuration limitations, it is possible to develop kernel modules that perform direct transfer from device to GPU memory. For example, GPU Direct RDMA [33] has been used in [34] to acquire camera image into GPU memory for fast image processing.

## 9.5 Conclusions

This study has presented some insights and discussions regarding nature of CPU and GPU applications in real-time. The results can be summarized as follows:

- Memory transfer between CPU and GPU memory represents the main bottleneck in real-time applications.
- The width/height of a tomographic image was used as a parameter to describe latencies of GPU processing and memory transfers. Approximately constant latencies were observed across the range of different sizes of images even though the amount of data (float) grows quadratically. An image of size 1024x1024 of float data is 4 MB. This suggests that DMA overhead dominates at operating transfers of couple MB in size and smaller. This complies with the theoretical bandwidth of 288 GB/s of K40. There was no practical reason to process bigger images, because higher resolution does not contribute to the quality of tomographic features.
- As a rule of thumb, single-threaded applications should never be developed, but instead the focus must always be on multi-threaded implementations.
- As a rule of thumb, given the time restriction of 100 $\mu$s (the common time scale of real-time control in fusion), it is recommended to use multi-threaed CPUs for input vectors up to size of 400 floats in case of matrix-vector multiplication algorithm. For bigger input vectors GPU is a preferred choice.
- Latency caused by kernel scheduling does not depend on the kernel code itself, only on number of GPU threads/blocks required. Therefore launching the same kernel multiple times or launching different kernels with the same number of threads/blocks makes no difference in latency (at least the scheduling latency). The same is valid for GPU kernels in MARTe context (GAM).
- Due to the fact that real-time CPU applications are oftenly run by specially designated and configured RTTs (**R**eal **T**ime **T**hread) certainly CPU applications experience less jitter, while GPUs are peripheral devices, which are subject to system management processes such as kernel scheduling, instruction/data transfers, memory caching, which cause device behaviour and hardware responses to be less deterministic (more jitter). GPU jitter is inherent and under normal system configuration it is always present and it is always higher than CPU-thread jitter. Therefore it is recommended to use GPU when expected jitter is much lower than expected processing latency.
- Expected lattency and jitter in real-time GPU applications can be definitely improved by using RDMA technology.
- In cases of complex algorithms with many functional parts a mix of CPU and GPU processing can be the best approach. Multicore CPU processing is however limited within MARTe context, because each CPU RTT thread operates another instance of a GAM. Multicore CPU processing within a single GAM is not an option (to my experience), which makes GPU as a viable option for parallel processing in GAMs.

# 10 Summary

Computation advances of the last half-century have depended critically on the rapid growth of single-processor performance at ever-decreasing cost and with manageable increases in power consumption. That growth stemmed from increasing the number and speed of transistors on a processor chip by reducing their size and (with improvements in memory, storage, and networking capacities) resulted in ever more capable computer systems. It was important to support *Moore's law* with new hardware to maintain the sequential stored-program model that was developed for computers in the 1940s. Once power efficiency became a real problem for the hardware the first multicore processors started appearing in 00's. Yet *sequential programming model* is still dominating over *parallel programming model* in general public and major part of scientific community. It seems that the hardware changed faster than the habits of consumers and programmers. Anyway the end of dramatic exponential growth in single-processor performance marks the end of its dominance in computing world. The era of sequential computing is giving way to a new era of parallel computing, where parallelism is at its core. Multicore CPUs are the natural succession for the need of general parallel computing and GPGPUs for data-parallel computing. GPGPUs are particularly well-suited to address problems that can be expressed as data-parallel computations with high arithmetic intensity (high ratio of arithmetic operations to memory operations), i.e. where SIMD computation model can be exploited. Since the same instruction is executed for all data elements, there is a lower requirement for sophisticated flow control (unlike CPUs). And since it is executed on many data elements and has high arithmetic intensity, the memory access latency can be hidden with calculations instead of big data caches. Equiped with powerful modern APIs (CUDA, OpenCL, OpenMP) GPGPUs became invaluable tools in every aspect of computer aided progress. Many applications that process large data sets can use a data-parallel programming model to speed up the computations. In 3D rendering large sets of pixels and vertices are mapped to parallel threads. Similarly, image and media processing applications, such as post-processing of rendered images, video encoding and decoding, image scaling, stereo vision and pattern recognition can map image blocks and pixels to parallel processing threads. But most important that now parallel processing is paving its way to variety of sciences, such as general signal processing, physics simulations, computational finance and computational biology. Recent technology predictions speculate, that in order to completely satisfy data-parallel and task-parallel capability demand on hardware a new cross-breed architecture between GPUs and CPUs should emerge.

# Part III
# Plasma simulations using PIC method

## 11 Plasma basics

All baryonic (ordinary) matter in the universe can be found in one of 4 aggregate states: solid, liquid, gas and plasma. All states are defined by the level of bonding that exists in the substance. In solids particles are closely packed and form strong bonds, that manifest in a rigid/crystaline substance of fixed volume and shape. In liquids particles are less densely packed and form weaker bonds, that allow substance to flow, be mobile and "shapeless" by conserving volume. In gases particles form virtually no bonds, they experience Brownian walk and do not conserve volume. In plasma bonds within individual atoms are broken, i.e. nuclei and electrons are disassociated and free to move independently. Due to that plasmas are electrically conductive, respond strongly to electromagnetic fields and can produce magnetic fields and electric currents. Yet charge quasi-neutrality condition must hold and it is a defining feature of plasma state, otherwise it is classified as ionized gas instead. Plasma is called the 4th state of matter and comprises about 99.999% of all matter in the observable universe (excluding dark matter) according to current scientific consensus.

The plasma modelling is complicated due to fact that the charged particles both produce and are subject to electromagnetic fields at the same time. Interparticle collisions must also be taken into account, which introduces another level of complexity. The inelastic collisions are generation and loss of particles, which allow species to exchange energy, and can be interpreted as an average friction force between different particle species. The overall system is difficult to analyze and posses nonlinear nature. To make progress with such a complicated system, various simplifying approximations are needed. The interparticle collisions are considered independently, since they occur on space and time scales that are usually much shorter than those of the effective fields. Collisions contribute to the equilibrium velocity distribution of the species. The velocity distribution is averaged over velocities to obtain the macroscopic motion. The overall macroscopic motion is a compound effect of internal and external fields acting on plasma. Self-generated internal fields are usually nonlinear and may be linearized only in particular phenomena, such as waves in plasmas. In order to model plasmas in fuller and more complete manner simulations of various kinds are designed. Two main groups of simulations exist: kinetic and fluid, which respectively stand as Lagrangian and Eulerian approaches to solving Boltzmann equation. Kinetic simulations deal with plasma as collection of particles and employ techniques, such as gyrokinetic theory, Fokker-Planck theory and PIC method. Fluid simulatations deal with plasma as a fluid and employ techniques, such as fluid dynamics, MHD (**M**agneto**h**ydro**d**ynamics) and gyrofluid theory. The choice of the "right" simulation highly depends on physics and phenomena of interest. In order to simulate plasmas in highly complex experiments such as tokamaks, integrated modelling is the way. There can be many different simulations involved to model plasma in various tokamak regions (core, edge, SOL (**S**crape-**O**ff **L**ayer) and divertor) and during various phases of tokamak discharge (start-up, sustainment, unexpected disturbance, shut down). Indeed any simulation must represent reality and "be physical" as much as possible in order to be scientifically useful and valuable. Basic plasma description provides plasma parameters, which are universal and unequivocal across theories and methods and are first-hand "reality/physics check" references.

### 11.1 Poisson equation

Plasma is a fluid/collection of charged particles fulfilling a quasi-neutrality condition and displaying collective behaviour. Plasma's electromagnetic nature is governed by Maxwell's equations (Eq. 16):

$$\left\{\begin{array}{rlcll} \varepsilon_0\nabla\cdot & \mathbf{E}(\mathbf{r},t) & = & \rho(\mathbf{r},t) & \text{Gauss' law} \\ \mu_0\nabla\cdot & \mathbf{H}(\mathbf{r},t) & = & 0 & \text{Gauss' law for magnetism} \\ \nabla\times & \mathbf{E}(\mathbf{r},t) & = & -\mu_0\frac{\partial\mathbf{H}(\mathbf{r},t)}{\partial t} & \text{Faraday's law} \\ \nabla\times & \mathbf{H}(\mathbf{r},t) & = & \varepsilon_0\frac{\partial\mathbf{E}(\mathbf{r},t)}{\partial t}+\mathbf{J} & \text{Ampere's law} \end{array}\right. \tag{16}$$

where $\mathbf{E}(\mathbf{r},t)$ and $\mathbf{H}(\mathbf{r},t)$ are the electric and magnetic field vectors, $\varepsilon_0 \approx 8.854 \times 10^{-12}$ [F/m] is the vacuum permittivity and $\mu_0 = 4\pi \times 10^{-7}$ [H/m] is the vacuum permeability. Maxwell set of equations can be reduced to a *electrostatic* approximation by making $\nabla \times \mathbf{E} = 0$ (Faraday's law). The *electrostatic* approximation does not take into account magnetic field evolution. Often this is enough to describe most of the plasmas, except where magnetic fields are strong and with a non-negligible time variation, e.g. tokamak plasmas. Since the curl of a gradient is zero, this implies that the electric field can be derived from the gradient of a scalar potential:

$$\mathbf{E}(\mathbf{r}, t) = -\nabla\phi(\mathbf{r}, t) \tag{17}$$

Combined with Gauss' law (Eq. 16) one obtains Poisson's equation:

$$\nabla^2\phi(\mathbf{r}, t) = -\frac{\rho(\mathbf{r}, t)}{\varepsilon_0} \tag{18}$$

Poisson's equation constitutes the fundamental basis for the further developments and simulations of this thesis.

## 11.2 Plasma frequency

Plasma is able to oscillate at certain frequency called plasma frequency. Plasma frequency is a characteristic parameter of plasma and can be easily derived. Consider 1D plasma slab, where one of the charged species is displaced from its quasi-neutral position by an infinitesimal distance. The charge density develops on the leading face of the slab, which in turn generates the electric field in the $x$ direction:

$$E = \frac{\sigma}{\varepsilon_0} = \frac{en_\mathrm{e}x}{\varepsilon_0}, \tag{19}$$

where $n_e$ is electron density, $x$ is charge displacement and $e \approx 1.60217662 \times 10^{-19}$ $[C]$ is the elementary charge. The force acting on electrons is $F_e = -eE$. Using Newton's second law one gets:

$$m_\mathrm{e}\frac{\partial^2 x}{\partial t^2} = -\frac{e^2 n_\mathrm{e} x}{\varepsilon_0} \quad \longrightarrow \quad \frac{\partial^2 x}{\partial t^2} + \frac{e^2 n_\mathrm{e}}{m_\mathrm{e}\varepsilon_0}x = 0, \tag{20}$$

where $m_\mathrm{e} \approx 9.10938356 \times 10^{-31}$ $[kg]$ is electron mass, $\omega_\mathrm{e}^2 \equiv \frac{e^2 n_\mathrm{e}}{m_\mathrm{e}\varepsilon_0}$ is called *plasma frequency.* Plasma frequency gives a clear indication for simulation integration time $\Delta t$, which has to be smaller than $\omega_\mathrm{e}^{-1}$. Otherwise there is no consistent interplay of particles and electric fields and simulation does not produce plasma behaviour. In such case particles usually exit the system before completing one plasma oscillation.

## 11.3 Debye length

Debye length (named after the dutch-american physical chemist Peter Debye (1884 – 1966)) is a characteristic parameter of plasma and is used to understand the spatial scale of validity for quasi-neutrality condition. Analogically to plasma frequency, which defines chearacteristic temporal scale, Debye length defines characteristic spatial scale of plasma. Derrivation starts with a Poisson' equation for a negative point charge in 1D:

$$\frac{\partial^2\phi(x)}{\partial x^2} = -\frac{e(n_\mathrm{i}(x) - n_\mathrm{e}(x))}{\varepsilon_0} \tag{21}$$

The plasma is quasi-neutral at infinity $n_\mathrm{i} = n_\mathrm{e} = n_\infty$ and it is in thermal equilibrium, i.e. electron density follows a Maxwell-Boltzmann law $n_\mathrm{e}(x) = n_\infty e^{\frac{e\phi(x)}{k_\mathrm{B}T_\mathrm{e}}}$. Then Poisson's equation (Eq. 21) becomes:

$$\frac{\partial^2\phi(x)}{\partial x^2} = \frac{en_\infty\left(e^{\frac{e\phi(x)}{k_\mathrm{B}T_\mathrm{e}}} - 1\right)}{\varepsilon_0} \tag{22}$$

Using Taylor expansion $[e^x = 1 + x^2 + \frac{1}{2}x^2 + \ldots]$ around $\phi(x) = 0$:

$$\frac{\partial^2\phi(x)}{\partial x^2} = \frac{en_\infty\left(1 + \frac{e\phi(x)}{k_\mathrm{B}T_\mathrm{e}} + \frac{1}{2}\left(\frac{e\phi(x)}{k_\mathrm{B}T_\mathrm{e}}\right)^2 + \ldots - 1\right)}{\varepsilon_0} \tag{23}$$

One assumes a "warm" plasma, where the particle energy of thermal movement is much higher than the energy of electrostatic evolution, i.e. $\frac{e\phi(x)}{k_\mathrm{B}T_\mathrm{e}} \ll 1$. Then Eq. 23 is approximated:

$$\frac{\partial^2\phi(x)}{\partial x^2} \approx \frac{n_\infty e^2}{\varepsilon_0 k_\mathrm{B} T_\mathrm{e}}\phi(x) = \frac{\phi(x)}{\lambda_\mathrm{D}^2} \tag{24}$$

$$\phi(x) = \phi_0 e^{-\frac{|x|}{\lambda_\mathrm{D}}} \tag{25}$$

where $\lambda_\mathrm{D} \equiv \sqrt{\frac{\varepsilon_0 k_\mathrm{B} T_\mathrm{e}}{n_\infty e^2}}$ is called *Debye length.* Debye length is the scale, over which electrons screen out electric fields in plasmas, i.e. *Debye shielding.* In other words, the Debye length is the distance, within which quasi-neutrality is broken down. A *Debye sphere* is a volume, whose radius is the Debye length. Debye shielding is a purely electrostatic effect, because it does not depend on species mass, but only temperature and density. In order to properly simulate plasma, spatial resolution must respect Debye length, so that $\Delta x < \lambda_\mathrm{D}$.

## 11.4 CFL condition

Plasma parameters such as Debye length $\lambda_\mathrm{D}$ and plasma frequency $\omega_\mathrm{p}$ basically determine what kind of plasma one is simulating and what $\Delta x$ and $\Delta t$ has to be chosen as simulation parameters in order to obtain physical results. These two parameters alone determine, whether simulation is going to produce physical behaviour or not. However another validity condition exists that relates $\Delta x$ and $\Delta t$ in single statement. The relation between $\Delta t$ and $\Delta x$ is stated as CFL (**C**ourant-**F**riedrichs-**L**ewy) condition [35]:

$$\Delta t \sum_{i=1}^{n} \frac{v_i}{\Delta x_i} \overset{[n=3]}{=} \frac{3\Delta t v}{\Delta x} < 1, \tag{26}$$

where $n$ is the dimensionality of the problem, $v$ is the particle velocity against which CFL is computed (usually set to value of thermal velocity $v_\mathrm{th}$). CFL condition comes as a general stability requirement for finite-difference numerical solutions of partial differential equations. In the simulations of this thesis one does NOT strictly follow CFL condition, but rather explore independent parameter configurations for $\Delta t$ and $\Delta x$ together with other paramters such as solver tolerance and number of particles per cell. In simple language CFL does not allow particles to move across more than one cell per time step.

## 11.5 Collisionless sheath

At the edge of a bounded plasma experiments demonstrate a layer of violated quasi-neutrality, where ions density exceeds electron density. This layer is called sheath. The cause of sheath formation lies in the fact, that ions and electrons have very different mobilities, which are direct effect of the mass difference between ions and electrons. Due to the density difference, there is a build up of positive charge density, which results in generation of electric fields next to the surface and domain boundaries (Fig. 33). Those developed fields balance the fluxes of electrons and ions reaching the wall, i.e. decelerates electrons and accelerates ions until equilibrium is reached. In such a way the bulk plasma is shielding itself from edge plasma by building up plasma potential $\phi_\mathrm{p}$ (Fig. 35). In fact the sheath is a natural response of plasma due to surfaces and domain boundaries. The thickness of such a layer is several Debye lengths $\lambda_\mathrm{D}$ thick, which depends on plasma characteristics such as temperature and density. One assumes electrons and ions of bulk densities $n_\mathrm{e}$ and $n_\mathrm{i}$ and temperatures $T_\mathrm{e}$ and $T_\mathrm{i}$ respectively. The coordinate origin is set at the plasma-sheath interface point (interface between quasi-neutrality and non-neutrality regions) (Fig. 33). At the coordinate orgin $n_\mathrm{e}(x)|_{x=0} = n_\mathrm{i}(x)|_{x=0} = n_\mathrm{s}$, $\phi(x)|_{x=0} = 0$ and ions have speed $u(x)|_{x=0} = u_\mathrm{s}$. Ion energy conservation is given by:

$$\frac{1}{2}m_\mathrm{i}u^2(x) = \frac{1}{2}m_\mathrm{i}u_\mathrm{s}^2 - e\phi(x), \tag{27}$$

where $m_\mathrm{i}$ is ion mass. Ion momentum conservation is given by:

$$n_\mathrm{i}(x)u(x) = n_\mathrm{s}u_\mathrm{s} \tag{28}$$

By combining Eq. 27 and Eq. 28 one expresses $n_\mathrm{i}(x)$:

$$n_\mathrm{i}(x) = \frac{n_\mathrm{s} u_\mathrm{s}}{u(x)} = \frac{n_\mathrm{s} u_\mathrm{s}}{\sqrt{u_\mathrm{s}^2 - \frac{2e\phi(x)}{m_\mathrm{i}}}} = \frac{n_\mathrm{s}}{\sqrt{1 - \frac{2e\phi(x)}{m_\mathrm{i} u_\mathrm{s}^2}}} \tag{29}$$

One can construct the RHS of Poisson equation (Eq. 21) by having ion density $n_\mathrm{i}(x)$ (Eq. 29) and Maxwell-Boltzmann electron density $n_\mathrm{e}(x) = n_\mathrm{s} e^{\frac{e\phi(x)}{k_\mathrm{B} T_\mathrm{e}}}$:

$$\frac{\partial^2 \phi(x)}{\partial x^2} = \frac{e n_\mathrm{s}}{\varepsilon_0} \left( e^{\frac{e\phi(x)}{k_\mathrm{B} T_\mathrm{e}}} - \frac{1}{\sqrt{1 - \frac{2e\phi(x)}{m_\mathrm{i} u_\mathrm{s}^s}}} \right) \tag{30}$$

Unlike equation 25 the equation 30 is nonlinear, it is not truncated at higher orders. In fact Eq. 30 allows to predict *presheath*. Presheath is a region of concerved quasi-neutrality but smaller density than bulk plasma (Fig. 33). Presheath is the region, where ion-acoustic instabilities develop.

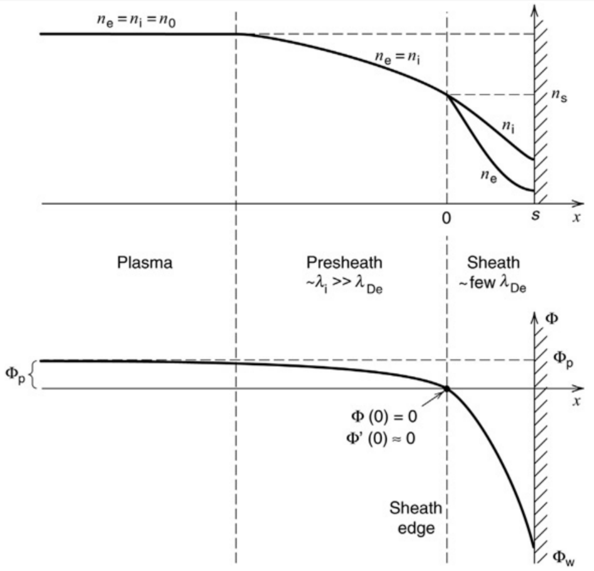


Figure 33: Density (top) and potential (bottom) profiles in case of sheath formation

According to [36] the Debye sheath depth is an approximation:

$$d = 1.018 \lambda_\mathrm{D} \left( \frac{\phi_\mathrm{p}}{T_\mathrm{e}} \right)^{\frac{3}{4}} \tag{31}$$

## 11.6 Plasma potential

Plasma potential $\phi_\mathrm{p}$ is primary effect of sheaths in plasma and one of the main characteristics of equilibrated plasmas. In a collisionless sheath description (Fig. 30) the $\phi_\mathrm{p}$ is a combined effect of *sheath* and *presheath* presence, therefore one can write:

$$\phi_\mathrm{p} = \phi_\mathrm{presheath} + \phi_\mathrm{sheath} \tag{32}$$

According to [36] potential drop in the presheath:

$$\phi_\mathrm{presheath} = \frac{T_\mathrm{e}[\mathrm{eV}]}{2} \tag{33}$$

According to [36] potential drop in the sheath:

$$\phi_\mathrm{sheath} = \frac{T_\mathrm{e}[\mathrm{eV}]}{2} \ln \frac{m_\mathrm{i}}{2\pi m_\mathrm{e}} \tag{34}$$

Eq. 32 becomes:

$$\phi_\mathrm{p} = \frac{T_\mathrm{e}[\mathrm{eV}]}{2} \left( 1 + \ln \frac{m_\mathrm{i}}{2\pi m_\mathrm{e}} \right) \tag{35}$$

## 11.7 Thermostat theory

Collisionless PIC codes are known to produce good results during transient state of plasma, i.e. it procudes correct $\phi_\mathrm{p}$ for a unequilibrated plasmas for very short time scales. However in order to obtain complete *sheath* and *presheath* formation and obtain correct $\phi_\mathrm{p}$ one has to simulate much longer time evolutions, i.e. steady state is required. However simply extending *transient state* simulation for longer duration leads to incorrect results. The fact that that ions and electrons have such different mobilities makes electron escape domain much faster than ions, which drops the electron temperature $T_\mathrm{e}$ and drains the energy from the system. This effect is called *electron cooling* and in a long run severely alters simulation output. In fact *electron cooling* is an artificial effect of the simulation, which occurs due to the fact that simulation has NO collisions. In order to compensate for electron cooling one has to inject energy into the system. Numerically it is performed by regenerating lost electrons and rescaling the velocities to maintain required temperature in the system. According to kinetic theory the statistical distribution of a velocity vector $\vec{v}$:

$$f(\vec{v}) = f(v_x, v_y, v_z) = f(v_x)f(v_y)f(v_z), \tag{36}$$

where each of the directional components follow Gaussian distributions:

$$f(v_{x,y,z}) = \sqrt{\frac{m}{2\pi k_\mathrm{B} T_\mathrm{e}}} e^{-\frac{m_\mathrm{e} v_{x,y,z}^2}{2k_\mathrm{B}T_\mathrm{e}}}, \tag{37}$$

which makes Eq. 36 into:

$$f(\vec{v}) = \sqrt{\left(\frac{m}{2\pi k_\mathrm{B} T_\mathrm{e}}\right)^3} e^{-\frac{m_\mathrm{e}\left(v_x^2+v_y^2+v_z^2\right)}{2k_\mathrm{B}T_\mathrm{e}}} = \sqrt{\left(\frac{m}{2\pi k_\mathrm{B} T_\mathrm{e}}\right)^3} e^{-\frac{m_\mathrm{e}\vec{v}^2}{2k_\mathrm{B}T_\mathrm{e}}} \tag{38}$$

The modulus of the velocity vector $|\vec{v}|$ follows Maxwell-Boltzmann distribution:

$$f(|\vec{v}|) = \sqrt{\left(\frac{m_\mathrm{e}}{2\pi k_\mathrm{B} T_\mathrm{e}}\right)^3} 4\pi|\vec{v}|^2 e^{-\frac{m_\mathrm{e}|\vec{v}|^2}{2k_\mathrm{B}T_\mathrm{e}}} \tag{39}$$

Electron regeneration follows distributions Eq. 38 and Eq. 39 (which carry the same meaning, just are different representations of velocity space). In a long run only slow (less energetic) electrons remain in the system, because fast (more energetic) electrons manage to escape the domain, which leads to *electron cooling*. The temperature in the system is calculated as:

$$T = \frac{T_x + T_y + T_z}{3}, \text{ where } T_{x,y,z} = \frac{m_\mathrm{e}}{k_\mathrm{B}}\left(\left\langle v_{x,y,z}^2\right\rangle - \left\langle v_{x,y,z}\right\rangle^2\right), \tag{40}$$

where $\left\langle v_{x,y,z}^2\right\rangle = \frac{v_{x,y,z}^2}{N}$ and $\left\langle v_{x,y,z}\right\rangle^2 = \frac{v_{x,y,z}^2}{N^2}$, where $N$ is number of paticles in the scope. Eq. 40 becomes:

$$T = \frac{m_\mathrm{e}}{3k_\mathrm{B}}\left(\left\langle v_x^2\right\rangle - \left\langle v_x\right\rangle^2 + \left\langle v_y^2\right\rangle - \left\langle v_y\right\rangle^2 + \left\langle v_z^2\right\rangle - \left\langle v_z\right\rangle^2\right) = \frac{m_\mathrm{e}}{3k_\mathrm{B}}\left(\left\langle \vec{v}^2\right\rangle - \left\langle \vec{v}\right\rangle^2\right) = \frac{2E_\mathrm{k}}{3k_\mathrm{B}} \tag{41}$$

In order to compensate for the lost energy, a virtual *thermostat* has to be introduced into system. A thermostat is a process/algorithm, which rescales velocities for a fraction of particles, i.e. particles are compensated for the lost velocities (energy). The simplest form of a thermostat is a **Berendsen thermostat**. The Berendsen thermostat [37] is an algorithm to rescale the velocities of particles in simulations to control the simulation temperature. It is defined as a first-order differential equation:

$$\frac{\mathrm{d}T}{\mathrm{d}t} = \frac{T_0 - T}{\tau} \tag{42}$$

Solution:

$$T(t) = T_0 e^{-\frac{t}{\tau}}, \tag{43}$$

where $t$ is time, $T$ is temperature, $T_0$ is initial system temperature, $\tau$ is the time constant of the thermostat. Floating plasma potential $\phi_\mathrm{p}$ is proportional to temperature of the system $\phi_\mathrm{p}(t) \sim T(t)$.

Ratio $\frac{t}{\tau}$ is proportional to the portion of particles $\frac{N_{\Delta t}}{N_{\text{total}}}$, whose velocities are rescaled each time step $\Delta t$. By rescaling velocities for a portion of particles one can simulate thermostat process (heat bath with a constant temperature coupled to the plasma):

$$\phi_{\text{p}} \sim \phi_{\text{p}0} e^{-\frac{N_{\Delta t}}{N_{\text{total}}}}, \tag{44}$$

where $\frac{N_{\Delta t}}{N_{\text{total}}}$ is thermostat efficiency.

# 12 Plasma simulation

In 1872 Ludwig Boltzmann devised a fundamental description for evolving thermodynamic systems, which up-to-date stands as a cornerstone in statistical physics. It describes transport of heat, momentum and mass, allows to derive fluid equations, magnetohydrodynamic equations, plasma and fluid properties such as viscosity, thermal conductivity and electrical conductivity. Boltzmann equation (Eq. 45) is solved for the particle distribution function $f$, which holds the statistical properties of the system.

$$\underbrace{\left(\frac{\partial f}{\partial t}\right)_{\text{Coll}}}_{\text{collisional term}} = \frac{\partial f}{\partial t} + \underbrace{\mathbf{v} \cdot \frac{\partial f}{\partial \mathbf{x}}}_{\text{diffusion term}} + \underbrace{\frac{\mathbf{F}}{m} \cdot \frac{\partial f}{\partial \mathbf{v}}}_{\text{force term}}, \tag{45}$$

where $f(\mathbf{x}, \mathbf{v}, t)$ is a multidimentional particle distribution function over phase space, $\mathbf{F}$ is the force field that fluid particles are subjected to, $\mathbf{v}$ is the velocity field that fluid particles are participating in, $m$ is the mass of a fluid particle. The equation is composed of three components of different nature: collisional term, diffusion term and force term. Any imbalance in these three terms results in evolution of distribution function $f$. In 1938 Anatoly Vlasov [38] modified Boltzmann equation by removing the collisional term due to the difficulties of describing plasmas with long-range Coulomb interactions. The modified version of Boltzmann equation is reffered as "collisionless Boltzmann equation" or Vlasov equation (Eq. 46).

$$\frac{\mathrm{d}f}{\mathrm{d}t} = \frac{\partial f}{\partial t} + \mathbf{v} \cdot \frac{\partial f}{\partial \mathbf{x}} + \frac{\mathbf{F}}{m} \cdot \frac{\partial f}{\partial \mathbf{v}} = 0, \tag{46}$$

Vlasov equation is a nonlinear integro-differential equation, which can be solved analytically only for simplified problems, but for realistic problems only numerical solutions are possible. Numerical schemes to solve Vlasov equation can be grouped into three groups: Eulerian, Lagrangian and semi-Lagrangian. The main difference of these schemes lie in the treatment of the distribution function $f$. Eulerian schemes follow the change of the distribution function $f$ at a specific point in phase space, Lagrangian schemes see the distribution function $f$ as a collection of discrete bits (particles), which are followed individually along their trajectories in the phase space. The mesh is deforming with the fluid evolution in Eulerian schemes and the mesh is rigid in Lagrangian schemes. Semi-Lagrangian scheme obtain features from both Eulerian and Lagrangian schemes by using a deforming mesh from Eulerian scheme and having discrete equations from the Lagrangian scheme.

# 13 PIC method

PIC (**P**article-**I**n-**C**ell) method is a Lagrangian scheme proposed by R. W. Hockney in 1988 [39] and C. K. Birdsall in 1991 [40, 41]. Throughout years PIC method became very popular in simulating plasmas due to its straightforward and universal implementation and ability to simulate plasma behaviour on microscopic scales. In PIC the plasma is described as a collection of charged particles, which are moved by self-generated and boundary-induced electric fields. In this thesis PIC will be used only to simulate *electrostatic* behaviour. *Electrodynamic* effects (Ampere's law, Faraday's law) are not taken into account. Maxwell's equations are reduced to a single Poisson equation. It is a substantial simplification of physics only valid under assumptions that electromagnetic effects are negligible and plasma density is low. PIC method is comprised of 5 steps:

1. The charge of particles is projected onto the mesh

2. Electrostatic potential field is computed by solving Poisson equation

3. Electrostatic electric field is computed from electrostatic potential field, which is used to interpolate Lorentz force for each particle

4. Moving particles to new locations according to interpolated Lorentz force

5. Processing particle collisions (optional)

## 13.1 Charge projection

Charge density carried by plasma particles has to be distributed to mesh points. There are three schemes to do it: NGP (**N**earest **G**rid **P**oint), CIC (**C**loud **I**n **C**ell), TSC (**T**riangular **S**haped **C**loud). NGP assigns particle charge to the nearest mesh point, CIC linearly interpolates particle position in the mesh cell and distributes charge according to the distance to the nearest mesh points, TSC uses nonlinear interpolation to distribute charge among nearest mesh points. CIC and TSC exist in higher order representation by including additional layers of vicinity mesh points. First order CIC (Eq. 47) is the most usable scheme, because its implementation is simple, accurate and computationaly efficient. CIC inspired the name of PIC method. Therefore it is used for simulation results of this thesis. First order CIC distributes the charge among nearest mesh points according to the relative size of volume segments that particle coordinates divide the cell to (Fig. 34).

$$
\begin{cases}
\rho_{i,j,k} & = q \cdot n_{\text{top north east}} & = q \cdot \frac{V_{\text{top north east}}}{V_{\text{cell}}} & = q \cdot \frac{(x_{i+1}-x)(y_{j+1}-y)(z_{k+1}-z)}{\Delta x \Delta y \Delta z} \\
\rho_{i+1,j,k} & = q \cdot n_{\text{top north west}} & = q \cdot \frac{V_{\text{top north west}}}{V_{\text{cell}}} & = q \cdot \frac{(x-x_i)(y_{j+1}-y)(z_{k+1}-z)}{\Delta x \Delta y \Delta z} \\
\rho_{i,j+1,k} & = q \cdot n_{\text{top south east}} & = q \cdot \frac{V_{\text{top south east}}}{V_{\text{cell}}} & = q \cdot \frac{(x_{i+1}-x)(y-y_j)(z_{k+1}-z)}{\Delta x \Delta y \Delta z} \\
\rho_{i+1,j+1,k} & = q \cdot n_{\text{top south west}} & = q \cdot \frac{V_{\text{top south west}}}{V_{\text{cell}}} & = q \cdot \frac{(x-x_i)(y-y_j)(z_{k+1}-z)}{\Delta x \Delta y \Delta z} \\
\rho_{i,j,k+1} & = q \cdot n_{\text{bottom north east}} & = q \cdot \frac{V_{\text{bottom north east}}}{V_{\text{cell}}} & = q \cdot \frac{(x_{i+1}-x)(y_{j+1}-y)(z-z_k)}{\Delta x \Delta y \Delta z} \\
\rho_{i+1,j,k+1} & = q \cdot n_{\text{bottom north west}} & = q \cdot \frac{V_{\text{bottom north west}}}{V_{\text{cell}}} & = q \cdot \frac{(x-x_i)(y_{j+1}-y)(z-z_k)}{\Delta x \Delta y \Delta z} \\
\rho_{i,j+1,k+1} & = q \cdot n_{\text{bottom south east}} & = q \cdot \frac{V_{\text{bottom south east}}}{V_{\text{cell}}} & = q \cdot \frac{(x_{i+1}-x)(y-y_j)(z-z_k)}{\Delta x \Delta y \Delta z} \\
\rho_{i+1,j+1,k+1} & = q \cdot n_{\text{bottom south west}} & = q \cdot \frac{V_{\text{bottom south west}}}{V_{\text{cell}}} & = q \cdot \frac{(x-x_i)(y-y_j)(z-z_k)}{\Delta x \Delta y \Delta z}
\end{cases}
\tag{47}
$$

where $\{\rho_{i,j,k},\ \rho_{i+1,j,k},\ \rho_{i,j+1,k},\ \rho_{i+1,j+1,k},\ \rho_{i,j,k+1},\ \rho_{i+1,j,k+1},\ \rho_{i,j+1,k+1},\ \rho_{i+1,j+1,k+1}\}$ are the charge density values calculated onto the mesh; $\{n_{\text{top north east}},\ n_{\text{top north west}},\ n_{\text{top south east}},\ n_{\text{top south west}},\ n_{\text{bottom north east}},\ n_{\text{bottom north west}},\ n_{\text{bottom south east}},\ n_{\text{bottom south west}}\}$ are the particle density values; $\{V_{\text{top north east}},\ V_{\text{top north west}},\ V_{\text{top south east}},\ V_{\text{top south west}},\ V_{\text{bottom north east}},\ V_{\text{bottom north west}},\ V_{\text{bottom south east}},\ V_{\text{bottom south west}}\}$ are the cell volume segments that particle coordinates divide the cell to; $V_{\text{cell}}$ is the cell volume; $\{x_i,\ x_{i+1},\ y_j,\ y_{j+1},\ z_k,\ z_{k+1}\}$ are the cell node coordinates; $\{x, y, z\}$ are the particle coordinates; $\{\Delta x, \Delta y, \Delta z\}$ are the cell dimensions; $q$ is the weighted charge of particle (charge of *macro-particle*).

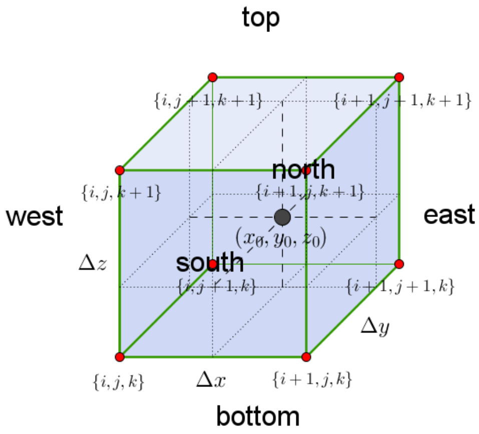


Figure 34: Particle in a cell at position $(x_0, y_0, z_0)$

## 13.2 Poisson equation

Once charge density is distributed over the grid, next step is to solve Poisson equation (Eq. 48) and find the configuration of electrostatic potential field $\phi$.

$$\nabla^2 \phi = -\frac{\rho}{\varepsilon_0}, \tag{48}$$

where $\phi$ is the electrostatic potential field, $\rho$ is the charge density, $\varepsilon_0 = 8.854187 \cdot 10^{-12}$ [F/m] is the vacuum permittivity. Poisson equation is a partial differential equation, which aquires 3D form (Eq. 49):

$$\frac{\partial^2 \phi(x,y,z)}{\partial x^2} + \frac{\partial^2 \phi(x,y,z)}{\partial y^2} + \frac{\partial^2 \phi(x,y,z)}{\partial z^2} = -\frac{\rho(x,y,z)}{\varepsilon_0} \tag{49}$$

Discretized Poisson equation:

$$\frac{\Delta^2 \phi_{i,j,k}}{\Delta x^2} + \frac{\Delta^2 \phi_{i,j,k}}{\Delta y^2} + \frac{\Delta^2 \phi_{i,j,k}}{\Delta z^2} = -\frac{\rho_{i,j,k}}{\varepsilon_0} \tag{50}$$

Applying FDM (**F**inite **D**ifference **M**ethod):

$$\begin{cases} \frac{\Delta^2 \phi_{i,j,k}}{\Delta x^2} = \frac{\Delta\left(\phi_{i+\frac{1}{2},j,k} - \phi_{i-\frac{1}{2},j,k}\right)}{\Delta x^2} = \frac{\phi_{i+1,j,k} - 2\phi_{i,j,k} + \phi_{i-1,j,k}}{\Delta x^2} \\ \frac{\Delta^2 \phi_{i,j,k}}{\Delta y^2} = \frac{\Delta\left(\phi_{i,j+\frac{1}{2},k} - \phi_{i,j-\frac{1}{2},k}\right)}{\Delta y^2} = \frac{\phi_{i,j+1,k} - 2\phi_{i,j,k} + \phi_{i,j-1,k}}{\Delta y^2} \\ \frac{\Delta^2 \phi_{i,j,k}}{\Delta z^2} = \frac{\Delta\left(\phi_{i,j,k+\frac{1}{2}} - \phi_{i,j,k-\frac{1}{2}}\right)}{\Delta z^2} = \frac{\phi_{i,j,k+1} - 2\phi_{i,j,k} + \phi_{i,j,k-1}}{\Delta z^2} \end{cases} \tag{51}$$

Discretized Poisson equation (Eq. 50) becomes:

$$\frac{\phi_{i+1,j,k} - 2\phi_{i,j,k} + \phi_{i-1,j,k}}{\Delta x^2} + \frac{\phi_{i,j+1,k} - 2\phi_{i,j,k} + \phi_{i,j-1,k}}{\Delta y^2} + \frac{\phi_{i,j,k+1} - 2\phi_{i,j,k} + \phi_{i,j,k-1}}{\Delta z^2} = -\frac{\rho_{i,j,k}}{\varepsilon_0} \tag{52}$$

Assuming uniform spatial discretization $\Delta x = \Delta y = \Delta z$ Eq. 52 becomes:

$$\frac{\phi_{i+1,j,k} + \phi_{i-1,j,k} + \phi_{i,j+1,k} + \phi_{i,j-1,k} + \phi_{i,j,k+1} + \phi_{i,j,k-1} - 6\phi_{i,j,k}}{\Delta x^2} = -\frac{\rho_{i,j,k}}{\varepsilon_0} \tag{53}$$

Rearranging:

$$6\phi_{i,j,k} - \phi_{i+1,j,k} - \phi_{i-1,j,k} - \phi_{i,j+1,k} - \phi_{i,j-1,k} - \phi_{i,j,k+1} - \phi_{i,j,k-1} = \frac{\rho_{i,j,k} \cdot \Delta x^2}{\varepsilon_0} := b_{i,j,k} \tag{54}$$

By having index ranges $\begin{cases} 1 \le i \le m \\ 1 \le j \le n \\ 1 \le k \le l \end{cases}$ equation Eq. 54 is expressed as a linear system of size $mnl$:

$$\mathbf{A}_{mnl\times mnl} \cdot \boldsymbol{\phi}_{mnl} = \mathbf{b}_{mnl}, \tag{55}$$

$$\text{where} \qquad \mathbf{A}_{mnl\times mnl} = \begin{bmatrix} \mathbf{B}_{nl\times nl} & -\mathbf{I}_{nl\times nl} & \iddots & 0 \\ -\mathbf{I}_{nl\times nl} & \mathbf{B}_{nl\times nl} & \ddots & 0 \\ \ddots & \ddots & \ddots & -\mathbf{I}_{nl\times nl} \\ 0 & 0 & -\mathbf{I}_{nl\times nl} & \mathbf{B}_{nl\times nl} \end{bmatrix}, \tag{56}$$

$$\text{where} \qquad \mathbf{B}_{nl\times nl} = \begin{bmatrix} \mathbf{C}_{l\times l} & -\mathbf{I}_{l\times l} & \iddots & 0 \\ -\mathbf{I}_{l\times l} & \mathbf{C}_{l\times l} & \ddots & 0 \\ \ddots & \ddots & \ddots & -\mathbf{I}_{l\times l} \\ 0 & 0 & -\mathbf{I}_{l\times l} & \mathbf{C}_{l\times l} \end{bmatrix},$$

$$\text{where} \qquad \mathbf{C}_{l\times l} = \begin{bmatrix} 6 & -1 & \iddots & 0 \\ -1 & 6 & \ddots & 0 \\ \ddots & \ddots & \ddots & -1 \\ 0 & 0 & -1 & 6 \end{bmatrix}$$

$$\text{and} \qquad \boldsymbol{\phi}_{mnl} = \begin{bmatrix} \phi_{1,1,1} \\ \vdots \\ \phi_{1,1,l} \\ \vdots \\ \phi_{1,n,l} \\ \vdots \\ \phi_{m,n,l} \end{bmatrix}, \quad \mathbf{b}_{mnl} = \frac{\Delta x^2}{\varepsilon_0} \begin{bmatrix} \rho_{1,1,1} \\ \vdots \\ \rho_{1,1,l} \\ \vdots \\ \rho_{1,n,l} \\ \vdots \\ \rho_{m,n,l} \end{bmatrix} \tag{57}$$

and $\mathbf{I}_{nl\times nl}, \mathbf{I}_{l\times l}$ are identity matrices.

### 13.2.1 Conjugate Gradient method

In previous section Poisson equation (Eq. 48) was reduced to a linear system of size $mnl$:

$$\mathbf{Ax} = \mathbf{b} \tag{58}$$

Matrix **A** in Eq. 58 is huge and sparse. The best methods to solve sparse linear systems are iterative methods, because unlike the direct methods (SV, LU and QR decompositions) iterative methods allow to save memory and computation time. The CG (**C**onjugate **G**radient) is the most popular iterative method that provides high accuracy within acceptable duration. In order to solve Eq. 55 matrix **A** must be symmetric and positive-definite:

$$\begin{cases} \mathbf{A}^{\mathrm{T}} = \mathbf{A} & \text{— symmetric} \\ \mathbf{x}^{\mathrm{T}}\mathbf{Ax} > 0 & \text{— positive-definite} \end{cases} \tag{59}$$

If Eq. 59 satisfies the conditions, then CG is based on the idea of minimizing the function (Eq. 60):

$$f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^{\mathrm{T}}\mathbf{Ax} - \mathbf{x}^{\mathrm{T}}\mathbf{b} \tag{60}$$

Function (Eq. 60) is minimized, when its gradient is zero:

$$\nabla f(\mathbf{x}) = \mathbf{Ax} - \mathbf{b} = 0 \tag{61}$$

which is equivalent to Eq. 58. The iteration starts with an arbitrary guess for $\mathbf{x} = \mathbf{x}_0$, which evaluates the gradient $\nabla f(\mathbf{x})$ at a certain value. The residual gradient value after $k$ iterations is expressed as:

$$\mathbf{r}_k = \mathbf{b} - \mathbf{A}\mathbf{x}_k, \tag{62}$$

where $\mathbf{x}_k$ is the solution after $k$ iterations. Up to this point CG follows identically GD (**G**radient **D**escent) method. The difference between CG and GD lies in how the minimization (solution) direction is determined. GD performs function minimization by following its steepest direction. GD convergence is known to be inefficient and to have oscillating trajectory, because steepest descent constantly "overshoots" the best path to minimum. GD also fails, when search direction is orthogonal to the gradient. Convergence speed heavily depends on the initital guess. Therefore CG unlike GD insists the search direction not to be residual $\mathbf{r}_k$, but instead a *conjugate direction* $\mathbf{p}_k$:

$$\mathbf{p}_k = \mathbf{r}_k - \sum_{i<k} \frac{\mathbf{p}_i^{\mathrm{T}}\mathbf{A}\mathbf{r}_k}{\mathbf{p}_i^{\mathrm{T}}\mathbf{A}\mathbf{p}_i}\mathbf{p}_i, \tag{63}$$

where $i$ stands for iterations previous to $k$. The next iteration towards solution is given by:

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k, \text{ where } \quad \alpha_k = \frac{\mathbf{r}_k^{\mathrm{T}}\mathbf{r}_k}{\mathbf{p}_k^{\mathrm{T}}\mathbf{A}\mathbf{p}_k} \tag{64}$$

The algorithm has to store search direction and residue vectors and to perform many matrix-vector multiplications. By reusing variables and storages the algorithm can made less computationally expensive with additional variable $\beta$.

### 13.2.2 Generalized Minimum Residual method

In realistic problems the matrix $\mathbf{A}$ is not symmetric and/or non-positive-definite due the boundary conditions imposed to the problem. In such a case the ordinary Conjugate Gradient algorithm fails to converge. CG is the special case of the more general GMRES (**G**eneralized **M**inimum **RES**idual) method [42]. GMRES minimizes the function:

$$f(\mathbf{x}) = \frac{1}{2}\left|\mathbf{A}\mathbf{x} - \mathbf{b}\right|^2 = \frac{1}{2}\mathbf{r}^{\mathrm{T}}\mathbf{r} \tag{65}$$

GMRES is a more universal iterative algorithm for solving sparse linear systems than CG.

### 13.2.3 Preconditioning

CG convergence and efficiency can be even more improved by using *preconditioning* [43, 44]. The convergence rate of CG is determined by the spectrum of eigenvalues of the matrix $\mathbf{A}$ [45]. Convergence rate can be accelerated by replacing the system (Eq. 58) with a preconditioned system (Eq. 66):

$$\mathbf{M}^{-1}\mathbf{A}\mathbf{x} = \mathbf{M}^{-1}\mathbf{b} \tag{66}$$

Preconditioned systems are useful only if $\mathbf{M}\mathbf{z} = \mathbf{r}$ can be solved with less computational work than the original system (Eq. 58) and matrix $\mathbf{M}$ must be symmetric and positive-definite too. Numerical experiments indicate that often preconditioning matrix $\mathbf{M}$ can be an *incomplete Cholesky decomposition* of $\mathbf{A}$ [46, 47]. Pseudocode of unconditioned CG and precondtitioned CG are here in {Appendix D} (Alg. 1, 2) respectively.

### 13.2.4 Boundary conditions

In linear algebraic systems the matrix contain all the information about the system. If the matrix is square, each matrix row can be represented as a node in a grid. The numeric entries in each matrix row represent arithmetic relation of that node to all other nodes in the grid. Poisson matrix (Eq. 56) is a square matrix with 1st order FDM (**F**inite **D**ifference **M**ethod) encoded for solving Poisson equation (2nd order non-homogeneous partial differential equation) in 3D domain (Fig. 35). However boundary conditions are excluded up to this point and are not represented in the Poisson matrix (Eq. 56). In order to impose bounary condition one has to modify Poisson matrix (Eq. 56). Since each row represents a particular node AND its relations to ALL other nodes, by changing numerical entries in that particular row one can modify the “purpose” of the node in the grid. The node’s “purpose” can be changed from “solvable node” (regular FDM) to an “imposed node” (Neumann condition) or “imposed difference node” (Dirichlet condition). “Imposed nodes” and “imposed difference nodes” do not participate in the CG (**C**onjugate **G**radient) convergence. This is my personal jargon to describe Neumann and Dirichlet conditions. In this way each node can be “assigned” to an be an ordinary, Neumann or Dirichlet node. In this way a problem designer can form and build structures in the domain by “assigning” chunks and formations of multiple nodes to “serve” a particular purpose, i.e. defining constant electric potential and constant electric field structures (electric insulator and conductor) in the domain. Nodes as the rows of Poisson matrix (Eq. 56) are distinguised by particular numerical pattern (1st order FDM) and has to be handled accurately when modifying them, because there are 27 types of them, which can be organized in 4 major groups: volume nodes, wall nodes, edge nodes and corner nodes. Each of the groups has a slightly different FDM pattern, that has to be taken into account for modifying. Below (Fig. 13.2.4) there is an attempt to describe modification of Poisson matrix (Eq. 56) presented in a most understandable fasion.

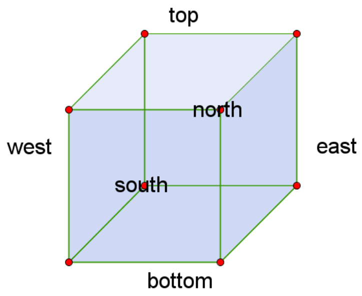


Figure 35: 3D domain

$$
\left[\begin{array}{ccccccccccccc}
\ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots \\
\ddots & -1 & \ddots & -1 & \ddots & -1 & 6 & -1 & \ddots & -1 & \ddots & -1 & \ddots \\
\ddots & -1 & \ddots & -1 & \ddots & \ddots & 6 & -1 & \ddots & -1 & \ddots & -1 & \ddots \\
\ddots & -1 & \ddots & -1 & \ddots & -1 & 6 & \ddots & \ddots & -1 & \ddots & -1 & \ddots \\
\ddots & -1 & \ddots & \ddots & \ddots & -1 & 6 & -1 & \ddots & -1 & \ddots & -1 & \ddots \\
\ddots & -1 & \ddots & -1 & \ddots & -1 & 6 & -1 & \ddots & \ddots & \ddots & -1 & \ddots \\
\ddots & \ddots & \ddots & -1 & \ddots & -1 & 6 & -1 & \ddots & -1 & \ddots & -1 & \ddots \\
\ddots & -1 & \ddots & -1 & \ddots & -1 & 6 & -1 & \ddots & -1 & \ddots & \ddots & \ddots \\
\ddots & -1 & \ddots & \ddots & \ddots & \ddots & 6 & -1 & \ddots & -1 & \ddots & -1 & \ddots \\
\ddots & -1 & \ddots & \ddots & \ddots & -1 & 6 & \ddots & \ddots & -1 & \ddots & -1 & \ddots \\
\ddots & -1 & \ddots & -1 & \ddots & \ddots & 6 & -1 & \ddots & \ddots & \ddots & -1 & \ddots \\
\ddots & -1 & \ddots & -1 & \ddots & -1 & 6 & \ddots & \ddots & \ddots & \ddots & -1 & \ddots \\
\ddots & \ddots & \ddots & \ddots & \ddots & -1 & 6 & -1 & \ddots & -1 & \ddots & -1 & \ddots \\
\ddots & \ddots & \ddots & -1 & \ddots & -1 & 6 & -1 & \ddots & \ddots & \ddots & -1 & \ddots \\
\ddots & \ddots & \ddots & -1 & \ddots & \ddots & 6 & -1 & \ddots & -1 & \ddots & -1 & \ddots \\
\ddots & \ddots & \ddots & -1 & \ddots & -1 & 6 & \ddots & \ddots & -1 & \ddots & -1 & \ddots \\
\ddots & -1 & \ddots & \ddots & \ddots & -1 & 6 & -1 & \ddots & -1 & \ddots & \ddots & \ddots \\
\ddots & -1 & \ddots & -1 & \ddots & -1 & 6 & -1 & \ddots & \ddots & \ddots & \ddots & \ddots \\
\ddots & -1 & \ddots & -1 & \ddots & \ddots & 6 & -1 & \ddots & -1 & \ddots & \ddots & \ddots \\
\ddots & -1 & \ddots & -1 & \ddots & -1 & 6 & \ddots & \ddots & -1 & \ddots & \ddots & \ddots \\
\ddots & \ddots & \ddots & \ddots & \ddots & \ddots & 6 & -1 & \ddots & -1 & \ddots & -1 & \ddots \\
\ddots & \ddots & \ddots & \ddots & \ddots & -1 & 6 & \ddots & \ddots & -1 & \ddots & -1 & \ddots \\
\ddots & \ddots & \ddots & -1 & \ddots & \ddots & 6 & -1 & \ddots & \ddots & \ddots & -1 & \ddots \\
\ddots & \ddots & \ddots & -1 & \ddots & -1 & 6 & \ddots & \ddots & \ddots & \ddots & -1 & \ddots \\
\ddots & -1 & \ddots & \ddots & \ddots & \ddots & 6 & -1 & \ddots & -1 & \ddots & \ddots & \ddots \\
\ddots & -1 & \ddots & \ddots & \ddots & -1 & 6 & \ddots & \ddots & -1 & \ddots & \ddots & \ddots \\
\ddots & -1 & \ddots & -1 & \ddots & \ddots & 6 & -1 & \ddots & \ddots & \ddots & \ddots & \ddots \\
\ddots & -1 & \ddots & -1 & \ddots & -1 & 6 & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots \\
\ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots
\end{array}\right]
\longrightarrow
\left[\begin{array}{ccccccccccccc}
\ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots \\
\ddots & 0 & \ddots & 0 & \ddots & 0 & 1 & 0 & \ddots & 0 & \ddots & 0 & \ddots \\
\ddots & 0 & \ddots & 0 & \ddots & \ddots & 1 & 0 & \ddots & 0 & \ddots & 0 & \ddots \\
\ddots & 0 & \ddots & 0 & \ddots & 0 & 1 & \ddots & \ddots & 0 & \ddots & 0 & \ddots \\
\ddots & 0 & \ddots & \ddots & \ddots & 0 & 1 & 0 & \ddots & 0 & \ddots & 0 & \ddots \\
\ddots & 0 & \ddots & 0 & \ddots & 0 & 1 & 0 & \ddots & \ddots & \ddots & 0 & \ddots \\
\ddots & \ddots & \ddots & 0 & \ddots & 0 & 1 & 0 & \ddots & 0 & \ddots & 0 & \ddots \\
\ddots & 0 & \ddots & 0 & \ddots & 0 & 1 & 0 & \ddots & 0 & \ddots & \ddots & \ddots \\
\ddots & 0 & \ddots & \ddots & \ddots & \ddots & 1 & 0 & \ddots & 0 & \ddots & 0 & \ddots \\
\ddots & 0 & \ddots & \ddots & \ddots & 0 & 1 & \ddots & \ddots & 0 & \ddots & 0 & \ddots \\
\ddots & 0 & \ddots & 0 & \ddots & \ddots & 1 & 0 & \ddots & \ddots & \ddots & 0 & \ddots \\
\ddots & 0 & \ddots & 0 & \ddots & 0 & 1 & \ddots & \ddots & \ddots & \ddots & 0 & \ddots \\
\ddots & \ddots & \ddots & \ddots & \ddots & 0 & 1 & 0 & \ddots & 0 & \ddots & 0 & \ddots \\
\ddots & \ddots & \ddots & 0 & \ddots & 0 & 1 & 0 & \ddots & \ddots & \ddots & 0 & \ddots \\
\ddots & \ddots & \ddots & 0 & \ddots & \ddots & 1 & 0 & \ddots & 0 & \ddots & 0 & \ddots \\
\ddots & \ddots & \ddots & 0 & \ddots & 0 & 1 & \ddots & \ddots & 0 & \ddots & 0 & \ddots \\
\ddots & 0 & \ddots & \ddots & \ddots & 0 & 1 & 0 & \ddots & 0 & \ddots & \ddots & \ddots \\
\ddots & 0 & \ddots & 0 & \ddots & 0 & 1 & 0 & \ddots & \ddots & \ddots & \ddots & \ddots \\
\ddots & 0 & \ddots & 0 & \ddots & \ddots & 1 & 0 & \ddots & 0 & \ddots & \ddots & \ddots \\
\ddots & 0 & \ddots & 0 & \ddots & 0 & 1 & \ddots & \ddots & 0 & \ddots & \ddots & \ddots \\
\ddots & \ddots & \ddots & \ddots & \ddots & \ddots & 1 & 0 & \ddots & 0 & \ddots & 0 & \ddots \\
\ddots & \ddots & \ddots & \ddots & \ddots & 0 & 1 & \ddots & \ddots & 0 & \ddots & 0 & \ddots \\
\ddots & \ddots & \ddots & 0 & \ddots & \ddots & 1 & 0 & \ddots & \ddots & \ddots & 0 & \ddots \\
\ddots & \ddots & \ddots & 0 & \ddots & 0 & 1 & \ddots & \ddots & \ddots & \ddots & 0 & \ddots \\
\ddots & 0 & \ddots & \ddots & \ddots & \ddots & 1 & 0 & \ddots & 0 & \ddots & \ddots & \ddots \\
\ddots & 0 & \ddots & \ddots & \ddots & 0 & 1 & \ddots & \ddots & 0 & \ddots & \ddots & \ddots \\
\ddots & 0 & \ddots & 0 & \ddots & \ddots & 1 & 0 & \ddots & \ddots & \ddots & \ddots & \ddots \\
\ddots & 0 & \ddots & 0 & \ddots & 0 & 1 & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots \\
\ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots
\end{array}\right]
\begin{array}{c}
\\
\text{volume} \\
\text{west wall} \\
\text{east wall} \\
\text{south wall} \\
\text{north wall} \\
\text{bottom wall} \\
\text{top wall} \\
\text{south west edge} \\
\text{south east edge} \\
\text{north west edge} \\
\text{north east edge} \\
\text{bottom south edge} \\
\text{bottom north edge} \\
\text{bottom west edge} \\
\text{bottom east edge} \\
\text{top south edge} \\
\text{top north edge} \\
\text{top west edge} \\
\text{top east edge} \\
\text{bottom south west corner} \\
\text{bottom south east corner} \\
\text{bottom north west corner} \\
\text{bottom north east corner} \\
\text{top south west corner} \\
\text{top south east corner} \\
\text{top north west corner} \\
\text{top north east corner} \\
\\
\end{array}
$$

Figure 36: Modification of Poisson matrix rows for Neumann boundary conditions for each group of nodes

Matrix modification for Dirichlet conditions is described only for <u>wall nodes</u>. Dirichlet conditions only make sence describing domains with “open walls”, where they imply electric field continuation and periodicity of the domain in the direction of the “open wall”. Dirichlet conditions in the <u>volume nodes</u> would mean a source of constant electric field, which is a nonphysical case at least in NBI physics.

$$\begin{bmatrix}
\ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots \\
\ddots & -1 & \ddots & -1 & \ddots & \ddots & 6 & -1 & \ddots & -1 & \ddots & -1 & \ddots \\
\ddots & -1 & \ddots & -1 & \ddots & -1 & 6 & \ddots & \ddots & -1 & \ddots & -1 & \ddots \\
\ddots & -1 & \ddots & \ddots & \ddots & -1 & 6 & -1 & \ddots & -1 & \ddots & -1 & \ddots \\
\ddots & -1 & \ddots & -1 & \ddots & -1 & 6 & -1 & \ddots & \ddots & \ddots & -1 & \ddots \\
\ddots & \ddots & \ddots & -1 & \ddots & -1 & 6 & -1 & \ddots & -1 & \ddots & -1 & \ddots \\
\ddots & -1 & \ddots & -1 & \ddots & -1 & 6 & -1 & \ddots & -1 & \ddots & \ddots & \ddots \\
\ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots
\end{bmatrix}
\longrightarrow
\begin{array}{cc}
\left[\begin{matrix}
\ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots \\
\ddots & 0 & \ddots & 0 & \ddots & \ddots & 1 & -1 & \ddots & 0 & \ddots & 0 & \ddots \\
\ddots & 0 & \ddots & 0 & \ddots & -1 & 1 & \ddots & \ddots & 0 & \ddots & 0 & \ddots \\
\ddots & 0 & \ddots & \ddots & \ddots & 0 & 1 & 0 & \ddots & -1 & \ddots & 0 & \ddots \\
\ddots & 0 & \ddots & -1 & \ddots & 0 & 1 & 0 & \ddots & \ddots & \ddots & 0 & \ddots \\
\ddots & \ddots & \ddots & 0 & \ddots & 0 & 1 & 0 & \ddots & 0 & \ddots & -1 & \ddots \\
\ddots & -1 & \ddots & 0 & \ddots & 0 & 1 & 0 & \ddots & 0 & \ddots & \ddots & \ddots \\
\ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots
\end{matrix}\right]
&
\begin{matrix}
\\
\text{west wall} \\
\text{east wall} \\
\text{south wall} \\
\text{north wall} \\
\text{bottom wall} \\
\text{top wall} \\
\\
\end{matrix}
\end{array}$$

Figure 37: Modification of Poisson matrix rows for Dirichlet boundary conditions for wall nodes

Once Neumann and Dirichlet conditions are encoded into Poisson matrix, the system can be solved only using GMRES ( 13.2.2). Modified matrix is no longer symmetric, therefore CG (13.2.1) is no more applicable, because by design CG converges only for symmetric matrices.

### 13.2.5 Initial conditions

Whereas Poisson matrix **A** (Eq. 56) describes the system completely including boundary conditions, the vector **b** (Eq. 57) sets the <u>initial conditions</u> of the system. An initial value of the electric potential is assigned to each node in the domain, before solver iteration starts. Initial values are as well assigned to the Neumann/Dirichlet nodes. Neumann/Dirichlet nodes acquire a numeric value, which will be kept at constant or at constant difference throughtout solver iteration due to aforediscussed (13.2.4) modifications in the Poisson matrix.

## 13.3 Electric field computation

Once Poisson equation is solved (Eq. 48 by iterative means and scalar potential field $\phi$ is known, the electric field computation on grid nodes is straightforward. The electric field vector is defined as the negative gradient of the potential:

$$\mathbf{E} = -\nabla\phi \tag{67}$$

$$E_x(x,y,z)\hat{\mathbf{x}} + E_y(x,y,z)\hat{\mathbf{y}} + E_z(x,y,z)\hat{\mathbf{z}} = -\frac{\partial\phi(x,y,z)}{\partial x}\hat{\mathbf{x}} - \frac{\partial\phi(x,y,z)}{\partial y}\hat{\mathbf{y}} - \frac{\partial\phi(x,y,z)}{\partial z}\hat{\mathbf{z}} \tag{68}$$

Discretized using central difference scheme:

$$\begin{cases}
E_{x\ i,j,k} = \frac{\phi_{i+1,j,k} - \phi_{i-1,j,k}}{2\Delta x} \\
E_{y\ i,j,k} = \frac{\phi_{i,j+1,k} - \phi_{i,j-1,k}}{2\Delta y} \\
E_{z\ i,j,k} = \frac{\phi_{i,j,k+1} - \phi_{i,j,k+1}}{2\Delta z}
\end{cases} \tag{69}$$

Particles that are present in the domain feel the electric field of the grid nodes. The effective field value at the exact position of the particle has to be determined from the electric field values of the nearest nodes (cell nodes) by first-order trilinear (3D) interpolation (Eq. 70):

$$\left\{\begin{array}{lllllll}
E_x & = & E_{x\ i,j,k} & \frac{(x_{i+1}-x)(y_{j+1}-y)(z_{k+1}-z)}{\Delta x \Delta y \Delta z} & + \ E_{x\ i+1,j,k} & \frac{(x-x_i)(y_{j+1}-y)(z_{k+1}-z)}{\Delta x \Delta y \Delta z} & + \\
 & + & E_{x\ i,j+1,k} & \frac{(x_{i+1}-x)(y-y_j)(z_{k+1}-z)}{\Delta x \Delta y \Delta z} & + \ E_{x\ i+1,j+1,k} & \frac{(x-x_i)(y-y_j)(z_{k+1}-z)}{\Delta x \Delta y \Delta z} & + \\
 & + & E_{x\ i,j,k+1} & \frac{(x_{i+1}-x)(y_{j+1}-y)(z-z_k)}{\Delta x \Delta y \Delta z} & + \ E_{x\ i+1,j,k+1} & \frac{(x-x_i)(y_{j+1}-y)(z-z_k)}{\Delta x \Delta y \Delta z} & + \\
 & + & E_{x\ i,j+1,k+1} & \frac{(x_{i+1}-x)(y-y_j)(z-z_k)}{\Delta x \Delta y \Delta z} & + \ E_{x\ i+1,j+1,k+1} & \frac{(x-x_i)(y-y_j)(z-z_k)}{\Delta x \Delta y \Delta z} & \\
E_y & = & E_{y\ i,j,k} & \frac{(x_{i+1}-x)(y_{j+1}-y)(z_{k+1}-z)}{\Delta x \Delta y \Delta z} & + \ E_{y\ i+1,j,k} & \frac{(x-x_i)(y_{j+1}-y)(z_{k+1}-z)}{\Delta x \Delta y \Delta z} & + \\
 & + & E_{y\ i,j+1,k} & \frac{(x_{i+1}-x)(y-y_j)(z_{k+1}-z)}{\Delta x \Delta y \Delta z} & + \ E_{y\ i+1,j+1,k} & \frac{(x-x_i)(y-y_j)(z_{k+1}-z)}{\Delta x \Delta y \Delta z} & + \\
 & + & E_{y\ i,j,k+1} & \frac{(x_{i+1}-x)(y_{j+1}-y)(z-z_k)}{\Delta x \Delta y \Delta z} & + \ E_{y\ i+1,j,k+1} & \frac{(x-x_i)(y_{j+1}-y)(z-z_k)}{\Delta x \Delta y \Delta z} & + \\
 & + & E_{y\ i,j+1,k+1} & \frac{(x_{i+1}-x)(y-y_j)(z-z_k)}{\Delta x \Delta y \Delta z} & + \ E_{y\ i+1,j+1,k+1} & \frac{(x-x_i)(y-y_j)(z-z_k)}{\Delta x \Delta y \Delta z} & \\
E_z & = & E_{z\ i,j,k} & \frac{(x_{i+1}-x)(y_{j+1}-y)(z_{k+1}-z)}{\Delta x \Delta y \Delta z} & + \ E_{z\ i+1,j,k} & \frac{(x-x_i)(y_{j+1}-y)(z_{k+1}-z)}{\Delta x \Delta y \Delta z} & + \\
 & + & E_{z\ i,j+1,k} & \frac{(x_{i+1}-x)(y-y_j)(z_{k+1}-z)}{\Delta x \Delta y \Delta z} & + \ E_{z\ i+1,j+1,k} & \frac{(x-x_i)(y-y_j)(z_{k+1}-z)}{\Delta x \Delta y \Delta z} & + \\
 & + & E_{z\ i,j,k+1} & \frac{(x_{i+1}-x)(y_{j+1}-y)(z-z_k)}{\Delta x \Delta y \Delta z} & + \ E_{z\ i+1,j,k+1} & \frac{(x-x_i)(y_{j+1}-y)(z-z_k)}{\Delta x \Delta y \Delta z} & + \\
 & + & E_{z\ i,j+1,k+1} & \frac{(x_{i+1}-x)(y-y_j)(z-z_k)}{\Delta x \Delta y \Delta z} & + \ E_{z\ i+1,j+1,k+1} & \frac{(x-x_i)(y-y_j)(z-z_k)}{\Delta x \Delta y \Delta z} &
\end{array}\right. \tag{70}$$

Electric field at the particle position $(x, y, z)$ is equal $\vec{E} = (E_x, E_y, E_z)$. The same calculation has to be repeated for all the particles present in the simulation.

## 13.4 Particle motion

Once electric field is known at the particle position, Newtons's laws can be applied to move the particle. Particle dynamics is expressed in the 2nd law of Newton, i.e. acceleration (change of velocity) is proportionate to the acting force. In this this case the acting force is Lorentz force. Lorentz force is the force that acts upon a charged particle in the presence of electromagnetic fields.

$$\frac{\mathrm{d}\vec{v}}{\mathrm{d}t} = \frac{\vec{F}}{m} = \frac{q}{m}(\vec{E} + \vec{v} \times \vec{B}), \tag{71}$$

where $\vec{E}$ is the electric field vector at the particle position, $\vec{B}$ is the magnetic field vector at the particle position, $\vec{v}$ is the particle velocity and $q$ is the charge of the particle. The particle position follows simple kinetic relation (definition of velocity).

$$\frac{\mathrm{d}\vec{x}}{\mathrm{d}t} = \vec{v} \tag{72}$$

### 13.4.1 Forward-difference motion integration

It may seem that integration of motion equations (Eq. 71, 72)) is trivial. One could use forward-difference discrete integration scheme to obtain recurrent form (in the case of $\vec{B} = 0$):

$$\begin{cases} \vec{v}_{n+1} = \vec{v}_n + \frac{q}{m}\vec{E}\Delta t \\ \vec{x}_{n+1} = \vec{x}_n + \vec{v}_n \Delta t \end{cases} \tag{73}$$

However this scheme produces **<u>incorrect</u>** results. Practice proved that systems described with this scheme are unstable and do not conserve energy regardless of the size of time step $\Delta t$.

### 13.4.2 Leapfrog method

*Leapfrog* method is a viable attempt to have a stable integration scheme. In forward-difference scheme using velocity $\vec{v}_n$ to produce the velocity $\vec{v}_{n+1}$ accumulates the discretization error and drives the system unstable. Therefore using an average velocity between steps could compensate the error. Averaging of velocity is obtained by displacing velocity integration by half a time step $\frac{\Delta t}{2}$. That inspired the name of the method, because integration of velocity and position schematically remind of two frogs leaping over each other every half a time step (Fig. 38).

$$\begin{cases} \vec{v}_0 \longleftarrow \vec{v}_0 - \frac{q}{m}\vec{E}\frac{\Delta t}{2} \\ \vec{v}_{n+1} = \vec{v}_n + \frac{q}{m}\vec{E}\Delta t \\ \vec{x}_{n+1} = \vec{x}_n + \vec{v}_n\Delta t \end{cases} \tag{74}$$

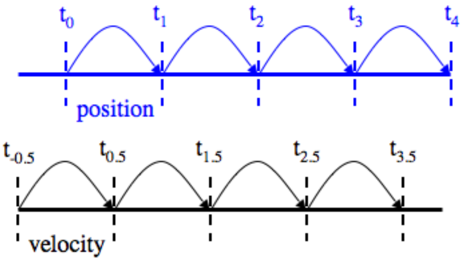


Figure 38: Leapfrog method: particle position is evaluated at integral time steps, velocity is evaluated at half in between steps

Leapfrog method works ok for the case $\vec{B} = 0$. However if magnetic field $\vec{B}$ contribution is taken into account, integration starts experiencing instability by violating energy conservation in the same fashion as forward-difference scheme does.

### 13.4.3 Boris algorithm

In 1971 J. P. Boris proposed an integration scheme [48], which was so accurate and resilient to error accumulation, that up-to-date it remains the main algorithm for particle advancement in electromagnetic simulations. It is capable to track the particle for an arbitrarily large number of time steps even with accuracy than fourth-order Runge-Kutta scheme [49, 50]. By discretizing equation Eq. 72 one obtains:

$$\frac{\vec{v}_{n+1} - \vec{v}_n}{\Delta t} = \frac{q}{m}\left(\vec{E} + \frac{\vec{v}_{n+1} + \vec{v}_n}{2} \times \vec{B}\right) \tag{75}$$

By using substitutions $\begin{bmatrix} \vec{v}_n & = \vec{v}^- - \frac{q\vec{E}}{m}\frac{\Delta t}{2} \\ \vec{v}_{n+1} & = \vec{v}^+ + \frac{q\vec{E}}{m}\frac{\Delta t}{2} \end{bmatrix}$ Eq. 75 becomes:

$$\frac{\vec{v}^+ - \vec{v}^-}{\Delta t} = \frac{q}{m}\left(\frac{\vec{v}^+ + \vec{v}^-}{2} \times \vec{B}\right) \tag{76}$$

Vectors $\vec{v}^+$ and $\vec{v}^-$ have the same magnitude but different directions (Fig. 39). Therefore Eq. 76 represents a rotation from $\vec{v}^-$ and $\vec{v}^+$ (it can be proved by multiplying Eq. 76 with $(\vec{v}^+ + \vec{v}^-)$) [40]. The solution is contructed in such a way:

$$\vec{v}^+ = \vec{v}^- + \left(\vec{v}^- + \vec{v}^- \times \vec{t}\right) \times \vec{s}, \quad \text{where} \quad \begin{cases} \vec{s} = \frac{2\vec{t}}{1+\vec{t}^2} \\ \vec{t} = \frac{q\vec{B}}{m}\frac{\Delta t}{2} \end{cases} \tag{77}$$

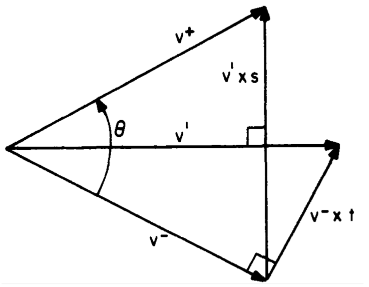


Figure 39: Geometry of $\vec{v}^+$, $\vec{v}^-$, $\vec{t}$ and $\vec{s}$

Boris algorithm has an amazing feature to conserves energy exactly, even when there is no electric field. And its relativistic implementation is straightforward.

## 13.5 Collisions

Modelling collisions is complicated and expensive in computational resources. Therefore if the problem does not require collisions, researchers try to avoid including collisional processes into simulations. There are two approaches to model collisions in plasma: binary collisions [51] and MCC (**M**onte **C**arlo **c**ollisions) [41, 52, 53]. However binary collisions are computationaly expensive and in some sense remind the case of N-body problem. MCC is more computationally friendly, because at its core it has a Monte Carlo based computation. In general Monte Carlo method is useful for solving/modelling problems, which have many degrees of freedom, nonlinear nature, many variables and/or uncertainty in inputs. Monte Carlo gives its evaluations in probabilistic sense. By repeated random sampling over parameter/variable space Monte Carlo gathers the output values into a histogram. If the *ergodic theorem* holds true for the system/process, the empirical measures (histogram) must approach the stationary probability distribution, which is then intepreted as the ultimate probabilistic description of the system/process. In other words Monte Carlo takes uniform random distribution and uses it as an input, the problem/system itself works as a filter/“sieve” to shape the original random distribution into a output distribution. Monte Carlo mehod was invented in the late 1940s by Stanislaw Ulam, while he was working on nuclear weapons projects at the Los Alamos National Laboratory.

### 13.5.1 Monte Carlo collisions (MCC)

In weakly ionized plasmas the most dominant collisional process is the interaction between charged and neutral particles. Computationally the two groups of particles are called the *background species* ($H_2$) and *particle species*. Background species density is described as a smooth function of time and space (it can be and most often is uniform and constant in time). Background species do not participate in the PIC process as particles, they are just assumed in the background with a certain defined density. Particle species participate in the PIC process, move in response to the local electric field and therefore their distribution evolve temporally and spatially. Let one assume particle species $s$ (*incident species*) have $N$ possible types of collisions with another particle species (*target species*). The kinetic energy of the $i$th particle of the incident species $s$ is:

$$\mathcal{E}_i = \frac{1}{2} m_s \vec{v}_i^2 = \frac{1}{2} m_s \left( v_{x\,i}^2 + v_{y\,i}^2 + v_{z\,i}^2 \right) \tag{78}$$

This energy is needed in calculating the collision cross sections. The total collision cross section $\sigma_{\mathrm{Total}}(\mathcal{E}_i)$ is the sum:

$$\sigma_{\mathrm{Total}}(\mathcal{E}_i) = \sigma_1(\mathcal{E}_i) + \cdots + \sigma_N(\mathcal{E}_i) = \sum_j^N \sigma_j(\mathcal{E}_i), \tag{79}$$

where $\sigma_j(\mathcal{E}_i)$ is the cross section of the $j$th type of collision between the incident species $s$ and target species. The collision probability for the $i$th particle is calculated based on the distance $\Delta s_i = v_i \Delta t$ travelled in each time step $\Delta t$:

$$P_i = 1 - e^{-\Delta s \sigma_{\mathrm{Total}}(\mathcal{E}_i) n_{\mathrm{target}}(\vec{x}_i)} = 1 - e^{-v_i \Delta t \sigma_{\mathrm{Total}}(\mathcal{E}_i) n_{\mathrm{target}}(\vec{x}_i)}, \tag{80}$$

where $n_{\mathrm{target}}(\vec{x}_i)$ is the local density of the target species at the position of the $i$th particle. In order to intiate a collision a random number is chosen from interval $[0, 1]$. If the random number is less than $P_i$, then collision is counted as happened and another random number is drawn to determine the type of collision. The energy and scattering angle of collision products are determined based on the model in power for that type of collision. A collision is assumed to take place in the time interval $[t, t + \Delta t]$ at the current position $\vec{x}_i(t)$ of the $i$th particle, therefore time step $\Delta t$ determines the accuracy of the collisional model. The accuracy is challenged, if many collisions take place during $\Delta t$. The probability for $N$ collisions to take place during $\Delta t$ is equal $P_i^N$. However MCC allows only one collision per particle per $\Delta t$, therefore each iteration generates an error $r$.

$$r \approx \sum_{k=2}^{\infty} P_i^k = \frac{P_i^2}{1 - P_i} \tag{81}$$

Error $r$ has to be managed in order not to exceed reasonable limits. An error limit of 0.01 requires $P_i < 0.95$, i.e. time step $\Delta t$ must set so that $\Delta s_i \sigma_{\mathrm{Total}}(\mathcal{E}_i) n_{\mathrm{target}}(\vec{x}) \leq 0.1$ [52].

### 13.5.2 Null Collision method

Calculating $P_i$ requires looking up the particle's kinetic energy. Doing that for all particles each time step makes the task very computationally expensive. The look-up can be reduced by introducing a virtual constant collision frequency $\nu'$ such that

$$\nu' = \max_{\vec{x},\mathcal{E}}(n_{\text{target}}\sigma_{\text{Total}}v) = \max_{\vec{x}}(n_{\text{target}})\max_{\mathcal{E}}(\sigma_{\text{Total}}v) \tag{82}$$

This virtual collision frequency flattens the total collision frequency $n_{\text{target}}(\vec{x})\sigma_{\text{Total}}(\mathcal{E})v$ by adding missing part and thus giving constant value over $\vec{x}$ and $\mathcal{E}$ (Fig. 40). The added part is called *null collision* process, since it does not represent any real physical interaction. However it helps to reduce the computational effort. Probability of a *null collision* is given by:

$$P_{\text{null}} = 1 - e^{-\nu'\Delta t} \tag{83}$$

Colliding particles are chosen randomly without duplicating and the each particle checked for the type of the collision using:

$$\begin{array}{rrll} & R & \leq \frac{\nu_1(\mathcal{E}_i)}{\nu'} & \text{collision type 1} \\ \frac{\nu_1(\mathcal{E}_i)}{\nu'} < & R & \leq \frac{\nu_1(\mathcal{E}_i)+\nu_2(\mathcal{E}_i)}{\nu'} & \text{collision type 2} \\ & & \vdots & \\ \sum_j^N \frac{\nu_j(\mathcal{E}_i)}{\nu'} < & R & & \textit{null collision} \end{array} \tag{84}$$

where $R$ is the a random number $R \in [0, 1]$. For a simulation with $N_s$ particles, the standard MCC method requires $N_s$ evaluations of $P_i$, whereas Null Collision method requires only $N_s P_{\text{null}}$ evaluations of $P_i$.

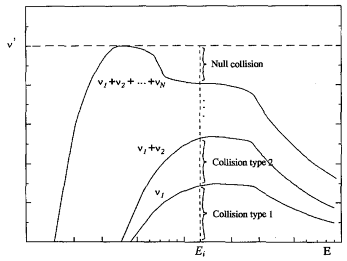


Figure 40: Null Collision frequency sets the upper limit for the look-up

# 14 Results

## 14.1 Solver evaluation

The main and the crucial part of PIC codes performance and validity is the domain solutions for $\phi$. This part of the code determines completely the quality of the PIC simulation. Therefore one desires to have the best and most reliable numericals solvers in order to produce the best and the most accurate domain solution for $\phi$. The computational plaform that the PIC code will be based on is GPGPU platform. Therefore one has to develop the code for GPGPU and use numerical tools and libraries built for GPGPU. In case of domain solvers the CG (13.2.1) and GMRES13.2.2 based solutions are realised by a `Cusp` library [54], which is a library for sparse linear algebra and graph computations based on `Thrust` library, which is built on CUDA. `Cusp` provides a flexible, high-level interface for manipulating sparse matrices and solving sparse linear systems on Nvidia GPU architecture.

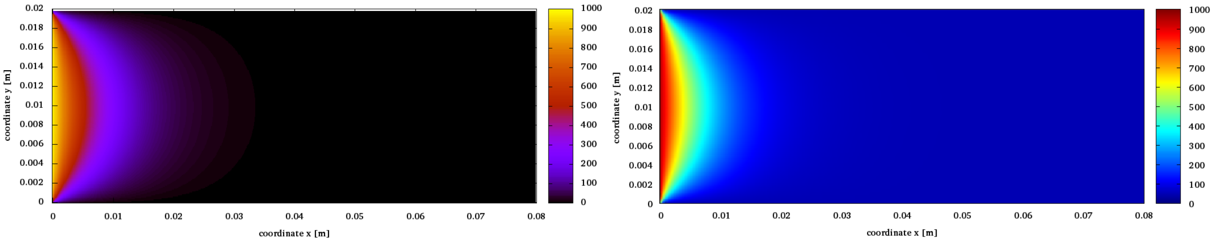
One would like to assure that solver is giving reasonable output and test the quality of the solution. Three tests are carried to evaluate the solutions for a test model of 2D rectangular domain of size 800 × 200 with a left wall set to 1000 [V] as a Dirichlet boundary condition (Fig. 41):

1. **Solver comparison**. `Cusp` solver output is compared to an output of a well-known `Comsol` software suite used in engineering. Figures 43 and 44 present the results for potential and electric field

calculations respectively. It is evident that solutions given by two solvers are identical for Fig. 43 and Fig. 44abd, except Fig. 44c. Fig. 44c profile contains numeric noise caused by discrete differentiation performed on a non-rectangular grid used in `Comsol`. Numerical noise was unintentionally exposed here, because the expected solution must be a zero profile. Nevertheless this validates and legitimates `Cusp` solver as a trustful tool for further studies, even as a better solver in case of Fig. 44c.

2. **Stencil study**. `Cusp` solver is able to provide solution using different orders of the solution. Size of the stencil determines the order of the solution. Stencil is defined as a total number of nodes participating in a solution of a particular node. It is common, that stencils of higher orders provide more accurate solution in expense of higher computational cost. However if the higher orders do not influence the outcome significantly, it is better to use lower order stencils in order to save computational resources and time. Figures 45 and 46 show no significant advantages of using `9pt` stencil. Therefore `5pt` stencil is used for further developments.

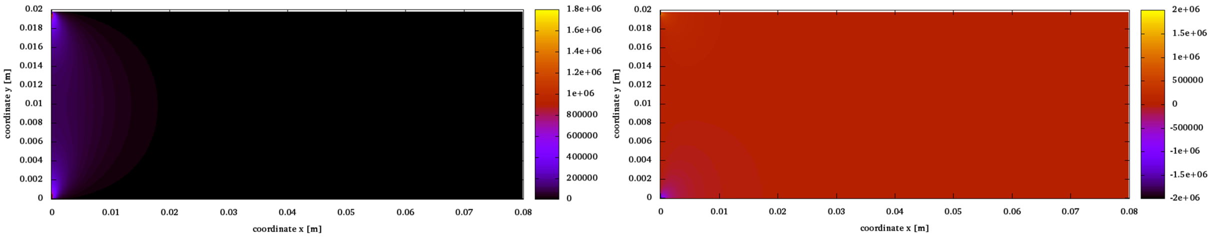
3. **Tolerance study**. `Cusp` solver has a flexible error tolerance as an input parameter. Solution quality as well as computational expenditures heavily depends on the set tolerance. Therefore one is interested in an optimal tolerance value, which is enough to provide a good quality solution for a reasonable computational cost. Figures 47 and 48 demonstrate that `Cusp` solver fails completely to provide the solution at tolerance $10^{-4}$. At tolerance $10^{-5}$ the solver is giving a reasonable solution, however tolerance $10^{-6}$ provides visible improvement. Therefore tolerance $10^{-6}$ is used for further studies and tolerance $10^{-7}$ is discarded as unnecessary.



(a) `Cusp` solver

(b) `Comsol` solver

Figure 41: Potential $\phi$ map with a left wall set at 1000 [V] (Dirichlet boundary condition) solved by different solvers

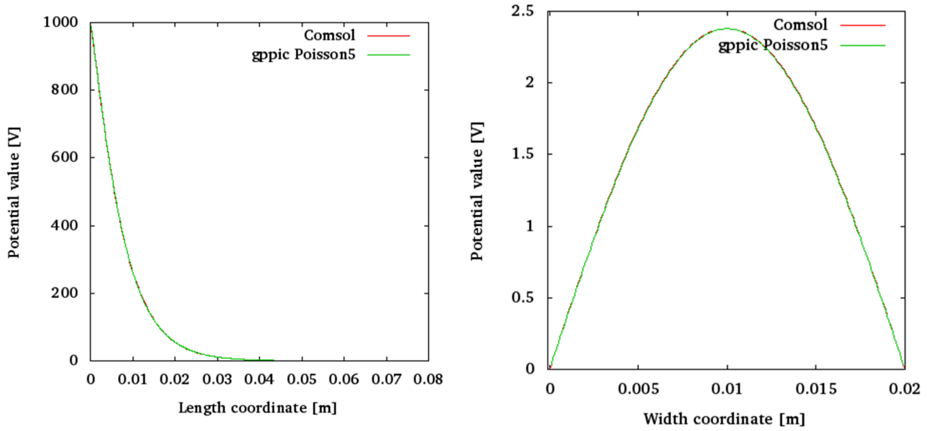


(a) Longitudinal electric field $\mathbf{E}_x$

(b) Transverse electric field $\mathbf{E}_y$

Figure 42: Electric field $\mathbf{E}$ maps: (a) longitudinal (b) transverse

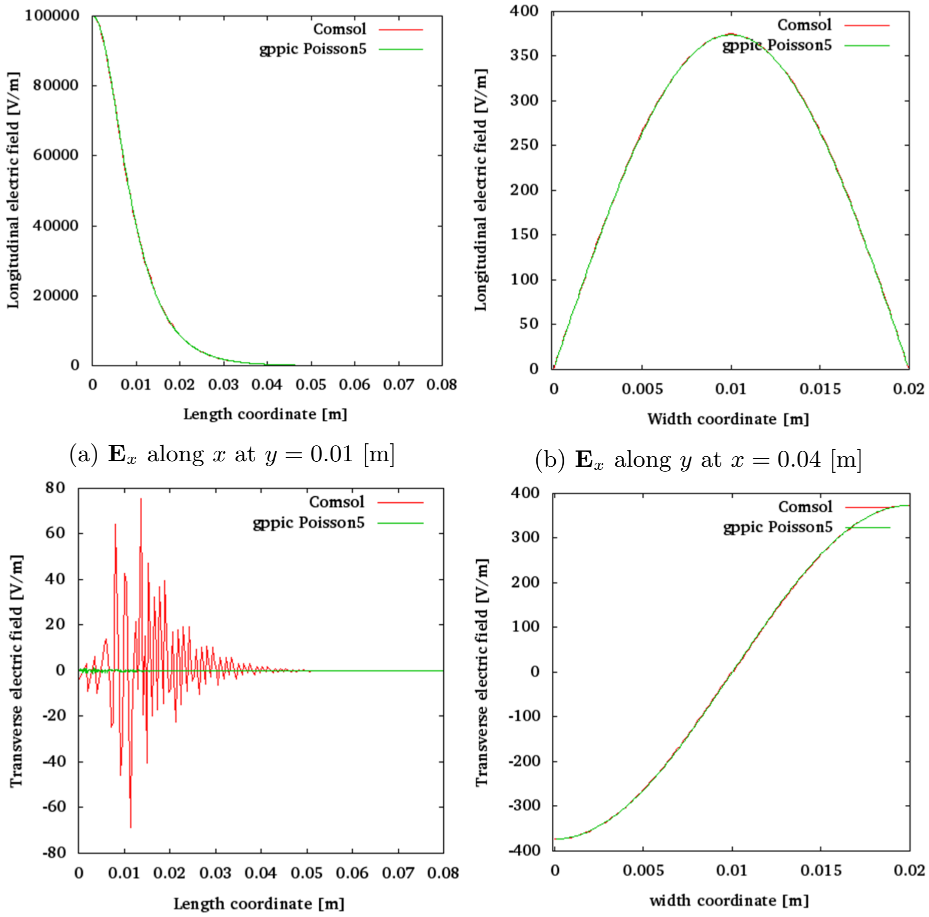
#### 14.1.1 Comsol vs Cusp



(a) $\phi$ along $x$ at $y = 0.01$ [m]

(b) *phi* along $y$ at $x = 0.04$ [m]

Figure 43: Potential $\phi$ profiles solved by different solvers: (a) longitudinal (b) transverse



(a) $\mathbf{E}_x$ along $x$ at $y = 0.01$ [m]

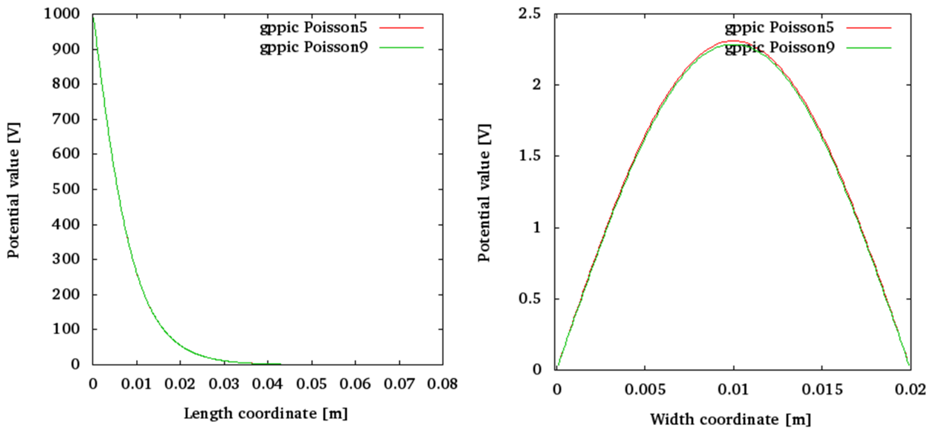
(b) $\mathbf{E}_x$ along $y$ at $x = 0.04$ [m]

(c) $\mathbf{E}_y$ along $x$ at $y = 0.01$ [m]

(d) $\mathbf{E}_y$ along $y$ at $x = 0.04$ [m]

Figure 44: Electric $\phi$ profiles solved by different solvers: (a) longitudinal along $x$ (b) longitudinal along $y$ (c) transverse along $x$ (d) transverse along $y$

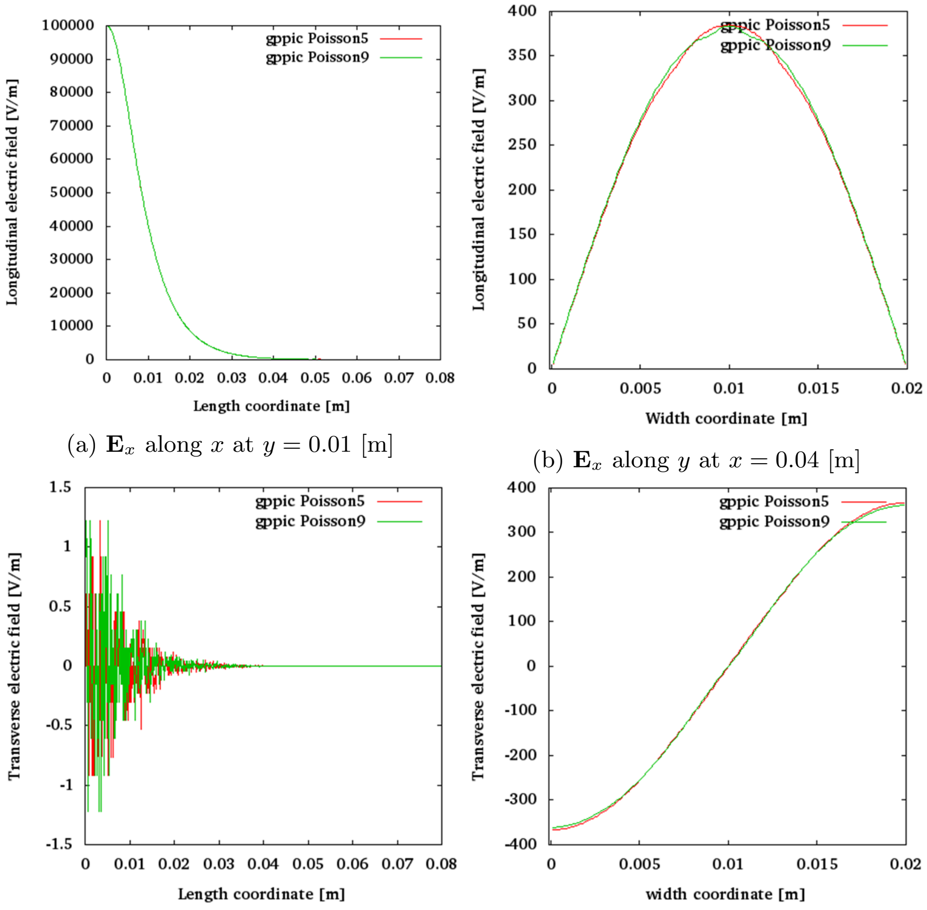
### 14.1.2 5pt stencil vs 9pt stencil



(a) along $x$ at $y = 0.01$ [m]

(b) along $y$ at $x = 0.04$ [m]

Figure 45: Potential $\phi$ profiles solved with different stencils: (a) longitudinal (b) transverse



(a) $\mathbf{E}_x$ along $x$ at $y = 0.01$ [m]

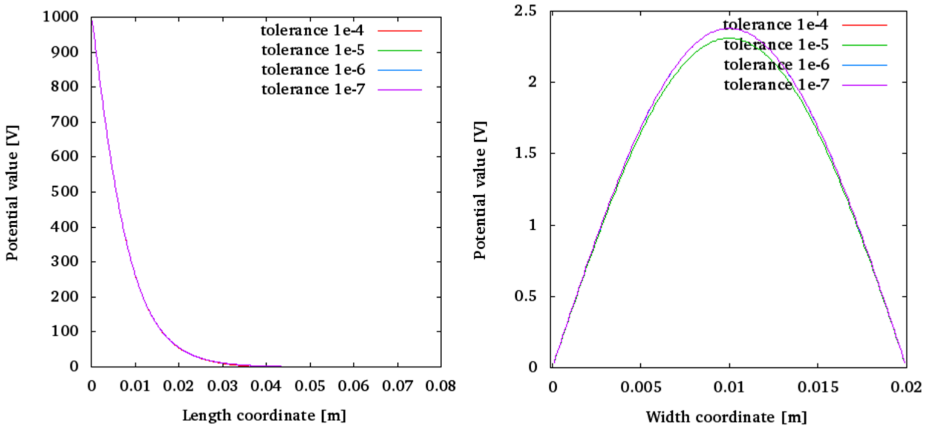
(b) $\mathbf{E}_x$ along $y$ at $x = 0.04$ [m]

(c) $\mathbf{E}_y$ along $x$ at $y = 0.01$ [m]

(d) $\mathbf{E}_y$ along $y$ at $x = 0.04$ [m]

Figure 46: Electric $\phi$ profiles solved with different stencils: (a) longitudinal along $x$ (b) longitudinal along $y$ (c) transverse along $x$ (d) transverse along $y$

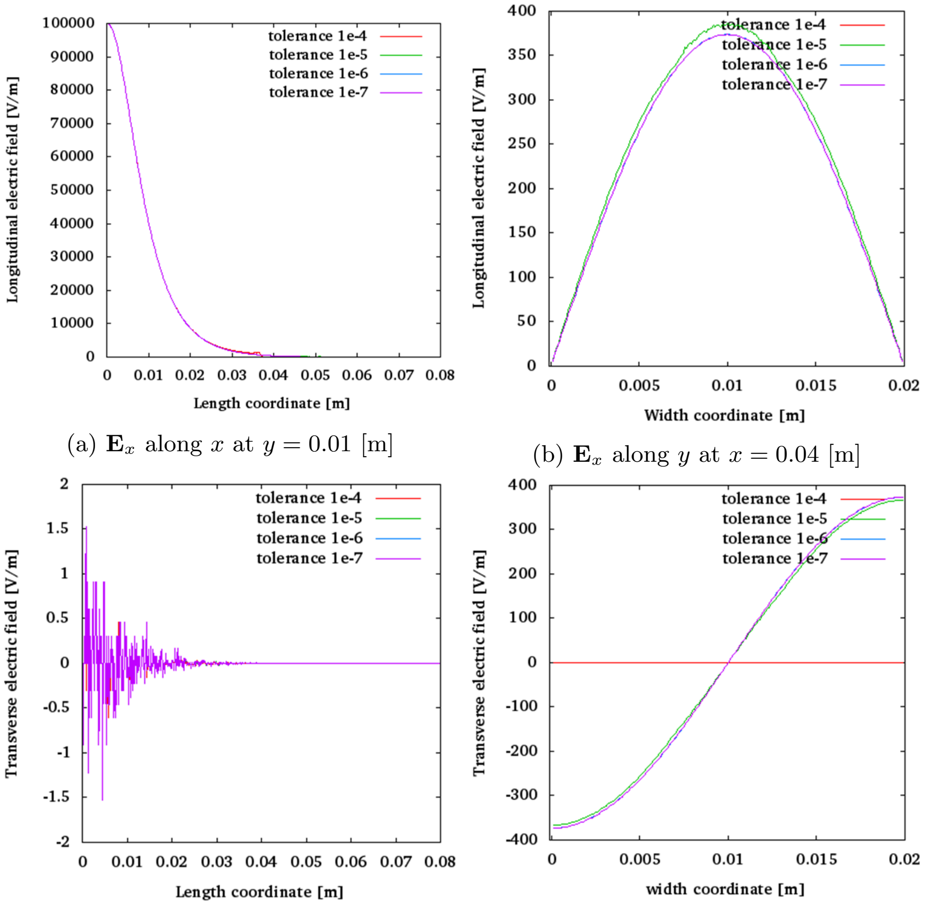
### 14.1.3 Tolerance study



(a) along $x$ at $y = 0.01$ [m]

(b) along $y$ at $x = 0.04$ [m]

Figure 47: Potential $\phi$ profiles solved with different tolerances: (a) longitudinal (b) transverse



(a) $\mathbf{E}_x$ along $x$ at $y = 0.01$ [m]

(b) $\mathbf{E}_x$ along $y$ at $x = 0.04$ [m]

(c) $\mathbf{E}_y$ along $x$ at $y = 0.01$ [m]

(d) $\mathbf{E}_y$ along $y$ at $x = 0.04$ [m]

Figure 48: Electric $\phi$ profiles solved with different tolerances: (a) longitudinal along $x$ (b) longitudinal along $y$ (c) transverse along $x$ (d) transverse along $y$

## 14.2 Transient state simulation

In the previous section solver validity was tested against various options. Solver proved to produce valid solutions. Domain solution for $\phi$ is the main part for PIC simulation. Once assured that this step works well, one can built a primitive PIC simulation. The simplest setup one can come up with is a box with arbitrary choice for boundary conditions. The choice of ours is a box with two opposite walls grounded to 0 volts (Fig. 49). Programmatically each grounded wall is realised by a combined set of wall nodes, surrounding edge nodes and surrounding corner nodes, which follows Dirichlet condition (Poisson matrix modified accordingly 13.2.4). Each "open" wall is (south,north,bottom and top) realised by Neumann condition (Poisson matrix modified accordingly 13.2.4). Particle movement with regard to "open" walls is set to follow periodic condition, e.g. particles crossing the south "open" wall are instantly "teleported" to the opposite side of the domain (next to the north "open" wall) and vice versa. Periodic conditions are implemented as south⇔north transition and bottom⇔north transition, i.e. coordinates of each particle follows conditions $(x, y, z)_i = (x, y \mod Y, z \mod Z)_i$, where Y=0.1 cm and Z=1 cm are domain dimensions. Particle movement with regard to grounded walls (west and east) is restricted to loss condition. Once particle crosses west or east wall, it is excluded from simulation and do not participate in the dynamics anymore, i.e. if $x_i < 0$ cm or $1$ cm $< x$, particle$_i \longrightarrow$ disabled.

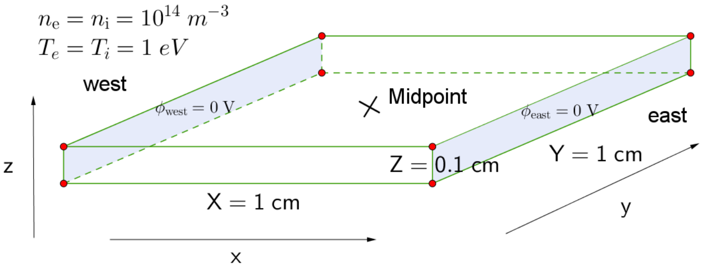


Figure 49: Domain characteristics for transient plasma state simulations with west- and east- wall potentials grounded $\phi_{\text{west}} = \phi_{\text{east}} = 0$ [V] and south-, north-, bottom-, top- walls are set as "open"

In fact the whole simulation setup is chosen to mimic a 1D case of PIC, where two grounded walls are infinite and particles are treated as charged infinite sheets. Given that periodic condtions work well and bounary conditions is implemented correctly , then one can save computational resources and use Z-direction-reduced domain instead of a "fat" 3D cubicle. It may seem, that it is an excessive effort to build 3D code for 1D simulation, but the motivation lies in the fact that physics should not change between 3D and 1D simulations. In fact this particular setup serves as a thorough benchmark for the 3D code. If the 3D code is able to mimic a trusted 1D simulation and reproduce its results, one can infer that the 3D code is producing valid physics. One of the trusted 1D simulations are found in [36]. It is a 1D PIC simulation for a fully ionized collisionless electron-H$^+$ion plasma of density $10^{14}$ [m$^{-1}$] between two grounded parallel plates with electron and ion temperature $T_\text{e} = T_\text{i} = 1$ [eV]. The focus of the simulation is the temporal evolution of midpoint potential value $\phi_\text{mid}$ (Fig. 50).

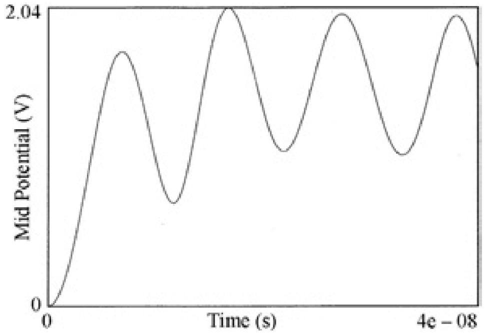


Figure 50: Reference simulation: midpoint potential $\phi_\text{mid}$ evolution over transient plasma state in [36]

Quality of PIC simulations and produced physical behaviour heavily depends on chosen parameters. There are 4 parameters that fully and completely determine the simulation quality: solver tolerance, integration time step $\Delta t$, cell size $\Delta x$ and number of particles per cell. Solver tolerance and number of particles per cell are parameters, which has solely numerical origin. They are not related to any physical magnitude. Meanwhile $\Delta t$ and $\Delta x$ are simulation parameters, which are of physical origin. In particularly $\Delta t$ is closely related to the inverse of plasma frequency $\omega_{\mathrm{pe}}$ (Eq. 85) and $\Delta x$ is closely related to the Debye length $\lambda_{\mathrm{D}}$ (Eq. 86).

$$\Delta t < \omega_{\mathrm{pe}}^{-1} = \frac{1}{\sqrt{\frac{n_{\mathrm{e}}e^2}{m_{\mathrm{e}}\epsilon_0}}} = \frac{1}{\sqrt{\frac{10^{14}\cdot(1.602\times10^{-19})^2}{9.109\times10^{-31}\cdot 8.854\times10^{-12}}}} = 1.77273\times10^{-9}\left[\frac{\mathrm{rad}}{\mathrm{s}}\right], \tag{85}$$

$$\Delta x < \lambda_{\mathrm{D}} = \sqrt{\frac{\epsilon_0 k_{\mathrm{B}} T_{\mathrm{e}}[\mathrm{K}]}{n_{\mathrm{e}}e^2}} = \sqrt{\frac{\epsilon_0 T_{\mathrm{e}}[\mathrm{eV}]}{n_{\mathrm{e}}e}} = \sqrt{\frac{8.854\times10^{-12}\cdot 1}{10^{14}\cdot 1.602\times10^{-19}}} = 0.000743427\ [\mathrm{m}] \tag{86}$$

where $n_{\mathrm{e}}$ is the electron density, $e$ is the electron charge, $m_{\mathrm{e}}$ is the electron mass, $\epsilon_0$ is the vacuum permittivity.

### 14.2.1 Tolerance study

| Parameter | Value |
|---|---|
| $\Delta t$ | $4\times10^{-10}$ [s] |
| $\Delta x$ | 0.0001 [m] |
| N per cell | 100 |
| tolerance | variable |

Table 5: Simulation parameters

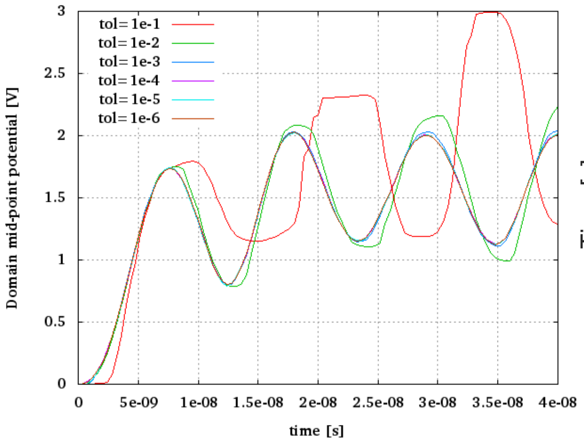


Figure 51: Midpoint potential $\phi_{\mathrm{mid}}$ evolution over transient plasma state with various solver tolerances

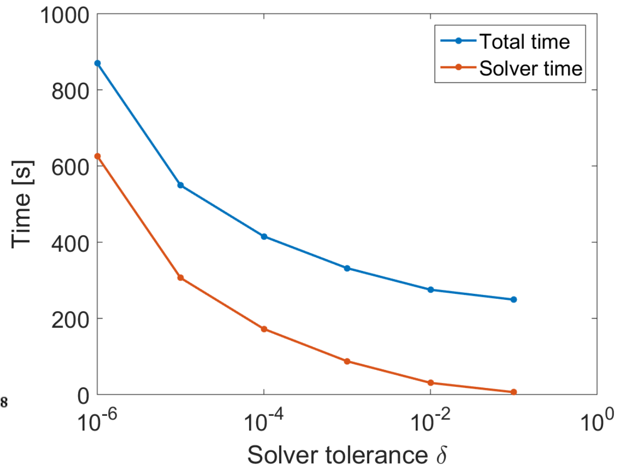


Figure 52: Code execution duration using various solver tolerances

### 14.2.2 Time step $\Delta t$ study

| Parameter | Value |
|---|---|
| $\Delta t$ | variable |
| $\Delta x$ | 0.0001 [m] |
| N per cell | 100 |
| tolerance | $10^{-6}$ |

Table 6: Simulation parameters

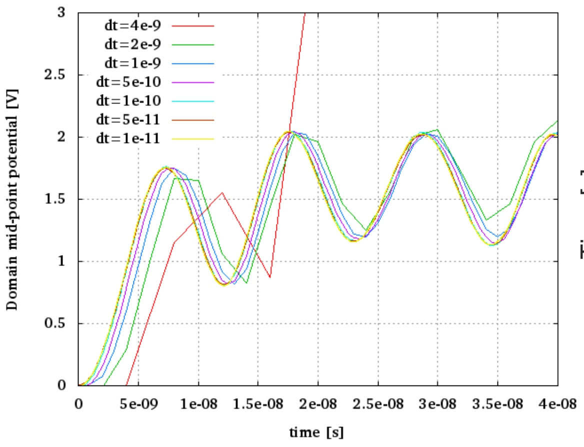


Figure 53: Midpoint potential $\phi_{\text{mid}}$ evolution over transient plasma state with various integration step sizes $\Delta t$

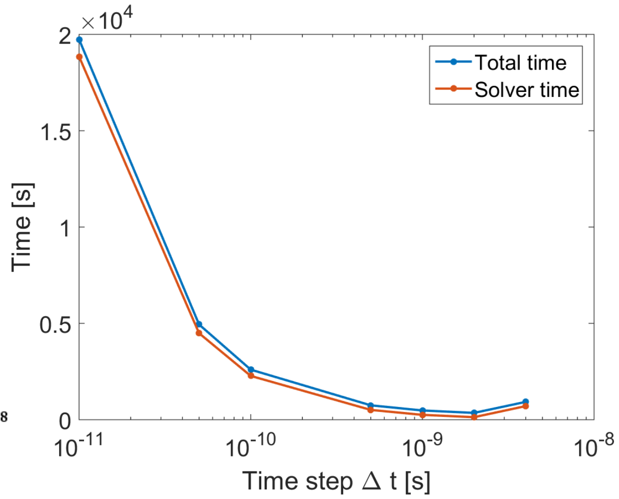


Figure 54: Code execution duration using various integration step sizes $\Delta t$

### 14.2.3 Cell size $\Delta x$ study

| Parameter | Value |
|---|---|
| $\Delta t$ | $5 \times 10^{-10}$ [s] |
| $\Delta x$ | variable |
| N per cell | 100 |
| tolerance | $10^{-6}$ |

Table 7: Simulation parameters

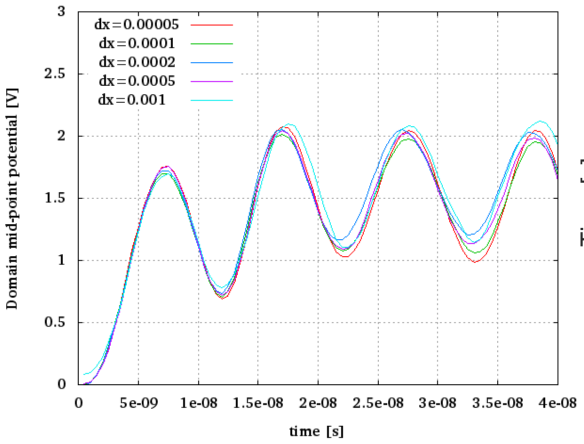


Figure 55: Midpoint potential $\phi_{\text{mid}}$ evolution over transient plasma state with various domain cell sizes $\Delta x$

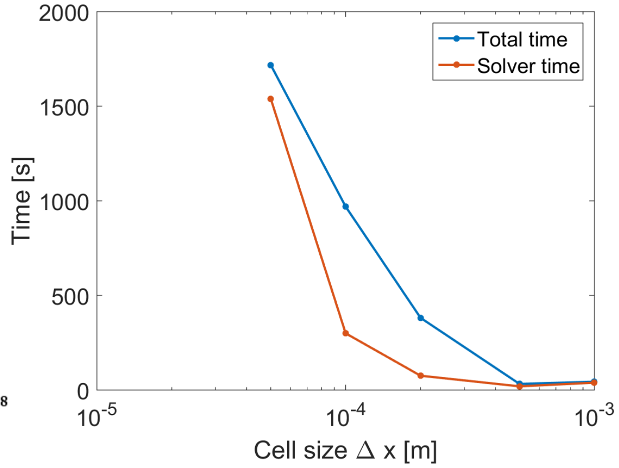


Figure 56: Code execution duration using various domain cell sizes $\Delta x$

### 14.2.4 Number of particles per cell study

| Parameter | Value |
|---|---|
| $\Delta t$ | $5 \times 10^{-10}$ [s] |
| $\Delta x$ | 0.001 [m] |
| N per cell | variable |
| tolerance | $10^{-6}$ |

Table 8: Simulation parameters

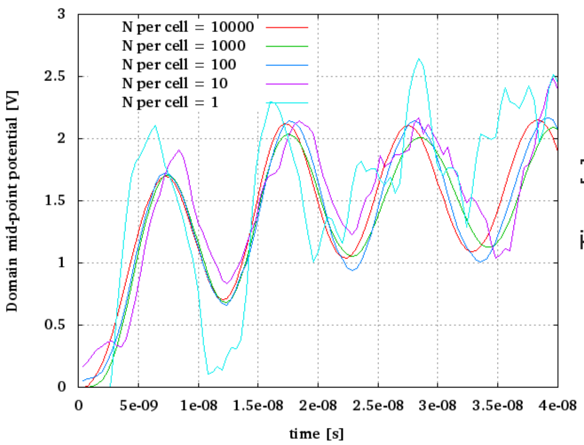


Figure 57: Midpoint potential $\phi_{\mathrm{mid}}$ evolution over transient plasma state with various numbers of particles per cell

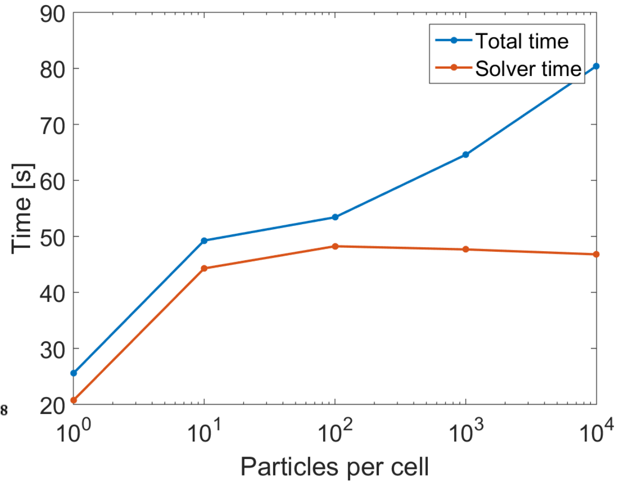


Figure 58: Code execution duration using various with various numbers of particles per cell

### 14.2.5 Discussion

In general code output is very close to the reference simulation in (Fig. 50) [36].

- Tolerance study (Fig. 51) suggests that $10^{-4}$ and smaller is a good choice for tolerance parameter. Solver duration and total duration follows the same nonlinear scaling towards smaller tolerance values (Fig. 52). This is due to the fact, that solver time constitutes the major part of the total code duration.

- Time step study (Fig. 53) suggests that $\Delta t = 10^{-9}$ [s] and below is valid, because time step condition set by Eq. 85 is respected. Solver duration (Fig. 54) constitutes the constant and major part of the total code duration throughout the range of $\Delta t$, because $\Delta t$ has no influence on that.

- Cell size study (Fig. 55) is valid for all tested $\Delta x$ values (Fig. 55), because cell size condition set by Eq. 86 is respected. Bigger $\Delta x$ were impossible to test and see the degenerated output, because code was crashing due to unknown error caused by too big $\Delta x$. Solver duration (Fig. 56) is bigger with smaller $\Delta x$ due to the fact, that grid increases in size (but not physical domain), i.e. in order to conserve the same physical box more nodes had to be present in the grid, therefore more computation for solver.

- Number of particles per cell study (Fig. 57) suggests that the number of 100 and above is a valid choice, which is supported by [55]. Solver duration (Fig. 58) saturates with respect to bigger numbers of particles per cell, because grid configuration stays constant and requires the same computational effort. Meanwhile total duration is growing, because the total number of particles is increasing, therefore there is more processing required for particle dynamics (but not for grid processing).

In principle code is producing predictable output, when input parameters are correct. This conclusion allows to develop simulations with higher complexity, because proper parameters can be chosen by respecting the Eq. 85, Eq. 86 and Eq. 26 conditions.

## 14.3 Steady state simulation

A logical and consistent step in development of PIC plasma simulations is simulating *sheath* and its formation. In order to obtain proper sheath formation one has to run simulation until steady state is achieved. However it is not enough just to extend the simulation duration. This straightforward approach to obtain steady state suffers from a phenomenon called *electron cooling*, where electron temperature $T_{\mathrm{e}}$ drops quickly. *Electron cooling* is an artificial effect of the simulation, which occurs due to the fact that simulation has NO collisions. In order to compensate for *electron cooling* one has to introduce thermostat process in the system. For a steady state simulation one will choose a cubic box of dimensions $4\times4\times4$ [cm] ($80 \times 80 \times 80$ nodes) with all the walls, edges and corners grounded to 0 V (Fig. 59) (applied Dirichlet conditions with respective Poisson matrix modification (13.2.4)). Inside the box there will be a fully ionized collisionless electron-$H^+$ ion plasma of density $10^{15}$ [$m^{-3}$] with electron and ion temperatures $T_e = 1$ [eV] and $T_i = 1$ [eV]. Particle movement with regard to grounded walls (all walls) is restricted to loss condition. Once particle crosses any wall, it is excluded from simulation and do not participate in the dynamics anymore, i.e. if $x_i < 0$ or $4 < x_i$ or $y_i < 0$ or $4 < y_i$ or $z_i < 0$ or $4 < z_i$ [cm], particle$_i \longrightarrow$ disabled. Simulations were run to simulate a fixed amount of time of 2 [$\mu$s] with time step $\Delta t = 0.5$ [ns] $< \frac{1}{\omega_{pe}} \approx 0.56$ [ns] (4000 steps) and cell size $\Delta x = 0.5$ [mm] $< \lambda_D \approx 0.743$ [mm]. The goals of the simulation:

1. to observe *electron cooling* and apply thermostat process to influence the temperature of the system

2. to obtain plasma potential $\phi_p$ and sheath depth $d$ at the steady state

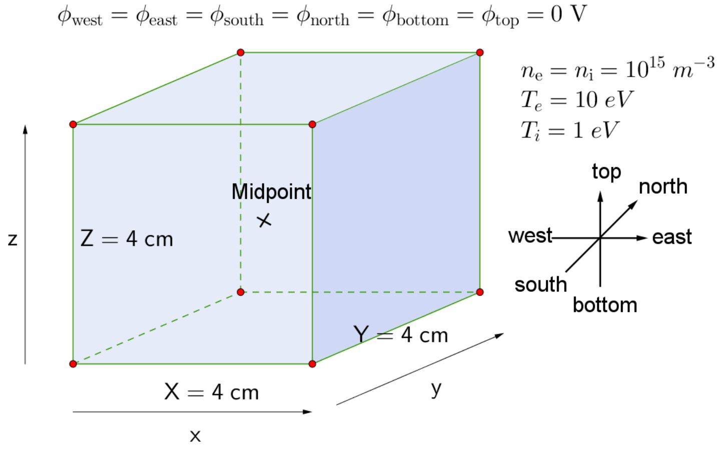

$\phi_{west} = \phi_{east} = \phi_{south} = \phi_{north} = \phi_{bottom} = \phi_{top} = 0$ V
$n_e = n_i = 10^{15}\ m^{-3}$
$T_e = 10\ eV$
$T_i = 1\ eV$
Midpoint
z
Z = 4 cm
Y = 4 cm
X = 4 cm
x
y
top
north
west
east
south
bottom

Figure 59: Domain characteristics for steady plasma state simulations with west-, east- , south-, north-, bottom- and top-wall potentials grounded $\phi_{west} = \phi_{east} = \phi_{south} = \phi_{north} = \phi_{bottom} = \phi_{top} = 0$ [V]

### 14.3.1 Plasma potential

The interest is to compare the observed plasma potential $\phi_p$ (Fig. 61) to the theoretical value. The observed electron temperature $T_e$ (Fig. 63) is used to calculate the theoretical plasma potential $\phi_p$ (Eq. 35). The whole study lets to inspect the quality and validity of the steady state simulation. The results are summarised in the Table 9.

| Thermostat parameter | Effective value | Observed $T_e$ [eV] | Observed $\phi_p$ [V] | Theoretical $\phi_p$ [V] |
|---|---|---|---|---|
| 0.0000 | 0.0000 | 2.32567 | 6.1101 | 7.76489 |
| 0.0001 | 0.00190138 | 2.63306 | 7.06627 | 8.7912 |
| 0.0005 | 0.00771014 | 3.59907 | 10.5001 | 12.0165 |
| 0.0010 | 0.0134763 | 4.93395 | 13.4884 | 16.4734 |
| 0.0050 | 0.0494515 | 7.34738 | 22.5942 | 24.5313 |
| 0.0100 | 0.0880169 | 8.37488 | 26.8717 | 27.9618 |

Table 9: Comparison of observed $\phi_p$ with theoretical $\phi_p$ (Eq. 35)

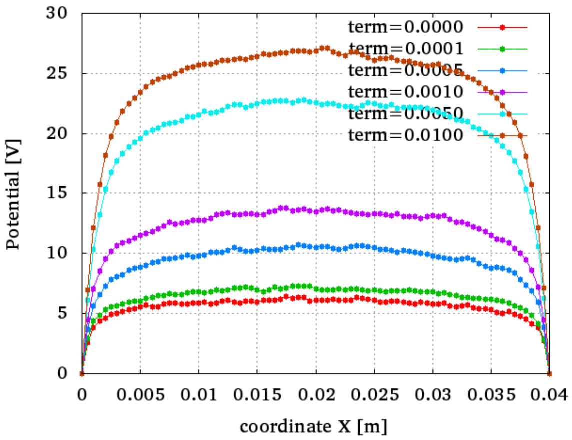


Figure 60: Electric potential $\phi$ profile along X direction at the time $t = 2$ [$\mu$s]

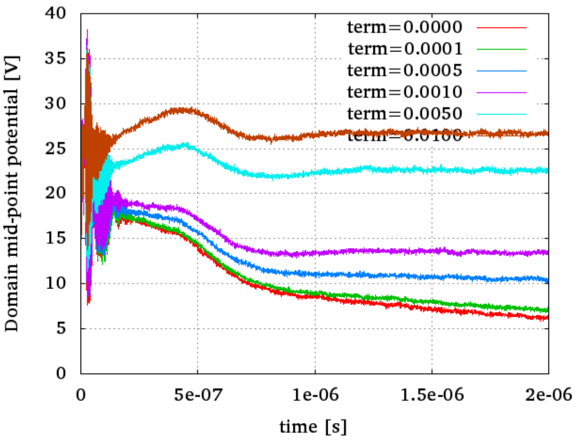


Figure 61: Electric potential $\phi$ evolution at the midpoint of the domain

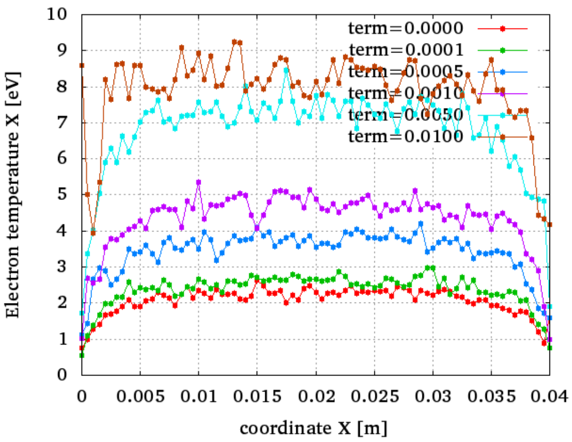


Figure 62: Electron temperature $T_e$ profile along X direction at the time $t = 2$ [$\mu$s]

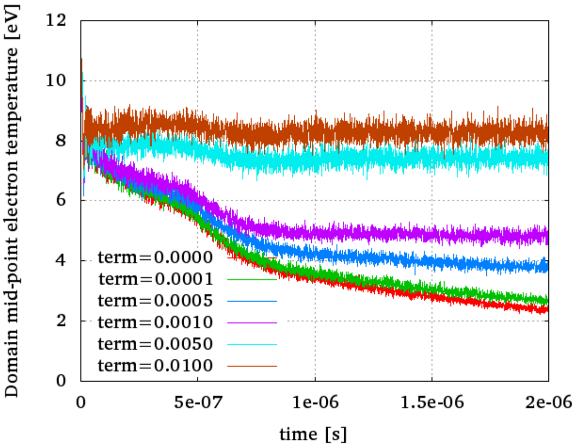


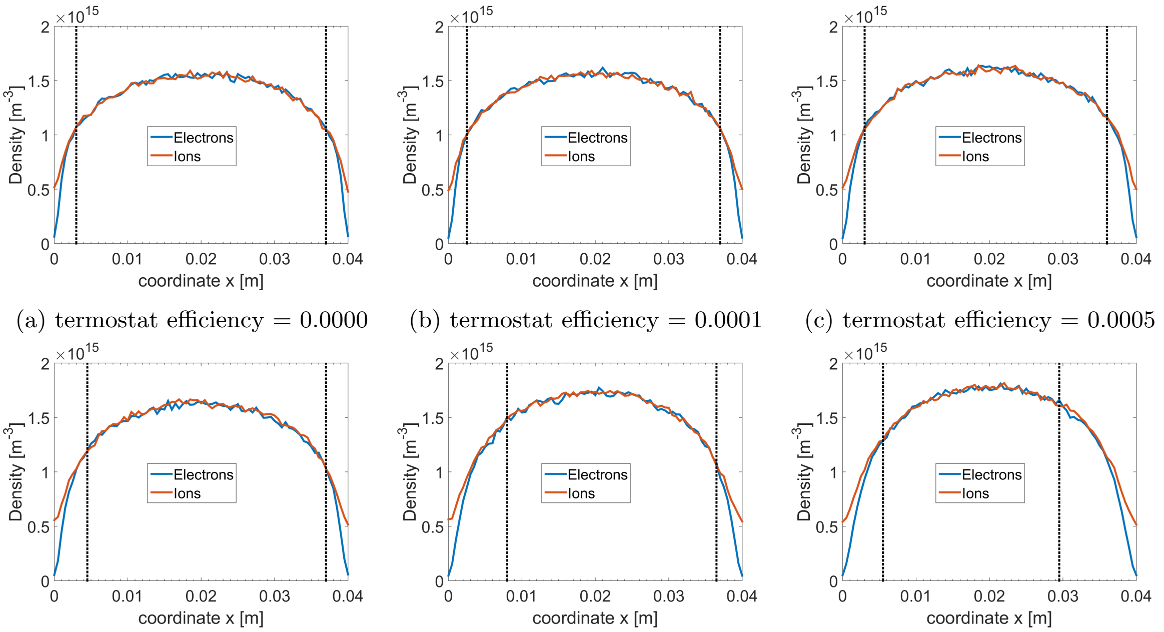
Figure 63: Electron temperature $T_e$ evolution at the midpoint of the domain

### 14.3.2 Sheath depth

The interest is to compare the observed sheath depths (Fig. 64) to the theoretical values. The observed electron temperature $T_e$ (Fig. 63) and plasma potential $\phi_p$ (Fig. 61) are used to calculate the theoretical sheath depths $d$ (Eq. 31). The whole study lets to inspect the quality and validity of the steady state simulation. The results are summarised in the Table 10.

| Thermostat parameter | Observed $T_e$ [eV] | Observed $\phi_p$ [V] | Observed $d$ [m] | Theoretical $d$ [m] |
|---|---|---|---|---|
| 0.0000 | 2.32567 | 6.1101 | 0.0030 | 0.000753123 |
| 0.0001 | 2.63306 | 7.06627 | 0.00275 | 0.00083989 |
| 0.0005 | 3.59907 | 10.5001 | 0.0035 | 0.00113038 |
| 0.0010 | 4.93395 | 13.4884 | 0.00375 | 0.00136396 |
| 0.0050 | 7.34738 | 22.5942 | 0.00575 | 0.0020083 |
| 0.0100 | 8.37488 | 26.8717 | 0.0080 | 0.00228718 |

Table 10: Comparison of observed sheath depth $d$ with Eq. 31



(a) termostat efficiency = 0.0000 (b) termostat efficiency = 0.0001 (c) termostat efficiency = 0.0005

(d) termostat efficiency = 0.0010 (e) termostat efficiency = 0.0050 (f) termostat efficiency = 0.0100

Figure 64: Electron and ion density profiles at steady state under various thermostat conditions with black lines indicating the observed bulk-sheath interface, where quasi-neutrality is broken

### 14.3.3 Discussion

- *Electron cooling* was observed in Fig. 63 (red curve). Thermostat effect is evident by stabilized electron temperature $T_\mathrm{e}$ at certain values, which go in correspondence to steady state plasma potential $\phi_\mathrm{p}$ (Fig. 61). Thermostat effect is entirely governed by thermostat efficiency (Eq. 44).

- Simulated plasma potential $\phi_\mathrm{p}$ and electron temperature $T_\mathrm{e}$ relation was tested against theoretical prediction Eq. 35 (Tab. 9) and is in acceptable agreement (Fig. 65).

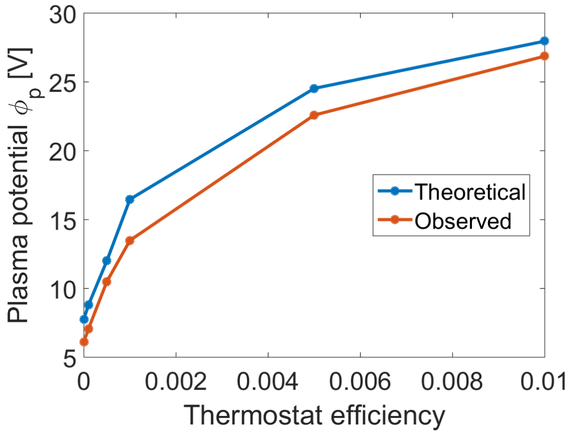


Figure 65: Comparison of theoretical and observed plasma potential $\phi_\mathrm{p}$ values using Table 9

- Sheath formation was observed and its depth was approximatelly defined from (Fig. 64) by taking average depth of left and right sheaths. This value was compared to theoretical prediction Eq. 31 (Tab. 10) and is NOT in a good agreement (Fig. 66). This may be caused by the fact, that bulk-sheath interface is not clearly manifested in the data and its determination is partially subjective, hence error-prone.

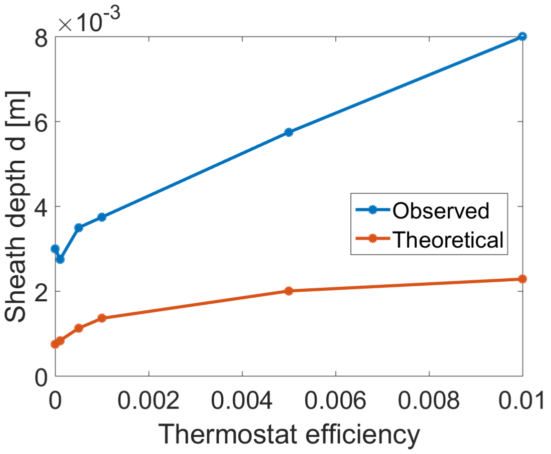


Figure 66: Comparison of theoretical and observed sheath depth $d$ values using Table 10

## 14.4 Negative ion beam extraction

In the previous section the GPU code has proved itself to actually produce reasonable and realistic physics. Yet the simulations did not have much of a practical use. In this section the PIC engine is exploited to simulate the extaction region of an NBI system. Negative ion beams are extremely important in fusion physics, since they will be providing significant part of heating power in fusion reactor (up to 50% in ITER)(Tab. 1) [56] and major part of DEMO heating and current drive [57]. Negative ion beams are subject to beam losses such as aberrations and beam divergence, which are caused by a non-optimal geometry and voltage configuration of plasma grid. Simulations take up a big part in the studies of NBI systems. 3D NBI simulations are extermely computationaly costly due to 3D domains and 3D particle data, which inflate memory and processing requirements by an order at least. Therefore scientists have no choice but to rely on 2D simulations and work on parameter scaling in order to make reasonable parameter extrapolations for 3D NBI geometry. In this thesis a full 3D simulation of NBI extraction grid is presented. The general interest of the simulation is to observe $H^-$ beam extraction and the special interest is to observe distinct properties of the beam between $H^-$ ions of different origin: volume-produced $H^-$ an surface-produced $H^-$.

### 14.4.1 Model description

Simulation domain contains an circular apperture (Fig. 67). Simulation domain models the *plasma grid* of MAMuG (**M**ulti-**A**perture **Mu**lti-**G**rid) design. Domain is a 3D box of dimensions $17 \times 19 \times 19$ [mm] ($137 \times 153 \times 153$ nodes) with a circular hole of radius 7 mm (56 nodes) and thickness 2 mm (16 modes) spanning from 9 to 11 mm with respect to the west wall. The west and east walls and the apperture comply with Dirichlet conditions. The west wall and apperture are grounded $\phi = 0$ [V] structures, while $\phi$ at east wall is set according to an *acceleration voltage*, which will take a parameter role in the simulations. The south, north, bottom and top walls comply with Neumann conditions in order to simulate plasma continuity in $Y$ and $Z$ directions. Particle loss and generation conditions are governed by a set of rules. Each set of rules varies depending on the species involved. Overall each species is programmed in order to mimick the real situation in beam extraction chamber. The behaviour can be summarized as:

1. There 3 species involved in the simulation: electrons $e^-$, positive ions $H^+$ and negative ions $H^-$. Negative ions $H^-$ are subdivided into two groups. One group is (re)generated in the volume and the other is (re)generated in at the surface.

2. Electrons $e^-$, positive ions $H^+$ and negative ions $H^-$ (volume) are (re)generated in a narrow region of 2 mm (16 nodes) width spanning from 2 to 4 mm with respect to the west wall (the red region in Fig. 68).

3. Negative ions H$^-$ (surface) are (re)generated in very narow region of 0.125 mm (1 node) next to the west side of the apperture in order to simulate surface emission. Surface emission is simulated by a half-Gaussian in the $X$ coordinate:

$$f(v_x) = \begin{cases} \sqrt{\frac{m}{2\pi k_\mathrm{B} T_\mathrm{e}}} e^{-\frac{m_\mathrm{e} v_x^2}{2k_\mathrm{B} T_\mathrm{e}}} & \text{if } v_x < 0 \\ 0 & \text{if } 0 \leq v_x \end{cases} \tag{87}$$

4. All species follow periodic boundary conditions such as south⇔north transition and bottom⇔north transition, i.e. coordinates of each particle follows conditions $(x, y, z)_i = (x, y \mod Y, z \mod Z)_i$, where Y=19 mm and Z=19 mm are domain dimensions.

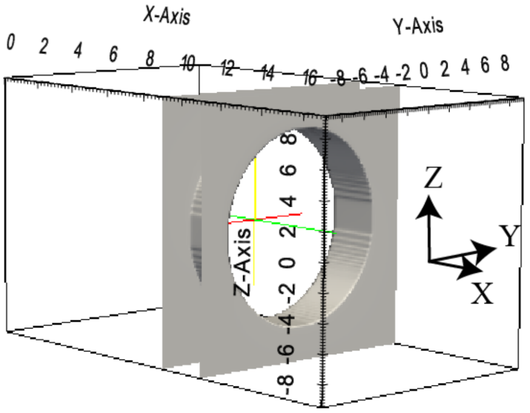


Figure 67: 3D model of an extraction aperture with a circular hole

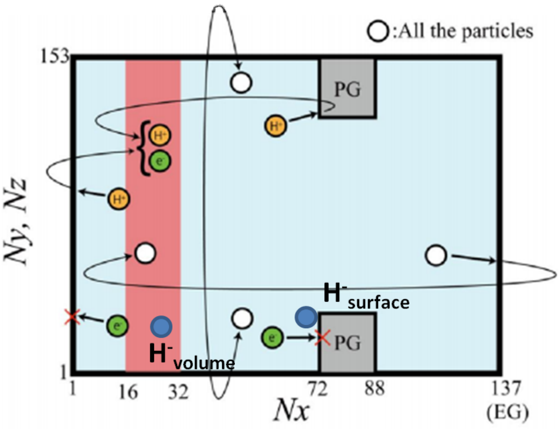


Figure 68: 2D projection of an extraction apperture with a circular hole and visualized particle management rules/conditions (Tab. 11)

| Species | Loss conditions | | | (Re)generation conditions | |
|---|---|---|---|---|---|
| | West wall | East wall | Apperture | Number of particles | Location |
| Electrons e$^-$ | Send to buffer | Immediate regeneration | Send to buffer | Pairwise with H$^+$ | Slab |
| Positive ions H$^+$ | Send to buffer | Immediate regeneration | Send to buffer | Pairwise with e$^-$ | Slab |
| Negative ions (volume) H$^-$ | Send to buffer | Send to buffer | Send to buffer | 412 macro-particles/$\Delta t$ | Slab |
| Negative ions (surface) H$^-$ | Send to buffer | Send to buffer | Send to buffer | Full buffer | Apperture surface |

Table 11: Particle management rules/conditions

### 14.4.2 Plasma parameters

As already mentioned simulation contains 4 species. Electrons e$^-$, positive ions H$^+$ and negative ions H$^-$ (volume) have reference density values, which are present at the start of simulation. Density of H$^-$ (volume) starts with 0 and is constantly generated at 200 A/m$^2$ and later evolves to a certain steady value, when losses equilibrates the generation. Densities, temperature and the other plasma parameters are given in Tab. 12.

| Parameter | Value | |
|---|---|---|
| $n_\mathrm{e}$ | $10^{-17}$ | [m$^{-3}$] |
| $n_{\mathrm{H}^+}$ | $10^{-17}$ | [m$^{-3}$] |
| $n_{\mathrm{H}^-\mathrm{volume}}$ | $5 \times 10^{-15}$ | [m$^{-3}$] |
| $n_{\mathrm{H}^-\mathrm{surface}}$ | variant during simulation | |
| $T_\mathrm{e}$ | 3 | [eV] |
| $T_{\mathrm{H}^+}$ | 1 | [eV] |
| $T_{\mathrm{H}^-\mathrm{volume}}$ | 1 | [eV] |
| $T_{\mathrm{H}^-\mathrm{surface}}$ | 1 | [eV] |
| $\Delta t$ | $5 \times 10^{-12}$ | [s] |
| $\Delta x$ | 0.000125 | [m] |
| $\lambda_\mathrm{Debye}$ | $\approx 40.7 \times 10^{-6}$ | [m] |

Table 12: Simulation parameters

Surface emission is assumed at $j$ =200 A/m$^2$ according to [58]. The emission area is calculated from the model's geometry (Fig. 68) as $A = (152^2 - 56^2\pi) \cdot \Delta x^2 = (152^2 - 56^2\pi) \cdot 0.000125^2 \approx 0.000207$ [m$^2$]. Therefore number of injected macro-particles per time step $\Delta t$ is derived as:

$$j = \frac{J}{A} = \frac{MNq}{A\Delta t} \quad \longrightarrow N = \frac{jA\Delta t}{Mq} = \frac{200 \cdot 0.000207 \cdot 5 \times 10^{-12}}{3139.13 \cdot 1.602 \times 10^{-19}} \approx 412, \tag{88}$$

where $M = 3139.13$ is the macro-particle factor, $N$ is the number of macro-particles entering domain per $\Delta t$, $\Delta t = 5 \times 10^{-12}$ [s] and $q$ is the elementary charge.

### 14.4.3 Double sheath

One of the goals in NBI extraction simulations is to observe so-called *double sheath*. The simulation is able to produce *double sheath* as a result of H$^-$ surface generation (Fig. 69).

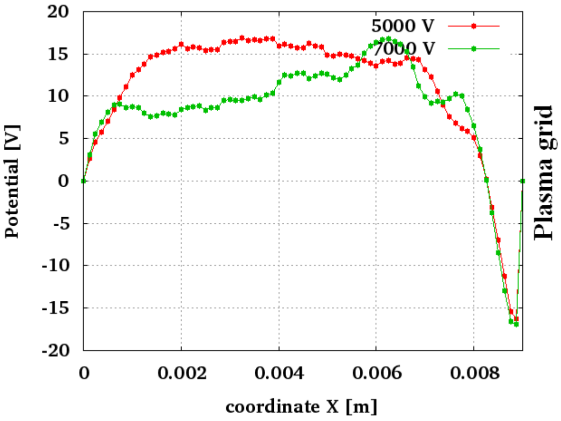


Figure 69: Observation of double sheath near the plasma grid (profile obtained at the corner of domain along X axis)

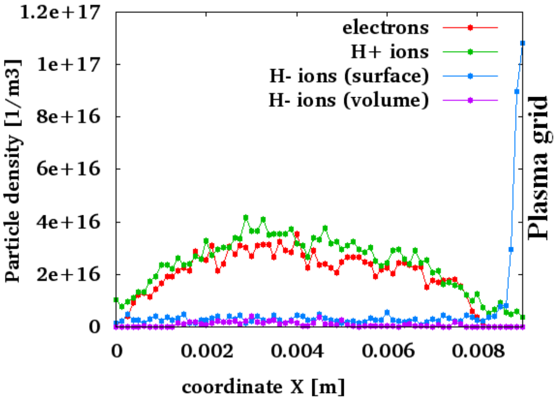


Figure 70: Density profiles of species (obtained at the corner of domain along X axis)

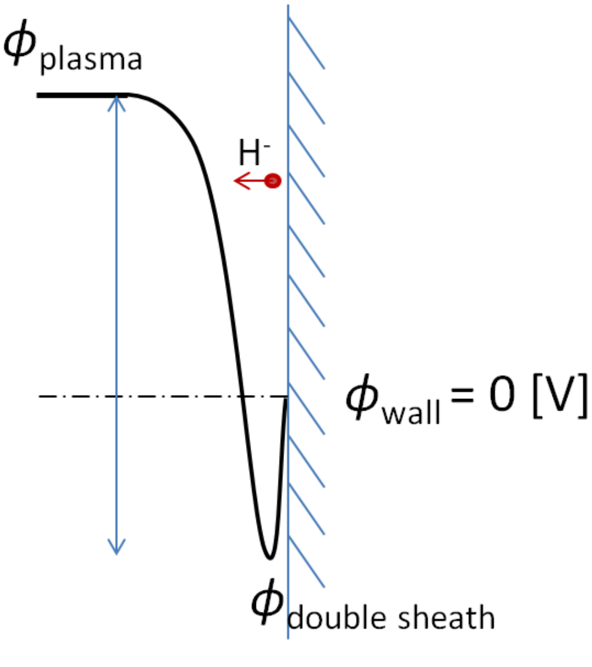
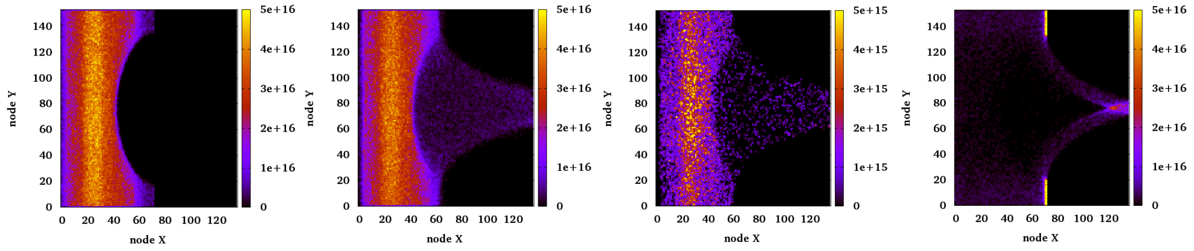


Figure 71: Schematic *double sheath* profile

*Double sheath* acquires its name from the fact that the potential profile obtains 0 [V] value twice. Plasma potential falls from a positive bulk value to zero (same as ordinary "single" sheath), but then crosses to the negative values and becomes 0 [V] again, when approaching grounded structure (metal). The negative part of potential profile is caused by high concentration of H$^-$ near the surface (Fig. 70). *Double sheath* has ambivalent effect on the surface-emitted H$^-$ ions. The accumulation of H$^-$ ions builds up a negative space charge, which gets resolved into $\phi_{\text{double sheath}}$ (Fig. 71). This potential pushes the newly emitted H$^-$ ions back to the wall and only those H$^-$ ions with sufficient energy escape the trap. Once escaped they are violently accelerated by the $\phi_{\text{plasma}} - \phi_{\text{double sheath}}$ towards the west wall of the simulation domain. These escaped H$^-$ ions are visible in Fig. 72d, 73d as constant density flux moving from the apperture to the west. This means, that fewer H$^-$ ions end up in the extracted beam, which is counter-productive. Advanced NBI studies have demonstrated, that a triangle-shaped profile of an apperture hole weakens the effect of the *double sheath* around hole edges and increased H$^-$ extraction.

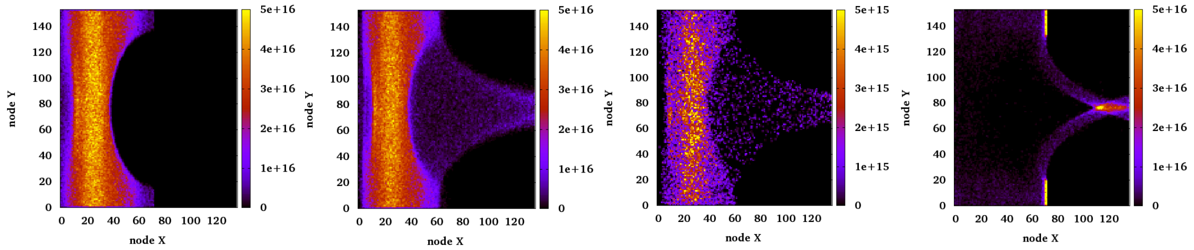
### 14.4.4 Meniscus formation and negative beam extraction

Under presense of extraction voltage plasma tries to screen itself, which leads to *meniscus* formation. Meniscus is the boundary, where the quasi-neutrality is broken in case of extraction voltage presense. Observation of meniscus formation is a key indication of the realistic simulation. Meniscus formation is associated with H$^+$ ions (Fig. 72a), while the electron beam is a concomitant effect on electron species (Fig. 72b). Meniscus shape determines the beam (electrons, H$^-$) characteristics as beam particles start accelerating at meniscus boundary. Particular interest is devoted to H$^-$ beam, which is decomposed into two components in this simulation: volume-produced H$^-$ beam (Fig. 72c) and surface-produced H$^-$ beam (Fig. 72d). Meniscus shape and beam characteristics are direct consequences of extraction voltage. Fig. 72 and Fig. 73 demonstrate two simulations, which differ only by extraction voltage: 5000 [V] and 7000 [V] respectively.



(a) H$^+$ ions (b) electrons (c) H$^-$ ions (volume) (d) H$^-$ ions (surface)

Figure 72: Density distribution at XY mid-plane at the end of NBI simulation (1 $\mu$s) under 5000 V of extraction voltage



(a) H$^+$ ions (b) electrons (c) H$^-$ ions (volume) (d) H$^-$ ions (surface)

Figure 73: Density distributions at XY mid-plane at the end of NBI simulation (1 $\mu$s) under 7000 V of extraction voltage

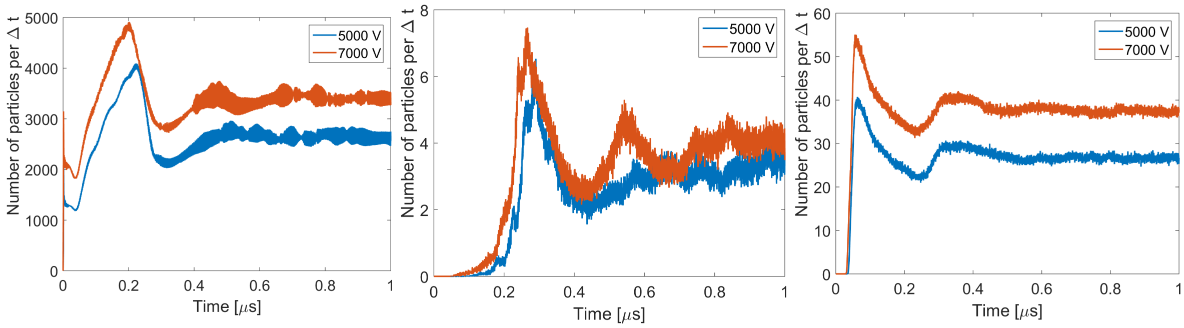
The temporal evolution of extracted particles are presented in Fig. 74. The number of extracted electrons correspond to 724.432 [A/m$^2$] and 947.335 [A/m$^2$] (Fig. 74a), volume H$^-$ ions to 0.836 [A/m$^2$] and 1.115 [A/m$^2$] (Fig. 74b), surface H$^-$ ions to 7.523 [A/m$^2$] and 10.588 [A/m$^2$] (Fig. 74c) of extracted current density at steady state respectively for 5000 [V] and 7000 [V] extraction voltage.

Important conclusions can be drawn by comparing the extracted current with back-scattered currents (Fig. 75). The number of extracted electrons correspond to 61.298 [A/m$^2$] and 55.726 [A/m$^2$] (Fig. 75a), volume H$^-$ ions to 0.0027 [A/m$^2$] and 0 [A/m$^2$] (Fig. 75b), surface H$^-$ ions to 15.325 [A/m$^2$] and 16.718 [A/m$^2$] (Fig. 75c) of back-scattered current density at steady state respectively for 5000 [V] and 7000 [V] extraction voltage.

### 14.4.5 Beam shape and extraction voltage scaling

Beam characteristics such beam width and shape are determined by the apperture shape and extraction voltage. In theory extraction voltage $V_{\text{ext}}$ is derived from *Child-Langmuir law* and in this case is defined as:

$$V_{\text{ext}} = \alpha \cdot \left( \frac{I_{\text{e}^-}}{P_{\text{e}^-}} + \frac{I_{\text{H}^-\text{volume}}}{P_{\text{H}^-\text{volume}}} + \frac{I_{\text{H}^-\text{surface}}}{P_{\text{H}^-\text{surface}}} \right)^{\frac{2}{3}}, \tag{89}$$

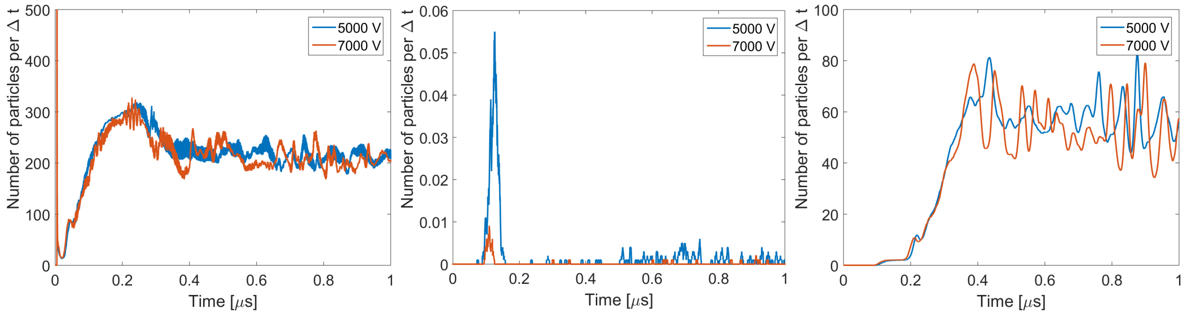


(a) electrons

(b) H$^-$ ions (volume)

(c) H$^-$ ions (surface)

Figure 74: Particles counts at the east wall (extracted particles)



(a) electrons

(b) H$^-$ ions (volume)

(c) H$^-$ ions (surface)

Figure 75: Particles counts at the west wall (back-scattered particles)

where $I_{\mathrm{e}^-}$ and $I_{\mathrm{H}^-}$ are beam currents, $P_{\mathrm{e}^-}$ and $P_{\mathrm{H}^-}$ are beam *perveances* of electrons and H$^-$ ions respectively, $\alpha$ is a scaling factor used to scale the experimental results to the theoretical limit. The value of *perveance* indicates, how significant the space charge effect is on the beam's motion and is defined as:

$$P = k\frac{A}{d^2}, \tag{90}$$

where $k$ is perveance constant, $A$ is apperture hole area and $d$ is averaged beam trajectory length. Apperture hole area $A$ can be obtained from Fig. 68, $d$ is obtained from Fig. 72 and Fig. 73, $k$ is 2.33395 $\mu$perv and 0.0544(5) $\mu$perv are perveances for for electrons and H$^\pm$ ions respectively [59]. Therefore Eq. 90 for 5000 [V] becomes:

$$\begin{aligned} P_{\mathrm{e}^-} &= 2.33395 \times 10^{-6} \cdot \tfrac{\pi \cdot (56\Delta x)^2}{(75\Delta x)^2} &\approx 4.1 \times 10^{-6} \text{ [perv]} \qquad (91)\\ P_{\mathrm{H}^-_{\mathrm{volume}}} = P_{\mathrm{H}^-_{\mathrm{surface}}} &= 54.45 \times 10^{-9} \cdot \tfrac{\pi \cdot (56\Delta x)^2}{(75\Delta x)^2} &\approx 9.5 \times 10^{-8} \text{ [perv]} \qquad (92)\end{aligned}$$

For 7000 [V] Eq. 90 becomes:

$$\begin{aligned} P_{\mathrm{e}^-} &= 2.33395 \times 10^{-6} \cdot \tfrac{\pi \cdot (56\Delta x)^2}{(80\Delta x)^2} &\approx 3.6 \times 10^{-6} \text{ [perv]} \qquad (93)\\ P_{\mathrm{H}^-_{\mathrm{volume}}} = P_{\mathrm{H}^-_{\mathrm{surface}}} &= 54.45 \times 10^{-9} \cdot \tfrac{\pi \cdot (56\Delta x)^2}{(80\Delta x)^2} &\approx 8.4 \times 10^{-8} \text{ [perv]} \qquad (94)\\ & & (95)\end{aligned}$$

$I_{\mathrm{e}^-}$, $I_{\mathrm{H}^-\mathrm{volume}}$ and $I_{\mathrm{H}^-\mathrm{surface}}$ are calculated by $I = \frac{MNq}{\Delta t}$, where $M = 3139.13$ is the macro-particle factor, $q$ is the elementary charge, $\Delta t = 5 \times 10^{-12}$ [s] and the number $N$ of macro-particles exiting the domain per $\Delta t$ is found from Fig. 74. For the case of 5000 V:

$$I_{\mathrm{e}^-} = \frac{2200 \cdot 3139.13 \cdot 1.602 \times 10^{-19}}{5 \times 10^{-12}} \approx 0.22\ [\mathrm{A}] \tag{96}$$

$$I_{\mathrm{H}^-_{\mathrm{volume}}} = \frac{3 \cdot 3139.13 \cdot 1.602 \times 10^{-19}}{5 \times 10^{-12}} \approx 0.0003\ [\mathrm{A}] \tag{97}$$

$$I_{\mathrm{H}^-_{\mathrm{surface}}} = \frac{27 \cdot 3139.13 \cdot 1.602 \times 10^{-19}}{5 \times 10^{-12}} \approx 0.0027\ [\mathrm{A}] \tag{98}$$

For the case of 7000 V:

$$I_{\mathrm{e}^-} = \frac{3400 \cdot 3139.13 \cdot 1.602 \times 10^{-19}}{5 \times 10^{-12}} \approx 0.34\ [\mathrm{A}] \tag{99}$$

$$I_{\mathrm{H}^-_{\mathrm{volume}}} = \frac{4 \cdot 3139.13 \cdot 1.602 \times 10^{-19}}{5 \times 10^{-12}} \approx 0.0004\ [\mathrm{A}] \tag{100}$$

$$I_{\mathrm{H}^-_{\mathrm{surface}}} = \frac{38 \cdot 3139.13 \cdot 1.602 \times 10^{-19}}{5 \times 10^{-12}} \approx 0.0038\ [\mathrm{A}] \tag{101}$$

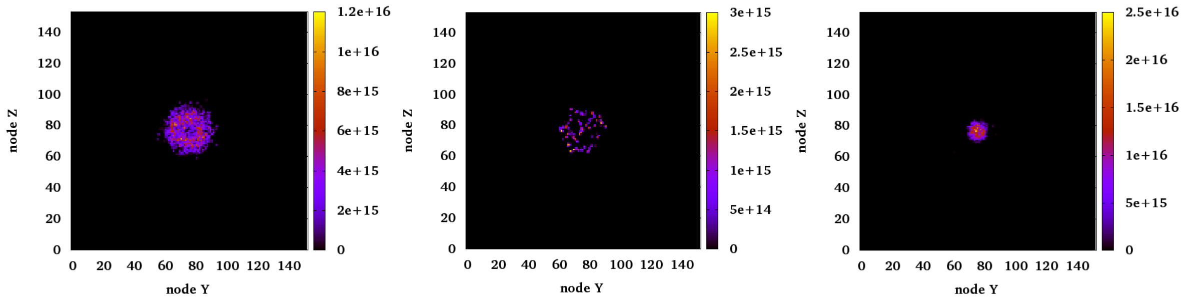
Now one calculates Eq. 89 from the given perveances and currents in cases of 5000 [V] and 7000 [V]:

$$V_{\mathrm{ext}} \approx \alpha \cdot 1945\ [\mathrm{V}]; \qquad V_{\mathrm{ext}} \approx \alpha \cdot 2767\ [\mathrm{V}] \tag{102}$$

Now one determines the scaling factors $\alpha$ by comparing Eq. 102 to the 5000 [V] and 7000 [V]:

$$\alpha = \frac{5000}{1945} \approx 2.57; \qquad \alpha = \frac{7000}{2767} \approx 2.53 \tag{103}$$

The scaling factor $\alpha$ obtains a similar values in both cases of $V_{\mathrm{ext}}$.

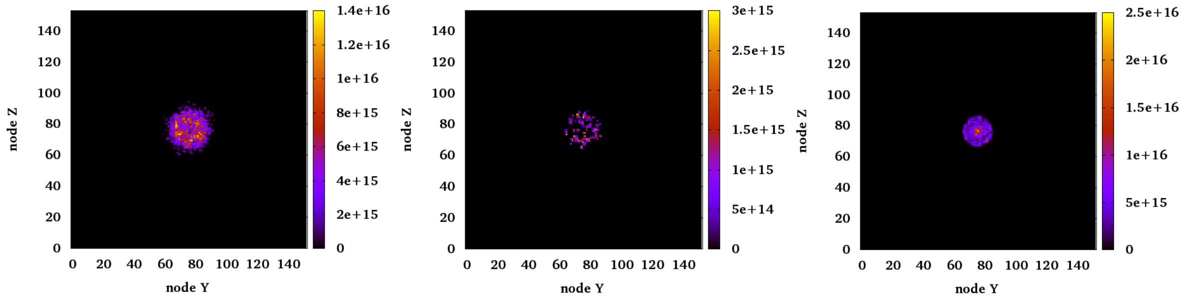


(a) electrons

(b) H⁻ ions (volume)

(c) H⁻ ions (surface)

Figure 76: Density distributions at YZ wall-plane (beam cross-section) under 5000 V of extraction voltage



(a) electrons

(b) H⁻ ions (volume)

(c) H⁻ ions (surface)

Figure 77: Density distributions at YZ wall-plane (beam cross-section) under 7000 V of extraction voltage

### 14.4.6 Discussion

- Two negative ion beam extraction simulations were produced of 5000 [V] and 7000 [V] extraction voltage respectively (Fig. 72,73) and evolved up to a steady state (Fig. 74, 75).

- *Double sheath* was produced by the simulation. Two potential profiles do not follow same shape in the bulk plasma (Fig. 69), because the plasma potential is oscillating, nevertheless plasma potential is fixed and stern in the *double sheath* region.

- The effect of higher extraction voltage is seen as a bigger depth of the meniscus (Fig. 72a, 73a).

- The effect of higher extraction voltage is visible as a higher $H^-$ beam convergence (Fig. 76c, 77c).

- The extracted currents of $H^-$ (7.523 [A/m$^2$] @ 5000 [V]), 10.588 [A/m$^2$] @ 7000 [V] Fig. 74c) are significantly lower than the $H^-$ surface production of 200 [A/m$^2$].

- The extracted currents of $H^-$ (7.523 [A/m$^2$] @ 5000 [V]), 10.588 [A/m$^2$] @ 7000 [V] Fig. 74c) are even lower than the $H^-$ back-scattered currents (15.325 [A/m$^2$] @ 5000 [V], 16.718 [A/m$^2$] @ 7000 [V]).

- Simulations follow a scaling law of extraction voltage, which is contained in the coefficient $\alpha \approx 2.5$ (Eq. 89, 103). It indicates that the simulations produce consistant results, which follow scaling laws. It is an important validation of the quality of NBI simulations produced by my code.

## 14.5 Simulation of Space Charge Compensation

High-energy (MeV) hydrogen neutral beams for fusion are produced by neutralizing a negative ion beam (Fig. 5). Negative ions are generated in a plasma sources, extracted and accelerated by a MAMuG (**M**ulti-**A**perture **Mu**lti-**G**rid) section (Fig. 78). Then the beam exits MAMuG section and enters the so-called *drift* region, where it meets the ambient $H_2$ gas. In this region the negative ions experience collisions with ambient gas $H_2$ and thus produces positive ions $H^+$. The total charge density of the drift region from being negative (due to presence $H^-$) starts to continuously approach zero (effect of collisions between $H^-$ and $H_2$ neutrals). This process is called SCC (**S**pace **C**harge **C**ompensation). Due to significant change in charge density during transient phase the beam optics is heavily influenced, e. i. the diverging stream of negative ions $H^-$ focuses itself reaching divergence $\approx 0$ at the steady state.

### 14.5.1 Model description

The geometry of a drift stage is approximated to a 2D domain with a particular setup of Dirichlet and Neumann conditions represeting functional parts of the system. There are 3 functional parts of the domain: domain boundary, aperture, slits and reppelers (Fig. 79). Domain boundary represents the walls of the vacuum vessel that drift region is located in and it is grounded to $\phi = 0$ [V] (except slits). In 3D the aperture is a planar piece with circular holes. In a projected 2D cross-cut the aperture and holes are collection of material chunks and vacant slits, where aperture is grounded to $\phi = 0$ [V] and slits satisfy Neumann condition at the domain boundary ($E_\perp = 0$ [V/m]). According to NIO1 (**N**egative **I**on **O**ptimization 1) design there are 3 slits (3×3 holes) (Fig. 78). Repellers are another structure following slit (hole) geometry. They have purpose to reppel unwanted positive ions (product of collissions $H^-$ and neutrals $H_2$) not to travel back to the slits and can be set to arbitrary $\phi = \phi_R$ value. The simulated drift region takes up an area of size $440 \times 52.8$ [mm$^2$]. The $H^-$ beamlets enter through the slits on the west boundary having energy of 60 keV obtained at acceleration stage. During the drift $H^-$ beamlets interact with $H_2$ neutrals via Null Collision method (13.5.2), which gradually lead to SCC and self-focusing. Parameters of the simulation were chosen according to a plasma of density $n_e = 10^{15}$ [m$^{-3}$] at 10 [eV], i.e. time step $\Delta t = 0.2$ [ps] $< \omega_{pe}^{-1} \approx 0.56$ [ns] and cell size $\Delta x = 0.4$ [mm] $< \lambda_D \approx 0.743$ [mm].

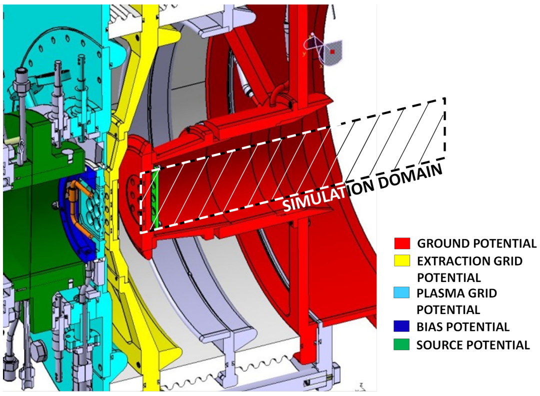


Figure 78: NIO1 drift stage

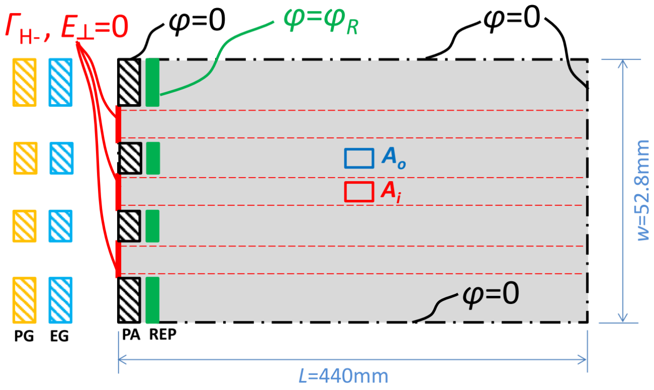


Figure 79: Simulation domain (domain boundary, aperture, slits and reppelers) and its corresponding Dirichlet/Neumann conditions

### 14.5.2 Implementation of collisions

Initially simuation contains only background $H_2$ neutrals and beam ions $H^-$. Gradualy collision processes (Fig. 80) built up $e$, $H^-$, $H^+$ and $H_2^+$, which move in a bath of $H_2$ of uniform density $n = 1.16\times10^{19}$ [$m^{-3}$]. There are 2 families of collisions implemented: elastic and inelastic. Elastic collisions include processes such as:

- Electrons elastically collide with $H_2$, where velocity distribution is modelled according to [52].

Inelasic collisions include processes such as:

- A fast projectile ($H^{-1}$, $H^0$, $H^{+1}$) causes $H_2$ ionization, where trajectory of generated free electron is determined by differential cross section $\sigma(W)$ [60] and scattering angles $\chi(W)$ [61].
- A fast projectile ($H^{-1}$, $H^0$, $H^{+1}$) undergoes electron detachment (single, double (stripping)), whereafter trajectories of projectile and stripped electron are kept undisturbed, but charge of the projectile is changed.
- A fast electron causes $H_2$ ionization, trajectories of generated electrons is determined by differential cross sections $\sigma(W)$ [62] and scattering angles $\chi(W)$
- A fast projectile ($H^0$, $H^{+1}$) causes charge exchange with $H_2$, where $H_2^+$ is generated with an isotropic initial velocity corresponding to a temperature of 0.03 eV.

Energy and angle distributions are sampled in the forms respectively:

$$W(E,R) = \frac{a_0(E)}{a_1(E) - R} + a_2(E) \tag{104}$$

$$\chi(E,R) = \frac{b_0(E)}{b_1 - (b_1 R)^{b_2}} + b_3 \tag{105}$$

where $R$ is a random number in the interval $[0,1]$ and $E$ is the energy of the $\underline{H}$ projectile, $W(E,R)$ is sampled-fitted energy, $a_0, a_1, a_2, b_0, b_1, b_2$ are fitting parameters. In order to sample any secondary electron, ones uses a 2D inverse cumulative distribution $W(E,R)$ of energies.

$$W(R,E) = \frac{10^{\sum_{i=0}^{2} a_{1i}(log_{10}E)^i}}{10^{\sum_{i=0}^{2} a_{2i}(log_{10}E)^i} - R\cdot 10^{\sum_{i=0}^{6} c_i(\log_{10}E)^i}} - \sum_{i=0}^{2} a_{1i}(\log_{10}E)^i \tag{106}$$

The angular distribution $\chi(W)$ of inelastically scattered electrons follows Eq. 105.

| | | | |
|---|---|---|---|
| $H_2 + e$ | $\longrightarrow$ | $H_2 + e$ | elastic scattering |
| $H_2 + e$ | $\longrightarrow$ | $H_2^+ + e + e$ | ionization |
| $H_2 + H$ | $\longrightarrow$ | $H_2^+ + H^-$ | charge exchange |
| $H_2 + H$ | $\longrightarrow$ | $H_2 + H^+ + e$ | electron detachment |
| $H_2 + H$ | $\longrightarrow$ | $H_2^+ + H + e$ | ionization |
| $H_2 + H^-$ | $\longrightarrow$ | $H_2 + H^+ 2e$ | double electron detachment |
| $H_2 + H^-$ | $\longrightarrow$ | $H_2 + H + e$ | electron detachment |
| $H_2 + H^-$ | $\longrightarrow$ | $H_2^2 + H^- + e$ | ionization |
| $H_2 + H^+$ | $\longrightarrow$ | $H_2^+ + H$ | charge exchange |
| $H_2 + H^+$ | $\longrightarrow$ | $H_2^+ + H^+ + e$ | ionization |

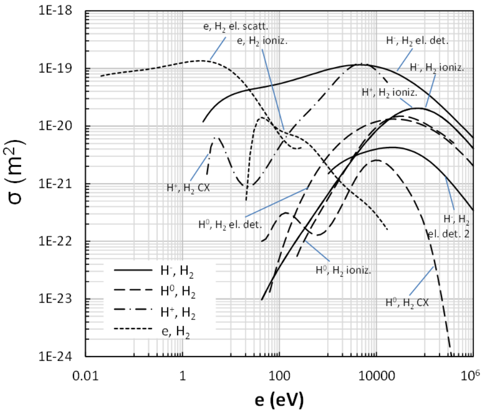


Figure 80: Collisional processes included in the model [63]

### 14.5.3 Simulation results and discussion

SCC simulation was run for two cases of repeller voltages $\phi_R = 0$ [V] and $\phi_R = 100$ [V]. The total simulated time comprised 15 $\mu$s, where plasma formation and evolution was observed. Fig. 81 shows the H− density map at four time instances for the case $\phi_R = 0$ [V].

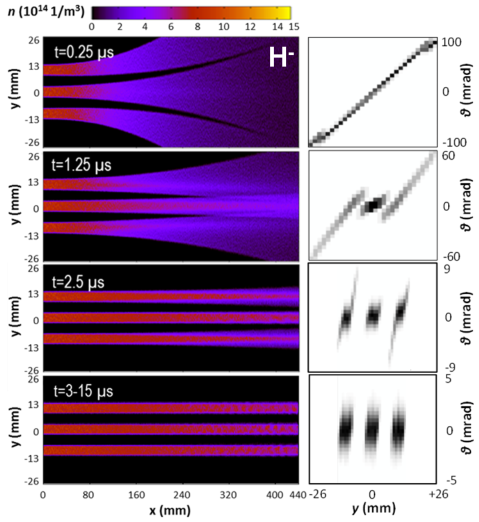


Figure 81: H density map at different different time instances (left) and emittance plot at 410 mm from the repeller (right)

It is possible to see that compensation of the central beamlet occurs earlier with respect to the peripheral beamlets. All beamlets reach a steady compensation at $t \approx 3$ [$\mu$s]. This number is in a good agreement with the theoretical expectations for the SCC time $\tau = \frac{1}{n\sigma_{\text{ioniz}} v_{\text{beam}}} \approx 2.5$ [$\mu$s], where $\sigma_{\text{ioniz}}$ is ionization cross-section. Density profiles for the five species ($H^-$, $H^0$, $H^+$, $H_2^+$, e), taken along $y$ at a distance of about 1.7 cm from the repeller (Fig. 82).

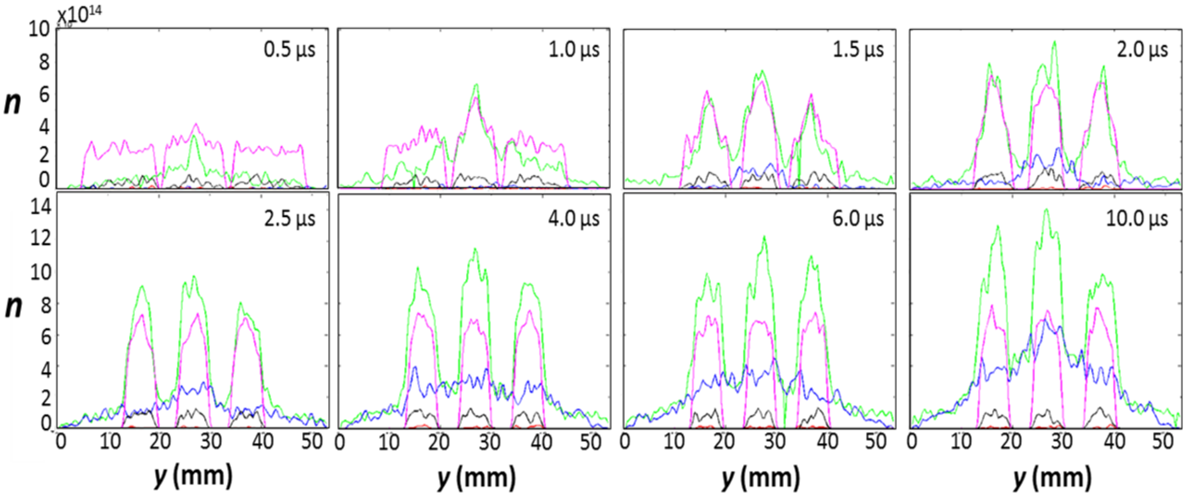


Figure 82: Vertical profile of particle densities at $x = 200$ mm. Beam ions, molecular ions and electrons are indicated in the last frame

Two phases are recognizable. The first phase ($< 2$ $\mu$s) is controlled by the beam space charge and consists of $H_2^+$ accumulating in the potential wells of the beamlets until space charge overcompensation is reached. In a second phase electrons start to accumulate as well to compensate the presence of slow molecular ions, hence Maxwellian plasma is building up in the background of the beams (Fig. 82). At equilibrium the $H_2^+$ density balances the sum of the imposed $H^-$ density (primary species) and the electron density (secondary species), which by diffusion fills also the space between beamlets. The velocity distribution of intra-beamlet and inter-beamlet plasma particles in respective regions $A_{\mathrm{in}}$ and $A_{\mathrm{out}}$ (Fig. 79) are exposed in Fig. 83.

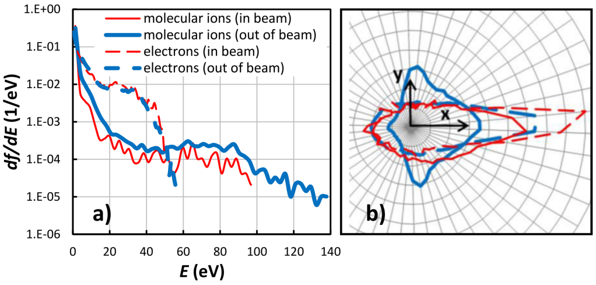


Figure 83: Normalized energy distribution (a) and angular distribution (b) for ions (solid) and electrons (dashed) acquired within the beamlet (thin) and in-between beamlets (thick)

Ion and electron energy distribution are very similar in the two regions (Fig. 83a). The average ion energies are $E_{i,in} = 1.3$ eV and $E_{i,out} = 2.2$ eV, while average electron energies are $E_{e,in} = 9$ eV and $E_{e,out} = 8$ eV. Electrons have more-or-less the same angular distribution in two regions. Ions have different angular distributions in two regions (Fig. 83b), which means they were transversally expelled from the beamlet region.

A self-potential-compensation parametrization of the beam $\psi$ can be defined from the potential $U$ at a given time and the initial potential $U_0$ generated by the beam charges at very beginning of the simulation ($t = 0.14$ $\mu$s):

$$\phi(s) = 1 - \frac{U(s)}{U_0(s)}, \tag{107}$$

where $s$ is a path along the domain. $\psi$ is very useful to compare the extend of self-compension in the beams of the two cases of interest. $\psi(y)$ reaches 1.08 and 1.02 for two cases $\phi_{\mathrm{R}} = 100$ [V] and $\phi_{\mathrm{R}} = 0$ [V] respectively (Fig. 84). A similar result is demonstrated by $\psi(x)$ for the case $\phi_{\mathrm{R}} = 0$ [V] at the four time instances (Fig. 85).

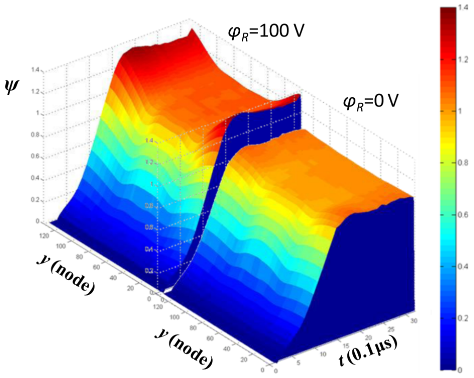


Figure 84: Temporal evolution of compensation parameter $\psi(y)$ at $x = 8$ mm.

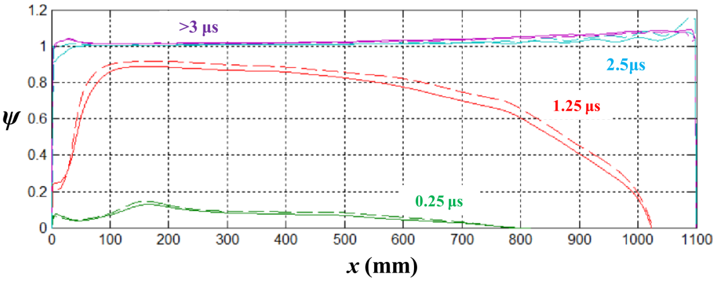


Figure 85: Compensation parameter $\psi(x)$ along the central (dashed) and lateral beamlets (solid) at different time instances (case $\phi_{\mathrm{R}} = 0$ [V])

# 15 Conclusions

PIC method provides a lot of opportunity and flexibility for plasma simulations in both collisionless and collisional approaches. PIC algorithm posses a straight-forward and robust nature towards the simulation, which allows codes to be built from scratch independently (as it was done for this thesis) and custom-optimize for a particular computational load. However GPU-based implementation of PIC introduces a higher level of complexity into the coding and optimization. At the same time GPGPU platform provides additional opportunities for the high-end codes. Since GPGPUs are designed and oriented to massive parallel computation with extremely high throughputs, PIC method can reach unseen levels of quality. The code is able to fully and completely exploit GPGPU power, it was tested and works well in a wide range of parametric space. Performance-wise there are two parametrical constituents of the PIC code setup: the domain size and number of particles. These two code parts map well onto two functional parts of GPU harware: GPU RAM and GPU multiprocessor. Nvidia Tesla K40 was used for running PIC code. Tesla K40 is one of the latest Nvidia products for HPC built on Kepler architecture and released in 2014. It runs on 745 MHz clock with 12 GB of RAM and 48 kB of shared memory. The code was able to operate in 3D domains of size up to $137 \times 153 \times 153$ nodes and evolve up to $10^8$ particles within reasonable duration (it depends on the length of simulated time). In fact the code by design can operate on even bigger setups, but in this case it was limited by 12 GB of RAM of Nvidia Tesla K40. In a sense GPU-based implementation is essential for high-end PIC codes, because any CPU-based implementation would suffer from lack of available parallelization and too high computational loads. And above all the code seems to procude valid physical results according to current knowledge of plasma physics.

# Part IV
# Tomography

The word *tomography* is derived from ancient greek words τομὸς (*tomos* - slice, section) and γρὰφω (*graphō* - to write). Tomography refers to imaging by sections or sectioning, through the use of any kind of penetrating rays. The method exists in radiology, archaeology, biology, atmospheric science, geophysics, oceanography, plasma physics, materials science, astrophysics, quantum information, and other sciences. Mainly it is based on the mathematical procedure called *tomographic reconstruction.*

## 16 Tomographic problem

### 16.1 Radon transform

In 1917 Austrian mathematician Johann Radon (1887-1956) introduced an integral 2D function transform (presently known as *Radon transform*)[64]. The transform is constructed by relating a 2D function $g(x, y)$ to its *line integrals* (projections). *Radon transform* takes multiple 1D projections and assembles them into a representation called *sinogram.* Using operator $\mathcal{R}$ Radon tranform can be expressed as:

$$\mathcal{R}g(x, y) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} g(x, y)\delta(r - x\cos\theta - y\sin\theta)\,\mathrm{d}x\,\mathrm{d}y \tag{108}$$

With the help of vector notation one can extend definition to a multidimensional case:

$$\mathcal{R}g(\mathbf{x}) = \int g(\mathbf{x})\delta(r - \mathbf{x}\cdot\mathbf{t})\,\mathrm{d}\mathbf{x}, \tag{109}$$

where $\mathbf{t}$ is a unit vector in the scanning direction $\theta$ and $L(r, \theta)$ is the integration line lying at a distance $r$ from the origin (Fig. 86). In fact word "*sinogram*" derives from "*sine*", because projections tend to draw sinusoids in the *sinogram* (Fig. 87b, 87d).

### 16.2 Rotation theorem

*Rotation theorem* needs to be stated before one moves further. This theorem is necessary to prove *Projection-slice theorem* (16.4). *Rotation theorem* deals with the question, whether an image $g(x, y)$ and its Fourier tranform $G(u, v)$ follow an arbitrary rotation accordingly.

**Rotation theorem.**
If $g(x, y)^2 \supset G(u, v)$, then

$$g(x\cos\theta - y\sin\theta, x\cos\theta + y\sin\theta)^2 \supset G(u\cos\theta - v\sin\theta, u\sin\theta + v\sin\theta) \tag{110}$$

Translated to human language, the theorem says that if one rotates an image $g(x, y)$ by an angle $\theta$, its Fourier representation $G(u, v)$ will rotate by the same angle $\theta$. Even though it may seem to be obvious, but one has to ensure that a clock-wise rotation of $g(x, y)$ does not result in a counter-clockwise rotation of $G(u, v)$ and vice versa. In order not to dwell too much on mathematical formalities, we are going to skip the proof this theorem.

### 16.3 Projection definition

Projection operation $\mathcal{P}_\theta$ of a 2D function $g(x, y)$ is defined as its *line integral.* There exist infinitely many of such *line integrals* each representing a particular direction $\theta$. Thus one can define a zero-angle and arbitary angle projections $P_0$ and $P_\theta$ respectively:

$$P_0(x) = \mathcal{P}_0 g(x, y) = \int_{-\infty}^{\infty} g(x, y)\,\mathrm{d}y \tag{111}$$

$$P_\theta(r) = \mathcal{P}_\theta g(x,y) = \int\limits_{L(r,\theta)} g(x,y)\,\mathrm{d}L, \tag{112}$$

where $L(r,\theta)$ is the path (a line) of a *line integral* (Fig. 86). A set or a collection of these projections $\{P_{\theta_i} \dots P_{\theta_n}\}$ is called a *scan* or *sinogram* in tomography, where $P_{\theta_i}$'s are acquired by a function of time or/and by an array of cameras.

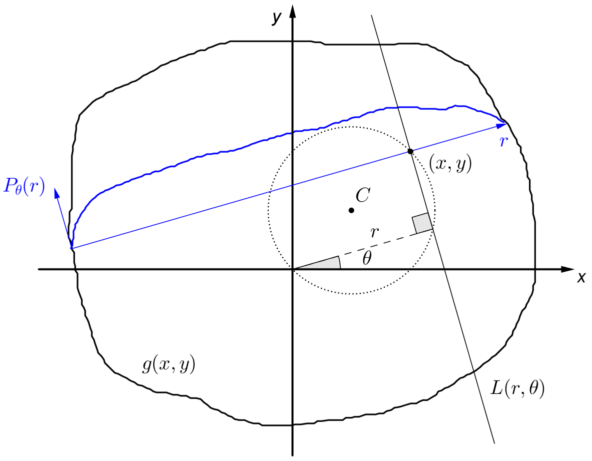


Figure 86: Geometry of Radon tranform

## 16.4 Projection-slice theorem

Fourier transform of a projection of an image $g(x,y)$ taken at an angle $\theta$ gives a slice of the 2D Fourier transform $G(u,v)$, subtending an angle $\theta$ with the $u$-axis.

*Proof.*
One starts with the 2D Fourier transform relation:

$$G(u,v) = \int\limits_{-\infty}^{\infty}\int\limits_{-\infty}^{\infty} g(x,y)e^{-i2\pi[ux+vy]}\,\mathrm{d}x\,\mathrm{d}y \tag{113}$$

One sets $v = 0$, because it is the slice along $v = 0$ that it is desired to be obtained:

$$\int\limits_{-\infty}^{\infty}\left[\int\limits_{-\infty}^{\infty} g(x,y)\,\mathrm{d}y\right] e^{-i2\pi\, ux}\,\mathrm{d}x = G(u,0) \tag{114}$$

One sees that the integral in brackets is exactly $P_0(x) = \mathcal{P}_0 g(x,y)$ (Eq. 111). In terms of the projection operator $\mathcal{P}_0$ and Fourier operator $\mathcal{F}$ the previous equation (Eq. 114) can be written as:

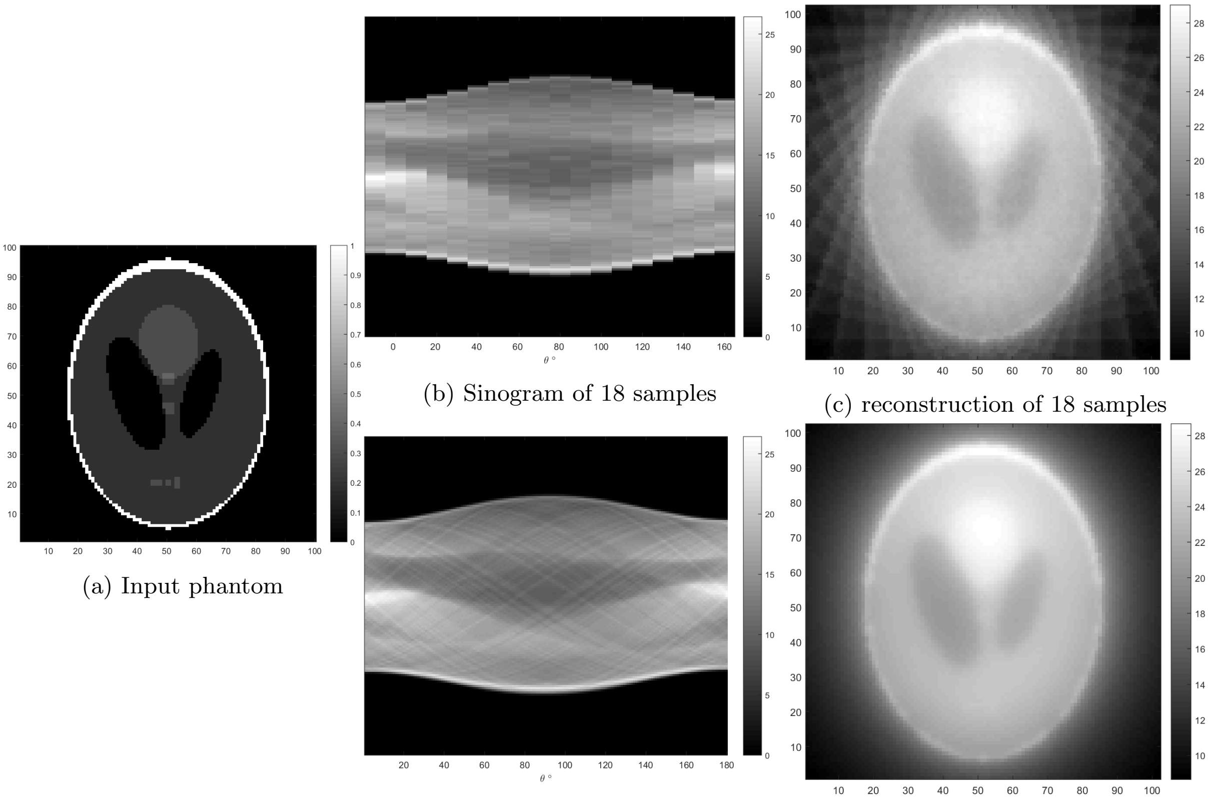
$$^1\mathcal{F}\{\mathcal{P}_0 g(x,y)\} = [^2\mathcal{F}g(x,y)]_{\theta=0} \tag{115}$$

The left side is a 1D Fourier transform of the projection at angle $\theta = 0$. The right side is the slice of 2D Fourier transform subtending angle $\theta = 0$ with the $u$-axis. And they are equivalent. This completes a partial proof for the special case $\theta = 0$. △
Knowing that *Rotation theorem* (16.2) holds true one can extend Eq. 115 to a general case for any angle $\theta$:

$$^1\mathcal{F}\{\mathcal{P}_\theta g(x,y)\} = [^2\mathcal{F}g(x,y)]_\theta \tag{116}$$

$$^1\mathcal{F}P_\theta(x) = [G(u,v)]_\theta \tag{117}$$

This completes the proof for any angle $\theta$. □



(a) Input phantom

(b) Sinogram of 18 samples

(c) reconstruction of 18 samples

(d) Sinogram of 180 samples

(e) reconstruction of 180 samples

Figure 87: *Radon transform* and its inverse (reconstruction)

*Projection-slice* theorem tells that *Radon transformation* is uniquely reversible given an infinite number of projections, i.e. *inverse Radon transform* exists. In practice one never has infinite number of projections, thus reconstruction is never perfect. And *inverse Radon transform* proves to be extremely sensitive to noise. In Fig. 87c one can clearly see projection artefacts, which are caused by low samping (low number of projections). In Fig. 87e high density of projection artefacts (high sampling) results into a halo effect and overall whitening of the image.

# 17 Cormack solution

A cross-section of a physical object, that is undergoing tomographic inspection, is characterized by a linear absorption (attenuation) coefficient $g$ represented in 2D domain. It is assumed that attenuation occurs exponentially in the object and along the inspection line $L$. The relation of an input intensity $I_0$ and output intensity $I$ of beam passing through media can be expressed as:

$$I = I_0 \exp\left(-\int\limits_L g(x,y)\,\mathrm{d}L\right), \tag{118}$$

where $g(x, y)$ is the map of attenuation values and $L$ is the path (assumed line) that the beam took. The integral inside the exponent is a projection of $g(x, y)$ along $L$ (Fig. 88). One defines this projection value as:

$$f_L = \int\limits_L g(x,y)\,\mathrm{d}L = \ln\frac{I_0}{I} \tag{119}$$

One can extend this relation into a function of variables $(p, \phi)$:

$$f(p, \phi) = \int\limits_{L(p,\phi)} g(x, y)\, \mathrm{d}L \tag{120}$$

And by changing from Cartesian to polar coordinates, one gets:

$$f(p, \phi) = \int\limits_{L(p,\phi)} g(r, \theta)\, \mathrm{d}L \tag{121}$$

In fact $f(p, \phi)$ is the *Radon transform* of $g(r, \theta)$, i.e. $f(p, \phi) = \mathcal{R}g(r, \theta)$. If one could obtain $g(r, \theta) = \mathcal{R}^{-1}f(p, \phi)$ easily, the "Quest of Tomography" would be finished. However only approximate solutions exist. The papers [65, 66] on mathematical treatment of *inverse Radon transform* earned Nobel Prize to Allan McLeod Cormack in 1979. He shares the prize in the category of "Physiology or Medicine" with Godfrey Hounsfield, who has built the first Computerized Tomography scanner using Cormack's calculations.

**Derivation**
Hereafter the mathematical passage will be described, how Cormack reached his solution. In order to tackle the aforementioned problem, first one has to convert $f(p, \phi)$ into a set of integrals $f_n(p, \phi)$. One supposes that $g(r, \theta)$ is finite, single-valued and continuous (well-behaved) in $r$ direction, but not in $\theta$. It is also contained in a unitary circle, because one can always normalize by $r$. Then expansion of $g(r, \theta)$ into harmonic series (Fourier series) with respect to variable $\theta$ is:

$$g(r, \theta) = \sum_{n=-\infty}^{+\infty} g_n(r) e^{in\theta}, \text{ where } \quad g_n(r) = \frac{1}{2\pi} \int\limits_0^{2\pi} g(r, \theta) e^{-in\theta}\, \mathrm{d}\theta \tag{122}$$

The arc created by $L(p, \phi)$ and circle of radius $r$ can always be divided into to two equal arcs by a normal from $L(p, \phi)$ to the origin $O$ (Fig. 88). Pairs of symetric points ($(r, \theta)$ and $(r, 2\phi - \theta)$) have same contributions to $f$, then $\mathrm{d}f$ can be written as:

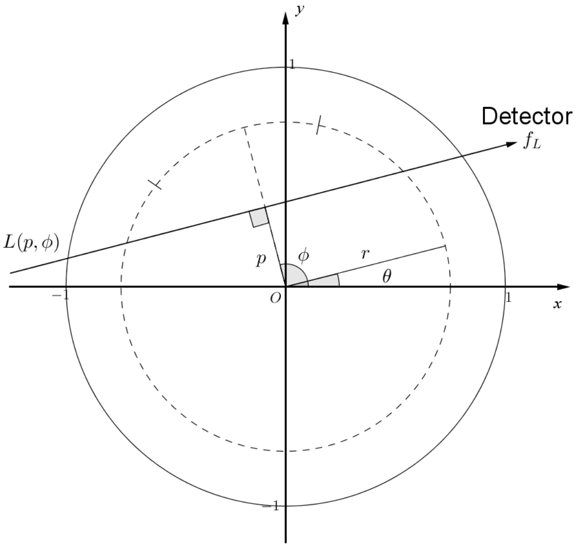


Figure 88: Geometry of Cormack's solution

$$\mathrm{d}f = \sum_{n=-\infty}^{+\infty} g_n(r)[e^{in\theta} + e^{in[2\phi-\theta]}]\, \mathrm{d}L \tag{123}$$

Simplyfied:

$$\mathrm{d}f = 2 \sum_{n=-\infty}^{+\infty} g_n(r) e^{in\phi} \cos(n[\theta - \phi])\,\mathrm{d}L \tag{124}$$

Combining Eq. 121 and Eq. 124 one gets:

$$f(p,\phi) = 2 \int\limits_{L(p,\phi)} \sum_{n=-\infty}^{+\infty} g_n(r) e^{in\phi} \cos(n[\theta - \phi])\,\mathrm{d}L \tag{125}$$

Knowing that $L = \sqrt{r^2 - p^2} \longrightarrow \mathrm{d}L = \frac{r\,\mathrm{d}r}{\sqrt{r^2-p^2}}$ and $\theta - \phi = \arccos(\frac{p}{r})$, Eq. 125 becomes:

$$f(p,\phi) = 2 \sum_{n=-\infty}^{+\infty} e^{in\phi} \int\limits_{p}^{1} \frac{g_n(r)\cos(n \arccos\frac{p}{r})}{\sqrt{r^2 - p^2}}\, r\mathrm{d}r \tag{126}$$

By looking accurately, one can notice that Eq. 126 became a sum of integrals (what was desired). Indeed when one applies Fourier expansion for $f(p,\phi)$ (the same as it was done for $g(r,\theta)$ (Eq. 122)), the integral in the sum (Eq. 126) are the Fourier coeficients $f_n$ (Eq. 127):

$$f(p,\phi) = \sum_{n=-\infty}^{+\infty} f_n(p) e^{in\phi}, \text{ where } \quad f_n(p) = \frac{1}{2\pi} \int\limits_{0}^{2\pi} f(p,\phi) e^{-in\phi}\,\mathrm{d}\phi \tag{127}$$

becomes

$$f(p,\phi) = \sum_{n=-\infty}^{+\infty} f_n(p) e^{in\phi}, \text{ where } \quad f_n(p) = 2 \int\limits_{p}^{1} \frac{g_n(r)\cos(n \arccos\frac{p}{r})}{\sqrt{r^2 - p^2}}\, r\mathrm{d}r \tag{128}$$

The construct "$\cos(n \arccos(x))$" (Eq. 128) is a polynomial of degree $n$ known as Tschebycheff polynomial of the first kind. It is denoted by $T_n(x)$, and Eq. 128 can be written as:

$$f_n(p) = 2 \int\limits_{p}^{1} \frac{g_n(r) T_n(\frac{p}{r})}{\sqrt{r^2 - p^2}}\, r\mathrm{d}r \tag{129}$$

This completes the derivation. △

**Solution**

To solve the set of equations (Eq. 129), multiply both sides by $\frac{\frac{z}{p} T_n(\frac{p}{z})}{\sqrt{p^2 - z^2}}$, integrate $\int\limits_{z}^{1} \ldots\,\mathrm{d}p$ and change the order of integrals on the right hand side:

$$\int\limits_{z}^{1} \frac{\frac{z}{p} T_n(\frac{p}{z}) f_n(p)}{\sqrt{p^2 - z^2}}\,\mathrm{d}p = 2 \int\limits_{z}^{1} g_n(r)\,\mathrm{d}r \int\limits_{z}^{r} \frac{rz T_n(\frac{p}{z}) T_n(\frac{p}{r})}{p\sqrt{r^2 - p^2}\sqrt{r^2 - z^2}}\,\mathrm{d}p \tag{130}$$

Let one denote the right hand integral of d$p$ by $I_n(r,z)$:

$$I_n(r,z) = rz \int\limits_{z}^{r} \frac{T_n(\frac{p}{z}) T_n(\frac{p}{r})}{p\sqrt{r^2 - p^2}\sqrt{r^2 - z^2}}\,\mathrm{d}p \tag{131}$$

It can be shown that $I_{n+1} = I_{n-1}$ and $I_0 = I_1 = \frac{\pi}{2}$, so

$$I_n(r,z) = \frac{\pi}{2} \tag{132}$$

Hence Eq. 130 becomes

$$\int_z^1 \frac{\frac{z}{p}T_n(\frac{p}{z})f_n(p)}{\sqrt{p^2-z^2}}\,\mathrm{d}p = 2I_n(r,z)\int_z^1 g_n(r)\,\mathrm{d}r = \pi\int_z^1 g_n(r)\,\mathrm{d}r \tag{133}$$

By differentiating with respect to $z$ one gets:

$$g_n(r) = -\frac{1}{\pi}\frac{\mathrm{d}}{\mathrm{d}z}\int_z^1 \frac{zT_n(\frac{p}{z})f_n(p)}{p\sqrt{p^2-z^2}}\,\mathrm{d}p \tag{134}$$

Changing notation $z \to r$:

$$g_n(r) = -\frac{1}{\pi}\frac{\mathrm{d}}{\mathrm{d}r}\int_r^1 \frac{rT_n(\frac{p}{r})f_n(p)}{p\sqrt{p^2-r^2}}\,\mathrm{d}p \tag{135}$$

This completes the solution of Eq. 129. △

Finally mutually defined correspondence between $f_n(p)$ and $g_n(r)$ (Eq. 129 and Eq. 135) is obtained, i.e. $f_n(p)$ and $g_n(r)$ are unique solutions of each other.

# 18 Fourier-Bessel expansion

At this point reconstruction can be already obtained by calculcating directly $g_n(r)$ (Eq. 135), where values of $f_n(p)$ are experimentally obtained from the detector array. However with regard to computational efficiency an integration is a lengthy and expensive mathematical procedure, which means it is completely unsuitable for real-time applications. Thus one need to simplify the integration to something more primitive. One can assume $f_n(p)$ to be composite of a some set of orthogonal functions, i.e. *basis functions.* In the original paper [66] Cormack proposed to assign $f_n(p)$ with Zernicke polynomials $R_n^m$, because they lead to particularly simple expansions for the $f_n$. Cormack was not particularly concerned about plasma tomography, therefore Zernicke polynomials worked well for him. Indeed Zernicke polynomials proved to be not useful in plasma tomography [67], because they tend to generate spike artifacts on the edge of $g(r,\theta)$, which arise from the fact that $R_n^m(1) = 1$. From basic plasma physics it is known, that plasma density approaches 0 at normalized minor radius $r = 1$. Fortunately there exists another complete set of orthogonal functions - Bessel functions of the first kind $J_n$. They have the desired behaviour when approaching plasma edge, i.e. $J_n(x_{nl}) = 0$. This expansion was first proposed by Ling Wang and Robert S. Granetz in [68, 69]. In Eq. 129 one has to replace $g_n(r)$ with a Bessel function:

$$f_n(p) = 2\int_p^1 \frac{g_n(r)T_n(\frac{p}{r})}{\sqrt{r^2-p^2}}\,r\mathrm{d}r \longrightarrow 2\int_p^1 \frac{J_n(x_{nl}r)T_n(\frac{p}{r})}{\sqrt{r^2-p^2}}\,r\mathrm{d}r, \tag{136}$$

where $x_{nl}$ is the $(l+1)$'th root of Bessel function $J_n$ of order $n$, i.e. $x_{nl}$ is defined as $J_n(x_{nl}) = 0$. In order to simplify Eq. 136 one has to reduce the integral term and overcome the singularity at $r = p$.

**Derivation**
Fourier expansions of $g(r,\theta)$ and $f(p,\phi)$:

$$g(r,\theta) = \sum_{n=-\infty}^{+\infty} g_n(r)e^{in\theta} \quad = \sum_{n=0}^{\infty} g_n^{\mathrm{c}}(r)\cos(n\theta) + g_n^{\mathrm{s}}(r)\sin(n\theta), \text{ where } \quad g_n^{\mathrm{c,s}}(r) = \sum_{l=0}^{\infty} a_{nl}^{\mathrm{c,s}} g_{nl}(r) \tag{137}$$

$$f(p,\phi) = \sum_{n=-\infty}^{+\infty} f_n(p)e^{in\phi} \quad = \sum_{n=0}^{\infty} f_n^{\mathrm{c}}(p)\cos(n\phi) + f_n^{\mathrm{s}}(p)\sin(n\phi), \text{ where } \quad f_n^{\mathrm{c,s}}(p) = \sum_{l=0}^{\infty} a_{nl}^{\mathrm{c,s}} f_{nl}(p) \tag{138}$$

2D Fourier transform of $g(x,y)$:

$$g(\mathbf{k}) = \int\int g(x,y)e^{i\mathbf{k}\cdot\mathbf{r}}\,\mathrm{d}x\,\mathrm{d}y \tag{139}$$

Rotating the coordinate axis so that $\mathbf{p} \parallel \mathbf{k}$, Eq. 139 becomes:

$$g(\mathbf{k}) = \int\int g(x,y)e^{ikr\cos(\phi-\theta)}\,\mathrm{d}x\,\mathrm{d}y = \int\limits_0^{2\pi}\mathrm{d}\theta\int\limits_0^{\infty} g(r,\theta)e^{ikr\cos(\phi-\theta)}r\,\mathrm{d}r \tag{140}$$

By inserting Eq. 137 into Eq. 140 and carrying out the integration over $\theta$ {Appendix A}:

$$g(\mathbf{k}) = \sum_{n=0}^{\infty}\int\limits_0^{\infty}[g_n^{\mathrm{c}}(r)\cos(n\theta) + g_n^{\mathrm{s}}(r)\sin(n\theta)]i^n 2\pi J_n(kr)r\,\mathrm{d}r \tag{141}$$

The inverse Fourier transform of Eq. 141 then gives:

$$\begin{aligned}
f(p,\phi) &= \frac{1}{2\pi}\int\limits_{-\infty}^{\infty} g(\mathbf{k})e^{-ikp}\,\mathrm{d}k = \\
&= \sum_{n=0}^{\infty} i^n\left[\cos(n\phi)\int\limits_0^{\infty} g_n^{\mathrm{c}}(r)r\,\mathrm{d}r\int\limits_{-\infty}^{\infty} J_n(kr)e^{-ikp}\,\mathrm{d}k + \sin(n\phi)\int\limits_0^{\infty} g_n^{\mathrm{s}}(r)r\,\mathrm{d}r\int\limits_{-\infty}^{\infty} J_n(kr)e^{-ikp}\,\mathrm{d}k\right] = \\
&= \sum_{n=0}^{\infty} f_n^{\mathrm{c}}(p)\cos(n\phi) + f_n^{\mathrm{s}}(p)\sin(n\phi), \text{ where } \quad f_n^{\mathrm{c,s}}(p) = i^n\int\limits_0^{\infty} g_n^{\mathrm{c,s}}(r)r\,\mathrm{d}r\int\limits_{-\infty}^{\infty} J_n(kr)e^{-ikp}\,\mathrm{d}k
\end{aligned} \tag{142}$$

Now one assigns $g_{nl}(r) = \begin{cases} J_n(x_{nl}r) & ,r \leq 1 \\ 0 & ,r > 1 \end{cases}$ in Eq. 137 and Eq. 142:

$$g(r,\theta) = \sum_{n=0}^{\infty} g_n^{\mathrm{c}}(r)\cos(n\theta) + g_n^{\mathrm{s}}(r)\sin(n\theta), \text{ where } \quad g_n^{\mathrm{c,s}}(r) = \sum_{l=0}^{\infty} a_{nl}^{\mathrm{c,s}} J_n(x_{nl}r) \tag{143}$$

$$\begin{aligned}
f(p,\phi) &= \sum_{n=0}^{\infty} f_n^{\mathrm{c}}(p)\cos(n\phi) + f_n^{\mathrm{s}}(p)\sin(n\phi), \text{ where} \\
f_n^{\mathrm{c,s}}(p) &= i^n\int\limits_0^{1}\left[\sum_{l=0}^{\infty} a_{nl}^{\mathrm{c,s}} J_n(x_{nl}r)\right] r\,\mathrm{d}r\int\limits_{-\infty}^{\infty} J_n(kr)e^{-ikp}\,\mathrm{d}k = \sum_{l=0}^{\infty} i^n a_{nl}^{\mathrm{c,s}}\int\limits_{-\infty}^{\infty} e^{-ikp}\,\mathrm{d}k\int\limits_0^{1} J_n(x_{nl}r)J_n(kr)r\,\mathrm{d}r
\end{aligned} \tag{144}$$

The integral $\int\limits_0^1 J_n(x_{nl}r)J_n(kr)r\,\mathrm{d}r$ in Eq. 144 can be solved using equation 8 in chapter 5.11 of [70]. A cylinder function $\mathscr{C}_\mu$ and its conjugate $\bar{\mathscr{C}_\mu}$ of order $\mu$ satisfy:

$$\int\limits^{z} z\mathscr{C}_\mu(kz)\bar{\mathscr{C}}_\mu(lz)\,\mathrm{d}z = \frac{z[l\mathscr{C}_\mu(kz)\bar{\mathscr{C}}_{\mu+1}(lz) - k\mathscr{C}_{\mu+1}(kz)\bar{\mathscr{C}}_\mu(lz)]}{k^2 - l^2} \tag{145}$$

Bessel functions are *cylindrical fuctions*, so Eq. 145 can be expressed as:

$$\int\limits^{r} rJ_n(kr)J_n(x_{nl}r)\,\mathrm{d}r = \frac{r[x_{nl}J_n(kr)J_n'(x_{nl}r) - kJ_n'(kr)J_n(x_{nl}r)]}{k^2 - x_{nl}^2} \tag{146}$$

Integrating Eq. 146 from 0 to 1 one gets:

$$\int\limits_0^{1} rJ_n(kr)J_n(x_{nl}r)\,\mathrm{d}r = \left.\frac{r[x_{nl}J_n(kr)J_n'(x_{nl}r) - kJ_n'(kr)J_n(x_{nl}r)]}{k^2 - x_{nl}^2}\right|_0^1 = \frac{x_{nl}J_n(k)J_n'(x_{nl})}{k^2 - x_{nl}^2} \tag{147}$$

Using Eq. 147, Eq. 144 becomes:

$$
\begin{aligned}
f(p,\phi) &= \sum_{n=0}^{\infty} f_n^{\mathrm{c}}(p)\cos(n\phi) + f_n^{\mathrm{s}}(p)\sin(n\phi) && \text{, where}\\
f_n^{\mathrm{c,s}}(p) &= \sum_{l=0}^{\infty} a_{nl}^{\mathrm{c,s}} f_{nl}(p) && \text{, where}\\
f_{nl}(p) &= i^n x_{nl} J_n'(x_{nl}) \int\limits_{-\infty}^{\infty} \frac{J_n(k)e^{-ikp}}{k^2 - x_{nl}^2}\,\mathrm{d}k && (148)
\end{aligned}
$$

Using the integral representation of Bessel function $J_n(k) = \frac{1}{\pi i^n}\int\limits_0^{\pi} e^{ik\cos\theta}\cos(n\theta)\,\mathrm{d}\theta$ Eq. 148 becomes:

$$
\begin{aligned}
f_{nl}(p) &= i^n x_{nl} J_n'(x_{nl}) \int\limits_{-\infty}^{\infty} \frac{e^{-ikp}}{k^2 - x_{nl}^2}\left[\frac{1}{\pi i^n}\int\limits_0^{\pi} e^{ik\cos\theta}\cos(n\theta)\,\mathrm{d}\theta\right]\mathrm{d}k =\\
&= \frac{x_{nl}J_n'(x_{nl})}{\pi}\int\limits_0^{\pi}\cos(n\theta)\,\mathrm{d}\theta \int\limits_{-\infty}^{\infty}\frac{e^{ik[\cos\theta - p]}}{k^2 - x_{nl}^2}\,\mathrm{d}k \qquad (149)
\end{aligned}
$$

Using the identity $\int\limits_{-\infty}^{\infty}\frac{e^{ik[\cos\theta-p]}}{k^2-x_{nl}^2}\,\mathrm{d}k = -\frac{2\pi}{x_{nl}}\sin(x_{nl}|\cos\theta - p|)$ {Appendix B}, Eq. 149 becomes:

$$
\begin{aligned}
f_{nl}(p) &= \frac{x_{nl}J_n'(x_{nl})}{\pi}\int\limits_0^{\pi}\cos(n\theta)\,\mathrm{d}\theta\left[-\frac{2\pi}{x_{nl}}\sin(x_{nl}|\cos\theta - p|)\right] =\\
&= -2J_n'(x_{nl})\int\limits_0^{\pi}\cos(n\theta)\sin(x_{nl}|\cos\theta - p|)\,\mathrm{d}\theta \qquad (150)
\end{aligned}
$$

After some algebra Eq. 150 becomes {Appendix C}:

$$
f_{nl}(p) = -2J_n'(x_{nl})\sum_{m=0}^{\infty} 2J_m(x_{nl})\sin\left(m\frac{\pi}{2} - x_{nl}p\right)\left[\frac{\sin([m+n]\arccos p)}{m+n} + \frac{\sin([m-n]\arccos p)}{m-n}\right] \qquad (151)
$$

This completes the derivation. △

The final complete description of $f(p,\phi)$ with Bessel functions as the basis:

$$
\begin{aligned}
f(p,\phi) &= \sum_{n=0}^{\infty}\left[\cos(n\phi)\sum_{l=0}^{\infty} a_{nl}^{\mathrm{c}} f_{nl}(p) + \sin(n\phi)\sum_{l=0}^{\infty} a_{nl}^{\mathrm{s}} f_{nl}(p)\right]\text{, where} \qquad (152)\\
f_{nl}(p) &= -2J_n'(x_{nl})\sum_{m=0}^{\infty} 2J_m(x_{nl})\sin\left(m\frac{\pi}{2} - x_{nl}p\right)\left[\frac{\sin([m+n]\arccos p)}{m+n} + \frac{\sin([m-n]\arccos p)}{m-n}\right] \qquad (153)
\end{aligned}
$$

Having started with general *Radon transform*, then implemented Cormack theory and developed through Fourier-Bessel expansion eventually one reached a final form of $f(p,\phi)$. It needed explicit mathematical treatment in order to get rid of of integrals and singularities to be enough ”computationally friendly” to be implemented as a numerical algorithm. As well one already has obtained reconstruction expression at Eq. 143, which can be reformulated to:

$$
g(r,\theta) = \sum_{n=0}^{\infty}\left[\cos(n\theta)\sum_{l=0}^{\infty} a_{nl}^{\mathrm{c}} J_n(x_{nl}r) + \sin(n\theta)\sum_{l=0}^{\infty} a_{nl}^{\mathrm{s}} J_n(x_{nl}r)\right] \qquad (154)
$$

Index $n$ represents the angular component of the expansion. Index $l$ represents the radial component of the expansion. In practice indices $n$ and $l$ do not tend to infinity, but have limits $N$ and $L$, which are defined by the physical design of the experiment. $N$ and $L$ are the parameters of reconstruction algorithm and are defined by the system configuration. $N$ is defined from the number of sensor arrays. $L$ is defined from the number of sensors in a sensor array. According to Nyquist criterion resolution of a system is limited to roughly half number of its sensors. The tomography system has dimensionality of 2D. The size of the first dimension is limited by $N = \frac{\text{number of sensor arrays}}{2}$, the size of the second dimension is limited by $L = \frac{\text{number of sensors in a sensor array}}{2}$. With the indexing starting at 0, $n$ can obtain values up to $N-1$ and $l$ can obtain values up to $L-1$:

$$\begin{cases} n & \in \{0, 1, \cdots, N-1\} \\ l & \in \{0, 1, \cdots, L-1\} \end{cases}$$

The system installed at ISTTOK has 3 sensor arrays with 16 sensors each, therefore $N = 3/2 \approx 2$ and $L = 16/2 = 8$:

$$\begin{cases} n & \in \{0, 1\} \\ l & \in \{0, 1, 2, 3, 4, 5, 6, 7\} \end{cases}$$

# 19 Solution and reconstruction

In a real system Eq. 152 obtains discrete form, i.e. continuous function $f(p, \phi)$ becomes a set of discrete values $f_i = f(p_i, \phi_i)$. In fact $f_i$ is the measured intensity of a sensor $i$, which is located at point $(p_i, \phi_i)$. The set of points $(p_i, \phi_i)$ describes the overall positioning and geometry of sensor arrays in the tokamak. The set of points $(p_i, \phi_i)$ is defined by a physical system design. By defining elements $C^{\mathrm{c}}_{i,nl} = f_{nl}(p_i)\cos(n\phi_i)$ and $C^{\mathrm{s}}_{i,nl} = f_{nl}(p_i)\sin(n\phi_i)$ Eq. 152 can be expressed as a linear system:

$$f_i = \sum_{n=0}^{N-1} \sum_{l=0}^{L-1} \left[ C^{\mathrm{c}}_{i,nl} a^{\mathrm{c}}_{nl} + C^{\mathrm{s}}_{i,nl} a^{\mathrm{s}}_{nl} \right] = \sum_{n=0}^{N-1} \sum_{l=0}^{L-1} C^{\mathrm{c,s}}_{i,nl} a^{\mathrm{c,s}}_{nl} \tag{155}$$

In matrix form:

$$\mathbf{f} = \mathbf{C} \cdot \mathbf{a} \tag{156}$$

$\mathbf{C}$ is called a *contribution matrix*, because it describes how much each *basis function* contributes to a signal in each detector, $\mathbf{f}$ is a back-calculated signal data and $\mathbf{a}$ is a to-be-solved *coeficient vector*. More explicitly:

$$\begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_S \end{bmatrix} = \begin{bmatrix} C^{\mathrm{c}}_{1,\,00} & \cdots & C^{\mathrm{c}}_{1,\,N-1\,L-1} & C^{\mathrm{s}}_{1,\,00} & \cdots & C^{\mathrm{s}}_{1,\,N-1\,L-1} \\ C^{\mathrm{c}}_{2,\,00} & \cdots & C^{\mathrm{c}}_{2,\,N-1\,L-1} & C^{\mathrm{s}}_{2,\,00} & \cdots & C^{\mathrm{s}}_{2,\,N-1\,L-1} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ C^{\mathrm{c}}_{S,\,00} & \cdots & C^{\mathrm{c}}_{S,\,N-1\,L-1} & C^{\mathrm{s}}_{S,\,00} & \cdots & C^{\mathrm{s}}_{S,\,N-1\,L-1} \end{bmatrix} \cdot \begin{bmatrix} a^{\mathrm{c}}_{00} \\ \vdots \\ a^{\mathrm{c}}_{N-1\,L-1} \\ a^{\mathrm{s}}_{00} \\ \vdots \\ a^{\mathrm{s}}_{N-1\,L-1} \end{bmatrix} \tag{157}$$

where $S$ is total number of sensors in the system, i.e. $S = \#$ of arrays $\times$ $\#$ of sensors per array $= 3 \times 16$ in ISTTOK. Due to the fact that $C^{\mathrm{s}}_{i,nl}\Big|_{n=0} = C^{\mathrm{s}}_{i,0l} = f_{nl}(p_i)\sin(0 \cdot \phi_i) = 0$ system (Eq. 157) can be reduced to:

$$\begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_S \end{bmatrix} = \begin{bmatrix} C^{\mathrm{c}}_{1,\,00} & \cdots & C^{\mathrm{c}}_{1,\,N-1\,L-1} & C^{\mathrm{s}}_{1,\,10} & \cdots & C^{\mathrm{s}}_{1,\,N-1\,L-1} \\ C^{\mathrm{c}}_{2,\,00} & \cdots & C^{\mathrm{c}}_{2,\,N-1\,L-1} & C^{\mathrm{s}}_{2,\,10} & \cdots & C^{\mathrm{s}}_{2,\,N-1\,L-1} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ C^{\mathrm{c}}_{S,\,00} & \cdots & C^{\mathrm{c}}_{S,\,N-1\,L-1} & C^{\mathrm{s}}_{S,\,10} & \cdots & C^{\mathrm{s}}_{S,\,N-1\,L-1} \end{bmatrix} \cdot \begin{bmatrix} a^{\mathrm{c}}_{00} \\ \vdots \\ a^{\mathrm{c}}_{N-1\,L-1} \\ a^{\mathrm{s}}_{10} \\ \vdots \\ a^{\mathrm{s}}_{N-1\,L-1} \end{bmatrix} \tag{158}$$

Note that in general *contribution matrix* C is not a square matrix. It has number $S$ of rows and number $2NL - L = (2N-1)L$ of columns. Basic theory of linear equation systems tells that, when a matrix is not square, the system does not have uniquely defined solution. When matrix has more columns than rows, the system is *underdetermined* (it has more variables than equations). When matrix has more rows than columns, the system is *overdetermined* (it has more equations than variables). In this case the system has $S = 48$ equations of $(2N-1)L = (2 \cdot 2 - 1) \cdot 8 = 24$ variables. Bare in mind that $N$ and $L$ are only limiting parameters, one has freedom to have less variables in the system. In fact the choice of best $N$ and $L$ is a delicate issue, which will be discussed later. As mentioned before *overdetermined* systems do not have a unique solution, however one can always have an approximate "best fit" solution.

## 19.1 Singular Value Decomposition

**S**ingular **V**alue **D**ecomposition (SVD) is a matrix factorization, which enables to solve *overdetermined* linear systems. Solution is obtained as a minimization of summed squared error of the system, i.e. solution is a fit by least squares. Using SVD one can decompose *contribution matrix* $\mathbf{C}$ to its factors:

$$\mathbf{C} = \mathbf{U} \cdot \mathbf{W} \cdot \mathbf{V}^{\mathrm{T}} \tag{159}$$

where $\mathbf{U}$ is an *orthogonal matrix*, $\mathbf{W}$ is a *diagonal matrix* with *singular values* on its diagonal and $\mathbf{V}^{\mathrm{T}}$ is a *transpose* of an *orthogonal matrix* $\mathbf{V}$. In order to solve equation systems one needs to invert $\mathbf{C}$ into $\mathbf{C}^{-1}$:

$$\mathbf{C}^{-1} = \left[\mathbf{U} \cdot \mathbf{W} \cdot \mathbf{V}^{\mathrm{T}}\right]^{-1} = \mathbf{U}^{-1} \cdot \mathbf{W}^{-1} \cdot \left[\mathbf{V}^{\mathrm{T}}\right]^{-1} = \frac{\mathbf{U}^{\mathrm{T}} \cdot \mathbf{V}}{\mathbf{W}} \tag{160}$$

Last transition was possible due to the property of *orthogonal matrices*: $\mathbf{U}^{-1} = \mathbf{U}^{\mathrm{T}}$; $\mathbf{V}^{-1} = \mathbf{V}^{\mathrm{T}}$. $\frac{1}{\mathbf{W}}$ is *diagonal matrix* with reciprocal *singular values* on its diagonal. Strickly speaking $\mathbf{C}^{-1}$ is an incorrect mathematical notation here due to the fact that $\mathbf{C}$ is not a square matrix. In fact notation of *Moore-Penrose pseudoinverse* [71] $\mathbf{C}^{+}$ should be used instead. However for the purpose of readability and derivation one used $\mathbf{C}^{-1}$. The C code of SVD [72] is built on algorithm called Householder reduction {Appendix E}.

## 19.2 Reconstruction

Once *pseudo-inverse contribution matrix* $\mathbf{C}^{+}$ is acquired with SVD, *coeficient vector* $\mathbf{a}$ is obtained by:

$$\mathbf{a} = \mathbf{C}^{+} \cdot \mathbf{f} \tag{161}$$

Now it is straightforward to obtain the actual reconstruction image. By defining *basis functions* $B^{\mathrm{c}}_{nl}(r,\theta) = J_{nl}(x_{nl}r)\cos(n\theta)$ and $B^{\mathrm{s}}_{nl}(r,\theta) = J_{nl}(x_{nl}r)\sin(n\theta)$ Eq. 154 can be expressed as:

$$g(r,\theta) = \sum_{n=0}^{N-1}\sum_{l=0}^{L-1}\left[a^{\mathrm{c}}_{nl}B^{\mathrm{c}}_{nl}(r,\theta) + a^{\mathrm{s}}_{nl}B^{\mathrm{s}}_{nl}(r,\theta)\right] = \sum_{n=0}^{N-1}\sum_{l=0}^{L-1} a^{\mathrm{c,s}}_{nl}B^{\mathrm{c,s}}_{nl}(r,\theta) = \sum_{i=0}^{(2N-1)L-1} a_i B_i(r,\theta) \tag{162}$$

In vector form:

$$\mathbf{g} = \sum_{i=0}^{(2N-1)L-1} a_i \mathbf{B}_i \tag{163}$$

where $a_i$ are elements of *coeficient vector* $\mathbf{a}$ and $\mathbf{B}_i$ are elements (*bases*) of *basis matrix* $\mathbf{B}$. To put in words, reconstructed image is a weighted sum of *basis functions* $B_i$ weighted by *coeficients* $\mathbf{a}_i$, which is obtained by applying *pseudo-inverse contribution matrix* $\mathbf{C}^{+}$ to sensor data $\mathbf{f}$.

# 20 Calculation of Contribution matrix

In section 19 it was demonstrated how to construct a linear set of equations and obtain its approximate solution. However obtaining *contribution matrix* $\mathbf{C}$ itself still remains a problem. Calculation of $\mathbf{C}$ requires calculation of Eq. 153, which is an analytic-like expression and contains infinite sum. Fortunately there is a way to overcome this problem by exploiting so-called *Fourier-Generic* method introduced by Pedro Carvalho in [73]:

$$g(r,\theta) = \sum_{n=0}^{N-1} \left[ \cos(n\theta) \sum_{l=0}^{L-1} a^{\mathrm{c}}_{nl} g_{nl}(r) + \sin(n\theta) \sum_{l=0}^{L-1} a^{\mathrm{s}}_{nl} g_{nl}(r) \right] \tag{164}$$

where $g_{nl}(r)$ can be any function, which presents some level of radial localization along $l$. This method allows to simplify calculation of $f_i$ to a vector dot product:

$$f_i = \mathbf{p}_i \cdot \mathbf{g} \tag{165}$$

where $f_i$ is a datum of sensor $i$, $\mathbf{p}_i$ is a *projection vector* of sensor $i$ and $\mathbf{g}$ is the emmissivity distribution expressed as a vector. This description can be extended to matrix-vector product:

$$\mathbf{f} = \mathbf{P} \cdot \mathbf{g} \tag{166}$$

where $\mathbf{P}$ is a *projection matrix*. Eq. 166 represents the *synthetic* diagnostic of a system. In fact any emmissivity distribution (image) can be fed as $\mathbf{g}$ to obtain virtual values on sensors, i.f. signal $\mathbf{f}$ is back-calculated. If instead of image $\mathbf{g}$ an image of *basis* $\mathbf{b}_i$ is fed in, one obtains a virtual signal $\mathbf{f}$ generated by that *basis*. In this way a matrix can be constructed containing virtual signals generated by all *bases* $\mathbf{B}$, which exactly what *contribution matrix* $\mathbf{C}$ is:

$$\mathbf{C} = \mathbf{P} \cdot \mathbf{B} \tag{167}$$

## 20.1 How does *projection matrix* P look like?

*Projection matrix* $\mathbf{P}$ is comprised of *projection vectors* $\mathbf{p}_i$. Projection vector $\mathbf{p}_i$ is obtained from the percentage of intersection each pixel with each pair of viewlines (Fig. 89). Each $\mathbf{p}_i$ visualized as a 2D picture demonstrates viewing areas of each sensor sensor $i$ (Fig. 90). Since each $\mathbf{p}_i$ depends on imposed geometry and positioning of sensors, so does *projection matrix* $\mathbf{P}$. It is also obvious that viewing angles no longer represent 1D line integrals, but instead integrals over speficic area imposed by the viewing angle. Due to this fact an assumption present in orginal Fourier-Bessel derivation, that sensors are point-line and have no width, is no longer valid. *Fourier-Generic* method allows to dispose of this unnecessary assumption and give a more straighforward and intuitive approach.

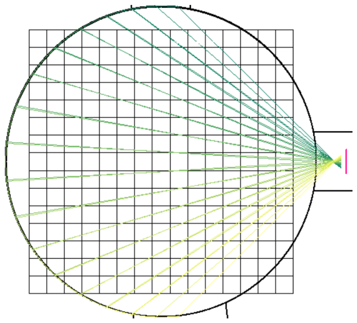


Figure 89: Viewing lines of the equatorial detector

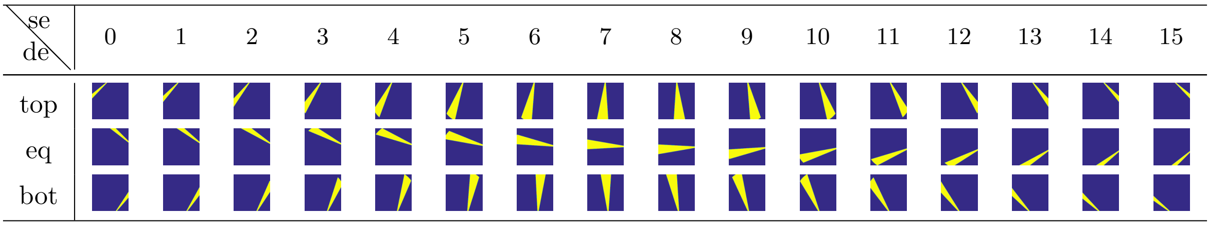


Figure 90: Sensors and their viewing angles arranged according to their positions in detectors

### 20.2 How does *basis matrix* B look like?

*Basis matrix* $\mathbf{B}$ is comprised of *basis vectors* $\mathbf{b}_j$. In fact $\mathbf{b}_j$ visualized as a 2D picture demonstrates a *basis function* $B_j(r,\theta)$. At most one can have $L(2N-1) = 8(2\cdot 2-1) = 24$ *bases* in case of ISTTOK setup (Fig. 91, 92). Bessel-sine *bases* of order $n = 0$ are not depiceted, because they yield 0.

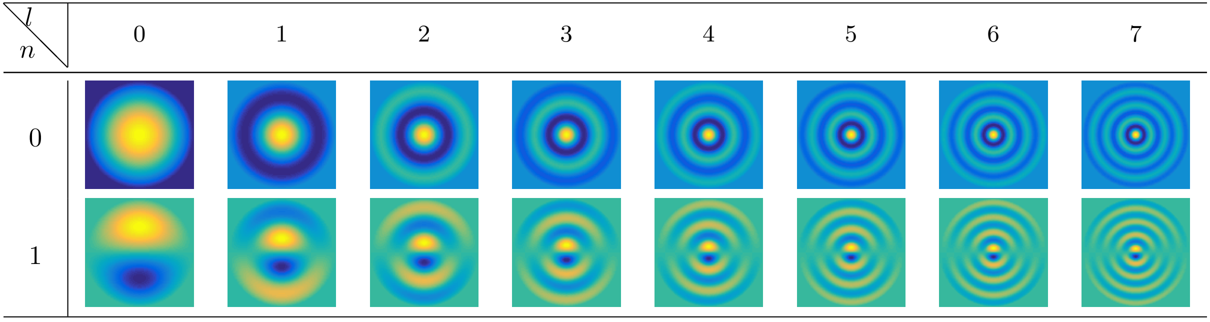


Figure 91: Basis functions $B^{\mathrm{c}}_{nl}(r,\theta) = J_{nl}(x_{nl}r)\cos(n\theta)$

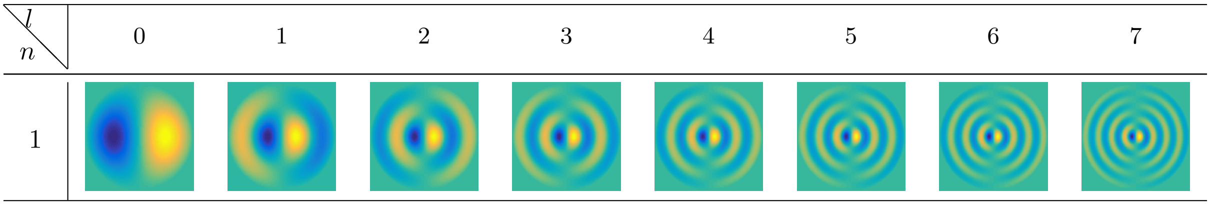


Figure 92: Basis functions $B^{\mathrm{s}}_{nl}(r,\theta) = J_{nl}(x_{nl}r)\sin(n\theta$

# 21 Algorithm testing and optimization

## 21.1 Error optimization

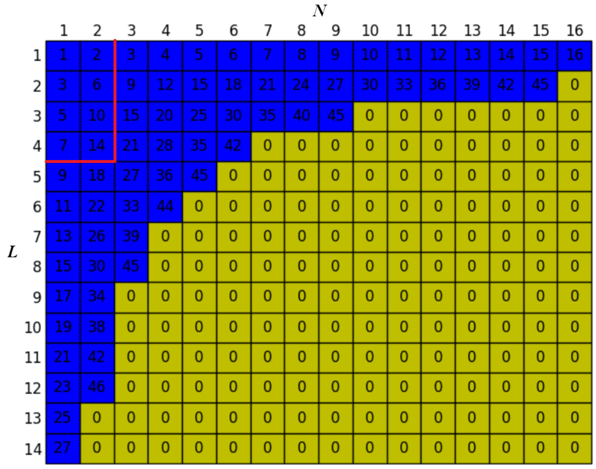
Algorithm testing and optimization is a big part of the development of tomography algorithms. Proper selection of limiting $N$ and $L$ is a crucial decision, which defines the quality/defectiveness of reconstruction. $N$ and $L$ set the limit angular and radial resolution. $N$ deals with the "angular order" of the expected image, i.e. how much angular dependence will be contained in the image. $L$ deals with "radial order" of the expected image, i.e. how much radial dependence will be contained in the image. It is a bit controversial that one has to choose the property of the reconstruction image apriori, because in ideal case it is the reconstruction, which has to indicate and demonstrate the properties of the emission distribution. In order to arrive to a smart guess of what $N$ and $L$ are, a study on phantoms and their reconstructions was conducted. A master student César Alves designed a Python code TOT (**T**omography **O**ptimization **T**ool) for studying phantoms and their reconstructions, which is presented in his Master thesis [74]. He carried a study to obtain the best geometry setup and the best pair of $\{N, L\}$. This study was conducted as an optimization process based on errors $\epsilon$ and $\chi^2$ as minimization criteria. Error $\epsilon$ is defined as an average absolute difference between reconstruction image pixel and phantom image pixel.

$$\epsilon = \frac{1}{K}\sum_{i}^{K}|g_i - \widetilde{g}_i| \tag{168}$$

where $\mathbf{g}$ is the phantom image, $\widetilde{\mathbf{g}}$ is the acquired reconstruction image and $K$ is the number of pixels. Error $\epsilon$ is very useful to describe reconstruction quality. $\epsilon = 0$ would indicate a perfect reconstruction, anyway $\epsilon < 0.1$ is considered a valid reconstruction. $\chi^2$ is another type of statistical error, which describes the "goodness of fit". It is called *Pearson's test* and it typically summarizes the discrepancy between observed values and the values expected under the model in question.

$$\chi^2 = \frac{1}{\nu}\sum_{i}^{S}\frac{\left(f_i - \widetilde{f}_i\right)^2}{f_i} = \frac{1}{S - L(2N-1)}\sum_{i}^{S}\frac{\left(f_i - \widetilde{f}_i\right)^2}{f_i} \tag{169}$$

where $f_i$ is phantom-generated theoretical signal value at sensor $i$, $\widetilde{f}_i$ is a obtained signal value at sensor $i$, $\nu$ is the number of degrees of freedom of the system, where $S$ is the number of observed quantities (number of sensors = 48) and $L(2N-1)$ is the number of fitted paramaters (number of *basis functions*). Noise level $w$ is present in the system and is contained within $\widetilde{\mathbf{f}}$. Varying the geometry of the detector setup and looking at different pairs of limiting $\{N, L\}$ allowed César Alves to select the best geometry configuration and the best $\{N, L\}$ (Fig. 93, 94, 95).



| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
| 2 | 3 | 6 | 9 | 12 | 15 | 18 | 21 | 24 | 27 | 30 | 33 | 36 | 39 | 42 | 45 | 0 |
| 3 | 5 | 10 | 15 | 20 | 25 | 30 | 35 | 40 | 45 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 4 | 7 | 14 | 21 | 28 | 35 | 42 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 5 | 9 | 18 | 27 | 36 | 45 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 6 | 11 | 22 | 33 | 44 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 7 | 13 | 26 | 39 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 8 | 15 | 30 | 45 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 9 | 17 | 34 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 10 | 19 | 38 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 11 | 21 | 42 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 12 | 23 | 46 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 13 | 25 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 14 | 27 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Figure 93: $\{N, L\}$ sets that result in $\leq 48$ *basis* functions (blue) and those which satisfy Nyquist criterion (inside red rectangular) [74]

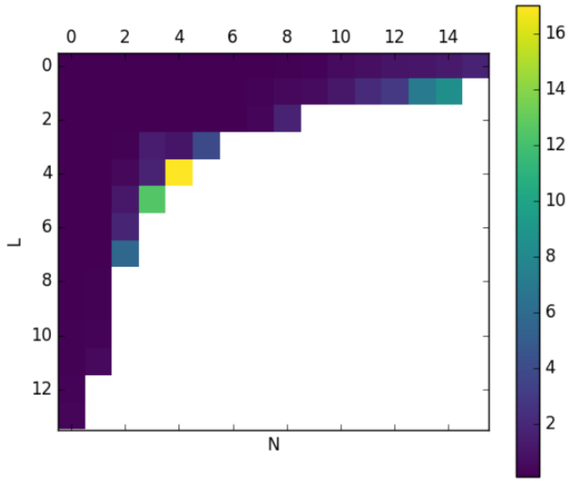


Figure 94: Error $\epsilon$ with minimum at $\{N, L\} = \{2, 2\}$ [74]

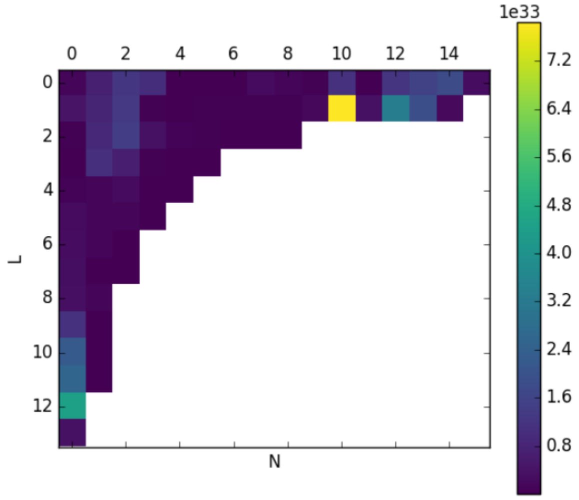


Figure 95: Error $\chi^2$ with minimum at $\{N, L\} = \{3, 5\}$ [74]

According to César Alves the optimal $\{N, L\}$ that produces least $\epsilon$ and $\chi^2$ and fits into Nyquist criterion happens to be $\{2, 2\}$, which results in 6 *basis functions* (4 of Bessel-cosine origin and 2 of Bessel-sine origin). Geometry of detector positioning was also optimized and optimal detector coordinates were obtained [74], which are encoded into *Projection matrix* $\mathbf{P}$ (Fig. 90). All these conditions serves as starting point for numerical processing and will be unquestionable within the scope of this thesis.

## 21.2 Error maps

As mentioned in previous paragraph, the quality of a single reconstruction can be described with two errors $\epsilon$ and $\chi^2$. However one is interested and concerned how reconstructions would perform in general case under various conditions and various positioning of sources (phantoms). In order to obtain this information, one can construct an *error map*. *Error map* is a 2D represenation of tokamak cross-section, where each points indicates a likely estimate of a reconstruction error (Fig. 96). Since two numbers $\epsilon$ and $\chi^2$ describe a single reconstruction, therefore two error maps can be built by placing (scanning) the phantom across reconstruction domain and calculating the corresponding error value at that point. This technique enables to explore the properties of particular phantom and observe areas where reconstuction is most effective in the tokamak cross-section. Error maps are presented in the section 22. (Fig. 101, 102, 108, 109, 115, 116).

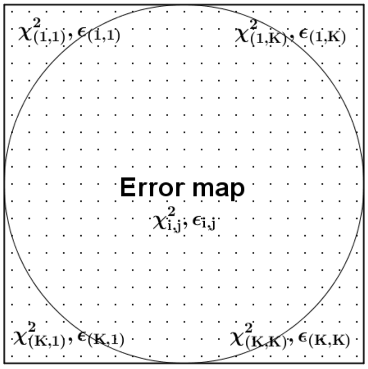

$\chi^2_{(1,1)}, \epsilon_{(1,1)}$
$\chi^2_{(1,K)}, \epsilon_{(1,K)}$
Error map
$\chi^2_{i,j}, \epsilon_{i,j}$
$\chi^2_{(K,1)}, \epsilon_{(K,1)}$
$\chi^2_{(K,K)}, \epsilon_{(K,K)}$

Figure 96: Error maps are 2D domains (scalar fields) of error values, where $\chi^2$ and $\epsilon$ are obtained by reconstructions of a particular phantom for each point of the map

### 21.3 Error correction grid (selective reconstruction)

So far one explored the possibilities of reconstruction model binded to particular *contribution matrix* $\mathbf{C}$, which was built using *projection matrix* $\mathbf{P}$ and *basis matrix* $\mathbf{B}$. However there is intrinsic assumption, that has to be mentioned. So far it was assumed, that reconstruction happens at the center of tokamak cross-section, i.e. all *basis functions* were centered. In a more realistic scenario plasma can be experiencing Shafranov shift, i.e. off-centered. Therefore a reconstruction model based on a centered *basis functions* would perform poorly. If only one could know the offset of the plasma center and perform the reconstruction accordingly, the quality of the reconstruction would be improved significantly. With the following development one will try to approach the complication of reconstructions for offset plasmas, even in the case when the offset is unknown. Instead of having only one single reconstruction model, one can define multiple *contribution matrices* $\{\mathbf{C}_1, \mathbf{C}_2, \dots\}$, their pseudo-inverses $\{\mathbf{C}_1^+, \mathbf{C}_2^+, \dots\}$, *basis matrices* $\{\mathbf{B}_1, \mathbf{B}_2, \dots\}$ and *coeficient vectors* $\{\mathbf{a}_1, \mathbf{a}_2, \dots\}$, where each pair of $\{\mathbf{B}_i, \mathbf{a}_i\}$ considers a reconstruction model for a particular offset from the tokamak axis. One can organize these multiple models in a pseudo-matrix relationship, which represent a set of multiple recontructions $\{\mathbf{g}_1, \mathbf{g}_2, \dots\}$, which cover a certain area of tokamak cross-section (Fig. 97).

$$\left\{ \begin{array}{ccc} \mathbf{B}_{1,1} & \cdots & \mathbf{B}_{1,J} \\ \vdots & \ddots & \vdots \\ \mathbf{B}_{I,1} & \cdots & \mathbf{B}_{I,J} \end{array} \right\} \circ \left\{ \begin{array}{ccc} \mathbf{a}_{1,1} & \cdots & \mathbf{a}_{1,J} \\ \vdots & \ddots & \vdots \\ \mathbf{a}_{I,1} & \cdots & \mathbf{a}_{I,J} \end{array} \right\} = \left\{ \begin{array}{ccc} \mathbf{g}_{1,1} & \cdots & \mathbf{g}_{1,J} \\ \vdots & \ddots & \vdots \\ \mathbf{g}_{I,1} & \cdots & \mathbf{g}_{I,J} \end{array} \right\} \tag{170}$$

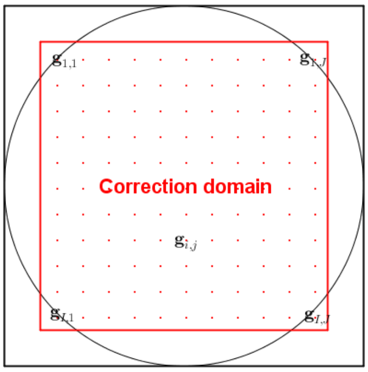


Figure 97: Error correction domain depicted in the red square

In theory one could perform all the reconstructions defined in a grid and select the most suitable one by comparing them to the original phantom, i.e. selecting reconstruction with best visual correspondence and the least $\epsilon$ (Eq. 168). However it is computationally unpractical and even impossible in a real-time scenario. In real-time scenario one has a chance to perform only a single reconstruction at a time and there is no phantom to compare to. Therefore one has to choose only one reconstruction from the grid and perform it. The criterion for the selection is the best fit of back-projected signal $\mathbf{f}$ to the arriving signal $\widetilde{\mathbf{f}}$, i.e. selecting the minimum value of $\chi^2$ (Eq. 171) among all possible reconstructions.

$$\chi^2_{\min} = \min\{\chi^2_1, \chi^2_2, \dots\} \tag{171}$$

The index of $\chi^2_{\min}$ indicates (Eq. 172), which reconstruction has to be selected for processing.

$$\arg\chi^2_{\min} = \arg\min\{\chi^2_1, \chi^2_2, \dots\} \tag{172}$$

The obtained index is pointing to a particular $\mathbf{B}_i$ and corresponding $\mathbf{a}_i$, which are used to reconstruct the image (Eq. 163). Correction grid is defined during offline state of real-time application. Sets $\{\mathbf{B}_i\}$, $\{\mathbf{C}_i\}$ and $\{\mathbf{C}_i^+\}$ are calculated for each point of correction grid accordingly during offline state. During online state $\{\mathbf{a}_i\}$ are calculated by Eq. 161, $\{\mathbf{f}_i\}$ is calculated by Eq. 156, selection is performed with Eq. 172 and reconstruction is processed by Eq. 163. In the section 22 one can clearly see improved quality between center-based reconstructions and error-corrected reconstructions for single-source phantoms (Fig. 99,100). However error correction grid does not dramatically improve double-source (Fig. 106,107) and ring-source reconstructions Fig. 113,114). The effect of error correction grid is also very evident in error maps for single-source reconstructions (Fig. 103,104) and ring-source reconstructions (Fig. 117,118). In general error-correction grid cannot guarantee always a good quality reconstruction, even if $\chi^2$ and $\epsilon$ values indicate good reconstruction, but surely does increase chances to have an improved reconstruction.

# 22 Results

## 22.1 Single source study

### 22.1.1 Phantoms

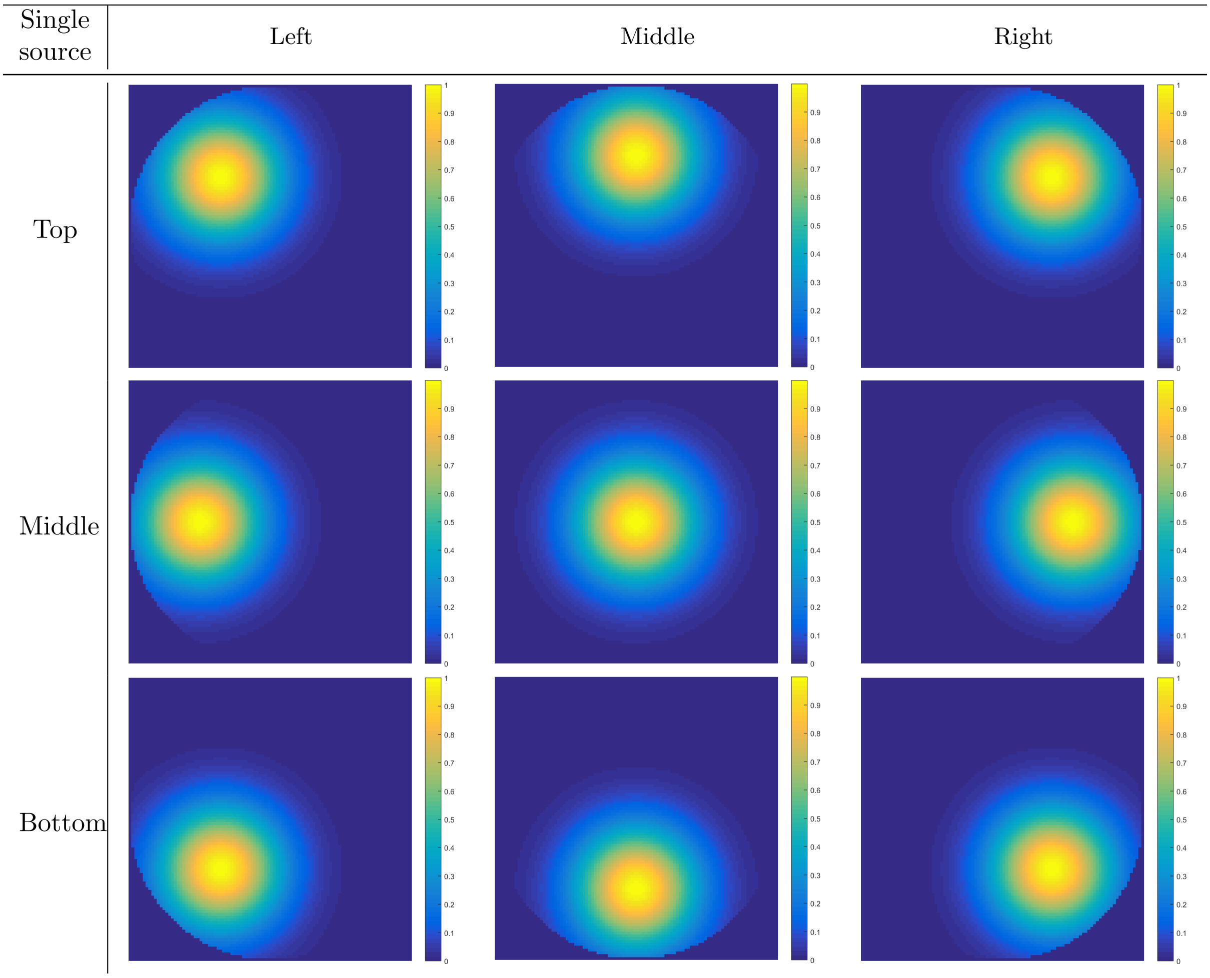


Figure 98: Single source: phantoms of Gaussian width $\sigma = 0.3$ and off-center displacement $\Delta r = 0.5$

### 22.1.2 Reconstructions

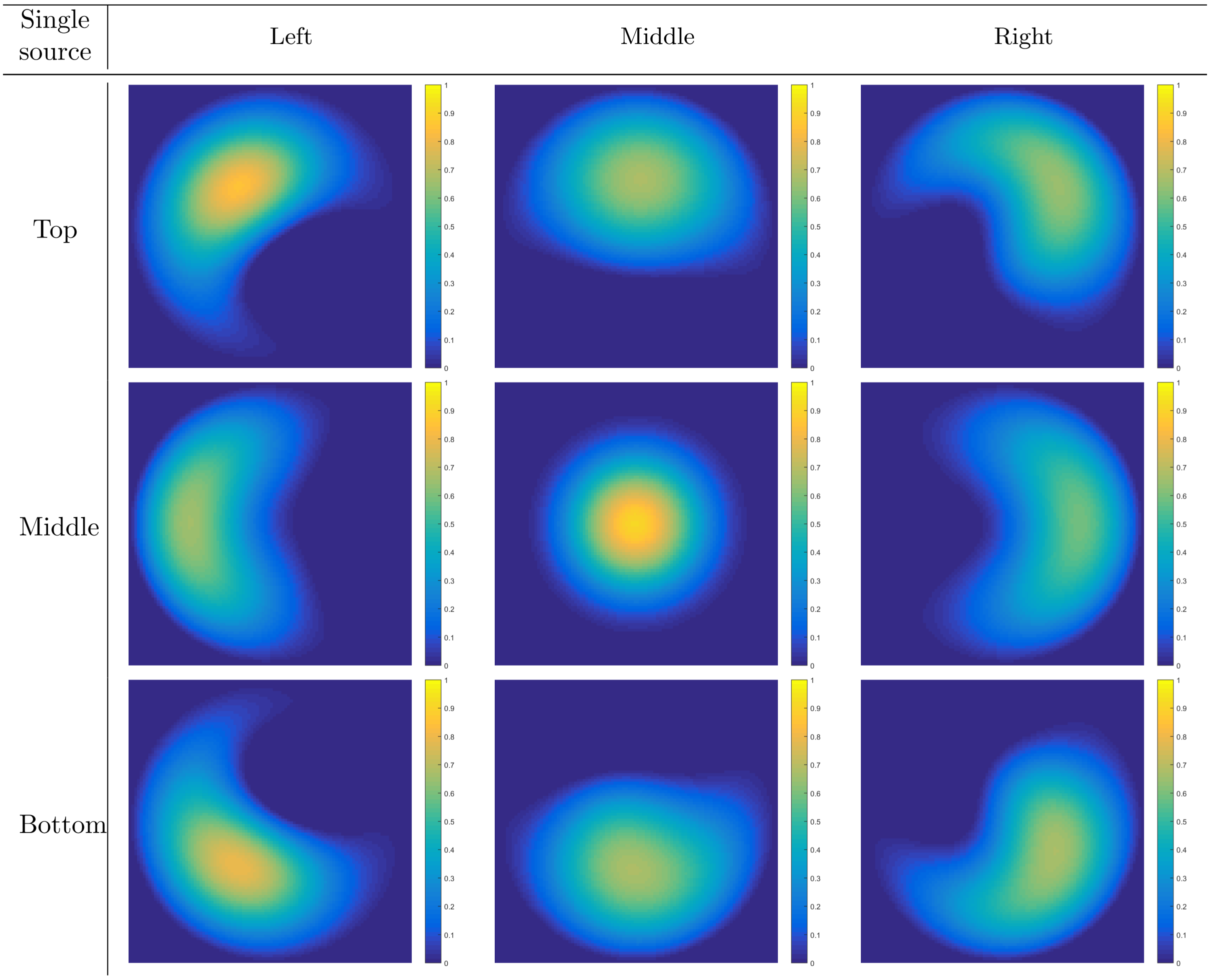


Figure 99: Single source: reconstructions of Fig. 98

| Single source | Left | Middle | Right |
|---|---|---|---|
| Top | 331.680 | 903.209 | 1865.886 |
| Middle | 1401.224 | 2.513 | 42.837 |
| Bottom | 263.089 | 976.746 | 2034.286 |

Table 13: Single source: $\chi^2$ error

| Single source | Left | Middle | Right |
|---|---|---|---|
| Top | 0.086040 | 0.087075 | 0.084600 |
| Middle | 0.086579 | 0.014968 | 0.081817 |
| Bottom | 0.085229 | 0.088241 | 0.083980 |

Table 14: Single source: $\epsilon$ error

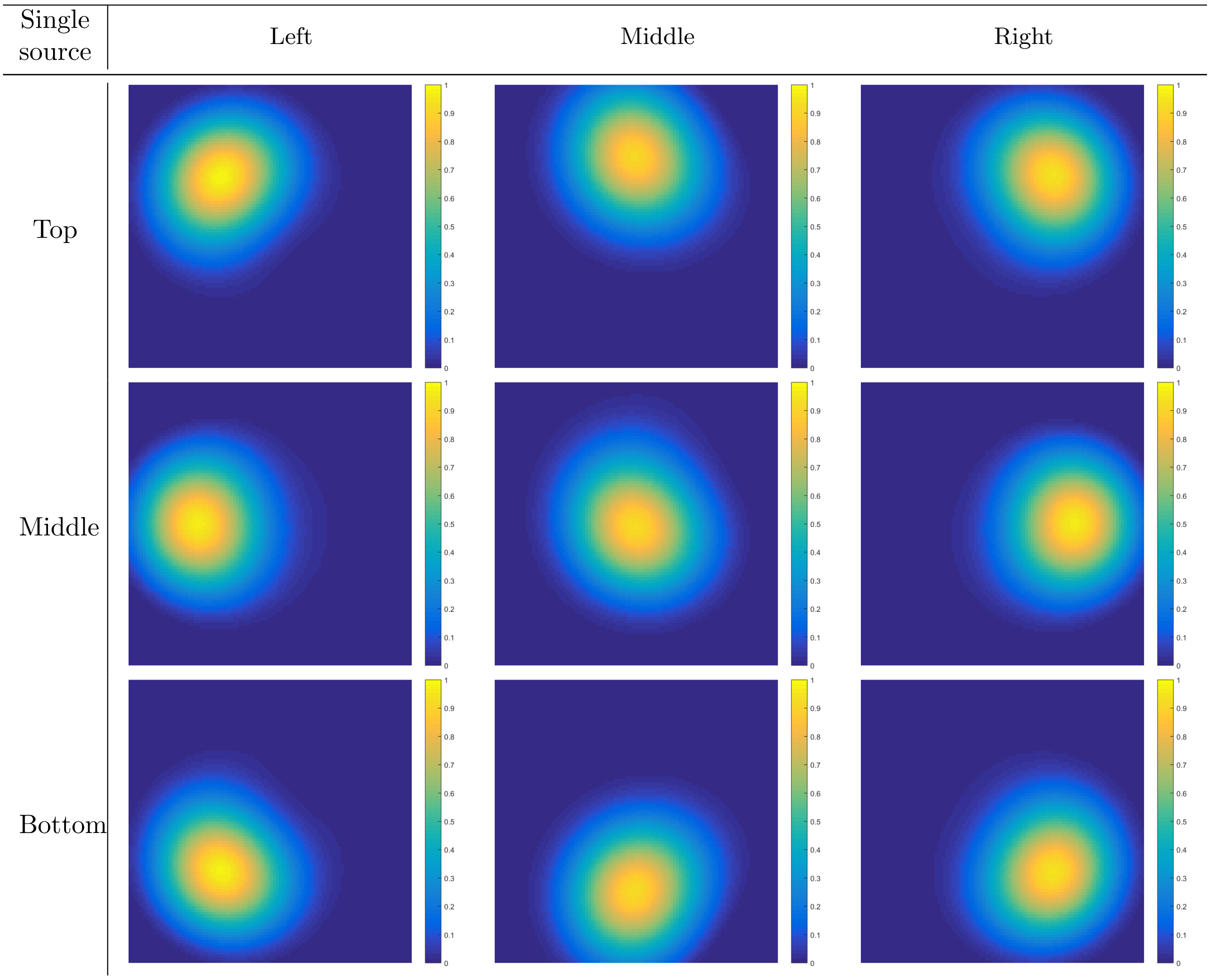


Figure 100: Single source: error-corrected reconstructions of Fig. 98

| Single source | Left | Middle | Right |
|---|---|---|---|
| Top | 0.588 | 0.599 | 0.454 |
| Middle | 0.581 | 1.254 | 0.266 |
| Bottom | 0.485 | 0.615 | 0.524 |

Table 15: Single source: $\chi^2$ error after minimization

| Single source | Left | Middle | Right |
|---|---|---|---|
| Top | 0.014183 | 0.015633 | 0.012596 |
| Middle | 0.010698 | 0.015532 | 0.010965 |
| Bottom | 0.014122 | 0.015545 | 0.012538 |

Table 16: Single source: $\epsilon$ error after minimization

### 22.1.3 Error maps uncorrected

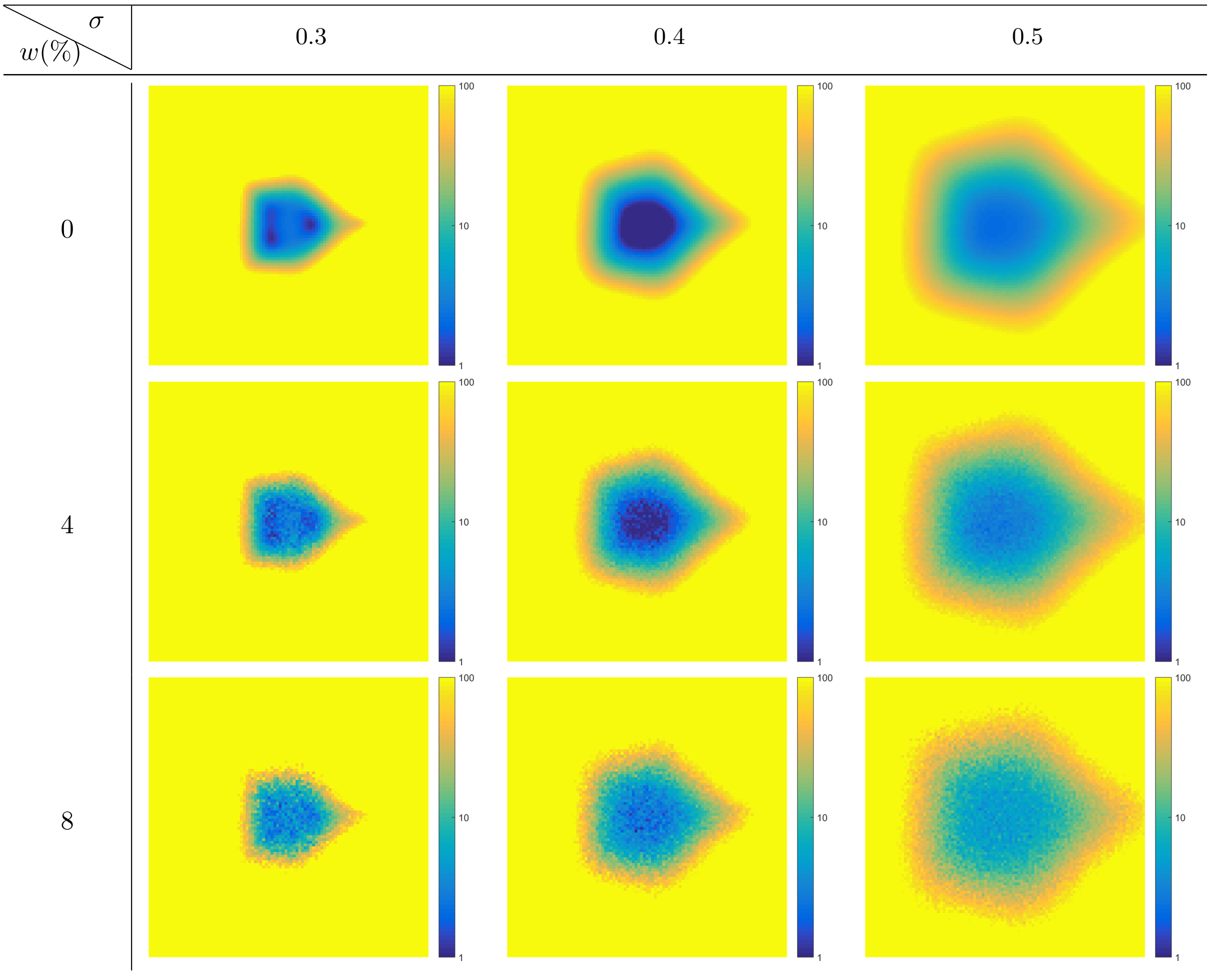


Figure 101: Single source: error $\chi^2$ maps

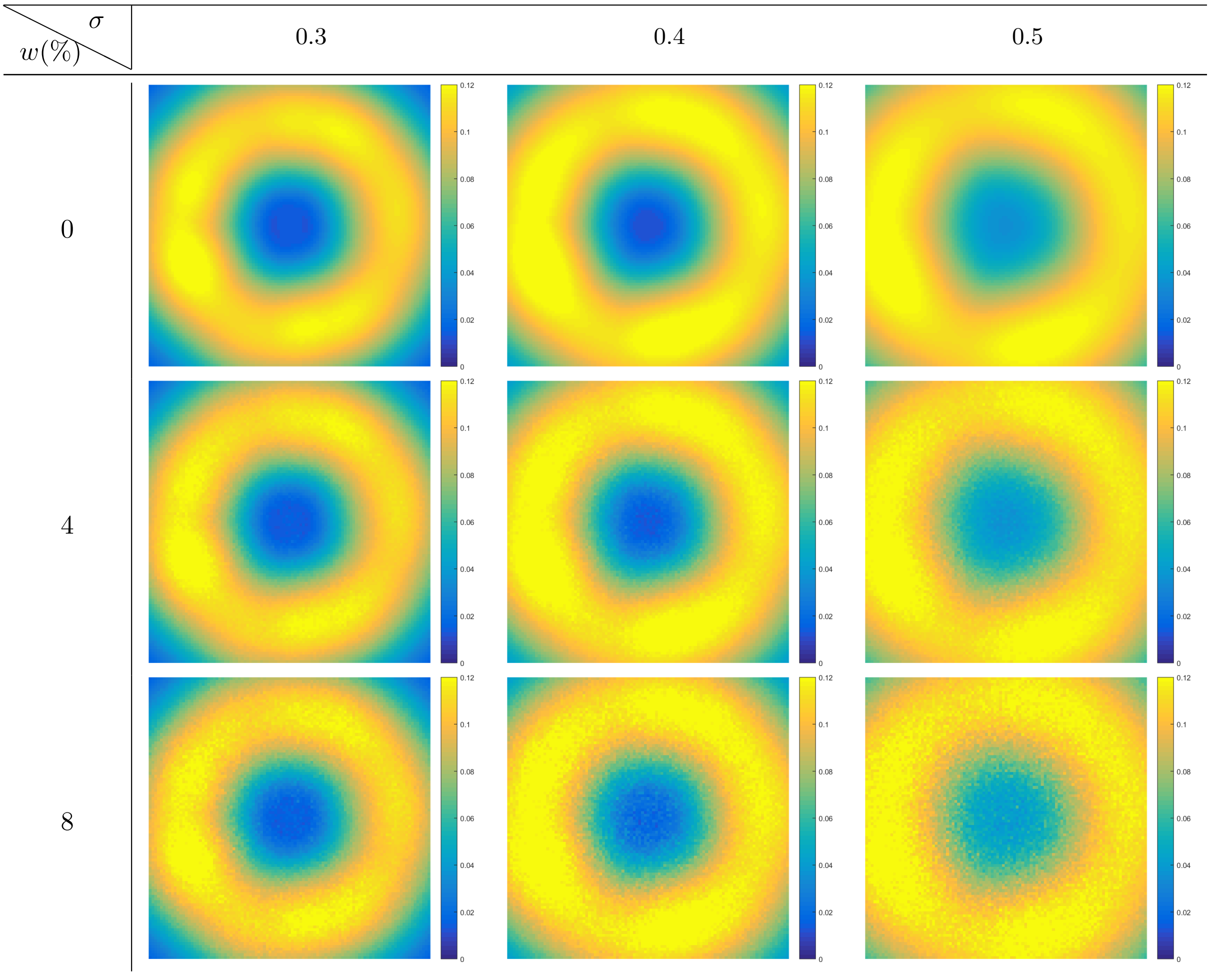


Figure 102: Single source: error $\epsilon$ maps

#### 22.1.4 Error maps corrected

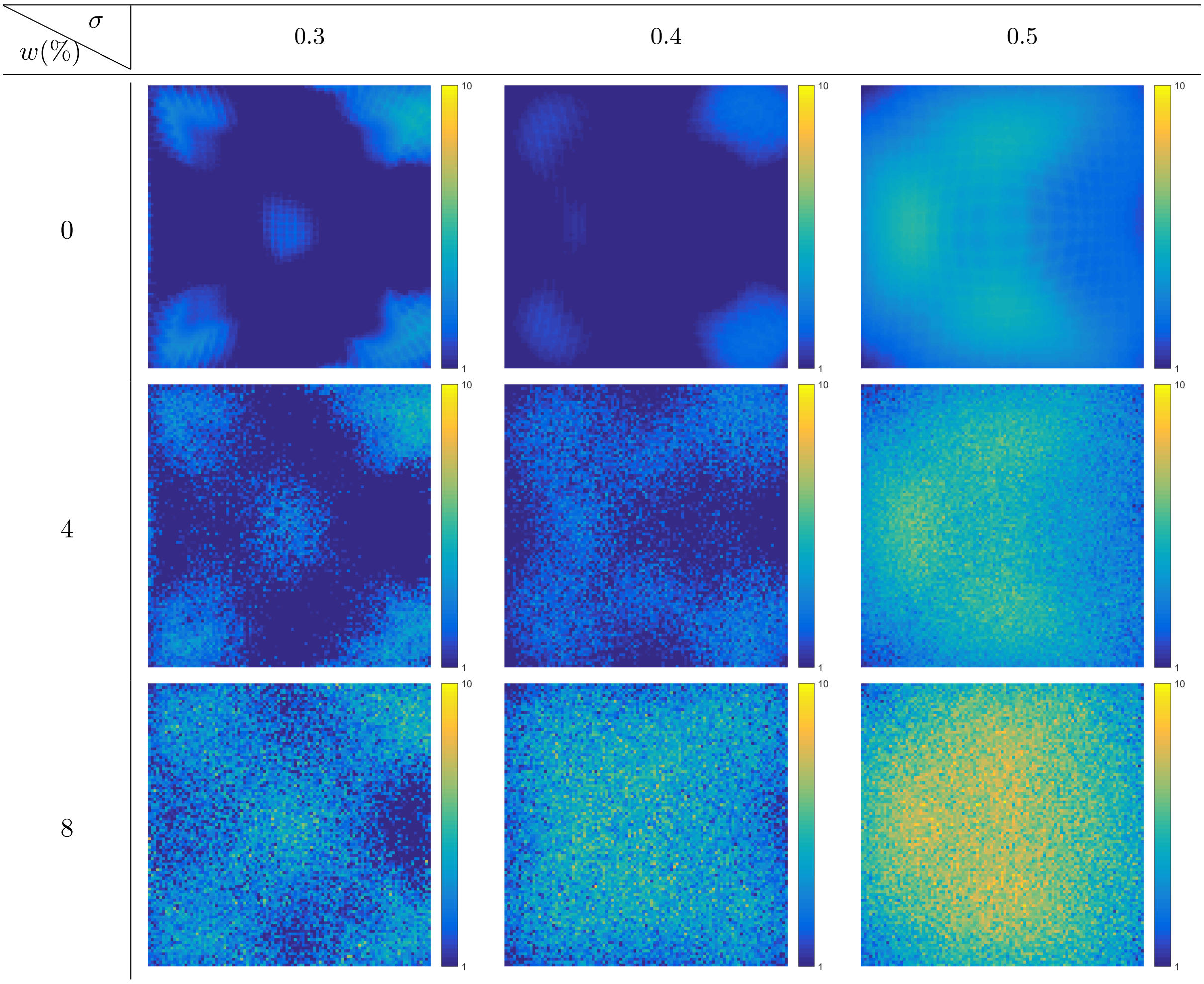


Figure 103: Single source: error-corrected $\chi^2$ maps

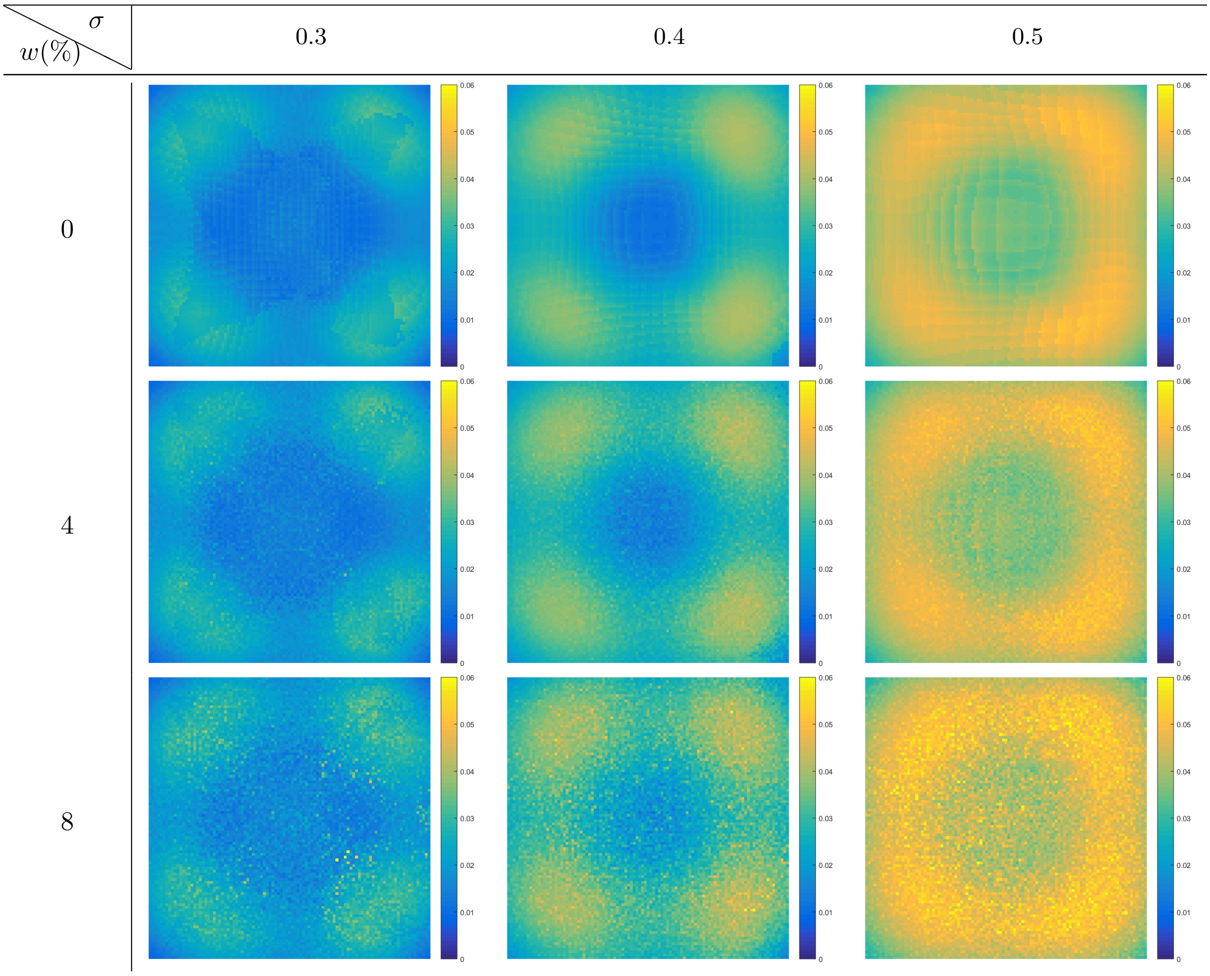


Figure 104: Single source: error-corrected $\epsilon$ maps

### 22.1.5 Discussion

Given phantoms (Fig. 98) represent a plasma situation in tokamak. They are called *single source* phantoms, due to the fact that they only a have single maximum intensity value. Phantoms are built as normalized 2D Gaussian functions (Gaussian blobs):

$$g(x,y) = e^{-\frac{(x-x_{\mathrm{c}})^2+(y-y_{\mathrm{c}})^2}{2\sigma^2}}, \tag{173}$$

where $x_{\mathrm{c}}$, $y_{\mathrm{c}}$ are the center coordinates of the Gaussian blob, $\sigma = 0.3$ is the relative width of the Gaussian blob with respect to the minor radius $r$ of the tokamak. By varying $x_{\mathrm{c}}$ and $y_{\mathrm{c}}$ by a shift value $\Delta r = 0.5$ there are 9 phantoms generated: 1 central and 8 off-central, all organized in a table (Fig. 98). The goal is to reconstruct the given phantoms by using Fourier-Bessel reconstruction algorithm. There are two versions of the available algorithm: the "centered" and the "selective". Both versions are applied to the given phantoms and results are collected in tables, where each entry is associated with particular phantom. "Centered" reconstructions are collected in Fig. 99 and "selective" reconstructions in Fig. 100. One can see that "selective" algorithm outperforms the "centered" algorithm by producing more accurate reconstructions of the input phantoms. Difference is visible by eye and it is confirmed by error assessments ($\chi^2$,$\epsilon$). Tables 13 and 14 entries have much higher values for "centered" reconstructions than respective tables 15 and 16, which proves the "selective" reconstrucion to be a more accurate and useful algorithm.

The next step is to build "centered" reconstructions for every possible center coordinates $-1 < \{\Delta x_{\mathrm{c}}, \Delta y_{\mathrm{c}}\} < 1$ and collect errors ($\chi^2$, $\epsilon$) into corresponding matrices (Fig. 101, Fig. 102). Matrices are constructed for 3 sizes of a phantom ($\sigma = \{0.3, 0.4, 0.5\}$) permutated with 3 levels of signal noise ($\omega = \{0\,\%, 4\,\%, 8\,\%\}$). These maps demonstrate that the least-error reconstructions are concentrated around the center, which is a consequence of two factors:

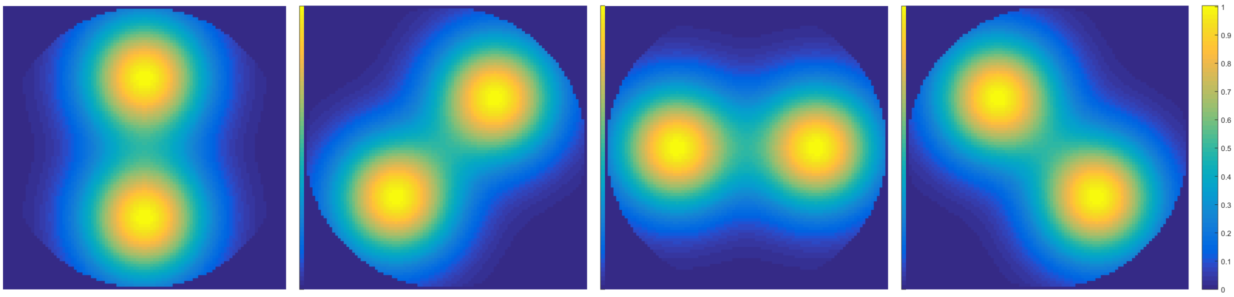
- "centered" algorithm uses basis functions, which are centered

- phantom is partially lost next to the edge of the unit circle (domain edge), therefore the recontruction is inaccurate and generating a big error near the edge

The same approach is applied to the "selective" algorithm. The domain is scanned by a phantom and errors are collected into matrices (Fig. 103, Fig. 104). One observes a huge reduction of error values ($\chi^2$, $\epsilon$) compared to Fig. 101 and Fig. 102. This is even a stronger proof that "selective" algorithm is more accurate and useful. Nevertheless the center of the domain still remains pronounced as the region of lesser error and edges as the higher error. The edge influence is as important as in the "centered" version, but the fact, that the variety of basis functions is bounded to a rectangular region (Fig. 97), becomes important. In general the noise level $\omega$ and phantom size $\sigma$ do not play major role in the topology for any of the error maps, but they do influence the size of the lesser/higher error regions and smoothness respectively.

## 22.2 Double source study

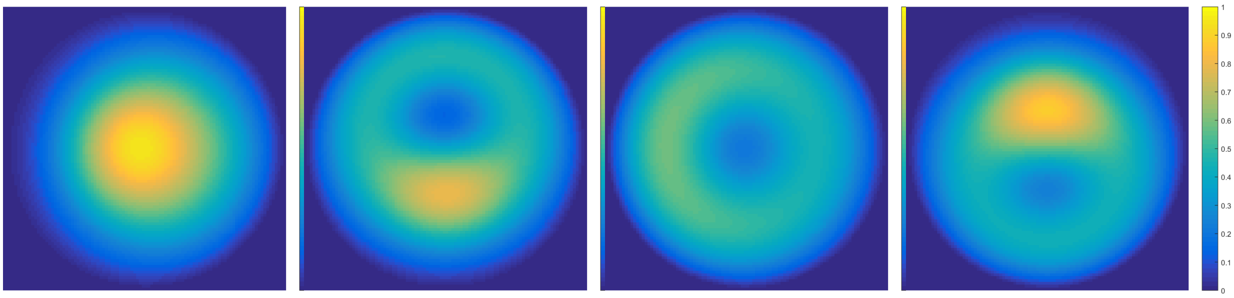
### 22.2.1 Phantoms



vertical diagonal horizontal antidiagonal

Figure 105: Double source: phantoms of Gaussian width $\sigma = 0.3$ and separation $d = 1$

### 22.2.2 Reconstructions

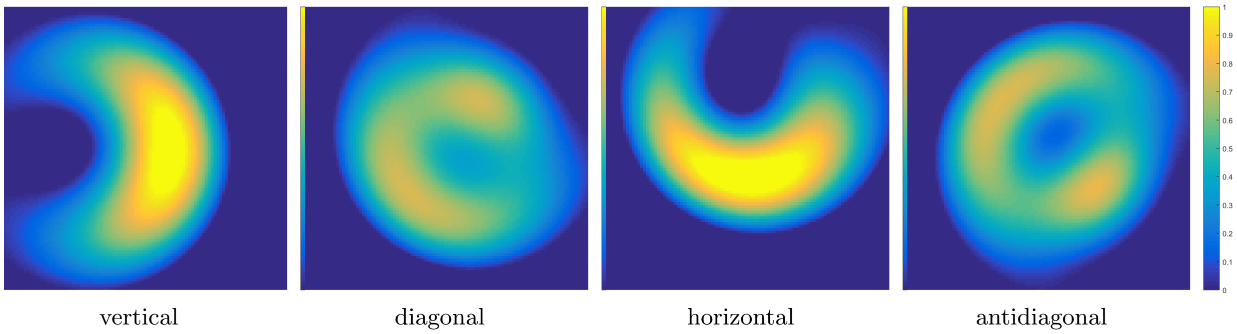


vertical diagonal horizontal antidiagonal

Figure 106: Double source: reconstructions of Fig. 105

| | vertical | diagonal | horizontal | antidiagonal |
|---|---|---|---|---|
| $\epsilon$ | 0.162869 | 0.160606 | 0.163373 | 0.156664 |
| $\chi^2$ | 63.341 | 73.683 | 57.800 | 70.501 |

Table 17: Double source: errors $\chi^2$ and $\epsilon$



vertical diagonal horizontal antidiagonal

Figure 107: Double source error-corrected reconstructions of Fig. 105

| | vertical | diagonal | horizontal | antidiagonal |
|---|---|---|---|---|
| $\epsilon$ | 0.134854 | 0.170407 | 0.146829 | 0.154345 |
| $\chi^2$ | 31.665 | 25.690 | 34.299 | 28.723 |

Table 18: Double source: errors $\chi^2$ and $\epsilon$ after minimization

### 22.2.3 Error maps uncorrected

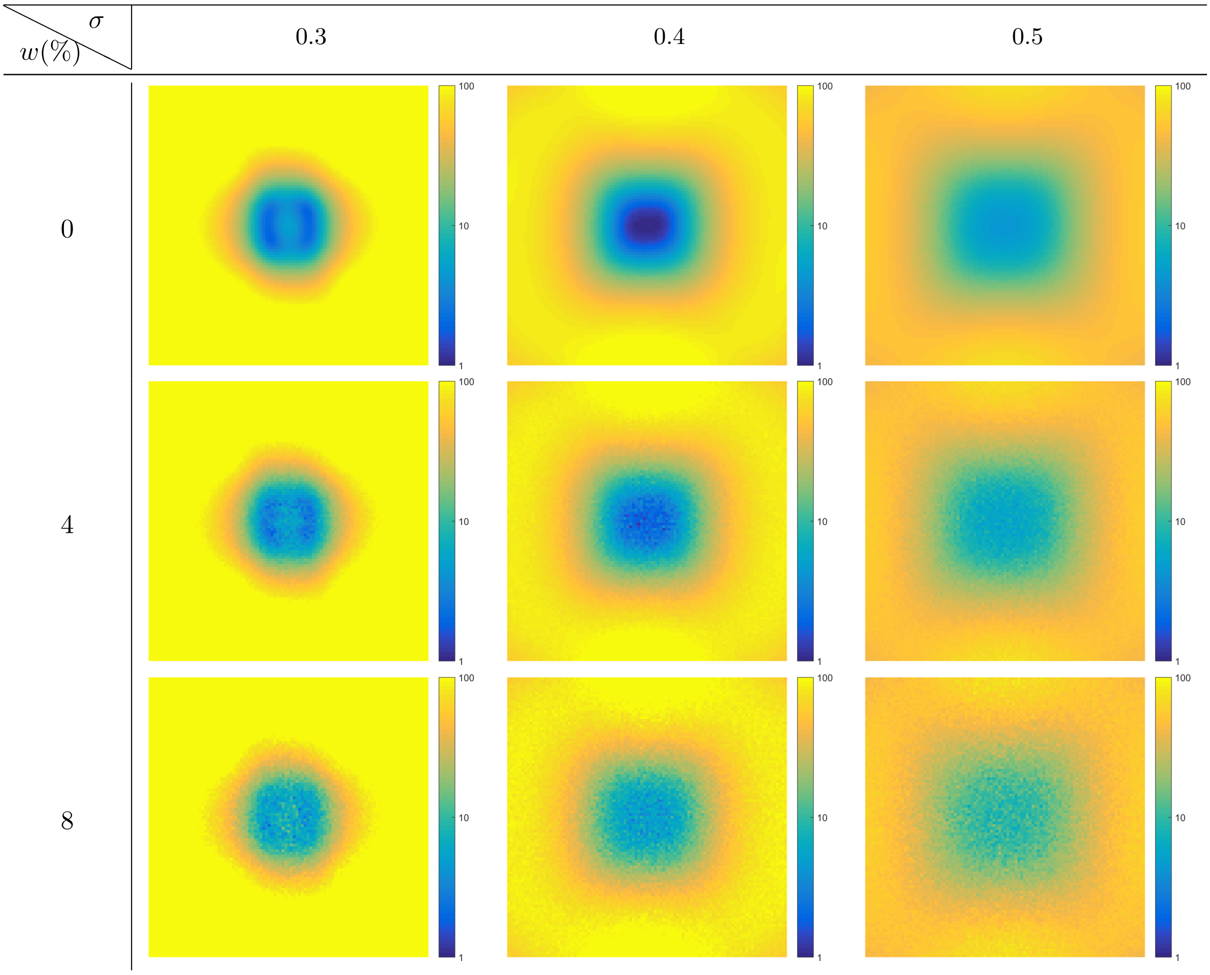


Figure 108: Double source: error $\chi^2$ maps

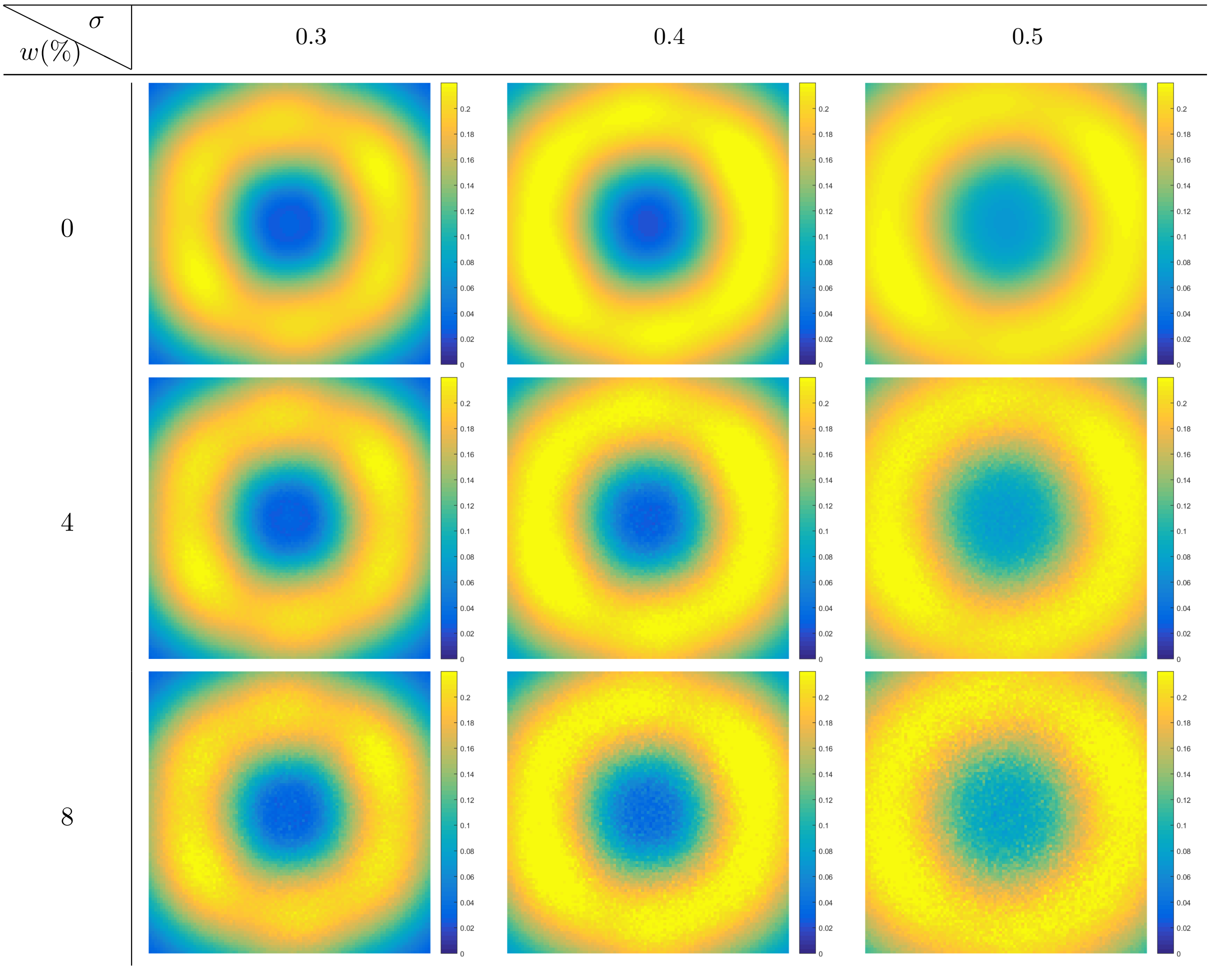


Figure 109: Double source: error $\epsilon$ maps

#### 22.2.4 Error maps corrected

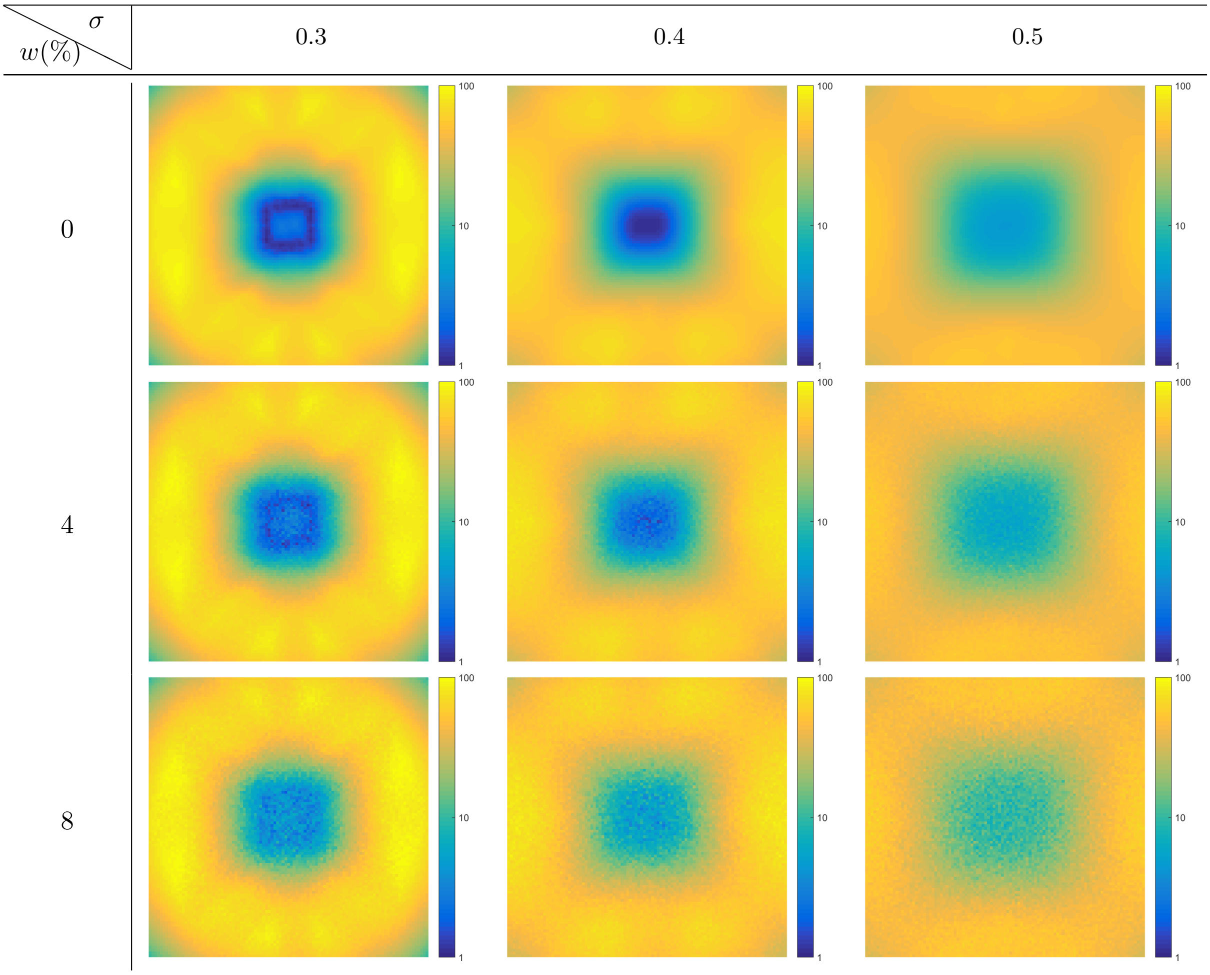


Figure 110: Double source: error-corrected $\chi^2$ maps

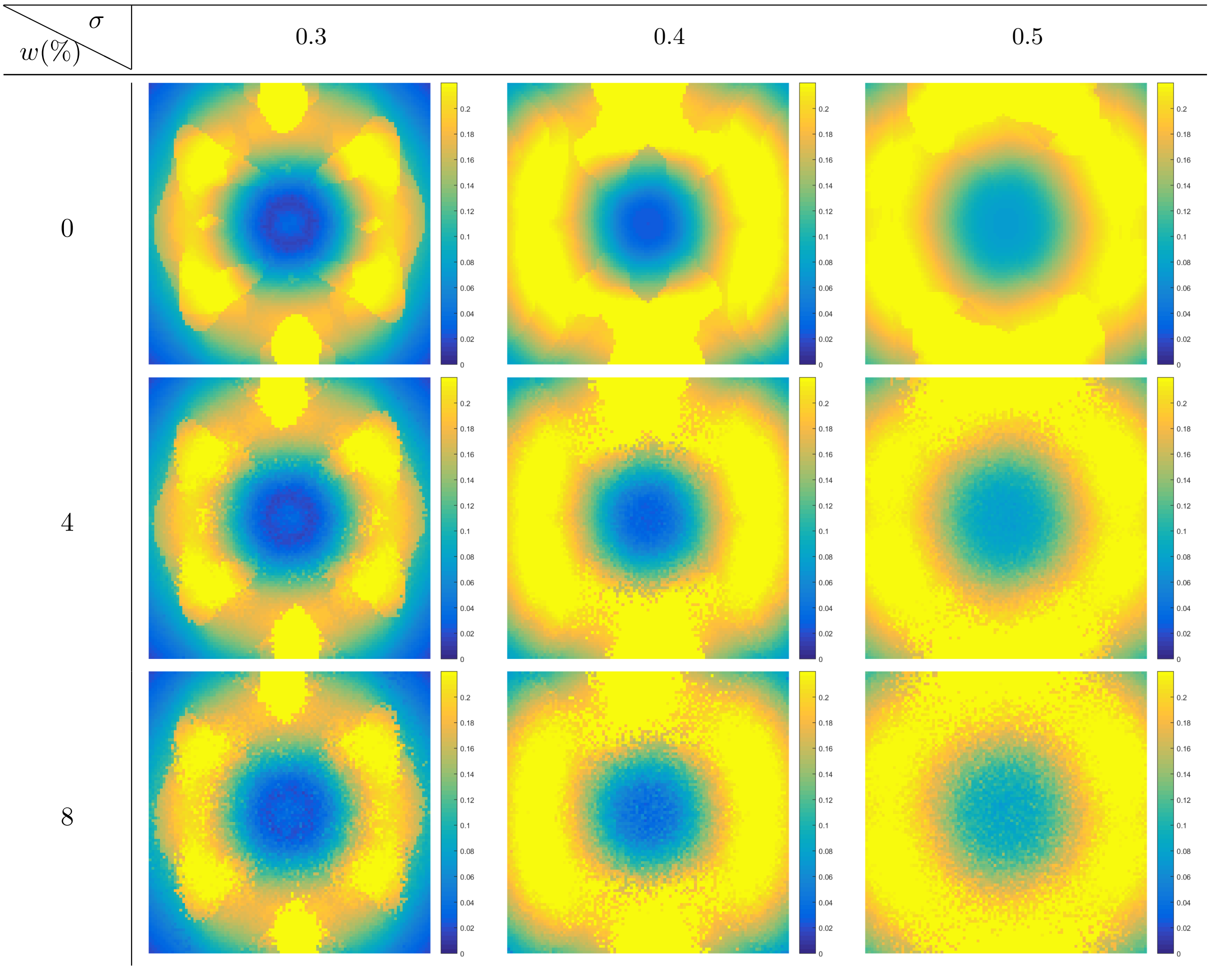


Figure 111: Double source: error-corrected $\epsilon$ maps

### 22.2.5 Discussion

Given phantoms (Fig. 105) represent a plasma situation in tokamak. They are called *double source* phantoms, due to the fact that they have 2 maximum intensity values. Phantoms are built as two normalized 2D Gaussian functions (two Gaussian blobs):

$$g(x,y) = e^{-\frac{(x-x_\mathrm{c})^2+(y-y_\mathrm{c})^2}{2\sigma^2}} + e^{-\frac{(x+x_\mathrm{c})^2+(y+y_\mathrm{c})^2}{2\sigma^2}}, \tag{174}$$

where $x_\mathrm{c}$, $y_\mathrm{c}$ are the center coordinates of one of the Gaussian blobs, $\sigma = 0.3$ is the relative width of the Gaussian blob with respect to the minor radius $r$ of the tokamak. By varying $x_\mathrm{c}$ and $y_\mathrm{c}$ by a shift value $\Delta r = 0.5$ there are 4 phantoms generated: vertical, diagonal, horizontal and antidiagonal, all organized in a table (Fig. 105). The goal is to reconstruct the given phantoms by using Fourier-Bessel reconstruction algorithm. There are two versions of the available algorithm: the "centered" and the "selective". Both versions are applied to the given phantoms and results are collected in tables, where each entry is associated with particular phantom. "Centered" reconstructions are collected in Fig. 106 and "selective" reconstructions in Fig. 107. One can see that neither "centered" nor "selective" algorithm produces a viable reconstruction. Error tables (Tab. 17, Tab. 107) signify a slight improvement in reconstruction quality from "centered" to "elective" algorithm. And maybe by looking very attentively one could at least identify the correct allignment in the case of "selective" algorithm (Fig. 107).

The next step is to build "centered" reconstructions for every possible center coordinates $-1 < \{\Delta x_\mathrm{c}, \Delta y_\mathrm{c}\} < 1$ and collect errors ($\chi^2$, $\epsilon$) into matrices (Fig. 108, Fig. 109). Matrices are constructed for 3 sizes of a phantom ($\sigma = \{0.3, 0.4, 0.5\}$) permutated with 3 levels of signal noise ($\omega = \{0\,\%, 4\,\%, 8\,\%\}$).

These maps demonstrate that the least-error reconstructions are concentrated around the center, which is a consequence of two factors:

- approaching center two Gaussian blobs merge into a single blob, where "centered" algorithm uses basis functions, which are centered
- phantom is partially lost next to the edge of the unit circle (domain edge), therefore the recontruction is inaccurate and generating a big error near the edge

The same approach is applied to the "selective" algorithm. The domain is scanned by a phantom and errors are collected into corresponding matrices (Fig. 110, Fig. 111). As single source error maps, the double source error maps have regions of lesser errors that are located around the center. Maps have additional property, i.e. they are symmetrical with respect to the horizon, because by scanning phantoms one obtains duplicates across the horizontal line. However unlike for single source error maps the double source error maps DO NOT demonstrate any significant improvement between "centered" and "selective" algorithms. In fact $\epsilon$ map even becomes worse for "selective" reconstructions (Fig. 111). This all has to do with the fundamental reconstruction parameters $\{N, L\} = \{2, 2\}$, which represent the angular and radial orders of the basis functions. Apparently the set $\{2, 2\}$ does not work well for reconstructions of double source neither for "centered" nor for "selective" algorithms. As in single source error maps, the noise level $\omega$ and phantom size $\sigma$ do not play major role in the topology for any of the error maps, but they do influence the size of the lesser/higher error regions and smoothness respectively.

## 22.3 Ring source study

### 22.3.1 Phantoms

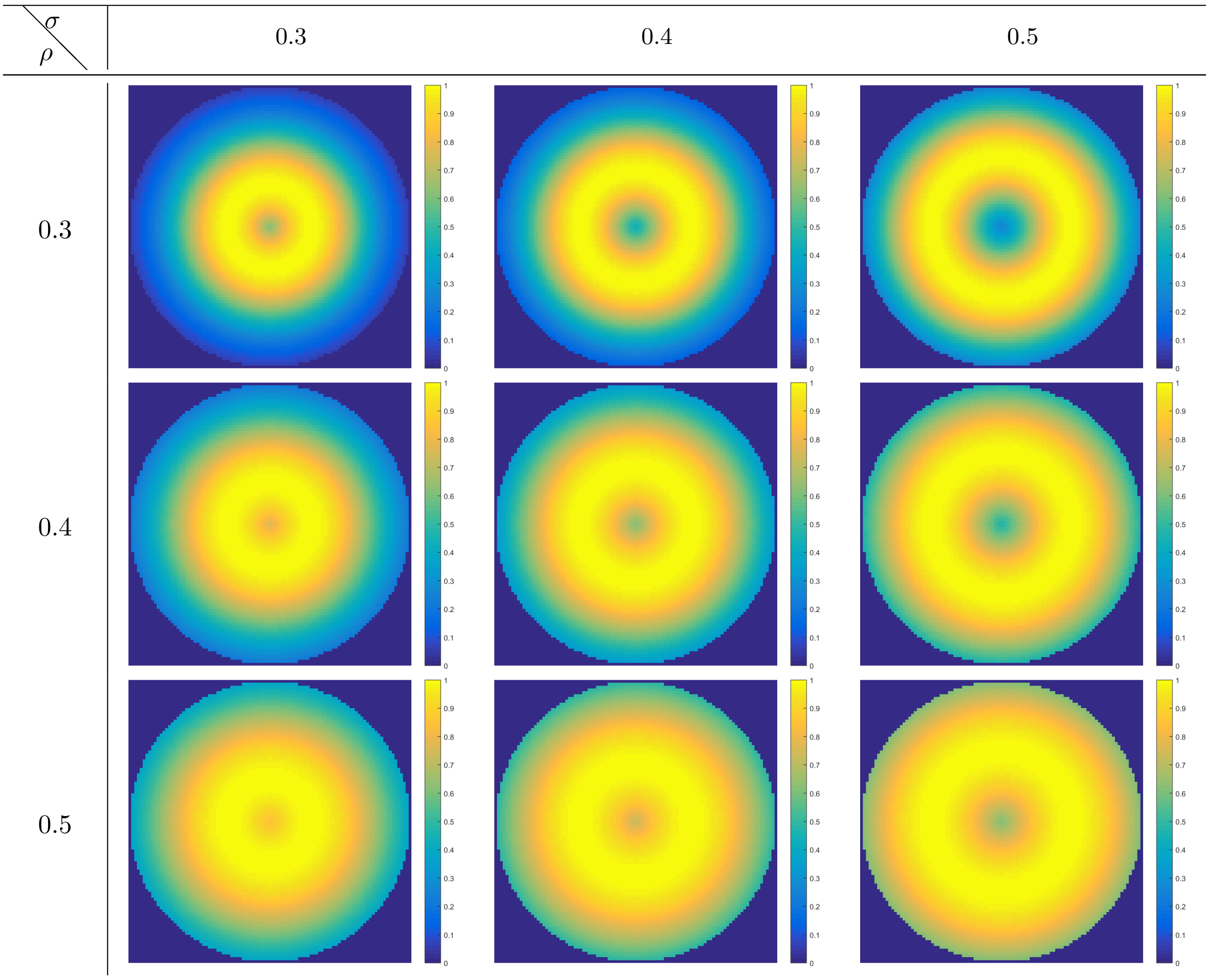


Figure 112: Ring source: phantoms of Gaussian width $\sigma$ and radius $\rho$

### 22.3.2 Reconstructions

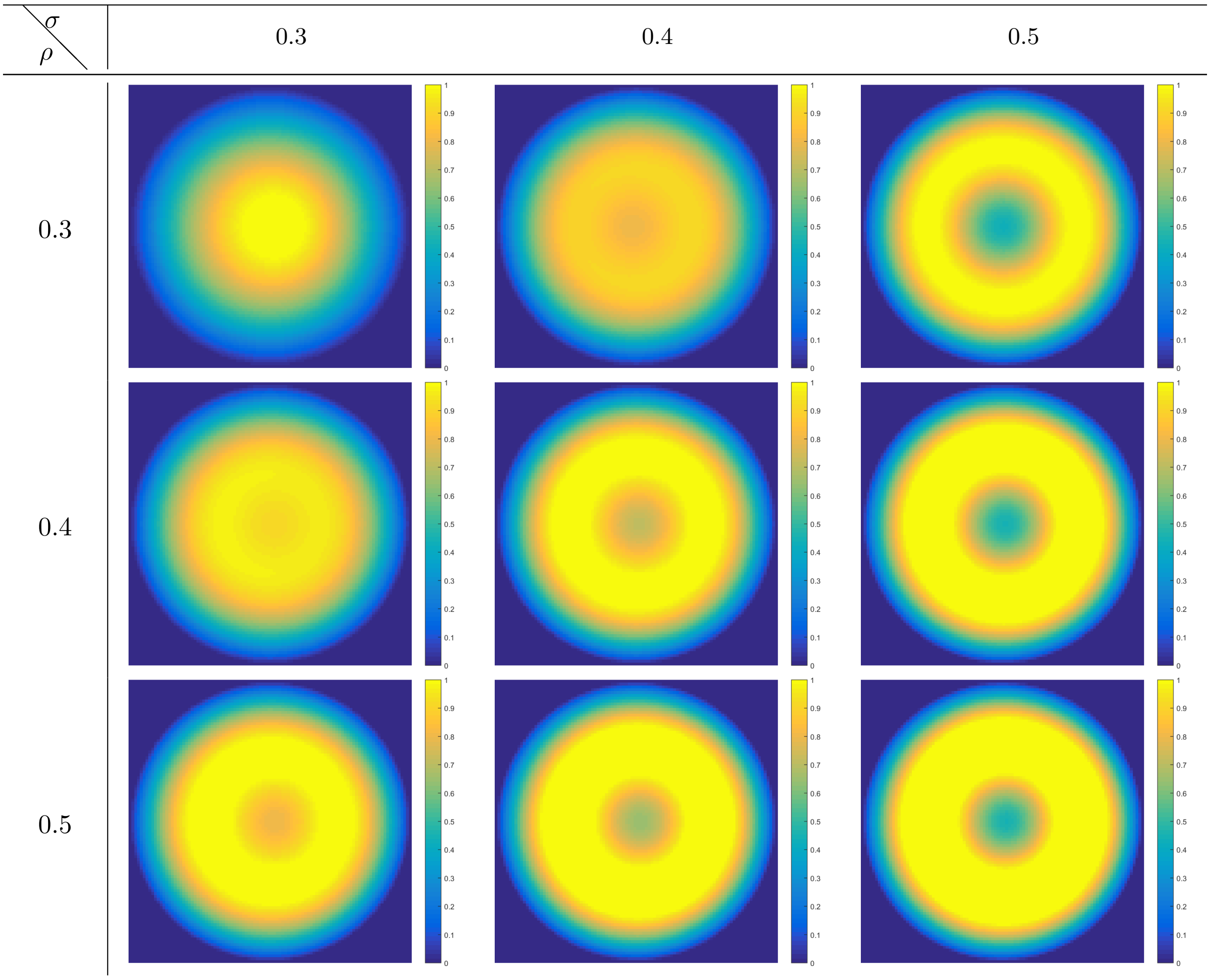


Figure 113: Ring source: reconstructions of Fig. 112

| $\sigma$ $\rho$ | 0.3 | 0.4 | 0.5 |
|---|---|---|---|
| 0.3 | 0.900 | 0.591 | 1.611 |
| 0.4 | 2.298 | 4.569 | 8.416 |
| 0.5 | 7.183 | 10.320 | 14.243 |

Table 19: Ring source: $\chi^2$ error

| $\sigma$ $\rho$ | 0.3 | 0.4 | 0.5 |
|---|---|---|---|
| 0.3 | 0.037745 | 0.043049 | 0.049432 |
| 0.4 | 0.048506 | 0.076824 | 0.117782 |
| 0.5 | 0.097914 | 0.130164 | 0.166185 |

Table 20: Ring source: $\epsilon$ error

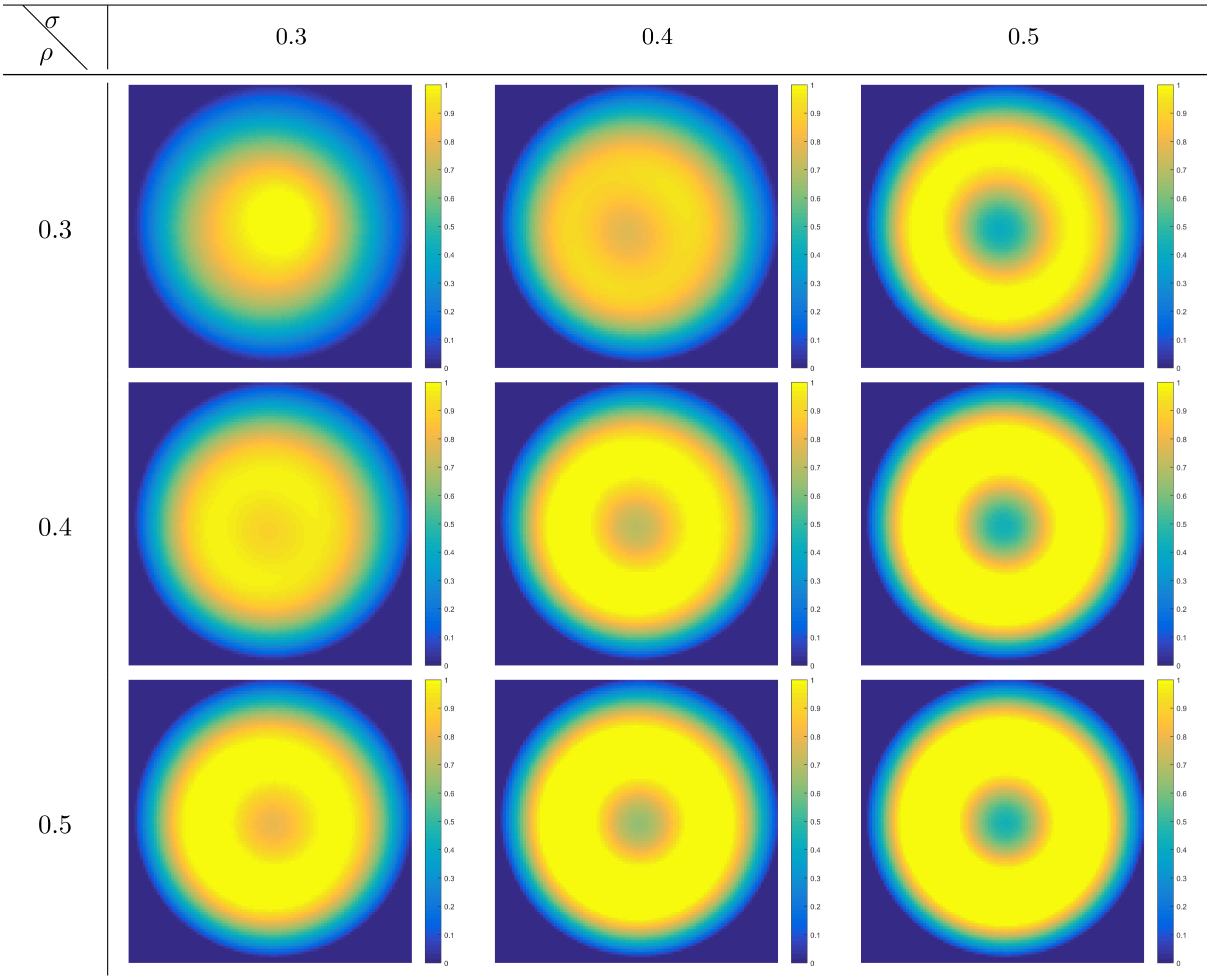


Figure 114: Ring source: error-corrected reconstructions of Fig. 112

| $\sigma$ \ $\rho$ | 0.3 | 0.4 | 0.5 |
|---|---|---|---|
| 0.3 | 0.856 | 0.695 | 1.897 |
| 0.4 | 2.469 | 4.775 | 8.609 |
| 0.5 | 7.304 | 10.420 | 14.290 |

Table 21: Ring source: $\chi^2$ error after minization

| $\sigma$ \ $\rho$ | 0.3 | 0.4 | 0.5 |
|---|---|---|---|
| 0.3 | 0.037954 | 0.044242 | 0.052961 |
| 0.4 | 0.051085 | 0.080433 | 0.122115 |
| 0.5 | 0.101213 | 0.133964 | 0.170331 |

Table 22: Ring source: $\epsilon$ error after minimization

### 22.3.3 Error maps uncorrected

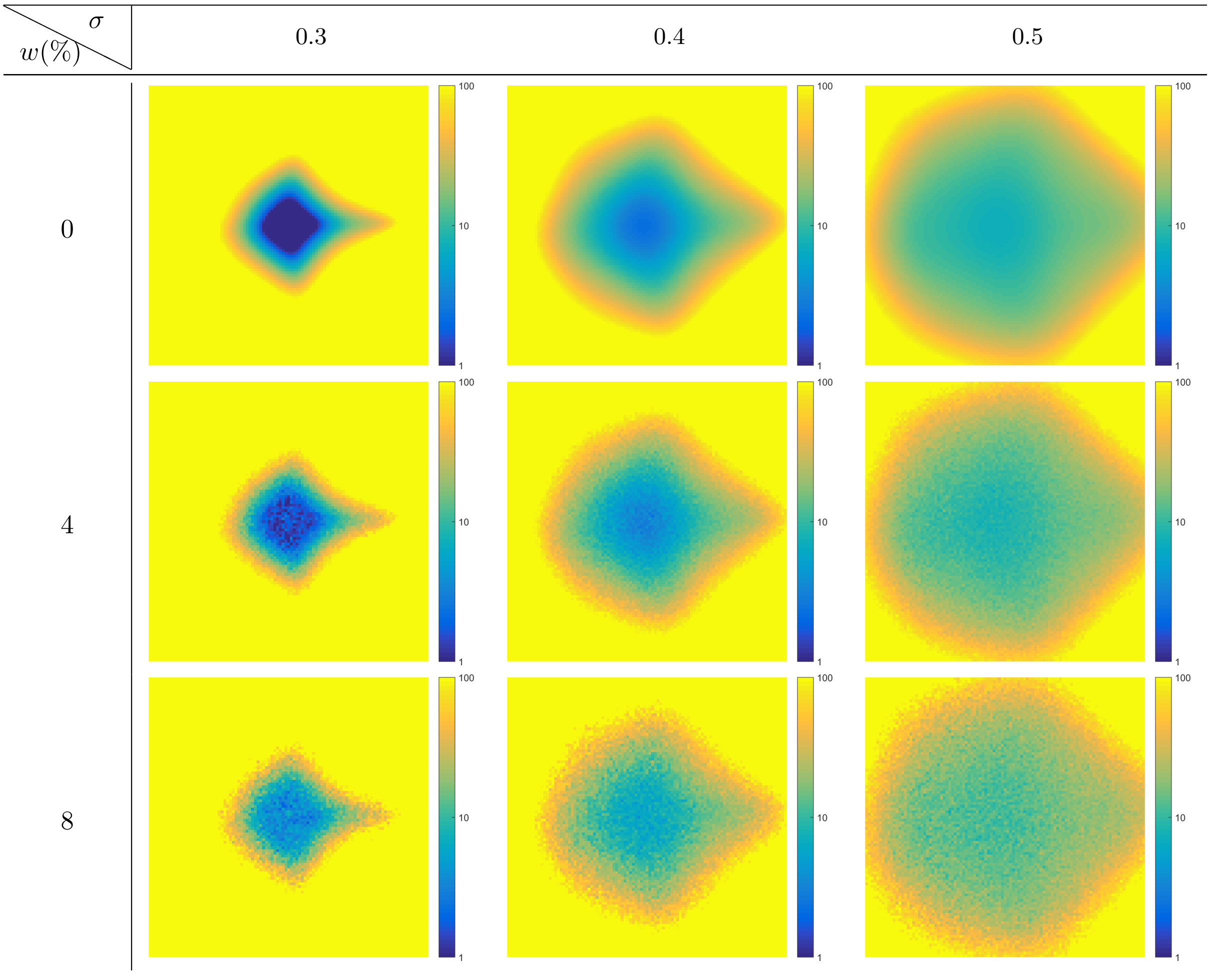


Figure 115: Ring source: error $\chi^2$ maps for ring radius $\rho = 0.3$

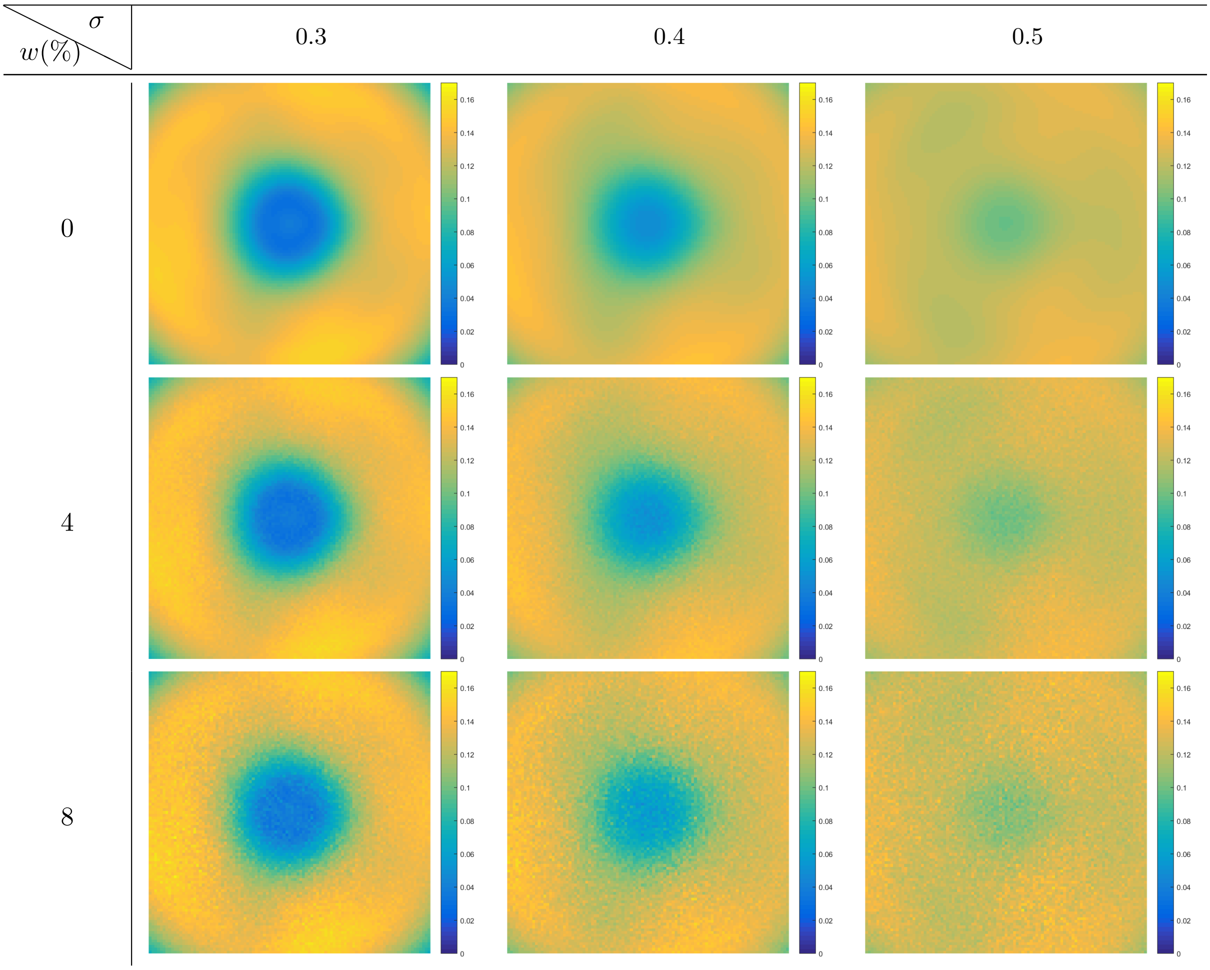


Figure 116: Ring source: error $\epsilon$ maps for ring radius $\rho = 0.3$

### 22.3.4 Error maps corrected

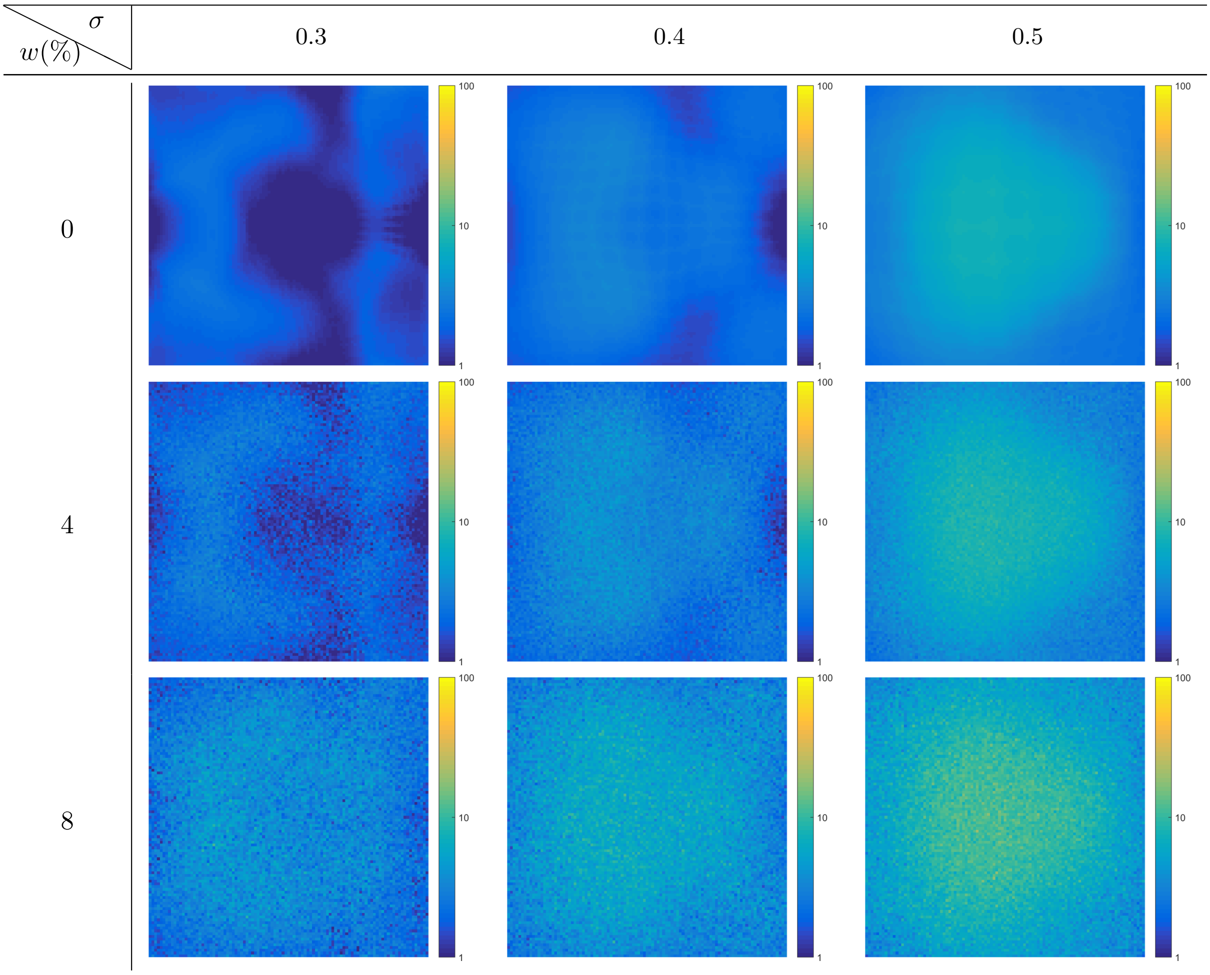


Figure 117: Ring source: error-corrected $\chi^2$ maps for ring radius $\rho = 0.3$

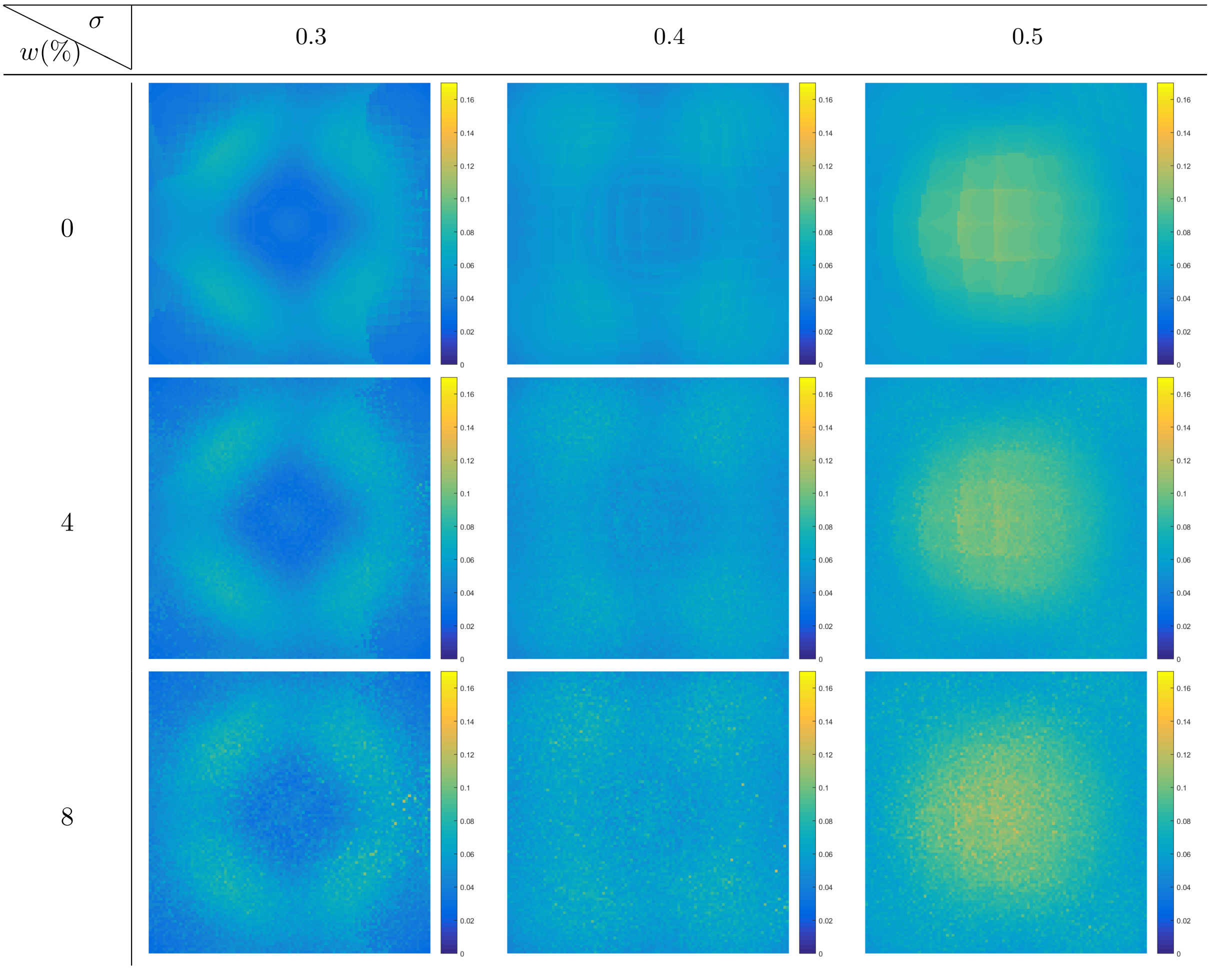


Figure 118: Ring source: error-corrected $\epsilon$ maps for ring radius $\rho = 0.3$

### 22.3.5 Discussion

Given phantoms (Fig. 112) represent a plasma situation in tokamak. They are called *ring source* phantoms, due to the fact that the maximum intensity value follow a ring. Phantoms are built as a normalized 2D Gaussian function stretched along a ring of particular radius $\rho$:

$$g(x, y) = e^{-\frac{\left(\sqrt{(x-x_c)^2+(y-y_c)^2}-\rho\right)^2}{2\sigma^2}}, \tag{175}$$

where $x_c$, $y_c$ are the center coordinates of one of the Gaussian ring, $\sigma$ is the relative width of the Gaussian ring with respect to the minor radius $r$ of the tokamak, and $\rho$ is the radius of the Gausian ring. By varying $\sigma = \{0.3, 0.4, 0.5\}$ and $\rho = \{0.3, 0.4, 0.5\}$ there are 9 phantoms generated all organized in a table (Fig. 112. The goal is to reconstruct the given phantoms by using Fourier-Bessel reconstruction algorithm. There are two versions of the available algorithm: the “centered” and the “selective”. Both versions are applied to the given phantoms and results are collected in tables, where each entry is associated with particular phantom. “Centered” reconstructions are collected in Fig. 113 and “selective” reconstructions in Fig. 114. One can see that both “centered” and “selective” algorthm produces a viable reconstructions. Error tables (Tab. 19, Tab. 20, Tab. 21, Tab. 22) signify an equivalent reconstruction quality for both “centered” and “selective” algorithms. There is no improvement, because “centered” algorithm already is producing a decent result due to the fact that for ring sources only centered phantoms were considered (Fig. 112).

The next step is to build “centered” reconstructions for every possible center coordinates $-1 < \{\Delta x_c, \Delta y_c\} < 1$ and collect errors ($\chi^2$, $\epsilon$) into matrices (Fig. 115, Fig. 116). Matrices are constructed for 3 sizes of a phantom width ($\sigma = \{0.3, 0.4, 0.5\}$) permutated with 3 levels of signal noise

($\omega = \{0\,\%, 4\,\%, 8\,\%\}$). These maps demonstrate that the least-error reconstructions are concentrated around the center, which is a consequence of two factors:

- approaching center the Gaussian ring morphs into single Gaussian blob, where "centered" algorithm uses basis functions, which are centered
- phantom is partially lost next to the edge of the unit circle (domain edge), therefore the recontruction is inaccurate and generating a big error near the edge

The same approach is applied to the "selective" algorithm. The domain is scanned by a phantom and errors are collected into corresponding matrices (Fig. 117, Fig. 118). As single and double source errors maps, the ring source error maps have regions of lesser errors that are located around the center. Like the single source error maps the ring source error maps DO demonstrate significant improvement between "centered" and "selective" algorithms. However the concern is, whether the improvement over error maps does actually mean improved reconstruction. The fact is that some reconstructions may produce low error values, but fail to represent the actual ring phantom. As in single and double source error maps, the noise level $\omega$ and phantom size $\sigma$ do not play major role in the topology for any of the error maps, but they do influence the size of the lesser/higher error regions and smoothness respectively.

## 22.4 Noise sensitivity

### 22.4.1 Without error correction

| $w$ (%) | Single source | | | | | | | | | Double source | | | | Ring source |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | top middle | top right | middle right | bottom right | bottom middle | bottom left | middle left | top left | center | vertical | diagonal | horizontal | antidiagonal | |
| 0 | 903.209 | 1865.89 | 42.8373 | 2034.29 | 976.746 | 263.089 | 1401.22 | 331.68 | 2.51326 | 63.3406 | 73.6831 | 57.7997 | 70.5012 | 1.61055 |
| 1 | 898.736 | 1871.45 | 43.2707 | 2051.47 | 984.277 | 269.015 | 1398.41 | 332.733 | 2.4829 | 63.6852 | 73.1803 | 57.5735 | 70.4036 | 1.67432 |
| 2 | 906.645 | 1865.99 | 43.5477 | 2001.75 | 983.488 | 266.031 | 1430.44 | 318.221 | 2.57138 | 64.2091 | 73.91 | 57.9987 | 70.596 | 1.89271 |
| 3 | 896.911 | 1843.06 | 43.3517 | 1973.96 | 984.195 | 272.669 | 1397.71 | 323.31 | 2.65253 | 64.2082 | 74.2433 | 58.052 | 71.3223 | 2.19073 |
| 4 | 947.604 | 1834.94 | 46.3861 | 2083.87 | 922.002 | 267.561 | 1419.65 | 331.088 | 2.98643 | 64.3232 | 75.0489 | 58.6638 | 71.0321 | 2.71898 |
| 5 | 863.573 | 1960.49 | 47.1348 | 1961.4 | 992.276 | 292.641 | 1379.52 | 368.819 | 2.92732 | 63.0335 | 75.0619 | 60.4794 | 71.151 | 2.92719 |
| 6 | 886.013 | 1854.28 | 50.567 | 2096.38 | 1069.27 | 252.134 | 1345.2 | 408.786 | 2.99375 | 64.6077 | 74.4574 | 60.2615 | 69.9106 | 4.10938 |
| 7 | 1113.98 | 1899.85 | 50.4423 | 2099.12 | 946.298 | 321.581 | 1435.42 | 388.669 | 3.26266 | 68.0997 | 78.2137 | 60.7069 | 73.2834 | 4.64509 |
| 8 | 895.114 | 2068.98 | 51.4977 | 1980.16 | 961.964 | 340.87 | 1485.69 | 499.901 | 4.14575 | 67.4697 | 77.6172 | 59.1681 | 71.6084 | 6.3925 |
| 9 | 964.509 | 1795.72 | 55.9847 | 2090.54 | 1047.64 | 337.203 | 1484.96 | 300.76 | 5.27496 | 63.6181 | 76.828 | 60.8666 | 75.7226 | 7.70442 |
| 10 | 1113 | 1922.08 | 55.4709 | 2118.93 | 1007.36 | 324.3 | 1411.69 | 388.131 | 5.78133 | 70.4919 | 82.7168 | 63.5018 | 71.5802 | 8.37355 |

Table 23: Absolute error $\chi^2$ dependence on noise level $w$

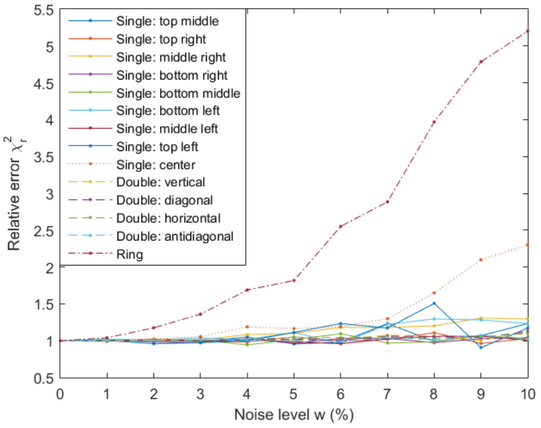


Figure 119: Relative error $\chi^2_{\mathbf{r}}$ dependence on noise level $w$

| $w$ (%) | Single source | | | | | | | | | Double source | | | | Ring source |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | top middle | top right | middle right | bottom right | bottom middle | bottom left | middle left | top left | center | vertical | diagonal | horizontal | antidiagonal | |
| 0 | 0.0870748 | 0.0845998 | 0.0818168 | 0.0839798 | 0.0882414 | 0.0852287 | 0.0865794 | 0.0860397 | 0.0149677 | 0.162869 | 0.160606 | 0.163373 | 0.156664 | 0.0494323 |
| 1 | 0.0870605 | 0.0846385 | 0.0818861 | 0.083997 | 0.0883451 | 0.0852587 | 0.0866041 | 0.0858613 | 0.0148629 | 0.162868 | 0.160746 | 0.163355 | 0.156637 | 0.0493715 |
| 2 | 0.086944 | 0.0847182 | 0.0817859 | 0.0841048 | 0.0885031 | 0.0852415 | 0.0865842 | 0.086163 | 0.0149738 | 0.16335 | 0.160632 | 0.163377 | 0.156594 | 0.0499277 |
| 3 | 0.0870342 | 0.0845806 | 0.0820201 | 0.0838602 | 0.0883774 | 0.0850903 | 0.086035 | 0.085884 | 0.0150982 | 0.163133 | 0.160915 | 0.163471 | 0.157094 | 0.0529427 |
| 4 | 0.087208 | 0.0850366 | 0.0815077 | 0.0839234 | 0.0884423 | 0.0849498 | 0.0862629 | 0.0861821 | 0.0155293 | 0.162505 | 0.159783 | 0.164484 | 0.157168 | 0.0521546 |
| 5 | 0.0868571 | 0.0847666 | 0.0818985 | 0.0840486 | 0.08792 | 0.0849377 | 0.0864034 | 0.0852 | 0.0154093 | 0.162859 | 0.160813 | 0.163552 | 0.157015 | 0.0550622 |
| 6 | 0.0873694 | 0.0840539 | 0.0820121 | 0.0841398 | 0.0890027 | 0.0863647 | 0.0868311 | 0.0854237 | 0.0156539 | 0.162522 | 0.158938 | 0.161973 | 0.157259 | 0.0576544 |
| 7 | 0.0889668 | 0.0850709 | 0.0821172 | 0.0841222 | 0.0879352 | 0.0858781 | 0.0875371 | 0.08608 | 0.0159416 | 0.163119 | 0.162335 | 0.164361 | 0.156206 | 0.0592532 |
| 8 | 0.0873463 | 0.0861951 | 0.0831798 | 0.0833738 | 0.0884308 | 0.0858105 | 0.0869494 | 0.0851426 | 0.0168713 | 0.164214 | 0.160976 | 0.1643 | 0.157328 | 0.059386 |
| 9 | 0.0879164 | 0.0844488 | 0.0820899 | 0.0837114 | 0.0894671 | 0.0861666 | 0.0882865 | 0.0878058 | 0.019114 | 0.161938 | 0.16234 | 0.164179 | 0.157117 | 0.0624621 |
| 10 | 0.0881358 | 0.0856238 | 0.0819846 | 0.0841646 | 0.0881514 | 0.0852327 | 0.0868231 | 0.0881076 | 0.0182347 | 0.165289 | 0.161292 | 0.162888 | 0.157285 | 0.0658472 |

Table 24: Absolute error $\epsilon$ dependence on noise level $w$

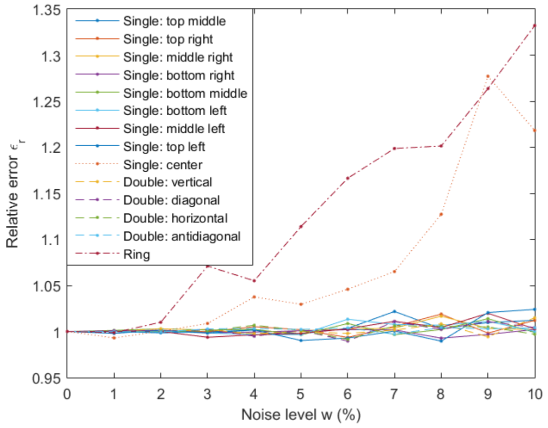


Figure 120: Relative error $\epsilon_{\mathrm{r}}$ dependence on noise level $w$

### 22.4.2 With error correction

| $w$ (%) | Single source | | | | | | | | | Double source | | | | Ring source |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | top middle | top right | middle right | bottom right | bottom middle | bottom left | middle left | top left | center | vertical | diagonal | horizontal | antidiagonal | |
| 0 | 0.598505 | 0.454287 | 0.266373 | 0.524408 | 0.61529 | 0.484698 | 0.581441 | 0.588493 | 1.25376 | 31.6647 | 25.69 | 34.299 | 28.7232 | 1.89749 |
| 1 | 0.610067 | 0.436997 | 0.284828 | 0.564275 | 0.652617 | 0.502415 | 0.584499 | 0.615128 | 1.27504 | 31.6932 | 25.792 | 34.2811 | 28.8461 | 1.96569 |
| 2 | 0.640947 | 0.502753 | 0.386084 | 0.620323 | 0.653271 | 0.616744 | 0.583968 | 0.737993 | 1.26118 | 31.7318 | 25.8021 | 34.3993 | 28.8383 | 2.11004 |
| 3 | 0.716593 | 0.638615 | 0.394535 | 0.695201 | 0.759283 | 0.769987 | 0.655835 | 0.851201 | 1.34748 | 32.0206 | 25.7488 | 34.2018 | 29.0502 | 2.29641 |
| 4 | 0.732812 | 0.730698 | 0.4679 | 0.873744 | 0.820756 | 0.872399 | 0.798548 | 0.917729 | 1.66754 | 31.5991 | 26.379 | 34.1857 | 28.87 | 3.01145 |
| 5 | 0.877213 | 1.01231 | 0.732183 | 1.07574 | 1.02544 | 1.25163 | 1.04212 | 1.14717 | 1.81051 | 32.1431 | 26.4057 | 34.7973 | 29.4932 | 3.36194 |
| 6 | 1.31444 | 1.16289 | 0.975425 | 1.27298 | 1.42215 | 1.41729 | 1.10475 | 1.42848 | 1.72226 | 32.9652 | 27.638 | 34.2765 | 29.2313 | 4.20985 |
| 7 | 1.29313 | 1.3242 | 1.25947 | 1.43485 | 1.16343 | 1.63118 | 1.43771 | 1.62636 | 2.24393 | 32.8365 | 27.1661 | 35.1759 | 30.1982 | 5.11257 |
| 8 | 1.50085 | 1.65003 | 1.59793 | 1.88699 | 1.73704 | 1.6927 | 2.00267 | 1.7007 | 2.45486 | 33.7243 | 27.9819 | 33.9518 | 30.4807 | 5.81717 |
| 9 | 1.78654 | 1.75505 | 1.77452 | 2.16928 | 2.37497 | 2.27552 | 2.07153 | 2.26433 | 3.06666 | 32.9963 | 29.4434 | 35.26 | 30.4972 | 7.31008 |
| 10 | 2.50587 | 2.23758 | 1.91485 | 2.51461 | 2.98595 | 2.62515 | 2.97408 | 2.39233 | 3.82539 | 33.4925 | 28.4903 | 34.6899 | 32.0825 | 8.2366 |

Table 25: Absolute error $\chi^2$ dependence (after minimization) on noise level $w$

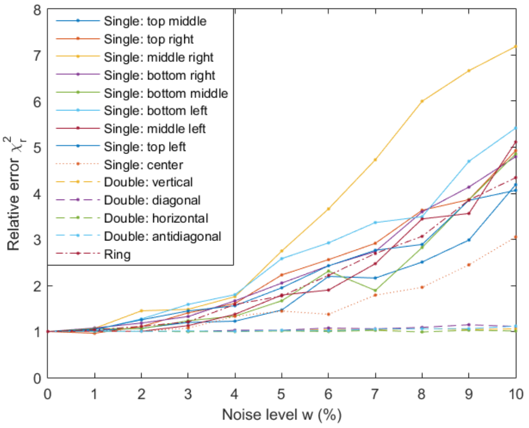


Figure 121: Relative error $\chi^2_{\mathbf{r}}$ (after minimization) dependence on noise level $w$

| $w$ (%) | Single source | | | | | | | | | Double source | | | | Ring source |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | top middle | top right | middle right | bottom right | bottom middle | bottom left | middle left | top left | center | vertical | diagonal | horizontal | antidiagonal | |
| 0 | 0.0156332 | 0.0125957 | 0.0109654 | 0.0125375 | 0.0155448 | 0.014122 | 0.0106981 | 0.0141828 | 0.0155316 | 0.134854 | 0.170407 | 0.146829 | 0.154345 | 0.0529608 |
| 1 | 0.0141772 | 0.0132074 | 0.0109248 | 0.0125412 | 0.0144898 | 0.0142052 | 0.0107239 | 0.0142442 | 0.0155572 | 0.134984 | 0.170614 | 0.150411 | 0.154163 | 0.053505 |
| 2 | 0.0139132 | 0.0137825 | 0.0112767 | 0.0145222 | 0.0118319 | 0.014348 | 0.0114324 | 0.0145418 | 0.0156582 | 0.135028 | 0.170019 | 0.152794 | 0.154895 | 0.054102 |
| 3 | 0.0139072 | 0.0149738 | 0.0109781 | 0.0145152 | 0.0138791 | 0.0145259 | 0.0116896 | 0.0152333 | 0.015511 | 0.134833 | 0.171275 | 0.150485 | 0.156035 | 0.0562214 |
| 4 | 0.0134111 | 0.0158239 | 0.0118789 | 0.0168488 | 0.0144625 | 0.0151553 | 0.0129227 | 0.0148635 | 0.0161533 | 0.135391 | 0.169153 | 0.15169 | 0.157741 | 0.0577382 |
| 5 | 0.0153736 | 0.018807 | 0.0118747 | 0.0166448 | 0.0166218 | 0.0158035 | 0.0128408 | 0.0154526 | 0.0151601 | 0.135307 | 0.168986 | 0.15382 | 0.153913 | 0.0574798 |
| 6 | 0.0156658 | 0.0173288 | 0.0133416 | 0.0187206 | 0.016303 | 0.0176111 | 0.0145436 | 0.0177127 | 0.0160856 | 0.137405 | 0.165635 | 0.147826 | 0.155316 | 0.0562758 |
| 7 | 0.0154989 | 0.0164929 | 0.0129823 | 0.0196783 | 0.0150366 | 0.0194726 | 0.0140592 | 0.0195469 | 0.0166343 | 0.137663 | 0.165414 | 0.154099 | 0.150807 | 0.0640096 |
| 8 | 0.0163878 | 0.0169108 | 0.0142564 | 0.0295495 | 0.0164983 | 0.0196152 | 0.0167254 | 0.0185986 | 0.0187141 | 0.137506 | 0.162607 | 0.149405 | 0.151122 | 0.065363 |
| 9 | 0.0165464 | 0.0209551 | 0.0163559 | 0.0227132 | 0.0189166 | 0.0219725 | 0.0164735 | 0.0216768 | 0.0200337 | 0.135549 | 0.162698 | 0.150537 | 0.150382 | 0.0710918 |
| 10 | 0.0181486 | 0.0225853 | 0.015303 | 0.0308818 | 0.0202477 | 0.0220178 | 0.0204765 | 0.023726 | 0.0193953 | 0.138851 | 0.15647 | 0.154997 | 0.145732 | 0.0690138 |

Table 26: Absolute error $\epsilon$ (after minimization) dependence on noise level $w$

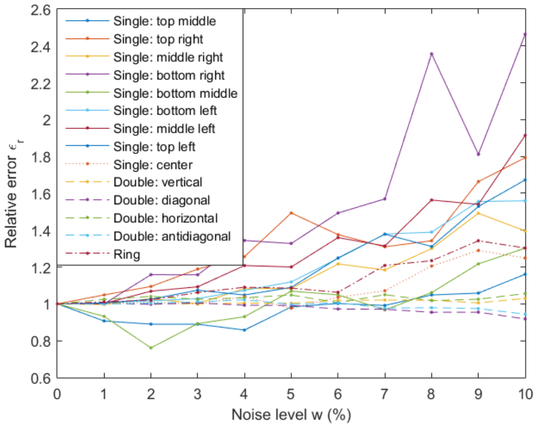


Figure 122: Relative error $\epsilon_{\mathrm{r}}$ (after minimization) dependence on noise level $w$

### 22.4.3 Discussion

Noise sensitivity is a study to observe, how reconstruction-generated error reacts to various noise levels in signal. Single, double and ring sources were gathered into unified studies to observe error dependence for "centered" algorithm (Tab. 23, Tab. 24) and "selective" algorithm (Tab. 25, Tab. 26) respectively. The obtained data was normalized to the data of zero noise level $\omega = 0\,\%$ and plotted in Fig. 119 and Fig. 120 for "centered" algorithm and Fig. 121 and Fig. 122 for "selective" algorithm. In the case of "centered" reconstructions, the highest sensitivity is pronounced by a single source central and ring phantoms for both $\chi^2$ and $\epsilon$. This can be explained, that both of those phantoms are located in the center of domain, where the "centered" algorithm works the best. Therefore the noise presense in the signal actually affects the reconstruction quality. Meanwhile all other phantoms are not affected much by increasing noise levels, because those reconstructions are already bad in the first place, due to "central" algorithm not being able to process well the off-centered and double source phantoms. In the case of "selective" reconstructions, all the phantoms demonstate increased sensitivity towards higher noise levels. This means, that "selective" algorithm improves reconstructions up to the point, where noise becomes leading factor in error generation and not the off-centricity of the phantom.

## 22.5 Performance study

The complete computational pipeline of "selective" reconstrucion algorithm is comprised of 6 steps, and they are integrated into MARTe framework [31]. One is interested in the performance (duration) of each step and the complete time. The performance is measured with respect to two parameters: the size of reconstruction map and the size of the reconstruction grid:

1. Copying the signal data $\tilde{\mathbf{f}}$ from CPU RAM to GPU RAM. The duration does not depend neither on map size nor grid size, because arriving signal $\tilde{\mathbf{f}}$ is always the same size, therefor copy time does not change (Fig. 123).

2. Calculating tomographic coeficient vector $\mathbf{a}$ (Eq. 161). The duration does not depend on map size, but exponentially depends on grid size (Fig. 124), because the set of pseudo-inverse contribution matrices $\{\mathbf{C}^{+}\}$ grows quadratically with grid size and needs more processing. It grows, until reaches flat ceiling. One suspects that at the flat region GPU card engages more than one streaming processor, therefore the maximum timing is the duration of the busiest processor.

3. Calculating back-projected signals $\mathbf{f}$ (Eq. 156). The duration exponentially depends on the grid size (Fig. 125), because the set of pseudo-inverse contribution matrices $\{\mathbf{C}^{+}\}$ grows quadratically with grid size and needs more processing. It is the heaviest part of the algorithm.

4. Calculating $\chi^2_{\mathrm{min}}$ (Eq. 171) and $\arg\chi^2_{\mathrm{min}}$ (Eq. 172). The duration exponentially depends on the grid size (Fig 126), because the set of pseudo-inverse contribution matrices $\{\mathbf{C}^{+}\}$ grows quadratically with grid size and needs more processing.

5. Performing reconstruction **g** (Eq. 163). The timing depends on the map size (Fig. 127), because the map size is the size of reconstructed picture.
6. Copying the image data **g** from GPU RAM to CPU RAM. The duration depends on the map size, because it defines the amount of data to be copied (Fig. 128).

Complete duration is plotted in (Fig. 129). It depends both on map size and grid size, but grid size is a stronger parameter. In order to stay within the duration of 100 $\mu$s (typical time for active tokamak control in real-time) the algorithm has to process small size reconstrucions on a error-correction grid of a small size. A good choicee would be not to go above map size = 100 and grid size = 3.

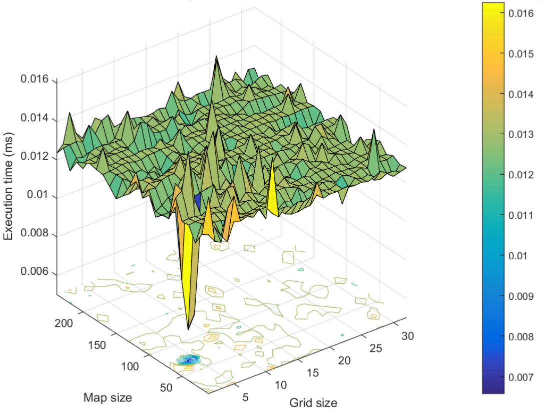


Figure 123: Step 1. $\tilde{\mathbf{f}}$: CPU RAM → GPU RAM

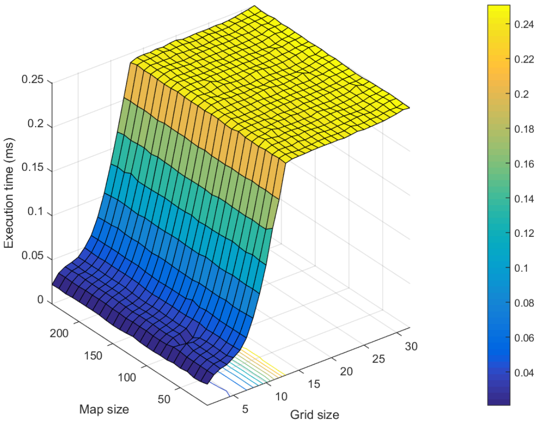


Figure 124: Step 2. Calculation of **a**

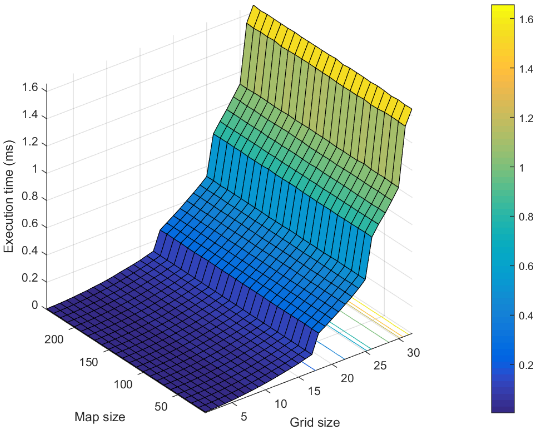


Figure 125: Step 3. Calculation of **f**

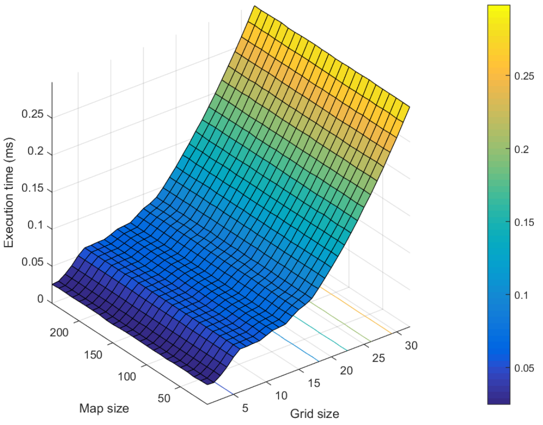


Figure 126: Step 4. Calculation of $\chi^2_{\min}$ and $\arg \chi^2_{\min}$

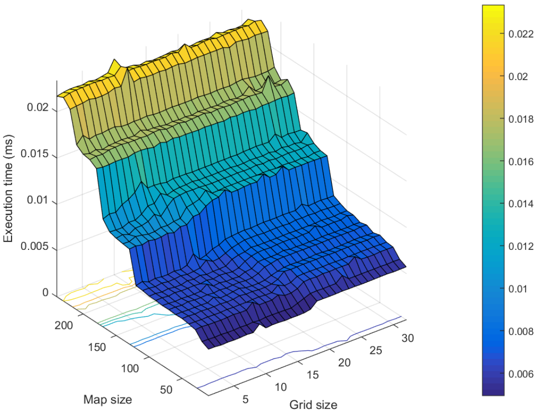


Figure 127: Step 5. Performing reconstruction **g**

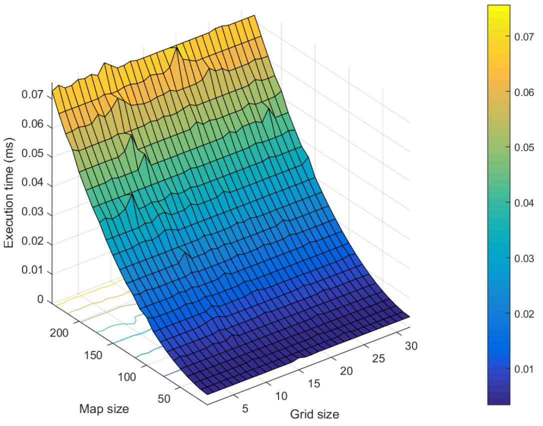


Figure 128: Step 6. **g**: GPU RAM → CPU RAM

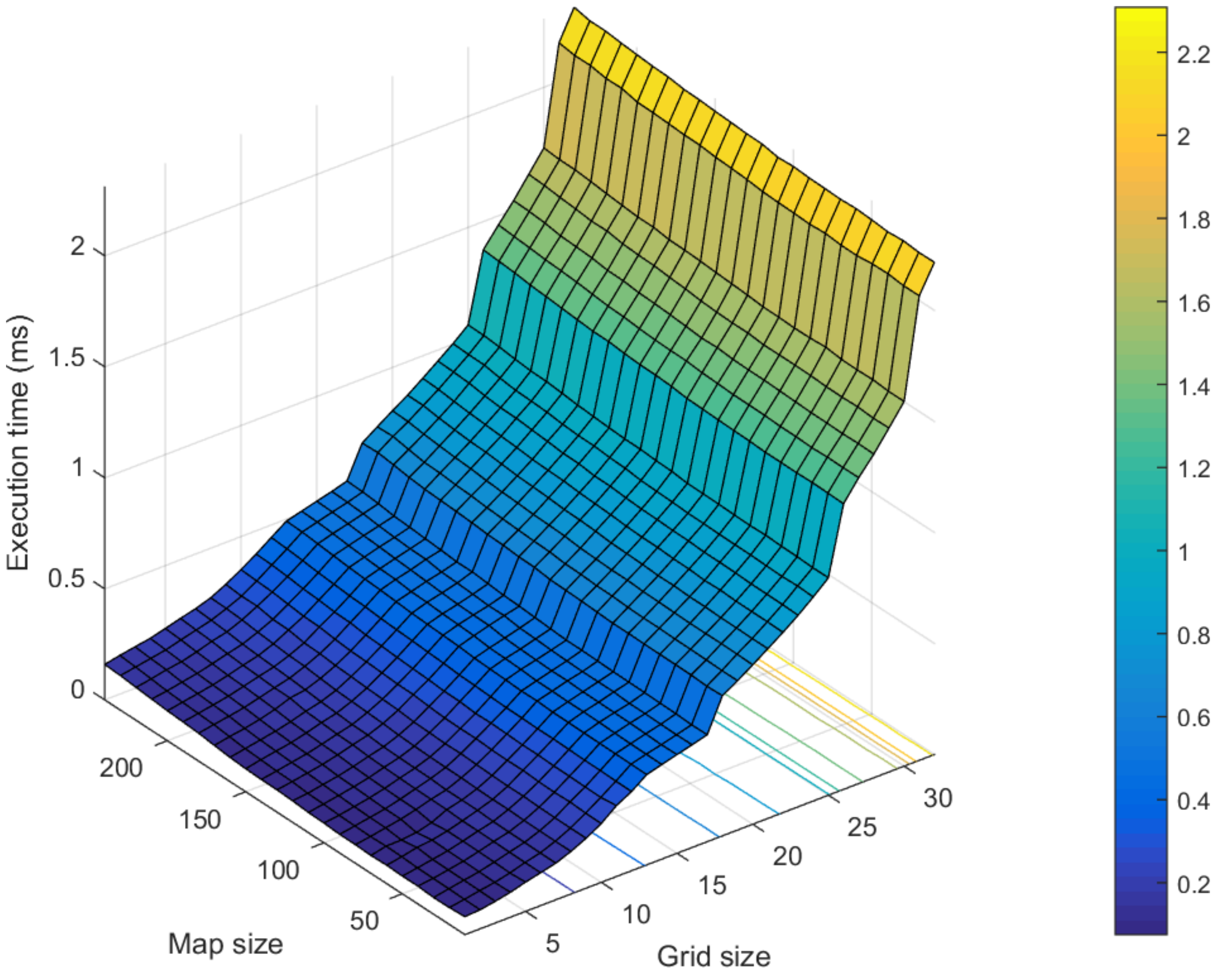


Figure 129: Total time of all the algorithm steps

# 23 Conclusions

- Selective reconstruction proves to be effective for single source and ring source phantoms.
- Both centered and selective reconstructions are ineffective for the double source phantoms, because the chosen angular and radial orders $\{N, L\} = \{2, 2\}$ are not suitable for double source phantoms.
- By using "centered" algorithm the sensitivity to noise is only relevant to centered phantoms such as centered single source and ring source phantoms.
- By using "selective" algorithm the sensitivity to noise increases for all phantoms, which is signature of improved reconstructions, because only good reconstruction are sensitive to noise.
- Code performance heavily depends on the size of the error correction grid and weakly depends on the size of the reconstruction image.
- In order to fit under $100\,\mu$s (characteristic time scale in real-time control for ISTTOK) one can use maximum size of $3 \times 3$ error correction grid and up to size $150 \times 150$ reconstruction image. One meets the universal omnipresent principle of quality versus quantity. If one wants a high quality reconstruction (selection grid size is big) with properly selected basis functions, one has to pay penalty for performance.
- this GPU-based algorithm is a substantial improvement with respect to the former tomography algorithm at ISTTOK developed by [67] in terms of image resolution and reconstruction speed. It was only able to produce images of $15 \times 15$ images and "selective" reconstruction was not an option.
- this GPU-based algorithm is the first real-time GPU code integrated in the MARTe framework.

Nvidia GeForce GTX 480 was used for real-time tomography codes. GeForce GTX 480 is a graphics card originally designed for gaming purposes built on Fermi architecture and released in 2010. It runs 700 MHz clock with 1.5 GB of RAM and 48 kB of shared memory.

# Appendices

## A Appendix

**Fourier-Bessels Integration of $g(\mathbf{k})$ over $\theta$**

By inserting Eq. 137 into Eq. 140 one gets:

$$g(\mathbf{k}) = \int\limits_0^{2\pi} \mathrm{d}\theta \int\limits_0^{\infty} \sum_{n=0}^{\infty} [g_n^{\mathrm{c}}(r)\cos(n\theta) + g_n^{\mathrm{s}}(r)\sin(n\theta)] e^{ikr\cos(\phi-\theta)} r\,\mathrm{d}r \tag{A.1}$$

By defining $\beta = \phi - \theta \longrightarrow \mathrm{d}\beta = -\mathrm{d}\theta$ Eq. A.1 becomes:

$$g(\mathbf{k}) = \sum_{n=0}^{\infty} \int\limits_0^{\infty} r\,\mathrm{d}r \int\limits_{\phi-2\pi}^{\phi} [g_n^{\mathrm{c}}(r)\cos(n[\phi-\theta]) + g_n^{\mathrm{s}}(r)\sin(n[\phi-\theta])] e^{ikr\cos\beta}\,\mathrm{d}\beta \tag{A.2}$$

Using trigonometric identities

$$\cos(n[\phi-\beta]) = \cos(n\phi)\cos(n\beta) + \sin(n\phi)\sin(n\beta)$$
$$\sin(n[\phi-\beta]) = \sin(n\phi)\cos(n\beta) - \sin(n\beta)\cos(n\phi)$$

Eq. A.2 becomes

$$\begin{aligned} g(\mathbf{k}) = \sum_{n=0}^{\infty} \int\limits_0^{\infty} r\,\mathrm{d}r \int\limits_{\phi-2\pi}^{\phi} & [g_n^{\mathrm{c}}(r)[\cos(n\phi)\cos(n\beta) + \sin(n\phi)\sin(n\beta)] + \\ & + g_n^{\mathrm{s}}(r)[\sin(n\phi)\cos(n\beta) - \sin(n\beta)\cos(n\phi)]] e^{ikr\cos\beta}\,\mathrm{d}\beta = \\ = \sum_{n=0}^{\infty} \int\limits_0^{\infty} r\,\mathrm{d}r \int\limits_{\phi-2\pi}^{\phi} & [g_n^{\mathrm{c}}(r)\cos(n\phi) + g_n^{\mathrm{s}}\sin(n\phi)]\cos(n\beta) e^{ikr\cos\beta} + \\ & + [g_n^{\mathrm{c}}(r)\sin(n\phi) - g_n^{\mathrm{s}}\cos(n\phi)]\sin(n\beta) e^{ikr\cos\beta}\,\mathrm{d}\beta \end{aligned} \tag{A.3}$$

$\cos(n\beta)$ and $\sin(n\beta)$ are $2\pi$ periodical, therefore it is true:

$$\int\limits_{\phi-2\pi}^{\phi} \cos(n\beta) e^{ikr\cos\beta}\,\mathrm{d}\beta = \int\limits_0^{2\pi} \cos(n\beta) e^{ikr\cos\beta}\,\mathrm{d}\beta \tag{A.4}$$

$$\int\limits_{\phi-2\pi}^{\phi} \sin(n\beta) e^{ikr\cos\beta}\,\mathrm{d}\beta = \int\limits_0^{2\pi} \sin(n\beta) e^{ikr\cos\beta}\,\mathrm{d}\beta \tag{A.5}$$

Integral representation of Bessel function (Eq. 9.1.21 in [75]):

$$J_n(z) = \frac{i^{-n}}{2\pi} \int\limits_0^{2\pi} e^{iz\cos\beta}\cos(n\beta)\,\mathrm{d}\beta \tag{A.6}$$

Inserting Eq. A.4 and A.5 and into Eq. A.3 and using the represention A.6 one gets:

$$\begin{aligned} g(\mathbf{k}) = \sum_{n=0}^{\infty} & \int\limits_0^{\infty} [g_n^{\mathrm{c}}(r)\cos(n\phi) + g_n^{\mathrm{s}}\sin(n\phi)] i^n 2\pi J_n(kr) r\,\mathrm{d}r + \\ & + \int\limits_0^{2\pi} [g_n^{\mathrm{c}}(r)\sin(n\phi) - g_n^{\mathrm{s}}\cos(n\phi)]\sin(n\beta) e^{ikr\cos\beta}\,\mathrm{d}\beta \end{aligned} \tag{A.7}$$

Noting that $\sin(n\beta)$ is odd around $\beta = \pi$, the second term of Eq. A.7 is zero:

$$g(\mathbf{k}) = \sum_{n=0}^{\infty} \int_{0}^{\infty} [g_n^{\mathrm{c}}(r)\cos(n\theta) + g_n^{\mathrm{s}}(r)\sin(n\theta)] i^n 2\pi J_n(kr) r \, \mathrm{d}r \tag{A.8}$$

# B Appendix

### Integration of $\int\limits_{-\infty}^{\infty} \frac{e^{itx}}{x^2-a^2}\,\mathrm{d}x$

This integral resists conventional integration techniques of elementary calculus, but is possible to solve it by extending it to complex plane and calculating countour integrals.

$$\int\limits_{-\infty}^{\infty} \frac{e^{itx}}{x^2-a^2}\,\mathrm{d}x \xrightarrow{x\to z} \oint\limits_C f(z)\,\mathrm{d}z = \oint\limits_C \frac{e^{itz}}{z^2-a^2}\,\mathrm{d}z = \int\limits_{\text{arc}} \frac{e^{itz}}{z^2-a^2}\,\mathrm{d}z + \int\limits_{-b}^{b} \frac{e^{itz}}{z^2-a^2}\,\mathrm{d}z \tag{B.1}$$

Cauchy's Residue Theorem will be used in order to calculate this integral with two poles at $z = \pm a$

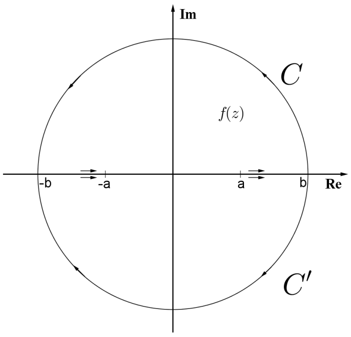


Figure 130: Top semi-circle $C$ represents case $t > 0$ (counts counter-clockwise), bottom semi-circle $C'$ represents case $t < 0$ (counts clockwise).

(Fig. 130):

$$\oint\limits_C f(z)\,\mathrm{d}z = 2\pi i \sum_k \mathrm{Res}(f(z), z_k) \tag{B.2}$$

Applying Residue Theorem for $t > 0$:

$$\int\limits_C \frac{e^{itz}}{z^2-a^2}\,\mathrm{d}z = 2\pi i \left[\mathrm{Res}\left(\frac{e^{itz}}{z^2-a^2}, a\right) + \mathrm{Res}\left(\frac{e^{itz}}{z^2-a^2}, -a\right)\right] = 2\pi i \left[\frac{e^{ita}}{2a} - \frac{e^{-ita}}{2a}\right] =$$

$$= \frac{2\pi}{a}\frac{e^{ita}-e^{-ita}}{-2i} = -\frac{2\pi}{a}\sin(ta) \tag{B.3}$$

Applying Residue Theorem for $t < 0$:

$$\int\limits_{C'} \frac{e^{itz}}{z^2-a^2}\,\mathrm{d}z = -2\pi i \left[\mathrm{Res}\left(\frac{e^{itz}}{z^2-a^2}, a\right) + \mathrm{Res}\left(\frac{e^{itz}}{z^2-a^2}, -a\right)\right] = -2\pi i \left[\frac{e^{ita}}{2a} - \frac{e^{-ita}}{2a}\right] =$$

$$= -\frac{2\pi}{a}\frac{e^{ita}-e^{-ita}}{-2i} = \frac{2\pi}{a}\sin(ta) \overset{t<0}{=} \frac{2\pi}{a}\sin(-|t|a) = -\frac{2\pi}{a}\sin(|t|a) \tag{B.4}$$

Now one can come back to real-argumented integral:

$$\int\limits_{-\infty}^{\infty} \frac{e^{itx}}{x^2-a^2}\,\mathrm{d}x = \begin{cases} -\frac{2\pi}{a}\sin(ta) & ,t>0 \\ -\frac{2\pi}{a}\sin(|t|a) & ,t<0 \end{cases} \overset{\text{symmetry}}{=} -\frac{2\pi}{a}\sin(|t|a) \tag{B.5}$$

With the variables of one's concern $\{x, t, a\} \longrightarrow \{k, \cos\theta - p, x_{nl}\}$ one gets:

$$\int\limits_{-\infty}^{\infty} \frac{e^{ik[\cos\theta-p]}}{k^2-x_{nl}^2}\,\mathrm{d}k = -\frac{2\pi}{x_{nl}}\sin(x_{nl}|\cos\theta-p|) \tag{B.6}$$

# C Appendix

**Integration of** $\int\limits_0^\pi \cos(n\theta)\sin(x_{nl}|\cos\theta - p|)\,\mathrm{d}\theta$

One needs to simplify Eq. 150:

$$f_{nl}(p) = -2J'_n(x_{nl}) \int\limits_0^\pi \cos(n\theta)\sin(x_{nl}|\cos\theta - p|)\,\mathrm{d}\theta \tag{C.1}$$

One removes absolute symbol by:

$$\begin{aligned}
f_{nl}(p) = -2J'_n(x_{nl}) &\left[ \int\limits_0^{\arccos p} \cos(n\theta)\sin(x_{nl}[\cos\theta - p])\,\mathrm{d}\theta - \int\limits_{\arccos p}^{\pi} \cos(n\theta)\sin(x_{nl}[\cos\theta - p])\,\mathrm{d}\theta \right] = \\
= -2J'_n(x_{nl}) &\left[ \int\limits_0^{\arccos p} \cos(n\theta)\sin(x_{nl}\cos\theta)\cos(x_{nl}p)\,\mathrm{d}\theta - \int\limits_0^{\arccos p} \cos(n\theta)\cos(x_{nl}\cos\theta)\sin(x_{nl}p)\,\mathrm{d}\theta + \right. \\
&\left. - \int\limits_{\arccos p}^{\pi} \cos(n\theta)\sin(x_{nl}\cos\theta)\cos(x_{nl}p)\,\mathrm{d}\theta + \int\limits_{\arccos p}^{\pi} \cos(n\theta)\cos(x_{nl}\cos\theta)\sin(x_{nl}p)\,\mathrm{d}\theta \right] \\
= -2J'_n(x_{nl}) &\left[ \cos(x_{nl}p) \int\limits_0^{\arccos p} \cos(n\theta)\sin(x_{nl}\cos\theta)\,\mathrm{d}\theta - \sin(x_{nl}p) \int\limits_0^{\arccos p} \cos(n\theta)\cos(x_{nl}\cos\theta)\,\mathrm{d}\theta + \right. \\
&\left. - \cos(x_{nl}p) \int\limits_{\arccos p}^{\pi} \cos(n\theta)\sin(x_{nl}\cos\theta)\,\mathrm{d}\theta + \sin(x_{nl}p) \int\limits_{\arccos p}^{\pi} \cos(n\theta)\cos(x_{nl}\cos\theta)\,\mathrm{d}\theta \right]
\end{aligned} \tag{C.2}$$

To solve these integrals it is necessary to know how to calculate $\sin(x_{nl}\cos\theta)$ and $\cos(x_{nl}\cos\theta)$. Equation 1 in chapter 2.1 of [70] will help:

$$e^{\frac{1}{2}z[t - \frac{1}{t}]} = \sum_{n=-\infty}^{\infty} t^n J_n(z) \tag{C.3}$$

Making the substitution $t \longrightarrow e^{i\left[\frac{\pi}{2} - \eta\right]}$:

$$\text{l.h.s: } e^{\frac{1}{2}z\left[e^{i\left[\frac{\pi}{2}-\eta\right]} - \frac{1}{e^{i\left[\frac{\pi}{2}-\eta\right]}}\right]} = e^{\frac{1}{2}z\left[e^{i\left[\frac{\pi}{2}-\eta\right]} - e^{-i\left[\frac{\pi}{2}-\eta\right]}\right]} = e^{iz\sin\left(\frac{\pi}{2}-\eta\right)} = e^{iz\cos\eta} = \cos(z\cos\eta) + i\sin(z\cos\eta) \tag{C.4}$$

$$\text{r.h.s: } \sum_{n=-\infty}^{\infty} e^{in\left[\frac{\pi}{2}-\eta\right]} J_n(z) = \sum_{n=-\infty}^{\infty} \cos\left(n\left[\frac{\pi}{2} - \eta\right]\right) J_n(z) + i \sum_{n=-\infty}^{\infty} \sin\left(n\left[\frac{\pi}{2} - \eta\right]\right) J_n(z) \tag{C.5}$$

From Eq. C.4 and C.5 one can deduce:

$$\cos(z\cos\eta) = \sum_{n=-\infty}^{\infty} \cos\left(n\left[\frac{\pi}{2} - \eta\right]\right) J_n(z) \tag{C.6}$$

$$\sin(z\cos\eta) = \sum_{n=-\infty}^{\infty} \sin\left(n\left[\frac{\pi}{2} - \eta\right]\right) J_n(z) \tag{C.7}$$

With the variables of one's concern $\{\eta, z, n\} \longrightarrow \{\theta, x_{nl}, m\}$:

$$\cos(x_{nl}\cos\theta) = \sum_{m=-\infty}^{\infty} \cos\left(m\left[\frac{\pi}{2}-\theta\right]\right) J_m(x_{nl}) \tag{C.8}$$

$$\sin(x_{nl}\cos\theta) = \sum_{m=-\infty}^{\infty} \sin\left(m\left[\frac{\pi}{2}-\theta\right]\right) J_m(x_{nl}) \tag{C.9}$$

Using Eq. C.8 and Eq. C.9, the 1st integral of Eq. C.2:

$$\begin{aligned}
&\int\limits_0^{\arccos p} \cos(n\theta)\sin(x_{nl}\cos\theta)\,\mathrm{d}\theta = \\
&= \int\limits_0^{\arccos p} \cos(n\theta)\left[\sum_{m=-\infty}^{\infty} \sin\left(m\left[\frac{\pi}{2}-\theta\right]\right) J_m(x_{nl})\right]\mathrm{d}\theta = \\
&= \sum_{m=-\infty}^{\infty} J_m(x_{nl}) \int\limits_0^{\arccos p} \cos(n\theta)\sin\left(m\left[\frac{\pi}{2}-\theta\right]\right)\mathrm{d}\theta = \\
&= \sum_{m=-\infty}^{\infty} J_m(x_{nl}) \int\limits_0^{\arccos p} \cos(n\theta)\left[\sin\left(m\frac{\pi}{2}\right)\cos(m\theta) - \cos\left(m\frac{\pi}{2}\right)\sin(m\theta)\right]\mathrm{d}\theta \\
&= \sum_{m=-\infty}^{\infty} J_m(x_{nl}) \left[\sin\left(m\frac{\pi}{2}\right)\int\limits_0^{\arccos p} \cos(m\theta)\cos(n\theta)\,\mathrm{d}\theta - \cos\left(m\frac{\pi}{2}\right)\int\limits_0^{\arccos p} \sin(m\theta)\cos(n\theta)\,\mathrm{d}\theta\right] = \\
&= \sum_{m=-\infty}^{\infty} J_m(x_{nl}) \left[\sin\left(m\frac{\pi}{2}\right)\int\limits_0^{\arccos p} \frac{1}{2}[\cos([m+n]\theta)+\cos([m-n]\theta)]\,\mathrm{d}\theta - \right.\\
&\qquad \left. - \cos\left(m\frac{\pi}{2}\right)\int\limits_0^{\arccos p} \frac{1}{2}[\sin([m+n]\theta)+\sin([m-n]\theta)]\,\mathrm{d}\theta\right] = \\
&= \sum_{m=-\infty}^{\infty} J_m(x_{nl}) \left[\frac{1}{2}\sin\left(m\frac{\pi}{2}\right)\left[\left.\frac{\sin([m+n]\theta)}{m+n}\right|_0^{\arccos p} + \left.\frac{\sin([m-n]\theta)}{m-n}\right|_0^{\arccos p}\right] - \right.\\
&\qquad \left. -\frac{1}{2}\cos\left(m\frac{\pi}{2}\right)\left[\left.\frac{-\cos([m+n]\theta)}{m+n}\right|_0^{\arccos p} + \left.\frac{-\cos([m-n]\theta)}{m-n}\right|_0^{\arccos p}\right]\right] = \\
&= \sum_{m=-\infty}^{\infty} J_m(x_{nl}) \left[\frac{1}{2}\sin\left(m\frac{\pi}{2}\right)\left[\frac{\sin([m+n]\arccos p)}{m+n} + \frac{\sin([m-n]\arccos p)}{m-n}\right] - \right.\\
&\qquad \left. -\frac{1}{2}\cos\left(m\frac{\pi}{2}\right)\left[\frac{-\cos([m+n]\arccos p)+1}{m+n} + \frac{-\cos([m-n]\arccos p)+1}{m-n}\right]\right] \qquad \text{(C.10)}
\end{aligned}$$

Using Eq. C.8 and Eq. C.9, the $2^{\text{nd}}$ integral of Eq. C.2:

$$
\begin{aligned}
&\int\limits_0^{\arccos p} \cos(n\theta)\cos(x_{nl}\cos\theta)\,\mathrm{d}\theta = \\
&= \int\limits_0^{\arccos p} \cos(n\theta)\left[\sum_{m=-\infty}^{\infty}\cos\left(m\left[\frac{\pi}{2}-\theta\right]\right)J_m(x_{nl})\right]\mathrm{d}\theta = \\
&= \sum_{m=-\infty}^{\infty} J_m(x_{nl}) \int\limits_0^{\arccos p} \cos(n\theta)\cos\left(m\left[\frac{\pi}{2}-\theta\right]\right)\mathrm{d}\theta = \\
&= \sum_{m=-\infty}^{\infty} J_m(x_{nl}) \int\limits_0^{\arccos p} \cos(n\theta)\left[\cos\left(m\frac{\pi}{2}\right)\cos(m\theta)+\sin\left(m\frac{\pi}{2}\right)\sin(m\theta)\right]\mathrm{d}\theta \\
&= \sum_{m=-\infty}^{\infty} J_m(x_{nl}) \left[\cos\left(m\frac{\pi}{2}\right)\int\limits_0^{\arccos p}\cos(m\theta)\cos(n\theta)\,\mathrm{d}\theta+\sin\left(m\frac{\pi}{2}\right)\int\limits_0^{\arccos p}\sin(m\theta)\cos(n\theta)\,\mathrm{d}\theta\right] = \\
&= \sum_{m=-\infty}^{\infty} J_m(x_{nl}) \left[\cos\left(m\frac{\pi}{2}\right)\int\limits_0^{\arccos p}\frac{1}{2}[\cos([m+n]\theta)+\cos([m-n]\theta)]\,\mathrm{d}\theta+\right. \\
&\qquad\qquad \left.+\sin\left(m\frac{\pi}{2}\right)\int\limits_0^{\arccos p}\frac{1}{2}[\sin([m+n]\theta)+\sin([m-n]\theta)]\,\mathrm{d}\theta\right] = \\
&= \sum_{m=-\infty}^{\infty} J_m(x_{nl}) \left[\frac{1}{2}\cos\left(m\frac{\pi}{2}\right)\left[\left.\frac{\sin([m+n]\theta)}{m+n}\right|_0^{\arccos p}+\left.\frac{\sin([m-n]\theta)}{m-n}\right|_0^{\arccos p}\right]+\right. \\
&\qquad\qquad \left.+\frac{1}{2}\sin\left(m\frac{\pi}{2}\right)\left[\left.\frac{-\cos([m+n]\theta)}{m+n}\right|_0^{\arccos p}+\left.\frac{-\cos([m-n]\theta)}{m-n}\right|_0^{\arccos p}\right]\right] = \\
&= \sum_{m=-\infty}^{\infty} J_m(x_{nl}) \left[\frac{1}{2}\cos\left(m\frac{\pi}{2}\right)\left[\frac{\sin([m+n]\arccos p)}{m+n}+\frac{\sin([m-n]\arccos p)}{m-n}\right]+\right. \\
&\qquad\qquad \left.+\frac{1}{2}\sin\left(m\frac{\pi}{2}\right)\left[\frac{-\cos([m+n]\arccos p)+1}{m+n}+\frac{-\cos([m-n]\arccos p)+1}{m-n}\right]\right] \qquad \text{(C.11)}
\end{aligned}
$$

Using Eq. C.8 and Eq. C.9, the 3$^{\text{rd}}$ integral of Eq. C.2:

$$\int\limits_{\arccos p}^{\pi} \cos(n\theta)\sin(x_{nl}\cos\theta)\,\mathrm{d}\theta =$$

$$= \int\limits_{\arccos p}^{\pi} \cos(n\theta)\left[\sum_{m=-\infty}^{\infty}\sin\left(m\left[\frac{\pi}{2}-\theta\right]\right)J_m(x_{nl})\right]\mathrm{d}\theta =$$

$$= \sum_{m=-\infty}^{\infty} J_m(x_{nl}) \int\limits_{\arccos p}^{\pi} \cos(n\theta)\sin\left(m\left[\frac{\pi}{2}-\theta\right]\right)\mathrm{d}\theta =$$

$$= \sum_{m=-\infty}^{\infty} J_m(x_{nl}) \int\limits_{\arccos p}^{\pi} \cos(n\theta)\left[\sin\left(m\frac{\pi}{2}\right)\cos(m\theta)-\cos\left(m\frac{\pi}{2}\right)\sin(m\theta)\right]\mathrm{d}\theta$$

$$= \sum_{m=-\infty}^{\infty} J_m(x_{nl}) \left[\sin\left(m\frac{\pi}{2}\right)\int\limits_{\arccos p}^{\pi}\cos(m\theta)\cos(n\theta)\,\mathrm{d}\theta - \cos\left(m\frac{\pi}{2}\right)\int\limits_{\arccos p}^{\pi}\sin(m\theta)\cos(n\theta)\,\mathrm{d}\theta\right] =$$

$$= \sum_{m=-\infty}^{\infty} J_m(x_{nl}) \left[\sin\left(m\frac{\pi}{2}\right)\int\limits_{\arccos p}^{\pi}\frac{1}{2}[\cos([m+n]\theta)+\cos([m-n]\theta)]\,\mathrm{d}\theta - \right.$$

$$\left. -\cos\left(m\frac{\pi}{2}\right)\int\limits_{\arccos p}^{\pi}\frac{1}{2}[\sin([m+n]\theta)+\sin([m-n]\theta)]\,\mathrm{d}\theta\right] =$$

$$= \sum_{m=-\infty}^{\infty} J_m(x_{nl}) \left[\frac{1}{2}\sin\left(m\frac{\pi}{2}\right)\left[\frac{\sin([m+n]\theta)}{m+n}\Bigg|_{\arccos p}^{\pi} + \frac{\sin([m-n]\theta)}{m-n}\Bigg|_{\arccos p}^{\pi}\right] - \right.$$

$$\left. -\frac{1}{2}\cos\left(m\frac{\pi}{2}\right)\left[\frac{-\cos([m+n]\theta)}{m+n}\Bigg|_{\arccos p}^{\pi} + \frac{-\cos([m-n]\theta)}{m-n}\Bigg|_{\arccos p}^{\pi}\right]\right] =$$

$$= \sum_{m=-\infty}^{\infty} J_m(x_{nl}) \left[\frac{1}{2}\sin\left(m\frac{\pi}{2}\right)\left[\frac{-\sin([m+n]\arccos p)}{m+n} + \frac{-\sin([m-n]\arccos p)}{m-n}\right] - \right.$$

$$\left. -\frac{1}{2}\cos\left(m\frac{\pi}{2}\right)\left[\frac{-(-1)^{m+n}+\cos([m+n]\arccos p)}{m+n} + \frac{-(-1)^{m-n}+\cos([m-n]\arccos p)}{m-n}\right]\right] \tag{C.12}$$

Using Eq. C.8 and Eq. C.9, the 4th integral of Eq. C.2:

$$\int\limits_{\arccos p}^{\pi} \cos(n\theta)\cos(x_{nl}\cos\theta)\,\mathrm{d}\theta =$$

$$= \int\limits_{\arccos p}^{\pi} \cos(n\theta)\left[\sum_{m=-\infty}^{\infty}\cos\left(m\left[\frac{\pi}{2}-\theta\right]\right)J_m(x_{nl})\right]\mathrm{d}\theta =$$

$$= \sum_{m=-\infty}^{\infty} J_m(x_{nl}) \int\limits_{\arccos p}^{\pi} \cos(n\theta)\cos\left(m\left[\frac{\pi}{2}-\theta\right]\right)\mathrm{d}\theta =$$

$$= \sum_{m=-\infty}^{\infty} J_m(x_{nl}) \int\limits_{\arccos p}^{\pi} \cos(n\theta)\left[\cos\left(m\frac{\pi}{2}\right)\cos(m\theta)+\sin\left(m\frac{\pi}{2}\right)\sin(m\theta)\right]\mathrm{d}\theta$$

$$= \sum_{m=-\infty}^{\infty} J_m(x_{nl}) \left[\cos\left(m\frac{\pi}{2}\right)\int\limits_{\arccos p}^{\pi}\cos(m\theta)\cos(n\theta)\,\mathrm{d}\theta + \sin\left(m\frac{\pi}{2}\right)\int\limits_{\arccos p}^{\pi}\sin(m\theta)\cos(n\theta)\,\mathrm{d}\theta\right] =$$

$$= \sum_{m=-\infty}^{\infty} J_m(x_{nl}) \left[\cos\left(m\frac{\pi}{2}\right)\int\limits_{\arccos p}^{\pi}\frac{1}{2}[\cos([m+n]\theta)+\cos([m-n]\theta)]\,\mathrm{d}\theta + \right.$$

$$\left. + \sin\left(m\frac{\pi}{2}\right)\int\limits_{\arccos p}^{\pi}\frac{1}{2}[\sin([m+n]\theta)+\sin([m-n]\theta)]\,\mathrm{d}\theta\right] =$$

$$= \sum_{m=-\infty}^{\infty} J_m(x_{nl}) \left[\frac{1}{2}\cos\left(m\frac{\pi}{2}\right)\left[\left.\frac{\sin([m+n]\theta)}{m+n}\right|_{\arccos p}^{\pi} + \left.\frac{\sin([m-n]\theta)}{m-n}\right|_{\arccos p}^{\pi}\right] + \right.$$

$$\left. + \frac{1}{2}\sin\left(m\frac{\pi}{2}\right)\left[\left.\frac{-\cos([m+n]\theta)}{m+n}\right|_{\arccos p}^{\pi} + \left.\frac{-\cos([m-n]\theta)}{m-n}\right|_{\arccos p}^{\pi}\right]\right] =$$

$$= \sum_{m=-\infty}^{\infty} J_m(x_{nl}) \left[\frac{1}{2}\cos\left(m\frac{\pi}{2}\right)\left[\frac{-\sin([m+n]\arccos p)}{m+n} + \frac{-\sin([m-n]\arccos p)}{m-n}\right] + \right.$$

$$\left. + \frac{1}{2}\sin\left(m\frac{\pi}{2}\right)\left[\frac{-(-1)^{m+n}+\cos([m+n]\arccos p)}{m+n} + \frac{-(-1)^{m-n}+\cos([m-n]\arccos p)}{m-n}\right]\right] \tag{C.13}$$

By combining Eq. {C.10,C.11,C.12,C.13} into Eq. C.2:

$$
\begin{aligned}
&f_{nl}(p) = -2J_n'(x_{nl}) \sum_{m=-\infty}^{\infty} J_m(x_{nl})\cdot \\
&\cdot\Bigg[\cos(x_{nl}p)\Bigg[\frac{\sin\left(m\frac{\pi}{2}\right)}{2}\left[\frac{\sin([m+n]\arccos p)}{m+n}+\frac{\sin([m-n]\arccos p)}{m-n}\right]- \\
&\qquad -\frac{\cos\left(m\frac{\pi}{2}\right)}{2}\left[\frac{-\cos([m+n]\arccos p)+1}{m+n}+\frac{-\cos([m-n]\arccos p)+1}{m-n}\right]\Bigg]- \\
&-\sin(x_{nl}p)\Bigg[\frac{\cos\left(m\frac{\pi}{2}\right)}{2}\left[\frac{\sin([m+n]\arccos p)}{m+n}+\frac{\sin([m-n]\arccos p)}{m-n}\right]+ \\
&\qquad +\frac{\sin\left(m\frac{\pi}{2}\right)}{2}\left[\frac{-\cos([m+n]\arccos p)+1}{m+n}+\frac{-\cos([m-n]\arccos p)+1}{m-n}\right]\Bigg]- \\
&-\cos(x_{nl}p)\Bigg[\frac{\sin\left(m\frac{\pi}{2}\right)}{2}\left[\frac{-\sin([m+n]\arccos p)}{m+n}+\frac{-\sin([m-n]\arccos p)}{m-n}\right]- \\
&\qquad -\frac{\cos\left(m\frac{\pi}{2}\right)}{2}\left[\frac{\cos([m+n]\arccos p)-(-1)^{m+n}}{m+n}+\frac{\cos([m-n]\arccos p)-(-1)^{m-n}}{m-n}\right]\Bigg]+ \\
&+\sin(x_{nl}p)\Bigg[\frac{\cos\left(m\frac{\pi}{2}\right)}{2}\left[\frac{-\sin([m+n]\arccos p)}{m+n}+\frac{-\sin([m-n]\arccos p)}{m-n}\right]+ \\
&\qquad +\frac{\sin\left(m\frac{\pi}{2}\right)}{2}\left[\frac{\cos([m+n]\arccos p)-(-1)^{m+n}}{m+n}+\frac{\cos([m-n]\arccos p)-(-1)^{m-n}}{m-n}\right]\Bigg]\Bigg]
\end{aligned}
$$

$$
\begin{aligned}
&f_{nl}(p) = -2J_n'(x_{nl}) \sum_{m=-\infty}^{\infty} J_m(x_{nl})\cdot \\
&\cdot\Bigg[\Bigg[\cos(x_{nl}p)\frac{\sin\left(m\frac{\pi}{2}\right)}{2}-\sin(x_{nl}p)\frac{\cos\left(m\frac{\pi}{2}\right)}{2}\Bigg]\left[\frac{\sin([m+n]\arccos p)}{m+n}+\frac{\sin([m-n]\arccos p)}{m-n}\right]- \\
&-\Bigg[\cos(x_{nl}p)\frac{\cos\left(m\frac{\pi}{2}\right)}{2}+\sin(x_{nl}p)\frac{\sin\left(m\frac{\pi}{2}\right)}{2}\Bigg]\left[\frac{1-\cos([m+n]\arccos p)}{m+n}+\frac{1-\cos([m-n]\arccos p)}{m-n}\right]- \\
&-\Bigg[\cos(x_{nl}p)\frac{\sin\left(m\frac{\pi}{2}\right)}{2}-\sin(x_{nl}p)\frac{\cos\left(m\frac{\pi}{2}\right)}{2}\Bigg]\left[\frac{-\sin([m+n]\arccos p)}{m+n}+\frac{-\sin([m-n]\arccos p)}{m-n}\right]+ \\
&+\Bigg[\cos(x_{nl}p)\frac{\cos\left(m\frac{\pi}{2}\right)}{2}+\sin(x_{nl}p)\frac{\sin\left(m\frac{\pi}{2}\right)}{2}\Bigg]\left[\frac{\cos([m+n]\arccos p)-(-1)^{m+n}}{m+n}+\frac{\cos([m-n]\arccos p)-(-1)^{m-n}}{m-n}\right]\Bigg]=
\end{aligned}
$$

$$
\begin{aligned}
&f_{nl}(p) = -2J_n'(x_{nl}) \sum_{m=-\infty}^{\infty} J_m(x_{nl})\cdot \\
&\cdot\Bigg[\left[\cos(x_{nl}p)\sin\left(m\frac{\pi}{2}\right)-\sin(x_{nl}p)\cos\left(m\frac{\pi}{2}\right)\right]\left[\frac{\sin([m+n]\arccos p)}{m+n}+\frac{\sin([m-n]\arccos p)}{m-n}\right]- \\
&-\Bigg[\cos(x_{nl}p)\frac{\cos\left(m\frac{\pi}{2}\right)}{2}+\sin(x_{nl}p)\frac{\sin\left(m\frac{\pi}{2}\right)}{2}\Bigg]\left[\frac{1-\cos([m+n]\arccos p)}{m+n}+\frac{1-\cos([m-n]\arccos p)}{m-n}\right]+ \\
&+\Bigg[\cos(x_{nl}p)\frac{\cos\left(m\frac{\pi}{2}\right)}{2}+\sin(x_{nl}p)\frac{\sin\left(m\frac{\pi}{2}\right)}{2}\Bigg]\left[\frac{\cos([m+n]\arccos p)-(-1)^{m+n}}{m+n}+\frac{\cos([m-n]\arccos p)-(-1)^{m-n}}{m-n}\right]\Bigg]
\end{aligned}
\tag{C.14}
$$

Now one will decompose the summation in Eq. C.14 into three parts $\{m = 0,\, m < 0,\, m > 0\}$:

$$
\begin{aligned}
f_{nl}(p) &= -2J'_n(x_{nl}) \left[ J_0(x_{nl}) \cdot \ldots + \sum_{m=-\infty}^{-1} J_m(x_{nl}) \cdot \ldots + \sum_{m=1}^{\infty} J_m(x_{nl}) \cdot \ldots \right] = \\
&= -2J'_n(x_{nl}) \left[ J_0(x_{nl}) \cdot \ldots + \sum_{m=1}^{\infty} J_{-m}(x_{nl}) \cdot \ldots |_{-m} + \sum_{m=1}^{\infty} J_m(x_{nl}) \cdot \ldots \right]
\end{aligned}
\tag{C.15}
$$

With the property of Bessel functions $J_{-m}(x) = (-1)^m J_m(x)$ from equation 9.1.5 in [75] one rewrites Eq. C.15:

$$
f_{nl}(p) = -2J'_n(x_{nl}) \left[ J_0(x_{nl}) \cdot \ldots + \sum_{m=1}^{\infty} (-1)^m J_m(x_{nl}) \cdot \ldots |_{-m} + \sum_{m=1}^{\infty} J_m(x_{nl}) \cdot \ldots \right] \tag{C.16}
$$

Now one needs to find member $\sum\limits_{m=1}^{\infty} (-1)^m J_m(x_{nl}) \cdot \ldots |_{-m}$:

$$
\begin{aligned}
&\sum_{m=1}^{\infty} (-1)^m J_m(x_{nl}) \cdot \ldots |_{-m} = \sum_{m=1}^{\infty} (-1)^m J_m(x_{nl}) \cdot \\
&\cdot \Bigg[ \left[ \cos(x_{nl}p) \sin\left(-m\frac{\pi}{2}\right) - \sin(x_{nl}p) \cos\left(-m\frac{\pi}{2}\right) \right] \left[ \frac{\sin([-m+n]\arccos p)}{-m+n} + \frac{\sin([-m-n]\arccos p)}{-m-n} \right] - \\
&- \left[ \cos(x_{nl}p) \frac{\cos\left(-m\frac{\pi}{2}\right)}{2} + \sin(x_{nl}p) \frac{\sin\left(-m\frac{\pi}{2}\right)}{2} \right] \left[ \frac{1-\cos([-m+n]\arccos p)}{-m+n} + \frac{1-\cos([-m-n]\arccos p)}{-m-n} \right] + \\
&+ \left[ \cos(x_{nl}p) \frac{\cos\left(-m\frac{\pi}{2}\right)}{2} + \sin(x_{nl}p) \frac{\sin\left(-m\frac{\pi}{2}\right)}{2} \right] \left[ \frac{\cos([-m+n]\arccos p) - (-1)^{-m+n}}{-m+n} + \frac{\cos([-m-n]\arccos p) - (-1)^{-}}{-m-n} \right.
\end{aligned}
$$

$$
\begin{aligned}
&= \sum_{m=1}^{\infty} (-1)^m J_m(x_{nl}) \cdot \\
&\cdot \Bigg[ \left[ -\cos(x_{nl}p) \sin\left(m\frac{\pi}{2}\right) - \sin(x_{nl}p) \cos\left(m\frac{\pi}{2}\right) \right] \left[ \frac{\sin([m-n]\arccos p)}{m-n} + \frac{\sin([m+n]\arccos p)}{m+n} \right] - \\
&+ \left[ \cos(x_{nl}p) \frac{\cos\left(m\frac{\pi}{2}\right)}{2} - \sin(x_{nl}p) \frac{\sin\left(m\frac{\pi}{2}\right)}{2} \right] \left[ \frac{1-\cos([m-n]\arccos p)}{m-n} + \frac{1-\cos([m+n]\arccos p)}{m+n} \right] + \\
&- \left[ \cos(x_{nl}p) \frac{\cos\left(m\frac{\pi}{2}\right)}{2} - \sin(x_{nl}p) \frac{\sin\left(m\frac{\pi}{2}\right)}{2} \right] \left[ \frac{\cos([m-n]\arccos p) - (-1)^{m-n}}{m-n} + \frac{\cos([m+n]\arccos p) - (-1)^{m+n}}{m+n} \right] \Bigg]
\end{aligned}
\tag{C.17}
$$

One needs to sum Eq. C.14 and Eq. C.17, but if done directly it is not possible to obtain any simplification, because of $(-1)^m$ in Eq. C.17. In order to get rid of $(-1)^m$ one needs to sum even $m$ and odd $m$

contributions separately. Even $m$:

$$\begin{aligned}
&f_{nl}(p) = -2J_n'(x_{nl}) \sum_{m \bmod 2=0}^{\infty} J_m(x_{nl}) \cdot \\
&\cdot\Bigg[\Bigg[-\sin(x_{nl}p)\cos\left(m\frac{\pi}{2}\right)\left[\frac{\sin([m+n]\arccos p)}{m+n}+\frac{\sin([m-n]\arccos p)}{m-n}\right]- \\
&\quad-\cos(x_{nl}p)\frac{\cos\left(m\frac{\pi}{2}\right)}{2}\left[\frac{1-\cos([m+n]\arccos p)}{m+n}+\frac{1-\cos([m-n]\arccos p)}{m-n}\right]+ \\
&\quad+\cos(x_{nl}p)\frac{\cos\left(m\frac{\pi}{2}\right)}{2}\left[\frac{\cos([m+n]\arccos p)-(-1)^{m+n}}{m+n}+\frac{\cos([m-n]\arccos p)-(-1)^{m-n}}{m-n}\right]\Bigg]+ \\
&+\Bigg[-\sin(x_{nl}p)\cos\left(m\frac{\pi}{2}\right)\left[\frac{\sin([m-n]\arccos p)}{m-n}+\frac{\sin([m+n]\arccos p)}{m+n}\right]- \\
&\quad+\cos(x_{nl}p)\frac{\cos\left(m\frac{\pi}{2}\right)}{2}\left[\frac{1-\cos([m-n]\arccos p)}{m-n}+\frac{1-\cos([m+n]\arccos p)}{m+n}\right]+ \\
&\quad-\cos(x_{nl}p)\frac{\cos\left(m\frac{\pi}{2}\right)}{2}\left[\frac{\cos([m-n]\arccos p)-(-1)^{m-n}}{m-n}+\frac{\cos([m+n]\arccos p)-(-1)^{m+n}}{m+n}\right]\Bigg]\Bigg]= \\
&\quad= 4J_n'(x_{nl}) \sum_{m \bmod 2=0}^{\infty} J_m(x_{nl})\sin(x_{nl}p)\cos\left(m\frac{\pi}{2}\right)\left[\frac{\sin([m+n]\arccos p)}{m+n}+\frac{\sin([m-n]\arccos p)}{m-n}\right]
\end{aligned} \tag{C.18}$$

Odd $m$:

$$\begin{aligned}
&f_{nl}(p) = -2J_n'(x_{nl}) \sum_{m \bmod 2=1}^{\infty} J_m(x_{nl}) \cdot \\
&\cdot\Bigg[\Bigg[\cos(x_{nl}p)\sin\left(m\frac{\pi}{2}\right)\left[\frac{\sin([m+n]\arccos p)}{m+n}+\frac{\sin([m-n]\arccos p)}{m-n}\right]- \\
&\quad-\sin(x_{nl}p)\frac{\sin\left(m\frac{\pi}{2}\right)}{2}\left[\frac{1-\cos([m+n]\arccos p)}{m+n}+\frac{1-\cos([m-n]\arccos p)}{m-n}\right]+ \\
&\quad+\sin(x_{nl}p)\frac{\sin\left(m\frac{\pi}{2}\right)}{2}\left[\frac{\cos([m+n]\arccos p)-(-1)^{m+n}}{m+n}+\frac{\cos([m-n]\arccos p)-(-1)^{m-n}}{m-n}\right]\Bigg] \\
&+(-1)\Bigg[-\cos(x_{nl}p)\sin\left(m\frac{\pi}{2}\right)\left[\frac{\sin([m-n]\arccos p)}{m-n}+\frac{\sin([m+n]\arccos p)}{m+n}\right]- \\
&\quad-\sin(x_{nl}p)\frac{\sin\left(m\frac{\pi}{2}\right)}{2}\left[\frac{1-\cos([m-n]\arccos p)}{m-n}+\frac{1-\cos([m+n]\arccos p)}{m+n}\right]+ \\
&\quad+\sin(x_{nl}p)\frac{\sin\left(m\frac{\pi}{2}\right)}{2}\left[\frac{\cos([m-n]\arccos p)-(-1)^{m-n}}{m-n}+\frac{\cos([m+n]\arccos p)-(-1)^{m+n}}{m+n}\right]\Bigg]\Bigg]= \\
&\quad= -4J_n'(x_{nl}) \sum_{m \bmod 2=1}^{\infty} J_m(x_{nl})\cos(x_{nl}p)\sin\left(m\frac{\pi}{2}\right)\left[\frac{\sin([m+n]\arccos p)}{m+n}+\frac{\sin([m-n]\arccos p)}{m-n}\right]
\end{aligned} \tag{C.19}$$

And there is one special case, when $m = 0$ in Eq. C.14 and Eq. C.17:

$$\begin{aligned} f_{nl}(p) = -2J_n'(x_{nl})J_0(x_{nl}) \Bigg[ &- \sin(x_{nl}p) \left[ \frac{\sin(n \arccos p)}{n} + \frac{\sin(-n \arccos p)}{-n} \right] - \\ &- \frac{\cos(x_{nl}p)}{2} \left[ \frac{1 - \cos(n \arccos p)}{n} + \frac{1 - \cos(-n \arccos p)}{-n} \right] + \\ &+ \frac{\cos(x_{nl}p)}{2} \left[ \frac{\cos(n \arccos p) - (-1)^n}{n} + \frac{\cos(-n \arccos p) - (-1)^{-n}}{-n} \right] \Bigg] = \\ = 4J_n'(x_{nl})J_0(x_{nl}) \sin(x_{nl}p) \frac{\sin(n \arccos p)}{n} & \end{aligned} \tag{C.20}$$

Putting together all the cases $\{m \bmod 2 = 0,\ m \bmod 2 = 1,\ m = 0\}$ from Eq. {C.18,C.19,C.20} one assembles the final outcome:

$$\begin{aligned} f_{nl}(p) =& 4J_n'(x_{nl})J_0(x_{nl}) \sin(x_{nl}p) \frac{\sin(n \arccos p)}{n} + \\ &+ 4J_n'(x_{nl}) \sum_{m \bmod 2 = 0}^{\infty} J_m(x_{nl}) \sin(x_{nl}p) \cos\left(m\frac{\pi}{2}\right) \left[ \frac{\sin([m+n] \arccos p)}{m+n} + \frac{\sin([m-n] \arccos p)}{m-n} \right] - \\ &- 4J_n'(x_{nl}) \sum_{m \bmod 2 = 1}^{\infty} J_m(x_{nl}) \cos(x_{nl}p) \sin\left(m\frac{\pi}{2}\right) \left[ \frac{\sin([m+n] \arccos p)}{m+n} + \frac{\sin([m-n] \arccos p)}{m-n} \right] = \\ =& -2J_n'(x_{nl}) \sum_{m=0}^{\infty} 2J_m(x_{nl}) \sin\left(m\frac{\pi}{2} - x_{nl}p\right) \left[ \frac{\sin([m+n] \arccos p)}{m+n} + \frac{\sin([m-n] \arccos p)}{m-n} \right] \end{aligned} \tag{C.21}$$

# D Appendix

### Pseudocode of Conjugate Gradient

**Algorithm 1** Conjugate Gradient (unconditioned)

```
$\mathbf{r}_0 := \mathbf{b} - \mathbf{A}\mathbf{x}_0$
$\mathbf{p}_0 := \mathbf{r}_0$
$k := 0$
loop
    $\alpha_k := \frac{\mathbf{r}_k^{\mathrm{T}}\mathbf{r}_k}{\mathbf{p}_k^{\mathrm{T}}\mathbf{A}\mathbf{p}_k}$
    $\mathbf{x}_{k+1} := \mathbf{x}_k + \alpha_k \mathbf{p}_k$
    $\mathbf{r}_{k+1} := \mathbf{r}_k - \alpha_k \mathbf{A}\mathbf{p}_k$
    if $\sqrt{\mathbf{r}_k^{\mathrm{T}}\mathbf{r}_k} < \epsilon$ then return $\mathbf{x}_{k+1}$
    end if
    $\beta_k := \frac{\mathbf{r}_{k+1}^{\mathrm{T}}\mathbf{r}_{k+1}}{\mathbf{r}_k^{\mathrm{T}}\mathbf{r}_k}$
    $\mathbf{p}_{k+1} := \mathbf{r}_{k+1} + \beta_k \mathbf{p}_k$
    $k := k + 1$
end loop
```

**Algorithm 2** Conjugate Gradient (preconditioned)

```
$\mathbf{r}_0 := \mathbf{b} - \mathbf{A}\mathbf{x}_0$
$\mathbf{z}_0 := \mathbf{M}^{-1}\mathbf{r}_0$
$\mathbf{p}_0 := \mathbf{z}_0$
$k := 0$
loop
    $\alpha_k := \frac{\mathbf{r}_k^{\mathrm{T}}\mathbf{r}_k}{\mathbf{p}_k^{\mathrm{T}}\mathbf{A}\mathbf{p}_k}$
    $\mathbf{x}_{k+1} := \mathbf{x}_k + \alpha_k \mathbf{p}_k$
    $\mathbf{r}_{k+1} := \mathbf{r}_k - \alpha_k \mathbf{A}\mathbf{p}_k$
    if $\sqrt{\mathbf{r}_k^{\mathrm{T}}\mathbf{r}_k} < \epsilon$ then return $\mathbf{x}_{k+1}$
    end if
    $\mathbf{z}_k := \mathbf{M}^{-1}\mathbf{r}_{k+1}$
    $\beta_k := \frac{\mathbf{z}_{k+1}^{\mathrm{T}}\mathbf{r}_{k+1}}{\mathbf{z}_k^{\mathrm{T}}\mathbf{r}_k}$
    $\mathbf{p}_{k+1} := \mathbf{z}_{k+1} + \beta_k \mathbf{p}_k$
    $k := k + 1$
end loop
```

# E Appendix

### Algorithm of SVD

Generally algorithms for computing singular values are analogs of algorithms for computing eigenvalues of symmetric matrices. The idea is always to find square roots of eigenvalues of $\mathbf{A}^{\mathrm{T}}$ without actually computing $\mathbf{A}^{\mathrm{T}} \cdot \mathbf{A}$. Matrix $\mathbf{A}$ has a dimension of $m \times n$. Algorithm assumes $m \geq n$; if $m < n$, then the algorithm may be applied to $\mathbf{A}^{\mathrm{T}}$. The algorithm is presented as if the matrix is real. Nevertheless, it can be applied for complex matrices too. Algorithm is constructed by 3 parts: Householder reduction (E.0.1), Golub-Reinch SVD (E.0.2) and Golub-Kahan SVD (E.0.3) (it is used iteratively in E.0.2). The developments of this algorithm can be found in [76][77][78][79].

### E.0.1 Householder reduction to bidiagonal form

Input: $\mathbf{A}_{m\times n}$
Ouput: $\mathbf{B}_{n\times n}$, $\mathbf{U}_{m\times n}$, $\mathbf{V}_{n\times n}$, so that $\mathbf{B}$ is upper bidiagonal, $\mathbf{U}$ and $\mathbf{V}$ are products of Householders matrices and $\mathbf{A} = \mathbf{U} \cdot \mathbf{B} \cdot \mathbf{V}^{\mathrm{T}}$

1. $\mathbf{B} \leftarrow \mathbf{A}$ (This can be omitted if $\mathbf{A}$ is to be overwritten with with $\mathbf{B}$)
2. $\mathbf{U} = \mathbf{I}_{m\times n}$
3. $\mathbf{V} = \mathbf{I}_{n\times n}$
4. For $k = \{1, \ldots, n\}$
   (a) Determine Householder matrix $\mathbf{Q}_k$ with the property that left multiplication by $\mathbf{Q}_k$ leaves components $\{1, \ldots, k-1\}$ unaltered such that:
   $$\mathbf{Q}_k \cdot \begin{bmatrix} 0 \\ \vdots \\ 0 \\ B_{k-1,k} \\ B_{k,k} \\ B_{k+1,k} \\ \vdots \\ B_{m,n} \end{bmatrix} = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ B_{k-1,k} \\ s \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \text{ where } s = \pm\sqrt{\sum_{i=k}^{m} B_{i,k}^2}$$
   (b) $\mathbf{B} \leftarrow \mathbf{Q}_k \cdot \mathbf{B}$
   (c) $\mathbf{U} \leftarrow \mathbf{U} \cdot \mathbf{Q}_k$
   (d) If $k \leq n-2$, determine Householder matrix $\mathbf{P}_{k+1}$ leaves components $\{1, \ldots, k\}$ unaltered such that:
   $$\begin{bmatrix} 0 & \cdots & 0 & B_{k,k} & B_{k,k+1} & B_{k,k+2} & \cdots & B_{k,n} \end{bmatrix} \cdot \mathbf{P}_{k+1} = \begin{bmatrix} 0 & \cdots & 0 & B_{k,k} & s & 0 & \cdots & 0 \end{bmatrix},$$
   where $s = \pm\sqrt{\sum\limits_{j=k+1}^{n} B_{k,j}^2}$
   (e) $\mathbf{B} \leftarrow \mathbf{B} \cdot \mathbf{P}_{k+1}$
   (f) $\mathbf{V} \leftarrow \mathbf{P}_{k+1}\mathbf{V}$

### E.0.2 Golub-Reinsch SVD

Input: $\mathbf{A}_{m\times n}$
Ouput: $\mathbf{\Sigma}_{n\times n}$, $\mathbf{U}_{m\times n}$, $\mathbf{V}_{n\times n}$, so that $\mathbf{\Sigma}$ is diagonal, $\mathbf{U}$ and $\mathbf{V}$ have orthogonal columns and $\mathbf{A} = \mathbf{U} \cdot \mathbf{\Sigma} \cdot \mathbf{V}^{\mathrm{T}}$

1. Apply algorithm (E.0.1)

2. Repeat:

   (a) Set $B_{i,i+1} = 0$, for all $i = \{1, \dots, n-1\}$, which satisfy $|B_{i,i+1}| \leq \varepsilon(|B_{i,i}| + |B_{i+1,i+1}|)$.

   (b) Determine the smallest $p$ and the largest $q$ so that $\mathbf{B}$ can be blocked as

$$\mathbf{B} = \begin{bmatrix} \mathbf{B}_{1,1} & 0 & 0 \\ 0 & \mathbf{B}_{2,2} & 0 \\ 0 & 0 & \mathbf{B}_{3,3} \end{bmatrix} \begin{array}{c} p \\ n - p - q \\ q \end{array}$$

   where $B_{3,3}$ is diagonal and $B_{2,2}$ has non-zero superdiagonal entry.

   (c) If $q = n$, set $\Sigma =$ the diagonal portion of of $\mathbf{B}$. STOP

   (d) If $B_{i,i} = 0$,then for $i = \{p+1, \dots, n-q-1\}$ apply Givens rotations so that $B_{i,i+1} = 0$ and $\mathbf{B}_{2,2}$ is still upper bidiagonal,
   else apply algorithm E.0.3 to $\mathbf{B}$, $\mathbf{U}$, $\mathbf{V}$

### E.0.3 Golub-Kahan SVD

Input: $\mathbf{B}_{n\times n}$, $\mathbf{Q}_{m\times n}$, $\mathbf{P}_{n\times n}$, where $\mathbf{B}$ is upper bidiagonal, $\mathbf{Q}$ and $\mathbf{P}$ have orthogonal columns and $\mathbf{A} = \mathbf{Q} \cdot \mathbf{B} \cdot \mathbf{P}^{\mathrm{T}}$
Ouput: $\mathbf{B}_{n\times n}$, $\mathbf{Q}_{m\times n}$, $\mathbf{P}_{n\times n}$, the output of $\mathbf{B}$ has smaller off-diagonal elements than the input $\mathbf{B}$

1. Let $\mathbf{B}_{2,2}$ be the diagonal block of $\mathbf{B}$ with row and column indices $\{p+1, \dots n-q\}$

2. Set $\mathbf{C} =$ lower right $2 \times 2$ submatrix of $B_{2,2}^{\mathrm{T}}$

3. Obtain eigenvalues $\lambda_1$, $\lambda_2$ of $\mathbf{C}$. Set $\mu =$ whichever of $\lambda_1$, $\lambda_2$ that is closer to $C_{2,2}$

4. Set $k = p+1$, $\alpha = B_{k,k}^2 - \mu$, $\beta = B_{k,k}B_{k,k+1}$

5. For $k = p+1, \dots, n-q-1$

   (a) Determine $c = \cos\theta$ and $s = \sin\theta$ with the property that:

$$\begin{bmatrix} \alpha & \beta \end{bmatrix} \cdot \begin{bmatrix} c & s \\ -s & c \end{bmatrix} = \begin{bmatrix} \sqrt{\alpha^2 + \beta^2} & 0 \end{bmatrix}$$

   (b) $\mathbf{B} \leftarrow \mathbf{B} \cdot \mathbf{R}_{k,k+1}(c, s)$, where $\mathbf{R}_{k,k+1}(c, s)$ is Givens rotation matrix that acts on a columns $k$ and $k+1$ during right multiplication

   (c) $\mathbf{P} \leftarrow \mathbf{P} \cdot \mathbf{R}_{k,k+1}(c, s)$

   (d) $\alpha = B_{k,k}$, $\beta = B_{k,k+1}$

   (e) Determine $c = \cos\theta$ and $s = \sin\theta$ with the property that:

$$\begin{bmatrix} c & -s \\ s & c \end{bmatrix} \cdot \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} \sqrt{\alpha^2 + \beta^2} \\ 0 \end{bmatrix}$$

   (f) $\mathbf{B} \leftarrow \mathbf{R}_{k,k+1}(c, -s) \cdot \mathbf{B}$, where $\mathbf{R}_{k,k+1}(c, -s)$ is Givens rotation matrix that acts on rows $k$ and $k+1$ during left multiplication

   (g) $\mathbf{Q} \leftarrow \mathbf{Q} \cdot \mathbf{R}_{k,k+1}(c, s)$

   (h) If $k \leq n-q-1$, then $\alpha = B_{k,k+1}$ and $\beta = B_{k,k+2}$

# Acknowledgements

This PhD was a great life adventure. I learnt a lot in terms of coding, physics and human relations. It will be extremely useful in the upcoming endevours of my life. Please find below a table of people I want to thank (Tab. 27):

| Person(s) | Institution | Level of greatfulness | Cause/Reason |
|---|---|---|---|
| **Fusion-DC** board | Ghent University | Very greatful | for giving me this opportunity to be a **Fusion-DC** student and seek a PhD title in fusion science |
| Paolo Bettini | Padova University, RFX | Very greatful | for his hard work as an administrator and coordinator of **Fusion-DC** section in Padova and fruitful collaboration with me on scientific endeavours |
| Gabriele Manduchi | RFX | Very greatful | for being a great supervisor and helping me with the scientific endeavours |
| Horácio Fernandes<br>Bernardo Carvalho | IST, IPFN | Very greatful | for a warm welcome to IST and their concern of my well-being during the mobility |
| Pedro Carvalho | IST, IPFN | Very greatful | for being the ISTTOK tomography pioneer and tireless helper with tomography stuff |
| César Alves | IST, IPFN | Very greatful | for a great collaboration on tomographic algorithms and coding stuff |
| Emanuele Sartori<br>Pierluigi Veltri | RFX | Extremely greatful | for long-lasting collaboration on NBI simulations and effective motivations on life stuff |
| Leonardo Pigatto<br>Carlo Baltador<br>Yangyang Zhang<br>Marco Gottardo<br>Daniele Aprile<br>Matteo Zaupa<br>Ondřey Kudláček<br>Pietro Vicenzi<br>Vadim Yanovskiy<br>Oisin McCormack<br>Matteo Vallar<br>Mattia Dan<br>Ferdinando Gasparini | RFX | Extremely greatful | 1) for being wonderful trustworthy friends and colleagues<br>2) for making a graceful mess in the office, which is a phenomenon of self-inducing chaos in a form of human behaviour balancing between madness and idiocy :) |

Table 27: Table of my greatfulness to the people, what have been part of my PhD experience

# References

[1] U. E. I. Administration, *Annual Energy Outlook 2016 With Projections to 2040.* Washington, DC 20585, U.S.A: U.S. Department of Energy, August 2016.

[2] R. M. Dell and D. A. J. Rand, *Clean energy.* Royal Society of Chemistry, October 2004.

[3] R. G. Watts, *Innovative Energy Strategies for $CO_2$ stabilization.* Cambridge University Press, 2002.

[4] R. A. Dunlap, *An introduction to the Physics of Nuclei and Particles.* California: Thomson Brooks/Cole, 2004.

[5] M. K. et al, "High fusion performance from deuterium-tritium plasmas in jet," *Nuclear Fusion*, vol. 39, no. 2, p. 209, 1999.

[6] R. J. H. et al, "Results from d—t experiments on tftr and implications for achieving an ignited plasma," *Nuclear Fusion*, vol. 357, no. 1752, pp. 443–469, 1999.

[7] R. Aymar, "The iter reduced cost design," *Fusion Engineering and Design*, vol. 50, pp. 13–25, 2000.

[8] I. H. Hutchinson, *Principes of Plasma Diagnostics: Second Edition.* Cambridge, Massachusetts: Cambridge University Press, 2001.

[9] M. Kikuchi, K. Lackner, and M. Q. Tran, *Fusion Physics.* Vienna, Austria: IAEA, 2012.

[10] L. A. B. et al, "Neutral beam injection experiments in ormak," *Plasma Physics and Controlled Nuclear Fusion Research: Fifth Conference Proceedings*, vol. 1, pp. 113–125, 11–15 Tokyo 1974.

[11] K. B. et al, "Neutral-beam heating in the adiabatic toroidal compressor," *Plasma Physics and Controlled Nuclear Fusion Research: Fifth Conference Proceedings*, vol. 1, pp. 77–82, 11–15 Tokyo 1974.

[12] K. Ushigusa and J.-. team, "Steady state operation research in jt-60u," *Fusion Energy*, vol. 1, pp. 37–55, 7–11 Montreal 1996.

[13] R. S. Hemsworth and T. Inoue, "Positive and negative ion sources for magnetic fusion," *Fusion Energy*, vol. 33, no. 6, pp. 1799–1813, December 2005.

[14] M. Hanada, N. Akino, and N. Ebisawa, "Development of multi-megawatt negative ion sources and accelerators for neutral beam injectors," *Fusion energy*, vol. 33, no. 29, p. 1, 2000.

[15] I. G. Brown, *The Physics and Technology of Ion Sources: 2nd, Revised and Extended Edition.* Vienna, Austria: WileyVCH Verlag GmbH & Co, September 2004.

[16] Y. Belchenko, G. Dimov, and V. Dudnikov, "A powerful injector of neutrals with a surface-plasma source of negative ions," *Fusion Energy*, vol. 14, no. 1, pp. 113–114, 1974.

[17] S. H. Fuller and L. I. Millett, *The Future of Computing Performance: Game Over or Next Level?* Washington, D.C.: The National Academies Press, 2011.

[18] D. P. Rodgers, "Improvements in multiprocessor system design," *ISCA '85 Proceedings of the 12th annual international symposium on Computer architecture*, pp. 225–231, 1985.

[19] www.top500.org, "June 2016," *TOP500*, p. 1, 2016.

[20] V. Hindriksen, *The history of the PC from 2000 – 2012.* www.streamcomputing.eu/blog/2011-05-06/the-history-of-the-pc-from-2000-2012: Stream Computing: Performance Engineers, 2011.

[21] www.wikipedia.com, "Cray," *Wikipedia*, p. 1, 2016.

[22] www.wikipedia.com, "Vector processor," *Wikipedia*, p. 1, 2016.

[23] www.wikipedia.com, "Silicon graphics," *Wikipedia*, p. 1, 2016.

[24] K. M. et al, "Why do commodity graphics hardware boards (gpus) work so well for acceleration of computed tomography?," *Computational Imaging V*, p. 194, 2007.

[25] NVIDIA, *Cuda C Programming Guide.* www.nvidia.com: Morgan Kaufmann, 2015.

[26] D. B. Kirk and W. mei W. Hwu, *Pogramming Massively Parallel Processors: A Hands-on Appoach.* Burlington: Morgan Kaufmann, 2010.

[27] NVIDIA, *What is CUDA?* www.nvidia.com: NVIDIA, 2016.

[28] N. S. et al, "Designing efficient sorting algorithms for manycore gpus," *Parallel & Distributed Processing*, pp. 1–10, 2009.

[29] K. O. W. Group, *The OpenCL Specification.* www.khronos.org: Khronos Group, 2016.

[30] O. A. R. Board, *OpenMP Application Program Interface.* www.openmp.org: OpenMP Architecture Review Board, 2013.

[31] A. C. N. et al, "Marte: A multiplatform real-time framework," *IEEE Transactions on Nuclear Science*, vol. 57, no. 6, pp. 225–231, April 2010.

[32] P. J. de Paula Carvalho et al, "Tomographic visualization for plasma position control in isttok," *IEEE Transactions on Nuclear Science*, vol. 36, no. 4, pp. 1102–1103, August 2008.

[33] NVIDIA, *What is CUDA? Developing a Linux Kernel Module using GPUDirect RDMA.* http://docs.nvidia.com/cuda/gpudirect-rdma/#abstract: NVIDIA, 2016.

[34] J. N. et al, "Image acquisition and gpu processing application using irio technology and flexrio devices," *Real Time Conference (RT), 2016 IEEE-NPSS*, June 2016.

[35] R. Courant, K. Friedrichs, and H. Levy, "Über die partiellen differenzengleichungen der mathematischen physik," *Mathematische Annalen*, vol. 100, pp. 32–74, 1928.

[36] M. A. Lieberman and A. J. Lichtenberg, *Principles of Plasma Discharges and Materials Processing: Second edition.* Hoboken, New Jersey: John Wiley & Sons, Inc., 2005.

[37] H. J. B. et al, "Molecular dynamics with coupling to an external bath," *The Journal of chemical physics*, vol. 81, no. 8, pp. 3684–3690, 1984.

[38] A. A. Vlasov, "The vibrational properties of an electron gas," *Soviet Physics Uspekhi*, vol. 10, no. 6, p. 721, 1968.

[39] R. W. Hockney and J. Eastwood, *Computer Simulation Using Particles.* New York, London: Taylor & Francis, 1988.

[40] C. K. Birdsall and A. B. Langdon, *Plasma Physics via Computer Simulation.* Bristol, Philadelphia and New York: Adam Hilger, 1991.

[41] C. K. Birdsall, "Particle-in-cell charged-particle simulations plus monte carlo collisions with netural atoms, pic-mcc," *Transactions on Plasma Science*, vol. 19, no. 2, pp. 65–85, 1991.

[42] Y. Saad and M. H. Schultz, "Gmres: a generalized minimal residual algorithm for solving nonsymmetric linear systems," *Society for Industrial and Applied Mathematics*, vol. 7, no. 3, pp. 856–869, 1986.

[43] A. van der Sluis and H. A. van der Vorst, "The rate of convergence of conjugate gradients," *Numerische Matematik*, vol. 48, pp. 543–560, 1986.

[44] E. F. Kaasschieter, "Preconditioned conjugate gradients for solving singular systems," *Journal of Computational and Applied Mathematics*, vol. 24, pp. 265–275, 1988.

[45] R. S. Varga, *Matrix Iterative Analysis.* New Jersey: Prentice-Hall, 1962.

[46] J. A. Meijerink and H. A. van der Vorst, "An iterative solution method for linear systems of which the coefficient matrix is a symmetric m-matrix," *Mathematics of Computation*, vol. 31, no. 137, pp. 148–162, 1977.

[47] J. A. Meijerink and H. A. van der Vorst, "Guidelines for the usage of incomplete decompositions in solving sets of linear equations as they occur in practical problems," *Journal of Computational Physics*, vol. 44, pp. 134–155, 1981.

[48] J. P. Boris, "Relativistic plasma simulation-optimization of a hybrid code," *Proceedings of the Fourth Conference on Numerical Simulation of Plasmas*, vol. 2, no. 3, pp. 3–67, 1970.

[49] P. H. S. et al, "Efficiency of a boris-like integration scheme with spatial stepping," *Physical Review Special Topics-Accelerators and Beams*, vol. 5, no. 9, p. 094001, 2002.

[50] G. P. et al, "Boris push with spatial stepping," *Journal of Physics G: Nuclear and Particle Physics*, vol. 29, no. 8, pp. 1719–1722, 2003.

[51] T. Takizuka and H. Abe, "A binary collision model for plasma simulation with a particle code," *Journal of Computational Physics*, vol. 25, no. 3, pp. 205–219, 1977.

[52] V. Vahedi and M. Surendra, "A monte carlo collision model for the particle-in-cell method: applications to argon and oxygen discharges," *Computer Physics Communications*, vol. 87, no. 1–2, pp. 179–198, 1995.

[53] K. Nanbu, "Probability theory of electron-molecule, ion-molecule, moleculemolecule, and coulomb collisions for particle modeling of materials processing plasmas and cases," *IEEE Transactions on Plasma Science*, vol. 28, no. 3, pp. 971–990, 2000.

[54] N. Bell and M. Garland, *Cusp: Generic parallel algorithms for sparse matrix and graph computations.* www.cusplibrary.github.io: Version 0.5.1, 2012.

[55] S. Mochalskyy, *Modeling of the negative ion extraction from a hydrogen plasma source. Application to ITER Neutral Beam Injector.* Paris, France: University Paris-Sud, December 2011.

[56] V. A. et al, "Physics design of the injector source for iter neutral beam injector," *Review of Scientific Instruments*, vol. 85, p. 5, 2014.

[57] A. S. et al, "R&d around a photoneutralizer-based nbi system (siphore) in view of a demo tokamak steady state fusion reactor," *Nuclear Fusion*, vol. 55, p. 19pp, 2015.

[58] J. P. Boeuf, G. Fubiani, and L. Garrigues, "Issues in the understanding of negative ion extraction for fusion," *Plasma Sources Science and Technology*, vol. 25, no. 4, pp. 1–12, 2016.

[59] M. Cavenago, "Moderately converging ion and electron flows in two-dimensional diodes," *Review of Scientific Instruments*, vol. 83, p. 14, 2012.

[60] M. Rudd, Y. Kim, D. Madison, and T. J. Gay, "Electron production in proton collisions with atoms and molecules: energy distributions," *Reviews of Modern Physics*, vol. 64, no. 2, pp. 441–490, 1992.

[61] M. Gealy, G. Kerby, Y. Hsu, and M. Rudd, "Energy and angular distributions of electrons from ion impact on atomic and molecular hydrogen. ii. 20–114-kev $h^+$+$h_2$," *Physical Review A*, vol. 51, no. 3, pp. 2256–2264, 1995.

[62] Y. K. K. et al, *Electron-impact cross sections for ionization and excitation.* www.nist.gov/pml/data/ionization/index.cfm: National Institute of Standarts Technology, 2015.

[63] C. F. B. et al, *Atomic Data for Fusion.* www-cfadc.phy.ornl.gov/redbooks/one/1.html: The ORNL Redbooks, June 2015.

[64] J. Radon, "Über die bestimmung von funktionen durch ihre integralwerte längs gewisser mannigfaltigkeiten," *Berichte über die Verhandlungen der Sächsischen Akademie der Wissenschaften zu Leipzig*, vol. 69, pp. 262–277, 1917.

[65] A. M. Cormack, "Representation of a function by its line integrals, with some radiological applications," *Journal of Applied Physics*, vol. 34, no. 9, pp. 2722–2727, 1963.

[66] A. M. Cormack, "Representation of a function by its line integrals, with some radiological applications ii," *Journal of Applied Physics*, vol. 35, no. 10, pp. 2908–2913, 1964.

[67] P. J. de Paula Carvalho, *Tomography Algorithms for Real-Time Control in ISTTOK.* Lisbon, Portugal: Instituto Superior Téchnico, April 2010.

[68] L. Wang and R. S. Granetz, "A simplified expression for the radon transform of bessel basis functions in tomography," *Review of Scientific Instruments*, vol. 62, no. 10, pp. 842–843, 1991.

[69] L. Wang and R. S. Granetz, "An analytical expression for the radon transform of bessel basis function in tomography," *Review of Scientific Instruments*, vol. 62, no. 10, pp. 1115–1116, 1991.

[70] G. N. Watson, *A Treatise on the Theory of Bessel Functions.* Cambridge, England: Cambridge University Press, 1958.

[71] A. Dresden, "The fourteenth western meeting of the american mathematical society," *Bulletin of the American Mathematical Society*, vol. 26, pp. 385–396, 06 1920.

[72] W. H. P. et al, *Numerical Recipes in C, The Art of Scientific Computing, Third Edition.* Cambridge, New York, Port Chester, Melbourne, Sydney: Cambridge University Press, 2007.

[73] P. J. de Paula Carvalho et al, "Isttok plasma control with the tomography diagnostic," *Fusion Engineering and Design*, vol. 85, pp. 266–271, 2010.

[74] C. A. S. Alves, *Tomographic Determination of Emissivity Profiles in the ISTTOK Tokamak.* Lisbon, Portugal: Instituto Superior Téchnico, June 2016.

[75] M. Abramowitz and I. A. Stegun, *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables.* Washington D.C., USA: National Bureau of Standarts, 1972.

[76] G. H. Golub and W. Kahan, "Calculating the singular values and pseudo-inverse of a matrix," *Journal of the Society for Industrial and Applied Mathematics: Series B, Numerical Analysis*, vol. 2, no. 2, pp. 205–224, 1965.

[77] G. H. Golub, "Least squares, singular values, and matrix approximations," *Aplikace Matematiky*, vol. 13, no. 1, pp. 44–51, 1968.

[78] G. H. Golub and C. Reinsch, "Singular value decomposition and least squares solutions," *Journal of Numerical Mathematics*, vol. 14, pp. 403–420, 1970.

[79] G. H. Golub and C. F. V. Loan, *Matrix Computations. $3^{rd}$ edition.* Baltimore and London: The Johns Hopkins University Press, 1996.