UNIVERSIDADE FEDERAL DE MINAS GERAIS
Escola de Engenharia
Programa de Pós-Graduação em Engenharia Elétrica

Marlon Jesus Lizarazo Urbina

PARALLEL-GPU DGTD METHOD WITH A THIRD-ORDER LOCAL TIME STEPPING SCHEME

Belo Horizonte

2025

Marlon Jesus Lizarazo Urbina

PARALLEL-GPU DGTD METHOD WITH A THIRD-ORDER LOCAL TIME STEPPING SCHEME

Tese apresentada ao Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal de Minas Gerais, como requisito parcial à obtenção do título de Doutor em Engenharia Elétrica.

Orientador: Prof. Dr. Elson José da Silva

Belo Horizonte

2025

L789p

Lizarazo Urbina, Marlon Jesus.
Parallel-GPU DGTD method with a third-order local time stepping scheme [recurso eletrônico] / Marlon Jesus Lizarazo Urbina. - 2025.
1 recurso online (125 f. : il., color.) : pdf.

Orientador: Elson José da Silva.

Tese (doutorado) - Universidade Federal de Minas Gerais, Escola de Engenharia.

Inclui bibliografia.

1. Engenharia elétrica - Teses. 2. Galerkin, Métodos de - Teses. 3. Eletromagnetismo - Simulação por computador - Teses. 4. Computação paralela. - Teses. I. Silva, Elson José da. II. Universidade Federal de Minas Gerais. Escola de Engenharia. III. Título.

CDU: 621.3(043)

Ficha catalográfica elaborada pelo bibliotecário Reginaldo César Vital dos Santos CRB/6 2165
Biblioteca Prof. Mário Werneck, Escola de Engenharia da UFMG

UNIVERSIDADE FEDERAL DE MINAS GERAIS

Escola de Engenharia

COLEGIADO DO CURSO DE GRADUAÇÃO / PÓS-GRADUAÇÃO EM Engenharia Elétrica

FOLHA DE APROVAÇÃO

"Parallel-GPU DGTD Method With aThird-order Local Time Stepping Scheme"

Marlon Jesus Lizarazo Urbina

Tese de Doutorado submetida à Banca Examinadora designada pelo Colegiado do Programa de Pós-Graduação em Engenharia Elétrica da Escola de Engenharia da Universidade Federal de Minas Gerais, como requisito para obtenção do grau de Doutor em Engenharia Elétrica.

Aprovada em 30 de abril de 2025.

Por:

Referência: Processo nº 23072.223490/2025-39 SEI nº 4157402

Acknowledgments

O autor desta tese expressa seus agradecimentos:

A Dios, principalmente, por darme paciencia, fuerza y sabiduría para nunca rendirme y así poder alcanzar mis objetivos.

A mis padres, Cleofelina Urbina y Luis Lizarazo, por su apoyo y amor incondicional durante toda mi vida. Para ellos, mi mayor respeto y gratitud.

A mi hermano, Brayan Lizarazo, por su compañía y por los buenos momentos que hemos compartido y los que aún viviremos en nuestro camino por este mundo.

A mis abuelos, Evencio Urbina y Aminta Bernal, por ser mis segundos padres y brindarme todo su amor y cariño.

A mis tíos (Carmen, Yurley, Germán, Chuchin y Ana) y primos (Yurley Andrea, Camilo, Santiago, Nikolita, Sofía, Steven, Carlos y Robinson) por todos los buenos momentos, risas y su apoyo incondicional.

A María Africano, por acompañarme, apoyarme y, sobre todo, por siempre confiar en mí.

A mis amigos que se convirtieron en mi familia en Brasil (José, Andrés, Luigy y Diego), gracias por los buenos momentos, muchachos.

Ao meu orientador, o professor Elson José da Silva, pela paciência, pelos ensinamentos e pelo acompanhamento desde o mestrado até esta etapa.

Ao professor Ricardo Adriano, pelo apoio, pelos ensinamentos e por estar sempre disposto a ajudar.

Aos meus colegas de laboratório, sempre dispostos a ajudar. Obrigado pelos bate-papos e momentos de descontração.

Aos meus colegas de trabalho do IFMG - Campus Formiga, especialmente aos professores Mário, Édio e Everthon. Obrigado pela amizade e pelos bons momentos.

Ao PPGEE, pela formação acadêmica e profissional proporcionada neste doutorado.

À CAPES, pela bolsa de estudo.

“Dada tu buena fortuna, debes hacer lo que puedas para mejorar la suerte de los demás.”

Resumo

O uso crescente de métodos numéricos para resolver problemas eletromagnéticos de grande e multiescala tem impulsionado o desenvolvimento de estratégias para aumentar a eficiência do método de Galerkin Discontinuo no Domínio do Tempo (DGTD), sem comprometer a precisão. Este trabalho apresenta a combinação de duas dessas estratégias, visando melhorar o desempenho do DGTD e reduzir o tempo de execução. A primeira estratégia utiliza Unidades de Processamento Gráfico (GPUs) para acelerar os cálculos, aproveitando sua baixa latência e alto paralelismo. A segunda emprega uma técnica de avanço no tempo local (LTS), que permite que elementos da malha avancem de forma independente, evitando as limitações de um passo de tempo global (GTS). O estudo começa com a descrição das discretizações espacial e temporal do DGTD. Em seguida, é apresentada uma introdução às GPUs, com destaque para suas principais características e uma proposta de distribuição eficiente de dados para os cálculos. Depois, é introduzida uma abordagem LTS baseada no método de Runge-Kutta de terceira ordem (RK3), mantendo a precisão com polinômios do mesmo grau. Após o desenvolvimento das estratégias, elas são combinadas em uma técnica numérica mais eficiente. Para validar a proposta, são resolvidos problemas eletromagnéticos em duas e três dimensões. Testes iniciais em meios homogêneos, como uma cavidade metálica preenchida com ar, demonstram a precisão e o desempenho das estratégias que utilizam memória compartilhada e global da GPU, alcançando acelerações de até 24$\times$ em comparação com implementações em CPU. Validações adicionais mostram que o algoritmo LTS-RK3 preserva a precisão numérica ao mesmo tempo em que reduz o tempo de simulação em até 52% em um problema de espalhamento eletromagnético, quando comparado com a abordagem padrão GTS. Por fim, a estratégia combinada é aplicada a problemas complexos e multiescala, como o espalhamento por uma esfera multicamadas e a radiação de uma antena monopolo, alcançando reduções de tempo de aproximadamente 78% e 55%, respectivamente. Esses resultados confirmam que o método proposto melhora significativamente o desempenho computacional sem comprometer a precisão, superando a implementação padrão com GTS.

Palavras-chave: computação paralela; DGTD; LTS; GTS; problemas eletromagnéticos multiescala.

Abstract

The increasing use of numerical methods to solve large-scale and multiscale electromagnetic problems has driven the development of various strategies to enhance the efficiency of the Discontinuous Galerkin Time-Domain (DGTD) method without compromising accuracy. This work presents the combination of two such strategies aimed at improving the performance of the DGTD method and reducing execution time. The first strategy leverages Graphics Processing Units (GPUs) to accelerate computations by exploiting their low latency and high parallelism. The second employs a local time-stepping (LTS) technique, which allows different mesh elements to advance in time independently, thus avoiding the limitations imposed by a global time step (GTS). The study begins with a description of the spatial and temporal discretizations of the DGTD method. This is followed by an introduction to GPUs, highlighting their main characteristics and presenting an efficient data distribution scheme for executing DGTD computations. An LTS approach based on the third-order Runge-Kutta (RK3) method is then introduced, using a third-order polynomial to maintain accuracy. After developing both strategies, they are combined to form a more powerful and efficient numerical technique. To validate this approach, two-dimensional and three-dimensional electromagnetic problems are solved. Initial tests in homogeneous media, such as a metallic air-filled cavity, demonstrate the accuracy and performance of both shared and global GPU memory strategies, achieving speedups of up to 24$\times$ compared to CPU implementations. Further validation shows that the LTS-RK3 algorithm preserves numerical accuracy while reducing simulation time by up to 52% in an electromagnetic scattering problem when compared to the standard GTS approach. Finally, the combined strategy is applied to complex and multiscale problems, such as scattering by a multilayer sphere and radiation from a monopole antenna, achieving time reductions of nearly 78% and 55%, respectively. These results confirm that the proposed method significantly enhances computational performance while maintaining accuracy, outperforming the standard GTS implementation.

Keywords: parallel computing; DGTD; LTS; GTS; multiscale electromagnetic problems.

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

BC	Boundary Conditions.
CFL	Courant-Friedrichs-Lewy.
CPLTS	Causal Path Local Time Stepping.
CPU-DGTD-GTS	serial CPU-DGTD with GTS.
CUDA	Compute Unified Device Architecture.
DG	Discontinuous Galerkin.
DGTD	Discontinuous Galerkin Time Domain.
DOF	Degrees of Freedom.
FD	Frequency Domain.
FDTD	Finite Difference Time Domain.
FEM	Finite Element Method.
FFT	Fast Fourier Transform.
FVTD	Finite-Volume Time Domain.
GPR	Ground Penetrating Radar.
GPU-DGTD-LTS	Parallel-GPU DGTD method with LTS.
GPU	Graphics Processing Unit.
GPU-DGTD-GTS	Parallel-GPU DGTD method with GTS.
GTS	Global Time Stepping.
HDGTD	Hybridized Discontinuous Galerkin Time Domain.
HPC	High-Performance Computing.
IPDG	Interior Penalty Discontinuous Galerkin.
LF2	Second-order Leapfrog.
LF	Leapfrog.

LSRK4	Fourth-order Low-Storage Runge-Kutta.
LTS	Local Time Stepping.
MPI	Message Passing Interface.
MRT	Minimal Roundtrip.
NF/FF	Near-Field/Far-Field.
ODE	Ordinary Differential Equation.
PEC	Perfect Electric Conductor.
PMC	Perfect Magnetic Conductor.
PML	Perfectly Matched Layers.
RCS	Radar Cross-Section.
RK2	Second-order Runge-Kutta.
RK3	Third-order Runge-Kutta.
RK4	Fourth-order Runge-Kutta.
RK	Runge-Kutta.
RTCG	Run Time Code Generation.
SFZ	Scattered-Field Zone.
SIMD	Single Instruction Multiple Data.
SIMT	Single Instruction Multiple thread.
Si	Silicon.
SMA	Silver-Muller Absorbing.
SM	Streaming Multiprocessor.
TD	Time Domain.
TEM	Transverse electromagnetic.
TE	Transverse Electric.
TF/SF	Total-Field/Scattered-Field.
TFZ	Total-Field Zone.

TL	Transmission Line.
TM	Transverse Magnetic.
UPML	Uniaxial Perfectly Matched Layers.

LIST OF SYMBOLS

$\mathbf{E}$	Total electric field.
$\mathbf{J}$	Electric current density.
$\mathbf{H}$	Total magnetic field.
$\mathbf{D}$	Electric flux density.
$\mathbf{B}$	Magnetic flux density.
ε_0	Vacuum permittivity.
μ_0	Vacuum permeability.
ε_r	Relative permittivity.
μ_r	Relative permeability.
σ	Electric conductivity.
Q	Material matrix.
$\mathbf{q}$	State vector.
$\mathbf{F}(\mathbf{q})$	Flux term.
$\mathbf{F}^*(\mathbf{q})$	Numerical flux.
$\widehat{\mathbf{n}}$	Normal vector.
Z	Media impedance.
Y	Media admittance.
$\overline{\Lambda}$	Metric tensor.
$\mathbf{E}^{inc}$	Incident electric field.
$\mathbf{H}^{inc}$	Incident magnetic field.
$\mathbf{V}^{inc}$	Incident voltage.
Δ	Antenna gap width.
$\widehat{\mathbf{n}}_g$	Normal vector to the gap.

$\mathbf{M}_s$	Surface magnetic current.
I_{tot}	Total electric current.
N_p	Element nodes.
N_{fp}	Face nodes.
L_i	Lagrange polynomial.
$\mathcal{M}$	Mass matrix.
$\mathcal{S}$	Stiffness matrix.
$\mathcal{D}$	Differentiation matrix.
V	Vandermonde matrix.
Δt	Time step.
c	Speed of light in vacuum.
K	Number of elements.
χ	Third-order interpolating polynomial.
ω	Angular frequency.
η	Refractive index.
λ	Wavelength.
h	Maximum edge size factor.

CONTENTS

1 Introduction

This chapter introduces the principles of the discontinuous Galerkin time domain method and the state of art for different techniques used in the literature to improve its computational efficiency. A brief review of some parallel-GPU and Local Time Stepping (LTS) approaches for solving electromagnetic problems modeled by Maxwell's equations is presented.

1.1 Introduction to the DGTD Method and State of Art

Nowadays, the complexity of large-scale and multi-scale electromagnetic problems requires different kinds of advanced computational methods to solve Maxwell's equations. In this sense, the Discontinuous Galerkin Time Domain (DGTD) method appears as a popular, efficient, and accurate option to solve transient electromagnetic problems [1–4]. The DGTD method combines some advantages of other numerical methods such as the Finite-Difference Time Domain (FDTD) [5], the Finite Element Method (FEM) [6], and the Finite-Volume Time Domain (FVTD) [7]. As in the FDTD method, the DGTD presents an interesting simplicity in its implementation, simple parallelization, and easy portability to Graphics Processing Units (GPU). Moreover, the DGTD also has some advantages of the FEM method such as the adaptability of unstructured meshes and high-order spatial convergence. Finally, as in the FVTD method, the DGTD uses an approximation to guarantee the continuity of the solution between neighboring elements, known as the numerical flux. All these features make the DGTD an optimal alternative and a powerful numerical technique for solving large-scale and multi-scale electromagnetic problems.

Unstructured meshes with high-order finite elements are used in the discontinuous Galerkin spatial discretization. This allows accurate discretization of complex geometries using elements of different sizes ($h-adaptivity$), and high-order convergence of the solution can be obtained depending on the order of the basis functions ($p-adaptivity$). Furthermore, the Discontinuous Galerkin (DG) method can be applied both in the Time Domain (TD) and in the Frequency Domain (FD). The TD nature of this method offers many benefits in electromagnetic problems when compared to its FD counterpart. Problems involving the transient field effect of an arbitrary time signal excitation (e.g. scattering problems, ultra-wideband antennas, photonic crystal guides) can be studied directly and efficiently.

In addition, since the DG method uses discontinuous basis functions, the resulting mass matrix is block diagonal. This feature makes the DG method fully explicit and inherently parallelizable when combined with explicit time step methods [8]. Despite the DGTD success so far, the complexity of modern electromagnetic applications requires robust and efficient numerical methods, especially for large-scale and multi-scale problems. These requirements are even more critical when simulations are performed in the temporal domain. Recent hardware architectures such as multi-core CPUs and GPUs are widely available for computing. Consequently, numerical methods with the potential for parallelism that can be properly mapped to newer hardware architectures are also highly desirable.

In recent years, the interest in the use of multiple processors to accomplish difficult tasks with high efficiency has increased. This is known as High-Performance Computing (HPC) and basically refers to the practice of aggregating computing power in order to obtain a higher performance than one could get out of a common computer or workstation. This computing type is used to solve large-scale problems in science, engineering, and business. More recently, high-performance computing has evolved significantly due to the use of heterogeneous architectures, which refer to systems that use more than one kind of processor or cores such as CPU + GPU systems. To support joint CPU + GPU execution of an application, NVIDIA designed a programming model called Compute Unified Device Architecture (CUDA) [9]. In this type of architecture, the CPU is used for complex serial calculations, while the GPU is used for parallel computing tasks due to its excellent performance and high energy efficiency. Therefore, the use of GPUs for accelerated calculations has become increasingly popular in computational electromagnetic simulations [8, 10, 11].

These features turn the GPU into a better candidate than the CPU to accelerate the DGTD method. Emphasizing only contributions dealing with wave propagation problems in the DGTD method, GPUs were considered for the first time for computational electromagnetic applications in 2009 by Klockner *et al.* [12], where the development of a parallel-GPU DGTD method to solve Maxwell's equations in a 3D domain using an unstructured mesh was described. They showed that with a single NVIDIA GTX 280 GPU it was possible to accelerate the simulation time by a factor of 40 to 60 compared to serial computing on a CPU. On the other hand, Cabel *et al.* [13] show the implementation of the DGTD method to study human exposure to electromagnetic waves using a multi-GPU scheme. In that work, the calculation of electromagnetic field components was divided into three CUDA kernels, such as calculation of volume integrals, calculation of surface integrals, and updating of field components. The results presented in [13] showed an acceleration of a factor of 10 to 25 in the simulation time compared to the time spent by the CPU. To achieve those contributions, the authors used the original CUDA programming model based on the template mechanism of the C programming language. However, PyCUDA is presented in 2013 as a practical and mature open-source toolkit that supports GPU Run Time Code Generation (RTCG) [14]. As its name suggests, PyCUDA provides the connection between the high-level Python programming language and the NVIDIA CUDA compute architecture [15]. The GPU is optimally suited to the efficiency throughput while the CPU is responsible just for control and communication. That is, both GPU and CPU work at a higher level of abstraction. Therefore, a Python GPU code has no problem achieving the same performance potential as a C-controlled GPU code, but with the advantage of reducing the effort on the part of the programmer [16].

In more recent works, it can be seen that the application of the GPU to accelerate the DGTD method is still an area of great interest [8, 17–20]. This formulation has been used in multiple applications, such as the analysis of electromagnetic problems of electrically large objects [17, 19], instantaneous nonlinear effects on electromagnetic fields due to the field-dependent medium permittivity [18], antenna simulations [8], Ground Penetrating Radar (GPR) simulations [20], among others. However, from 2019 until now, we can see a clear increase in the use of a methodology that combines GPU parallelism and the possibility of using a time march with LTS [2, 21–23]. This combination significantly improves the computational efficiency of the DGTD method, especially when dealing with multi-scale problems (e.g. electromagnetic scattering problems).

The DGTD method supports time integration schemes based on implicit or explicit techniques, both approaches have advantages and disadvantages. For example, implicit time integration methods are unconditionally stable, i.e., the numerical stability does not depend on the time step. However, they require solving a linear equations system at each time step which compromises the efficiency of the method. On the other hand, explicit methods are considered simpler because no equations have to be solved to advance in time. Explicit methods use the current solution evaluated in intermediary stages to obtain the next time step solution. However, explicit methods are conditionally stable, which means that the time step is subject to a Courant-Friedrichs-Lewy (CFL) condition to maintain stability. This feature turns the explicit methods inefficient when dealing with multi-scale problems because the time step is chosen according to the minimal element size in the mesh to guarantee the stability CFL condition. In order to overcome these limitations, two approaches have been adopted in the literature: the first approach tries to combine the advantages of the implicit and explicit methods, but this scheme presents a considerable loss in both accuracy and stability [24]. The second approach uses explicit methods with local time stepping. These strategies based on LTS are most commonly used because of their simplicity and good results. Many LTS strategies have been proposed to improve the performance of the DGTD method in terms of computational efficiency. These strategies allow different size elements, to march arbitrarily in time while maintaining the stability of the solution. These LTS methods focus on two important aspects, which strongly affect the accuracy and computational efficiency of the temporal integration: the temporal integration method and the way the numerical flux is imposed.

The first LTS scheme to solve electromagnetic problems modeled by Maxwell's curl equations in the DGTD method was presented in 2006 by Piperno [24]. In that work, the author presents an implicit-explicit formulation, where the implicit part is performed using the Verlet method [25] and the midpoint rule. Then, the explicit part uses the well-known Leapfrog (LF) time scheme, which is a central difference method [26]. Piperno's work [24] demonstrated strong computational efficiency. However, the author clearly states that the LTS scheme should not be used to achieve higher accuracy compared to the global time step scheme. Montseny *et al.* [27] presented a recursive LF method in 2008. This method eliminates the implicit part of [24] by using the second-order LF (LF2) method in a recursive process. Cui *et al* [28] then proposed a new version of the Montseny recursive LF2 scheme in 2018, which uses a better element distribution. They achieved an improvement of almost 25% in the time simulation with a small loss of precision.

Currently, the most used time integration schemes applied in the DG scheme are based on the explicit Runge-Kutta (RK) methods [29, 30]. These methods are extremely useful because they provide a high order of approximation as well as the capacity to march in time with a large time step. There are many LTS strategies based on RK methods, for example, Trajan *et al.* [31] proposed an LTS scheme based on the Second-order RK (RK2) method to solve the shallow water equations in 2012. The results show that the LTS-RK2 scheme maintains the second-order convergence of the common RK2 and provides an interesting speed-up. Angulo *et al.* [32] proposed the Causal Path Local Time Stepping (CPLTS) method in 2014, applied in two different time integration methods, the LF2 and the Fourth-order Low-Storage RK (LSRK4). These methods were tested in electromagnetic problems and the results show a reduction in the numerical dissipation and dispersion when compared to the Montseny approach. Ashbourne [33] presented an efficient and precise RK LTS implementation in 2016, applicable to both RK3 and RK4 methods. The study employs interpolations matching the approximation order of the time integration methods (i.e., third-order interpolation for RK3 and fourth-order interpolation for RK4) to maintain continuity of the time solution across elements advancing with different time step sizes.

The benefits of applying GPU parallel computing and LTS schemes within the DG method have been demonstrated independently and in isolation. However, the combination of these techniques offers a more efficient numerical approach. Recent literature highlights two significant works that explore this integration in electromagnetic simulations (parallel-GPU DG method with LTS): The first study, presented by Shi *et al.* in 2019 [8], introduced a Hybridized DGTD method (HDGTD). This method combines the Interior Penalty Galerkin Discontinuous approach (IPDG) based on the Helmholtz equation with the standard DGTD method based on Maxwell's curl equations. The implementation also utilized an LF2 LTS scheme that applies a simple linear interpolation to ensure temporal continuity. However, the use of first-order interpolation limits the method's accuracy. A subsequent study by Ban *et al.* in 2020 [22] improved the HDGTD method by incorporating universal matrices, reducing memory usage and minimizing data exchange between the GPU and CPU. Despite this enhancement, the LTS scheme continued to rely on the LF2 method with first-order interpolation for time continuity. Recent advancements in hybrid parallel strategies have further exploited the DGTD method's parallel capabilities through supercomputing and Message Passing Interface (MPI) algorithms. For instance, in [23], a unified MPI+MPI algorithm achieved a parallel efficiency of 94% using 6400 cores. This work employed an LTS scheme based on Montseny's method [27], which offers a second-order approximation through a recursive LF2 approach. In another study, Li *et al.* [34] introduced the Minimum Number of Roundtrips (MNR) strategy to optimize communication topology across 16,000 supercomputer nodes. Despite achieving a parallel efficiency of 73.8%, this work relied on Montseny's method with first-order interpolation. Later, Li *et al.* [35] proposed a Minimal Roundtrip (MRT) strategy to balance communication loads in the DGTD method. This approach halved inter-processor communication time, yet it continued to use a second-order LTS scheme based on Montseny's method.

As can be seen, research on the parallelism of the DGTD-LTS method has advanced significantly, leveraging both CPU and GPU acceleration. However, a clear limitation persists in the reliance on LTS schemes restricted to LF2 with first-order interpolation. To address this gap, this work introduces an alternative approach to the parallel-GPU DGTD method with LTS schemes for solving electromagnetic problems. The proposed method employs high-order interpolations to ensure the continuity of the time solution. This parallel-GPU implementation was performed considering NVIDIA recommendations [15] to ensure optimal GPU performance. Additionally, the LTS scheme is based on the efficient Third-order Runge–Kutta (RK3) method [33]. The proposed LTS strategy incorporates third-order interpolations to ensure continuity between elements of different classes while maintaining the same precision order as the standard RK3 method.

1.2 Motivation and Contributions

1.2.1 Motivation

In recent years, various Discontinuous Galerkin Time Domain (DGTD) methods for solving Maxwell's equations have been developed. Among them, one of the most widely adopted and efficient approaches was introduced by Hesthaven *et al.* in 2002 [36]. As previously discussed, this method incorporates several advantages characteristic of other well-known numerical techniques, such as FEM, FDTD, and FVTD. To address large-scale and multi-scale electromagnetic problems, numerous modifications have been proposed in the literature to enhance the computational efficiency of the DGTD method. These enhancements often aim to accelerate computational operations by leveraging the low latency of GPUs or implementing an LTS scheme. The LTS approach ensures that each element advances in time based on its local size rather than being constrained by the smallest element in the computational domain. Both strategies are extensively documented and have demonstrated significant improvements in performance. Furthermore, hybrid methods combining GPU acceleration with LTS schemes have been explored [8,22], offering further computational gains. However, despite their effectiveness, these approaches often employ the LF2 time integration method and linear interpolation for coupling solutions across elements with varying time step sizes, which can compromise the accuracy of the method.

This work is motivated by the need to explore alternative implementations of a GPU-parallelized DGTD method with an LTS scheme that preserves high accuracy through the use of advanced interpolation techniques. As highlighted earlier, several LTS approaches based on the Runge-Kutta (RK) family of methods can be applied to this framework. One promising candidate is the causal path LTS scheme based on the LSERK4 method. This scheme offers fourth-order accuracy with low storage requirements. However, the need to compute five intermediate stages per time step can reduce its efficiency compared to other methods. Another alternative is the LTS RK4 scheme, which relies on the standard fourth-order Runge-Kutta (RK4) method [33]. While this approach maintains fourth-order accuracy and reduces the number of intermediate stages to four, it demands significant memory storage and suffers from a limited stability region. Finally, the LTS RK3 scheme, based on the standard third-order Runge-Kutta (RK3) method [33], provides a compelling alternative. It requires the computation and storage of only three intermediate stages per time step, resulting in lower memory demands. Although it has a slightly lower order of accuracy, the LTS RK3 scheme benefits from a broader stability region. Considering these factors, we have selected the LTS RK3 method as the most suitable approach for our study. Additionally, this method employs an efficient third-order interpolation to ensure continuity of the time solution between elements with differing time step sizes, further enhancing the accuracy and performance of the proposed DGTD framework.

1.2.2 Contributions

The main contribution of this thesis lies in the integration of two advanced acceleration strategies based on parallel-GPU computation and a high-order LTS scheme within the framework of the DGTD method. This combination is specifically designed to address the challenges of solving multi-scale electromagnetic problems efficiently and accurately. To realize this goal, several key steps were undertaken during the development of this work. These steps, which form the core of the methodology, are summarized below:

- Proposal of efficient parallel-GPU algorithms for handling element-wise operations and matrix-vector multiplications, which form the core of the CUDA kernels proposed in this work. These algorithms were developed following NVIDIA's recommendations to maximize the utilization of GPU capabilities.

- Proposal of an efficient LTS RK3 implementation for GPUs, designed to handle various levels of refinement while ensuring continuity between different element classes through the use of third-order polynomials. The final version of this implementation is capable of solving complex and multi-scale 2D and 3D electromagnetic problems, demonstrating its scalability and robustness in addressing challenging scenarios.
- Proposal of a novel DGTD method combining efficient GPU algorithms with the LTS RK3 scheme and third-order interpolation. The proposed DGTD method was rigorously validated by solving a range of 2D and 3D benchmark electromagnetic problems, demonstrating its accuracy, efficiency, and robustness across different scenarios.
- Proposal of the novel parallel-GPU DGTD with LTS method as an efficient and accurate powerful numerical technique for solving complex multi-scale electromagnetic problems.

Furthermore, the individual contributions mentioned above were instrumental in the development of the following journal paper, which was successfully accepted and subsequently published:

- Lizarazo, M.J.; Silva, E.J. A Parallel-GPU DGTD Algorithm with a Third-Order LTS Scheme for Solving Multi-Scale Electromagnetic Problems. *Mathematics* 2024, 12, 3663. https://doi.org/10.3390/math12233663 [37]

1.3 Thesis Organization

This work is organized into six chapters, including this one. Chapter 2 contains the principles of the DGTD method for solving Maxwell's equations. Spatial and time discretizations are described. Chapter 3 is devoted to the basic concepts of NVIDIA GPUs and the optimal data distribution to ensure the best GPU performance for the DGTD method. In chapter 4 the LTS RK3 method along with its modified implementation to be executed in the GPU are presented. The results of the parallel-GPU DGTD method, the DGTD with the LTS RK3 scheme, and the combination of both are presented in chapter 5. Finally, chapter 6 presents the conclusions and future research work.

2 The Discontinuous Galerkin Time Domain Method for Solving Maxwell's Equations

The DGTD method was proposed in the 1970s by Reed and Hill [38] to solve the linear neutron transport equation. This method is currently used in many areas of science due to its high-order convergence and simple implementation. It has successfully been used to solve a lot of differential equations in some areas as aerodynamics [39], nano-optical problems [40], elastodynamics [41,42], and quantum mechanics [43]. In the last years, some DGTD versions to solve Maxwell's equation have been proposed, one of the most popular and efficient was proposed in 2002 by Hesthaven and Warburton [36]. This chapter presents the modeling of Maxwell's equations using the DGTD method in non-dispersive dielectric media and some numerical integration schemes based on the Runge-Kutta methods are described [44].

2.1 Maxwell's Equations

The partial differential equations system that models the phenomena of wave propagation and wave interaction with objects is known as Maxwell's equations. These equations can be expressed as follows:

$$\nabla \times \mathbf{E} = -\frac{\partial \mathbf{B}}{\partial t} \tag{2.1a}$$

$$\nabla \times \mathbf{H} = \frac{\partial \mathbf{D}}{\partial t} + \mathbf{J} \tag{2.1b}$$

$$\nabla \cdot \mathbf{D} = \rho \tag{2.1c}$$

$$\nabla \cdot \mathbf{B} = 0 \tag{2.1d}$$

where Eqs. 2.1a and 2.1b are known as Faraday's law of induction and Ampère's law, respectively. Whereas Eqs. 2.1c and 2.1d are the Gauss' laws for electricity and magnetism, respectively. **E** and **H** are the electric and magnetic field intensities, respectively. **D** and **B** are the electric and magnetic flux densities. **D** is also called the electric displacement, and **B**, the magnetic induction. The electric and magnetic flux densities **D**, **B** are related to the field intensities **E**, **H** by the so-called constitutive relations. In a homogeneous and isotropic medium, they take the form:

$$\mathbf{D} = \varepsilon_0 \varepsilon_r \mathbf{E} \tag{2.2a}$$
$$\mathbf{B} = \mu_0 \mu_r \mathbf{H} \tag{2.2b}$$

where ε_0 is the electric permittivity, μ_0 is the magnetic permeability of vacuum. ε_r and μ_r are the relative electric permittivity and magnetic permeability of the medium, respectively.

Considering a dielectric medium $\mu_r = 1$ without currents $J = 0$ and substituting Eq. 2.2 in 2.1, the Maxwell curl equations can be represented as:

$$\nabla \times \mathbf{E} = -\mu_0 \frac{\partial \mathbf{H}}{\partial t} \tag{2.3a}$$
$$\nabla \times \mathbf{H} = \varepsilon \frac{\partial \mathbf{E}}{\partial t} \tag{2.3b}$$

where $\varepsilon = \varepsilon_0 \varepsilon_r$.

These curl equations can be divided into their respective vector components by applying the curl operation in Cartesian coordinates. The application of this curl operator yields three scalar equations both Faraday and Ampère laws, that is, a set of six scalar equations are needed to solve a 3D problem in cartesian coordinates. On the other hand, the non-variation in one spatial direction (e.g. $\hat{z}$-direction) must be considered when handling 2D problems. This non-variation implies that all partial derivatives of the fields with respect to $\hat{z}$ are eliminated. The set of six scalar equations which represent the field components of the Maxwell's equations in Cartesian coordinates is given by:

$$\mu_0 \frac{\partial H_x}{\partial t} = -\frac{\partial E_z}{\partial y} + \frac{\partial E_y}{\partial z} \tag{2.4a}$$
$$\mu_0 \frac{\partial H_y}{\partial t} = -\frac{\partial E_x}{\partial z} + \frac{\partial E_z}{\partial x} \tag{2.4b}$$
$$\mu_0 \frac{\partial H_z}{\partial t} = -\frac{\partial E_y}{\partial x} + \frac{\partial E_x}{\partial y} \tag{2.4c}$$
$$\varepsilon \frac{\partial E_x}{\partial t} = \frac{\partial H_z}{\partial y} - \frac{\partial H_y}{\partial z} \tag{2.4d}$$
$$\varepsilon \frac{\partial E_y}{\partial t} = \frac{\partial H_x}{\partial z} - \frac{\partial H_z}{\partial x} \tag{2.4e}$$
$$\varepsilon \frac{\partial E_z}{\partial t} = \frac{\partial H_y}{\partial x} - \frac{\partial H_x}{\partial y} \tag{2.4f}$$

Considering the non-variation in the $\hat{z}$-direction, the set of three scalar equations which represent the Transversal Magnetic to z (TMz) mode of Maxwell's equations in 2D is:

$$\mu_0 \frac{\partial H_x}{\partial t} = -\frac{\partial E_z}{\partial y} \tag{2.5a}$$

$$\mu_0 \frac{\partial H_y}{\partial t} = \frac{\partial E_z}{\partial x} \tag{2.5b}$$

$$\varepsilon \frac{\partial E_z}{\partial t} = \frac{\partial H_y}{\partial x} - \frac{\partial H_x}{\partial y} \tag{2.5c}$$

As can be seen in Eq. 2.5 the field components only vary spatially in the x-y coordinates, that is, a variation in the $\hat{z}$ plane vanishes allowing the analysis of 2D problems. It is important to remark that there is another mode called Transverse Electric (TE) which can be used to analyze 2D problems. These two modes TM and TE do not have any field component in common. Therefore, they can coexist without influencing each other.

2.2 Conservation Form

The conservation form is used to simplify the representation of Maxwell's equation. This formulation is very useful because Eq. 2.3 can be summarized into just one equation considering the unknown fields and fluxes as a vector. Also, media information is represented as a matrix. The conservation form of Maxwell's equation is given by:

$$Q \frac{\partial \mathbf{q}}{\partial t} + \nabla \cdot \mathbf{F}(\mathbf{q}) = \mathbf{S} \tag{2.6}$$

where Q is the material matrix with the media information:

$$Q = \begin{bmatrix} \mu & 0 \\ 0 & \varepsilon \end{bmatrix} \tag{2.7}$$

The state vector $\mathbf{q}$ is given by:

$$\mathbf{q} = \begin{bmatrix} \mathbf{H} \\ \mathbf{E} \end{bmatrix} \tag{2.8}$$

The flux term $\mathbf{F}(\mathbf{q})$ can be represented as:

$$\mathbf{F}_i(\mathbf{q}) = \begin{bmatrix} -e_i \times \mathbf{E} \\ e_i \times \mathbf{H} \end{bmatrix} \tag{2.9}$$

With $\mathbf{F}(\mathbf{q}) = \Big[\mathbf{F}_x(\mathbf{q}), \quad \mathbf{F}_y\mathbf{q}), \quad \mathbf{F}_z(\mathbf{q})\Big]^T$. Here e_i signifies the three Cartesian unit vectors, where $i = x, y, z$ and $\mathbf{S} = \Big[\mathbf{S}^E, \quad \mathbf{S}^H\Big]^T$ represent sources, currents, and terms introduced by the scattered field formulation. In this work, the source term is set to $\mathbf{S} = 0$ for simplicity.

Rewriting the set of Eq. 2.4 in the conservation form:

$$Q = \begin{pmatrix} \mu_0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \mu_0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \mu_0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \varepsilon & 0 & 0 \\ 0 & 0 & 0 & 0 & \varepsilon & 0 \\ 0 & 0 & 0 & 0 & 0 & \varepsilon \end{pmatrix} \quad ; \mathbf{q} = \begin{bmatrix} H_x \\ H_y \\ H_z \\ E_x \\ E_y \\ E_z \end{bmatrix} \quad ; \nabla \cdot \mathbf{F}(\mathbf{q}) = \begin{pmatrix} -\frac{\partial E_z}{\partial y} + \frac{\partial E_y}{\partial z} \\ -\frac{\partial E_x}{\partial z} + \frac{\partial E_z}{\partial x} \\ -\frac{\partial E_y}{\partial x} + \frac{\partial E_x}{\partial y} \\ \frac{\partial H_z}{\partial y} - \frac{\partial H_y}{\partial z} \\ \frac{\partial H_x}{\partial z} - \frac{\partial H_z}{\partial x} \\ \frac{\partial H_y}{\partial x} - \frac{\partial H_x}{\partial y} \end{pmatrix}$$

Now, Rewriting the set of Eq. 2.5 in the conservation form:

$$Q = \begin{pmatrix} \mu_0 & 0 & 0 \\ 0 & \mu_0 & 0 \\ 0 & 0 & \varepsilon \end{pmatrix} \quad ; \mathbf{q} = \begin{bmatrix} H_x \\ H_y \\ E_z \end{bmatrix} \quad ; \nabla \cdot \mathbf{F}(\mathbf{q}) = \begin{pmatrix} -\frac{\partial E_z}{\partial y} \\ \frac{\partial E_z}{\partial x} \\ \frac{\partial H_y}{\partial x} - \frac{\partial H_x}{\partial y} \end{pmatrix}$$

2.3 Local Variational Form

In the DG formalism, the domain Ω is represented by a set of non-overlapping elements, K, typically tetrahedrons for tridimensional problems or triangles for bidimensional problems, which are organized in an unstructured manner in order to geometrically conform the computational domain.

$$\Omega = \bigcup_{k=1}^{K} \Omega_k \tag{2.12}$$

Now, let us consider only a single element of the computational domain. The aim is to find a numerical approximation $\mathbf{q}_h$ of $\mathbf{q}$. For the DG spatial discretization, each element is discontinuous with respect to others. It means that the variational form must be local, therefore, the weak form is obtained by multiplying Eq. 2.6 by a regular test function $L_j(\mathbf{r}, t)$ which minimizes the residue. Finally, we integrate over the element Ω_k.

$$\int_{\Omega_k} \Big[Q\partial_t \mathbf{q}_h(\mathbf{r}, t) + \nabla \cdot \mathbf{F}(\mathbf{q}_h)\Big] L_j(\mathbf{r}, t) d\Omega = 0 \tag{2.13}$$

Now, let us apply Gauss's theorem over Eq. 2.13 to obtain the local statement:

$$\int_{\Omega_k} \Big[Q\partial_t \mathbf{q}_h(\mathbf{r}, t) L_j(\mathbf{r}, t) - \mathbf{F}(\mathbf{q}_h) \cdot \nabla L_j(\mathbf{r}, t)\Big] d\Omega = -\int_{\Gamma_{\Omega_k}} \widehat{\mathbf{n}} \cdot \mathbf{F}(\mathbf{q}_h) L_j(\mathbf{r}, t) d\Gamma \tag{2.14}$$

As can be seen in Eq. 2.14, only field values and derivatives on the element Ω_k are involved. This is a problem because we do not have information about the neighboring elements. Therefore, the connection between neighbor elements is needed to consider their contributions over the element Ω_k. Consequently, it suffices to substitute the flux $\mathbf{F}$ by a numerical flux $\mathbf{F}^*$ as the unique value to be used at the interface and obtained by combining information from both elements. With this, we recover the scheme:

$$\int_{\Omega_k} \Big[Q\partial_t \mathbf{q}_h(\mathbf{r},t) L_j(\mathbf{r},t) - \mathbf{F}(\mathbf{q}_h)\cdot \nabla L_j(\mathbf{r},t) \Big] d\Omega = -\int_{\Gamma_{\Omega_k}} \widehat{\mathbf{n}} \cdot \mathbf{F}^*(\mathbf{q}_h) L_j(\mathbf{r},t) d\Gamma \qquad (2.15)$$

and applying Gauss' theorem once again, we get:

$$\textstyle\int_{\Omega_k} \Big[Q\partial_t \mathbf{q}_h(\mathbf{r},t) + \nabla\cdot \mathbf{F}(\mathbf{q}_h) \Big] L_j(\mathbf{r},t) d\Omega = \int_{\Gamma_{\Omega_k}} \widehat{\mathbf{n}} \cdot \Big[\mathbf{F}(\mathbf{q}_h) - \mathbf{F}^*(\mathbf{q}^-,\mathbf{q}^+) \Big] L_j(\mathbf{r},t) d\Gamma \qquad (2.16)$$

This is the strong variational formulation of Maxwell's curl equations. In the right-hand side of Eq. 2.16, $\widehat{\mathbf{n}}$ represents the outwardly directed normal vector and $\mathbf{F}^*$ is the numerical flux, which is in terms of the local element $\mathbf{q}^-$ and its neighbor $\mathbf{q}^+$.

2.3.1 Numerical flux

The numerical flux was proposed for the first time in the FVM method. It appears to ensure the locality of the scheme and maintain the continuity of the solution [45]. The FVM method was introduced as an alternative to the well-known FDTD method to add geometric flexibility to the spatial discretization. However, this method has lost much popularity due to its low convergence rate [4]. The numerical flux is used in the DG method in order to enforce the continuity of the solution across element edges. In other words, the information through the interface between two elements is carried along the unit normal vector $\widehat{\mathbf{n}}$ [46]. Figure 1 shows the interaction of the flux between two neighbor triangular elements.

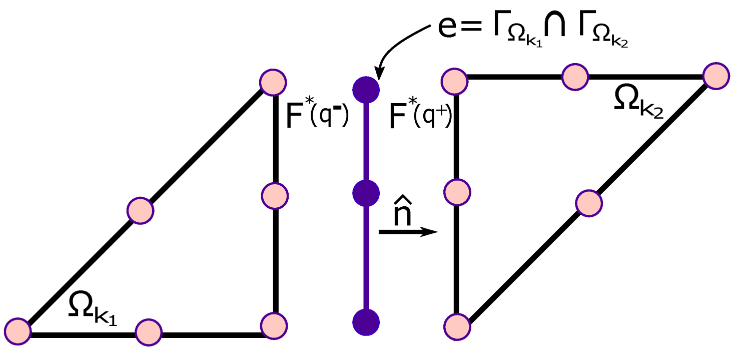


Figure 1 – Neighbor elements with the same edge e.

where e is the interface between elements Ω_{k_1} and Ω_{k_2}.

Moreover, there are many possibilities for choosing the numerical flux such as centered, upwind, and penalized. All of them use a penalization parameter, α, that has an impact on the precision and stability of the solution [47]. The centered flux ($\alpha = 0$) considers the interface solutions of the local and neighbor elements in order to calculate an average value between both solutions [48]. The upwind flux ($\alpha = 1$) is usually employed due to its precision and robustness. It introduces into the scheme some upwind terms that come from the solution of the Riemann problem [49]. Finally, the penalized flux uses a similar formulation of the upwind flux but considers a decimal penalization parameter ($0 < \alpha < 1$). According to [50], the upwind flux is the most used due to the maximum attenuation of non-physical modes. This means that the effects of spurious modes are controlled.

Hesthaven and Warburton presented a numerically stable and convergent scheme using the upwind flux [36]. It is given by:

$$\widehat{\mathbf{n}} \cdot \Big[\mathbf{F}(\mathbf{q}_h) - \mathbf{F}^*(\mathbf{q}^-, \mathbf{q}^+)\Big] = \frac{1}{2} \begin{cases} \bar{Z}^{-1}\Big(\widehat{\mathbf{n}} \times \Big[-Z^+\Delta\mathbf{H} + \alpha\widehat{\mathbf{n}} \times \Delta\mathbf{E}\Big]\Big) \\ \bar{Y}^{-1}\Big(\widehat{\mathbf{n}} \times \Big[Y^+\Delta\mathbf{E} + \alpha\widehat{\mathbf{n}} \times \Delta\mathbf{H}\Big]\Big) \end{cases} \tag{2.17}$$

where $\Delta E = E^- - E^+$ and $\Delta H = H^- - H^+$. $Z^\pm$ and $Y^\pm$ are respectively, the impedance and the conductance of the media:

$$Z^\pm = \sqrt{\frac{\mu^\pm}{\varepsilon^\pm}}, \quad Y^\pm = \frac{1}{Z^\pm} = \sqrt{\frac{\varepsilon^\pm}{\mu^\pm}}$$

$\bar{Z}$ and $\bar{Y}$ are their sums:

$$\bar{Z} = Z^+ + Z^- \quad , \quad \bar{Y} = Y^+ + Y^-$$

In Eq. 2.17 the superscript " $+$ " refers to field values from the neighbor element while superscript " $-$ " refers to field values from local element. The tangential field components are represented by the normal component of the flux. Hence, the objective of the right-hand side in Eq. 2.17 is to enforce the continuity of the tangential field components across the face of the elements. Expanding Eq. 2.17 and using the vector identity $\mathbf{a} \times (\mathbf{b} \times \mathbf{c}) = (\mathbf{a} \cdot \mathbf{c})\mathbf{b} - (\mathbf{a} \cdot \mathbf{b})\mathbf{c}$, the six scalar flux components for Maxwell's equations in 3D are given by:

$$\widehat{\mathbf{n}}\cdot\Big[\mathbf{F}(\mathbf{q}_h)-\mathbf{F}^*(\mathbf{q}^-,\mathbf{q}^+)\Big]=\frac{1}{2}\begin{cases}\bar{Y}^{-1}\Big(n_yY^+\Delta E_z-n_zY^+\Delta E_y-\alpha\Big[\Delta H_x-ndotH\cdot n_x\Big]\Big)\\ \bar{Y}^{-1}\Big(n_zY^+\Delta E_x-n_xY^+\Delta E_z-\alpha\Big[\Delta H_y-ndotH\cdot n_y\Big]\Big)\\ \bar{Y}^{-1}\Big(n_xY^+\Delta E_y-n_yY^+\Delta E_x-\alpha\Big[\Delta H_z-ndotH\cdot n_z\Big]\Big)\\ \bar{Z}^{-1}\Big(-n_yZ^+\Delta H_z+n_zZ^+\Delta H_y-\alpha\Big[\Delta E_x-ndotE\cdot n_x\Big]\Big)\\ \bar{Z}^{-1}\Big(-n_zZ^+\Delta H_x+n_xZ^+\Delta H_z-\alpha\Big[\Delta E_y-ndotE\cdot n_y\Big]\Big)\\ \bar{Z}^{-1}\Big(-n_xZ^+\Delta H_y+n_yZ^+\Delta H_x-\alpha\Big[\Delta E_z-ndotE\cdot n_z\Big]\Big)\end{cases}\tag{2.18}$$

where $ndotH = n_x\Delta H_x + n_y\Delta H_y + n_z\Delta H_z$ and $ndotE = n_x\Delta E_x + n_y\Delta E_y + n_z\Delta E_z$.

Finally, the Eq 2.18 can be represented in the 2D TMz case as:

$$\widehat{\mathbf{n}}\cdot\Big[\mathbf{F}(\mathbf{q}_h)-\mathbf{F}^*(\mathbf{q}^-,\mathbf{q}^+)\Big]=\frac{1}{2}\begin{cases}\bar{Y}^{-1}\Big(n_yY^+\Delta E_z-\alpha\Big[\Delta H_x-ndotH\cdot n_x\Big]\Big)\\ \bar{Y}^{-1}\Big(-n_xY^+\Delta E_z-\alpha\Big[\Delta H_y-ndotH\cdot n_y\Big]\Big)\\ \bar{Z}^{-1}\Big(-n_xZ^+\Delta H_y+n_yZ^+\Delta H_x-\alpha\Delta E_z\Big)\end{cases}\tag{2.19}$$

2.3.2 Boundary conditions

The correct imposition of Boundary Conditions (BC) is mandatory to solve partial differential equations (e.g. Maxwell's equations). This is important because bad BC imposition can lead to the divergence of the solution or to the convergence to a wrong solution. In this work, BCs are crucial to model the wave propagation of electromagnetic fields in a finite computational domain. Previously, it was explained that numerical flux can be used to connect adjacent elements, but it also serves to directly implement basic boundary conditions in weak form, just by modifying the jumps in the factors $\Delta\mathbf{E}$ and $\Delta\mathbf{H}$.

2.3.2.1 Perfect electric conductor (PEC)

The PEC condition requires that the tangential component of the electric field must be null and the tangential magnetic field component to be continuous.

$$\widehat{\mathbf{n}}\times\mathbf{E}=0\tag{2.24}$$

In our case to implement the PEC boundary condition, we use the same mirror principle used in [36]. To the electric field we assigned $\mathbf{E}^+ = -\mathbf{E}^-$. Thus, on a PEC boundary $\Delta\mathbf{E} = 2\mathbf{E}^-$. On the other hand, we assigned $\mathbf{H}^+ = \mathbf{H}^-$ to the magnetic field and consequently, $\Delta\mathbf{H} = 0$.

2.3.2.2 Perfect magnetic conductor (PMC)

The PMC condition is the reciprocal of the PEC one, accordingly:

$$\widehat{\mathbf{n}} \times \mathbf{H} = 0 \tag{2.25}$$

As in the PEC boundary conditions, for the PMC we use $\mathbf{H}^+ = -\mathbf{H}^-$ and $\mathbf{E}^+ = \mathbf{E}^-$. Finally, to the PMC case: $\Delta\mathbf{H} = 2\mathbf{H}^-$ and $\Delta\mathbf{E} = 0$.

2.3.2.3 Silver-Muller absorbing (SMA)

The first order SMA boundary condition provides an ideally null reflection coefficient for normal incidence because it is based on assuming that fields outside the computational domain propagate as normal plane waves to the interface [51].

$$\sqrt{\frac{\varepsilon_0}{\mu_0}}\widehat{\mathbf{n}} \times \mathbf{E} + \widehat{\mathbf{n}} \times \left(\widehat{\mathbf{n}} \times \mathbf{H}\right) = 0 \tag{2.22}$$

To implement this condition, it is enough to ensure that the tangential components of the electric and magnetic fields are null. It means that $\Delta\mathbf{E} = 2\mathbf{E}^-$ and $\Delta\mathbf{H} = 2\mathbf{H}^-$ on boundary nodes. This absorbing boundary condition is useful when the object that interacts with the waves is far enough from the boundary. However, its absorbing characteristics rapidly degrade with the variation of the incident angle [52].

2.3.3 Perfectly matched layers

The Perfectly Matched Layers (PML) technique is widely recognized as the most effective method for truncating open-space domains in electromagnetic simulations. This is largely due to its independence from factors such as frequency, wave polarization, and angle of incidence, which makes it a versatile and robust solution for absorbing outgoing waves. The fundamental concept of PML involves introducing a lossy dielectric layer of specific thickness at the truncated boundaries of the simulation domain. This layer is designed to facilitate the gradual attenuation of outgoing waves, ensuring that their amplitude decreases nearly to zero before they reach the simulation boundary. The seamless impedance matching at the interface ensures that reflections back into the computational domain are minimized, preserving the accuracy of the simulation. In practical implementations, the Uniaxial Perfectly Matched Layer (UPML) is a common variant that extends the PML approach by utilizing anisotropic media. This formulation typically requires fewer auxiliary variables and is computationally cheaper than the Convolutional Perfectly Matched Layer (CPML) because it does not involve memory-intensive convolutions. The governing equations of the PML are derived from Maxwell's equations. For time-harmonic fields in the frequency domain, the general form of Maxwell's equations in the UPML medium is expressed as [53]:

$$\nabla \times \boldsymbol{E} = -j\omega\mu_0\overline{\Lambda}\boldsymbol{H} \tag{2.23a}$$

$$\nabla \times \boldsymbol{H} = j\omega\varepsilon\overline{\Lambda}\boldsymbol{E} \tag{2.23b}$$

where the metric tensor $\overline{\Lambda}$ can be defined as:

$$\overline{\Lambda} = \begin{bmatrix} (SySz)/Sx & 0 & 0 \\ 0 & (SxSz)/Sy & 0 \\ 0 & 0 & (SxSy)/Sz \end{bmatrix} \tag{2.24}$$

with the UPML parameters for $i = x, y, z$ as:

$$S_i = 1 + \sigma_i/j\omega \tag{2.25}$$

The conductivity parameter have a gradual increase in the PML region as $\sigma_i = \sigma^i_{max}(d^i/d^i_{max})$, where d^i denotes the distance from the PML surface and d^i_{max} is the thickness of the PML. According to [54], the σ^i_{max} can be calculated as:

$$\sigma^i_{max} = \frac{0.8(v+1)}{\Delta d\sqrt{\varepsilon_0\mu_0}} \tag{2.26}$$

where Δd is the cell size in the PML region and v is a constant usually chosen to be 2 or 3.

Finally, the Eqs. 2.23a and 2.23b are expanded and converted to the time domain, yielding the next system of differential equations:

$$\mu_0\frac{\partial \mathbf{H}}{\partial t} = -\nabla \times \mathbf{E} - \mathscr{P} - \mu_0\overline{\Lambda}_1\mathbf{H} \tag{2.27a}$$

$$\frac{\partial \mathscr{P}}{\partial t} = -\overline{\Lambda}_2\mathscr{P} + \mu_0\overline{\Lambda}_3\mathbf{H} \tag{2.27b}$$

$$\varepsilon\frac{\partial \mathbf{E}}{\partial t} = \nabla \times \mathbf{H} - \mathscr{Q} - \varepsilon\overline{\Lambda}_1\mathbf{E} \tag{2.27c}$$

$$\frac{\partial \mathscr{Q}}{\partial t} = -\overline{\Lambda}_2\mathscr{Q} + \varepsilon\overline{\Lambda}_3\mathbf{E} \tag{2.27d}$$

where $\mathscr{P}$ and $\mathscr{Q}$ are auxiliary differential equations needed in the UPML formulation and the diagonal tensors $\overline{\Lambda}_1, \overline{\Lambda}_2, \overline{\Lambda}_3$ are given by:

$$\overline{\Lambda}_1 = \text{diag}[\sigma_z + \sigma_y - \sigma_x, \sigma_x + \sigma_z - \sigma_y, \sigma_y + \sigma_x - \sigma_z] \tag{2.28a}$$

$$\overline{\Lambda}_2 = \text{diag}[\sigma_x, \sigma_y, \sigma_z] \tag{2.28b}$$

$$\overline{\Lambda}_3 = \text{diag}[(\sigma_y - \sigma_x)(\sigma_z - \sigma_x), (\sigma_z - \sigma_y)(\sigma_x - \sigma_y), (\sigma_z - \sigma_z)(\sigma_y - \sigma_z)] \tag{2.28c}$$

2.3.4 Sources

The introduction of sources in solving electromagnetic problems is crucial because they influence the behavior and characteristics of the electromagnetic fields involved. Sources can be broadly categorized into two types: far and near [6]. Far sources, often treated as plane waves, simplify the analysis by assuming that the wavefronts are essentially parallel and the amplitude is constant over the region of interest. Near sources, on the other hand, require a more detailed and complex analysis as they produce fields that vary significantly with distance and direction. These sources are critical in applications like antenna design, where the spatial variation of the fields must be precisely understood and controlled. The properly imposition and modeling of these kind of sources ensure accurate solutions to electromagnetic problems. In this work, far sources have been introduced by making use of the Total-Field/Scattered-Field (TF/SF) formulation [55] and near sources are imposed by introducing surface electric and magnetic currents [6, 56].

2.3.4.1 Total-field/scattered-field formulation

The TF/SF formulation is widely used in scattering problems to incorporate incident fields as plane waves. This technique is widely used in FDTD simulations because it allows the division of electromagnetic fields into two components, the total field and the scattered field. This division is possible due to the linearity of Maxwell's equation [57]. The TF/SF formulation can be incorporate in the DGTD method by dividing the computational domain into two zones: the Total-Field Zone (TFZ) and the Scattered-Field Zone (SFZ). Then, the interface between these two zones is used to introduce the incident fields in a weak manner by modifying the flux terms in Eq. 2.17. Figure 2 illustrates the interface between the TFZ and SFZ where the incident field is imposed.

As can be seen in Figure 2, both the TFZ and the SFZ have elements with at least one face lying to the TFZ/SFZ interface. Assuming that within the TFZ a known waveform is propagating, $\mathbf{E}^{inc}$ and $\mathbf{H}^{inc}$, while in the SFZ the total field is zero, the flux

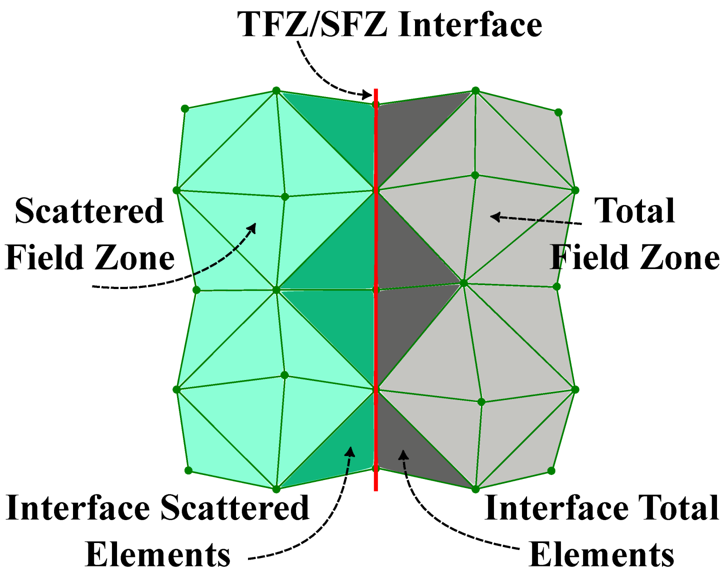


Figure 2 – Total Field/ Scattered Field interface.

terms in the faces nodes of the interface total elements belonging with the TFZ/SFZ interface are modified by:

$$\widehat{\mathbf{n}} \cdot \left[\mathbf{F}(\mathbf{q}_h) - \mathbf{F}^*(\mathbf{q}^-, \mathbf{q}^+)\right] = \frac{1}{2}\begin{cases} \bar{Z}^{-1}\left(\widehat{\mathbf{n}} \times \left[-Z^+(\Delta\mathbf{H} + \mathbf{H}^{inc}) + \alpha\widehat{\mathbf{n}} \times \left(\Delta\mathbf{E} + \mathbf{E}^{inc}\right)\right]\right) \\ \bar{Y}^{-1}\left(\widehat{\mathbf{n}} \times \left[Y^+(\Delta\mathbf{E} + \mathbf{E}^{inc}) + \alpha\widehat{\mathbf{n}} \times \left(\Delta\mathbf{H} + \mathbf{H}^{inc}\right)\right]\right) \end{cases} \tag{2.29}$$

On the other hand, for the face nodes of the interface scattered elements belonging with the TFZ/SFZ interface, the flux terms are modified by:

$$\widehat{\mathbf{n}} \cdot \left[\mathbf{F}(\mathbf{q}_h) - \mathbf{F}^*(\mathbf{q}^-, \mathbf{q}^+)\right] = \frac{1}{2}\begin{cases} \bar{Z}^{-1}\left(\widehat{\mathbf{n}} \times \left[-Z^+(\Delta\mathbf{H} - \mathbf{H}^{inc}) + \alpha\widehat{\mathbf{n}} \times \left(\Delta\mathbf{E} - \mathbf{E}^{inc}\right)\right]\right) \\ \bar{Y}^{-1}\left(\widehat{\mathbf{n}} \times \left[Y^+(\Delta\mathbf{E} - \mathbf{E}^{inc}) + \alpha\widehat{\mathbf{n}} \times \left(\Delta\mathbf{H} - \mathbf{H}^{inc}\right)\right]\right) \end{cases} \tag{2.30}$$

2.3.4.2 The delta-Gap source

The delta-gap source formulation is a classical feed model commonly used for analyzing linear antennas, such as dipoles and monopoles. This approach offers a cost-effective alternative to the well-known coaxial port method due to its simplified geometric representation [56]. The delta-gap technique involves imposing an impressed magnetic current sheet, which induces an incident electric field in the gap side surface. Figure 3 illustrates this scheme. Let us assume that a specific incident voltage, expressed as a time-dependent function $V^{inc}(t)$, needs to be established at the surface gap. Additionally, the relationship between the incident electric field and the incident voltage is given by:

$$\mathbf{E}^{inc} = -(V_{inc}(t)/\Delta)\widehat{I}_g \tag{2.31}$$

where Δ is the gap length and $\widehat{I}_g$ is the unit vector in the gap orientation.

In this formulation, the PEC boundary condition is imposed across the points of the surface gap, that is, the delta-gap simulates a commonly known hard source [58]. As mentioned before, the PEC boundary condition is applied by modifying the terms $\Delta\mathbf{E}$ and $\Delta\mathbf{H}$ in Eq. 2.17. In addition, the surface magnetic current can be introduced via numerical flux in a weak form considering the relationship:

$$\mathbf{M}_s = -\widehat{\mathbf{n}}_g \times \mathbf{E}^{inc} \tag{2.32}$$

where $\widehat{\mathbf{n}}_g$ is the normal vector to the gap. Now, substituting Eq. 2.31 in Eq. 2.32, we have:

$$\mathbf{M}_s = -\widehat{\mathbf{n}}_g \times \mathbf{E}^{inc} = -(V_{inc}(t)/\Delta)(\widehat{\mathbf{n}}_g \times \widehat{I}_g) \tag{2.33}$$

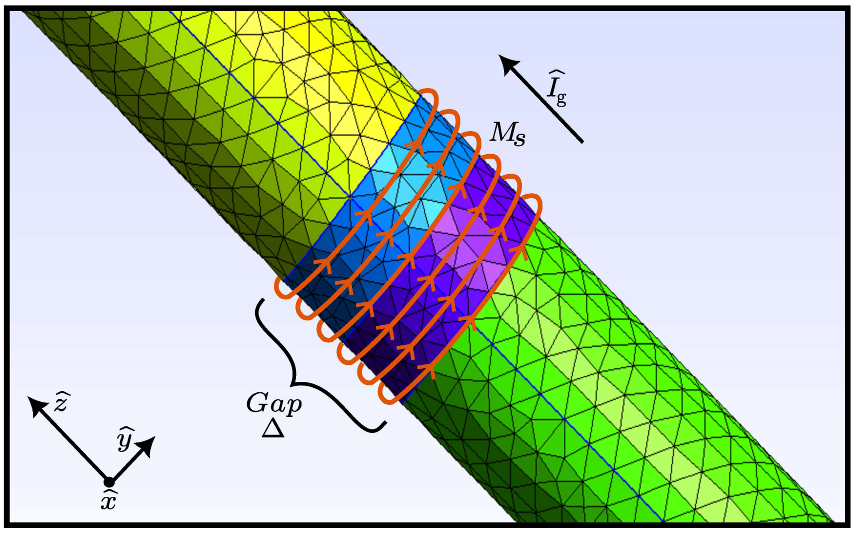


Figure 3 – Illustration of the magnetic current across the gap surface.

Finally, by applying the PEC boundary conditions and incorporating the magnetic current terms, the delta-gap formulation can be integrated into the DGTD method by modifying the flux terms of Eq. 2.17 at the gap nodes as follows:

$$\widehat{\mathbf{n}} \cdot \Big[\mathbf{F}(\mathbf{q}_h) - \mathbf{F}^*(\mathbf{q}^-, \mathbf{q}^+)\Big] = \frac{1}{2}\begin{cases} \bar{Z}^{-1}\Big(\widehat{\mathbf{n}} \times \Big[\Big(\widehat{\mathbf{n}} \times 2\mathbf{E}^-\Big) + \mathbf{M}_s\Big]\Big) \\ \bar{Y}^{-1}\Big(\Big[\widehat{\mathbf{n}} \times Y^+ 2\mathbf{E}^-\Big] + \mathbf{M}_s\Big) \end{cases} \tag{2.34}$$

This antenna feed model is a suitable choice due to its low implementation cost and ability to produce accurate results. However, it is important to note that simulations can be computationally intensive compared to other approaches, as the model does not directly absorb reflected waves. This can lead to increased simulation times, especially for complex antenna geometries.

2.3.4.3 The magnetic frill generator

The magnetic frill generator is a commonly used alternative in the FDTD method [59–61] to model coaxial waveports avoiding the complex discretization of cylindrical structures with a cube-based space partitioning. Although initially modeled as a hard source, imposing the total voltage or current and disregarding any reflected waves, the magnetic frill formulation can be improved by using a Transmission Line (TL) model that allow us to represent the total voltage at the coaxial aperture as the superposition of the incident, $V^{inc}(t)$, and reflected, $V^{refl}(t)$, voltages [62]. Let us assume a cylindrical coordinate system $(\widehat{\rho}, \widehat{\phi}, \widehat{z})$ with a coaxial aperture centered at the origin and oriented in the $\widehat{z}$ direction, as illustrated in Figure 4. According to [62], the incident electric field can be injected through an impressed magnetic current expressed as

$$\mathbf{M}_s = -(2V^{inc}(t) - Z_0 \cdot I^{tot}(t))/\rho \ln(l_b/l_a)\widehat{\phi} \tag{2.35}$$

where Z_0 is the characteristic impedance of the TL for the dominant Transverse Electromagnetic (TEM) mode. $I^{tot}(t)$ is the electric current flowing through the coaxial aperture and the antenna, with l_a and l_b being the radii of the inner and outer conductors.

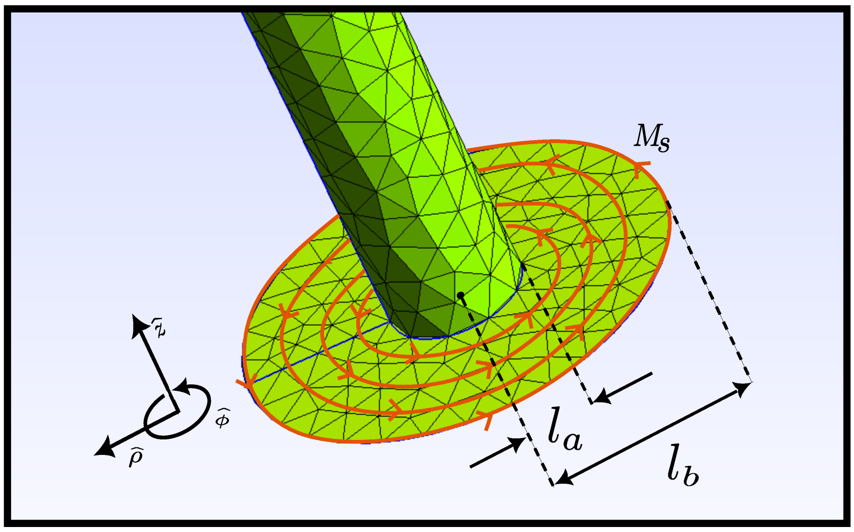


Figure 4 – Illustration of the impressed magnetic current across the coaxial aperture surface.

The first and second terms of Eq. 2.35 represent the incident voltage across the coaxial TL and the effects of time-domain reflections between the antenna and the coaxial TL, respectively. Accounting for reflected waves in this feed model reduces the simulation time compared to the delta-gap approach. In terms of implementation costs, the magnetic frill generator is introduced into the DGTD method by using the same flux expression of the delta-gap model described in Eq. 2.34 together with the PEC boundary condition on the coaxial aperture nodes. The only distinction lies in the calculation and inclusion of the electric current term, $I^{tot}(t)$, in the magnetic current expression at each time step. However, if the goal of the simulation is to calculate the input impedance/admittance of the antenna under test, both models require the calculation of the electrical current. Consequently, the implementation costs for both models are equivalent for most cases.

2.3.4.4 The coaxial waveport

Although the aforementioned antenna feeding approaches are numerically simple and attractive, their applicability is restricted in certain contexts [6]. In this sense, the coaxial waveport emerges as a more realistic and precise feed model, enabling the injection of the incident wave into the waveguide while simultaneously absorbing the reflected wave from the antenna, thereby preventing any spurious reflections. In this feed model, the dominant TEM mode is injected in a weak manner through flux terms by using the TF/SF formulation [63] while the effective PML absorbing boundary condition is applied to terminate the waveport [64]. Furthermore, the excitation is placed on the surface between the PML and physical regions. An illustration of this scheme can be seen in Figure 5.

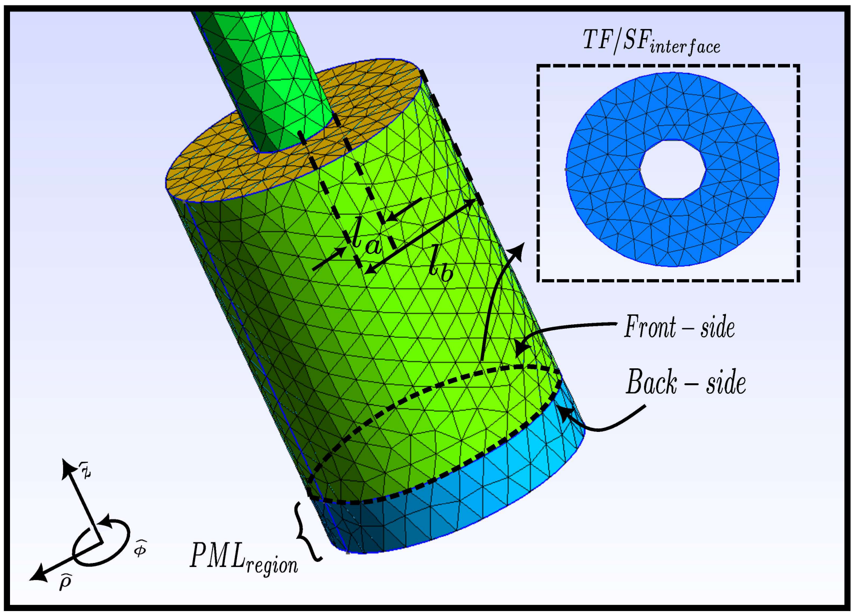


Figure 5 – Illustration of the coaxial waveport geometry.

The TF/SF formulation is used so that the incident TEM mode propagates only towards the physical region and no waves towards the PML region. This can be done by introducing the incident field terms E^{inc} and H^{inc} into the numerical flux expression of Eqs. 2.29 and 2.30 at the excitation surface nodes as

$$\mathbf{E}^{inc} = V^{inc}(t)\frac{1}{\rho \cdot \ln(l_b/l_a)}\widehat{\rho}, \quad \mathbf{H}^{inc} = V^{inc}(t)\frac{1}{\rho \cdot \eta \cdot \ln(l_b/l_a)}\widehat{\phi} \tag{2.36}$$

where η is the intrinsic impedance of the coaxial waveport with l_a and l_b being the inner and outer radii of the concentric conductors.

As can be seen in Figure 5, this antenna feed model increases the size of the computational domain due to the discretization of the coaxial cable, while increasing the memory and computation requirements. However, this is offset by the rapid absorption of the reflected waves by the PML, considerably reducing the number of time steps to obtain accurate results. In terms of implementation cost, the coaxial waveport presents a higher level of complexity compared with the delta-gap and magnetic frill models due to the need of the TF/SF formulation to inject the incident fields. Although the TF/SF method may be straightforward for experienced users of numerical methods such as FDTD [65] and FEM [6], novice users of the DGTD method may encounter some difficulties in its implementation and therefore opt for other simplified models.

2.4 Galerkin Semi-discretized Form

Suppose that the local solution can be represented in the following form:

$$\mathbf{q}_h^k = \sum_{i=1}^{N_p} q_i(\mathbf{r}_i, t)L_i(\mathbf{r}) = \sum_{n=1}^{N_p} \widehat{q}_n\psi_n(\mathbf{r}) \tag{2.37}$$

where L_i and ψ_n determine a nodal and modal local basis, respectively.

According to [4], the nodal and modal coefficients can be related as follows:

$$q_{nodal} = V\widehat{q}_{modal} \tag{2.38}$$

This matrix, V, is known as a generalized Vandermonde matrix, $V_{ij} = \psi_j(\mathbf{r}_i)$. Its function is to establish the connection between the modes $\widehat{q}$ and the nodal values q. Now, we choose the interpolating Lagrange polynomial as the function L_i to approximate the exact solution. It is well known that this polynomial basis has the Kronecker delta property:

$$L_i(\mathbf{r}_j) = \begin{cases} 1, & i = j \\ 0, & i \neq j \end{cases} \tag{2.39}$$

Lagrange polynomials are formed by the linear combination of monomials and can be represented in general by [57]:

$$L_i(\mathbf{r}_j) = \sum_{k,l,m=0}^{k+l+m\leq N} a^i_{k,l,m}x^k y^l z^m \tag{2.40}$$

where N represents the polynomial order, which determines the number of monomials in the polynomial basis and, consequently, the number of grid point nodes N_p within each element, considering:

$$N_p = N + 1 \qquad (1D) \tag{2.41a}$$

$$N_p = \frac{1}{2}(N+1)(N+2) \qquad (2D) \tag{2.41b}$$

$$N_p = \frac{1}{6}(N+1)(N+2)(N+3) \qquad (3D) \tag{2.41c}$$

To guarantee the numerical stable behavior of the generalized Vandermonde matrix V, an orthonormal polynomial basis for $\psi_j(\mathbf{r})$ must be used.

2.4.1 Three-dimensional canonical basis

considering a canonical basis in three dimensions defined on a space with the coordinates (r, s, t):

$$\begin{gathered} \psi_m(\mathbf{r}) = r^i s^j t^k, \quad (i,j,k) \geq 0; \quad i+j+k \leq N \\ m = 1 + \tfrac{(11+12N+3N^2)}{6} i, + \tfrac{(2N+3)}{2} j + k \\ -\tfrac{(2+N)}{2} i^2 - ij - \tfrac{j^2}{2} + \tfrac{i^3}{6}, \quad (i,j,k) \geq 0; \quad i+j+k \leq N \end{gathered} \tag{2.42}$$

The function ψ_m shown in Eq. 2.42 is the basis function considered on the reference tetrahedron I, See Figure 6, defined as [36]:

$$I = \{\mathbf{r} = (r,s,t) | (r,s,t) \geqslant -1; \quad r+s+t \leqslant -1\} \tag{2.43}$$

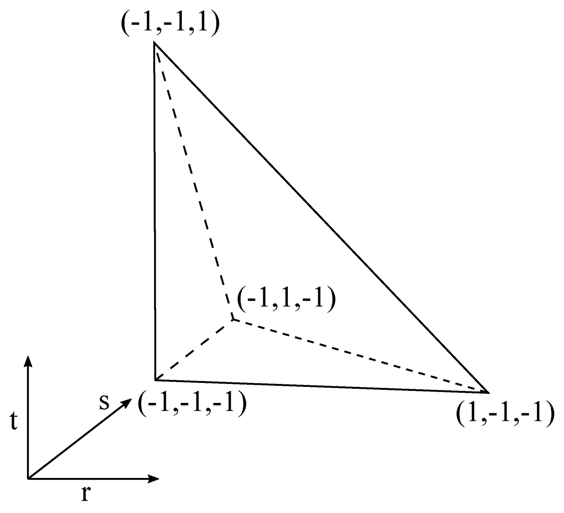


Figure 6 – Reference tetrahedron.

According to [4], the canonical basis shown in Eq. 2.42 can not be considered as a good choice because if the polynomial order N increases, the vandermonde matrix becomes poorly conditioned. To solve this problem, the polynomial basis can be orthonormalized applying the Gram-Schmidt process. The resulting basis is:

$$\psi_m(\mathbf{r}) = \sqrt{8} P_i(a) P_j^{(2i+1,0)}(b)(1-b)^i P_k^{(2i+2j+2,0)}(c)(1-c)^{i+j} \tag{2.44}$$

where

$$a = -2\frac{(1+r)}{s+t} - 1, \quad b = 2\frac{(1+s)}{1-t} - 1, \quad c = t$$

and $P_n^{\alpha,\beta}$ is the n-th order Jacobi polynomial. When $\alpha = \beta = 0$ it is the Legendre polynomial.

2.4.2 Two-dimensional canonical basis

Now, let us consider a canonical basis defined on a space with the coordinates (r, s):

$$\begin{gathered} \psi_m(\mathbf{r}) = r^i s^j, \quad (i,j) \geq 0; \quad i+j \leq N \\ m = j + (N+1)i + 1 - \tfrac{i}{2}(i-1), \quad (i,j) \geq 0; \quad i+j \leq N \end{gathered} \tag{2.45}$$

The function ψ_m shown in Eq. 2.45 is the basis function considered on the reference triangle I, See Figure 7, defined as [36]:

$$I = \{\mathbf{r} = (r,s) | (r,s) \geqslant -1; \quad r+s \leqslant 0\} \tag{2.46}$$

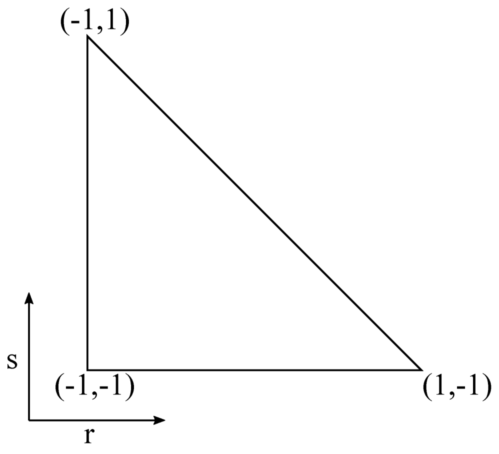


Figure 7 – Reference triangle.

Again, the canonical basis shown in Eq. 2.45 can not be considered as a good choice because if the polynomial order N increases, the vandermonde matrix becomes poorly conditioned. To solve this problem, the polynomial basis is orthonormalized applying the Gram-Schmidt process. The resulting basis is:

$$\psi_m(\mathbf{r}) = \sqrt{2}P_i(a)P_j^{(2i+1,0)}(b)(1-b)^i \tag{2.47}$$

where

$$a = 2\frac{1+r}{1-s} - 1, \qquad b = s,$$

and $P_n^{\alpha,\beta}$ is the n-th order Jacobi polynomial. When $\alpha = \beta = 0$ it is the Legendre polynomial.

2.4.3 Nodal distribution

The optimal distribution of the N_p grid point nodes is essential because a poorly chosen set generates computational problems as ill-conditioning matrices. As in FEM, the nodal distribution is done in a reference element and then, nodes are mapped into a physical element. There are some different ways to build a good nodal distribution [66–68]. However, all of them require substantial initial effort that can be avoided. A simple and constructive approach for a computation of a well-behaved family of nodal points of any order was found in [4]. This process maps a set of equidistant nodes into Legendre-Gauss-Lobatto distribution. Finally, the nodal distribution is calculated for each simulation, using a computational low cost algorithm and avoiding distribution tables. Figure 8 shows the N_p grid point nodes into a triangle and a tetrahedron considering a fourth-order polynomial.

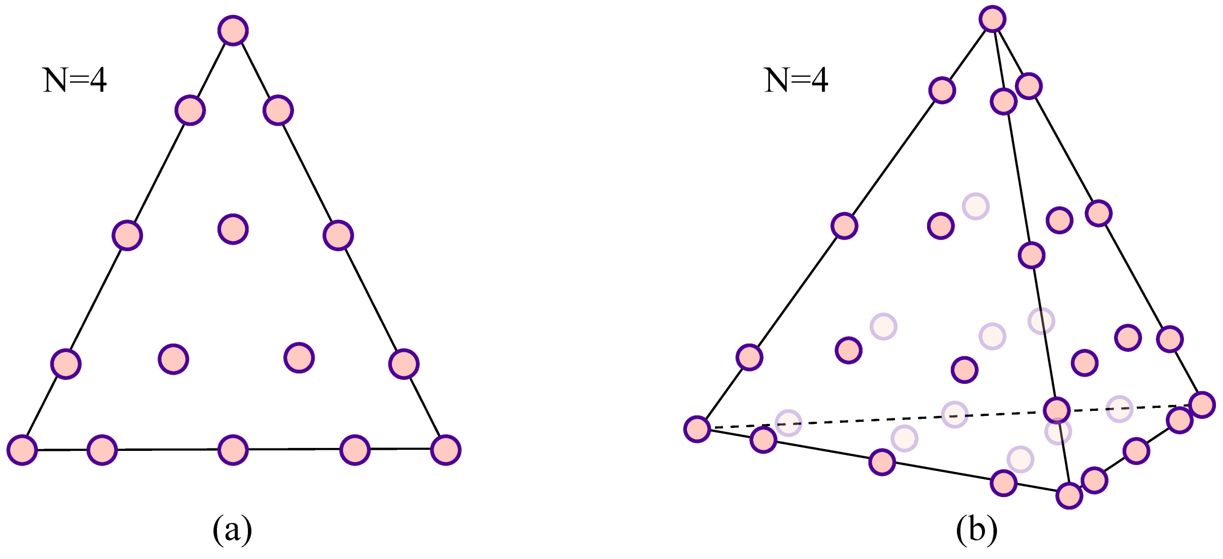


Figure 8 – Nodal distribution for $N = 4$ into a (a) triangle and (b) tetrahedron.

2.4.4 Mass and stiffness matrices

Considering only a local element Ω_k, the spatial discretization of the left-hand side of Eq. 2.16 can be represented by:

$$\int_{\Omega_k} \Big[Q\partial_t \mathbf{q}_h(\mathbf{r},t) + \nabla\cdot\mathbf{F}(\mathbf{q}_h)\Big] L_j(\mathbf{r},t)d\Omega = \int_{\Omega_k} \Big[Q^k\partial_t \mathbf{q}_h^k + \nabla\cdot\mathbf{F}(\mathbf{q}_h^k)\Big] L_j^k d\Omega \tag{2.48}$$

The approximation of each variable in the system is made using basis functions such that: $u_h^k = \sum_{i=1}^{N_p} u_t^{k,i} L_i^k(\mathbf{r})$, where N_p is the number of the unknown variables for each element. Coefficients depend on time and the basis functions of space. Therefore, using expansion by basis functions in Eq. 2.48, we obtain:

$$\begin{aligned}
\textstyle\int_{\Omega_k}\left(\mu_0^k\partial_t H_x^k + \partial_y E_z^k - \partial_z E_y^k\right) L_j^k d\Omega &= \mu_0^k \mathcal{M}^k \partial_t \mathbf{H}_x^k + [\mathcal{S}^{k,y}]^T \mathbf{E}_z^k - [\mathcal{S}^{k,z}]^T \mathbf{E}_y^k\\
\textstyle\int_{\Omega_k}\left(\mu_0^k\partial_t H_y^k + \partial_z E_x^k - \partial_x E_z^k\right) L_j^k d\Omega &= \mu_0^k \mathcal{M}^k \partial_t \mathbf{H}_y^k + [\mathcal{S}^{k,z}]^T \mathbf{E}_x^k - [\mathcal{S}^{k,x}]^T \mathbf{E}_z^k\\
\textstyle\int_{\Omega_k}\left(\mu_0^k\partial_t H_z^k + \partial_x E_y^k - \partial_y E_x^k\right) L_j^k d\Omega &= \mu_0^k \mathcal{M}^k \partial_t \mathbf{H}_z^k + [\mathcal{S}^{k,x}]^T \mathbf{E}_y^k - [\mathcal{S}^{k,y}]^T \mathbf{E}_x^k\\
\textstyle\int_{\Omega_k}\left(\varepsilon^k\partial_t E_x^k - \partial_y H_z^k + \partial_z H_y^k\right) L_j^k d\Omega &= \varepsilon^k \mathcal{M}^k \partial_t \mathbf{E}_x^k - [\mathcal{S}^{k,y}]^T \mathbf{H}_z^k + [\mathcal{S}^{k,z}]^T \mathbf{H}_y^k\\
\textstyle\int_{\Omega_k}\left(\varepsilon^k\partial_t E_y^k - \partial_z H_x^k + \partial_x H_z^k\right) L_j^k d\Omega &= \varepsilon^k \mathcal{M}^k \partial_t \mathbf{E}_y^k - [\mathcal{S}^{k,z}]^T \mathbf{H}_x^k + [\mathcal{S}^{k,x}]^T \mathbf{H}_z^k\\
\textstyle\int_{\Omega_k}\left(\varepsilon^k\partial_t E_z^k - \partial_x H_y^k + \partial_y H_x^k\right) L_j^k d\Omega &= \varepsilon^k \mathcal{M}^k \partial_t \mathbf{E}_z^k - [\mathcal{S}^{k,x}]^T \mathbf{H}_y^k + [\mathcal{S}^{k,y}]^T \mathbf{H}_x^k
\end{aligned} \tag{2.49}$$

where $\mathbf{H}_x^k, \mathbf{H}_y^k, \mathbf{H}_z^k, \mathbf{E}_x^k, \mathbf{E}_y^k, \mathbf{E}_z^k$ are vectors with dimension $N_p \times 1$, they contain the field component for each nodal value. Moreover, electric permittivity ε^k and magnetic permeability μ_0^k have the same dimension $N_p \times 1$ with the information about the media in element k. On the other hand, $\mathcal{M}^k$ and $\mathcal{S}^k$ are known as mass and stiffness matrices, respectively. Their dimensions are equal to $N_p \times N_p$.

$$\begin{aligned}
\mathcal{M}_{i,j}^k &= \textstyle\int_{\Omega_k} L_i^k L_j^k d\Omega\\
\mathcal{S}_{i,j}^{k,x} &= \textstyle\int_{\Omega_k} \partial_x L_i^k L_j^k d\Omega\\
\mathcal{S}_{i,j}^{k,y} &= \textstyle\int_{\Omega_k} \partial_y L_i^k L_j^k d\Omega\\
\mathcal{S}_{i,j}^{k,z} &= \textstyle\int_{\Omega_k} \partial_z L_i^k L_j^k d\Omega
\end{aligned} \tag{2.50}$$

Taking into account that V was built using an orthonormal basis, the mass matrix can be calculated as follows:

$$\mathcal{M}^k = J^k (VV^T)^{-1} \tag{2.51}$$

where J^k is the jacobian of the element k. In addition, since the DG method uses discontinuous basis functions, the resulting mass matrix is block diagonal. Now, the differentiation matrices $\mathcal{D}_r, \mathcal{D}_s$, and $\mathcal{D}_t$ are introduced to calculate the stiffness matrices [4].

These differentiation matrices are operators that transform point values, $u(\mathbf{r})$, to derivatives at the same points (e.g. $\mathbf{u}_h^{'} = \mathcal{D}_r \mathbf{u}_h$).

$$\tfrac{\partial}{\partial x} = \tfrac{\partial r}{\partial x}\mathcal{D}_r + \tfrac{\partial s}{\partial x}\mathcal{D}_s + \tfrac{\partial t}{\partial x}\mathcal{D}_t, \quad \tfrac{\partial}{\partial y} = \tfrac{\partial r}{\partial y}\mathcal{D}_r + \tfrac{\partial s}{\partial y}\mathcal{D}_s + \tfrac{\partial t}{\partial y}\mathcal{D}_t,$$
$$\tfrac{\partial}{\partial z} = \tfrac{\partial r}{\partial z}\mathcal{D}_r + \tfrac{\partial s}{\partial z}\mathcal{D}_s + \tfrac{\partial t}{\partial z}\mathcal{D}_t$$

where $\mathcal{D}_r$, $\mathcal{D}_s$, and $\mathcal{D}_t$ calculate derivatives in the 3D reference space (r, s, t). They can be calculated as follows:

$$V_{r,(i,j)} = \frac{\partial \psi_j(r_i, s_i, t_i)}{\partial r}$$
$$V_{s,(i,j)} = \frac{\partial \psi_j(r_i, s_i, t_i)}{\partial s}$$
$$V_{t,(i,j)} = \frac{\partial \psi_j(r_i, s_i, t_i)}{\partial t}$$

Considering the relations $\mathcal{D}_r = V_r V^{-1}$, $\mathcal{D}_s = V_s V^{-1}$, and $\mathcal{D}_t = V_t V^{-1}$ [4]. The expressions for the stiffness matrices are given by:

$$\begin{aligned} \mathcal{S}^{k,x} &= (\tfrac{\partial r}{\partial x}\mathcal{D}_r + \tfrac{\partial s}{\partial x}\mathcal{D}_s + \tfrac{\partial t}{\partial x}\mathcal{D}_t)(VV^T)^{-1} \\ \mathcal{S}^{k,y} &= (\tfrac{\partial r}{\partial y}\mathcal{D}_r + \tfrac{\partial s}{\partial y}\mathcal{D}_s + \tfrac{\partial t}{\partial y}\mathcal{D}_t)(VV^T)^{-1} \\ \mathcal{S}^{k,z} &= (\tfrac{\partial r}{\partial z}\mathcal{D}_r + \tfrac{\partial s}{\partial z}\mathcal{D}_s + \tfrac{\partial t}{\partial z}\mathcal{D}_t)(VV^T)^{-1} \end{aligned} \tag{2.52}$$

In addition, the mass and stiffness matrices are related by [4]:

$$\begin{aligned} \mathcal{S}_r &= \mathcal{M}\mathcal{D}_r \\ \mathcal{S}_s &= \mathcal{M}\mathcal{D}_s \\ \mathcal{S}_t &= \mathcal{M}\mathcal{D}_t \end{aligned} \tag{2.53}$$

Finally, the spatial discretization of the left-hand side of the strong variational formulation given by Eq. 2.16 can be obtained by substituting Eq. 2.53 into Eq. 2.49.

$$\begin{aligned} \mu_0^k \partial_t \mathbf{H}_x^k &= -\mathcal{D}_y \mathbf{E}_z^k + \mathcal{D}_z \mathbf{E}_y^k + \mathcal{M}^{-1} flux_{term}(\mathbf{H}_x^k) \\ \mu_0^k \partial_t \mathbf{H}_y^k &= -\mathcal{D}_z \mathbf{E}_x^k + \mathcal{D}_x \mathbf{E}_z^k + \mathcal{M}^{-1} flux_{term}(\mathbf{H}_y^k) \\ \mu_0^k \partial_t \mathbf{H}_z^k &= -\mathcal{D}_x \mathbf{E}_y^k + \mathcal{D}_y \mathbf{E}_x^k + \mathcal{M}^{-1} flux_{term}(\mathbf{H}_z^k) \\ \varepsilon^k \partial_t \mathbf{E}_x^k &= \mathcal{D}_y \mathbf{H}_z^k - \mathcal{D}_z \mathbf{H}_y^k + \mathcal{M}^{-1} flux_{term}(\mathbf{E}_x^k) \\ \varepsilon^k \partial_t \mathbf{E}_y^k &= \mathcal{D}_z \mathbf{H}_x^k - \mathcal{D}_x \mathbf{H}_z^k + \mathcal{M}^{-1} flux_{term}(\mathbf{E}_y^k) \\ \varepsilon^k \partial_t \mathbf{E}_z^k &= \mathcal{D}_x \mathbf{H}_y^k - \mathcal{D}_y \mathbf{H}_x^k + \mathcal{M}^{-1} flux_{term}(\mathbf{E}_z^k) \end{aligned} \tag{2.54}$$

The term $flux_{term}(\mathbf{q}^k)$ located on the right-hand side of Eq. 2.54 refers to the spatial discretization of the flux for each field component. These terms will be discussed in the next subsection. On the other hand, the first two terms of the right-hand side of Eq. 2.54 are known as volume terms. They are responsible for calculating the derivatives of the field components inside each element k. Considering that the differentiation matrices have dimension $N_p \times N_p$, the field derivatives are calculated by using a matrix-vector multiplication for each element. This matrix-vector multiplication is considered an easy-parallelizable operation.

Now, for the TMz mode the system of equations shown in 2.54 is reduced to:

$$\begin{aligned}
\mu_0^k \partial_t \mathbf{H}_x^k &= -\mathcal{D}_y \mathbf{E}_z^k + \mathcal{M}^{-1} flux_{term}(\mathbf{H}_x^k) \\
\mu_0^k \partial_t \mathbf{H}_y^k &= \mathcal{D}_x \mathbf{E}_z^k + \mathcal{M}^{-1} flux_{term}(\mathbf{H}_y^k) \\
\varepsilon^k \partial_t \mathbf{E}_z^k &= \mathcal{D}_x \mathbf{H}_y^k - \mathcal{D}_y \mathbf{H}_x^k + \mathcal{M}^{-1} flux_{term}(\mathbf{E}_z^k)
\end{aligned} \tag{2.55}$$

2.4.5 Flux discretization

Until now, the computation of the local matrices of the right-hand side of the strong variational formulation of Maxwell's curl equations has been discussed. However, to complete the semi-discretized form we have to explain how to discretize the flux term of Eq. 2.16. So, considering that the surface integral is split into four individual components, each of type:

$$\int_{\Gamma_{\Omega_k}} \widehat{\mathbf{n}} \cdot \Big[\mathbf{F}(\mathbf{q}_h) - \mathbf{F}^*(\mathbf{q}^-, \mathbf{q}^+)\Big] L_j d\Gamma = \sum_{i=1}^{N_{fp}} \widehat{\mathbf{n}} \cdot \Big[\mathbf{F}(\mathbf{q}_h^k) - \mathbf{F}^*(\mathbf{q}^-, \mathbf{q}^+)\Big] \int_{face} L_i^k L_j^k d\Gamma \tag{2.56}$$

where N_{fp} is the number of nodes in each face. In the 3D case $N_{fp} = (N+1)(N+2)/2$.

Assuming that all tetrahedrons have straight sides and the outwardly directed normal vector, $\widehat{\mathbf{n}}$, is constant on each face. The mass face matrices can be represented by:

$$\mathcal{M}_{i,j}^{k,face} = \int_{face} L_i^k L_j^k d\Gamma \tag{2.57}$$

Note that the mass face matrix can be calculated using the 2D vandermonde matrix, V^{2D}, which corresponds to a two-dimensional interpolation on the face element. Thus, the mass face matrix can be represented as:

$$\mathcal{M}_{i,j}^{k,face} = J^{face}(V^{2D}(V^{2D})^T)^{-1} \tag{2.58}$$

where J^{face} is the Jacobian for each face.

The information inside each one of the four mass face matrices is organized into a new matrix, $\mathscr{A}^k$, with dimension $N_p \times 4N_{fp}$. Therefore, the right-hand side of Eq. 2.16 can be rewriting as:

$$\int_{\Gamma_{\Omega_k}} \widehat{\mathbf{n}} \cdot \Big[\mathbf{F}(\mathbf{q}_h) - \mathbf{F}^*(\mathbf{q}^-, \mathbf{q}^+)\Big] L_j d\Gamma = \mathscr{A}^k \widehat{\mathbf{n}} \cdot \Big[\mathbf{F}(\mathbf{q}_h^k) - \mathbf{F}^*(\mathbf{q}^-, \mathbf{q}^+)\Big] \tag{2.59}$$

The right-hand side of Eq. 2.59 represents the terms $flux_{term}(\mathbf{q}^k)$ mentioned in the previous subsection. Finally, substituting Eq. 2.18 into Eq. 2.59 and replacing the result into Eq. 2.54, the complete semi-discretized form of the strong variational formulation represented by Eq. 2.16 is given by:

$$\begin{aligned}
\partial_t \mathbf{H}_x^k &= (-\mathscr{D}_y \mathbf{E}_z^k + \mathscr{D}_z \mathbf{E}_y^k + \mathit{LIFT}\ \mathbf{P}_{H_x}^k)/\mu_0^k \\
\partial_t \mathbf{H}_y^k &= (-\mathscr{D}_z \mathbf{E}_x^k + \mathscr{D}_x \mathbf{E}_z^k + \mathit{LIFT}\ \mathbf{P}_{H_y}^k)/\mu_0^k \\
\partial_t \mathbf{H}_z^k &= (-\mathscr{D}_x \mathbf{E}_y^k + \mathscr{D}_y \mathbf{E}_x^k + \mathit{LIFT}\ \mathbf{P}_{H_z}^k)/\mu_0^k \\
\partial_t \mathbf{E}_x^k &= (\mathscr{D}_y \mathbf{H}_z^k - \mathscr{D}_z \mathbf{H}_y^k + \mathit{LIFT}\ \mathbf{P}_{E_x}^k)/\varepsilon^k \\
\partial_t \mathbf{E}_y^k &= (\mathscr{D}_z \mathbf{H}_x^k - \mathscr{D}_x \mathbf{H}_z^k + \mathit{LIFT}\ \mathbf{P}_{E_y}^k)/\varepsilon^k \\
\partial_t \mathbf{E}_z^k &= (\mathscr{D}_x \mathbf{H}_y^k - \mathscr{D}_y \mathbf{H}_x^k + \mathit{LIFT}\ \mathbf{P}_{E_z}^k)/\varepsilon^k
\end{aligned} \tag{2.60}$$

where $\mathit{LIFT} = \mathcal{M}^{-1}\mathscr{A}^k$ is a matrix with dimension $N_p \times 4N_{fp}$ and $\mathbf{P}_{H_x}^k, \mathbf{P}_{H_y}^k, \mathbf{P}_{H_z}^k, \mathbf{P}_{E_x}^k, \mathbf{P}_{E_y}^k, \mathbf{P}_{E_z}^k$ are vectors with dimension $4N_{fp} \times 1$.

$$\begin{matrix} \mathbf{P}_{H_x}^k \\ \mathbf{P}_{H_y}^k \\ \mathbf{P}_{H_z}^k \\ \mathbf{P}_{E_x}^k \\ \mathbf{P}_{E_y}^k \\ \mathbf{P}_{E_z}^k \end{matrix} = \frac{1}{2} \left\{ \begin{matrix}
\bar{Y}^{-1}\Big(n_y Y^+ \Delta E_z - n_z Y^+ \Delta E_y - \alpha \Big[\Delta H_x - ndotH \cdot n_x\Big]\Big) \\
\bar{Y}^{-1}\Big(n_z Y^+ \Delta E_x - n_x Y^+ \Delta E_z - \alpha \Big[\Delta H_y - ndotH \cdot n_y\Big]\Big) \\
\bar{Y}^{-1}\Big(n_x Y^+ \Delta E_y - n_y Y^+ \Delta E_x - \alpha \Big[\Delta H_z - ndotH \cdot n_z\Big]\Big) \\
\bar{Z}^{-1}\Big(-n_y Z^+ \Delta H_z + n_z Z^+ \Delta H_y - \alpha \Big[\Delta E_x - ndotE \cdot n_x\Big]\Big) \\
\bar{Z}^{-1}\Big(-n_z Z^+ \Delta H_x + n_x Z^+ \Delta H_z - \alpha \Big[\Delta E_y - ndotE \cdot n_y\Big]\Big) \\
\bar{Z}^{-1}\Big(-n_x Z^+ \Delta H_y + n_y Z^+ \Delta H_x - \alpha \Big[\Delta E_z - ndotE \cdot n_z\Big]\Big)
\end{matrix} \right. \tag{2.61}$$

Note that the vectors $\mathbf{P}_{H_x}^k, \mathbf{P}_{H_y}^k, \mathbf{P}_{H_z}^k, \mathbf{P}_{E_x}^k, \mathbf{P}_{E_y}^k, \mathbf{P}_{E_z}^k$ can be calculated by using simple parallelizable element-wise arithmetic operations. Once these vectors have been calculated, they must be multiplied by matrix *LIFT*. This matrix-vector multiplication can also be parallelized. In addition, the set of equations representing the TMz mode in 2D can be derived from Eq. 2.60, eliminating the variations in the $\hat{z}$ plane. So, we get:

$$\begin{aligned}
\partial_t \mathbf{H}_x^k &= (-\mathscr{D}_y \mathbf{E}_z^k + \mathit{LIFT}\ \mathbf{P}_{H_x}^k)/\mu_0^k \\
\partial_t \mathbf{H}_y^k &= (\mathscr{D}_x \mathbf{E}_z^k + \mathit{LIFT}\ \mathbf{P}_{H_y}^k)/\mu_0^k \\
\partial_t \mathbf{E}_z^k &= (\mathscr{D}_x \mathbf{H}_y^k - \mathscr{D}_y \mathbf{H}_x^k + \mathit{LIFT}\ \mathbf{P}_{E_z}^k)/\varepsilon^k
\end{aligned} \tag{2.62}$$

where

$$\begin{matrix} \mathbf{P}^k_{H_x} \\ \mathbf{P}^k_{H_y} \\ \mathbf{P}^k_{E_z} \end{matrix} = \frac{1}{2}\begin{cases} \bar{Y}^{-1}\left(n_y Y^+ \Delta E_z - \alpha\left[\Delta H_x - ndotH \cdot n_x\right]\right) \\ \bar{Y}^{-1}\left(-n_x Y^+ \Delta E_z - \alpha\left[\Delta H_y - ndotH \cdot n_y\right]\right) \\ \bar{Z}^{-1}\left(-n_x Z^+ \Delta H_y + n_y Z^+ \Delta H_x - \alpha\left[\Delta E_z\right]\right) \end{cases} \tag{2.63}$$

It is important to remark that the *LIFT* matrix in the 2D case, shown in Eq. 2.62, must be calculated in a different way than in the 3D case. Thus, considering that the computational domain for 2D problems is represented by straight-side triangular elements, the right-hand side of Eq. 2.16 is split into three individual components, each of type:

$$\int_{\Gamma\Omega_k} \widehat{\mathbf{n}} \cdot \left[\mathbf{F}(\mathbf{q}_h) - \mathbf{F}^*(\mathbf{q}^-, \mathbf{q}^+)\right] L^k_j d\Gamma = \sum_{i=1}^{N_{fp}} \widehat{\mathbf{n}} \cdot \left[\mathbf{F}(\mathbf{q}^k_h) - \mathbf{F}^*(\mathbf{q}^-, \mathbf{q}^+)\right] \int_{edge} L^k_i L^k_j d\Gamma \tag{2.64}$$

where N_{fp} is the number of nodes in each edge. In the 2D case $N_{fp} = N + 1$.

The mass edge matrices can be represented by:

$$\mathcal{M}^{k,edge}_{i,j} = \int_{edge} L^k_i L^k_j d\Gamma \tag{2.65}$$

Note that the mass edge matrix can be calculated using the 1D vandermonde matrix, V^{1D}, which corresponds to a one-dimensional interpolation on the edge element. Thus, the mass edge matrix can be represented as:

$$\mathcal{M}^{k,edge}_{i,j} = J^{edge}(V^{1D}(V^{1D})^T)^{-1} \tag{2.66}$$

where J^{edge} is the Jacobian for each edge.

Once the mass edge matrices are calculated, the contribution of each one is organized into a new matrix, $\mathcal{A}^k$, with dimension $N_p \times 3N_{fp}$. Finally, the *LIFT* matrix for the 2D case is calculated by using $LIFT = \mathcal{M}^{-1}\mathcal{A}^k$. This *LIFT* matrix has a $N_p \times 3N_{fp}$ dimension.

2.5 Time Integration Methods

As can be noticed, the emphasis so far has been on spatial discretization and semi-discrete representation. However, it is fundamental to consider the numerical integration in the time domain to complete the full DGTD scheme. There are many approaches that can be used in time discretization [32, 69, 70]. These works present different kinds of time integration methods and their principal features. Although there are so many approaches, basically we can divide them into two large groups: Explicit and Implicit methods.

Explicit methods compute the system state at a future time step based on the currently known state. They are widely used due to their ease of implementation and ability to produce accurate results. However, their stability is limited, as they require a sufficiently small time step to prevent divergence. Considering a differential equation $y = F(y,t)$, an explicit method could express the system in a future time as $y_{n+1} = y_n + \Delta t F(y_n, t_n)$ where Δt is known as time step and its calculation will be discussed later.

On the other hand, implicit methods calculate the state of the system at a future time using the currently known state of the system and the same future state. This means that it is necessary to solve a system of equations to calculate the future state at each time interval. At first glance, implicit methods may seem more complex to program and take a long time to calculate. However, a sufficiently large time step, high stability, and high convergence are interesting features that can be used. For example, implicit methods are very used to solve non-linear problems where it is difficult to predict a future from the past state. Considering again a differential equation $y = F(y,t)$, the future state into an implicit method are defined as $y_{n+1} = y_n + \Delta t F(y_{n+1}, t_{n+1})$.

As mentioned before, the use of orthonormal polynomials as basis functions produces block-diagonal mass matrices. This feature allows a couple between DG and fully explicit integration methods. Considering that explicit methods have a low computational cost and easier implementation when compared with implicit methods, explicit approaches such as leapfrog (LF) and Runge-Kutta methods are widely used to discretize the time domain into DG method [28,36,56,71,72]. In this work we focus on the use of explicit high order RK methods for integration in the temporal dimension.

2.5.1 Runge Kutta methods

Numerical methods to solve ordinary differential equations can be divided into two categories: single-step and multi-step. Into the single-step methods can be found the RK which only needs the current value to calculate the next time solution. On the other hand, multi-step methods use many previous values to advance to the next time solution. In this section we explain the RK methods and why they are considered a single-step method. Firstly, consider the next differential equation:

$$\frac{d}{dt}\mathbf{y} = \mathbf{y}' = \mathbf{f}(\mathbf{y}, t), \quad t > t_0 \tag{2.67}$$
$$\mathbf{y}(t_0) = \mathbf{y}_0$$

where $\mathbf{y} = (y_1, y_2, ..., y_n)^T$ is the vector of unknowns and $\mathbf{f} : \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}^n$

The general s-stage RK method [73] for the ordinary differential equation system of Eq. 2.67 can be defined as:

$$\mathbf{y}_{n+1} = \mathbf{y}_n + \Delta t \sum_{i=1}^{s} b_i \mathbf{f}(\phi_i, t_n + c_i \Delta t) \tag{2.68}$$

and

$$\phi_i = \mathbf{y}_n + \Delta t \sum_{j=1}^{s} a_{ij} \mathbf{f}(\phi_j, t_n + c_j \Delta t), \quad 1 \leq i \leq s \tag{2.69}$$

where s is the number of stages and its number depends on the order approximation of the method, ϕ_i are the intermediate stages, and Δt is the time step defined on the interval $t_n \rightarrow t_{n+1}$. RK methods also can be represented by the known Butcher notation [74].

$$\begin{array}{c|cccc}
c_1 & a_{11} & a_{12} & \dots & a_{1s} \\
c_2 & a_{21} & a_{22} & \dots & a_{2s} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
c_s & a_{s1} & a_{s2} & \dots & a_{ss} \\
\hline
 & b_1 & b_2 & \dots & b_s
\end{array}$$

Table 1 – Butcher tableau for RK methods.

Coefficients a_{ij}, b_i and c_i are calculated by using a Taylor series expansion. However, for each order of the RK method (eg, second, third, fourth) the number of unknown coefficients is larger compared to the number of equations. This means that some unknowns must be specified a priori in order to determine the remaining parameters. Consequently, there are an infinite number of n-order RK methods [73]. Despite a large number of options for the RK methods, all of them are represented by the combination of the intermediate stages and the current time solution. As can be seen in Eq. 2.68, the next time step solution $\mathbf{y}_{n+1}$ is calculated based on the sum of the intermediate stages multiplied by constant values a_{ij}, b_i and c_i. It implies that every RK method can be implemented by using basic element-wise arithmetic operations such as sums and multiplications. This feature turns the RK methods very interesting candidates for parallel implementations.

2.5.2 Third-order Runge Kutta method (RK3)

The application of the Taylor series expansion in order to obtain a system of equations for the RK3 method results in a set of six equations with eight unknowns [73]. So, as mentioned before, there are several possible versions that would yield exactly the same results if the solution to the Ordinary Differential Equation (ODE) were cubic, quadratic, linear, or constant. Table 2 shows a Butcher notation of RK3 methods in function of a β parameter. It is important to remark that the constant value β must be chosen considering:

$$c_i = \sum_{j=1}^{s} a_{ij}, \quad i = 1, ..., s \tag{2.70}$$

$$\begin{array}{c|ccc}
0 & 0 & 0 & 0 \\
\frac{2}{3} & \frac{2}{3} & 0 & 0 \\
\frac{2}{3} & \frac{2}{3} - \frac{1}{4\beta} & \frac{1}{4\beta} & 0 \\
\hline
 & \frac{1}{4} & \frac{3}{4} - \beta & \beta
\end{array}$$

Table 2 – Butcher tableau for a third order RK family methods.

In order to satisfy Eq. 2.70, we choose $\beta = 3/8$. Modifications are shown in Table 3. Thus, the system of equation for the three-stage explicit RK3 method is given by:

$$\mathbf{y}_{n+1} = \mathbf{y}_n + \Delta t \frac{1}{4}(\mathbf{f}(\mathbf{y}_n, t_n) + \frac{3}{2}\mathbf{f}(\phi_2, t_n + \frac{2}{3}\Delta t) + \frac{3}{2}\mathbf{f}(\phi_3, t_n + \frac{2}{3}\Delta t)) \tag{2.71}$$

$$\phi_2 = \mathbf{y}_n + \frac{2}{3}\mathbf{f}(\mathbf{y}_n, t_n)\Delta t$$
$$\phi_3 = \mathbf{y}_n + \frac{2}{3}\mathbf{f}(\phi_2, t_n)\Delta t$$

$$\begin{array}{c|ccc}
0 & 0 & 0 & 0 \\
\frac{2}{3} & \frac{2}{3} & 0 & 0 \\
\frac{2}{3} & 0 & \frac{2}{3} & 0 \\
\hline
 & \frac{1}{4} & \frac{3}{8} & \frac{3}{8}
\end{array}$$

Table 3 – Butcher tableau for a third order RK family methods.

Note that in Eq. 2.71 the term $\mathbf{f}(\mathbf{y}_n, t_n)$ appears in the equation for ϕ_2, which appears in the equation for ϕ_3. This recurrence behavior makes RK methods efficient for computer calculations [73].

2.5.3 Fourth-order Runge Kutta method (RK4)

The fourth-order approximation is known as the most popular RK method. As in the third-order approach, there are several versions due to the number of equations is smaller than the number of unknowns. However, the most commonly used form and also called the classical fourth-order RK4 method is given by:

$$\mathbf{y}_{n+1} = \mathbf{y}_n + \Delta t\frac{1}{6}(\mathbf{f}(\mathbf{y}_n, t_n) + 2\mathbf{f}(\phi_2, t_n + \frac{1}{2}\Delta t) + 2\mathbf{f}(\phi_3, t_n + \frac{1}{2}\Delta t) + \mathbf{f}(\phi_4, t_n + \Delta t)) \quad (2.72)$$

$$\phi_2 = \mathbf{y}_n + \frac{2}{3}\mathbf{f}(\mathbf{y}_n, t_n)\Delta t$$
$$\phi_3 = \mathbf{y}_n + \frac{2}{3}\mathbf{f}(\phi_2, t_n)\Delta t$$
$$\phi_4 = \mathbf{y}_n + \frac{2}{3}\mathbf{f}(\phi_3, t_n)\Delta t$$

RK3 and RK4 methods are widely used for time integration into the DGTD method. These methods present good results due to their high order of approximation. However, it can be noticed that depending on the approximation order the number of intermediate arrays grows up. Considering that these arrays must be stored during the calculation of the solution at each time step, there would be a limitation in terms of memory, especially when dealing with large-scale problems. Therefore, the low-storage version was proposed to avoid this memory drawback.

2.5.4 Low-Storage Explicit Runge Kutta fourth order (LSERK4) method

The LSERK4 method is presented as an alternative to the common RK methods. [72, 75, 76]. This method is one of the most used in high-order DG schemes because it does not need to compute and evaluate derivatives, and also, produces low dispersion and low dissipation errors [47]. As in all time-stepping methods, the aim is to find the solution in the time step t_{n+1} taking as a reference the solution at the first time step t_n, starting with the given initial condition. Then, considering the Eq. 2.67 and applying the LSERK4 method we have:

$$\mathbf{p}_0 = \mathbf{y}_n$$
$$i \in [1, ...5] : \begin{cases} \phi_i = a_i\phi_{(i-1)} + \Delta t\mathbf{f}(\mathbf{p}_{(i-1)}, t_n + c_i\Delta t) \\ \mathbf{p}_i = \mathbf{p}_{(i-1)} + b_i\phi_i \end{cases} \quad (2.73)$$
$$\mathbf{y}_{n+1} = \mathbf{y}_5$$

where $\mathbf{y}_n$ and $\mathbf{y}_{n+1}$ are initial and final solution, respectively. They are separated by the time step Δt. Coefficients a_i, b_i and c_i can be found in [4].

As can be seen in Eq. 2.73, this method requires only two arrays ϕ_i and $\mathbf{p}_i$ that are modified themselves for each one of its five stages. Contrary to the common RK methods, the low-storage approach reduces memory usage significantly. However, the additional stage can make this method less interesting due to the added computational cost.

2.5.5 Conditional stability and time step

All of the previously discussed time integration methods are considered explicit and therefore subject to conditional stability, also known as the Courant-Friedrichs-Lewy (CFL) condition. When the time step Δt exceeds a critical value, the solution is subject to unphysical exponential growth. To avoid this issue, traditional approaches relate the size of the time step directly with the quality of the mesh [57]. One interesting approach was found in [4], where the time step is calculated with the minimum distance between the nodes into the smallest element in mesh,

$$\Delta t \leq C \Delta d_{min} \text{min} \left\{ r_{in}^{k} / c^{k} \right\} \tag{2.74}$$

where r_{in}^{k} is the radius of the incircle or insphere of the k element. Δd_{min} is the smallest distance between two nodes on the edges of the reference element. This length depends on the polynomial order N as $\Delta d_{min} \propto N^{-2}$. c_k is the maximum speed of light in k, and C is a constant factor of order 1.

This time step value guarantees and maintains the stability of the scheme. However, considering only the smallest mesh element implies an important restriction that compromises the efficiency of the method. Especially when dealing with multi-scale problems where the size of the smallest and largest elements presents a very large discrepancy. Many approaches based on multi-rate methods have been proposed in the literature to avoid this time-step restriction [27, 32, 33, 77]. These approaches utilize local time stepping (LTS), a technique that divides the mesh elements into classes or groups, with each class assigned a distinct time step determined by the smallest element within it. This method allows for more efficient time integration by adapting the time step locally rather than applying a uniform value across the entire domain. A more detailed explanation of local time stepping will be provided in Chapter 4.

2.6 Chapter Conclusions

This chapter presents the Discontinuous Galerkin Time-Domain (DGTD) method for solving Maxwell's equations in non-dispersive dielectric media. It begins with a review of Maxwell's equations in their vector and scalar forms, covering both three-dimensional and two-dimensional (TMz and TE) configurations. The conservation form and local variational formulation of the equations are introduced, with numerical fluxes used to handle discontinuities across element interfaces. Various boundary conditions are discussed, including PEC, PMC, and absorbing layers such as PML. The implementation of electromagnetic sources is also addressed, using techniques like TF/SF, delta-gap, magnetic frill, and coaxial waveport models. Spatial discretization is described in detail, employing orthonormal polynomial bases with attention to nodal distribution and the derivation of mass and stiffness matrices. Time integration is performed using explicit Runge-Kutta schemes, including low-storage variants, with a discussion on stability constraints. Overall, the chapter provides a comprehensive formulation and implementation strategy for the DGTD method.

3 Parallel Computing with GPUs

This chapter contains the basic concepts of GPU and its main features. First, the use of NVIDIA GPUs as parallel computing hardware along with its programming model will be discussed. Then, the differences between the GPU and the CPU will be exposed to understand the characteristics of heterogeneous computing. Finally, the hierarchical structures present in the GPU will be explained and discussed.

3.1 GPU: A Brief Introduction

In the early days of GPUs, they were used exclusively as graphics acceleration hardware. However, this has changed with the rise of interest in parallel computing. The main objective of parallel computing is nothing more than to improve the speed of our applications using all the resources provided by the GPU. In this sense, parallel computing can be defined as a way of doing several calculations simultaneously, considering the principle that a large and complex problem can be divided into several smaller and easier problems. Now, GPUs can be seen as a general-purpose processors for floating-point operations. In other words, a piece of hardware designed to do many arithmetic operations taking advantage of its multiple cores. GPUs present an improvement in performance when compared to CPUs due to the low latency and high bandwidth. Latency can be defined as the time it takes for an operation to start and complete, and is frequently expressed in microseconds. Bandwidth is defined as the amount of data that can be processed per unit of time and is often expressed in gigabytes/sec [9].

Although GPUs offer numerous advantages over conventional processor architectures, several factors must be carefully considered. One key consideration is the sensitivity of GPU code to hardware changes. This sensitivity becomes evident when components such as clock rates, bus widths, and memory sizes are modified. Additionally, GPUs provide multiple implementation strategies, some of which are more efficient than others. As a result, it is not uncommon to observe variations in execution times between codes that theoretically perform the same task [16]. In this work, we focus on the hardware of NVIDIA GPUs, making it essential to introduce key concepts related to the Compute Unified Device Architecture (CUDA). CUDA is a parallel computing platform and programming model developed by NVIDIA to solve complex computational problems efficiently. It enables parallel computing through standard programming languages such as C, C++, Fortran, and Python [9].

3.2 CUDA: A Parallel Computing Platform

Designed in 2006 by Nvidia, CUDA has arrived to help developers increase the efficiency of their applications by taking advantage of the acceleration power of GPUs. This is achieved by using a series of multiprocessors present on one or several graphics cards and maintaining a low learning curve for programmers familiar with standard programming languages. The CUDA programming model can be seen as a Single Instruction Multiple Data (SIMD) computer architecture because it refers to a type of parallel architecture where there are multiple cores in the computer and all cores are executing the same instruction at any given time. This parallel architecture has a very interesting advantage because programmers can write code and continue to think sequentially while it is achieved a high speed-up from parallel data operations. Considering that in the CUDA model, each piece of data is mapped separately to a thread, NVIDIA calls this architecture Single Instruction Multiple Thread (SIMT) [16].

Currently, the GPU cannot be thought of as a standalone hardware that can perform parallel operations by itself. On the contrary, the GPU must be considered as a co-processor that operates together with a CPU connected through a PCI-Express bus. The CPU, also called the host, is responsible for managing the code, environment and transferring data to the GPU. The GPU, also called the device, is used to perform intensive computing operations. This is a clear application of a heterogeneous platform where two different processors are used to complete a task [9]. It is important to note that the purpose of including the GPU in the programming model is not to replace the CPU because both perform different tasks. CPU computing is very good for serial-intensive tasks and GPU computing is good for parallel data-intensive tasks, i.e. one is the complement of the other and both form a powerful combination. It can be seen in Figure 9, where the sequential part of the code is handled by the CPU and the intensive parallel part is executed on the GPU.

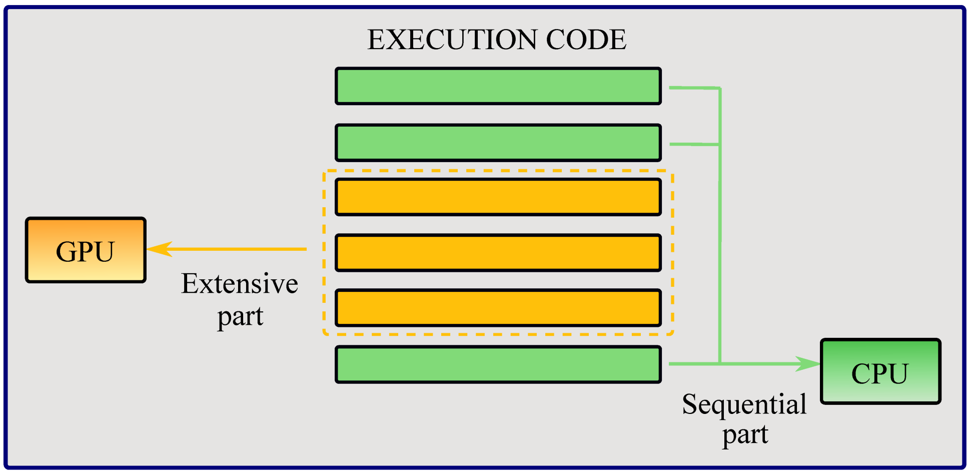


Figure 9 – CPU + GPU heterogeneous computing.

A CUDA program is divided into two parts: the host code that runs on CPU and the device code that runs on GPU. In this work, the host code is written using standard Python language and, the device code is written using CUDA C. The functions defined by the programmer in CUDA C are called kernels. These kernels are executed in parallel n times by n different CUDA threads simultaneously. Once the kernel is launched, the problem data is mapped to the threads, which are responsible for executing a particular function depending on the kernel instructions. These threads are grouped into blocks that at the same time form a grid. Therefore, it is the responsibility of the programmer to guarantee a good distribution or organization of threads depending on the problem. For that, it is necessary to know the hierarchy structure of threads and memory [9].

In principle is common to think that all threads are executed at the same time when the kernel is invoked. However, we should consider the hardware perspective to achieve the best performance. The GPU architecture is composed of an array of Streaming Multiprocessors (SM) which are the components designed to support the concurrent execution of hundreds of threads. Once a kernel is launched, the blocks in the program are distributed among all the available SMs for execution. Then, each SM divides the blocks into groups of 32 threads called warps. All threads in a warp execute the kernel instructions at the same time. However, even if all threads in a warp start the execution at the same time, it is possible that some threads have different behavior due to its independent execution path. This is the principal difference between SIMD and SIMT architectures. Because SIMD requires that all elements execute together in a unified synchronous form.

3.3 Threads: Hierarchy Structure

A thread can be defined as the smallest sequence of programmed instructions that can be managed independently. In a CUDA program, each thread has its own data, instructions address and register state. Every thread can be indexed using a three-dimensional vector, i.e, threads can be accessed using a 1D, 2D or 3D index. This index is used for calculating memory address locations and also for taking control decisions. In the hierarchical structure, threads are the first component, followed by blocks and finally grids. A block is defined as a set of threads where the maximum number per block is 1024. These blocks also can be organized using a three-dimensional vector which forms a grid. The number of blocks that composes a grid is defined by the data quantity of the program or by the number of SMs of the GPU. Figure 10 shows the two level thread hierarchy divided into block of threads and grid of blocks.

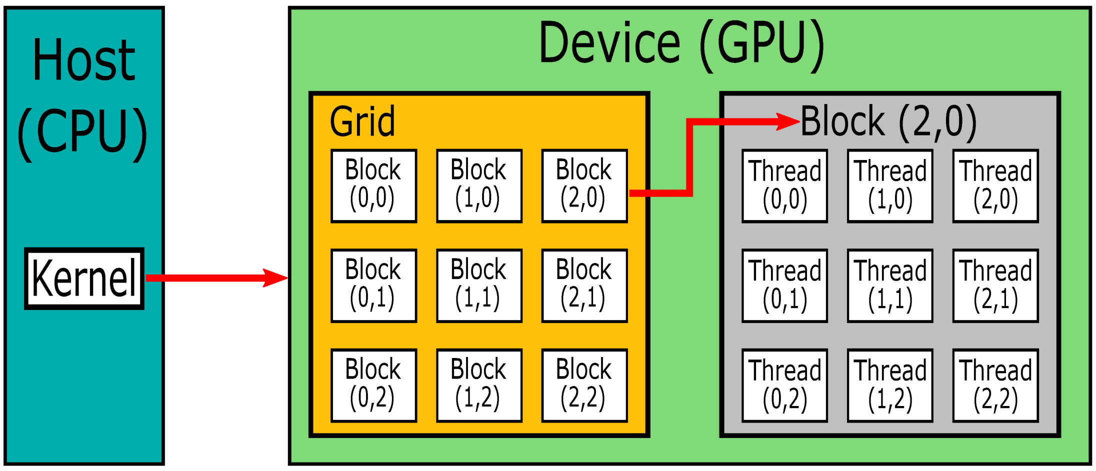


Figure 10 – Threads - hierarchy structure.

3.4 Memory Hierarchy

In the CUDA programming model, the program is composed of the host and the device, each with its own different memory. In this section, we will discuss about the device memory and its features. As mentioned before, in heterogeneous computation the data are initialized by the host and then are copied to the device. The data are processed inside the device and finally, it is sent back to the host in order to show the results or post-processing the data. This process can be carried out due to the possibility of allocating device memory from the host.

There are many different types of memory in the GPU, each one with different purposes, but almost all of them are available to the programmer. Within the main types, the constant and texture read-only memories are found. They are visible to all kernels but depending on the applications these types of memory can be slower than the global memory [9]. Another type of memory is global memory which is the most commonly used and largest memory on a GPU. Its global name refers to the lifetime due to its duration being the same as the application lifetime. This type of memory is very useful because it is accessible to all threads of all kernels, i.e., the global memory is analogous to the system memory of the CPU. Despite all these advantages, its main problem is the high latency.

Shared memory can be used to solve the high latency problem. This type of memory has much lower latency and much higher bandwidth when compared to global memory. GPU shared memory can be thought of as CPU cache memory, but programmable. This has a lifetime equal to the block, that is, when a block finished its execution, shared memory is allocated to another block. Because its lifetime depends on the time execution of each block, shared memory can be accessed only by the threads allocated in each block and not for all the threads. The shared memory size on NVIDIA GPUs is equal to 64KB, which is a disadvantage when compared to the global memory. Finally, each thread has its own local memory used to save the data during the execution of the program. This local memory can be accessed only by the threads. Figure 11 summarize the memory hierarchy of the GPU showing the three most important types of memories.

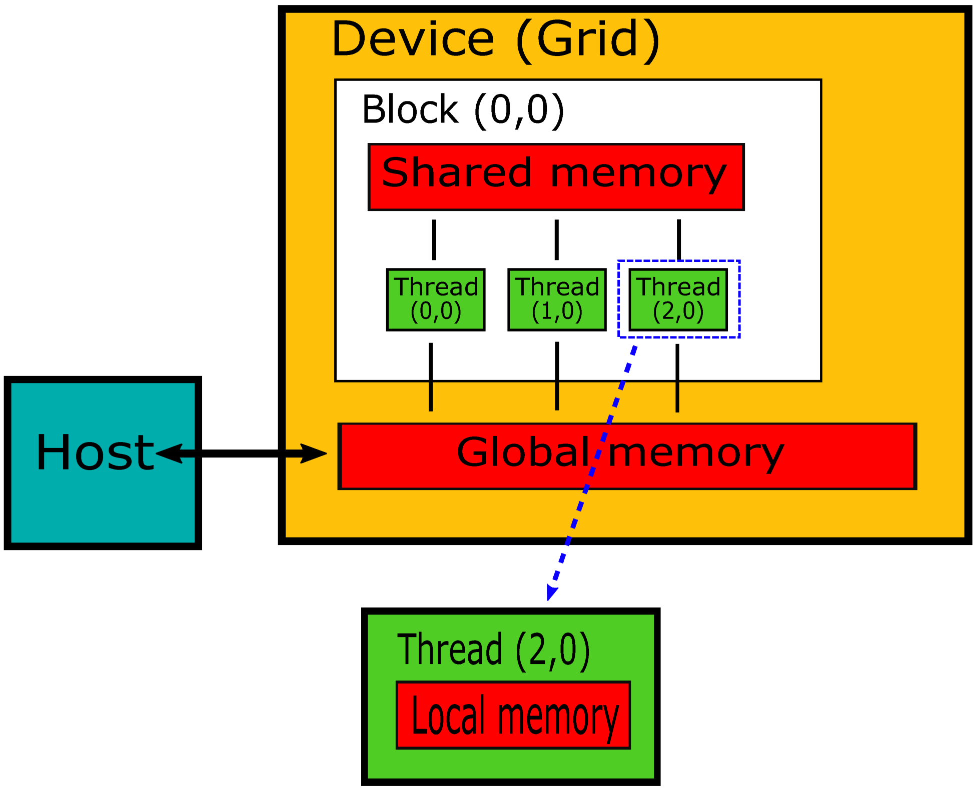


Figure 11 – Memory GPU structure.

3.5 DGTD on Graphics Processors

In order to optimize a CUDA program, it is critical to achieve an optimal balance in the distributions of the threads in each block. The NVIDIA GPUs have the restrictions of 1024 threads per block and 65536 blocks per grid dimension. Thus, it is necessary to create a good distribution considering these restrictions. As can be seen in [17], a large number of threads in a block reduces the memory latency but this feature also reduces the number of available registers. Therefore, NVIDIA recommends using blocks of 128 or 256 threads to obtain better latency values and increase the number of registers [15].

As was previously mentioned, the DGTD method can be considered a combination approach between the FEM method whose accuracy depends on the order of basis function, and the FVTD method whose neighboring cells are connected by the numerical flux. Dependence on numerical flux to ensure solution continuity makes DGTD a local method. In addition, the use of orthonormal basis function provides a fully explicit and inherently parallel method. The parallel-GPU with global time stepping (GTS) DGTD method (GPU-DGTD-GTS) can be divided naturally into three principal kernels [16]. First, the element surface integral kernel where the flux components of the unknown fields are calculated (i.e. terms $\mathbf{P_q}$ from Eq. 2.63). Second, the element volume integral kernel where the fields inside every element are calculated (i.e. the right-hand side terms $rhs_{\mathbf{q}}$ from Eq 2.62). This kernel also can be found in the literature as element local differentiation [17]. Third, the time integration kernel where the field components are updated in time by using a high order method, usually leapfrog (LF) or Runge-Kutta (RK) approaches (i.e. use the $rhs_{\mathbf{q}}$ terms calculated before and apply Eq. 2.68). These three CUDA kernels require an optimal thread distribution in order to guarantee a good GPU performance. The thread distribution methodology for each kernel will be discussed in the next sections with more details. The procedure of the GPU-DGTD-GTS method is summarized in the Algorithm 1. Note that the value of s depends directly on the order of approximation of the time integration RK method. The input arrays needed in the Algorithm 1 are summarized in Table 4. These input arrays were calculated on the host and then stored on the device for use in the kernels. In addition, the geometric factors mentioned in Table 4 are terms that depend on the shape of each element and they are used in the transformation from the reference to the local element.

Table 4 – List of inputs arrays for algorithm 1.

Array	Dimension	Description
$\mathbf{q}$	$N_{fc} \times N_p \times K$	Fields components
G_V	$K \times Dim^2$	Geometric factors for the volume kernel
G_S	$K \times N_{faces} \times (Dim + 1)$	Geometric factors for the surface kernel
$LIFT$	$N_p \times N_{faces} \times N_{fp}$	Matrix for the surface integration
$\mathcal{D}_m$	$N_p \times N_p \times Dim$	Differentiation matrices
$\mathbf{Flux}_{ind}$	$K \times N_{fp} \times 2$	Global index for local and neighbor element

The terms N_{fc} and Dim shown in Table 4 represent the number of field components and the spatial dimension, respectively. The other terms will be discussed in the next section. One of the most critical problems of the GPU-DGTD-GTS method in terms of consuming time is the data exchange between host and device. However, this operation is essential and cannot be omitted, as the field components must be updated at each time step to ensure accurate computation and maintain the data dependencies necessary for subsequent processing. Additionally, this data transfer is also important for post-processing tasks, which are typically handled on the CPU side. Therefore, it is recommended to minimize it as much as possible. This problem is not only present in the DGTD but also in other time-dependent methods such as FDTD and FETD [13]. Finally, most works in the literature recommend using shared memory as much as possible for computing calculations due to its lower latency and higher bandwidth [8, 17, 19].

Algorithm I: GPU-DGTD-GTS method

```
1  procedure $PAR\_MAXWELL(\mathbf{q}, G_V, G_S, LIFT, \mathcal{D}_m, \mathbf{Flux}_{ind})$
2      Initialize all variables                    // Create update matrices $\mathbf{P_q}$, $rhs_\mathbf{q}$
3      Calculate the time step value $\Delta t$
4      Copy data from the CPU to GPU
5      Define number of time steps $Nts$
6      for $k=0$ until $Nts$ do                                         // time loop
7          for $l=0$ until $s$ do                                       // RK s stages
8              <surface_integral_Kernel>                              //Calculate $\mathbf{P_q}$
9              <volume_integral_Kernel>                              // Calculate $rhs_\mathbf{q}$
10             <time_integration_Kernel>                             // Update $\mathbf{q}$
11         endfor
12         Copy data from the GPU to CPU
13     endfor
14     return q
15 end
```

3.5.1 The surface integral kernel

As can be seen in Algorithm 1, the surface integral kernel is the first step in our parallel-GPU DGTD program. This kernel is used to calculate the terms $\mathbf{P_q}$ shown in Eq. 2.63. In order to organize the data, we developed a routine, in which the number of threads and blocks are calculated depending on the number of elements and Degrees of Freedom (DOF) in the mesh. This routine uses the principle of one thread for each node in an element. Before starting with the program description, we would like to introduce some parameters that will be very useful later. K is the number of elements in the computational domain. N_p is the number of DOF for each field component in each element. This parameter is obtained depending on the polynomial order N. The number of DOF in each element face is called N_{fp}. Finally, N_{faces} is the number of faces in each element, e.g., for 2D problems with triangle discretization $N_{faces} = 3$. The decomposing task into an appropriate set of blocks depends directly on the problem and the polynomial order used. However, there are some strategies that can be interesting in order to take advantage of the hardware specifications. Remembering that the minimum execution unit on the GPU is a warp composed of 32 threads, we try to organize the number of threads into a block as a multiple of 32. According to the authors in [17], it is a good idea to choose a number of threads per block between 64 and 128. Thus, considering the hardware specifications we made a routine that find the smallest number of elements per block, K_f, to ensure the least amount of wasted memory. The routine to calculate the number of threads per block and the number of blocks used in the element surface integral kernel is summarized as follows:

- Step 1: Calculate the total number of threads needed in the problem. Note that the number of DOF for each flux component in each element is equal to $N_{fp} * N_{faces}$. Thus the total number of DOF for each flux component in the surface integral kernel is given by:

$$N_{threads} = N_{fp} * N_{faces} * K \tag{3.1}$$

- Step 2: Calculate the number of blocks necessary in the problem.

$$Blocks_{flux} = ceil(\frac{N_{threads}}{128}) \tag{3.2}$$

  The function *ceil* returns the smallest integer greater than $N_{threads}/128$.

- Step 3: Calculate the number of elements per block.

$$K_f = floor(\frac{128}{N_{fp} * N_{faces}}) \tag{3.3}$$

  The function *floor* returns the largest integer not greater than $128/(N_{fp} * N_{faces})$.

- Step 4: Calculate the number of threads used in each block

$$Threads_{flux} = N_{fp} * N_{faces} * K_f \tag{3.4}$$

- Step 5: Verify if the threads and blocks distribution is correct

$$while(Threads_{flux} * Blocks_{flux} < N_{threads}) \tag{3.5}$$

  If true, go to Step 6. Else, go to step 7.

- Step 6: Add one more block

$$Blocks_{flux} = Blocks_{flux} + 1 \tag{3.6}$$

- Step 7: Calculate the number of elements that must be added to complete the distribution

$$Kf_{pad} = Threads_{flux} * Blocks_{flux} - K \tag{3.7}$$

- Step 8: Modify the flux components size using a padding process (add some zeros to match the new domain size). Thus, the total number of elements in the computational domain is increased to match the DOF distribution. It can be seen in Figure 12.

$$Kf_{New} = K + Kf_{pad} \tag{3.8}$$

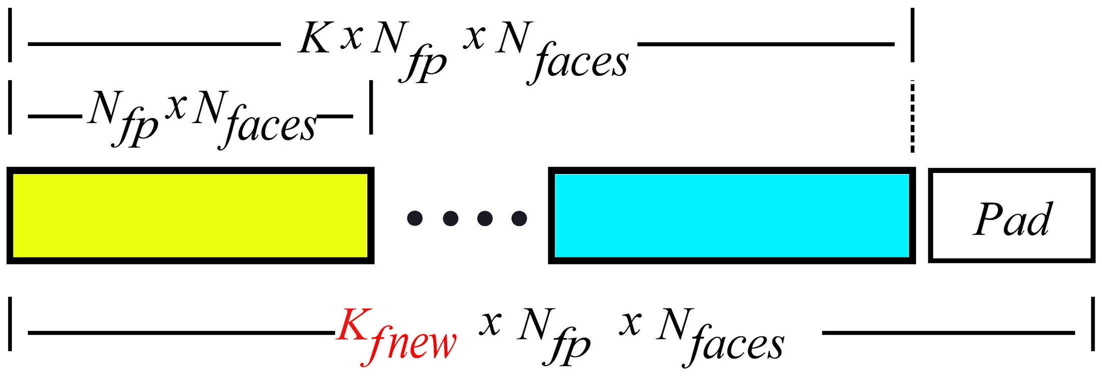

$K \, x \, N_{fp} \, x \, N_{faces}$
$N_{fp} \, x \, N_{faces}$
Pad
$K_{fnew} \, x \, N_{fp} \, x \, N_{faces}$

Figure 12 – DOF layout into the element surface integral kernel

This procedure guarantees a simple and optimal data distribution for the surface integral kernel considering the NVIDIA recommendations [15]. Moreover, all operations in this kernel are based on element-wise computations which can be handled easily in the global memory using a 1D threads and blocks distribution represented by $(Threads_{flux}, 1, 1)$ and $(Block_{flux}, 1, 1)$. This means that during the execution of the kernel, each block will use a number of $Threads_{flux}$ threads while the grid will use a number of $Block_{flux}$ blocks. Algorithm 2 summarizes the operations used in the surface integral kernel in order to calculate the terms $\mathbf{P_q}$ from Eq. 2.63. The dimensions of the input arrays in this algorithm are modified by applying the padding process as shown in Figure 12.

Algorithm II: surface integral kernel

```
1  procedure SUR_KERNEL($\mathbf{q}$,$G_S$, $\mathbf{Flux}_{ind}$, $\mathbf{P_q}$)
2      for each block of elements $Blocks_{flux}$ do
3          Calculate terms $\Delta\mathbf{q}$ using $\mathbf{q}^+ = \mathbf{q}[\mathbf{Flux}_{ind+}]$ and $\mathbf{q}^- = \mathbf{q}[\mathbf{Flux}_{ind-}]$
4          Apply boundary conditions on terms $\Delta\mathbf{q}$
5          Calculate $ndotH$ using $\Delta\mathbf{q}$ and $\widehat{\mathbf{n}}$ from $G_s$
6          Use $ndotH$, $\Delta\mathbf{q}$ and $G_s$ to calculate Eq. 2.63
7          Store the values in $\mathbf{P_q}$
8      endfor
9      return $\mathbf{P_q}$
10 end
```

It is important to emphasize that during the development of this kernel, numerical tests were conducted to compare the performance when using global and shared memory. The results showed no significant difference in the execution time between the two kernels, indicating that both shared memory and global memory are viable options for this task. This is because all the operations in the kernel are based on element-wise computations, which can be efficiently handled in global memory. Additionally, since all the threads have access to the global memory, applying boundary conditions is more straightforward when using it. Furthermore, using the NVIDIA Visual Profiler [78] which is a tool used to analyze and optimize CUDA applications, we found that the importance of this kernel in our parallel-GPU DGTD program is less than 4 %, see Figure 13. This means that even optimizing the flux kernel calculations, the impact on the entire program will not be significant. On the other hand, it can be seen in Figure 13 that the most important in terms of importance with almost 94% is the volume integral kernel. It makes sense because this kernel computes the curl terms of Eq. 2.62, which involve more complex operations (e.g. matrix-matrix multiplication) to calculate the $-x$, $-y$ and $-z$ derivatives of the electromagnetic field components.

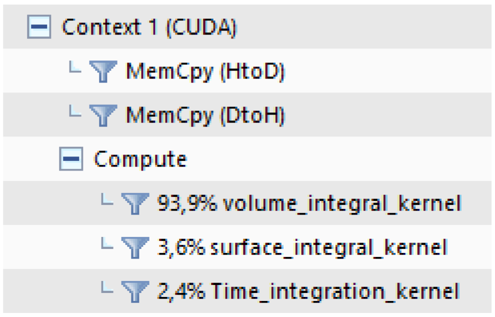

```
Context 1 (CUDA)
  MemCpy (HtoD)
  MemCpy (DtoH)
  Compute
    93,9% volume_integral_kernel
    3,6% surface_integral_kernel
    2,4% Time_integration_kernel
```

Figure 13 – Nvidia visual profiler: Kernel importance

3.5.2 The volume integral kernel

As mentioned before, this kernel is developed with the purpose of calculating the right-hand side of Eq. 2.62. The procedure of the DOF distribution discussed previously must be modified because it is necessary to handle matrix operations. These operations are more complicated when 1D blocks are used. Therefore, it is necessary to use a 2D block organization. Now, the matrix data is accessed using a set of threads which index the rows and columns respectively. Two fundamental operations must be done in this kernel. First, the matrix-matrix multiplication between the differentiation matrix, with dimension $N_p \times N_p$, and the field component matrix of size $N_p \times K$. Second, the matrix-matrix multiplication between the LIFT matrix, with dimension $N_p \times N_{faces} \cdot N_{fp}$, and the $\mathbf{P_q}$ term calculate in the surface kernel with size $N_{faces} \cdot N_{fp} \times K$. Again, the principle of one thread per output was used, considering a number between 64 and 128 threads per block.

Numerical tests conducted during the development of this kernel demonstrated an improvement in the execution time when shared memory was used. This is achieved by storing and loading the threads into shared memory in row-major order, as suggested in [9]. This ordering ensures that each thread accesses only one memory location per bank, avoiding the bank conflict problem and maximizing the memory bandwidth. Furthermore, the lower latency and higher memory bandwidth of shared memory make it a better option for boosting the performance of this kernel compared to global memory. The routine for calculating the number of threads and blocks used in the element volume integral kernel is summarized as follows:

- Step 1: Calculate the total number of threads needed in the problem

$$Nv_{threads} = N_p * K \tag{3.9}$$

- Step 2: Set the parameter N_p as the number of threads used in the x direction. Moreover, it is necessary to calculate the number of threads into the y direction which represents the number of elements K_v in each block. Therefore, each block is formed by a set of N_p threads in x dimension and K_v threads in y dimension. The value of the parameter K_v is calculated considering the NVIDIA recommendations [15].

$$K_v = floor(\frac{128}{N_p}) \tag{3.10}$$

- Step 3: Calculate the number of blocks needed in the problem

$$Blocks_{volume} = ceil(\frac{Nv_{threads}}{128}) \tag{3.11}$$

- Step 4: Verify if the threads and blocks distribution is correct

$$while(N_p * K_v * Blocks_{volume} < Nv_{threads}) \tag{3.12}$$

  If true, go to Step 5. Else, go to step 6.

- Step 5: Add one more block

$$Blocks_{volume} = Blocks_{volume} + 1 \tag{3.13}$$

- Step 6: Calculate the number of elements that must be added to complete the distribution

$$Kv_{pad} = Blocks_{volume} * K_v - K \tag{3.14}$$

- Step 7: Modify the field components size using a padding process (add some zeros to match the new domain size). Therefore, the total number of elements in the computational domain is increased in order to match the DOF distribution. It can be seen in Figure 14.

$$Kv_{New} = K + Kv_{pad} \tag{3.15}$$

As in the previous kernel, this procedure provides an optimal DOF distribution for 2D blocks represented by $(N_p, K_v, 1)$ and $Blocks_{volume}$. This means that during the execution of the volume integral kernel, each block will use a number of $N_p \times K_v$ threads while the grid will use a number $Blocks_{volume}$ blocks. This block configuration is very used in literature in order to hand with matrix operations [17]. As we mentioned before, this kernel was developed using both shared and global memory. However, due to the complexity of the matrix-matrix multiplication program, the use of shared memory provides better results in terms of computational efficiency when compared with global memory. The results of this kernel using both shared and global memory are discussed in chapter 5. Finally, the procedure for calculating the right-hand side of Eq. 2.62 is summarized in Algorithm 3. As in the surface kernel, the dimensions of the input arrays in this algorithm are modified by applying the padding process as shown in Figure 14.

Algorithm III: volume integral kernel (Shared memory)

```
1  procedure VOL_KERNEL($\mathbf{q}$,$G_v$,$\mathcal{D}_m$,LIFT , $\mathbf{P_q}$,$rhs_\mathbf{q}$)
2      for each block of elements $Blocks_{volume}$ do
3          Send arrays $\mathbf{q}, G_v, \mathcal{D}_m$, to shared memory
4          for each element $K_v$ of each block $Blocks_{volume}$ do
5              Load geometric factor from $G_v$
6              Load field components from $\mathbf{q}$
7              Load differentiation matrices from $\mathcal{D}_m$
8              Compute the volume terms of Eq. 2.62 and store in $rhs_\mathbf{q}$
9          endfor
10         Get $rhs_\mathbf{q}$ from shared memory and store in global memory
11         Send arrays $rhs_\mathbf{q}, LIFT, \mathbf{P_q}$ to shared memory
12         for each element $K_v$ of each block $Blocks_{volume}$ do
13             Load flux field components from $\mathbf{P_q}$
14             Load LIFT matrix
15             Load $rhs_\mathbf{q}$
16             Compute the surface terms of Eq. 2.62 and store in a temporal array
17             Update $rhs_\mathbf{q}$ using $rhs_\mathbf{q} = rhs_\mathbf{q}+$ temporal array
18         endfor
19         Get $rhs_\mathbf{q}$ from shared memory and store in global memory
20     endfor
21     return $rhs_\mathbf{q}$
22 end
```

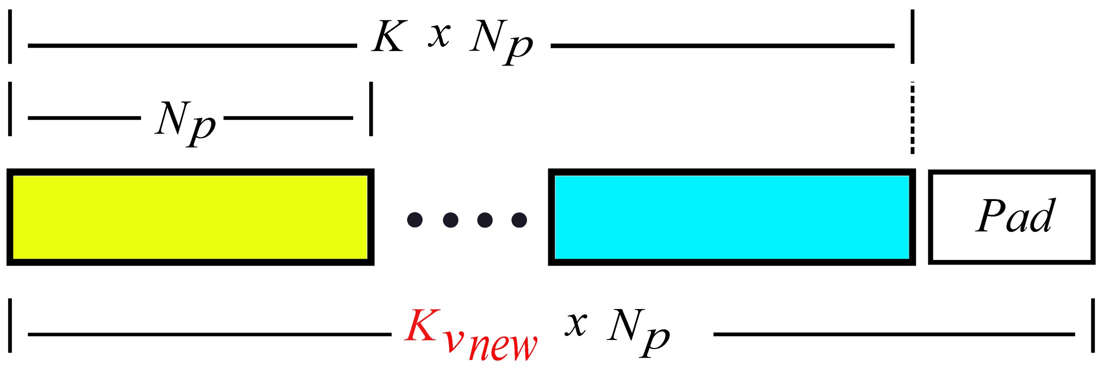


Figure 14 – DOF layout into the element volume integral kernel

3.5.3 The time integration kernel

In this kernel, it is performed the time integration by using the RK method which can be summarized into element-wise operations between the RK coefficients [4] and the field components. As can be seen in Algorithm 1, this integration method needs s stages to be performed depending on the order of approximation. The element-wise operations in this kernel are performed using the same DOF distribution presented in the volume integral kernel. According to the authors of [17], the time integration kernel is relatively simpler than the element volume and element surface integral kernels due to the field components are used only once during the calculations. Thus, it is not necessary to use shared memory. This statement is confirmed when viewing the importance of the kernel shown in Figure 13, which is less than 2.5%. The process to update the field components in the time integration kernel are summarized in Algorithm 4.

Algorithm IV: Time integration kernel

```
1 procedure TIME_KERNEL($\mathbf{q}$,$rhs_{\mathbf{q}}$,$\Delta t$,$l$)
2     for each block of elements $Blocks_{flux}$ do
3         Define the RK constant coefficients of stage $l$ $(a_{lj}, b_l, c_l)$   // See Section 2.5
4         Update term q using Eq. 2.68: $\mathbf{q} = \mathbf{q} + rhs_{\mathbf{q}} \cdot RK_{coefficients} \cdot \Delta t$
5     endfor
6     return q
7 end
```

3.6 Chapter Conclusions

This chapter introduces the fundamental principles of parallel computing with Graphics Processing Units (GPUs), with a focus on NVIDIA hardware and the CUDA programming model. It begins by tracing the evolution of GPUs from graphics-specific accelerators to powerful parallel computing platforms. The architectural differences between

CPUs and GPUs are discussed within the context of heterogeneous computing. The chapter then explores CUDA's thread and memory hierarchies, emphasizing strategies for efficient data handling and thread distribution to maximize performance. These concepts are applied to the implementation of the Discontinuous Galerkin Time-Domain (DGTD) method on GPUs, which is decomposed into three main CUDA kernels: surface integral, volume integral, and time integration. The chapter concludes with a detailed analysis of each kernel, presenting optimal thread-block configurations and discussing trade-offs involving latency, memory usage, and hardware limitations.

4 Local Time Stepping

The semi-discrete formulation of the DGTD method offers simplicity in implementation and flexibility in using explicit high-order time integration methods. These features open up the possibility of choosing a local time step that guarantees the stability of a set of elements, according to their geometrical size. This approach is clearly advantageous in terms of computational efficiency since each set of elements can advance in time using the maximum stable time step specific to its characteristics, rather than being constrained by a global time step dictated by the smallest element in the mesh. The literature contains numerous studies exploring the use of Runge-Kutta (RK) methods with local time-stepping procedures [31–33, 79]. Among these, the works of Krivodonova [79] and Trahan *et al.* [31] stand out as pioneering contributions to the development of LTS schemes for second-order RK methods (RK2). The primary distinction between these works lies in how the continuity of the solution across different classes of elements is maintained. Krivodonova [79] employs a second-order interpolating polynomial to enforce continuity, while Trahan *et al.* [31] utilize a linear combination of current and previous values to achieve the same goal.

In 2014, Angulo *et al.* [32] introduced the causal path LTS method for the Low-Storage Explicit RK fourth-order method (LSERK4). This method enforces solution continuity across element classes through recurrent application of the integration scheme, eliminating the need for previous solutions or interpolations. This approach effectively controls errors related to dispersion and dissipation, though its computational efficiency is limited by the need to compute five intermediate stages. In 2016, Ashbourne proposed an efficient LTS method for third- and fourth-order RK methods (RK3 and RK4) [33]. This method relies on third- and fourth-order interpolating polynomials to enforce solution continuity across different element classes. According to the author, both schemes perform well in terms of error and convergence rates. However, the RK4 LTS scheme reduces the stability region, necessitating smaller time steps to ensure convergence. Considering the features of these LTS methods, the LTS RK3 method emerges as the most favorable option, balancing efficiency, precision, and stability. Consequently, this chapter focuses on the formulation of the LTS RK3 method, as described in [33], and its application to the DGTD method. Finally, the chapter explores the integration of the parallel-GPU DGTD method with the LTS scheme (GPU-DGTD-LTS).

4.1 RK3 with LTS

Let us assume that we would like to numerically solve the system of two ordinary differential equations given by:

$$x' = f(x,y) \tag{4.1a}$$

$$y' = g(x,y) \tag{4.1b}$$

where $x = x(t)$ and $y = y(t)$.

Moreover, suppose that solutions x and y can be advanced in time with a different time step. That means, x uses a time step size Δt while y advance with a time step $\Delta t/\mathbb{K}$, $\mathbb{K} \in \mathbb{N}$. This approximation is based on approaches presented in [33, 79] where the authors show an efficient way to apply an LTS strategy to the Runge-Kutta methods. The strategy is divided into two basic steps: First, the solution with large time step, x, advances in time using an approximation of the inner stages of y. Second, solution y is achieved by using a polynomial approximation of solution x. These approximation use values of both current and previous time levels in order to obtain a solution that is at least as accurate as the local error of the Runge-Kutta stages. An interpolating polynomial $\chi(t)$ is created once solution x has been advanced one time step. This interpolating polynomial approximates the solution $x(t)$ on the time interval $[t_n, t_{n+1}]$ and must be at least as accurate as the Runge-Kutta scheme used (e.g, a third-order interpolating polynomial should be used for the RK3 method). Note that the y solution needs of $\mathbb{K}$ sub time steps to advance from t_n to t_{n+1}, so, both the interpolating polynomial and its derivatives are essential to calculate the inner stages of x.

This LTS strategy is based on the family of a three-stage, RK3 method shown in chapter 2. Specifically, we will focus on the RK3 method with the parameter $\beta = 3/8$ which result in the system of Eqs 2.71. First, we assume that both solutions x and y are known at time t_n. Next, considering that this approximation depends on previous values. We assume that the previous values were stored in the previous time level. These previous values are called $f(x_{n-1}, y_{n-1})$ and $g(x_{n-1}, y_{n-1})$. Note that to advance from t_{n-1} to t_n, it is necessary to choose a time step size that satisfies the CFL condition. For convenience, we use the time step of the y solution which can be considered as a "global time step" because it was chosen using the smallest element in the mesh. In order to advance the x solution from t_n to t_{n+1} we have to approximate the inner or intermediates stages of y. This can be done using the scheme shown below:

$$\phi_2^{(x)} = x_n + \frac{2}{3}\Delta t f(x_n, y_n) \tag{4.2a}$$

$$\phi_2^{(y)} = y_n + \frac{2}{3}\Delta t g(x_n, y_n) \tag{4.2b}$$

$$\phi_3^{(x)} = x_n + \frac{2}{3}\Delta t f(\phi_2^{(x)}, \phi_2^{(y)}) \tag{4.2c}$$

$$\phi_3^{(y)} = y_n + \frac{2}{3}\Delta t g(x_n, y_n) + \frac{4}{9}\Delta t^2 \left(\frac{g(x_n, y_n) - g(x_{n-1}, y_{n-1})}{\Delta t/\mathbb{K}}\right) \tag{4.2d}$$

$$x_{n+1} = x_n + \frac{\Delta t}{4}\left(f(x_n, y_n) + \frac{3}{2}f(\phi_2^{(x)}, \phi_2^{(y)}) + \frac{3}{2}f(\phi_3^{(x)}, \phi_3^{(y)})\right) \tag{4.2e}$$

The intermediate stages for the RK3 method can be seen in Eq. 4.2. However, the term $\phi_3^{(y)}$ presents a little difference compared with its counterpart term $\phi_3^{(x)}$. According to [33, 79] the final RK stage of y can be approximated by using Taylor series expansion:

$$\phi_3^{(y)} = y_n + \frac{2}{3}\Delta t f\left(x_n + \frac{2}{3}\Delta t f(x_n, y_n), y_n + \frac{2}{3}\Delta t g(x_n, y_n)\right) \tag{4.3a}$$

$$\phi_3^{(y)} = y_n + \frac{2}{3}\Delta t g(x_n, y_n) + \frac{4}{9}\Delta t^2 (g_x f + g_y g)(x_n, y_n) \tag{4.3b}$$

Note that the term $g(x_n, y_n)$ presents in Eq. 4.3b is known, so, we only need to find an approximation for the term $(g_x f + g_y g)(x_n, y_n)$. According to [33], this term can be considered as

$$(g_x f + g_y g)(x(t), y(t)) = \frac{d}{dt} g(x(t), y(t)) \tag{4.4}$$

Finally, this derivative can be calculated by using the backward difference method. It can be done using the stored values from the previous time step.

$$\frac{d}{dt} g(x(t), y(t)) = \frac{g(x_n, y_n) - g(x_{n-1}, y_{n-1})}{\Delta t/\mathbb{K}} \tag{4.5}$$

Once the solution x_n has advanced to x_{n+1}, it is necessary to construct a third-degree polynomial, $\chi(t)$, which interpolates the solution x along the interval $[t_n, t_{n+1}]$. Considering that to construct a n order polynomial we have to know $n + 1$ points, we require four points to create our approximation. These points are given by:

$$\chi(t_n) = x_n \tag{4.6a}$$
$$\chi^{'}(t_n) = f(x_n, y_n) \tag{4.6b}$$
$$\chi^{'}(t_n + 2\Delta t/3) = \left(f(\phi_2^{(x)}, \phi_2^{(y)}) + f(\phi_3^{(x)}, \phi_3^{(y)})\right)/2 \tag{4.6c}$$
$$\chi(t_{n+1}) = x_{n+1} \tag{4.6d}$$

Considering these four points the third-degree polynomial can be expressed by [33]:

$$\chi(t) = x_n + (t - t_n)f(x_n, y_n) + (t - t_n)^2\left(\frac{x_{n+1} - x_n - \Delta t f(x_n, y_n)}{\Delta t^2} - \Delta t\vartheta\right)$$
$$+(t - t_n)^3\vartheta, \qquad t_n \le t \le t_{n+1} \tag{4.7}$$

where ϑ is a parameter calculated under the assumption that $|x(t) - \chi(t)| = \mathcal{O}(\Delta t^4)$ for all interval $t_n \le t \le t_{n+1}$. According to [33] this is satisfied when:

$$\vartheta = \frac{1}{2\Delta t + 3(\Delta t/\mathbb{K})}\left(2\frac{x_{n+1} - x_n - \Delta t f(x_n, y_n)}{\Delta t^2} - \frac{f(x_n, y_n) - f(x_{n-1}, y_{n-1})}{(\Delta t/\mathbb{K})}\right) \tag{4.8}$$

Now, we have to advance the y solution through the sub time steps $t_{n,k} = t_n + k\Delta t/\mathbb{K}, \quad k = 0, 1, ..., \mathbb{K}$. In this notation, we use the term $y_{n,k}$ to denote the value of y at time $t_{n,k}$. Therefore, we set $y_{n,0} = y_n$ and $y_{n,\mathbb{K}} = y_{n+1}$.In addition, let consider $\phi_{i,k}^{(x)}, \phi_{i,k}^{(y)}$, $i = 2, 3$, as the intermediate RK stages at the fractional step k. Based on these considerations, the scheme to advance from $y_{n,k}$ to $y_{n,k+1}$ are presented as follows:

$$x_{n,k} = \chi(t_{n,k}) \tag{4.9a}$$
$$\phi_{2,k}^{(x)} = x_{n,k} + \frac{2}{3}\left(\frac{\Delta t}{\mathbb{K}}\right)\chi^{'}(t_{n,k}) \tag{4.9b}$$
$$\phi_{2,k}^{(y)} = y_{n,k} + \frac{2}{3}\left(\frac{\Delta t}{\mathbb{K}}\right)g(x_{n,k}, y_{n,k}) \tag{4.9c}$$
$$\phi_{3,k}^{(x)} = x_{n,k} + \frac{2}{3}\left(\frac{\Delta t}{\mathbb{K}}\right)\chi^{'}(t_{n,k}) + \frac{4}{9}\left(\frac{\Delta t}{\mathbb{K}}\right)^2\chi^{''}(t_{n,k}) \tag{4.9d}$$
$$\phi_{3,k}^{(y)} = y_{n,k} + \frac{2}{3}\left(\frac{\Delta t}{\mathbb{K}}\right)g(\phi_{2,k}^{(x)}, \phi_{2,k}^{(y)}) \tag{4.9e}$$
$$y_{n,k+1} = y_{n,k} + \frac{\Delta t/\mathbb{K}}{4}\left(g(x_{n,k}, y_{n,k}) + \frac{3}{2}g(\phi_{2,k}^{(x)}, \phi_{2,k}^{(y)}) + \frac{3}{2}g(\phi_{3,k}^{(x)}, \phi_{3,k}^{(y)})\right) \tag{4.9f}$$

The Eq. 4.9 shows all standard steps of the RK3 method for the y solution. However, there are some terms as $\chi^{'}(t_{n,k})$ and $\chi^{''}(t_{n,k})$ which must be discussed with more details. According to [33], we can consider that $f(x(t_{n,k}), y(t_{n,k})) = x^{'}(t_{n,k}) = \chi^{'}(t_{n,k}) + \mathcal{O}(\Delta t^3)$. Moreover, the third RK stage of the x solution in a Taylor series about $t_{n,k}$ yields

$$\phi^{(x)}_{3,k} = x_{n,k} + \frac{2}{3}\left(\frac{\Delta t}{\mathbb{K}}\right) f\left(x_{n,k} + \frac{2}{3}\left(\frac{\Delta t}{\mathbb{K}}\right) f(x_{n,k}, y_{n,k}), y_{n,k} + \frac{2}{3}\left(\frac{\Delta t}{\mathbb{K}}\right) g(x_{n,k}, y_{n,k})\right) \tag{4.10a}$$

$$\phi^{(x)}_{3,k} = x_{n,k} + \frac{2}{3}\left(\frac{\Delta t}{\mathbb{K}}\right) f(x_{n,k}, y_{n,k}) + \frac{4}{9}\left(\frac{\Delta t}{\mathbb{K}}\right)^2 (f_x f + f_y g)(x_{n,k}, y_{n,k}) \tag{4.10b}$$

Observe that the term $(f_x f + f_y g)(x_{n,k}, y_{n,k})$ can be approximated similarly as in Eq 4.4 by using the backward difference method. Finally, the proof that this scheme provides a third-order approximation is shown in [33].

4.2 Discontinuous Galerkin Method with LTS

The DG method along with the RK time integration scheme was discussed in chapter 2. As mentioned before, the most common DG implementation presents a system of ordinary differential equations solved by using a global time step method. Although this strategy presents good results in terms of precision, this approach presents an important limitation because the stability of the method depends on the smallest element in the mesh. Thus, the LTS strategy shown in the previous section can be used to alleviate this limitation.

This LTS strategy is relatively straightforward, first, we define the reference time step Δt which is calculated considering the largest element in mesh with size r. Then, elements are grouped into a "class" based on their size relative to the maximum element size. Finally, element k with size r_k will be store into class(i) whenever satisfy the condition $r/2^{i+1} \leq r_k < r/2^i$. Elements store in class(i) will advance with time step $\Delta t/2^{i+1}$. Additionally, the elements are organized depending on which class they are in and which class the neighbor elements are in. A neighbor element is defined as an element that shares an edge or a face in two or three dimensions, respectively. Inside the classes, elements are organized as follows: large boundary elements that have at least one neighboring element that belongs to a smaller element class. Small boundary elements that have at least one neighboring element that belongs to a larger element class. Interior elements which all neighboring elements belong to the same class. Figure 15 illustrates the boundary interface Γ_t between elements of different classes. Note that the elements on the left side are larger than the elements on the right side. Therefore, left elements are able to advance at a time step Δt while right elements $\Delta t/2$.

According to the right-hand side of the strong variational formulation presented in Eq. 2.16, the communication between the elements is done only by the use of the numerical flux. In addition, numerical flux can be used to impose boundary conditions using the principle of ghost cells [77]. Therefore, these ghost cells can be used to deal with the computation of the surface integral on the interface between elements of different classes. The strategy consists in advancing each class of elements with its relative time step as its own smaller problem, considering the neighboring elements from different classes as the physical boundaries. These neighboring elements use the principle of the ghost cells and they are needed to approximate each one of the RK stages. This LTS strategy presents a crucial advantage in terms of implementation due to the time stepping process only must be modified on interface elements. Thus, the standard RK method can be applied to the interior elements using their local stable time step. In addition, this LTS method can be implemented over a preexisting code with small modifications and maintaining the entire functionality.

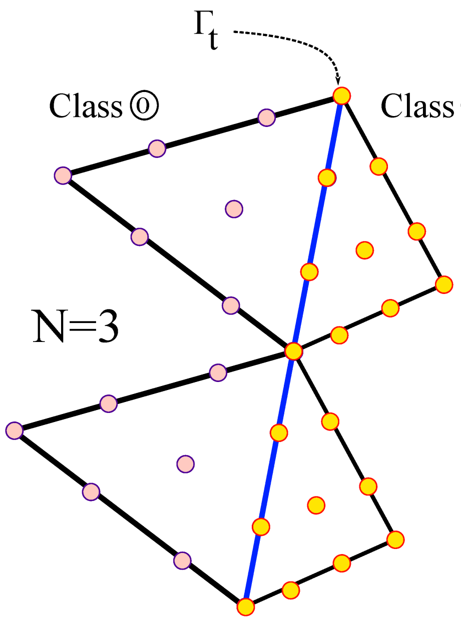


Figure 15 – Boundary between elements of different classes considering a polynomial basis $N = 3$.

To adapt the RK3-LTS scheme presented in the previous section to the DG method, we start by assuming that all elements are at the same time level t_n. It can be achieved easily by advancing all elements by using the time step of the class with smaller elements as a global time step. Also, assume that the term $f(\mathbf{u})$ has been stored at the time t_{n-1}. The term $\mathbf{u}$ contains the information of large and small interface elements. Now, the large elements start by advancing with a reference time step Δt as shows in Eq. 4.2a. Then, the values for the second RK stage in the small interface elements can be calculated by using Eq. 4.2b which can be rewritten as:

$$\phi_2^{si,G} = \mathbf{u}_n^{si} + \frac{2}{3}\Delta t f^{si}(\mathbf{u}_n) \tag{4.11}$$

where the term G denotes that these values will be used as physical boundaries applying the principle of ghost cells.

Next, the third intermediate stage for the large element class can be calculated by using Eq. 4.2c. Additionally, the third intermediate stage for the small element class is given by Eq. 4.2d which can be rewritten as:

$$\phi_3^{si,G} = \mathbf{u}_n^{si} + \frac{2}{3}\Delta t f^{si}(\mathbf{u}_n) + \frac{4}{9}\Delta t^2 \left(\frac{f^{si}(\mathbf{u}_n) - f^{si}(\mathbf{u}_{n-1})}{\Delta t/2}\right) \tag{4.12}$$

Note that previous values stored at the time t_{n-1} and the time step of the smaller class elements are used in Eq. 4.12. These values are fundamentals to apply the backward difference method. Finally, all intermediate stages are substituted in Eq. 4.2e and now large elements can advance from t_n to t_{n+1}.

Once the large elements have advanced in time, a third-order interpolation polynomial must be calculated to use it as a boundary condition along the large element interface. This polynomial is used to calculate the intermediate stages of large interface elements during the time-marching process of small elements. Rewritten the Eq. 4.7, the interpolating polynomial is given by:

$$\chi^{li}(t) = \mathbf{u}_n^{li} + (t - t_n) f^{li}(\mathbf{u}_n) + (t - t_n)^2 \left(\frac{\mathbf{u}_{n+1}^{li} - \mathbf{u}_n^{li} - \Delta t f^{li}(\mathbf{u}_n)}{\Delta t^2} - \Delta t \vartheta^{li}\right) + (t - t_n)^3 \vartheta^{li} \tag{4.13}$$

$$\vartheta^{li} = \frac{1}{2\Delta t + 3(\Delta t/2)} \left(2\frac{\mathbf{u}_{n+1}^{li} - \mathbf{u}_n^{li} - \Delta t f^{li}(\mathbf{u}_n)}{\Delta t^2} - \frac{f^{li}(\mathbf{u}_n) - f^{li}(\mathbf{u}_{n-1})}{\Delta t/2}\right) \tag{4.14}$$

for $t_n \leq t \leq t_{n+1}$.

The advancing process is very similar for the small class elements. However, in this case the time step is divided into $\mathbb{K}$ levels. These sub time step levels are defined by $t_{n,k} = t_n + k\Delta t/\mathbb{K}, \quad k = 0, 1, ..., \mathbb{K}$. Considering that small elements will advance with a time step $\Delta t/2$, the number of sub time step levels $\mathbb{K}$ is equal to 2. The ghost values on the large interface elements are given by:

$$u_{n,k}^{li,G} = \chi^{li}(t_{n,k}) \tag{4.15}$$

These ghost values are necessary to calculate the intermediate stages of the RK3 method. Substituting Eq. 4.15 into Eq. 4.9b and rewriting some terms we have the coefficients for the second intermediate stage on the large elements interface:

$$\phi_2^{li,G} = \chi^{li}(t_{n,k}) + \frac{2}{3}\left(\frac{\Delta t}{2}\right)\chi'^{li}(t_{n,k}) \tag{4.16}$$

Then, we apply Eq. 4.9c to calculate the second intermediate stage on small class elements. At this point we are ready to find the third and last intermediate stage, so, based on Eq. 4.9d the ghost coefficients for the third intermediate stage on the large element interface are given by:

$$\phi_3^{li,G} = \chi^{li}(t_{n,k}) + \frac{2}{3}\left(\frac{\Delta t}{2}\right)\chi'^{li}(t_{n,k}) + \frac{4}{9}\left(\frac{\Delta t}{2}\right)^2\chi''^{li}(t_{n,k}) \tag{4.17}$$

This ghost coefficient and the values of the third intermediate stage for the small class elements calculated by using Eq. 4.9e complete the computations of the whole intermediate stages of the RK3 method. Finally, these intermediate stages are substituting in Eq. 4.9f and the small class elements have been advanced from t_n to $t_n + \Delta t/2$. This process must be repeated in small elements until complete the total number of sub time steps $\mathbb{K}$.

Until now, this LTS method has been presented considering a problem that involves only two levels of refinement (e.g., elements with size r and $r/2$). However, this LTS method can be extended to consider multiple levels of refinement [33]. Let us consider a computational domain that, after a preprocessing stage, has been divided into three different element classes, and the interface elements between these classes have been identified. An illustration of this scheme is shown in Figure 16. Furthermore, the corresponding time step values were chosen as $\Delta t, \Delta t/2$ and $\Delta t/4$ from the class with the largest time step to the class with the smallest time step, respectively. Once the time step values are defined, the time-marching process is performed recursively, starting from the class with the largest time step to the class with the smallest, as described in Algorithm V.

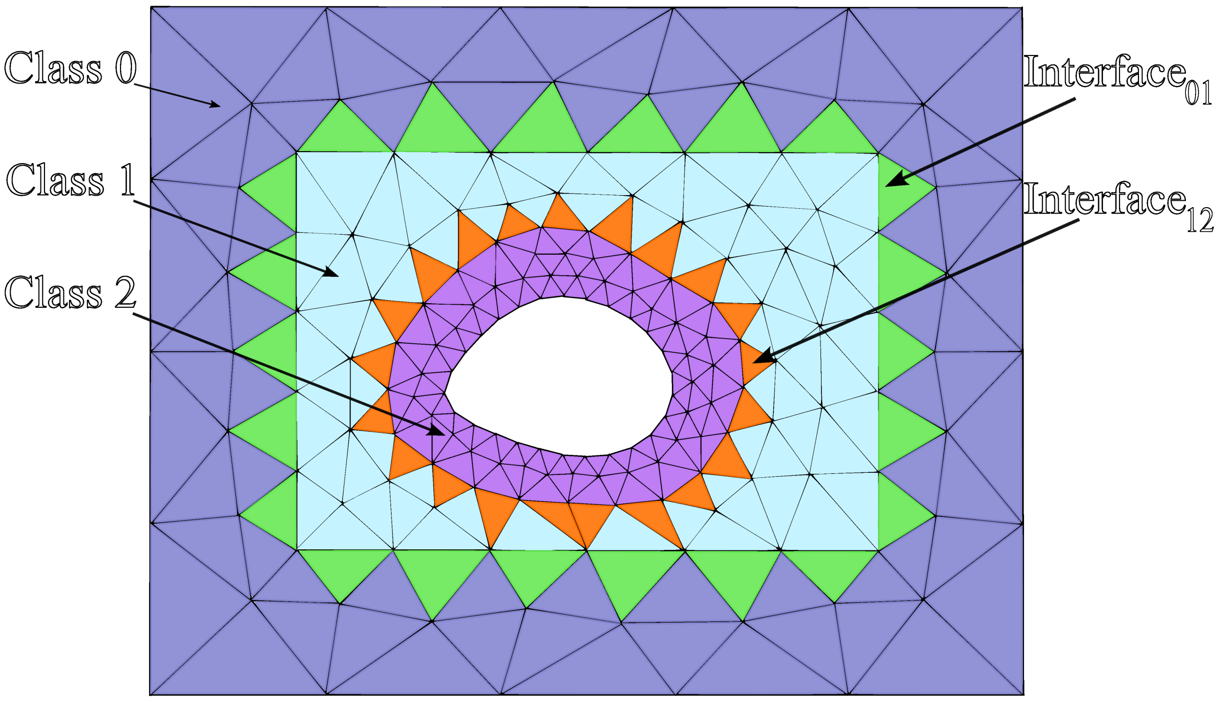


Figure 16 – Example of a 2D computational domain after the LTS classification process.

```
Algorithm V: LTS RK3 procedure for multiple levels of refinement
    Step 1: Advance all elements from time $t_{n-1}$ to $t_n$ using the local time step calculated
    for class 2, $\Delta t/2^2$, and store appropriate data.
    Step 2: Advance all elements in time using their own locally stable time step $\Delta t/2^i$.
    See Figure 17.
    - Begin by the largest elements of class 0 using Eq. 4.2.
    - Then, repeat the process for the other two classes.
    Step 3: Advance elements in class 2 by one local time step $\Delta t/4$. See Figure 18.
    - Calculate the interpolating polynomial on the interface between class 1 and class 2
    using Eq. 4.7.
    - Then, use Eq. 4.9 to advance class 2 in one local time step. Now class 1 and class 2 are
    at the same time level.
    Step 4: Advance elements in classes 1 and 2 with local time steps $\Delta t/2$ and $\Delta t/4$. See
    Figure 19.
    - Calculate the interpolating polynomial on the interface between class 0 and class 1
    using Eq. 4.7.
    - Then, use Eq. 4.9 to advance class 1 in one local time step. Now class 0 and class 1 are
    at the same time level.
    - Use Eq. 4.7 to update the elements in class 2 in one local time step.
    Step 5: Advance elements in class 2 by one local time step $\Delta t/4$. See Figure 20.
    - Repeat Step 3. Now classes 0, 1, and 2 are at the same time level.
    Step 6: Repeat Steps 2-5.
```

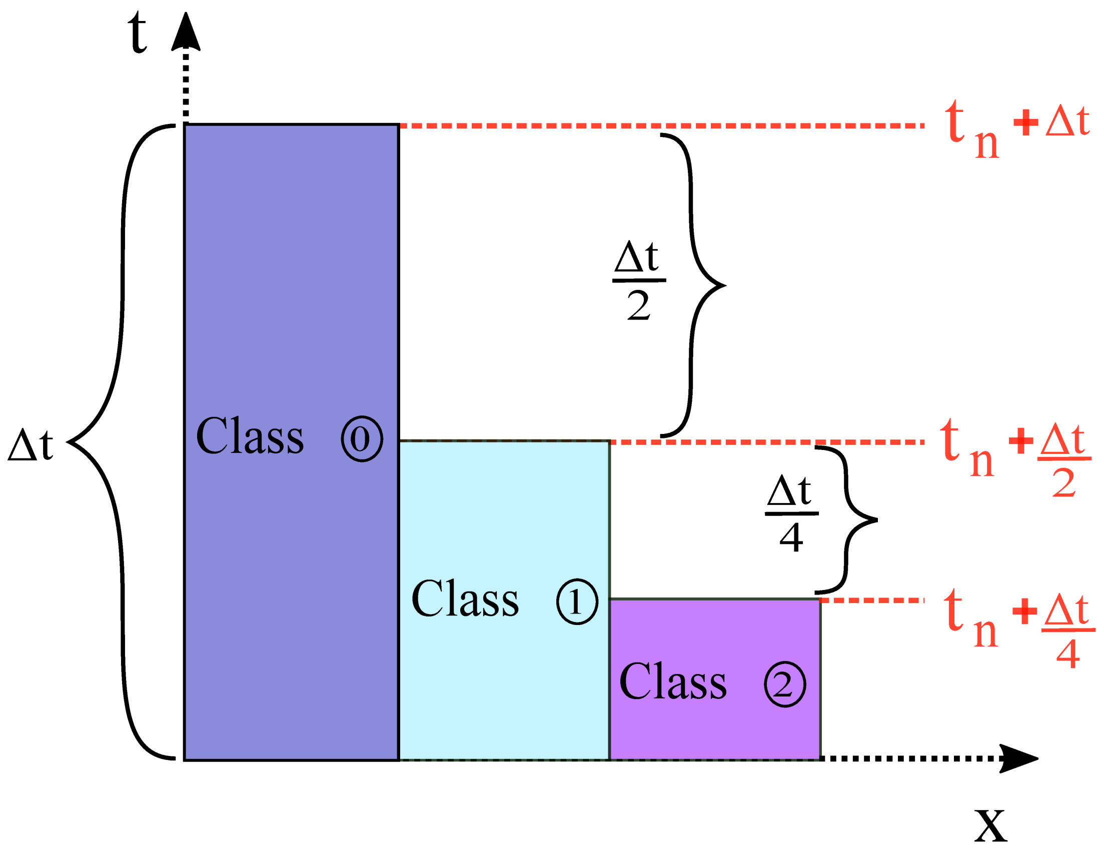


Figure 17 – Step 2 of Algorithm V

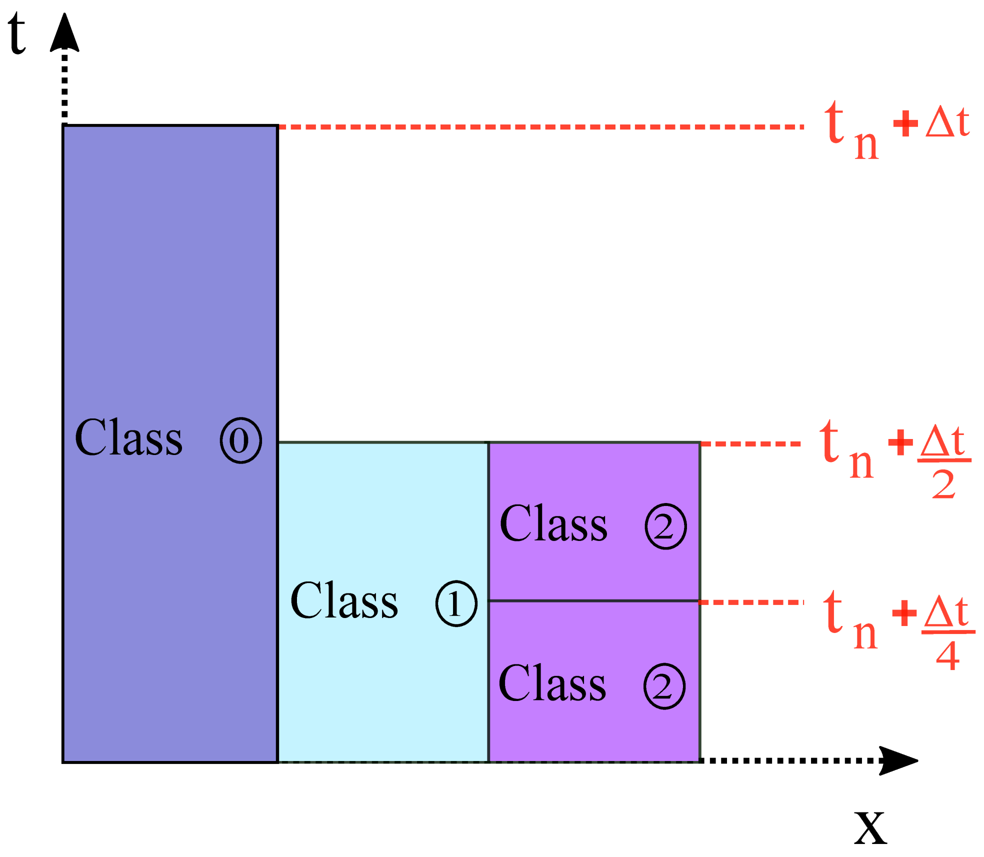


Figure 18 – Step 3 of Algorithm V

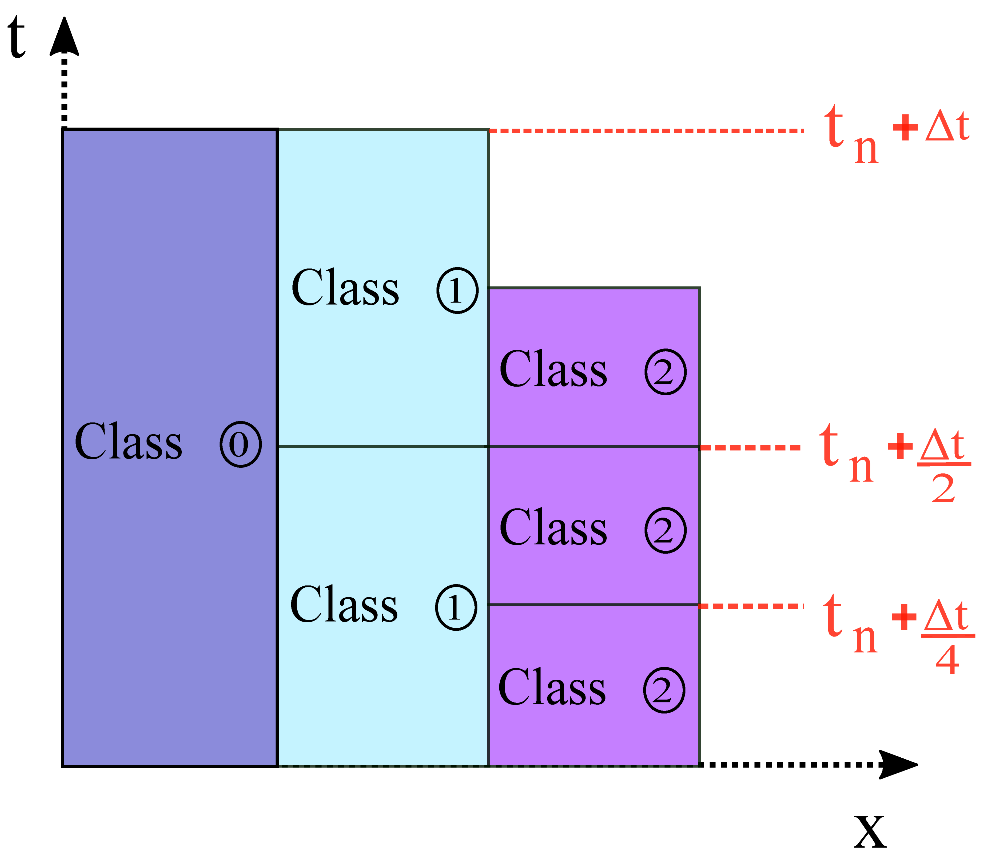


Figure 19 – Step 4 of Algorithm V

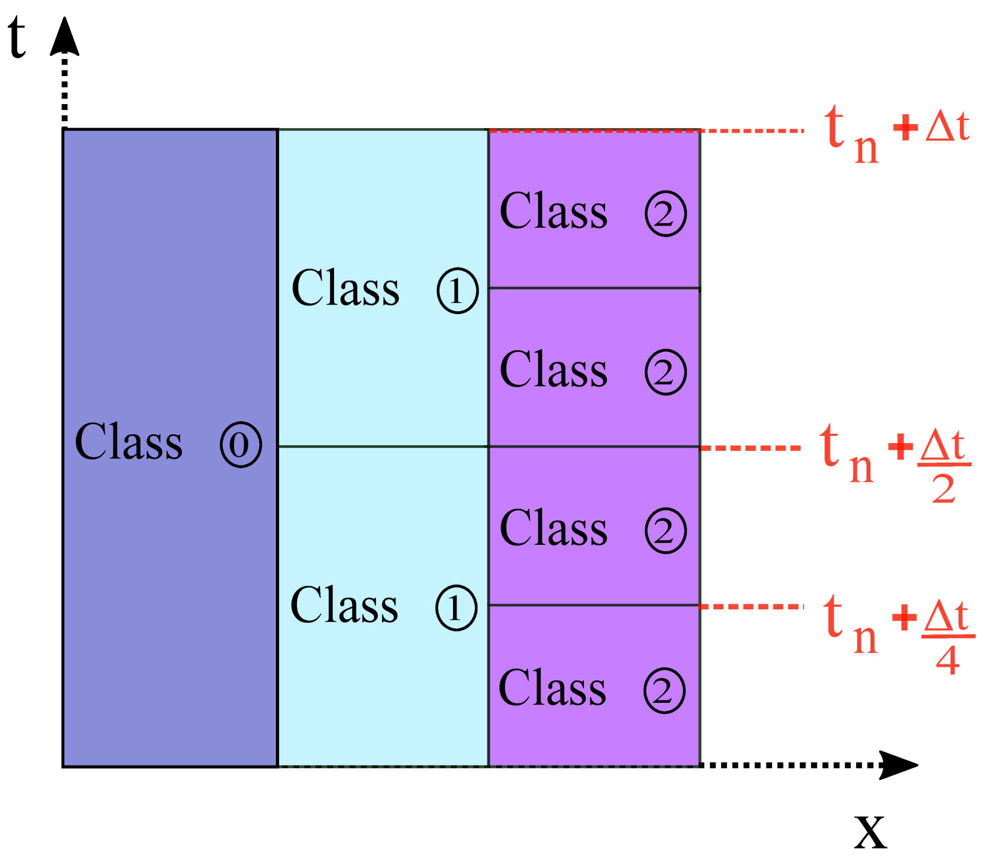


Figure 20 – Step 5 of Algorithm V

In principle, this LTS method looks more complex than the standard RK3 method due to the use of interpolations to calculate the intermediate stages on large boundary elements. However, it can be noticed that the number of intermediate stages is the same as the standard RK3, and additional evaluations of the function $f(\mathbf{u})$ are not required. The extra storage of some values in the element interfaces can be considered a possible drawback. For example, we need to store the values of $\mathbf{u}_n$, and $f(\mathbf{u}_n)$,$f(\mathbf{u}_{n-1})$ on all interface elements and $\mathbf{u}_{n+1}$ on all large interface elements. These previous time values are stored only in the elements located at the interfaces between different classes. According to [33], the total number of elements in the mesh is usually greater than the number of elements in the interfaces. Therefore, the memory requirements of this LTS method are not considered excessive. The memory requirements of both the LTS RK3 method and its classical counterpart are shown in Table 5.

Table 5 – Third-order RK memory requirements for classical and LTS scheme.

	Classical RK3	LTS RK3
Number of equations	$N_{Equations}$	$N_{Equations}$
Number of elements	K	K
DOF	N_p	N_p
Number of stages	3	3
Interface elements	0	N_{ie}
Large interface elements	0	N_{lie}

4.3 The parallel-GPU DGTD Method with LTS

According to the previous section 3.5, the execution of the DGTD method on graphic processors can be done in a straightforward way by using the CUDA programming model. Basically, we should be concerned with the organization of the threads in each block and the type of memory that will be used to do the calculations. This thread organization will depend specifically on the problem and some parameters such as the number of elements and the polynomial order of basis functions. Additionally, the type of memory that will be used depends on the operation that will be done. Operations with a high complexity will be performed in shared memory, whereas other operations will be performed in global memory. The execution of the DGTD method in a graphics processor is interesting to improve the computational efficiency of the method while maintaining the accuracy of the solution.

Moreover, section 4.2 shows another strategy widely used in the literature to improve computational efficiency known as local time step. Contrary to the parallel-GPU implementation, this LTS method does not guarantee the same precision of the solution and therefore it should not be used as a strategy to obtain better accuracy. This means that the LTS method should be seen as a way to accelerate the computation [24]. In this work, we choose an LTS method based on the RK3 method that, even losing a little precision, maintains the convergence and order of approximation of its classical counterpart. This loss of precision is caused by using interpolations in the middle of the time integration process.

Both parallel-GPU DGTD and DGTD with LTS are interesting modifications that improve the performance of the classical DGTD in terms of computational efficiency. However, these strategies have been presented as individual and separate methods so far. From this perspective, this work finds a way to combine these two modifications of the DGTD method in order to obtain a more powerful numerical technique. As mentioned before in chapter 1, there are a few works that explore the combination of the DGTD with GPU and LTS [8, 22]. Although these works presented very interesting results in the solved problems, they use an LTS scheme based on the second-order leapfrog method and a simple linear interpolation that may compromise the accuracy of the solution. This linear interpolation is given by:

$$u_{n+k/p} = \left(1 - \frac{k}{p}\right) u_n + \left(\frac{k}{p}\right) u_{n+1} \tag{4.18}$$

where $u_{n+k/p}$ represents the solution at the intermediate time $t_{n+k/p}$, $(k = 1, ..., p-1)$.

The Eq. 4.18 shows the expression used to calculate the intermediate solution between elements of different time step classes. Note that this interpolation only uses values from the solution at time t_n, u_n and t_{n+1}, u_{n+1}. Therefore, it can be considered a linear interpolation which introduces a loss of precision in the implementation. According to authors [8, 22], this LTS method based on LF2 method and linear interpolation ensures simple implementation and is very convenient for GPU. However, in this work we will show that an LTS method with a third-order interpolation that guarantees the same convergence and order of approximation as the RK3 method can also be done in the GPU in an easy way.

First, we start with the definition of the reference time step Δt which depends directly on the size of the largest element in the mesh. This Δt value will be our base to define all element classes in our method. Then, small elements will be organized into other classes depending on their size relative to the maximum element size. These other elements will advance in time depending on their class number $class(i)$ with a time step of $\Delta t/2^{i+1}$. Note that no modification of the method shown in section 4.2 has been presented so far. However, this element class organization is just the beginning of our proposal.

Once the elements are grouped into their respective classes, we start with the second part of our method. For the sake of convenience will be considered only two element classes. The $class(0)$ contains all elements which will advance with a time step Δt and the $class(1)$ whose elements will advance with a time step $\Delta t/2$. It is important to remark that elements in $class(1)$ are smaller than in $class(0)$. Now, it is time to consider the DOF distribution into the GPU. It will be done based on the same routines presented in section 3.5. The number of DOF distributions that must be done depends directly on the number of element classes in a factor of $DOF_D = N_{classes} + 1$. That is, we must create a DOF distribution for each class and one more for the entire computational domain. This additional DOF distribution is needed to start Algorithm V, which needs that all elements advance in time in a global time step from t_{n-1} to t_n. Therefore, this last DOF distribution can be calculated using exactly the same routines presented in section 3.5.

Now, the DOF distribution for the other two classes ($N_{classes}$) needs to be discussed in more detail. When the computational domain is split into two classes, many new parameters must be considered. For $class(0)$ we can extract for example K_{c0} which contains the information of the interior elements and K^i_{c0} which contains the information of the large interface elements. These parameters also can be extracted for the $class(1)$, we called them K_{c1} and K^i_{c1}. Considering that these parameters are different for each class, it is necessary to create a DOF distribution for each class individually. However, we can notice that even with different DOF distributions, there are some parameters that remain constant, for example, the number of nodes in each element N_p and the number of nodes in each edge N_{fp}. This feature makes the routines presented in section 3.5 very suitable for reuse with few changes.

Suppose that we are going to make the DOF distribution for the $class(0)$ into the GPU. According to Algorithm 1, the first CUDA kernel that will be executed is the element surface integral kernel. This kernel is responsible for surface integration. We begin by calculating the number of threads needed in the problem by using Eq. 3.1. Note that in this case, the parameter K which represents the total number of elements in the mesh has to be modified by K_{c0}, this is:

$$N_{threads,c0} = N_{fp} * N_{faces} * K_{c0} \tag{4.19}$$

Once the total number of threads necessary into the element surface integral kernel for the $class(0)$ is known, we calculate the number of threads in each block and the number of blocks. This process can be performed using the same routine presented in subsection 3.5.1, but, considering the Eq. 4.19 as Step 1. Due to the use of the same routine, we will find the same number of threads per block as in the global case. However, the difference appears when the total number of blocks for the element surface integral kernel is calculated. This is due to the fact that the number of elements in $class(0)$, K_{c0}, differs from the total number of elements in mesh, K. At the end of this routine, we obtain the parameter $Thread_{flux,c0}$ and $Blocks_{flux,c0}$ which will be considered for the DOF distribution in the kernel. As mentioned before, this DOF distribution ensures a simple and optimal data distribution that can be handled easily in the global memory using a 1D threads and blocks distribution represented by $(Threads_{flux,c0}, 1, 1)$ and $(Block_{flux,c0}, 1, 1)$ respectively.

Now it is time to calculate the DOF distribution for the volume integral kernel. Contrary to the surface kernel, the volume kernel has many complex operations based on matrix-matrix multiplications which are used to compute the spatial derivatives of field components. For the volume kernel, the number of DOF per element is equal to N_p. Therefore, the total number of threads necessary in this kernel can be calculated using Eq. 3.9 but modifying the parameter K by K_{c0}. This is:

$$Nv_{threads,c0} = N_p * K_{c0} \tag{4.20}$$

Once the total number of threads in the volume kernel is determined, the next step is to calculate the number of threads per block and the number of blocks. This calculation follows the same routine described in subsection 3.5.2, but with Eq. 4.20 used as Step 1. At the end of this routine, a 2D block organization is established, with a thread distribution of $(N_p, K_v, 1)$ and a total of $Blocks_{volume,c0}$ blocks in the grid. This DOF distribution can also be applied to the time integration kernel described in subsection 3.5.3. At this stage, the DOF distribution for the elements in $class(0)$ is complete. Notably, the same routines outlined in section 3.5 are used here with minimal modifications. This represents a key advantage of our proposed method, as the routines for DOF distributions are recursive and reusable. For the DOF distribution of $class(1)$, the same process is applied, with the sole change being the substitution of the parameter K_{c0} with K_{c1}.

Finally, we have to handle the calculation of the third-order polynomial interpolating represented by Eq. 4.13 and used in the calculation of intermediate stages of the LTS RK3 method. We try to reuse the same function created for the LTS algorithm executed on the CPU. However, this implementation requires the data exchange between the CPU and GPU for each time step and as mentioned before, this situation must be avoided as much as possible. The other alternative is the creation of one more CUDA kernel, which will be responsible for these calculations. At first glance, Eqs. 4.13 and 4.14 can be seen as complex operations due to the large number of terms involved. However, these equations can be represented by elementary operations that include simple arithmetic operations between vectors and constants. Additionally, it is easy to see that the interpolating polynomial χ^{li} will have a size of $N_p \times K^i_{c0}$. Therefore, the same DOF distribution shown in subsection 3.5.2 can be reused to organize the threads and blocks in this kernel. Figure 21 shows the DOF distribution for the different element classes 0 and 1. The DOF distribution for the kernel which calculates the interpolating polynomial is also shown. Note that the thread organization for both the surface and volume kernel are very similar for the two classes. They differ just in the number of blocks executed in the grid during the execution of each kernel. This difference between the number of blocks used for each element class is due to the difference in the number of elements for each class. The procedure to implement the Parallel-GPU DGTD method with LTS (GPU-DGTD-LTS) is summarized in Algorithm 6. It has been developed taking into consideration the Algorithm 1 shown in the previous section. For the sake of convenience, only two different element classes were considered. However, this idea can be extended for the purpose of considering multiple levels of refinement.

At first glance, Algorithm 6 is more complex than Algorithm 1 in terms of the number of operations that must be done to update the field components $\mathbf{q}$. In addition, the introduction of two loops in the new code implies an increase in the execution time, which is counterproductive in terms of efficiency. However, the computational gain in Algorithm 6 is achieved due to the decrease in the number of time steps Nts. For example: let us suppose that the program will need 2000 time steps to be executed and this number of time steps was calculated using the smallest element in mesh. Additionally, we know a priori that the domain can be divided into two classes. The first class contains the large elements that advance with a time step value, Δt_0, and the second class contains the small elements which advance with a time step value, Δt_1. This time step values are related by $\Delta t_0 = 2\Delta t_1$. In a global time step scheme, the field components for each element will be computed 2000 times (i.e. using the time step for the second class Δt_1). On the contrary, in an LTS scheme, the field components for each element in the first class will be computed 1000 times, and 2000 times for the elements in the second class.

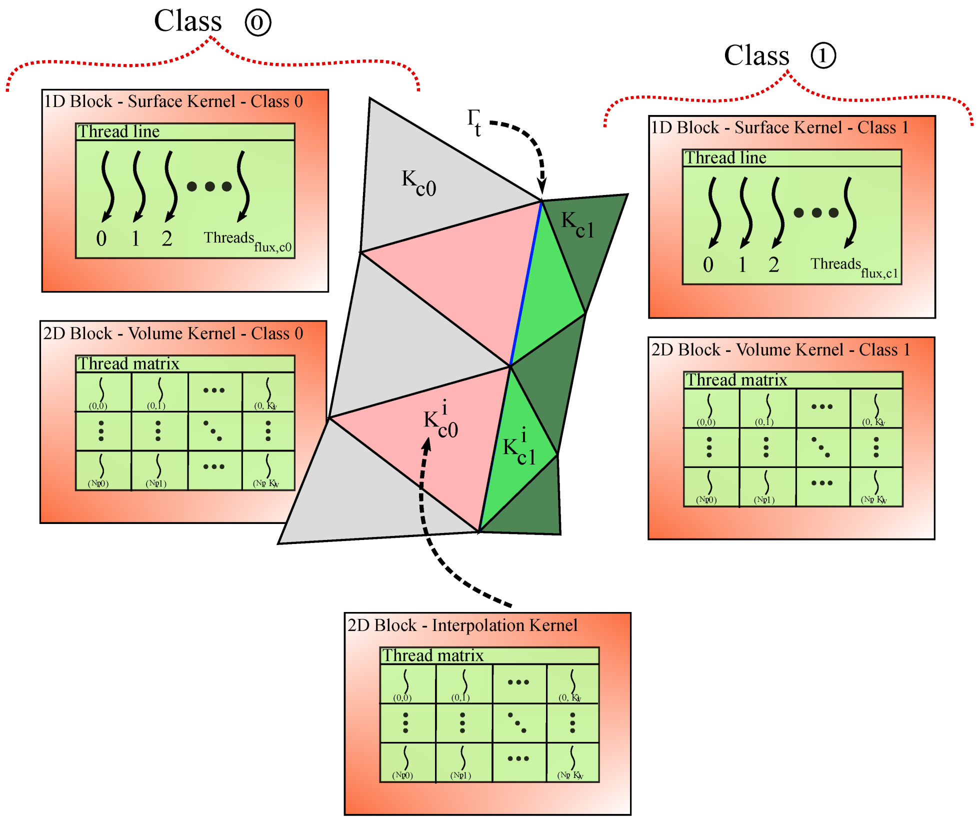

Class ⓪
Class ①
1D Block - Surface Kernel - Class 0
Thread line
0 1 2 Threads flux,c0
2D Block - Volume Kernel - Class 0
Thread matrix
1D Block - Surface Kernel - Class 1
Thread line
0 1 2 Threads flux,c1
2D Block - Volume Kernel - Class 1
Thread matrix
Γt
Kc0
Kc1
Kc0 i
Kc1 i
2D Block - Interpolation Kernel
Thread matrix

Figure 21 – DOF distributions used into the CUDA kernels

Algorithm VI: GPU-DGTD-LTS method

```
1  procedure PAR_LTS_MAXWELL($\mathbf{q}$,$G_V$,$G_S$,LIFT,$\mathcal{D}_m$,$\mathbf{Flux}_{ind}$)
2      Extract class(0) information from inputs    //Calculate $\mathbf{q0}$,$G_{V0}$,$G_{S0}$,$\mathbf{Flux}_{ind0}$
3      Extract class(1) information from inputs    //Calculate $\mathbf{q1}$,$G_{V1}$,$G_{S1}$,$\mathbf{Flux}_{ind1}$
4      Initialize variables    // Create update matrices $\mathbf{P_{q0}}$, $rhs_{\mathbf{q0}}$ ,$\mathbf{P_{q1}}$, $rhs_{\mathbf{q1}}$
5      Calculate the time step values for class(0), $\Delta t_0$, and class(1), $\Delta t_1$
6      Copy data from the CPU to GPU
7      Define number of time steps $Nts$    // Calculated using $\Delta t_0$
8      for $k = 0$ until $Nts$ do    // time loop
9          for $l = 0$ until $2$ do    // RK3 for class(0)
10             <surface_integral_Kernel_C0>    // Calculate $\mathbf{P_{q0}}$
11             <volume_integral_Kernel_C0>    // Calculate $rhs_{\mathbf{q0}}$
12             <time_integration_Kernel_C0>    // Update $\mathbf{q0}$
13         endfor
14         Update $\mathbf{q}$ using $\mathbf{q0}$
15         Calculate the interpolating polynomial $\chi$ to advance in class (1)
16         for $j = 0$ until $1$ do    // 2 refinement levels
17             for $l=0$ until $2$ do    // RK3 for class(1)
18                 <surface_integral_Kernel_C1>    // Calculate $\mathbf{P_{q1}}$
19                 <volume_integral_Kernel_C1>    // Calculate $rhs_{\mathbf{q1}}$
20                 <time_integration_Kernel_C1>    // Update $\mathbf{q1}$
21             endfor
22         endfor
23         Update $\mathbf{q}$ using $\mathbf{q1}$
24         Copy data from the GPU to CPU
25     endfor
26     return q
27 end
```

4.4 Chapter Conclusions

This chapter presents the formulation and implementation of a Local Time Stepping (LTS) strategy to improve the computational efficiency of the Discontinuous Galerkin Time-Domain (DGTD) method. It begins by exploring LTS schemes for Runge-Kutta (RK) methods, particularly focusing on a third-order LTS RK3 method that balances stability, accuracy, and performance. The LTS scheme enables different groups of mesh elements to evolve with distinct time steps based on their size, overcoming the limitations of global time step approaches that are constrained by the smallest elements.

The chapter details the mathematical foundation of the LTS RK3 method, including the use of third-order interpolating polynomials to preserve continuity across element interfaces. It further extends the scheme to the DGTD method, describing how elements are categorized into time step classes and how ghost cells are used for data exchange between classes. Finally, the chapter integrates the LTS method with GPU-based parallel computing, proposing an efficient hybrid scheme (GPU-DGTD-LTS). This implementation leverages CUDA kernels with customized Degrees of Freedom (DOF) distributions per element class and incorporates interpolation kernels to maintain third-order temporal accuracy. The approach significantly reduces the number of time steps for larger elements, offering substantial speedups while preserving the overall accuracy of the DGTD method.

5 Numerical Results

5.1 Numerical Tests on GPU

This section provides some two-dimensional numerical results to show the performance of the parallel-GPU DGTD method. Two wave propagation problems in homogeneous and heterogeneous media will be analyzed to evaluate the benefits of applying the DOF distribution proposed in Chapter 3. Additionally, a comparison between the global and shared memories in terms of time execution will be shown. These two-dimensional problems were chosen based on [80]. Results in this section were obtained by using the LSERK4, shown in subsection 2.5.4, as the time integration method. This method is widely used due to the low storage requirements and high order of approximation. However, the demand to compute 5 intermediate stages makes the LSERK4 less efficient in terms of computational time. Numerical tests were executed in the GPU NVIDIA GTX 1650-4GB and the CPU RYZEN 7 5800H with 16 GB of RAM. The computational domains used in the simulations were created using the GMSH mesh generator [81]. Table 6 shows the principal features of this graphic card. Finally, to evaluate the accuracy of the numerical scheme on the GPU, we compute the difference between the exact solution, let say E, and the approximate solution E_h using the L_2 norm given by.

$$e_r = \sqrt{\int_\Omega \frac{(E - E_h)^2}{E} d\Omega} \tag{5.1}$$

Table 6 – Graphic card features

Name	GTX 1650
Compute capability	7.5
Clock rate	1485 MHz
Global memory	4 GB
Bandwidth	128.1 GB/s
Number of multiprocessor	14
CUDA cores	896
Shared memory per multiprocessor	64 KB
Registers per multiprocessor	65536
Max threads per block	1024
Max threads per warp	32

5.1.1 Benchmark problem: metallic air-filled cavity

In the first test problem, we consider the metallic air-filled cavity also known as the rectangular waveguide. This problem consists of searching the determining modes in a rectangular waveguide by using elementary wave functions [82]. Generally, the modes in a rectangular waveguide are classified as TM to z and TE to z, this means, no H_z or no E_z components respectively. Considering the walls of the waveguide as PEC material, the tangential components of the electric field, E_z, must be vanished at the boundary of the domain. The computational domain Ω is a 2×2 m^2 square centered at the origin. The material inside the waveguide is considered vacuum $\varepsilon_r = \mu_r = 1$. In order to validate the GPU code, we use the exact time domain solution for the TMz mode which is given by:

$$H_x(x, y, t) = -\frac{\pi n}{\omega}\sin(m\pi x)\cos(n\pi y)\sin(\omega t) \tag{5.2a}$$

$$H_y(x, y, t) = \frac{\pi m}{\omega}\cos(m\pi x)\sin(n\pi y)\sin(\omega t) \tag{5.2b}$$

$$E_z(x, y, t) = \sin(m\pi x)\sin(n\pi y)\cos(\omega t) \tag{5.2c}$$

where ω is known as the resonance frequency given by:

$$\omega = \pi\sqrt{m^2 + n^2}, \quad (m, n) \geq 0 \tag{5.3}$$

Considering the solution at the time $t = 0$ and $m = n = 1$. The initial values of the field components are expressed as:

$$H_x(x, y, 0) = 0 \tag{5.4a}$$

$$H_y(x, y, 0) = 0 \tag{5.4b}$$

$$E_z(x, y, 0) = \sin(\pi x)\sin(\pi y) \tag{5.4c}$$

In this problem, two strategies were considered: the first one is the common serial CPU-DGTD with GTS (CPU-DGTD-GTS) method and the second one is the modified GPU-DGTD-GTS method. In the GPU strategy, the execution time of the CUDA program using both global and shared memory will be compared as mentioned in section 3.5.2. Simulations have been performed on a refined non-uniform grid whose characteristics are summarized in Table 7. Additionally, this table shows the global time step Δt which guarantees the stability of all elements in the mesh and the order of the polynomial basis functions N for the spatial integration.

Table 8 shows the numerical results of simulations in terms of global L_2 error, execution time, and speed-up of the parallel strategies compared with the serial implementation. As can be seen, there is a substantial difference in execution time when serial and

Table 7 – Mesh parameters used in the metallic cavity problem

N	3
Number of vertices	9340
Number of elements	18358
N_p	10
DOF	183580
Δt	2.56 ps

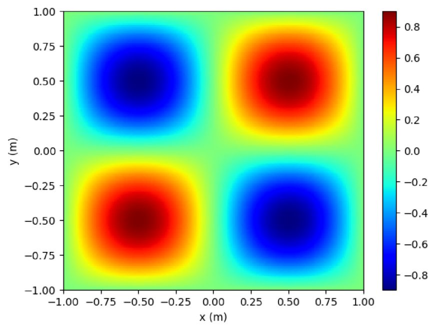
parallel schemes are compared. For example, the first GPU + Global memory strategy provides a computational gain of over 92% in terms of time spent when compared to its serial counterpart. On the other hand, the GPU + Shared memory implementation improves the computational gain even more, achieving a computational time reduction of more than 96%. This test confirms the advantage of shared memory in terms of latency compared to global memory, as discussed in Chapter 3. Shared memory is only used in the volume kernel, which calculates the spatial derivative of the electromagnetic fields through matrix-matrix operations.

It is worth noting that although the GPU parallel implementations involve a CPU-based preprocessing stage before kernel execution, its contribution to the total runtime is minimal. This preprocessing stage includes initializing all variables, copying data between the CPU and GPU, and calculating parameters required by the GPU kernels, as described in Algorithm 1. Specifically, the CPU preprocessing takes approximately 11.6 seconds for the CPU-only version and 13 seconds for the GPU-based implementations. When compared to the total execution times (3120 seconds for the CPU, and 221 and 130 seconds for the GPU implementations), these preprocessing times are negligible and do not significantly affect the reported speed-up values. In addition, the error presented by both GPU implementations is very similar compared to the serial value, with just a difference of 1×10^{-8}, as expected. All errors were calculated at time $t = 33.3$ ns. Note that the error for both GPU implementations is exactly the same, which is logical since all the calculations have been performed in the same way, with the only difference being the type of memory used to manipulate the data.

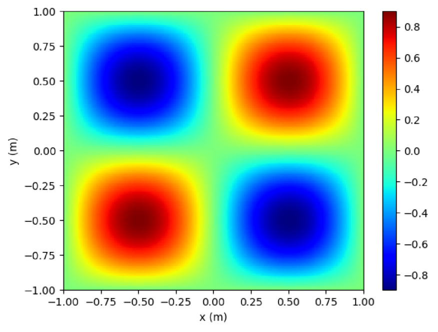
Figure 22 shows the field distribution for the TMz components E_z,H_x, and H_y after 12936 time steps. It can be seen that both CPU and GPU + Shared memory implementations provide similar results with a small difference in terms of global L_2 error according to Table 8. This small difference in accuracy can be considered negligible when compared with the computational gain, achieving a speed-up 24 times better than the CPU implementation. This first benchmark problem was considered with the aim of testing both GPU implementations and comparing which strategy is better than the other. According to the results, we can say that both GPU implementations are interesting in terms of accuracy and computational time reduction. However, it is clear that the use of shared memory provides better results in terms of computational efficiency.

Table 8 – Numerical results for the metallic air-filled cavity problem in terms of execution time, global error, and speed-up.

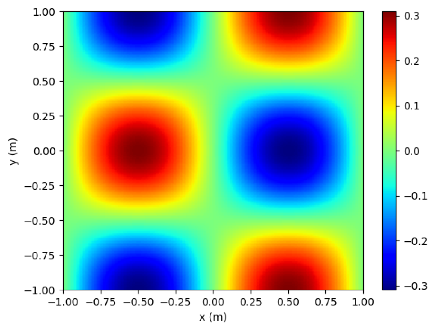
Method	Execution time (s)	Global L_2 error	Speed-up
CPU-DGTD-GTS	3120	2.12×10^{-6}	-
GPU-DGTD-GTS-Global memory	221	2.13×10^{-6}	14.1
GPU-DGTD-GTS-Shared memory	130	2.13×10^{-6}	24



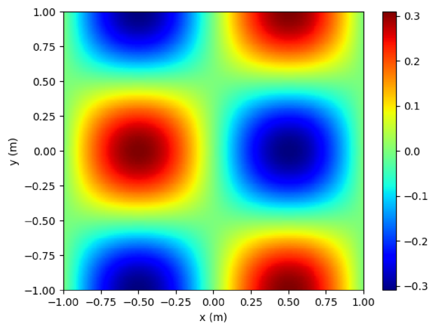
(a)



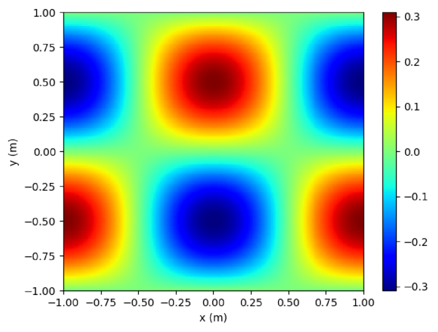
(b)



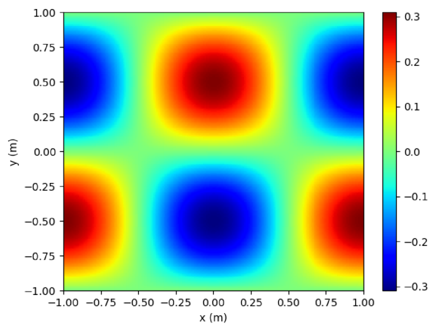
(c)



(d)



(e)



(f)

Figure 22 – Electromagnetic field components (E_z, H_x, H_y) for the TMz mode at $t = 33.3$ ns. Left column (a), (c), (e) shows the CPU implementation; right column (b), (d), (f) shows the GPU + shared memory version.

5.1.2 L-shaped photonic crystal guide

In order to test our GPU implementation on a complex two-dimensional problem, we chose the L-shaped photonic guide. This problem is considered of high complexity because involves a large number of scatterers and it is usually represented by large structures. The photonic crystal characterization is based on the construction of band diagrams. These structures can be studied if we know their symmetry properties, that is, we can understand the electromagnetic properties of the system only by knowing the symmetry properties. Basically, a bidimensional photonic crystal consists of a square lattice of parallel infinite dielectric rods in the air. This structure can be seen in Figure 23, where a is the lattice constant and r_a is the radius of the scatterers. Additionally, in Figure 23 we consider a system with continuous translational symmetry in the $\widehat{z}$ direction, that is, the system is invariant under any translation in a given direction. On the other hand, for the xy plane, the system has discrete translational symmetry, which means, the structure is invariant to a translation over a distance that is multiple of a certain length. According to Figure 23, the basic photonic crystal structure is filled with scatterers separately by a constant distance. However, it is necessary to insert the so-called defects in the crystal lattice to build some engineering applications. For instance, the photonic crystal in Figure 23 can be altered, removing or changing the characteristics of a dielectric column, creating waveguides that could be based on other devices such as logic gates [83]. Finally, applying some defects in the previous structure, we construct the same L-shaped photonic crystal guide used in [77], see Figure 24. The lattice constant $a = 0.57\mu m$ and the rods are assumed to have a circular cross-section of radius $r_a = 0.114\mu m$. The dielectric is chosen to have a refractive index $\eta \approx 3.4$ ($\varepsilon_r = 11.5$), appropriate for Silicon (Si). This crystal has a complete band gap for TM polarization between frequencies 0.35 and 0.42 ($\omega \boldsymbol{a}/2\pi c$). The incident pulse is placed in the left input of the waveguide and is given by:

$$E_z(x, y, t) = E_0 cos\left(\frac{\pi y}{d}\right) cos(2\pi f_m t) e^{-\left(\frac{t-t_0}{2\sigma}\right)^2} \tag{5.5}$$

where $d = 2(a - r_a)$ is the waveguide length, $\sigma_{BW} = 5 \times 10^{-14}$ s is a parameter which define the pulse bandwidth, $f_m = 2 \times 10^{14}$ Hz is the central frequency and $t_0 = 2.5 \times 10^{-13}$ s.

Table 9 – Mesh parameters used in the L-shaped photonic crystal guide problem

N	5
Number of vertices	6078
Number of elements	12107
N_p	21
DOF	254247
Δt	2.14 ps

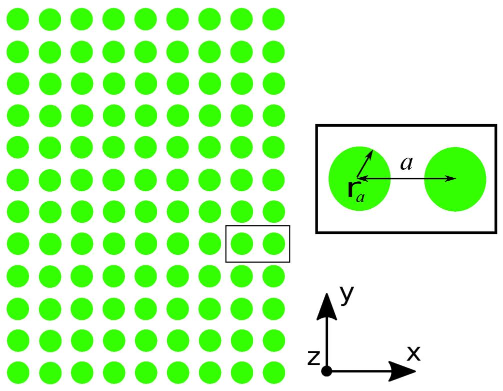


Figure 23 – Photonic crystal structure.

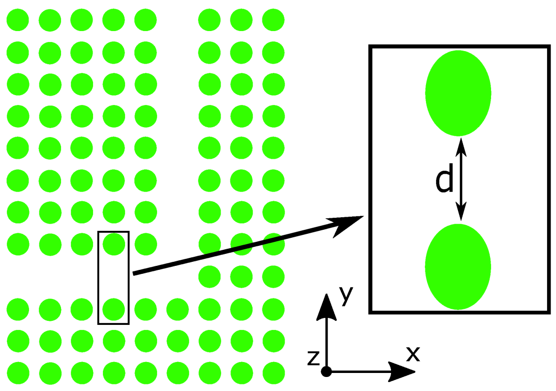


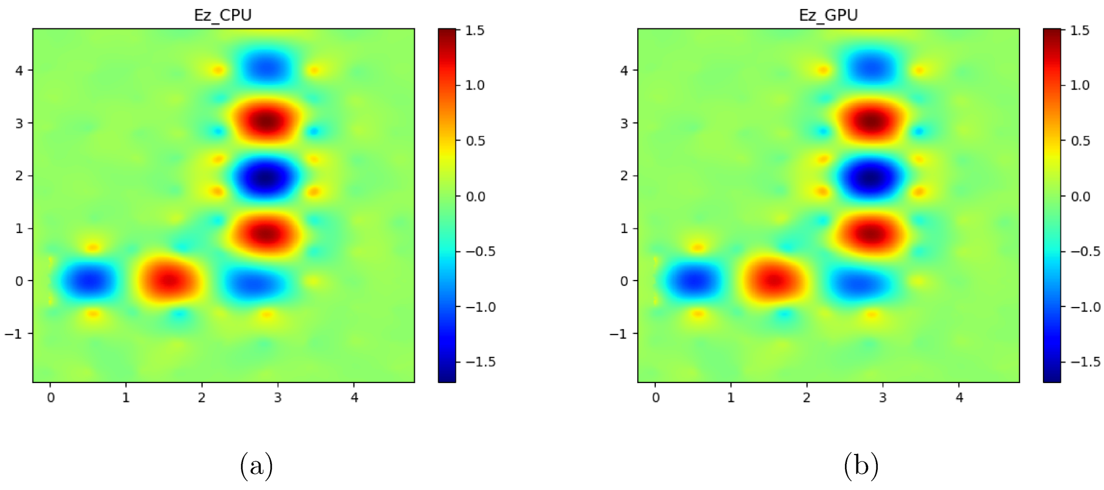
Figure 24 – L-shaped photonic crystal guide..

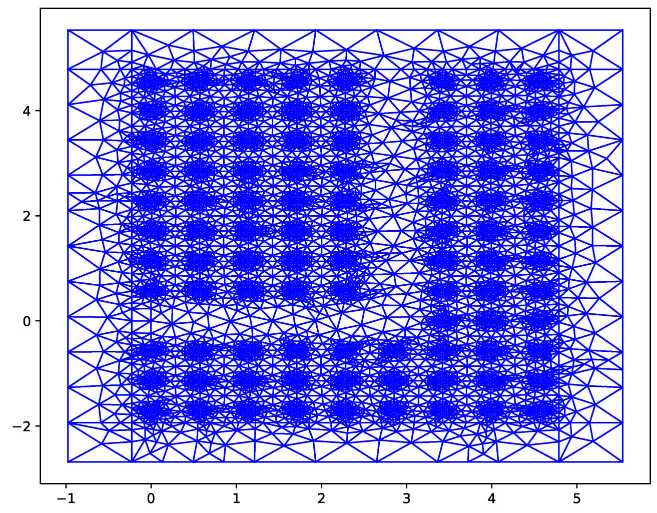
The simulations have been performed on a refined non-uniform grid whose characteristics are summarized in Table 9. This table also shows the global time step Δt and the order of the polynomial basis functions N. The field distribution of the E_z component inside the L-shaped waveguide using the CPU and GPU implementations is shown in Figure 25 (a) and (b), respectively. These results were obtained after 140198 time steps, and it can be seen that the GPU implementation provides very similar results compared to its CPU counterpart. Figure 25 (c) shows the triangular mesh used to represent the computational domain. The less refined region on the domain boundaries applies the PML boundary condition. In problems like this, where no analytical solution is available, comparing results between implementations is challenging. One effective approach is to evaluate the dispersion introduced by the waveguide on a propagated pulse. For this purpose, a field detector was placed at the waveguide output, and a Fast Fourier Transform (FFT) was computed from the time-domain data. Figure 26 presents a comparison between the FFT of the original pulse from Eq. 5.5 and those obtained from the CPU and GPU simulations after 249240 time steps. Table 10 summarizes the results in terms of dispersion and execution time. The frequency shift due to dispersion was less than 0.05% in both cases, confirming minimal pulse distortion.

Although both implementations involve a CPU-based preprocessing stage prior to time stepping, its contribution to the total runtime is negligible. Specifically, preprocessing took approximately 25 seconds in the CPU-only version and 30 seconds in the GPU-based case, representing less than 0.03% and 1.3% of the total execution times, respectively. As the complexity of the problem increases, the relative weight of the preprocessing time becomes even more insignificant. These results therefore demonstrate that the preprocessing stage has a minimal impact on performance and, consequently, will not be considered in future discussions.

Table 10 – Mesh parameters used in the L-shaped photonic crystal guide problem

Method	Execution time (s)	Central frequency (Hz)	Speed-up
CPU-DGTD-GTS	96421	1.9993×10^{14}	-
GPU-DGTD-GTS	2354	1.9991×10^{14}	41





(c)

Figure 25 – Field distribution of component Ez inside the L-shaped waveguide at time t = 300 ns. (a) CPU and (b) GPU implementations, respectively. (c) Non-uniform triangular mesh used to represent the computational domain.

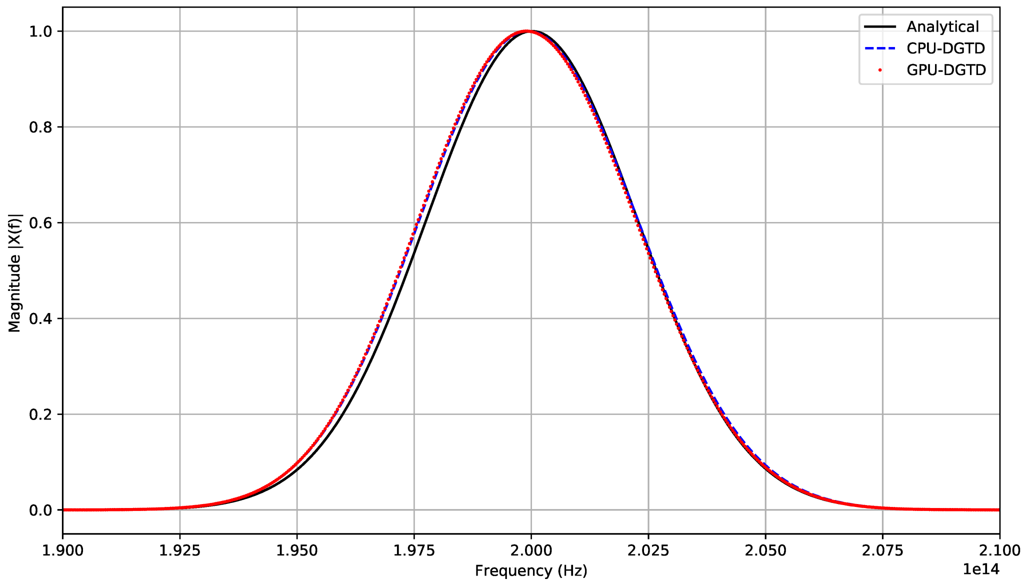


Figure 26 – FFT of the pulse propagated along the waveguide.

5.2 Numeric Tests with LTS

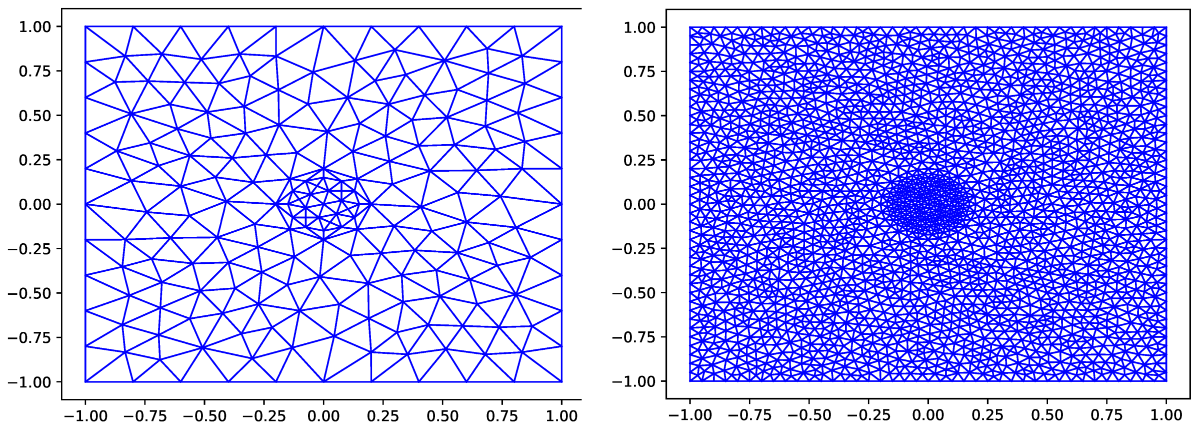
This section shows the results of the application of the LTS algorithm presented in section 4. As we mentioned before, this LTS algorithm is based on the RK3 method which demands the computation of three stages to advance the solution in time. Although this time integration method has a less order of approximation than the LSERK4, it has several features that make it suitable to apply an LTS strategy [33]. To validate the implementation of the DGTD method with the LTS scheme, two test cases have been selected. The first case revisits the benchmark problem of a metallic air-filled cavity discussed in the previous section, while the second case examines the scattering phenomenon caused by a cylindrical structure illuminated by a monochromatic plane wave.

5.2.1 Benchmark problem: metallic air-filled cavity

In this problem, we use a set of different unstructured triangular meshes whose characteristics are summarized in Table 11. These meshes were successively refined by a factor of 2 in the maximum edge size factor h. The meshes 1 and 2 are shown in Figure 27. Note that both meshes have a more refined region in the central part of the computational domain. These meshes were created with the purpose of dividing the elements into two different classes. $Class(0)$ contains the large elements and $Class(1)$ contains the smaller elements. Information about the number of elements in each class and the number of large interface elements is shown in Table 11. Once the elements in the mesh were organized into the two classes, it is time to calculate the local stable time step for each group of elements. The time step for both classes is calculated individually by applying Eq. 2.74 with $C = 1$. These time step values for both classes in each mesh are shown in Table 11. According to [33], this LTS algorithm provides a convergence rate of ≈ 3 when $N = 2$. Therefore, we set the order of the basis functions as two.

Table 11 – Meshes used to the convergence analysis in the metallic air-filled cavity problem

	Mesh 1	Mesh 2	Mesh 3	Mesh 4	Mesh 5
Max edge size - $h(m)$	0.2	0.1	0.05	0.025	0.0125
No. of elements	368	1352	4822	18358	73516
No. of vertices	205	717	2492	9340	37079
No. of elements - Class 0	312	1224	4356	16676	66936
No. of elements - Class 1	56	128	466	1682	6580
No. of large interface elements	16	28	32	64	124
Time step - Class 0 (ps)	41.52	21.65	11.12	5.68	2.78
Time step - Class 1 (ps)	20.76	10.83	5.56	2.84	1.39



(a) (b)

Figure 27 – Unstructured triangular meshes. (a) $h = 0.2$, (b) $h = 0.05$.

The objective of this test is to examine the convergence rate for the LTS RK3 algorithm and compare it to the standard RK3 counterpart without LTS. These results also give us a preliminary idea of the potential speed-up that can be achieved by using the LTS algorithm. Comparisons in terms of the global error using the L_2 norm and the convergence rate between the CPU-DGTD-GTS and CPU-DGTD-LTS methods can be seen in Figure 28. All errors were calculated at time t=33.333ns.

Figure 28 illustrates that the global L_2 error for the CPU-DGTD-GTS method decreases consistently as the domain is refined, achieving a convergence rate close to 3, as expected. Similarly, the CPU-DGTD-LTS method exhibits comparable behavior, confirming that the third-order LTS scheme ensures proper continuity between element classes while maintaining the accuracy of the standard GTS approach. Figure 28 also

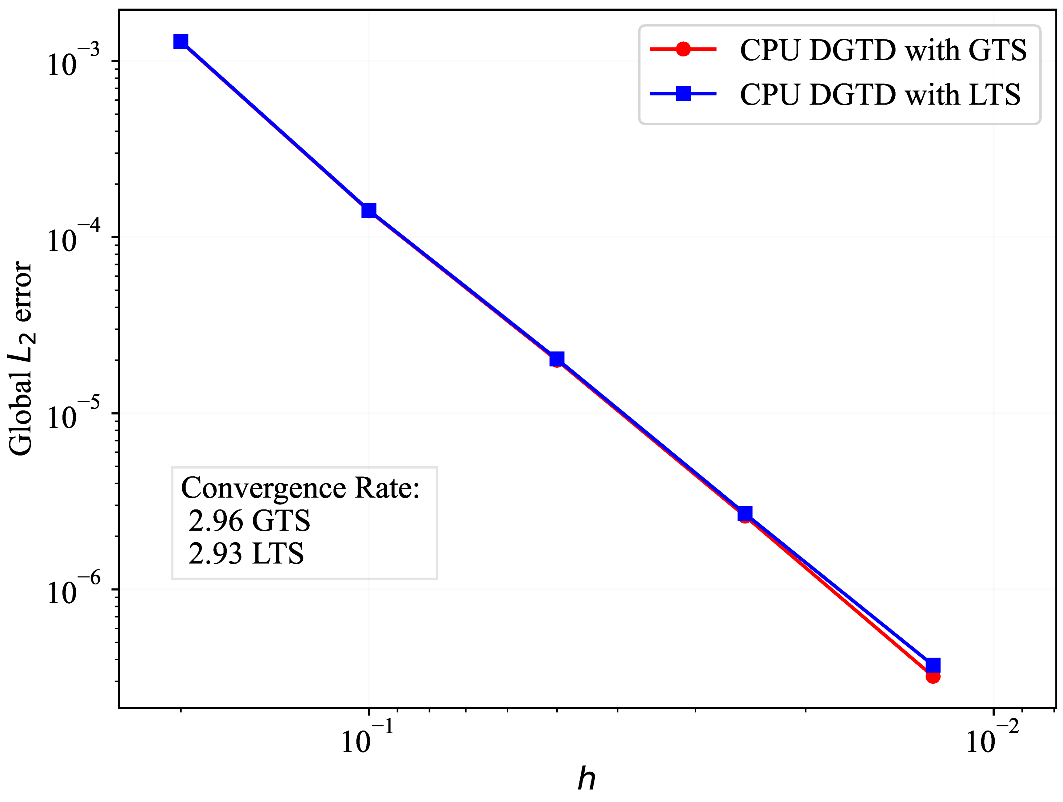


Figure 28 – Convergence plot for the metallic cavity problem in CPU implementations.

reveals a small difference between the global errors of the two approaches in the most refined mesh, resulting in a slight reduction in the convergence rate. This discrepancy is attributed to the third-order interpolation used in the LTS algorithm to couple the solutions between different element classes. To estimate the theoretical maximum computational gain from applying the LTS algorithm, we selected the most refined mesh, referred to as mesh 5. In this mesh, 6580 elements advance with the smallest time step, $\Delta t/2$, while 66936 elements advance with the largest time step, Δt. Completing one time cycle of size Δt in the LTS scheme requires $(6580 \times 2) + 66936 = 80096$ element-time-steps. Conversely, for the global time step scheme to complete one time cycle of size Δt, it requires $(6580 + 66936) \times 2 = 147032$ element-time-steps, as all elements advance with the smallest time step. The ratio of these numbers is $147032/80096 \approx 1.8$. Thus, the theoretical maximum computational gain for applying the LTS method in this problem is represented by this ratio. For mesh 5, the execution times of the CPU-DGTD-GTS and CPU-DGTD-LTS implementations were 1170.8 s and 712.51 s, respectively, yielding a ratio of 1.64. This result is close to the optimal theoretical value of 1.8.

5.2.2 Scattering by a PEC coated circular cylinder

To validate the implementation of the third-order LTS scheme with third-order interpolation in a more complex and multi-scale electromagnetic problem, the scattering by a PEC coated cylinder is analyzed [6]. The geometry of the problem is illustrated in Fig. 29, which shows a cross-sectional view of a coated PEC cylinder assumed to be infinitely long in the $\widehat{z}$ direction. The computational domain Ω is defined as a square with side length $\Omega_a = 5\,m$, centered at $(0, 0)$. A PML boundary condition with a thickness of $0.5\,m$ is applied in the $\widehat{x}$ and $\widehat{y}$ directions, resulting in an effective computational domain of $4 \times 4\; m^2$. The region outside the cylinder is assumed to be vacuum, characterized by $\mu_{r_1} = \varepsilon_{r_1} = 1$. The coating material is considered linear and isotropic, with a relative permittivity of $\varepsilon_{r_2} = 3$ and a relative permeability of $\mu_{r_2} = 1$. The internal PEC cylinder has a radius of $r_1 = 0.5\,m$, while the outer coated cylinder has a radius of $r_2 = 0.7\,m$. The incident plane wave is introduced using the Huygens principle and the TF/SF formulation [84]. The waveform of the incident plane wave is defined as follows:

$$g(t) = \cos(2\pi f t)\exp(-t^2/\tau^2) \tag{5.6}$$

where $f = 300$ MHz is the central frequency, and $\tau = 0.33\,ns$ is a time constant. The incident plane wave is chosen to have TMz polarization and propagation in the $\widehat{x}$ direction.

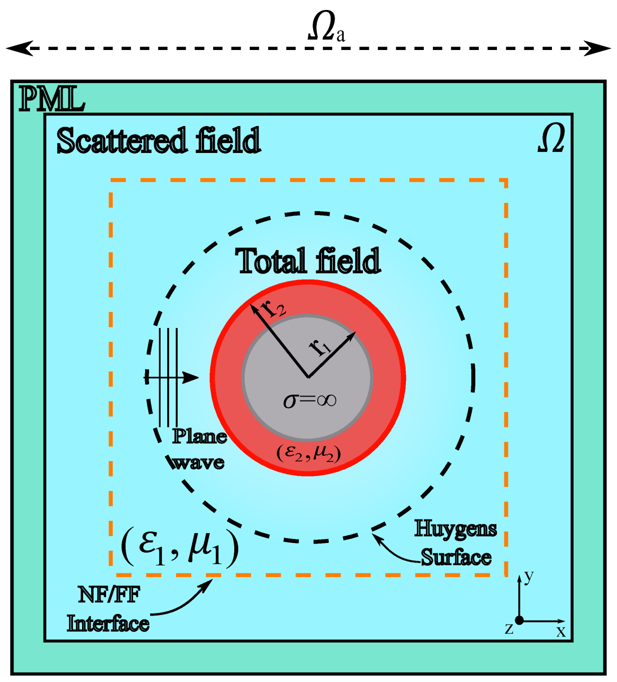
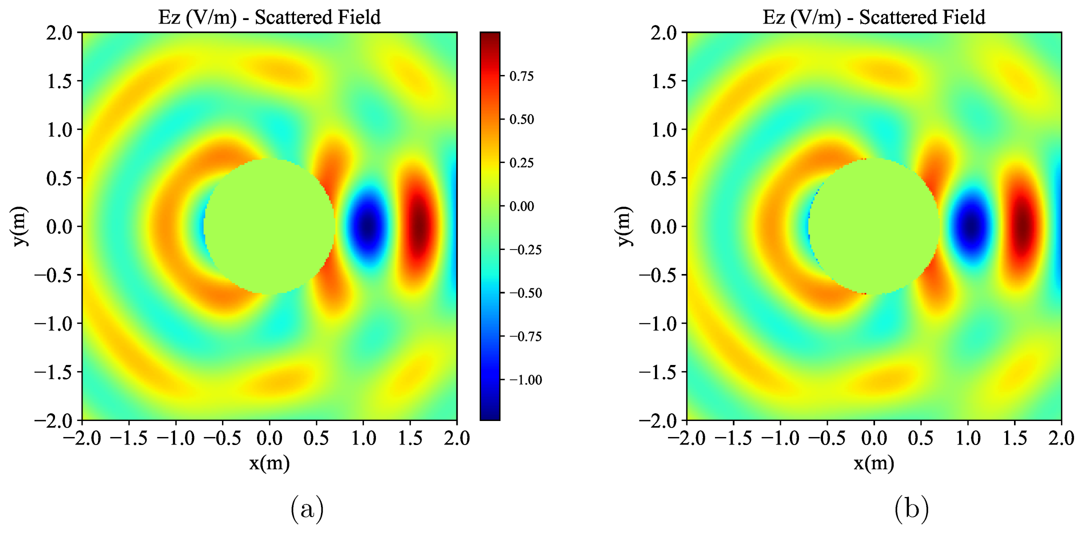


Figure 29 – Uniform plane wave illuminating a PEC coated circular cylinder.

The computational domain is discretized using a triangular mesh consisting of 32748 elements with a second-degree polynomial basis, resulting in 196488 DOF for each field component. The time-stepping scheme employs three different classes: $Class(0)$,

$Class(1)$, and $Class(2)$, which contain 84%, 10.1%, and 5.9% of the elements, respectively. The time step values used in the marching time process are $\Delta t_0 = 5.44$ ps, $\Delta t_1 = 2.72$ ps, and $\Delta t_2 = 1.36$ ps. These values were calculated using the Eq. 2.74 with $C = 1$. The relatively small proportion of elements requiring the smallest time step makes the LTS scheme particularly advantageous for this problem. The simulation was performed over a total duration of 66.66 ns.



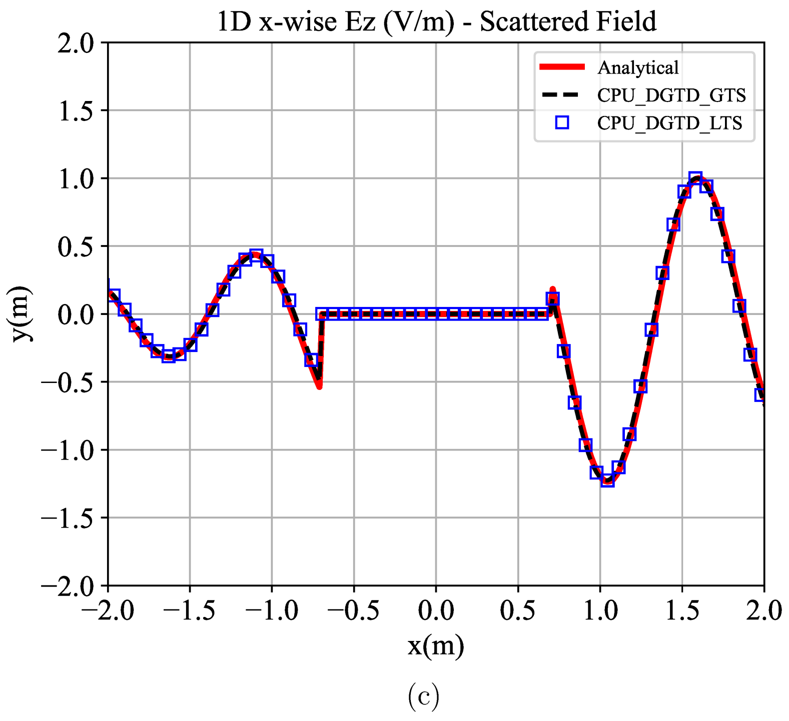


Figure 30 – Component Ez of the scattered field by a PEC coated cylinder. (a) GTS and (b) LTS implementations. (c) 1D x-wise distribution of Ez along $y = 0$.

To verify the accuracy of the CPU-DGTD-LTS method, both near- and far-field quantities were analyzed. Initially, the scattered field distribution of the E_z component was calculated using both the GTS and LTS approaches and compared with the analytical solution provided in [6]. This comparison is illustrated in Fig. 30, which also presents the x-wise distribution of the scattered E_z field at $y = 0$. The figure confirms that both numerical approaches yield results that closely match the analytical solution. Additionally, the bistatic Radar Cross Section (RCS) was computed and compared with both the analytical solution and the CPU-DGTD-GTS method, as shown in Fig. 31. The RCS was obtained using the Near-Field/Far-Field (NF/FF) transformation at 300 MHz. The results demonstrate that the CPU-DGTD-LTS method delivers accurate outcomes comparable to the analytical reference and the GTS approach.

Execution time, relative error of the E_z component, RCS, and speed-up results are summarized in Table 12. The LTS approach exhibits a notable performance gain, reducing the execution time by nearly 52% compared to the CPU-DGTD-GTS method. As with the previous case, the CPU-DGTD-LTS method introduces a slight loss in precision relative to the GTS counterpart, but it maintains the same order of accuracy.

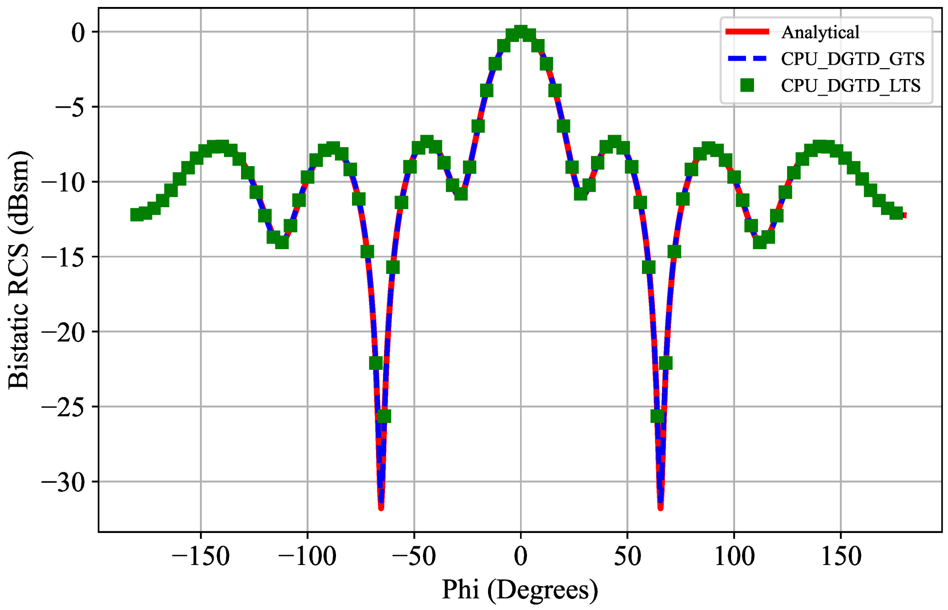


Figure 31 – RCS for TMz polarization for a PEC coated cylinder.

Table 12 – Results for the scattering by a PEC coated problem.

Method	Execution time (s)	RCS_{error}	Ez_{error}	Speed-up
CPU-DGTD-GTS	9134	0.048	0.033	-
CPU-DGTD-LTS	4402	0.052	0.034	2.1

5.3 Numerical Tests Combining GPU and LTS

This section aims to demonstrate the correct implementation of our proposed parallel GPU-DGTD with LTS (GPU-DGTD-LTS) method, described in section 4. This new approach, which forms the core contribution of this thesis, leverages the high bandwidth and low latency of GPUs combined with the efficient time-stepping distribution provided by the LTS method. This integration ensures more efficient time progression and, consequently, enhances the overall computational efficiency of the method. The numerical results in this section aim to demonstrate two key aspects of the implementation. First, the proposed GPU-DGTD-LTS method is evaluated as an accurate and efficient numerical technique for solving multi-scale electromagnetic problems. Second, the impact of the interpolation order used to ensure continuity between classes in the LTS algorithm is highlighted as crucial for achieving accurate results. To validate these aspects, two test problems are analyzed, comparing the validated GPU-DGTD-GTS method with the proposed GPU-DGTD-LTS approach. The first test revisits the benchmark problem of a metallic air-filled cavity to verify accuracy and convergence rate, providing a straightforward case for evaluating the performance of the GPU-DGTD-LTS implementation. The second test focuses on the scattering by a conducting sphere, a more complex problem involving a multi-scale discretization. This case is used to assess the robustness of the proposed method in handling intricate electromagnetic scenarios.

5.3.1 Benchmark problem: metallic air-filled cavity

In this case, the metallic air-filled cavity problem have been solved by using the same setup as in the previous subsection. That is, we use the same set of five successively refined meshes described in Table 11 and the elements have been organized into two classes. The simulation was carried out during 33.33 ns and the order of the basis function is two. In the GPU thread distribution, this order impacts the number of threads due to the variables N_{fp} and N_p. As previously mentioned in Chapter 3, the number of threads remains constant while the number of blocks depends on the number of elements. In this case, the values K_f and K_v are set to 14 and 21, respectively. Thus, the number of threads in the surface kernel is $N_{faces} \times N_{fp} \times K_f = 3 \times 3 \times 14 = 126$, and in the volume kernel, it is $N_p \times K_v = 6 \times 21 = 126$.

Comparisons in terms of the global error using the L_2 norm and the convergence rate between the GPU-DGTD-GTS and GPU-DGTD-LTS methods can be seen in Figure 32. This figure also illustrates the results of the LTS scheme using both linear and cubic interpolation in order to show the impact of high-order interpolation on the accuracy of the method. As can be seen in Figure 32, the GPU-DGTD-GTS method ensures a convergence rate of 2.84, as expected from the RK3 method. On the other hand, the results of our proposal with linear and cubic interpolation show a small loss of precision and consequently a small loss in the convergence rate, obtaining values of 2.69 and 2.71, respectively, when compared with standard GTS. This loss of precision was expected due to the error introduced by the first-order and third-order interpolations. Note that even with a loss of precision, the approximated solution maintains the same order of precision for each mesh. This test shows that the order of the interpolation impacts directly the accuracy of the LTS scheme but does not affect the convergence rate. The execution time in mesh 5 for the GTS, LTS with linear interpolation, and LTS with cubic interpolation implementations was 118.701 s, 80.66 s, and 81.05 s, respectively. Note that the difference between the execution time for the LTS with linear and cubic interpolation was minimal, demonstrating that cubic interpolation is the best choice in terms of accuracy and efficiency. The acceleration in comparing using GTS and LTS with cubic interpolation strategies was 1.46, which is close to the optimal value of 1.8, described in the previous subsection. The difference between these ratios is caused by the increase in the data exchange between the CPU and the GPU in implementing the LTS algorithm and the calculation of the interpolating polynomials.

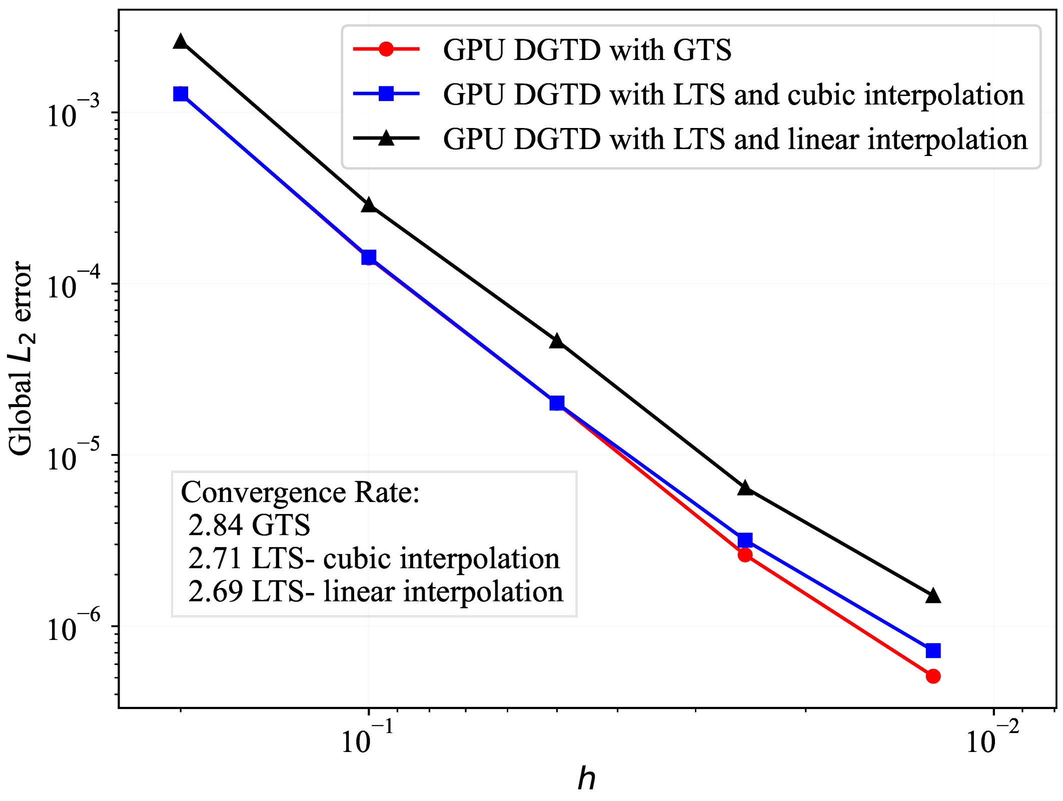


Figure 32 – Convergence plot for the metallic cavity problem in GPU implementations.

5.3.2 Scattering by a PEC sphere

This test problem is studied with the purpose of evaluating the accuracy and efficiency of the proposed GPU-DGTD-LTS method when dealing with near-field and far-field quantities in an electromagnetic scattering problem. In this case, a PEC sphere inside a vacuum background is illuminated by an x-polarized incident plane wave propagating in the $\widehat{z}$ direction. The computational domain Ω is composed of a $r_1 = 0.5$ m radius PEC sphere bounded by a cube with a side length of $\Omega_a = 3$ m centered at the point (0,0,0). Figure 33 shows a cross-sectional view of the geometry in the xz plane when $y = 0$. The problem is truncated using a PML absorbing boundary condition with a thickness of 0.5 m in the $\widehat{x}$, $\widehat{y}$ and $\widehat{z}$ directions. The incident x-polarized plane wave propagating in the $\widehat{z}$ direction was inserted by using the TF/SF formulation and was modeled using the same expression as in Eq. 5.6 with $f = 300$ MHz and $\tau = 0.33$ ns.

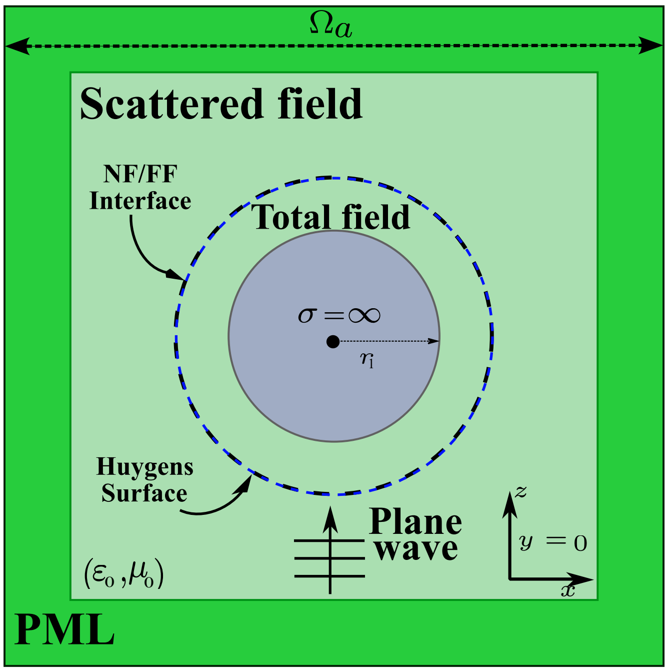


Figure 33 – Cross-sectional view of the xz plane for the PEC sphere problem.

The tetrahedral mesh is composed of 116249 elements of a second-degree polynomial order. This mesh was generated considering a maximum edge size factor $h = \lambda/20$, where λ is the wavelength, which depends on the speed of light and the central frequency. In this test problem, three different classes were used for the time stepping scheme. The number of elements in $Class(0)$, $Class(1)$, and $Class(2)$ is 54549, 4465, and 57235, respectively. The time step values $\Delta t_0 = 7.8$ ps, $\Delta t_1 = 3.9$ ps, and $\Delta t_2 = 1.95$ ps were calculated for each class considering Eq. 2.74 with $C = 1$. Finally, the simulation was performed for 17 ns.

To illustrate the importance of high-order interpolations in an LTS scheme, the **E** field was sampled at a critical point in the domain and compared to the standard GTS implementation. This critical point was located at the interface between two neighboring classes, where the interpolations were performed. Figures 34, 35 and 36 show the **E** field components x, y, and z over the simulation time. As shown, first-order interpolation introduces oscillations and amplitude errors, which are mitigated by third-order interpolation. Table 13 presents the maximum error values produced by the LTS method with first-order and third-order interpolation compared to those of the standard GTS approach. These results indicate that even in the worst case, the LTS method with a third-order interpolation scheme can closely match the standard GTS solution, with a maximum error of 2.4×10^{-3} in the Ex component. In contrast, the first-order LTS method, while providing a reasonable approximation to the GTS solution, exhibits reduced precision, resulting in a maximum error of 1.3×10^{-1} in the Ez component. This difference arises because the third-order scheme incorporates additional values to enhance the continuity between classes, leading to a smoother solution with fewer errors. These preliminary results demonstrate that high-order interpolation in an LTS scheme is not just a formality but a necessity, as lower-order interpolation can compromise the accuracy of the method.

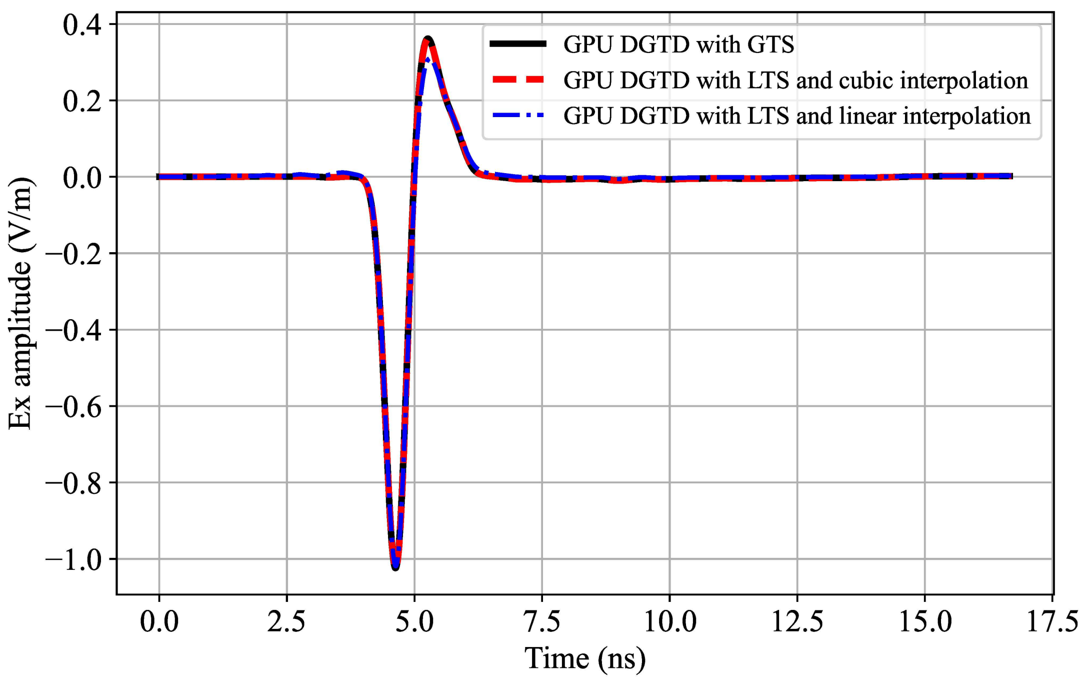


Figure 34 – Ex component for both GTS and LTS implementations at point (0.1,0,0.59).

Table 13 – Maximum error values of E field components comparing the LTS approach to the standard GTS method.

Method	Ex_{error}	Ey_{error}	Ez_{error}
GPU-DGTD-LTS-Cubic	2.4×10^{-3}	3.6×10^{-4}	3.8×10^{-5}
GPU-DGTD-LTS-Linear	6.5×10^{-2}	7.9×10^{-3}	1.3×10^{-1}

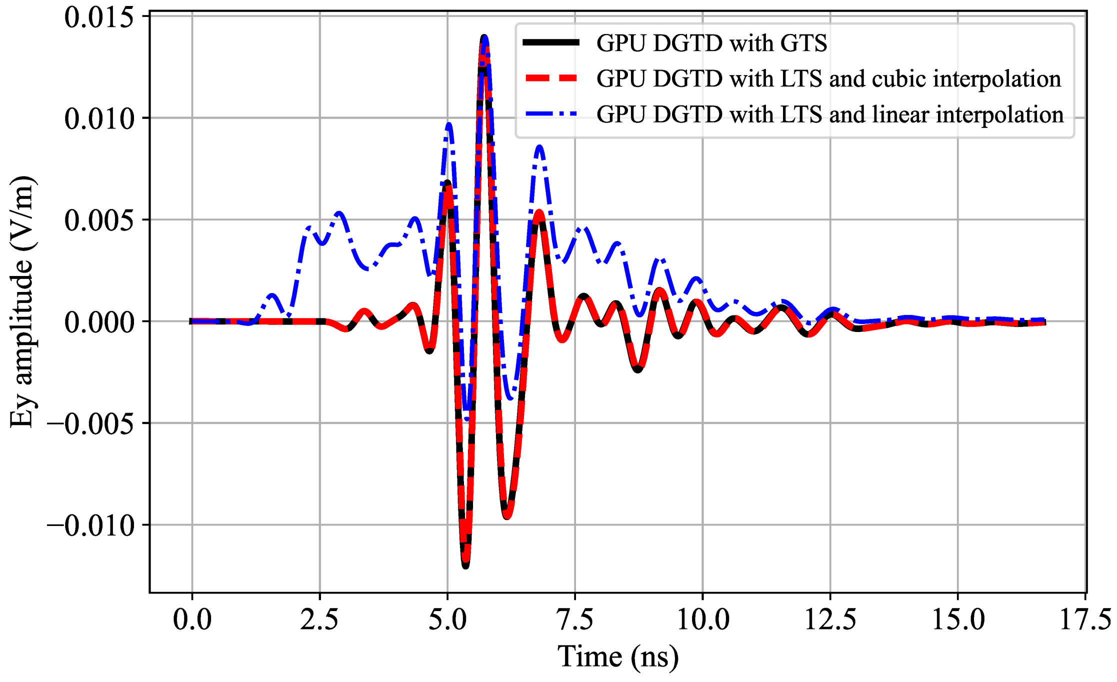


Figure 35 – Ey component for both GTS and LTS implementations at point (0.1,0,0.59).

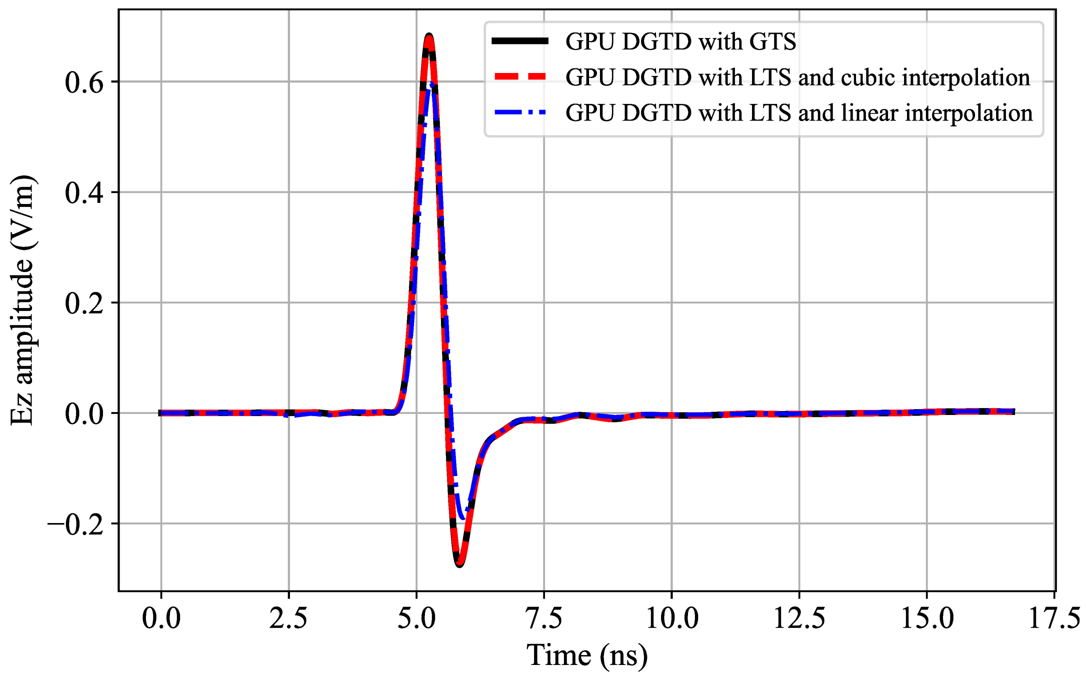


Figure 36 – Ez component for both GTS and LTS implementations at point (0.1,0,0.59).

The bistatic RCS is a crucial far-field parameter in electromagnetic scattering problems and is frequently employed for the verification of numerical methods. In addition, it is essential to understand the impact of first-order and cubic interpolations on far-field quantities. Consequently, the RCS was calculated and plotted in both the E-plane and H-plane, as depicted in Figures 37 and 38. The bistatic RCS is calculated by using the NF/FF formulation at 300 MHz. These results were compared with the analytical solution and the GPU-DGTD-GTS method. As illustrated in Figures 37 and 38, both the proposed GPU-DGTD-LTS method with cubic interpolation and the standard GPU-DGTD-GTS method demonstrate good agreement with the analytical solution. On the other hand, while the GPU-DGTD-LTS method with linear interpolation produces RCS values close to those of the analytical solution, amplitude errors are still present. This discrepancy is anticipated, as the far-field parameters are directly influenced by the near-field values. Table 14 summarizes the results in terms of the execution time, the relative error of the RCS, and the speed-up. As in the previous problem, the execution time difference between the LTS schemes with first-order and cubic interpolation is negligible. In this case, the time reduction for both LTS implementations was almost 60% when compared with the standard GPU-GTS case. Furthermore, the LTS scheme with cubic interpolation maintained the same order of precision as the standard case, proving to be the best choice for solving electromagnetic scattering problems.

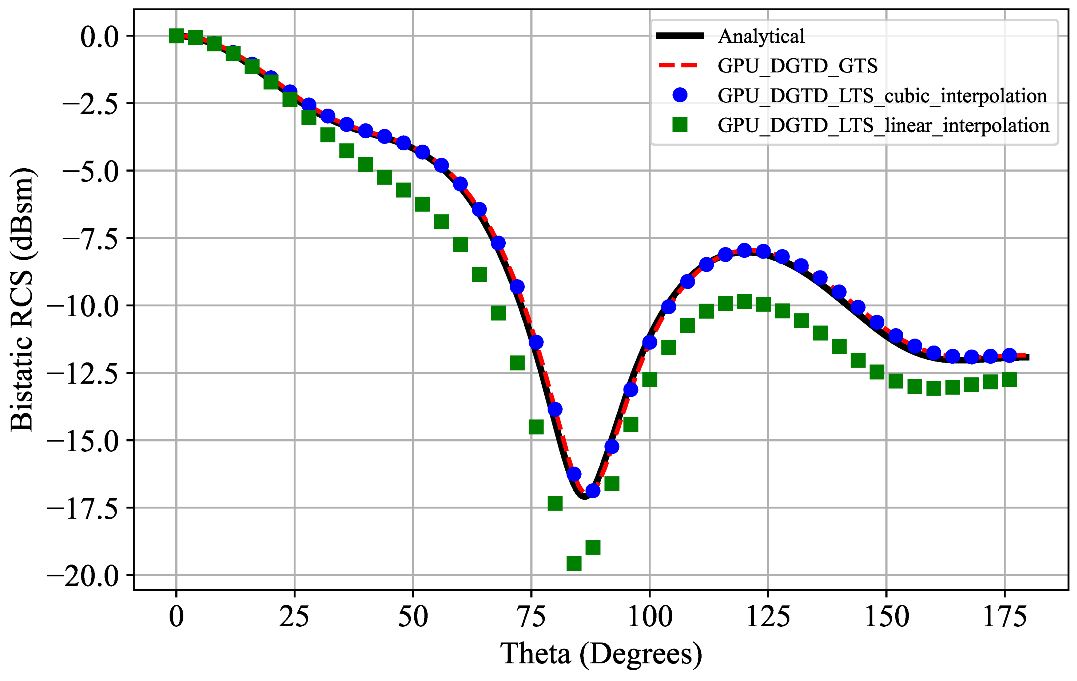


Figure 37 – The bistatic RCS for the PEC sphere problem in the E-plane ($\phi = 0$).

Table 14 – Results for the problem of scattering by a PEC sphere.

Method	Time (s)	E-Plane$_{error}$	H-Plane$_{error}$	Speed-Up
GPU-DGTD-GTS	822	0.0054	0.0025	-
GPU-DGTD-LTS-Cubic	339	0.0056	0.00259	2.42
GPU-DGTD-LTS-Linear	334	0.0271	0.042	2.46

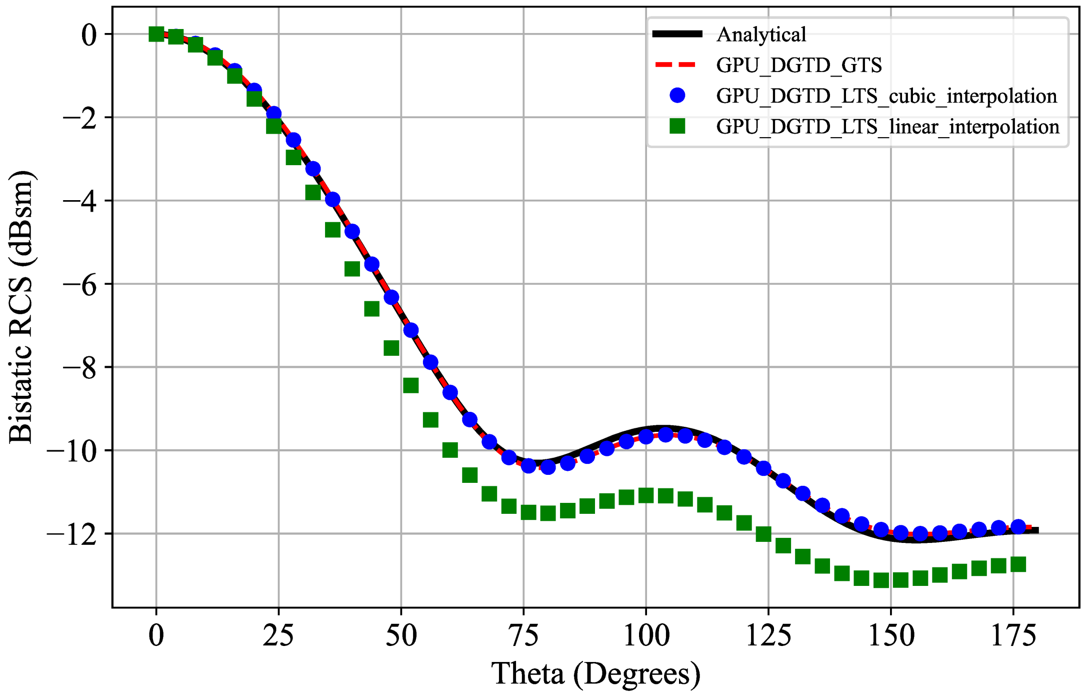


Figure 38 – The bistatic RCS for the PEC sphere problem in the H-plane ($\phi = 90$).

5.4 Application in Solving Complex Multi-scale Engineering Problems

This section concludes the numerical results chapter by presenting the application of the proposed GPU-DGTD-LTS method to solve complex multi-scale engineering problems. Two challenging numerical examples have been selected to evaluate the accuracy, efficiency, and robustness of the method when applied to realistic scenarios. The first example focuses on the scattering by a multilayer dielectric sphere. This problem is widely recognized as challenging for many numerical solvers due to its highly multi-scale nature, where different layers introduce variations in material properties and geometry that demand precise handling. By addressing this problem, the GPU-DGTD-LTS method demonstrates its capability to effectively manage the intricate interactions and achieve accurate results even in demanding cases. The second example involves calculating the input admittance of a monopole antenna. Antenna problems are particularly well-suited for multi-scale techniques, as highlighted in [63], because achieving accurate results requires extremely fine discretization near to the source feed location. This fine-scale resolution, combined with larger-scale domain characteristics, poses a significant challenge for traditional methods. The GPU-DGTD-LTS method, however, showcases its ability to handle these multi-scale requirements efficiently, providing reliable results while maintaining computational efficiency.

5.4.1 Scattering by a multilayer dielectric sphere

In order to show the performance of our proposal in more complex and multi-scale problems, we chose to study scattering by a multilayer dielectric sphere [6]. This problem is ideal for exploring the flexibility of the DGTD method in handling complex geometries and unstructured meshes. The multi-scale nature of this problem enabled us to leverage the proposed GPU-DGTD-LTS method with third-order interpolation, dividing the problem into several classes based on the size of the elements or the electromagnetic parameters of the media. It significantly improved the computational efficiency, especially in regions that required a fine spatial resolution. The problem consisted of four concentric spheres, with the innermost sphere modeled as a PEC material and the remaining spheres modeled as dielectric materials. The geometry of the problem is depicted in Figure 39, where a cross-sectional view of the xz plane can be seen when $y = 0$. The computational domain Ω is bounded by a cube with a side length of $\Omega_a = 3\,\text{m}$ centered at $(0, 0, 0)$. The PML boundary condition was used, with a thickness of $0.5\,\text{m}$ in all directions, that is, the real computational domain was $2^3\ \text{m}^3$. The region outside the multilayer sphere was assumed to be a vacuum, with $\mu_{r1} = \varepsilon_{r1} = 1$. The materials in the multilayer regions were assumed to be linear, isotropic, and non-magnetic, with relative permittivity of $\varepsilon_{r2} = 2$, $\varepsilon_{r3} = 3$, and $\varepsilon_{r4} = 4$. The radii of the spheres, from the innermost to the outermost, were $r_1 = 0.3\,\text{m}$, $r_2 = 0.4\,\text{m}$, $r_3 = 0.5\,\text{m}$, and $r_4 = 0.6\,\text{m}$, respectively. The incident x-polarized plane wave propagating in the $\widehat{z}$ direction was inserted by using the TF/SF formulation and was modeled using the same expression as in Eq. 5.6 with $f = 300$ MHz and $\tau = 0.33$ ns.

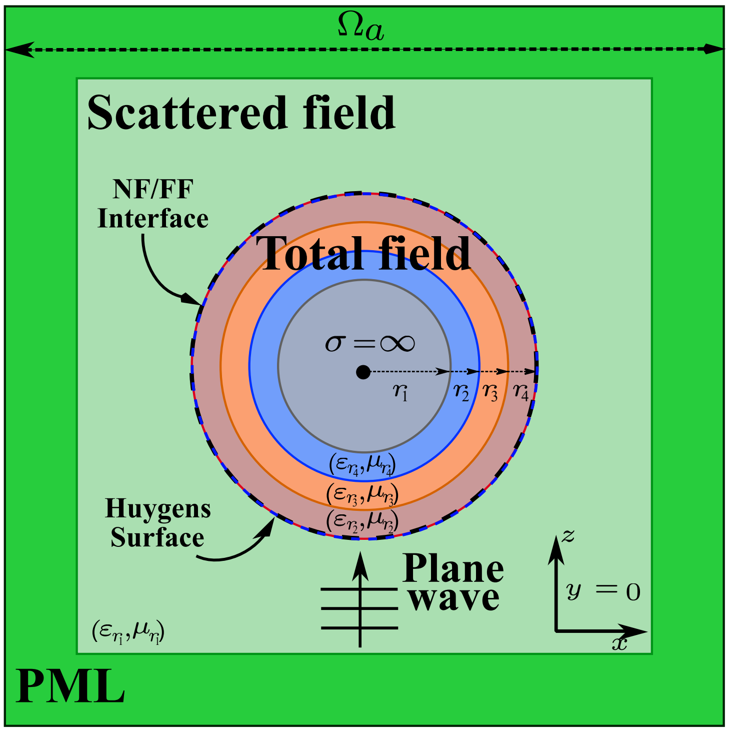


Figure 39 – Cross-sectional view of the xz plane for the multilayer sphere problem.

The computational domain consists of a tetrahedral mesh made of 193627 elements with a second-degree polynomial basis, which corresponds to more than 1.9 million DOF for each field component. This domain was built considering a maximum edge size factor $h = \lambda/20$ in each class. In this case, the time stepping scheme was used with four different classes. $Class(0)$, $Class(1)$, $Class(2)$ and $Class(3)$ contain 44.7%, 13.6%, 21.6%, and 20.1% of the elements, respectively. The time step values used in the time-marching process were defined as $\Delta t_0 = 5.28$ ps, $\Delta t_1 = 2.64$ ps, $\Delta t_2 = 1.32$ ps, and $\Delta t_3 = 0.66$ ps. These time step values were calculated using Eq. 2.74 with $C = (1/2)$. Finally, the simulation was carried out for 50 ns.

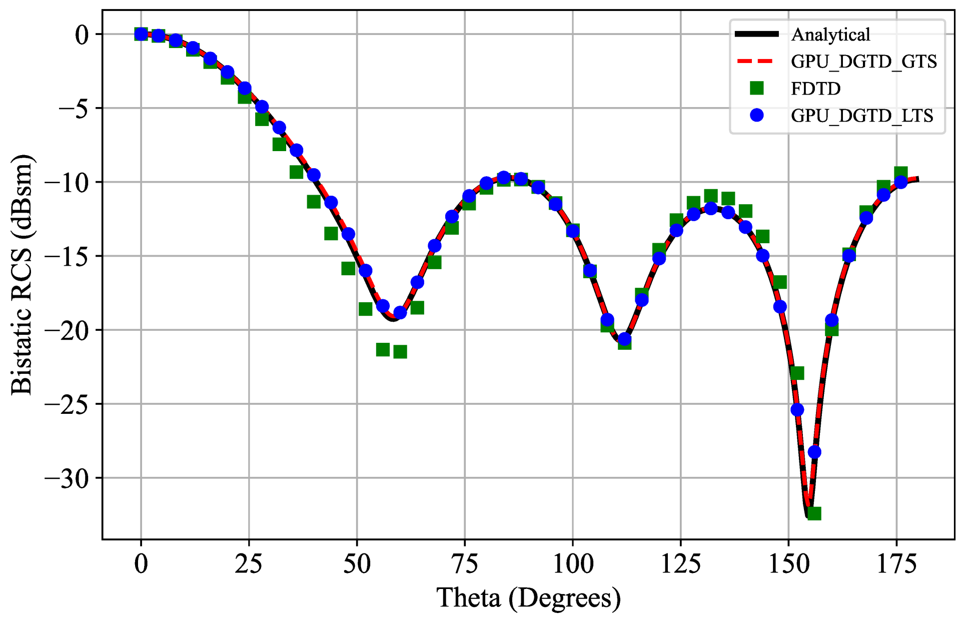


Figure 40 – Bistatic RCS of a multilayer sphere in the E-plane ($\phi = 0$).

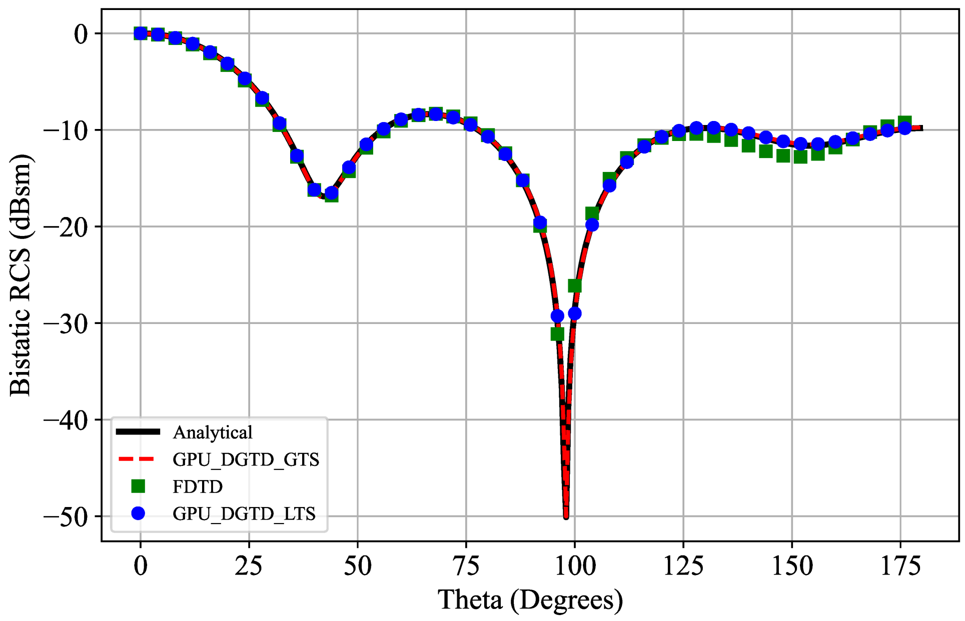


Figure 41 – Bistatic RCS of a multilayer sphere in the H-plane ($\phi = 90$).

As in the PEC sphere problem, the accuracy of the method was verified by calculating the RCS at 300 MHz in the E-plane and the H-plane. These results were compared with those of the analytical solution, the standard GPU-DGTD-GTS method, and the second-order FDTD method [58], as shown in Figures 40 and 41. The FDTD method was included in this analysis because its computational simplicity and accurate results make it a common choice for solving electromagnetic scattering problems in the time-domain. Figures 40 and 41 confirm that the proposed GPU-DGTD-LTS method with third-order interpolation provides excellent results when compared with the analytical solution and its GTS counterpart. On the other hand, although the FDTD results show good agreement with those of the analytical solution, there are small discrepancies, which may be related to the significant restriction of dealing with cube-based space partitioning making it difficult to accurately represent the curvatures in the problem. It is important to remark that the FDTD results were obtained under conditions similar to those of the DGTD case but using a fine grid discretization of $\Delta x = \Delta y = \Delta z = \lambda/100$. The results in terms of the execution time, the relative error of the RCS, and the speed-up for the DGTD and FDTD implementations are shown in Table 15. The results in terms of the execution time show a very interesting improvement for the LTS case, achieving a reduction of almost 78% when compared with the standard GPU-DGTD-GTS method. Similarly to the previous problem, the error values with the GPU-DGTD-LTS method show a slight loss of precision compared to those of its GTS counterpart while maintaining the same order of accuracy. Furthermore, the error values for the FDTD implementation are almost twice as large as those for the DGTD. This can be attributed to the inability to accurately represent the curvatures in the problem even if a fine grid model is used. Finally, the execution time and speed-up results for the FDTD method were not included in this table because there is no direct comparison between them and the DGTD algorithms described.

Table 15 – Results for the scattering by a multilayer sphere problem.

Method	Time (s)	E-Plane$_{error}$	H-Plane$_{error}$	Speed-Up
GPU-DGTD-GTS	18780	0.0049	0.0044	-
GPU-DGTD-LTS	4150	0.0051	0.0047	4.52
FDTD	-	0.011	0.0081	-

5.4.2 Quarter-wave monopole antenna

This problem evaluates the three antenna feed models described in subsection 2.3.4 by comparing their accuracy and simulation time in addressing a quarter-wave monopole antenna problem. The comparison involves determining the input admittance using time-domain waveforms of sampled voltage and current [58]. The FFT is then applied to these waveforms to compute input admittance across a broad frequency range. For all models, the mesh was discretized with a maximum edge size of $h = \lambda/20$, and a refinement factor of two was applied near the feed and antenna regions to ensure precise geometric representation. A second-order polynomial basis was employed in all simulations, and the simulations were run until the waveform amplitudes stabilized at values near 10^{-7}. Finally, the excitation's time-dependent voltage function was modeled using a Gaussian waveform, expressed as:

$$V^{inc}(t) = \exp((t - t_0)/\sigma)^2 \tag{5.7}$$

where $t_0 = 0.26\,ns$ is a time delay applied to the Gaussian pulse and $\sigma = 0.083\,ns$ is a time constant.

The geometry of the quarter-wave monopole antenna consists of a PEC cylinder centered at the origin, with a radius of $l_a = 3.75$ mm and a height of 180 mm. The antenna is mounted on a ground plane and oriented along the $\widehat{z}$ axis. To truncate the computational domain, PML absorbing boundary conditions are applied. For the delta-gap feed model, the surface magnetic current defined in Eq. 2.33 is applied to a cylindrical gap region of length $\Delta = 3.6$ mm, located between the ground plane and the antenna, as illustrated in Figure 3. This gap is considered part of the monopole's total height. By imposing the surface magnetic current across the gap, the input admittance can be calculated by dividing the total voltage across the gap by the current flowing through it. Since the delta-gap feed model is implemented as a hard source, the total voltage is determined using the same incident voltage expression provided in Eq. 5.7. The computational domain is discretized using a tetrahedral mesh consisting of 41495 elements organized in two classes. $Class(0)$ and $Class(1)$ contain 78.4% and 21.6% of the elements, respectively. The time step values used in the time-marching process were defined as $\Delta t_0 = 1.24$ ps and $\Delta t_1 = 0.62$ ps These time step values were calculated using Eq. 2.74 with $C = (1/2)$. Figure 42 presents the time-domain waveforms for the total voltage and current sampled during the simulation, along with the computed input admittance of the monopole antenna for a frequency range of 0 to 2.5 GHz.

The simulation was conducted for approximately $34\tau_a$, where τ_a represents the time it takes for an electromagnetic wave to traverse the monopole length in air. As shown in Figure 42b, the sampled current requires this duration for its energy to decay nearly to zero, as reflected waves are neglected in this feed model. While this approach results in a slow transient response, thereby extending the simulation time, the delta-gap feed model remains popular due to its simplicity and reliable performance [63]. This is evident in the strong agreement between the input admittance results and the reference values obtained from the frequency-domain FEM solver in ANSYS HFSS, as depicted in Figure 42c.

The magnetic frill approach, on the other hand, is implemented by applying the surface magnetic current defined in Eq. 2.35 to a coaxial aperture with inner and

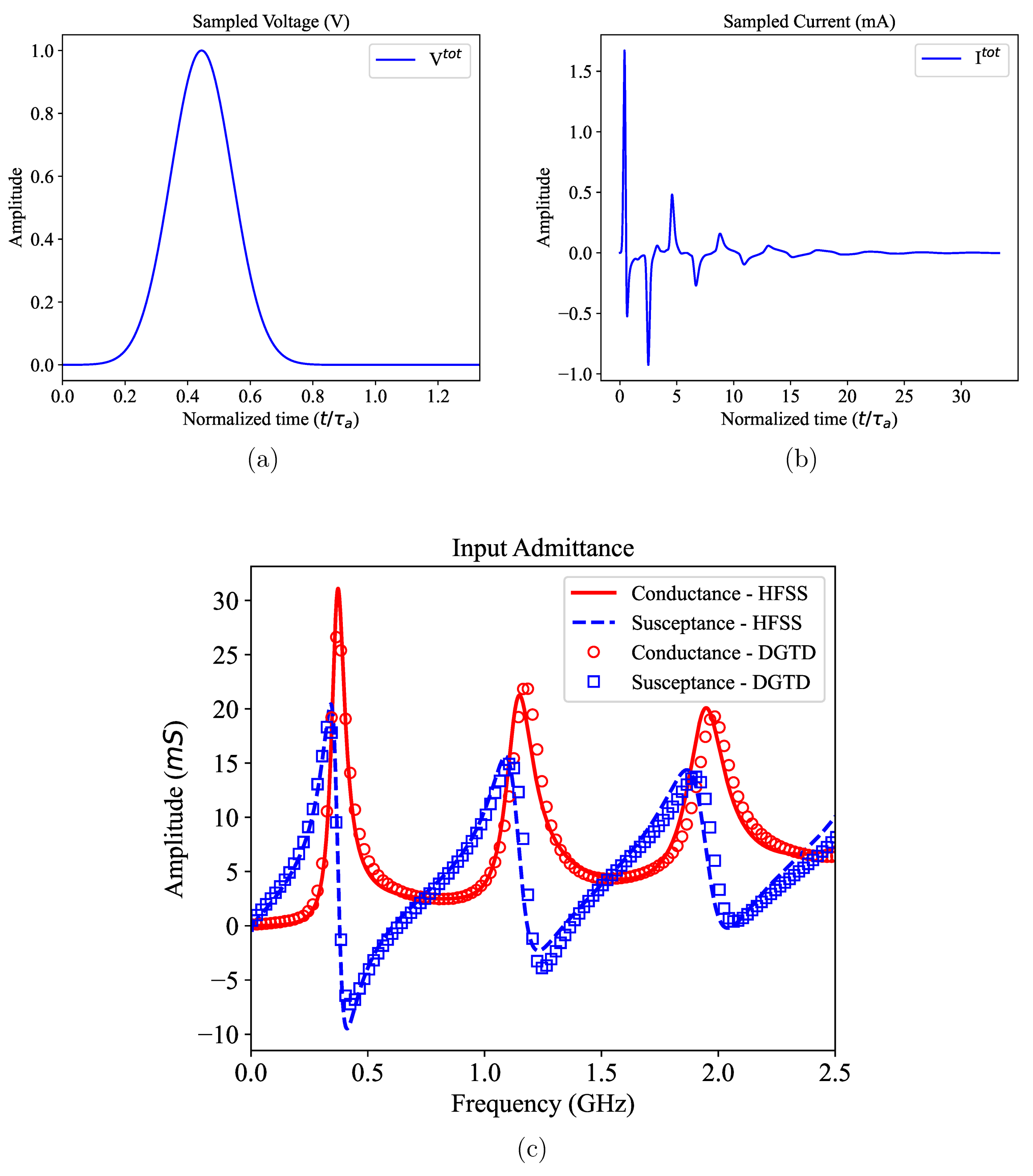
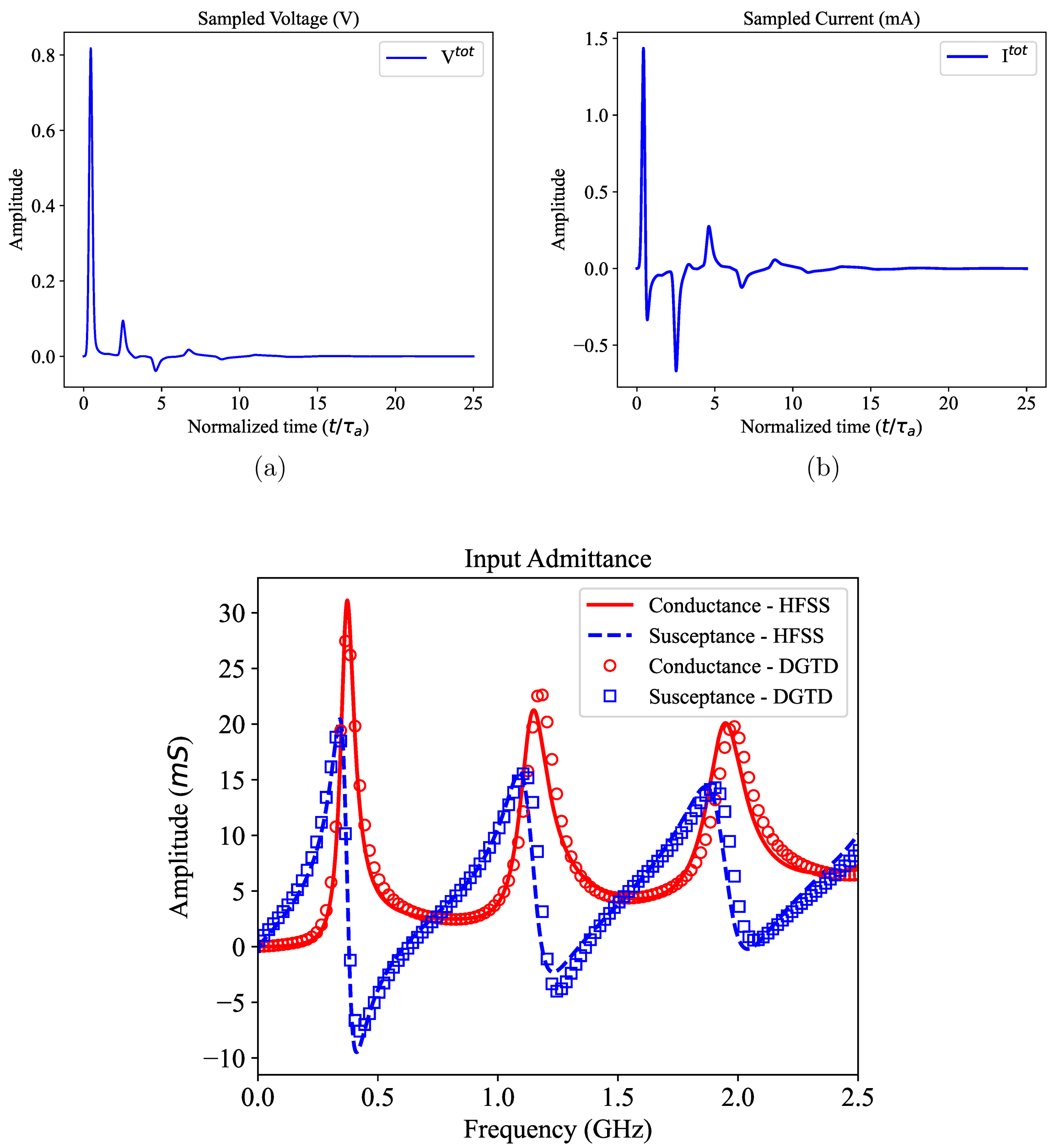


Figure 42 – Numerical results for the delta-gap antenna feed model. (a) Sampled voltage, (b) Sampled current and (c) Input admittance over the range 0-2.5 GHz

outer radii of $l_a = 3.75\,\text{mm}$ and $l_b = 12\,\text{mm}$, respectively, as illustrated in Figure 4. This aperture represents an air-filled coaxial waveport with a characteristic impedance of $Z_0 = 70\Omega$, modeled using the TL method. In this feed technique, the input admittance is calculated by dividing the current flowing between the coaxial aperture and the antenna by the total voltage in the coaxial aperture. The total voltage accounts for both the incident and reflected voltage components. The computational domain is discretized into 42210 tetrahedral elements divided into two classes. $Class(0)$ and $Class(1)$ contain 74.7% and 25.3% of the elements, with the same stable time step values of the previous case,

$\Delta t_0 = 1.24$ ps and $\Delta t_1 = 0.62$ ps. The time-domain waveforms for the total voltage and current, along with the input admittance results, are presented in Figure 43.



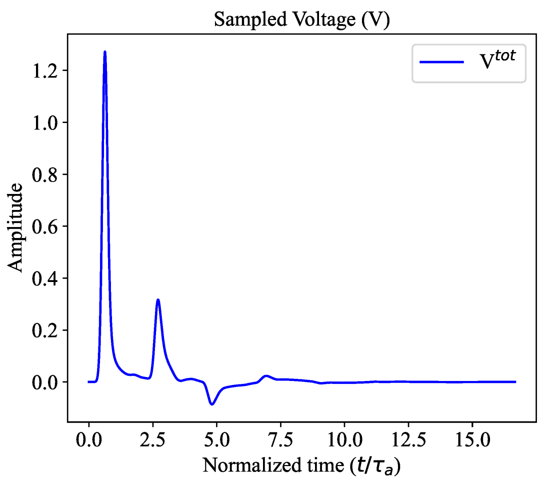
(c)

Figure 43 – Numerical results for the magnetic frill antenna feed model. (a) Sampled voltage, (b) Sampled current and (c) Input admittance over the range 0-2.5 GHz

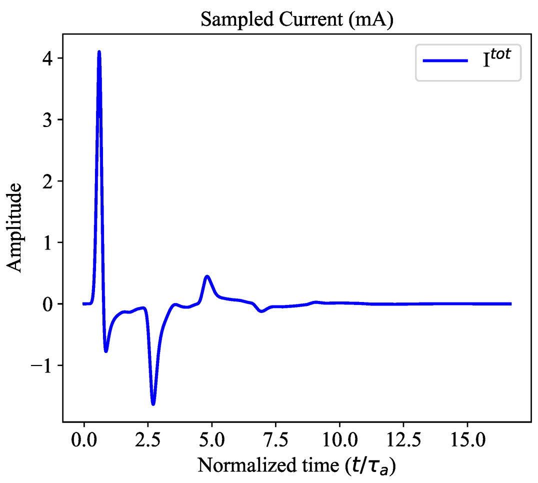
The consideration of reflected waves in this antenna feed model results in greater attenuation of voltage and current waveforms, significantly reducing simulation time compared to the delta-gap approach. This is evident in Figures 43a and 43b, where the waveform amplitudes decay to zero approximately $25\tau_a$ after the simulation begins. In other words, the magnetic frill method achieves a 25% reduction in simulation time relative to the delta-gap model, without increasing implementation complexity. Additionally, the input admittance results align well with the HFSS reference values, showing no notable differences in amplitude compared to the delta-gap approach. This confirms that both feed models are accurate and effective for solving this antenna problem, as illustrated in Figure 43c.

Finally, the most realistic antenna feed model is implemented by extending the computational domain with a cylindrical section representing the coaxial waveport and the PML absorption region, as depicted in Figure 5. The air-filled coaxial waveport is divided into two sections, separated by the TF/SF interface. The physical section consists of a 30 mm coaxial line where incident fields propagate toward the antenna, while the 5 mm PML region absorbs reflected waves. The coaxial waveport has inner and outer radii of $l_a = 3.75$ mm and $l_b = 12$ mm, respectively. The dominant TEM mode is introduced into the coaxial line by identifying the front-side and back-side nodes of the TF/SF interface and injecting the incident electric and magnetic fields, $\mathbf{E}^{inc}$ and $\mathbf{H}^{inc}$, as defined in Eq. 2.36. Once the TEM mode is established, the input admittance is calculated by dividing the current flowing between the coaxial waveport and the antenna by the total voltage at the coaxial aperture. These time-domain quantities are evaluated at the coaxial aperture to enable direct comparison with other feed models. The computational domain for this model is discretized into a tetrahedral mesh with 48353 elements organized in two classes. $Class(0)$ and $Class(1)$ contain 67.9% and 32.1% of the elements, respectively. The marching time simulation is carried out using the time step values $\Delta t_0 = 1.24$ ps and $\Delta t_1 = 0.62$ ps. Notably, this feed model increases the number of mesh elements by approximately 14% compared to the delta-gap approach and 13% compared to the magnetic frill model, as additional discretization is required for the coaxial line. Consequently, this approach is more computationally and memory-intensive than the other feed methods. The time-domain waveforms for total voltage and current, as well as the input admittance results across the frequency range of 0–2.5 GHz, are shown in Figure 44.

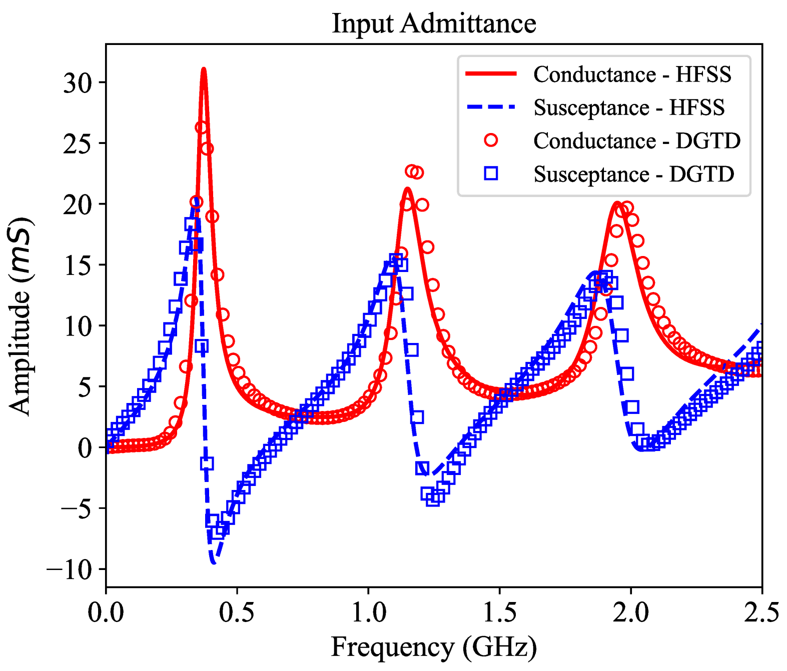
As illustrated in Figures 44a and 44b, the coaxial waveport feed model demonstrates remarkable efficiency in attenuating time-domain waveforms, primarily due to the PML absorption region positioned at the end of the cylindrical coaxial line. Both voltage and current waveforms decay to zero within approximately $15\tau_a$, corresponding to a 56% and 40% reduction in simulation time compared to the delta-gap and magnetic frill feed models, respectively. This significant time saving highlights the efficiency of the coaxial waveport model. However, these benefits come with an increased computational cost due to the higher number of mesh elements required to implement this approach. Therefore, the trade-off between reduced simulation time and increased computational demand must be carefully considered when assessing the overall efficiency and feasibility of this model for specific applications. Finally, as shown in Figure 44c, the input admittance values obtained using the coaxial waveport feed model exhibit excellent agreement with the reference



(a)



(b)



(c)

Figure 44 – Numerical results for the coaxial waveport antenna feed model. (a) Sampled voltage, (b) Sampled current and (c) Input admittance over the range 0-2.5 GHz

values generated by HFSS, further validating the accuracy and reliability of this approach.

Table 16 compares the simulation times of two methods, GPU-DGTD-GTS and GPU-DGTD-LTS, for the monopole antenna problem across three configurations: Delta-gap, Magnetic frill, and Coaxial waveport. The GPU-DGTD-GTS method required 3384 seconds, 2472 seconds, and 2016 seconds, respectively, while the GPU-DGTD-LTS method achieved significantly reduced times of 1526 seconds, 1104 seconds, and 912 seconds for the same configurations. These results show that the GPU-DGTD-LTS method reduces computation time by approximately 55% across all configurations, demonstrating consistent and significant performance gains. This efficiency underscores the practical advantages of the GPU-DGTD-LTS approach for tackling multi-scale engineering problems, such as antenna simulations, establishing it as a robust and effective numerical technique.

Table 16 – Time simulation results for the monopole antenna problem.

Method	Time execution (s)		
	Delta-gap	Magnetic frill	Coaxial waveport
GPU-DGTD-GTS	3384	2472	2016
GPU-DGTD-LTS	1526	1104	912

5.5 Chapter Conclusions

This chapter presents an extensive numerical validation of the proposed GPU-accelerated Discontinuous Galerkin Time-Domain (DGTD) method, including its extension with a Local Time Stepping (LTS) strategy. A series of two- and three-dimensional benchmark problems is solved to evaluate the method's accuracy, convergence rate, and computational performance. Initial tests in homogeneous media, such as a metallic air-filled cavity, demonstrate the accuracy and efficiency of both shared and global GPU memory strategies. Further validation includes the simulation of wave propagation in photonic crystal waveguides and scattering from perfectly electric conducting (PEC) and coated cylinders, highlighting the method's robustness and its ability to capture both near- and far-field phenomena. The LTS algorithm, based on a third-order Runge–Kutta scheme with polynomial interpolation, preserves numerical accuracy while reducing simulation time. Additionally, the combination of GPU parallel computing with the RK3-LTS scheme and third-order interpolation (GPU-DGTD-LTS) proves to be an accurate and efficient approach for solving complex and multi-scale electromagnetic problems.

Numerical results also demonstrate the impact of the interpolation order used to maintain continuity between different classes of elements, showing that high-order interpolation preserves solution accuracy without incurring significant computational costs. The final section applies the proposed method to realistic engineering problems, including a multilayer dielectric sphere and a monopole antenna with various feed models, demonstrating the method's practicality, scalability, and suitability for high-fidelity electromagnetic modeling in complex geometries.

6 Conclusions and Future Research

6.1 Conclusions

This work presents a comprehensive analysis of the DGTD method, focusing initially on two-dimensional problems while also extending its application to three-dimensional cases. Various strategies were implemented to enhance its computational efficiency without significantly compromising accuracy.

- The first strategy involves leveraging graphical processing units (GPUs) to execute computational operations in parallel. GPUs offer low latency and high bandwidth, making them ideal for accelerating numerical simulations. To maximize GPU performance, the distribution of threads was carefully tailored for each CUDA kernel following NVIDIA's recommended best practices. Moreover, the interplay between global and shared memory was extensively explored to identify the optimal configuration during program execution. During the development of this work, it was observed that shared memory is essential when the number of operations per kernel is substantial. For example, shared memory was employed in the volume kernel, where matrix-matrix multiplications are required to compute the field derivatives. Conversely, global memory was utilized for the surface and time integration kernels, as these primarily involve simpler element-wise operations. Furthermore, the methodology was successfully adapted to solve three-dimensional problems, demonstrating the scalability and robustness of the DGTD method in handling complex, high-dimensional scenarios.

  Performance results from the metallic cavity problem indicate a substantial reduction in execution time when comparing serial and parallel implementations. The GPU + Global memory strategy achieves a speed-up of over 14, while the GPU + Shared memory implementation further improves efficiency, providing an impressive speed-up of 24, all compared to the CPU implementation. This confirms the advantage of shared memory in reducing latency compared to global memory. Additionally, the L-shaped waveguide problem was used to further validate the robustness and efficiency of the GPU + Shared memory implementation for handling high-order spatial discretization, achieving a computational improvement of nearly 41 times over the CPU implementation.

- The second strategy explored in this work is the use of the LTS RK3 method combined with a third-order interpolation technique to improve computational efficiency. This approach leverages the discontinuity between elements in the DGTD method and the varying element sizes in an unstructured mesh. By assigning different local time steps to different regions of the computational domain, the LTS RK3 method optimizes time integration while maintaining numerical accuracy.

First, the benchmark metallic cavity problem is analyzed with a computational domain divided into two classes, demonstrating that the LTS RK3 method with third-order interpolation maintains the convergence rate and accuracy while reducing execution time by 40% compared to its GTS counterpart. Next, the method is validated for a complex, multi-scale electromagnetic scattering problem by increasing the number of class divisions to three. This confirms that the previous restriction on the number of classes was overcome, enabling the division of the domain into multiple classes, as discussed in Chapter 4. Finally, numerical results show that the LTS RK3 method achieves the same order of accuracy as the GTS approach while reducing execution time by 52%.

- After discussing the GPU and LTS strategies, we present our proposed approach, which integrates the LTS RK3 method with third-order interpolation, executed on a GPU platform. This approach combines the efficient distribution of local time steps across different element classes in the mesh with the low latency and high bandwidth of GPU acceleration. The synergy of these two strategies results in a powerful numerical technique that significantly enhances computational efficiency while maintaining high accuracy. Initially, the proposed GPU-DGTD-LTS method was tested with two key objectives. First, to demonstrate that it is both an accurate and efficient numerical technique for solving multi-scale electromagnetic problems. Second, to evaluate the impact of interpolation order on ensuring continuity between classes in the LTS algorithm. Numerical results from the metallic cavity and PEC sphere problems show that the GPU-DGTD-LTS method with third-order interpolation achieves speed-ups of 1.46 and 2.42, respectively. Additionally, these results confirm that high-order interpolation in an LTS scheme is not merely a formality but a necessity, as lower-order interpolation compromises accuracy.

  Once validated, the GPU-DGTD-LTS method with third-order interpolation was applied to two realistic and complex electromagnetic problems to assess its robustness and efficiency. The first case involved the scattering of a multilayer dielectric sphere, where the computational domain was divided into four classes to fully exploit the potential of the LTS scheme. Numerical results showed a speed-up of 4.52 compared to the standard GPU-DGTD-GTS, with negligible loss of precision, demonstrating the effectiveness of the proposed method. Additionally, the method was tested on a complex antenna problem, where the multi-scale nature of the computational domain required sophisticated numerical techniques capable of handling large discrepancies in element sizes within the mesh. The results showed strong agreement with the frequency-domain solver ANSYS HFSS, further demonstrating that the GPU-DGTD-LTS method with third-order interpolation is a reliable and efficient approach for solving such problems. In conclusion, the proposed method provides a robust, scalable, and efficient tool for addressing complex multi-scale 2D and 3D electromagnetic problems, paving the way for its application in a wide range of engineering and scientific fields.

6.2 Future Research

The methods developed in this thesis could be expanded in many research aspects that can be addressed in future works. Suggestions for future work are as follows:

- Development of new GPU algorithms integrating the multi-GPU capability such as MPI+GPU and the LTS RK3 scheme to further accelerate the time execution in numerical simulation. The main challenge for a MPI+GPU hybrid parallelism technique is to reach a good parallel scalability for the final implementation. To do this, a domain partition strategy for data parallelism on a multi-GPU platform is required. This domain partition is typically made employed the graph partitioning algorithms Metis and ParMetis in conformal meshes [35], [23] or by using automatic load balancing schemes in non-conformal meshes [2]. Once the load-balanced partitions are found, they are sent to different devices (GPU cores) to perform the calculations. However, these partitions are not completely independent, as neighboring partitions need to share data with each other to complete the calculations. Therefore, proper communication between neighboring partitions is essential to avoid latency issues and not restrict the scalability of the algorithm.

- A key direction for future research is conducting a more extensive study to validate the proposed method in complex, large-scale scenarios, such as those encountered in the simulation of antenna arrays. These scenarios pose significant challenges due to their high demands on memory and computational time, which necessitate the development and integration of more sophisticated techniques to manage the computational domain effectively. One important area of focus is the introduction of advanced boundary conditions tailored to the specific needs of large-scale simulations. For instance, the implementation of periodic boundary conditions could be particularly beneficial for simulating periodic antenna arrays, where the repetitive structure of the array can be leveraged to reduce computational overhead. By accurately modeling the electromagnetic interactions within and across unit cells of the array, periodic boundary conditions can enable efficient and precise simulations of large-scale systems without requiring the explicit inclusion of every element in the domain.

REFERENCES

[1] J. Liu, Z. Yan, J. Ji, and Q. Chang, “Applying pml technique with additional differential equation method in non-uniform mixed-element dgtd method,” *Optik*, p. 165219, 2020.

[2] J. Mi, Q. Ren, and D. Su, “Parallel subdomain level dgtd method with automatic load balancing scheme with tetrahedral and hexahedral elements,” *IEEE Transactions on Antennas and Propagation*, 2020.

[3] Y. Wang, R. Zhao, Z. Huang, and X. Wu, “A verlet time-stepping nodal dgtd method for electromagnetic scattering and radiation,” in *2019 IEEE International Conference on Computational Electromagnetics (ICCEM).* IEEE, 2019, pp. 1–3.

[4] J. S. Hesthaven and T. Warburton, *Nodal discontinuous Galerkin methods: algorithms, analysis, and applications.* Springer Science & Business Media, 2007.

[5] C. Torres-Verdin and T. M. Habashy, “A two-step linear inversion of two-dimensional electrical conductivity,” *IEEE Transactions on Antennas and Propagation*, vol. 43, no. 4, pp. 405–415, 1995.

[6] J.-M. Jin, *The finite element method in electromagnetics.* John Wiley & Sons, 2015.

[7] A. F. Peterson, S. L. Ray, R. Mittra, I. of Electrical, and E. Engineers, *Computational methods for electromagnetics.* IEEE press New York, 1998, vol. 351.

[8] Z. G. Ban and Y. Shi, “Gpu parallelization of wave equation based discontinuous galerkin time domain method,” in *2019 International Applied Computational Electromagnetics Society Symposium-China (ACES)*, vol. 1. IEEE, 2019, pp. 1–2.

[9] J. Cheng, M. Grossman, and T. McKercher, *Professional CUDA c programming.* John Wiley & Sons, 2014.

[10] Q. Yang, Z. Ban, S. Zhu *et al.*, “A nodal discontinuous galerkin time domain method based on wave equation,” *IEEE Antennas and Wireless Propagation Letters*, 2020.

[11] M. Moradi, V. Nayyeri, and O. M. Ramahi, “An unconditionally stable single-field finite-difference time-domain method for the solution of maxwell equations in three dimensions,” *IEEE Transactions on Antennas and Propagation*, vol. 68, no. 5, pp. 3859–3868, 2020.

[12] A. Klöckner, T. Warburton, J. Bridge, and J. S. Hesthaven, “Nodal discontinuous galerkin methods on graphics processors,” *Journal of Computational Physics*, vol. 228, no. 21, pp. 7863–7882, 2009.

[13] T. Cabel, J. Charles, and S. Lanteri, “Multi-gpu acceleration of a dgtd method for modeling human exposure to electromagnetic waves,” 2011.

[14] A. Klöckner, T. Warburton, and J. S. Hesthaven, “High-order discontinuous galerkin methods by gpu metaprogramming,” in *GPU Solutions to Multi-scale Problems in Science and Engineering.* Springer, 2013, pp. 353–374.

[15] C. Nvidia, “Compute unified device architecture programming guide,” 2007.

[16] A. Klockner, “High-performance high-order simulation of wave and plasma phenomena,” Ph.D. dissertation, Brown University, 2010.

[17] L. Zhao, G. Chen, and W. Yu, “Gpu accelerated discontinuous galerkin time domain algorithm for electromagnetic problems of electrically large objects,” *Progress In Electromagnetics Research B*, vol. 67, pp. 137–151, 2016.

[18] H.-T. Meng and J.-M. Jin, “Gpu acceleration of nonlinear modeling by the discontinuous galerkin time-domain method,” *The Applied Computational Electromagnetics Society Journal (ACES)*, pp. 156–159, 2018.

[19] H. Chen, L. Zhao, and W. Yu, “Gpu accelerated dgtd method for em scattering problem from electrically large objects,” in *2018 Cross Strait Quad-Regional Radio Science and Wireless Technology Conference (CSQRWC).* IEEE, 2018, pp. 1–2.

[20] D. Feng, S. Liu, X. Wang, X. Wang, and G. Li, “High-order gpu-dgtd method based on unstructured grids for gpr simulation,” *Journal of Applied Geophysics*, vol. 202, p. 104666, 2022.

[21] Y. Shi, P. Wang, Z. G. Ban, Q. Yang, and S. C. Zhu, “Application of hybridized discontinuous galerkin time domain method into the solution of multiscale electromagnetic problems,” in *2019 Photonics & Electromagnetics Research Symposium-Fall (PIERS-Fall).* IEEE, 2019, pp. 2325–2329.

[22] Z. G. Ban, Y. Shi, Q. Yang, P. Wang, S. C. Zhu, and L. Li, “Gpu-accelerated hybrid discontinuous galerkin time domain algorithm with universal matrices and local time stepping method,” *IEEE Transactions on Antennas and Propagation*, vol. 68, no. 6, pp. 4738–4752, 2020.

[23] Z. G. Ban, Y. Shi, and P. Wang, “Advanced parallelism of dgtd method with local time stepping based on novel mpi+ mpi unified parallel algorithm,” *IEEE Transactions on Antennas and Propagation*, vol. 70, no. 5, pp. 3916–3921, 2021.

[24] S. Piperno, “Symplectic local time-stepping in non-dissipative dgtd methods applied to wave propagation problems,” *ESAIM: mathematical modelling and numerical analysis*, vol. 40, no. 5, pp. 815–841, 2006.

[25] W. Huang and B. Leimkuhler, “The adaptive verlet method,” *SIAM Journal on Scientific Computing*, vol. 18, no. 1, pp. 239–256, 1997.

[26] A. Iserles, “Generalized leapfrog methods,” *IMA Journal of Numerical Analysis*, vol. 6, no. 4, pp. 381–392, 1986.

[27] E. Montseny, S. Pernet, X. Ferrières, and G. Cohen, “Dissipative terms and local time-stepping improvements in a spatial high order discontinuous galerkin scheme for the time-domain maxwell’s equations,” *Journal of computational physics*, vol. 227, no. 14, pp. 6795–6820, 2008.

[28] X. Cui, F. Yang, and M. Gao, “Improved local time-stepping algorithm for leapfrog discontinuous galerkin time-domain method,” *IET Microwaves, Antennas & Propagation*, vol. 12, no. 6, pp. 963–971, 2018.

[29] J. C. Butcher, “A history of runge-kutta methods,” *Applied numerical mathematics*, vol. 20, no. 3, pp. 247–260, 1996.

[30] J. Butcher, “Runge-kutta methods,” *Scholarpedia*, vol. 2, no. 9, p. 3147, 2007.

[31] C. J. Trahan and C. Dawson, “Local time-stepping in runge–kutta discontinuous galerkin finite element methods applied to the shallow-water equations,” *Computer Methods in Applied Mechanics and Engineering*, vol. 217, pp. 139–152, 2012.

[32] L. Angulo, J. Alvarez, F. L. Teixeira, M. F. Pantoja, and S. G. Garcia, “Causal-path local time-stepping in the discontinuous galerkin method for maxwell’s equations,” *Journal of Computational Physics*, vol. 256, pp. 678–695, 2014.

[33] A. Ashbourne, “Efficient runge-kutta based local time-stepping methods,” Master’s thesis, University of Waterloo, 2016.

[34] M. Li, Q. Wu, Z. Lin, Y. Zhang, and X. Zhao, “Novel parallelization of discontinuous galerkin method for transient electromagnetics simulation based on sunway supercomputers,” *The Applied Computational Electromagnetics Society Journal (ACES)*, pp. 795–804, 2022.

[35] ——, “A minimal round-trip strategy based on graph matching for parallel dgtd method with local time-stepping,” *IEEE Antennas and Wireless Propagation Letters*, vol. 22, no. 2, pp. 243–247, 2022.

[36] J. S. Hesthaven and T. Warburton, “Nodal high-order methods on unstructured grids: I. time-domain solution of maxwell’s equations,” *Journal of Computational Physics*, vol. 181, no. 1, pp. 186–221, 2002.

[37] M. J. Lizarazo and E. J. Silva, “A parallel-gpu dgtd algorithm with a third-order lts scheme for solving multi-scale electromagnetic problems,” *Mathematics*, vol. 12, no. 23, p. 3663, 2024.

[38] W. H. Reed and T. Hill, “Triangularmesh methodsfor the neutrontransportequation,” *Los Alamos Report LA-UR-73-479*, 1973.

[39] A. Silveira, R. Moura, A. Silva, and M. Ortega, “Higher-order surface treatment for discontinuous galerkin methods with applications to aerodynamics,” *International Journal for Numerical Methods in Fluids*, vol. 79, no. 7, pp. 323–342, 2015.

[40] A. Hille, R. Kullock, S. Grafström, and L. M. Eng, “Improving nano-optical simulations through curved elements implemented within the discontinuous galerkin method computational,” *Journal of Computational and Theoretical Nanoscience*, vol. 7, no. 8, pp. 1581–1586, 2010.

[41] S. Petersen, C. Farhat, and R. Tezaur, “A space–time discontinuous galerkin method for the solution of the wave equation in the time domain,” *International journal for numerical methods in engineering*, vol. 78, no. 3, pp. 275–295, 2009.

[42] M. Dumbser, M. Käser, and E. F. Toro, “An arbitrary high-order discontinuous galerkin method for elastic waves on unstructured meshes-v. local time stepping and p-adaptivity,” *Geophysical Journal International*, vol. 171, no. 2, pp. 695–717, 2007.

[43] Y. Xu and C.-W. Shu, “Local discontinuous galerkin methods for nonlinear schrödinger equations,” *Journal of Computational Physics*, vol. 205, no. 1, pp. 72–97, 2005.

[44] B. Cockburn, C. Shu, I. for Computer Applications in Science, and Engineering, *Runge-Kutta Discontinuous Galerkin Methods for Convection-dominated Problems*, ser. ICASE report. ICASE, NASA Langley Research Center, 2000. [Online]. Available: https://books.google.com.br/books?id=9vpFAQAAMAAJ

[45] K. Anastasiou and C. Chan, “Solution of the 2d shallow water equations using the finite volume method on unstructured triangular meshes,” *International Journal for Numerical Methods in Fluids*, vol. 24, no. 11, pp. 1225–1245, 1997.

[46] C.-W. Shu, “Discontinuous galerkin methods: general approach and stability,” *Numerical solutions of partial differential equations*, vol. 201, 2009.

[47] L. Angulo, *Time domain discontinuous Galerkin methods for Maxwell equations*. Granada: Editorial de la Universidad de Granada, 2014.

[48] M. Bernacki, L. Fezoui, S. Lanteri, and S. Piperno, “Parallel discontinuous galerkin unstructured mesh solvers for the calculation of three-dimensional wave propagation problems,” *Applied mathematical modelling*, vol. 30, no. 8, pp. 744–763, 2006.

[49] R. J. LeVeque *et al.*, *Finite volume methods for hyperbolic problems.* Cambridge university press, 2002, vol. 31.

[50] L. M. Díaz Angulo *et al.*, *Time domain discontinuous galerkin methods for maxwell equations.* Universidad de Granada, 2015.

[51] S. M. Rao, *Time domain electromagnetics.* Elsevier, 1999.

[52] K. Sankaran, "Accurate domain truncation techniques for time-domain conformal methods," Ph.D. dissertation, ETH Zurich, 2007.

[53] S. Wang, X. Wei, Y. Zhou, Q. Ren, Y. Jia, and Q. H. Liu, "High-order conformal perfectly matched layer for the dgtd method," *IEEE Transactions on Antennas and Propagation*, vol. 69, no. 11, pp. 7753–7760, 2021.

[54] G. Chen, L. Zhao, W. Yu, S. Yan, K. Zhang, and J.-M. Jin, "A general scheme for the discontinuous galerkin time-domain modeling and s-parameter extraction of inhomogeneous waveports," *IEEE Transactions on Microwave Theory and Techniques*, vol. 66, no. 4, pp. 1701–1712, 2018.

[55] A. Taflove and S. C. Hagness, "Computational electromagnetics: the finite-difference time-domain method," *Artech House, Boston*, pp. 149–161, 1995.

[56] J. Alvarez, L. D. Angulo, A. R. Bretones, C. M. d. J. van Coevorden, and S. G. Garcia, "Efficient antenna modeling by dgtd: Leap-frog discontinuous galerkin timedomain method." *IEEE Antennas and Propagation Magazine*, vol. 57, no. 3, pp. 95–106, 2015.

[57] K. Busch, M. Koenig, and J. Niegemann, "Discontinuous galerkin methods in nanophotonics," *Laser & Photonics Reviews*, vol. 5, no. 6, pp. 773–809, 2011.

[58] A. Elsherbeni and V. Demir, *The Finite-difference Time-domain Method for Electromagnetics with MATLAB Simulations.* SciTech Pub., 2009. [Online]. Available: https://books.google.com.br/books?id=3KMKOwAACAAJ

[59] A. Hajiaboli and M. Popovic, "Comparison of three fdtd modeling techniques for coaxial feed," in *2006 IEEE Antennas and Propagation Society International Symposium*, 2006, pp. 3432–3435.

[60] I. Capoglu and G. Smith, "The input admittance of a prolate-spheroidal monopole antenna fed by a magnetic frill," *IEEE Transactions on Antennas and Propagation*, vol. 54, no. 2, pp. 572–585, 2006.

[61] A. Hajiaboli, M. Popovic, and F. Hojat-Kashani, "Analysis of coaxial-fed electromagnetically coupled patch antenna using fdtd," in *2005 IEEE Antennas and Propagation Society International Symposium*, vol. 1A, 2005, pp. 138–141 Vol. 1A.

[62] S.-Y. Hyun, S.-Y. Kim, and Y.-S. Kim, “An equivalent feed model for the fdtd analysis of antennas driven through a ground plane by coaxial lines,” *IEEE Transactions on Antennas and Propagation*, vol. 57, no. 1, pp. 161–167, 2009.

[63] J. Alvarez, L. D. Angulo, A. R. Bretones, C. M. de Jong van Coevorden, and S. G. Garcia, “Efficient antenna modeling by dgtd: Leap-frog discontinuous galerkin timedomain method,” *IEEE Antennas and Propagation Magazine*, vol. 57, no. 3, pp. 95–106, 2015.

[64] J. Li, “Development of discontinuous galerkin methods for maxwell’s equations in metamaterials and perfectly matched layers,” *Journal of Computational and Applied Mathematics*, vol. 236, no. 5, pp. 950–961, 2011.

[65] A. Taflove, S. C. Hagness, and M. Piket-May, “Computational electromagnetics: the finite-difference time-domain method,” *The Electrical Engineering Handbook*, vol. 3, pp. 629–670, 2005.

[66] J. S. Hesthaven, “From electrostatics to almost optimal nodal sets for polynomial interpolation in a simplex,” *SIAM Journal on Numerical Analysis*, vol. 35, no. 2, pp. 655–676, 1998.

[67] M. A. Taylor, B. A. Wingate, and R. E. Vincent, “An algorithm for computing fekete points in the triangle,” *SIAM Journal on Numerical Analysis*, vol. 38, no. 5, pp. 1707–1720, 2000.

[68] Q. Chen and I. Babuška, “Approximate optimal points for polynomial interpolation of real functions in an interval and in a triangle,” *Computer Methods in Applied Mechanics and Engineering*, vol. 128, no. 3-4, pp. 405–417, 1995.

[69] T. Zhang, H. Bao, P. Gu, D. Ding, D. H. Werner, and R. Chen, “An arbitrary high-order dgtd method with local time-stepping for nonlinear field-circuit cosimulation,” *IEEE Transactions on Antennas and Propagation*, vol. 70, no. 1, pp. 526–535, 2021.

[70] X. Ji, T. Lu, W. Cai, and P. Zhang, “Discontinuous galerkin time domain (dgtd) methods for the study of 2-d waveguide-coupled microring resonators,” *Journal of lightwave technology*, vol. 23, no. 11, p. 3864, 2005.

[71] X.-W. Cui, F. Yang, L.-J. Zhou, M. Gao, F. Yan, and Z.-P. Liang, “A leap-frog discontinuous galerkin time-domain method of analyzing electromagnetic scattering problems,” *Chinese Physics B*, vol. 26, no. 10, p. 104101, 2017.

[72] J. Hesthaven and T. Warburton, “Discontinuous galerkin methods for the time-domain maxwell’s equations,” *ACES Newsletter*, vol. 19, no. ARTICLE, pp. 10–29, 2004.

[73] S. C. Chapra, R. P. Canale *et al.*, *Numerical methods for engineers.* Mcgraw-hill New York, 2011, vol. 1221.

[74] J. Park, D. J. Evans, K. Murugesan, S. Sekar, and V. Murugesh, "Optimal control of singular systems using the rk–butcher algorithm," *International Journal of Computer Mathematics*, vol. 81, no. 2, pp. 239–249, 2004.

[75] J. Williamson, "Low-storage runge-kutta schemes," *Journal of Computational Physics*, vol. 35, no. 1, pp. 48–56, 1980.

[76] R. Diehl, K. Busch, and J. Niegemann, "Comparison of low-storage runge-kutta schemes for discontinuous galerkin time-domain simulations of maxwell's equations," *Journal of Computational and Theoretical Nanoscience*, vol. 7, no. 8, pp. 1572–1580, 2010.

[77] F. E. de Souza, "Galerkin descontínuo no domínio do tempo aplicado a problemas com múltiplas escalas em nanofotônica," Ph.D. dissertation, Universidade Federal de Minas Gerais, 2019.

[78] T. Bradley, "Gpu performance analysis and optimization," *NVIDIA Corporation*, 2012.

[79] L. Krivodonova, "An efficient local time-stepping scheme for solution of nonlinear conservation laws," *Journal of Computational Physics*, vol. 229, no. 22, pp. 8537–8551, 2010.

[80] M. J. L. Urbina, "A dgtd method using curved elements to solve electromagnetic scattering problems," Master's thesis, Universidade Federal de Minas Gerais, 2020.

[81] C. Geuzaine and J.-F. Remacle, "Gmsh: A 3-d finite element mesh generator with built-in pre-and post-processing facilities," *International journal for numerical methods in engineering*, vol. 79, no. 11, pp. 1309–1331, 2009.

[82] R. Harrington, *Harrington time harmonic electromagnetic fields.* Wiley-IEEE Press, 2001.

[83] Y. Fu, X. Hu, and Q. Gong, "Silicon photonic crystal all-optical logic gates," *Physics letters A*, vol. 377, no. 3-4, pp. 329–333, 2013.

[84] K. Niknam and J. J. Simpson, "A tf/sf plane wave source condition for the constraint-preserving fvtd method," *IEEE Journal on Multiscale and Multiphysics Computational Techniques*, vol. 8, pp. 108–122, 2023.