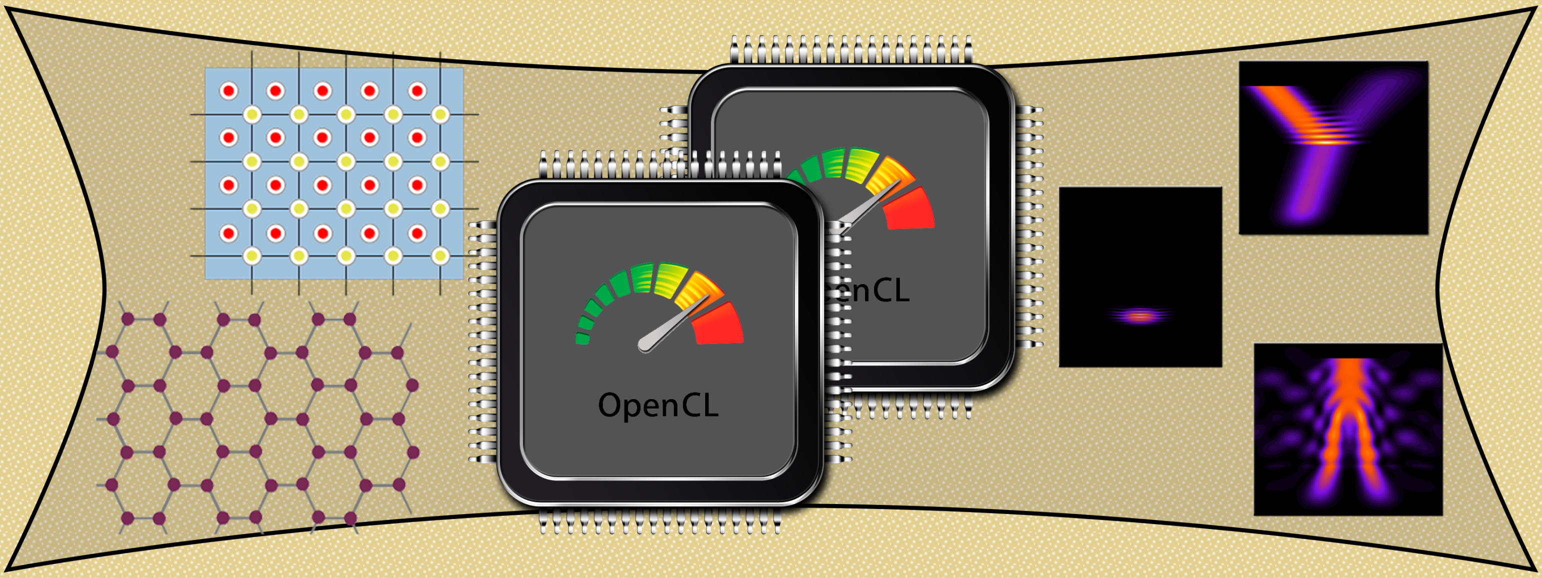


Manuel José Torres Rodrigues

Simulating the Propagation of Electron Waves in Graphene Superlattices: A Parallel Heterogeneous Approach

Março/2015

· U C ·

Universidade de Coimbra

Simulating the Propagation of Electron Waves in Graphene Superlattices: A Parallel Heterogeneous Approach

Manuel José Torres Rodrigues

Dissertação para obtenção do Grau de Mestre em
Engenharia Electrotécnica e de Computadores

Orientadores:	Doutor Gabriel Falcão Paiva Fernandes
	Doutor Mário Gonçalo Mestre Veríssimo Silveirinha

Júri

Presidente:	Doutora Maria do Carmo Raposo de Medeiros
Orientador:	Doutor Gabriel Falcão Paiva Fernandes
Vogal:	Doutor Vítor Manuel Mendes da Silva

Março de 2015

Agradecimentos

Gostaria de começar por agradecer aos meus orientadores, Doutor Gabriel Falcão e Doutor Mário Silveirinha, esta oportunidade, todo o apoio e ensinamentos transmitidos. Quero agradecer também ao David Fernandes por todo o seu auxílio e conselhos dados. A todos com quem partilhei o laboratório, um muito obrigado pelo companheirismo e ajuda oferecida. Deixo também uma palavra de apreço ao João Amaro, pelos recursos que me disponibilizou e que ajudaram a melhorar a qualidade do meu trabalho. Aos meus amigos, um agradecimento muito especial pela amizade e por todos os momentos partilhados. Agradeço aos meus pais e às minhas irmãs todo o apoio e encorajamento manifestado. Estou profundamente grato a todos pelo que cresci e pelos objectivos até aqui alcançados.

Abstract

Graphene, a recently discovered material, is demonstrating its power to revolutionize and drive the future of many industrial fields, as a consequence of the unusual and fascinating properties that it exhibits. The dynamics of electrons propagating in Graphene Superlattices (GSLs) is the subject addressed, with particular focus on recently proposed numerical solution based on the Finite-difference time-domain (FDTD) method. This numerical method has been playing an increasingly important role to solve electromagnetic problems. However, the nature of these techniques imposes very long and tedious simulation times. As a result, the demand for efficient and fast solutions is a requirement to attain simulations with pertinent problem dimensions and reasonable execution times. This thesis proposes a parallel computational approach based on the Open Computing Language (OpenCL) standard, to simulate the time evolution of the electron wave propagating in GSLs, allowing the exploration of heterogeneous computing platforms composed by Central Processing Unit (CPU), Graphics Processing Unit (GPU) and other modern processors. The implemented solution shows significant speed-ups, compared with the traditional tools used to perform this sort of simulations (Matlab and Mathematica), and also provides accurate results to study the behaviour of electron waves is these structures. A speed-up of 180x is observed, when comparing with the Mathematica version, and 100x for the Matlab version. Thus, the execution time is reduced from 36 hours (Mathematica) and 20 hours (Matlab) to a matter of minutes.

Keywords

Open Computing Language (OpenCL), CPU, GPU, Parallel Computing, Graphene Superlattice (GSL) , Finite-difference time-domain (FDTD)

Resumo

O grafeno, um material recentemente descoberto, tem vindo a demonstrar propriedades capazes de revolucionar o futuro de muitas áreas da indústria. A dinâmica dos electrões que se propagam em super-redes de grafeno, é o assunto abordado. Em particular, através de um recente estudo, que propõe um método numérico baseado nas diferenças-finitas no domínio do tempo, para caracterizar a evolução temporal dos electrões neste material. Este método tem vindo a desempenhar um papel cada vez mais importante na computação de modelos electromagnéticos. No entanto, a natureza deste processamento impõe tempos de simulação longos. Esta tese propõe uma abordagem paralela, baseada no standard OpenCL, para simular a evolução temporal da onda do electrão a propagar-se em super-redes de grafeno, permitindo a exploração de recursos em plataformas de computação heterogéneas compostas por Unidades de Processamento central CPUs, Unidades de Processamento Gráfico GPUs e outros tipos de processadores modernos. A solução implementada demonstra um ganho em tempo de simulação muito significativo, comparado com as versões homólogas obtidas em Mathematica e Matlab. Observou-se uma melhoria de 180x para o caso da versão em Mathematica e de 100x para a versão Matlab. Verifica-se assim uma redução na versão paralela mais eficiente, executando em arquitectura heterogénea, de 36 horas (Mathematica) e 20 horas (Matlab) para 12 minutos do tempo de simulação.

Palavras Chave

OpenCL, Unidade de Processsamento Central (CPU), Unidade de Processamento Gráfico (GPU), Computação Paralela, Super-rede de grafeno (GSL), Diferenças-finitas no domínio do tempo (FDTD)

Contents

List of Figures

List of Tables

List of Acronyms

OpenGL	Open Graphics Library
GFLOPS	Giga FLoating-point Operations Per Second
PML	Perfect Matched Layer
ALU	Arithmetic and Logic Unit
AMD	Advanced Micro Devices
GSL	Graphene Superlattice
FDTD	Finite-difference time-domain
API	Application Programming Interface
CPU	Central Processing Unit
CU	Compute Unit
CUDA	Compute Unified Device Architecture
DRAM	Dynamic Random Access Memory
FPGA	Field-Programmable Gate Array
GCN	Graphic Core Next
GPU	Graphics Processing Unit
HPC	High-Performance Computing
OpenCL	Open Computing Language
SIMD	Single Instruction Multiple Data
SIMT	Single Instruction Multiple Thread

1 Introduction

Contents

A continuous technological evolution has proven, throughout history, to be a powerful tool, advancing many scientific and social fields, and consequently improving quality of life. On the basis for this evolution is the realization and manipulation of new materials. Graphene, a carbon based material, is a 2-dimensional crystal that has been around for centuries, since the invention of the pencil [1]. In fact, graphite is composed by stacks of graphene layers and every time a pencil is pressed against a sheet of paper, one can actually be producing graphene layers. Even though graphene was theorized a long time ago, it was isolated for the first time in 2004 by K. S. Novoselov and A. K. Geim [2]. The main reasons behind this is that no one actually thought that graphene could exist in free state and no experimental tools to search one-atom-thick materials existed before. Since its inception, there has been extensive research in order to characterize this material. Among the graphene properties already known are the high conductivity of heat and charge carriers, transparency, strength and flexibility, which makes it one of the strongest and most flexible materials known [1]. These properties are very exciting and have the potential to revolutionize science and many industrial fields.

The characterization and study of all these properties has proven to be a challenge due to the complexity of mathematical models and difficulty to match such complexity to the existing computational resources.

To overcome these challenges, that cut across many scientific areas, there has been an effort to develop computer architectures and software models capable of addressing these problems and provide the right amount of resources, new techniques and concepts.

High-Performance Computing (HPC) platforms play an important role at this level. They allow scientific and engineering applications to enable the execution of computationally intensive tasks. Their evolution has been mainly due to the advancement of computer architectures, namely the Central Processing Unit (CPU) and, lately, the Graphics Processing Unit (GPU). Among others, these represent two architectures widely disseminated and they have been extensively used under the context of this work.

1.1 Objectives

If we consider the graphene time line, it is possible to understand that all major efforts are directed to understanding and characterizing the properties that this material exhibits. Thus, taking also into account the innovations in the computational field, this thesis is focused in the exploration of the computational resources necessary to develop better models and new and faster algorithms to properly characterize the electron wave propagation in graphene-based nanomaterials.

The main goals of this thesis are:

- Supported by a solid theoretical background, perform the study of time evolution of electron waves in graphene;
- Identification of the target parallel computing architectures to achieve significant performance improvements and a suitable programming framework for designing the program and also provide code portability;
- Selection of appropriate data structures for optimal exploitation of the memory hierarchy;
- Development of a parallel strategy and kernels for accelerating numerical methods based on the Finite-difference time-domain (FDTD);
- Design scalable parallel solutions for running on heterogeneous architectures comprised by CPUs and GPUs or the combination of both.

1.2 Main contributions

Commonly, the study of electron wave propagation is performed using Wolfram Mathematica software due to a high-level programming language and a hybrid symbolic-numeric methodology. Applying such software to this particular research field becomes impractical when considering the pertinent spatial dimensions and simulation times for graphene lattices. This results in simulations that can take weeks to complete on a conventional CPU. Thus, in order to better understand the algorithm [3], a Matlab approach was first adopted. A throughput performance upgrade was achieved, however the computation time still was unfeasible, resulting in simulations which took hours to perform. Therefore, the parallelization was a possible next natural step. After considering several platforms, GPUs were selected together with the use of Open Computing Language (OpenCL) framework, to simultaneously achieve throughput performance and code portability. The wave propagation simulation is an iterative process, which imposes limitations in the parallelization procedure. This is due to the nature of the execution of the FDTD algorithm that exhibits, particularly in this study, space and time dependencies. With the parallel solution devised, performance improvements reach up to 180x, reducing, thus, hours of simulations to a matter of minutes.

The contributions of this thesis to the scientific community resulted in a paper published in conference proceedings:

- David Fernandes, Manuel Rodrigues, Gabriel Falcão and Mário Silveirinha; ”Time dynamics of electron waves in graphene superlattices”, Proc Theory, Modelling and

Computational Methods for Semiconductors - European Session, Granada, Spain, Vol. 1, pp. 1 - 1, January, 2015.

Also, we have submitted an article for review:

- David Fernandes, Manuel Rodrigues, Gabriel Falcão and Mário Silveirinha, ”Time Evolution of Electron Waves in Graphene Superlattices”, In Physical Review B, April 2015.

Furthermore, it is under final preparation for submission the following article:

- Manuel Rodrigues, David Fernandes, Mário G. Silveirinha, and Gabriel Falcão. Electron wave propagation in graphene superlattices: Parallel Computational Models for Heterogeneous Architectures. International Journal of High Performance Computing Applications, Sage, May 2015.

In addition, the work developed provides a basis to study other properties, namely the conductivity of graphene, and is now being applied to such purpose. These studies also are computationally heavy and represents an extension on the work already developed.

1.3 Dissertation outline

This thesis is organized in 6 chapters. The following chapter presents a brief overview of multicore architectures, namely CPUs and GPUs. In chapter 3 a detailed description of the theoretical model is presented. The parallelization process of the pipeline is addressed in chapter 4. In chapter 5 the simulation results are discussed. Finally, the conclusions and future work are addressed in chapter 6.

2

Parallel Computing Architectures: Central Processing Unit and Graphics Processing Unit

Contents

The evolution of computer architectures has made a long way and is currently dominated by the massive use of two distinct platforms: the Central Processing Unit (CPU) and the Graphics Processing Unit (GPU). Conceptually, they are designed to cover different purposes. The CPU was created to perform general-purpose tasks, while the GPU was developed to manage graphics related functions such as image rendering. To better understand these architectures it is interesting to browse history and perceive the key milestones that led to the evolution of each architecture all the way up to today standards. It is also important to refer that improvements on the software and compilers side allowed the programmer to take advantage of all the newly available hardware resources, as we see a great amount of effort being put on the improvement and development of new programming paradigms to meet today's requirements. In what concerns the GPU, in the last decade, there has been a massive interest in applying it for general-purpose computing, mainly because of the flexibility and processing power offered.

In this chapter, an overview of CPU and GPU architectures is presented, by analyzing some of the most recent Nvidia, AMD and Intel architectures. Also, a detailed description regarding the OpenCL framework is given.

2.1 Central Processing Unit (CPU)

With silicon and semiconductor technological advances, the number of transistors doubling every two years in the die area according to Moore's Law [4], and also motivated by the increase of complexity in software applications, the CPU architecture is transforming and improving basically every year. Although, it is in constant evolution, the conceptual hardware model remains almost the same. This architecture comprehends a control unit and a datapath, where the first is responsible for managing the flow of instructions and the second performs operations. Today's mainstream CPU architectures offer a small number of complex cores highly optimized for single-thread execution by applying techniques like instruction pipeline, out-of-order execution, branch prediction, among others [5–7]. To minimize memory access time, hierarchical models of small but fast caches are implemented, taking advantage of temporal and spatial locality principles for code and data [8]. In recent years a paradigm shift in the hardware design has been noticed, mostly because of power consumption and heat dissipation walls [9], and architectures continue increasing processing power by adding more cores within the chip and enabling thread execution.

2.1.1 Intel's Sandy Brigde Architecture

To better visualize the CPU architecture philosophy, this section presents an overview on one of the many architectures released by Intel. The Sandy Bridge was the evolution from the Nehalem architecture and the first Intel chip to integrate a graphics processing unit. As depicted in figure 2.1, this computer architecture offers four cores with a hierarchical memory cache system. Each core is associated with two independent levels of cache (L1 and L2) enabling a faster execution of programs by reducing the memory latency when fetching instructions and data. Furthermore, the last level of the cache system (L3) is connected by a ring interconnector in order to share data between the cores and the adjacent GPU. The four cores contain not only Arithmetic and Logic Units (ALUs) but also a large set of resources to accelerate a program execution, such as branch predictors, registers renaming, buffers reordering, among others. Another feature introduced in this architecture is the support of thread execution (Intel's HyperThreading design). In particular, this hardware supports the execution of two treads per core, making a total of 8 available threads. [10]

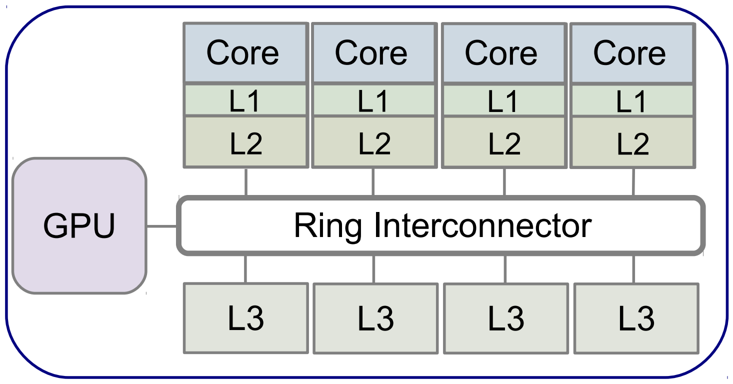


Figure 2.1: Intel Sandy Bridge architecture abstraction.

2.2 Graphics Processing Unit (GPU)

Specialized hardware for graphics purposes have been used since the 1970s [11]. Being the core purpose of these architectures to execute the graphics pipeline, they evolved from graphics-specific hardware to a massively parallel programmable processors [12]. The major motivation for this evolution was, and continues to be, the real-time graphics performance required to render complex 3D scenes mostly for videogames. This demand for performance drastically shifted the architectural hardware design from the CPU, because most of the available chip area is used by computational units benefiting throughput performance instead of low latency [13]. As programmable shaders emerged and after the introduction of new features (floating point arithmetic, special unit functions, double-precision support), it was realized that one could use GPUs for general-purpose processing [14]. The GeForce 8800, introduced by Nvidia in 2006, was the first graphics card using a unified shader model, named Compute Unified Device Architecture (CUDA) [15, 16]. In summary, these devices became more flexible and programmable, and at the same time provide a tremendous amount of computational power, due to the large number of included cores. Furthermore, another key attribute presented in this hardware is the incorporation of large buses, resulting in substantials bandwidths, to feed the computational units with enough data [17]. All these features make the GPU an optimal solution for parallel computing.

2.2.1 GPU architecture

The high-level view of a GPU relies in three fundamental features:

- Clusters of cores;
- Memory system;
- Interconnection Network.

Modern GPUs are designed to pack groups of cores, thus providing a coarse-grain level for data and task parallelism. These cores are characterized by being simple hardware units to perform arithmetic and logic units.

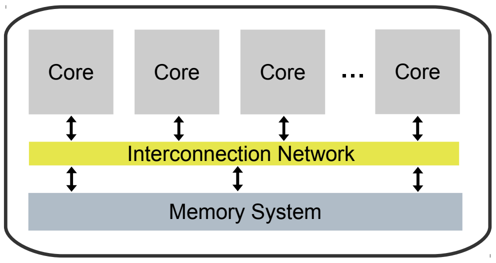


Figure 2.2: High-level view of a GPU.

The memory system presented in current GPUs is mostly based on Dynamic Random Access Memory (DRAM). This technology is characterized by the use of a capacitor to store the bit value. This feature allows the incorporation of more memory capacity in the chip area. A down side of this technology is the increase of latency due to the nature of the capacitors. These memory systems are composed by:

- Banks that store the actual data and have a 2-dimensional structure;
- Controller which is responsible to schedule commands to read/write data from/to banks;
- Bus that connects the banks to the controller.

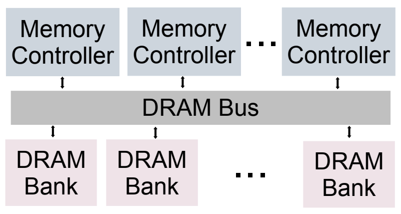


Figure 2.3: DRAM memory system.

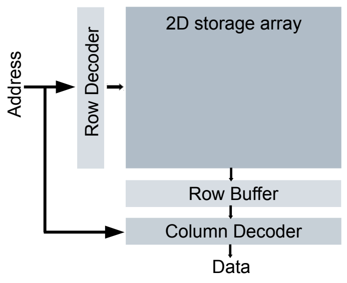


Figure 2.4: DRAM memory bank.

Successive addresses in memory are located in consecutive columns in the same row, thus sequential accesses to the same row in the bank have low latency. Also, current controllers schedule accesses to the same row before scheduling accesses to a different row. The element binding the clusters of cores and the memory system is usually a wide bus (represented in figure 2.2 by the interconnection network). This bus is characterized by large bandwidths in order to fetch enough data to the compute units. Additionally, these architectures offer other resources to improve the execution of the graphics pipeline, in particular, texture units. This is mainly a cache system, that captures 2D spatial locality, helping to fetch memory more rapidly.

2.2.2 Nvidia's Kepler architecture

A recent GPU architecture, the Kepler architecture, is shown in figure 2.5. It features five Graphics Processing Clusters (GPC) comprising, each one, three streaming multiprocessors. The GigaThread Engine is responsible to schedule and distribute groups of threads, called thread blocks, to the streaming multiprocessors to dynamically balance the workload across the device. The Kepler streaming multiprocessor (SMX) comprises 192 single-precision CUDA cores, 64 double-precision units, 32 special function units, 32 load/store units and a 64-Kbyte shared memory/L1 cache. Apart from this, it also offers some special units to improve graphics rendering performance (PolyMorph Engine 2.0). The streaming multiprocessor employs a Single Instruction Multiple Thread (SIMT) architecture, that executes concurrent threads in groups of 32 parallel threads, called warps, and they are managed and scheduled by four warp schedulers. The L1 cache provides low latency, high-bandwidth access to data shared by threads within the same threaded block. [18]

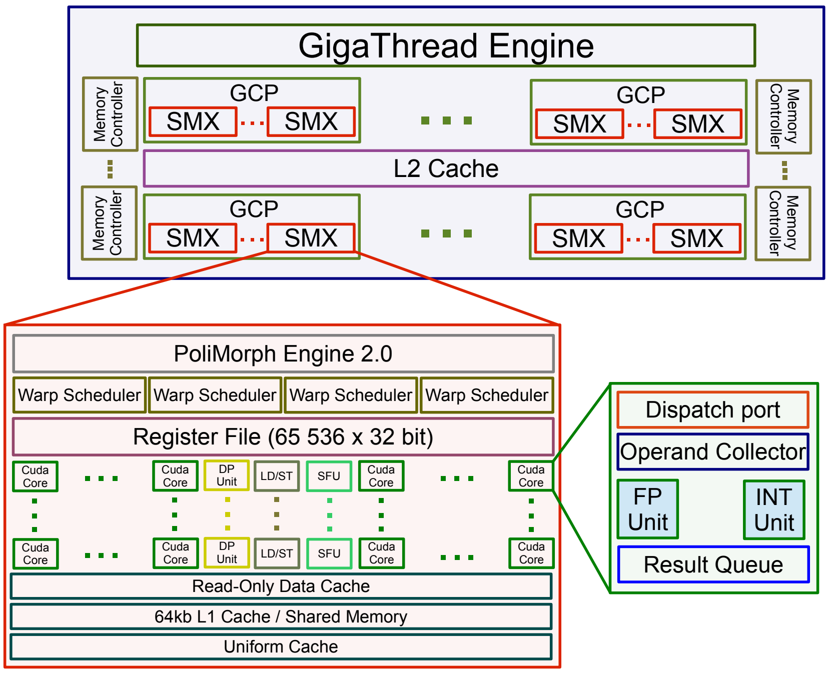


Figure 2.5: Kepler GPU computing architecture.

2.2.3 Advanced Micro Devices (AMD) Graphic Core Next (GCN) architecture

The GCN architecture is the current proposal from AMD to the GPU segment. This hardware features a command processor that is responsible for receiving commands and mapping them onto the two main pipelines (compute shaders and graphics shaders), clusters of Compute Units (CUs), a cache system and schedulers (ACE) that manage the work and resources allocation. The AMD Radeon R9 280X, one of the GPUs used in this work, is composed of 32 CUs. Within a compute unit there are four Single Instruction Multiple Data (SIMD) engine units, and each of these units comprises 16 ALUs. Thus each

SIMD unit executes a single operation across 16 work-items. The hardware also schedules groups of threads to execute concurrently, that are called wavefronts. The number of threads that compose a wavefront is 64. In this architecture, a wavefront takes 4 cycles to execute, thereby a quarter of a wavefront is executed on each cycle. All threads executing on the same Compute Unit (CU) can share data through the Local Data Share unit. Moreover, each SIMD unit executes an independent wavefront and possesses 64KB of vector General-Purpose Registers. [19]

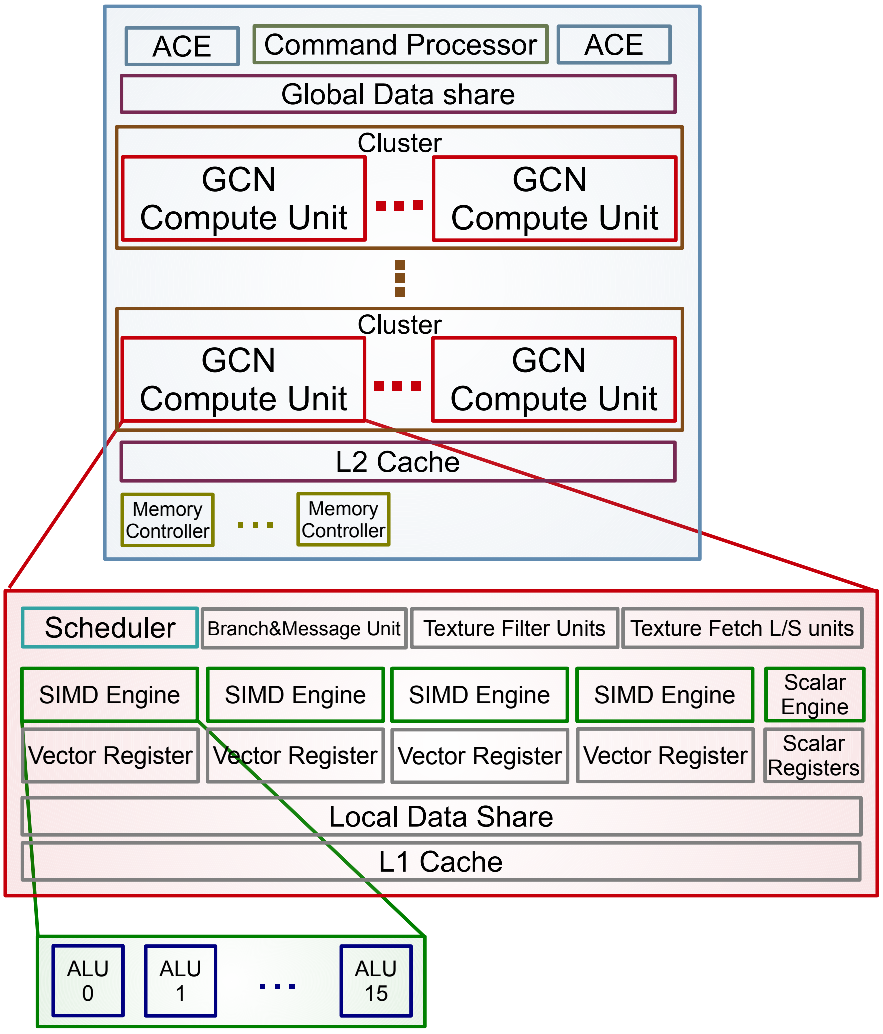


Figure 2.6: CGN computing architecture.

2.3 GPU Programming

The increasing interest on GPU architectures for performing general-purpose computing is helping the advancement of these devices from an hardware and software perspectives, as a consequence of a great deal of effort being made to widespread the concept of GPU computing. From these efforts emerged several programming frameworks, namely OpenCL and CUDA. In essence, GPUs are SIMD engines at the hardware level, however the programming is performed using threads, not SIMD instructions, which is designated by SIMT. Each thread executes the same code, but operates over a different data element, and the hardware groups threads executing the same instruction. As previously described, CPU and GPU hardware architectures have distinct characteristics, thereby it is important to consider some aspects in order to more efficiently utilize the hardware [20, 21]. This section discloses important considerations when programming these architectures.

2.3.1 Coalesced Memory accesses

In contrast to the CPU, GPU devices are memory latency tolerant, meaning that as long as the application executing is compute intensive, it can be computing data while other is being fetched from memory. Thus, one of the most important considerations, when developing GPU applications, concerns the management of memory operations. For an efficient use of the available hardware the program must perform coalesced memory accesses. To better understand how memory operations are performed in the GPU let's assume a group of threads executing in parallel and some scenarios when read/write operations from/to memory are performed.

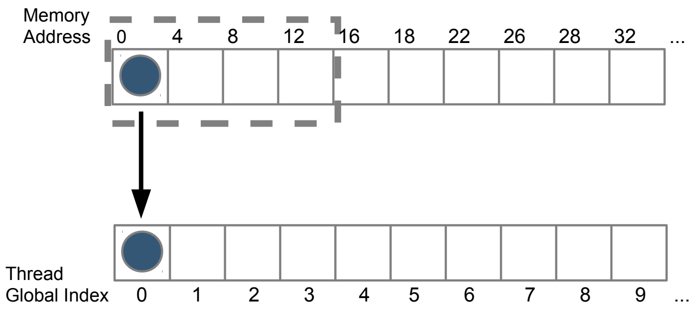


Figure 2.7: One thread accessing one element of memory.

Whenever a thread executes a read or write operation it always accesses a large portion of memory, as depicted in figure 2.7 by the dashed rectangle, even when one element of memory is being accessed. In this case, there is a waste of resources because we are not taking full advantage of the resources available. The ideal case is depicted in figure 2.8. Here, a group of threads executing in parallel are accessing successive elements in memory, which constitutes a coalesced memory access. There is just the need of a single memory access to fetch all the necessary data.

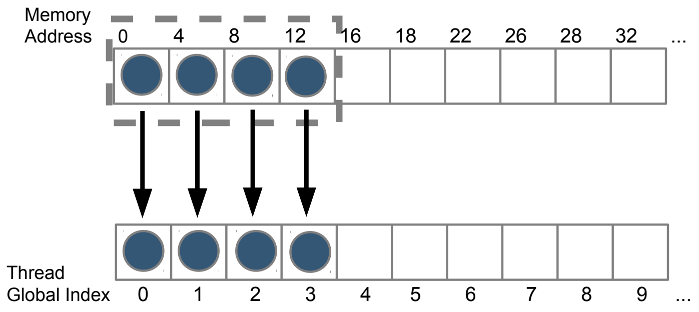


Figure 2.8: Coalesced memory access.

Another example is described by the access of memory locations with a stride. Figure 2.9 shows this feature. Even though the stride is constant, and the same amount of data is being accessed, this operation will take longer to conclude. This is because one memory access will not suffice to deliver all the data, considering the example shown in figure 2.8.

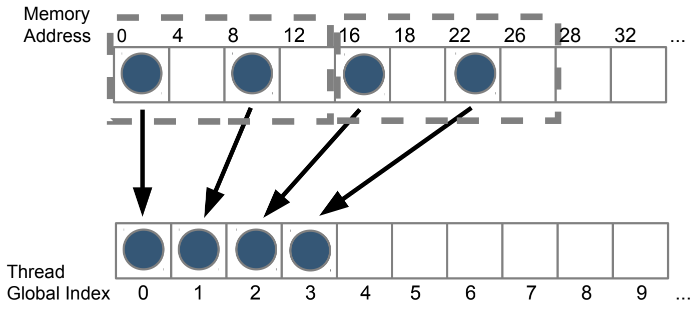


Figure 2.9: Strided memory access.

Therefore, memory operations represent a crucial element that needs to be considered to improve the throughput performance of programs being executed in GPUs.

2.3.2 Memory Bank conflicts

Bank conflicts occur when two or more work-items are fetching data from the same memory bank at the same instant. This results in a serialized operation, damaging performance. Although this is generically true, there is a special case that takes place when all the work-items in the work group are accessing the same memory bank. In this scenario the data is fetched to all work-items at the same operation (also called broadcast). [22]

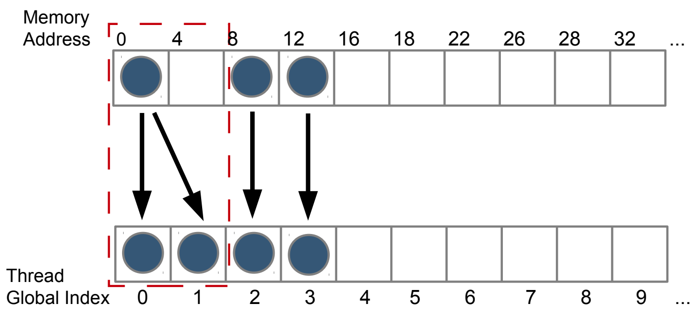


Figure 2.10: Bank conflict. Two threads requesting the same data element.

2.3.3 Branch divergence

Another issue to consider is related with conditional control flow instructions. To illustrate this idea let's assume that the hardware can execute 8 threads in parallel, and all of them are executing the same operations over a data set. When a conditional instruction appears there is a possibility of some threads execute path 1 and the remaining path 2. In this scenario the hardware schedules the execution in parallel of all threads that execute the same path. This leads to a waste of hardware resources, as illustrated in the figure 2.11.

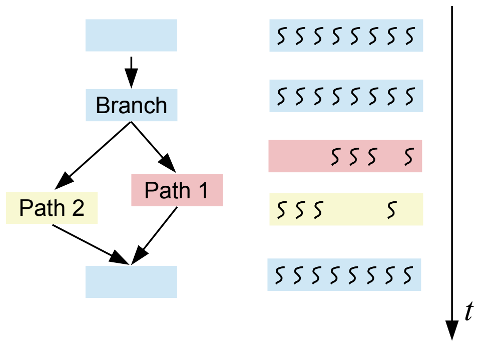


Figure 2.11: Branch divergence scenario where some threads execute path A and the remaining path B.

2.4 OpenCL

OpenCL is a project started by Apple and currently managed by the non-profit technology consortium Khronos group [23]. It defines an open standard that allows the programming of a heterogeneous collection of modern processors (CPUs, GPUs, FPGAs, etc). In his essence, OpenCL is a framework for parallel programming and it includes libraries, an Application Programming Interface (API) and a runtime system. It is also vendor agnostic and the hardware support is increasing every year [24].

OpenCL was the framework chosen to develop the proposed work of this thesis. As previously mentioned it allows parallel programming and most importantly gives the programmer the power of designing portable parallel programs that can run in a wide range of processors [25–27].

To better understanding how the standard works, a detailed review is disclosed. The OpenCL specification is organized in four hierarchical models [28]: platform, execution, memory and programming.

2.4.1 Platform Model

The platform model is a representation of an abstract hardware architecture that programmers target when building their OpenCL applications. The platform model is comprised of a Host connected to one or more OpenCL devices. An OpenCL device is a collection of Compute Units, which in turn are divided in Processing elements. Processing elements execute functions called kernels and can be compiled before or during the program execution. An OpenCL application runs on the host and submits commands to execute computation on the processing elements within a device. The processing elements execute a stream of instructions as SIMD units or as SPMD units.

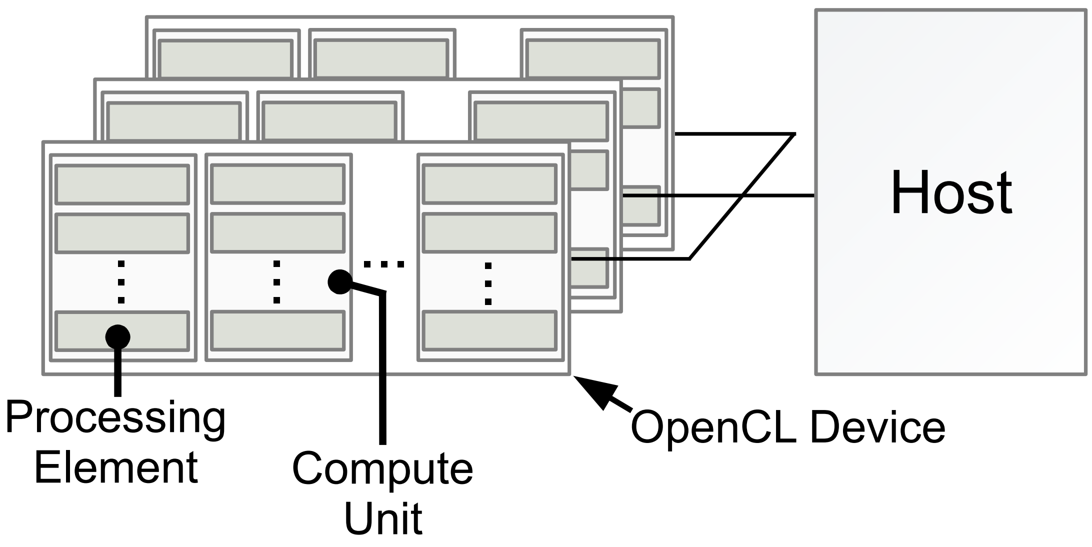
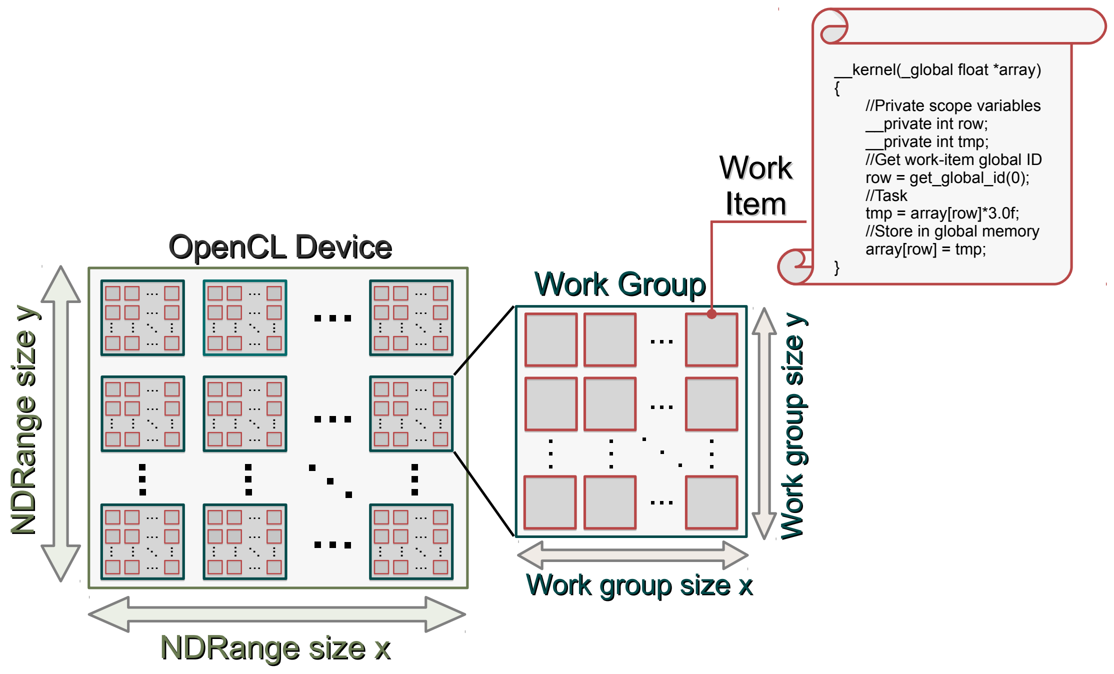


Figure 2.12: OpenCL platform model.

2.4.2 Execution Model

The execution model is divided in a host program and kernels. The host program is responsible for managing the execution of an OpenCL program. It does that by querying the platform where it is running and, accordingly to the query result, allocates the available resources needed for the kernels execution, specifically the OpenCL devices to be used, the kernels to be executed and the memory objects that kernels need. Kernels execute over an index space called NDRange, that is N-dimensional ($N = 1,2,3$) and can be seen as the amount of work to perform. An instance of a kernel is called work-item and executes in the processing elements within the compute unit. Work-items are grouped in work-groups, thus providing a more coarse-grained index space. Each work-item has a identifier in the global index space and in the local index space. When targeting a specific architecture, the work-group size should be an integer multiple of the unit of execution (warp-size for Nvidia or wavefront-size for AMD) since this represents the minimum level of execution granularity supported, otherwise the best performance will not be attained.



```
__kernel(_global float *array)
{
      //Private scope variables
      __private int row;
      __private int tmp;
      //Get work-item global ID
      row = get_global_id(0);
      //Task
      tmp = array[row]*3.0f;
      //Store in global memory
      array[row] = tmp;
}
```

Figure 2.13: Partitioning work-items into work groups.

2.4.3 Memory Model

OpenCL provides a memory model with data flow between regions of memory well defined. In detail, work-items in a kernel have access to four distinct memory regions:

- **Global Memory.** This memory region is accessible for read/write operations to all work-items in all work-groups. Depending on the capabilities of the device the global memory may be cached.

- **Constant Memory.** Only allows read operations and is available to all work-items. It is a region of the global memory but has dedicated hardware in order to lower the average access latency in respect to the Global Memory.

- **Local Memory.** A memory region that is shared by all work-items of the same work-group, and is accessible for read/write operations. Depending on the purpose, users may have to synchronize accesses to this memory in order to keep consistency of the data. This is done by barriers.

- **Private Memory.** Consists in a region of memory private to a work-item and is not visible to any other work-item.

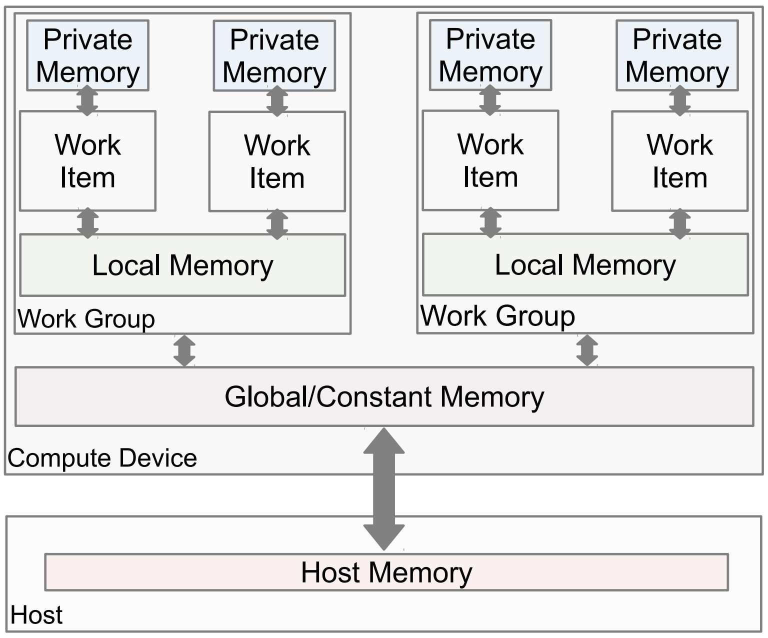


Figure 2.14: OpenCL memory model.

Depending on the application's nature, the exploration of this memory model can result in a performance increase, specially when using local memory. In this case, some requirements must be met by the data access pattern, such as data share between a work-group. Memory consistency must be also considered in order to preserve data integrity. OpenCL provides functions (barriers) to deal with these situations.

2.4.4 Programming Model

The OpenCL standard supports data and task parallel programming models and every device that supports OpenCL implements at least one of these models, although it is more common to see both of them implemented [28].

2.4.4.A Data Parallel Programming Model

A data parallel programming model can be seen as a sequence of instructions applied to a large collection of data. In a strictly data parallel model, there is a direct correspondence between the work-item and the data element over which a kernel can be executed in parallel. OpenCL implements a relaxed version of this model where a strict one-to-one mapping is not required and provides a hierarchical model where the programmer specifies the total number of work-items to execute in parallel and how they are divided among work-groups. Alternatively, the programmer only specifies the total number of work-items and all the rest is managed by the OpenCL framework.

2.4.4.B Task Parallel Programming Model

This case is specified by the model in which the parallelism is expressed by enqueuing multiple tasks to execute in parallel in the device. It can also be viewed as several work-groups executing independently and in parallel different kernels.

2.5 Load balancing techniques under the OpenCL context

When considering a set of tasks that exhibit different behaviors it is also important to match these tasks to the appropriate device. As an example, let's consider a task that manifests branch divergence, in contrast with a task that mainly executes arithmetic operations. In this scenario, it is reasonable to assume that the first is better well suited to perform on a CPU device and the latter on a GPU device. Thus, depending on the performance metric (time, power, accuracy, among others) the development and implementation of programming techniques to suit the intended behavior is an important consideration [29, 30]. OpenCL shows interesting features in this regard. First, because it is an open standard, the majority of modern processors support OpenCL, allowing the utilization of the different devices available in the computing platform. Second, it queries the platform at runtime enabling the resource allocation based on the nature of the available devices, without prior knowledge of platform capacities.

2.6 Summary

The evolution of hardware and software architectures made possible a wider range of computer resources available to the user. Thus, an overview on the CPU and GPU architectures was presented, highlighting the main differences between them. Accordingly, CPU is suited to sequential general-purpose applications, while GPUs provide a highly parallel architecture to efficiently perform data-parallel computations. To attain a parallel and portable solution, the OpenCL framework was chosen and a comprehensive description of the OpenCL specification was addressed.

3

Graphene and the electron wave dynamics

Contents

Finite-difference time-domain FDTD techniques are vastly applied to computationally model a wide range of scientific problems, especially in electromagnetism [31, 32]. This method is mainly characterized by the use of finite differences as approximations to both spatial and temporal derivatives. Also, because it is a numerical method, accuracy considerations must be accounted for. The FDTD method can solve complicated problems, but in general they are computationally expensive. This work analyses a recent proposal that relies in a FDTD numerical method to solve the propagation problem of electron waves in graphene lattices [3]. Accordingly, a brief characterization of the material in the study, a theoretical background and the description of the algorithm being accelerated are shown. Additionally, this chapter clarifies some issues which needed to be addressed in order to perform the simulations.

3.1 Graphene

Graphene is a two-dimensional carbon based nanomaterial, one atom thick, material where the carbon atoms are arranged in a honeycomb structure (figure 3.1). The studies conducted in the past 10 years unveiled great potential for this material, due to the unusual and interesting properties that graphene exhibits. Furthermore, the electronic properties have received an increased attention, thanks to the relativistic spectrum that characterizes graphene. As recently suggested, a suitable tailoring of the graphene structure may provide some control over the transport properties of electrons, for instance, applying an external periodic electrostatic potential on the surface of graphene. These heterostructures are known as graphene superlattices (GSLs) and may be realized through the application of different techniques, such as the use of a crystalline substrate, periodically patterned gates or deposition of adatoms on graphene's surface [33, 34]. Thus, a theoretical review of the effective medium and microscopic model for graphene superlattices and the FDTD solution is disclosed in the next section.

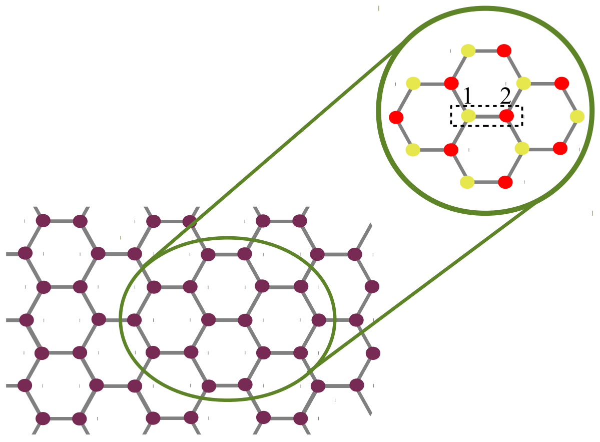


Figure 3.1: Graphene structure.

3.2 Microscopic and effective Hamiltonians of a GSL

The starting point of the following analysis is the massless Dirac equation, that describes the propagation of charge carriers in graphene:

$$(\hat{H}\psi)(r) = i\hbar\frac{\partial}{\partial t}\psi \tag{3.1}$$

The wave function represented by Ψ is a pseudospinor with two components $\Psi = \{\Psi_1, \Psi_2\}^T$. The reduced Plank's constant is $\hbar$ and the microscopic Hamiltonian, that takes into account all the granular details of the graphene material, is given by:

$$(\hat{H}\psi)(r) = -i\hbar v_F(\sigma \cdot \nabla)\psi + V(r)\psi \tag{3.2}$$

From the above equation, the Fermy velocity is $v_F = 10^6 m/s$, $\sigma = (\sigma_x, \sigma_y)$ are the Pauli matrices and $V(r)$ is an external electrostatic potential. Here we consider that this potential has one dimensional spatial variation:

$$V(x) = V_{av} + V_{osc} sin\left(\frac{2\pi}{a}\right) \tag{3.3}$$

where V_{av} in an average potential, V_{osc} is the maximum amplitude of the oscillating part of the potential and a is the period of the potential (figure 3.2). As recently suggested [35], electron waves in periodic systems can be described using an effective medium approach. Thus, the GSL can be regarded as a continuous medium characterized some effective parameters, namely an anisotropy ratio (χ) and an effective potential (V_{eff}) (figure 3.2).

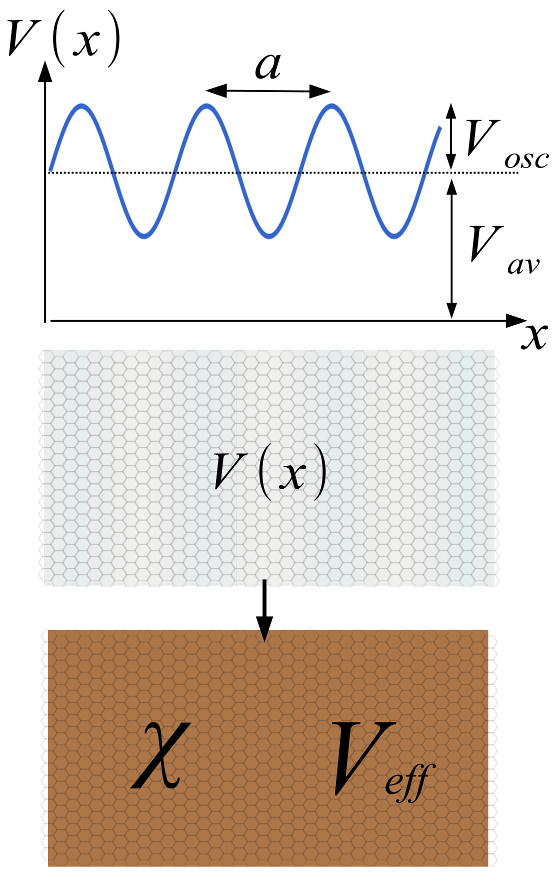


Figure 3.2: Representation of a graphene superlattice characterized by a sinusoidal-like periodic potential. Microscopic (top) and Effective medium (bottom) approaches.

The correspondent effective Hamiltonian can be given by:

$$(\hat{H}_{ef}\psi)(r) = (-i\hbar v_F \sigma(\chi)\cdot\nabla + V_{av})\cdot\psi(r) \tag{3.4}$$

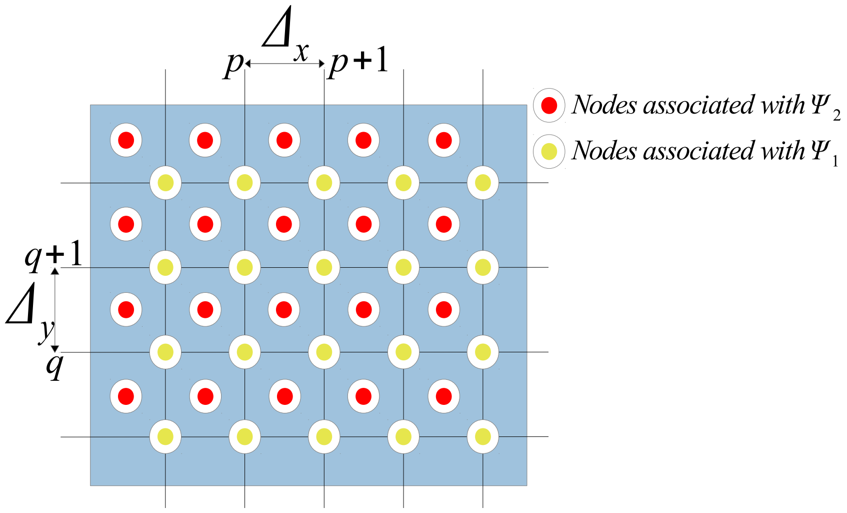
The previous formalism combined with the developed FDTD algorithm results in a system of equations, regarded as update equations, that characterize the time evolution of electrons propagating in GSLs.

3.3 The FDTD numerical solution

In the conception of the FDTD algorithm, there are some important considerations that are mainly related with the discretization of the problem in hands. In this particular case, there is the need to discretize the time and space domain. Thus, it is assumed that the pseudospinor is sampled at consecutive time intervals and a rectangular grid, in the

Cartesian space, is applied to sample the position of the pseudospinor. This enables the use of a finite difference model to calculate the partial derivatives that are presented in equations 3.1, 3.2 and 3.4. The notation adopted, for a generic function, may be written as:

$$F(x,y,t) = F(p\Delta_x, q\Delta_y, n\Delta_t) \equiv F(p,q,n) \tag{3.5}$$



$\varDelta_x$
p
$p+1$
$q+1$
$\varDelta_y$
q
Nodes associated with Ψ_2
Nodes associated with Ψ_1

Figure 3.3: Geometry of the grid for the FDTD method.

The proposed solution results in a pair of equations, each one for the correspondent pseudospinor. They enable the study of the time evolution of the electron wave by applying them in a loop fashion. The microscopic approach update equations are:

$$\begin{aligned}\Psi^{n+1}_{1,p,q}\left(1-\frac{V_{p,q}}{2\hbar i}\Delta_t\right) &= \Psi^{n}_{1,p,q}\left(1+\frac{V_{p,q}}{2\hbar i}\Delta_t\right)+\\ &-v_F\Delta_t\left[\left(\frac{1}{2\Delta_x}-i\frac{1}{2\Delta_y}\right)\Psi^{n+\frac{1}{2}}_{2,p+\frac{1}{2},q+\frac{1}{2}}-\left(\frac{1}{2\Delta_x}+i\frac{1}{2\Delta_y}\right)\Psi^{n+\frac{1}{2}}_{2,p-\frac{1}{2},q+\frac{1}{2}}+\right.\\ &\left.+\left(\frac{1}{2\Delta_x}+i\frac{1}{2\Delta_y}\right)\Psi^{n+\frac{1}{2}}_{2,p+\frac{1}{2},q-\frac{1}{2}}-\left(\frac{1}{2\Delta_x}-i\frac{1}{2\Delta_y}\right)\Psi^{n+\frac{1}{2}}_{2,p-\frac{1}{2},q-\frac{1}{2}}\right]\end{aligned} \tag{3.6}$$

$$\begin{aligned}\Psi^{n+1}_{2,p+\frac{1}{2},q+\frac{1}{2}}\left(1-\frac{V_{p+\frac{1}{2},q+\frac{1}{2}}}{2\hbar i}\Delta_t\right) &= \Psi^{n}_{2,p+\frac{1}{2},q+\frac{1}{2}}\left(1+\frac{V_{p+\frac{1}{2},q+\frac{1}{2}}}{2\hbar i}\Delta_t\right)+\\ &-v_F\Delta_t\left[\left(\frac{1}{2\Delta_x}+i\frac{1}{2\Delta_y}\right)\Psi^{n}_{1,p+1,q+1}-\left(\frac{1}{2\Delta_x}-i\frac{1}{2\Delta_y}\right)\Psi^{n}_{1,p,q+1}+\right.\\ &\left.+\left(\frac{1}{2\Delta_x}-i\frac{1}{2\Delta_y}\right)\Psi^{n}_{1,p+1,q}-\left(\frac{1}{2\Delta_x}+i\frac{1}{2\Delta_y}\right)\Psi^{n}_{1,p,q}\right]\end{aligned} \tag{3.7}$$

For the effective medium model approach update equations are:

$$\begin{aligned}
\Psi^{n+1}_{1,p,q}\left(1-\frac{V_{ef,p,q}}{2\hbar i}\Delta_t\right) &= \Psi^{n}_{1,p,q}\left(1+\frac{V_{ef,p,q}}{2\hbar i}\Delta_t\right)+\\
&-v_F\Delta_t\left[\left(\frac{1}{2\Delta_x}-i\frac{\frac{\chi_{p,q}}{2}+\frac{\chi_{p+\frac{1}{2},q+\frac{1}{2}}}{2}}{2\Delta_y}\right)\Psi^{n+\frac{1}{2}}_{2,p+\frac{1}{2},q+\frac{1}{2}}-\left(\frac{1}{2\Delta_x}+i\frac{\frac{\chi_{p,q}}{2}+\frac{\chi_{p-\frac{1}{2},q+\frac{1}{2}}}{2}}{2\Delta_y}\right)\Psi^{n+\frac{1}{2}}_{2,p-\frac{1}{2},q+\frac{1}{2}}+\right.\\
&\left.+\left(\frac{1}{2\Delta_x}+i\frac{\frac{\chi_{p,q}}{2}+\frac{\chi_{p+\frac{1}{2},q-\frac{1}{2}}}{2}}{2\Delta_y}\right)\Psi^{n+\frac{1}{2}}_{2,p+\frac{1}{2},q-\frac{1}{2}}-\left(\frac{1}{2\Delta_x}-i\frac{\frac{\chi_{p,q}}{2}+\frac{\chi_{p-\frac{1}{2},q-\frac{1}{2}}}{2}}{2\Delta_y}\right)\Psi^{n+\frac{1}{2}}_{2,p-\frac{1}{2},q-\frac{1}{2}}\right]
\end{aligned}\tag{3.8}$$

$$\begin{aligned}
\Psi^{n+1}_{2,p+\frac{1}{2},q+\frac{1}{2}}\left(1-\frac{V_{ef,p+\frac{1}{2},q+\frac{1}{2}}}{2\hbar i}\Delta_t\right) &= \Psi^{n}_{2,p+\frac{1}{2},q+\frac{1}{2}}\left(1+\frac{V_{ef,p+\frac{1}{2},q+\frac{1}{2}}}{2\hbar i}\Delta_t\right)+\\
&-v_F\Delta_t\left[\left(\frac{1}{2\Delta_x}+i\frac{\frac{\chi_{p+\frac{1}{2},q+\frac{1}{2}}}{2}+\frac{\chi_{p+1,q+1}}{2}}{2\Delta_y}\right)\Psi^{n}_{1,p+1,q+1}-\left(\frac{1}{2\Delta_x}-i\frac{\frac{\chi_{p+\frac{1}{2},q+\frac{1}{2}}}{2}+\frac{\chi_{p,q+1}}{2}}{2\Delta_y}\right)\Psi^{n}_{1,p,q+1}+\right.\\
&\left.+\left(\frac{1}{2\Delta_x}-i\frac{\frac{\chi_{p+\frac{1}{2},q+\frac{1}{2}}}{2}+\frac{\chi_{p+1,q}}{2}}{2\Delta_y}\right)\Psi^{n}_{1,p+1,q}-\left(\frac{1}{2\Delta_x}+i\frac{\frac{\chi_{p+\frac{1}{2},q+\frac{1}{2}}}{2}+\frac{\chi_{p,q}}{2}}{2\Delta_y}\right)\Psi^{n}_{1,p,q}\right]
\end{aligned}\tag{3.9}$$

3.4 Requirements for a functional simulation

The simulations conducted in this work address the time dynamics of an initial state and also the propagation of stationary electron waves. Accordingly, there are some differences. For the first case one applies all the formalism previously described. The propagation of stationary electron waves corresponds to a slightly different formalism that results from the addition of a term, in the equation 3.1, which represents a fictitious external source that injects carriers into the system. The final update equations for both models are quite similar to the previous described and can be viewed in the appendix A.1.

3.4.1 Initial state

Different from the stationary electron waves, where the initial state is null, the study of the dynamics of electronic states dependes on the initial state. Thus, for the time evolution problem it is assumed an initial electronic state of the form:

$$\Psi_{(p,q,0)}=\begin{pmatrix}1\\ \frac{\hbar v_F(k_x+ik_y)}{E_0-V_{av}}\end{pmatrix}\frac{e^{-\frac{(p\Delta_x-X_c)^2+(q\Delta_y-Y_c)^2}{2R^2}+ik_yq\Delta_y+ik_xp\Delta_x}}{\sqrt{2\pi}R}\tag{3.10}$$

where X_c, Y_c correspond to the center position and R the width of the Gaussian wave-packet. The space between nodes is Δ_x and Δ_y and the position of the node is determined by (p,q). The energy of the wave packet is E_0 and $k=(k_x,k_y)$ is the wave vector associated with the electronic state.

3.4.2 Perfect Matched Layer (PML)

In this section, we describe the perfect matched layer, whose role is to mimic the propagation in an unbounded structure. This layer is responsible for the absorption of the electron wave when it reaches the end of the graphene superlattice, without causing reflections that can interfere with the propagation leading to wrong results. Basically this layer is a complex-valued potential that increases exponentially. There is no state of art for PMLs in graphene supperlattices, so the tuning of this layer was obtained empirically.

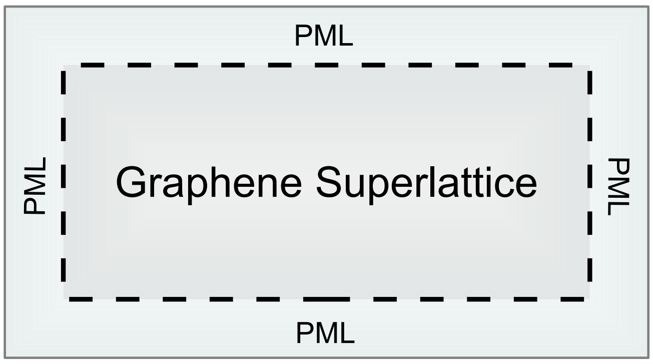


Figure 3.4: Perfect matched layer.

3.5 Summary

In this chapter a brief characterization of graphene, to contextualize this study, was presented. Then, a theoretical background, highlighting the key aspects to implement a functional algorithm, to study the electron wave propagation in graphene supperlattices was conducted.

In the next chapter a parallelization strategy is devised in order to accelerate the execution time of the simulation. From the description given, it is clear that the update equations play the key role in the simulations being, thus, the focus of attention when conceiving a parallel solution. In fact, depending on the number of iterations, they are responsible for the high execution times (representing more than 95% of the execution time).

4 Electron wave propagation on the Graphics Processing Unit

Contents

The objective of this work is to accelerate the execution of the effective medium and microscopic approaches in order to study the propagation of electron waves in graphene superlattices. As reviewed in the previous chapter, the update equations, derived from the application of the Finite-difference time-domain (FDTD) numerical solution, are the main focus when performance improvements are the main goal. Nevertheless, a discussion on how to parallelize all the tasks is presented. Thus, this chapter begins with an understanding on what are the necessary tasks to perform a functional simulation. Task flow and data dependencies are also identified. By gathering all this information a parallel strategy is devised and a discussion on how OpenCL can enable a solution in heterogeneous platforms is shown.

4.1 Problem analysis

The first step towards a functional implementation is the analysis and definition of the tasks to perform. The previous discussion provides a basis on how an algorithm can be implemented. It is important to clarify that, the two earlier depicted approaches (Microscopic and Effective) will be addressed. From an algorithmic perspective it is noticed that both share a similar execution model, i.e. given an initial state and applying iteratively the update equations, results in the propagation of the electron wave. By detailing both approaches it becomes clear that there are different aspects that need to be considered for each model. Thus, the next section discloses the features that characterize each model.

4.1.1 Microscopic Model: task flow and dependencies

In order to study the electron wave dynamics, an initial state must be defined. A potential applied to the graphene lattice has to be tailored and then, resorting in the respective update equations, executing in a loop fashion, the time evolution of the pseudospinor can be recorded. Hence, the following tasks are defined:

- **Potential.** As depicted in the previous chapter the potential applied to the graphene lattice may control the electron wave propagation. It is regarded as an input to the update equations and therefore requires a prior computation. For the studies conducted in this work, the potential considered, in this approach, is a one-dimensional sinusoid. Another aspect to consider is that, when computing the node, if it is located in the Perfect Matched Layer (PML) zone, an imaginary part must be accounted for.

- **Initial state.** The pseudospinor initial state is assumed to be a localized Gaussian wave-packet and is computed also for every node in the grid.

- **Update equations.** The update equations enable the time evolution of the electron wave (3.6, 3.7). They are applied to every node on the mesh and are executed in a loop fashion. The data resulting from the previous tasks, namely the initial state of the pseudospinor and the potential applied to the graphene lattice are the resources that this task uses to compute the node's current value.

4.1.2 Effective Medium Model: task flow and dependencies

In the case of the Effective Medium Model, an anisotropy ratio, an effective potential and an initial state must be previously determined, and then, by applying the corresponding update equations (3.8, 3.9), the wave propagation can be simulated. Consequently, the subsequent tasks are established:

- **Anisotropy ratio.** The anisotropy ratio considered in this study varies in the longitudinal dimension of the lattice, thus, only one line is required to be computed.

- **Potential.** The lattice is characterized by an individual potential associated with each node. Therefore, all the nodes in the mesh need to be computed. This potential is dependent on the anisotropy ratio, thus, a dependency between these tasks is identified.

- **Initial state.** The initial state is the only data feature similar in both approaches, thus it is obtained as described in the microscopic model.

- **Update equations.** As depicted in the section above, the update equations (3.8, 3.9) that allow the study of the electron wave propagation dynamics are employed similarly as previous described.

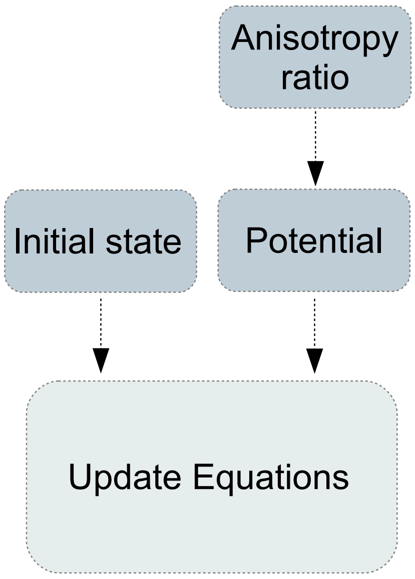


Figure 4.1: The Effective Medium Model tasks and dependencies.

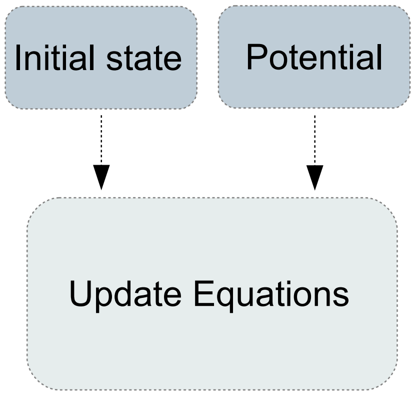


Figure 4.2: The Microscopic Model tasks and dependencies.

4.1.3 Update Equations

The Update equations consist of a pair of numerical functions that are used to determine the value of each node in a given time instant. Thus, they are applied to every node that defines the graphene lattice. Another characteristic is the time and space dependencies that they exhibit. Hence, in order to be executed properly, there are access data patterns and execution procedures that require some particular considerations.

4.1.3.A Time Dependencies

The computation of a time step is always based in the previous one. Consequently, when computing a new time step there is the need to first compute the pseudospinor Ψ_2 and thereafter the pseudospinor Ψ_1. Let us examine a simplified version of the update equations (3.6, 3.7), where it is just demonstrated the wave function time dependencies:

$$\Psi^{n+1}_{1,p,q} = \Psi^{n}_{1,p,q} - \Psi^{n+1}_{2,p+\frac{1}{2},q+\frac{1}{2}} + \Psi^{n+1}_{2,p-\frac{1}{2},q+\frac{1}{2}} - \Psi^{n+1}_{2,p+\frac{1}{2},q-\frac{1}{2}} + \Psi^{n+1}_{2,p-\frac{1}{2},q-\frac{1}{2}} \tag{4.1}$$

$$\Psi^{n+\frac{1}{2}}_{2,p+\frac{1}{2},q+\frac{1}{2}} = \Psi^{n}_{2,p+\frac{1}{2},q+\frac{1}{2}} - \Psi^{n}_{1,p+1,q+1} + \Psi^{n}_{1,p,q+1} - \Psi^{n}_{1,p+1,q} + \Psi^{n}_{1,p,q} \tag{4.2}$$

In equation (4.2) a new time step $(n+1)$ is obtained from the previous (n). Only then, the pseudospinor Ψ_1 can be computed.

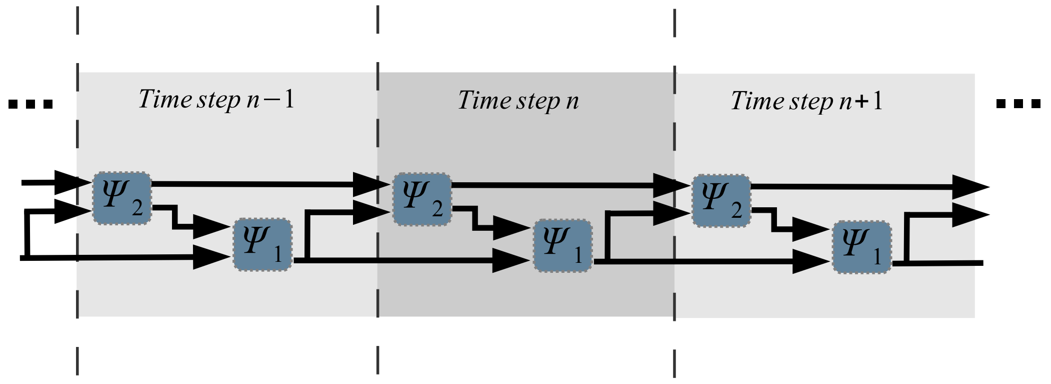


Figure 4.3: Time dependencies illustrated when computing a new time step.

4.1.3.B Space Dependencies

Once again, the starting point will be a simplified version of the update equations (3.6, 3.7):

$$\Psi^{n+1}_{1,p,q} = \Psi^{n}_{1,p,q} - \Psi^{n+1}_{2,p+\frac{1}{2},q+\frac{1}{2}} + \Psi^{n+1}_{2,p-\frac{1}{2},q+\frac{1}{2}} - \Psi^{n+1}_{2,p+\frac{1}{2},q-\frac{1}{2}} + \Psi^{n+1}_{2,p-\frac{1}{2},q-\frac{1}{2}} \tag{4.3}$$

$$\Psi^{n+\frac{1}{2}}_{2,p+\frac{1}{2},q+\frac{1}{2}} = \Psi^{n}_{2,p+\frac{1}{2},q+\frac{1}{2}} - \Psi^{n}_{1,p+1,q+1} + \Psi^{n}_{1,p,q+1} - \Psi^{n}_{1,p+1,q} + \Psi^{n}_{1,p,q} \tag{4.4}$$

Hence, to compute the node (p,q), for the pseudospinor Ψ_1, the neighbors must be accessed. The same behavior is verified for the pseudospinor Ψ_2.

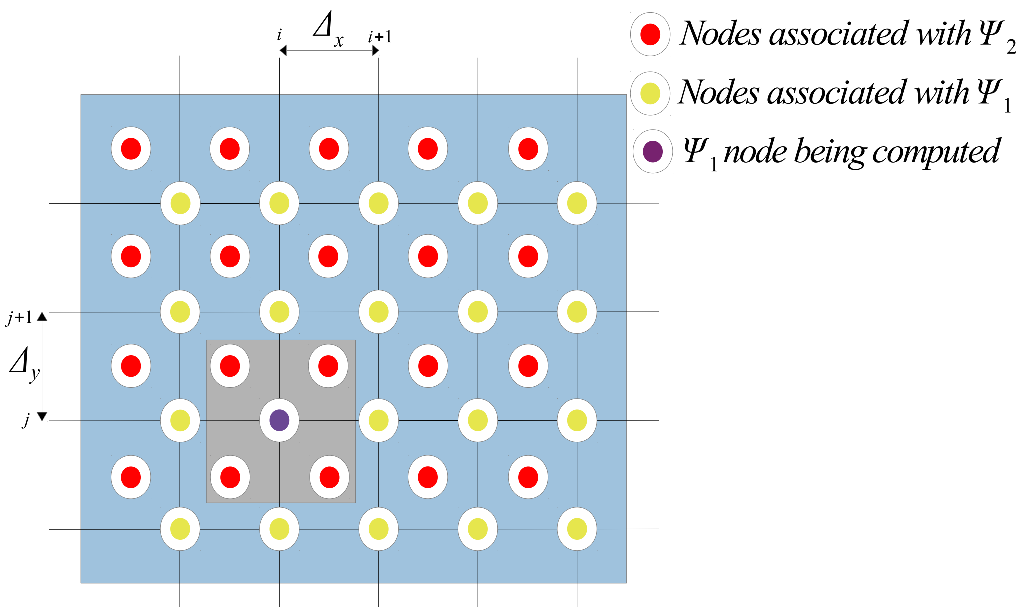


Figure 4.4: ψ_1 space dependencies.

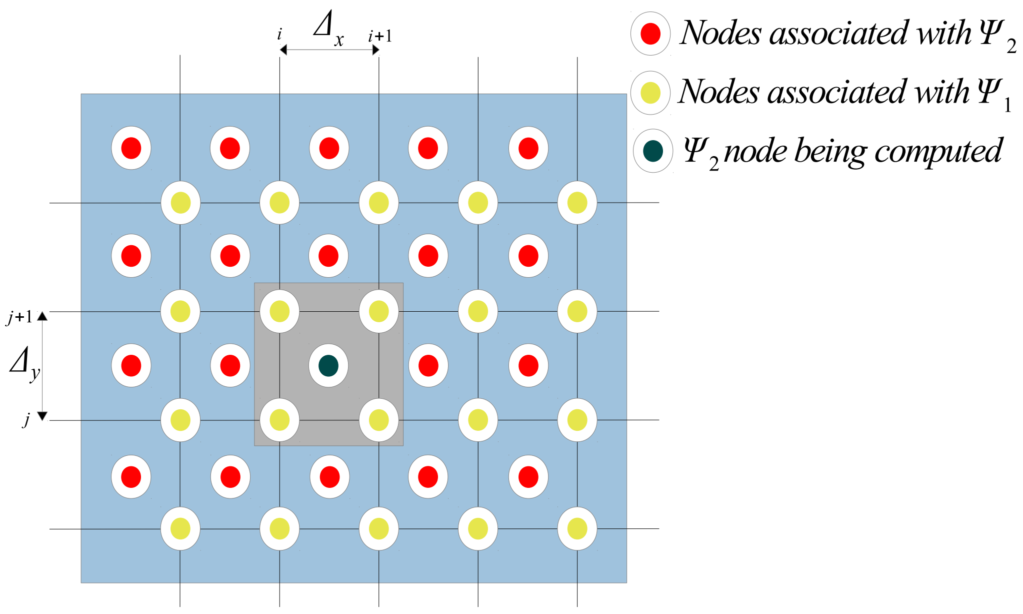


Figure 4.5: ψ_1 space dependencies.

4.2 Parallel approach

With all tasks and dependencies identified we can discuss a parallelization strategy. Based on the previous analysis, the tasks will be grouped using the following criteria:

- **Setup stage.** This stage is regarded as the configuration simulation step. Here, the Potential and Initial State tasks are executed and, when referring to the effective medium model, the anisotropy ratio as well.

- **Update stage.** In this phase, the update equations are executed.

Because the setup stage is only performed at the beginning of the execution and remains constant throughout the simulation, it is straightforward that the main concern, for performance improvements, is focused in execution of the Update phase. In fact, when profiling a simulation the first stage represents less than 1% of total execution time, and this value decreases even more when the number of time iterations grows.

4.2.1 Data structures

A suitable data structure can result in significant performance improvements. Therefore, a proper selection of data types is an important step when idealizing an implementation. We can begin the discussion by assessing the accuracy of the data structures, namely, single and double floating-point precision. Currently in GPUs, double floating-point precision offers less performance, when compared with single floating-point precision. To clarify the difference in performance, lets take the example of the Nvidia GTX Titan processing power. In terms of single-precision it offers 4500 Giga FLoating-point Operations Per Second (GFLOPS) while the double precision processing power is merely 1500 GFLOPS. Thus, a design decision, based on this feature, needs to be accounted for, when performance based applications are developed. This decision should be done regarding the nature of the problem. For the study being conducted and due to the adoption of arbitrary constants, it is possible to employ single-precision floating point to represent all the pertinent values.

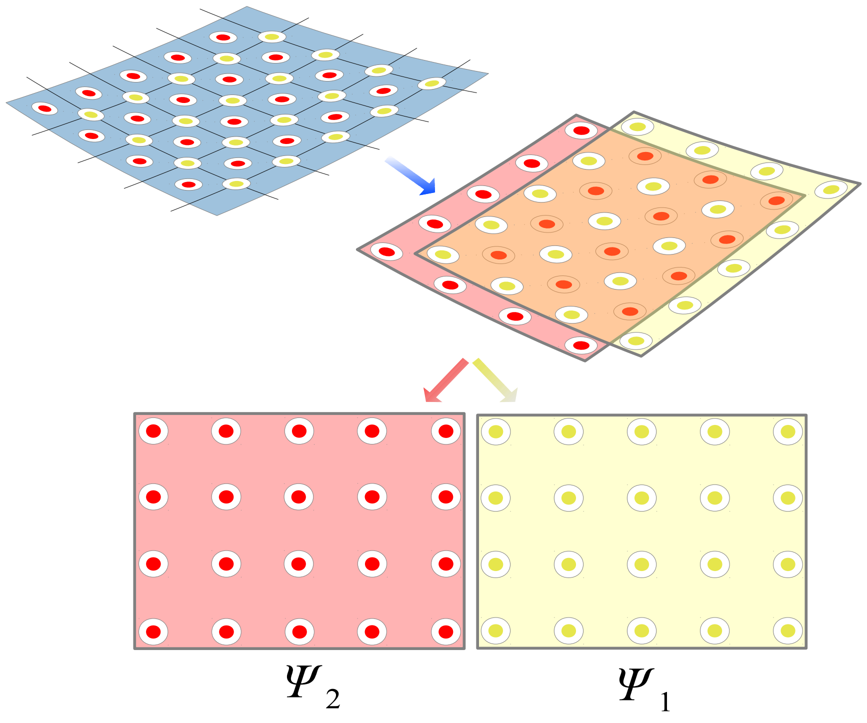


Figure 4.6: Graphene lattice discretized and the two matrices representing the node values of the pseudospinor.

As depicted before, the wave function Ψ is regarded as a pseudospinor with two components (figure 4.6). Furthermore, the wave function node value represents a complex number. Thus, for each pseudospinor component (Ψ_1 and Ψ_2) there is the need to store two values, corresponding to the real and imaginary part of the complex number. In high level software platforms, such as Mathematica and Matlab, complex number operations are supported, although in OpenCL this is not the case. The proposed solution, is to use vector type variables. Moreover, a float4 vector data type is chosen. Thereby, the first two entries of the float4 data type store the real and imaginary parts of the first pseudospinor component and the other two entries are reserved for the second psudospinor component. This same strategy is used to save the potential associated with each pseudospinor component. Another issue, concerning data structures, especially in GPUs, is the access data pattern when performing reading/writing operations from/to memory. Thus, coalesced memory accesses must be conducted, because the hardware enables efficient data transfer operations in the presence of aligned data addresses. Each work-item performs five memory operations, where one of them is a write operation and the remaining are readings. All write operations are coalesced, because the work item index writes in the same memory index. The read operations are also coalesced since each work item accesses adjacent nodes.

4.2.2 Setup stage

Even though this stage does not represent a performance bottleneck, it was also subject of parallelization, furthermore, task and data parallelism was extracted from the correspondent assignments. From the data point of view, each index of the corresponding NDRange is executed independently. Regarding the Effective Medium model, task parallelism cannot be executed straightforwardly due to the dependencies on the anisotropy ratio to compute the potential of the mesh. In the case of the Microscopic Model, both data and task parallelism can be achieved.

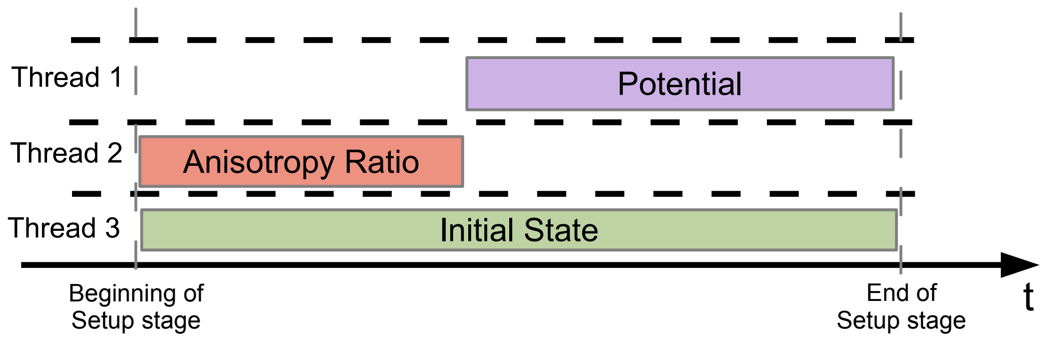


Figure 4.7: Setup stage time profile for the Effective Medium Model.

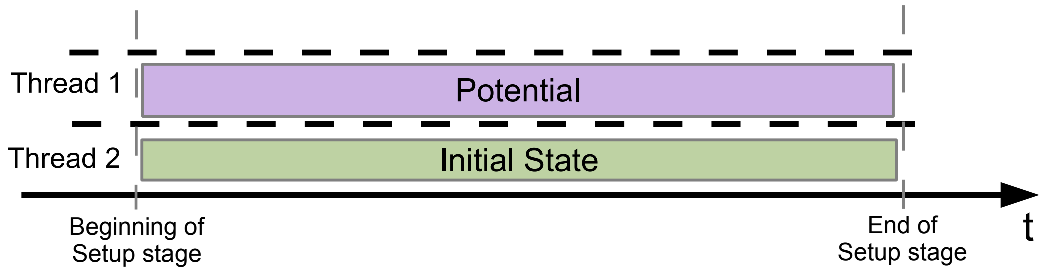
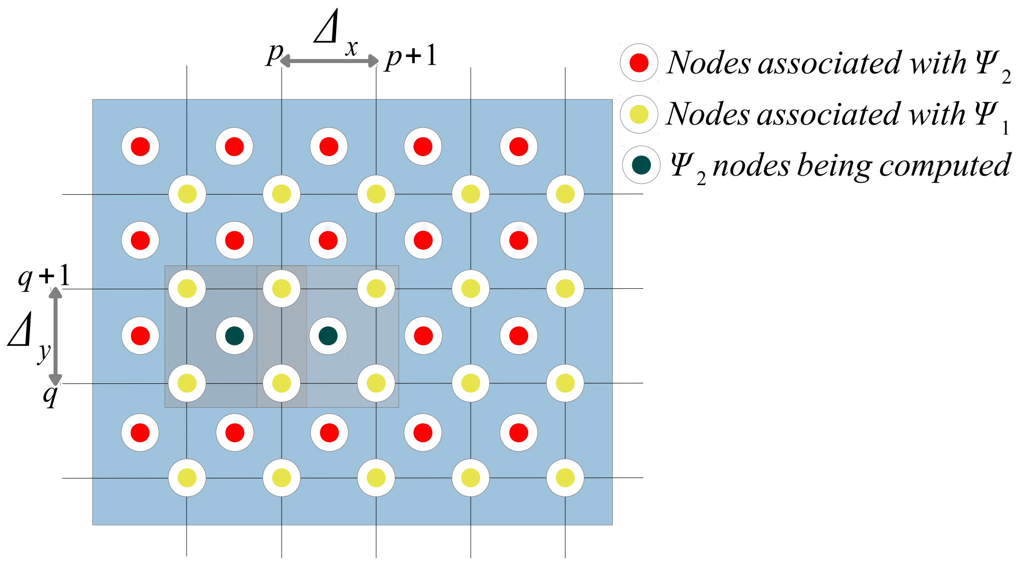


Figure 4.8: Setup stage time profile for the Microscopic Model.

4.2.3 Update stage parallel strategy

As already discussed, the Update stage consists in the implementation of two equations, each corresponding to one of the pseudospinors. The first obvious strategy was the development of a kernel that implements these two equations and applies them to each node. However, this strategy needs to ensure that when computing the node all the resources needed are available, namely the neighboring values of the respective pseudospinor. This approach leads to an increment of operations, mostly comprised of divergent instructions that will obviously affect the overall performance. To better understand this scenario let's consider the computation of a node that corresponds to the pseudospinor Ψ_1. All the adjacent nodes needed to update this value are computed in the same time step, therefore, there is the need of barriers and conditional instructions to assure that the values being accessed are already up to date.



Δ_x
p
$p+1$
$q+1$
Δ_y
q
Nodes associated with Ψ_2
Nodes associated with Ψ_1
Ψ_2 nodes being computed

Figure 4.9: Two nodes associated with the pseudospinor ψ_2 being computed in parallel.

To tackle this problem, the Update stage comprises two kernels, each one associated with a different pseudospinor. Therefore, at each time step, first we compute all the nodes associated with Ψ_2 and only then every node associated with Ψ_1. This strategy ensures that all neighboring nodes are available when the respective node is being computed. By doing this, no extra instructions will be needed to synchronize the computation of nodes. This synchronization is attained when interleaving the execution of the kernels, as demonstrated in figure 4.10.

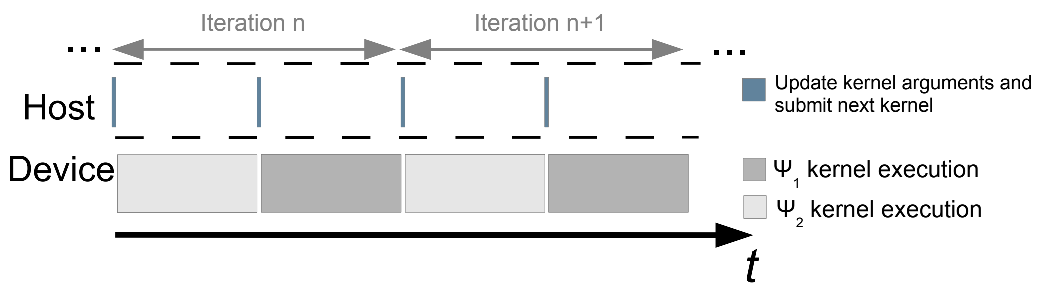


Figure 4.10: Execution of the update stage time profile.

4.3 Exploring the OpenCL memory model

From the two devised kernels, we implemented three versions of them in order to explore the memory model provided by OpenCL and at the same time measure the impact in throughput performance obtained. These crafted versions are based on the use of: global memory; textures; and local memory. As depicted in chapter 2, global memory features the slowest access times. On the other hand, textures (also known as images in OpenCL language) use caches to improve memory speed operations, thus providing average faster access times. Accordingly, the use of local data shared between work groups can sustain even faster memory operations. When devising the kernels based on local data to share among the work-items within a work-group, it has been noticed that divergent instructions (branch conditions) needed to be introduced in order to load the total volume of nodes needed by the work-items within the work group. Thus, as shown in figure 4.11, if we consider the local memory as a matrix, it needs the addition of a column and a row to store the node values required to compute all the nodes in the last column and row in the work-group.

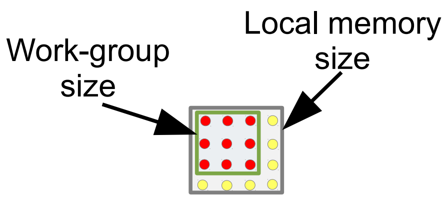


Figure 4.11: Local memory design to service all the work-items within the work-group.

Another aspect that can result in loss of performance, is the occurrence of bank conflicts. Adjacent nodes can access, at the same time, the value of two neighboring nodes.

Thus, these accesses are serialized by the hardware resulting once again in performance penalties. Although this is an acknowledge issue, the resolution of bank conflicts do not have a trivial solution. It needs a comprehensive assessment on how the index space of memory can be managed in order to keep memory consistency. Thus, there were no further developments regarding this aspect.

Another concern, in terms of performance improvements is due to the data transfers between host and device. This performance factor is controlled by the user, depending on how much data is required to conduct the study.

4.4 Exploring multiple GPU based platforms

One of the advantages of OpenCL is that, at runtime, a query of the platform can be performed, and accordingly to the resources available there is the possibility to adapt the program to the platforms available.

Because these simulations require the execution of the two previously described methods, Microscopic and Effective Medium, it is straightforward to say that a system comprised with two GPUs allows a faster overall performance, because we can distribute the work by both devices. Thus, this possibility was explored, and when the resources are available, the program distributes each model to a GPU. Otherwise, each model is executed in a serialized fashion.

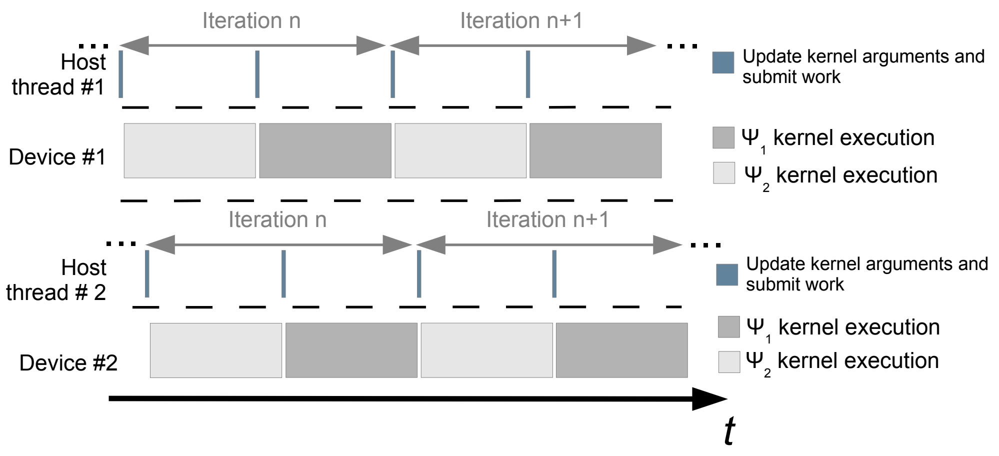


Figure 4.12: Execution of the update stage time profile, when exploring a muti-GPU system.

4.5 Summary

When idealizing a parallel approach there are important issues that need to be addressed. Among them, task and data dependencies are crucial to achieve the proposed goals. This chapter started with a task description of the simulation being accelerated. Then, a parallel strategy was detailed, highlighting the most relevant aspects to consider, such as data structures, task and data parallelization opportunities. With the parallel approach devised, it is important to assess if the performance met the expectations. Thus, in the next chapter, a performance evaluation and the simulations results is presented.

5
Simulation and Results Analysis

Contents

sGiven the solution devised in the previous chapters, this section of the thesis inspects the simulation results and assesses the performance of the strategy utilized. This chapter begins with a description of the hardware platforms used under the context of this work. Then, it displays a set of simulations performed in order to validate the obtained results. Finally, a performance evaluation of the proposed solution is investigated.

5.1 Hardware platforms setup

This section discloses the computer platforms used to execute all the relevant simulations of this work.

	Platform 1	**Platform 2**
CPU	Intel(R) Core(TM) i7-4790k 4GHz	Intel(R) Core(TM) i7 950 3.07GHz
RAM	32GB	6GB
GPU 1	Nvidia GeForce GTX Titan	AMD Radeon R9 280X
RAM	6GB	3GB
GPU 2	Nvidia Tesla k40	Nvidia Geforce GTX 680
RAM	12GB	2GB
OS	Ubuntu 14.04.1 LTS	Linux Debian 3.2.0-4-amd64
Language	C + OpenCL 1.1	C + OpenCL 1.2

Table 5.1: Platforms considered in the experimental evaluation

It can be noticed that the two platforms are composed by different devices, especially platform 2, where this system is composed by different devices from different vendors. The execution of the simulations is these platforms can only be achieved with Open Computing Language (OpenCL), which demonstrates the power of this standard.

5.2 Simulation results

To ensure that the developed work provides accurate results, it was proposed a set of scenarios for simulation. These simulations were designed to study and validate the devised solution. The next table displays the dimensions of the lattice. At this point is important to stress that these parameters correspond to the better setup. Because we are working with numerical methods, that constitute an approximation to the solution, there are some considerations that need to be accounted for. Thus, in this particular case, the use of a refined grid is the goal.

	Simbol	**Vaule**
Number of Time iterations	N_x	4096
Number of Time iterations	N_y	2048
Space between nodes x direction	Δ_x	0.025
Space between nodes y direction	Δ_y	0.025
Time interval	Δ_t	0.0062

Table 5.2: Constant parameters used in the simulations.

5.2.1 Propagation of stationary electron waves

This section begins with the analysis of the propagation of electron waves in a Graphene Superlattice (GSL) characterized by a potential that may allow the propagation of Gaussian electron waves without diffraction. Here, we analyze the results of both approaches (microscopic and effective medium).

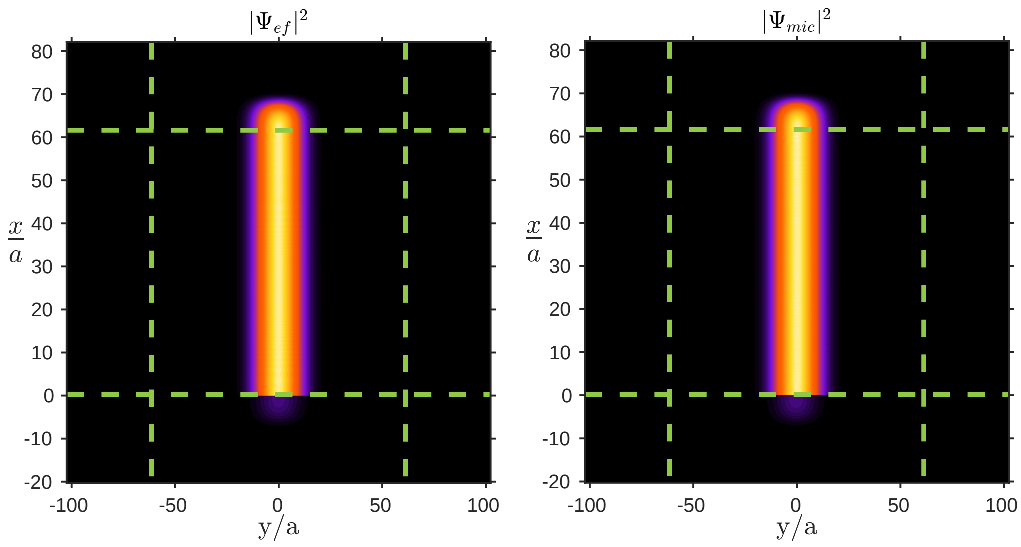


Figure 5.1: Density plot $|\Psi|^2$ obtained for the effective medium (left) and microscopic (right) theories.

Evaluating the displayed results in figure 5.1, it is noticed that the propagation of the Gaussian electron wave along the GSL, occurs without diffraction. Also, a further inspection of figure 5.2, that compares the longitudinal profiles of the wave function for both models, reveals an agreement of the two approaches. Since this simulation is characterized by a constant injection of electrons into the system, we aim, in these simulations, to reach a steady state.

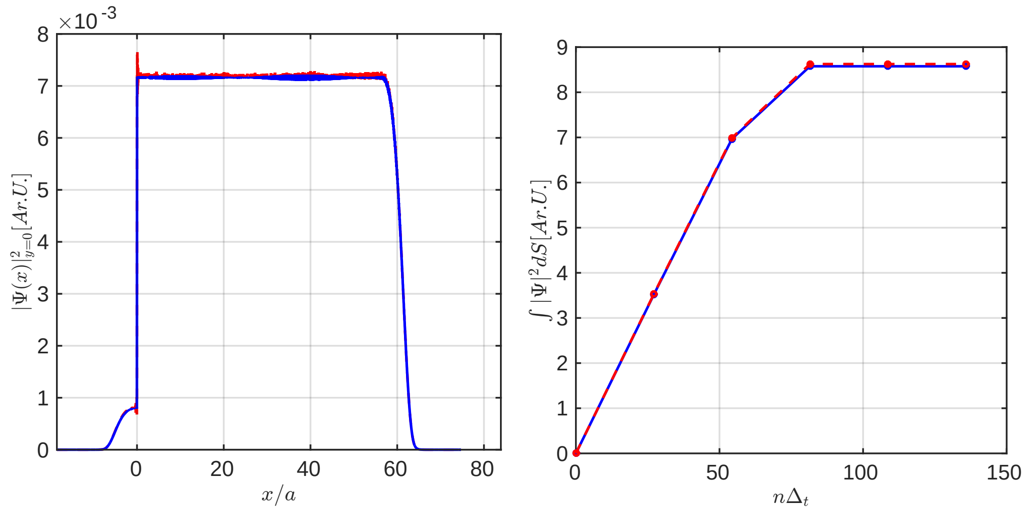


Figure 5.2: In the left, the surface integral of the probability density function $|\Psi|^2$. In the right, longitudinal profiles of the pseudospinor using the effective medium (blue curve) and microscopic (red curve) approaches.

The next figure shows the negative refraction of electron waves at the interface of two pristine graphene regions. This is obtained by tunning the average potential in the pristine graphene region.

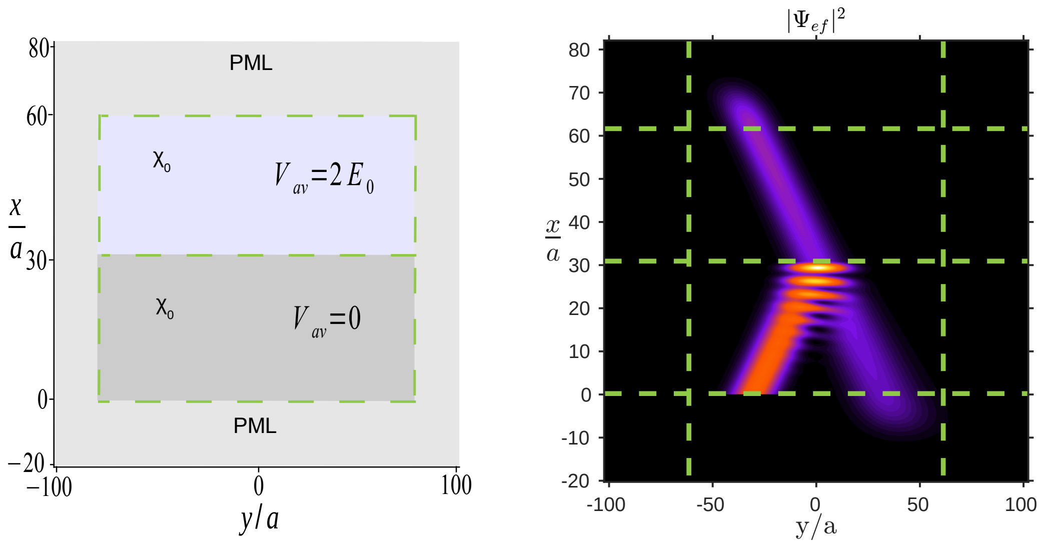


Figure 5.3: Left: Geometry of the graphene superlattice. Right: Density plot of $|\Psi|^2$ calculated for a Gaussian electron wave with energy E_0 that propagates in graphene pristine ($\chi_0 = 1.0$) and impinges on a region with $V_{av} = 2E_0$.

As can be seen in the above figure 5.3, a proper selection of the average potential enables the electron wave to propagate in the reversal direction, when compared with the conventional behavior of a refraction phenomenon. Another of the simulation scenarios tested comprehends a more complex structure, characterized by a GSL embedded in pristine graphene. Also a circular obstacle was included in the middle of the GSL. This barrier is conceived as an average potential greater than the one assigned to the GSL.

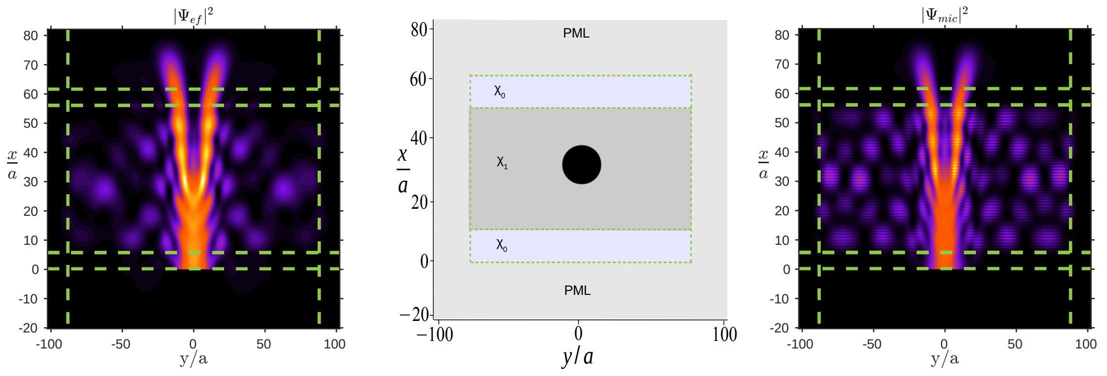


Figure 5.4: Left: Density plot of $|\Psi|^2$ calculated for a Gaussian electron wave, when the steady state was achieved, for the effective medium model. Center: Geometry of the structure. Right: Density plot of $|\Psi|^2$ calculated for a Gaussian electron wave, when the steady state was achieved, for the microscopic model.

Figure 5.4 demonstrates the similarity of the results given by the microscopic and effective medium models. An interesting feature shown in this example is the behavior of the electron wave when it reaches the circular obstacle. In can be noticed that the beam is diffracted and it is split in two directions.

5.2.2 Time evolution of an initial state

As described in this work, the simulations performed not only encompass the study of stationary waves, as they also cover the study of time dynamics of an initial electronic state propagating throughout the GSL. The fundamental difference in these scenarios is that there is no electron source constantly injecting electrons into the system. Thus, this type of time evolution problems never reach a steady-state. Also, keeping in mind that the graphene structure is surrounded by a Perfect Matched Layer (PML), the initial state will be absorbed and the surface integral of the probability density function will get to zero.

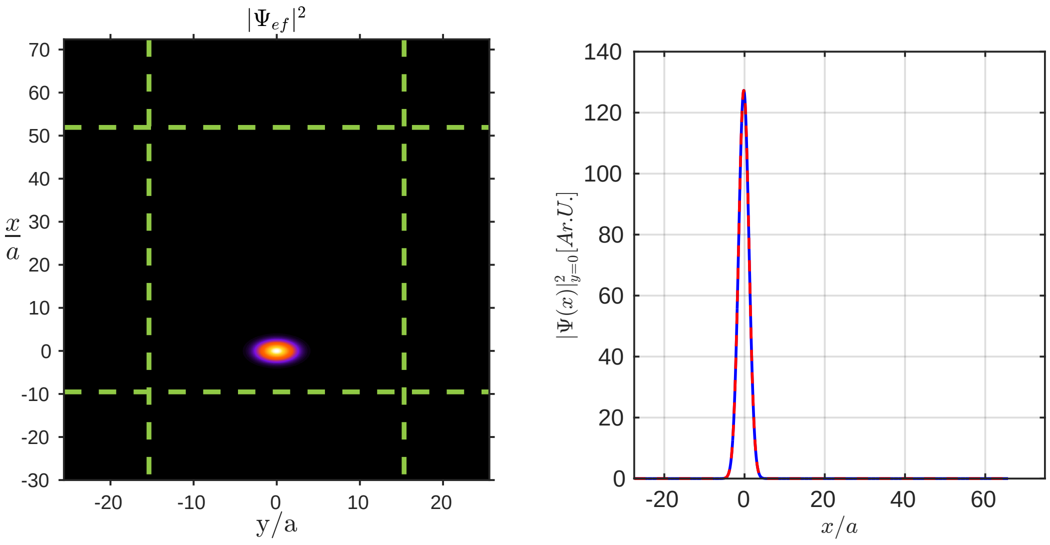


Figure 5.5: Left: Density plot of $|\Psi|^2$ that represents the inital state which propagates in pristine graphene ($\chi_0 = 1.0$). Right: Longitudinal profile of $|\Psi|^2$. Effective medium model is represented as the blue line and the microscopic as the red dashed line.

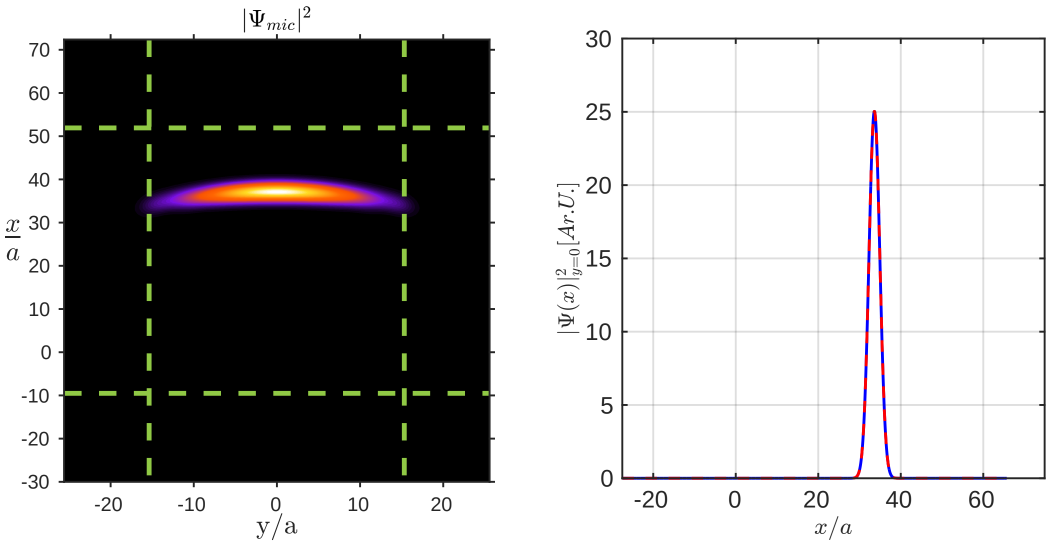


Figure 5.6: Left: Density plot of $|\Psi|^2$ obtained after 4000 time iterations. Right: Longitudinal profile of $|\Psi|^2$. Effective medium model is represented as the blue line and the microscopic as the red dashed line.

The figures above show the result for the simulation of an initial state propagating in a pristine graphene structure (pure graphene). This structure is isotropic thus the electrons have no defined direction, and that is why it is observed a widening of the beamwidth. Also, once again it is shown that the two models are in good agreement.

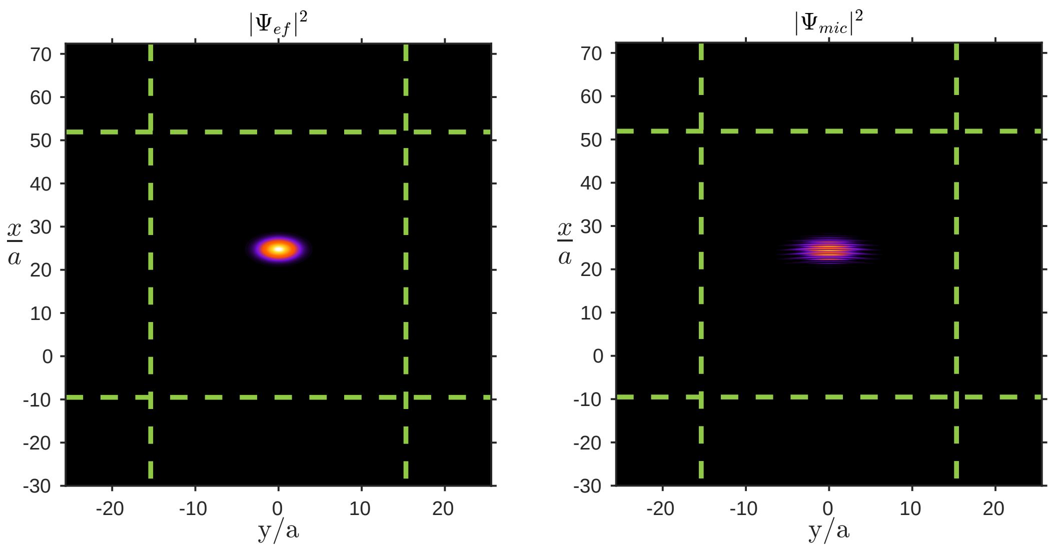


Figure 5.7: Left: Density plot of $|\Psi|^2$ obtained after 4000 time iterations for the effective medium model. Right: Density plot of $|\Psi|^2$ obtained after 4000 time iterations for the microscopic model.

Figure 5.7 displays a snapshot of the initial state propagating in a regime of extreme anisotropy, therefore a linear path is taken by the electronic state and there is no spread of the beamwidth.

5.3 Performance Evaluation

After validating the simulation results, it is now disclosed a comprehensive analysis on the performance of the devised solution. For this purpose there were considered four data sets and the simulation corresponds to a study of the time evolution of an initial state.

	Nodes (x direction)	**Nodes (y direction)**	**Total number of nodes**
Data Set 1	512	512	262,144
Data Set 2	1024	1024	1,048,576
Data Set 3	2048	2048	4,194,304
Data Set 4	4096	2048	8,388,608

Table 5.3: Data sets considered showing the dimensions of the grid for each data set.

	Simbol	**Vaule**
Space between nodes x direction	Δ_x	0.025
Space between nodes y directon	Δ_y	0.025
Time interval	Δ_t	0.0062
Number of time iterations	N_t	14000

Table 5.4: Constant parameters throughout the simulations.

In order to obtain a more accurate execution time, the following procedure was applied for each simulation:

- Execute the simulation in a loop fashion comprehending 10 iterations;
- Discard the best and worst execution times;
- The execution time of interest is the mean of the 8 remaining values.

5.3.1 OpenCL kernels evaluation

This section shows the results of an extensive study conducted to assess the impact on throughput performance of memory and work-group selection, in the execution of the kernels that comprise the update stage. Although, at first sight these two features may not seem related, the truth is that they are intertwined. To validate this claim there are shown some scenarios where it is investigated the performance for the use of global memory for different work-groups design.

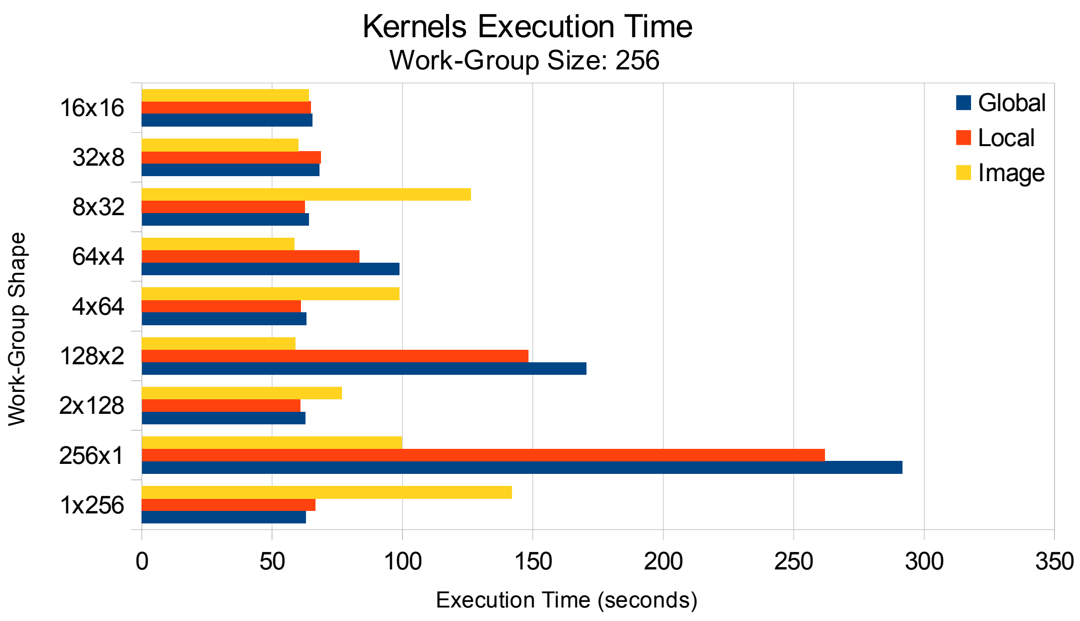


Figure 5.8: Kernels execution time (effective medium model) obtained for various work-group shapes. These results were acquired for data set 4, on platform 1.

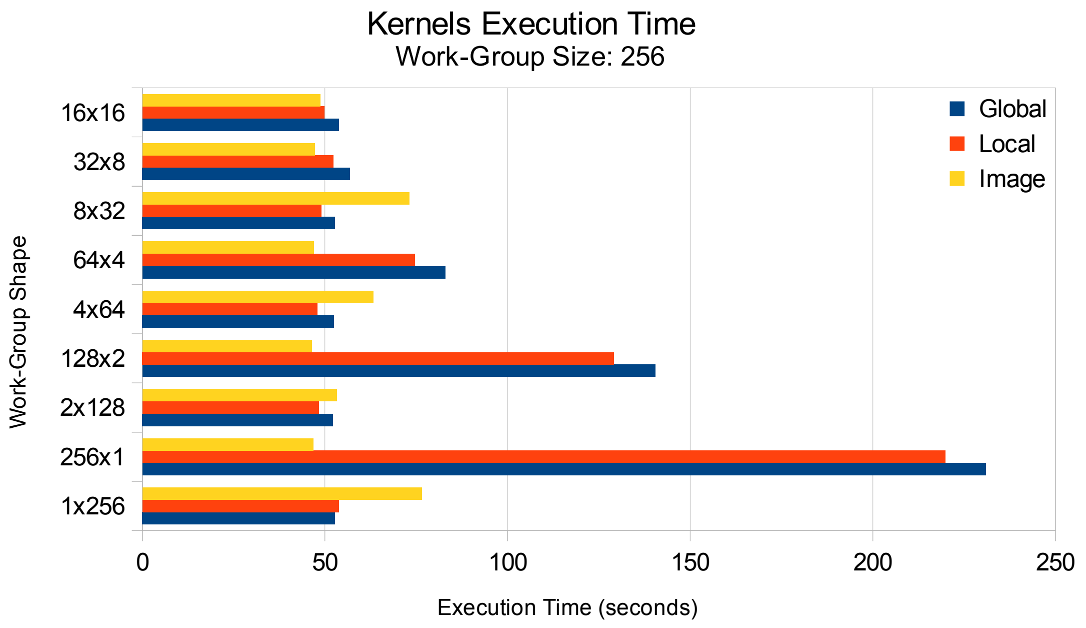


Figure 5.9: Kernels execution time (microscopic model) obtained for various work-group shapes. These results were acquired for data set 4, on platform 1.

The displayed results reveal that a row major selection for the work-group has a positive influence in global and local memory performance. This is explained by the scheme implemented by the Graphics Processing Unit (GPU)s to fetch data from memory. Also, as reviewed in subsection 2.2.1 consecutive address in memory are in the same column of the memory banks. Because memory in consecutive rows has a large stride, more operations will be needed to obtain the same amount of data, thus reducing the performance. Quite interestingly, the image version exhibits the opposite behavior. This is due to the fact that images rely on texture units that are designed to capture 2D spatial locality.

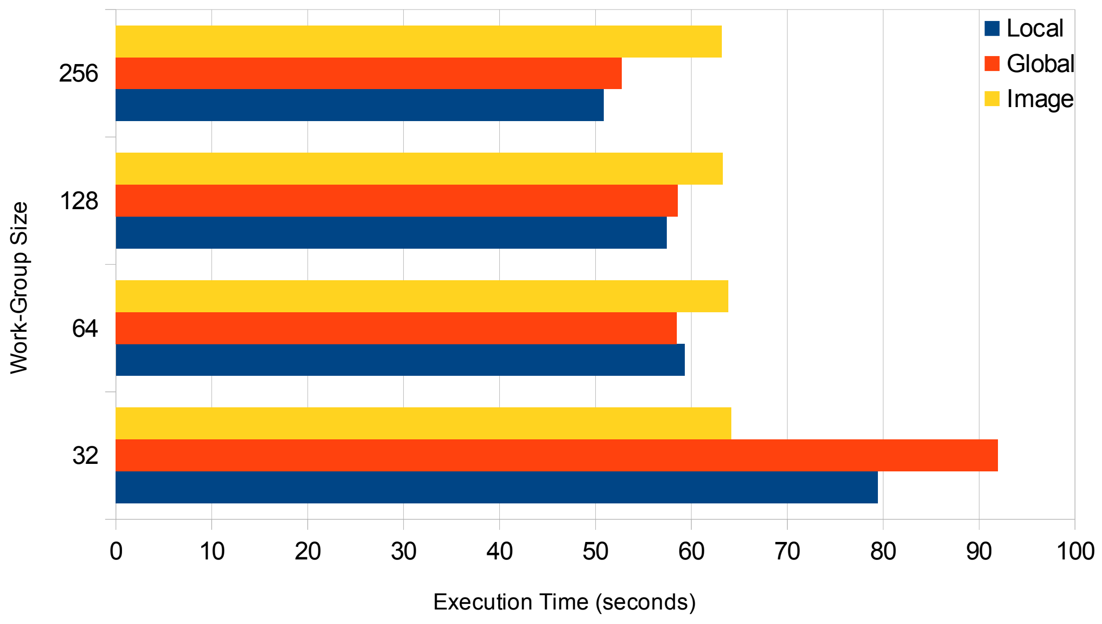


Figure 5.10: Work-group size influence in an AMD device (R9 280x).

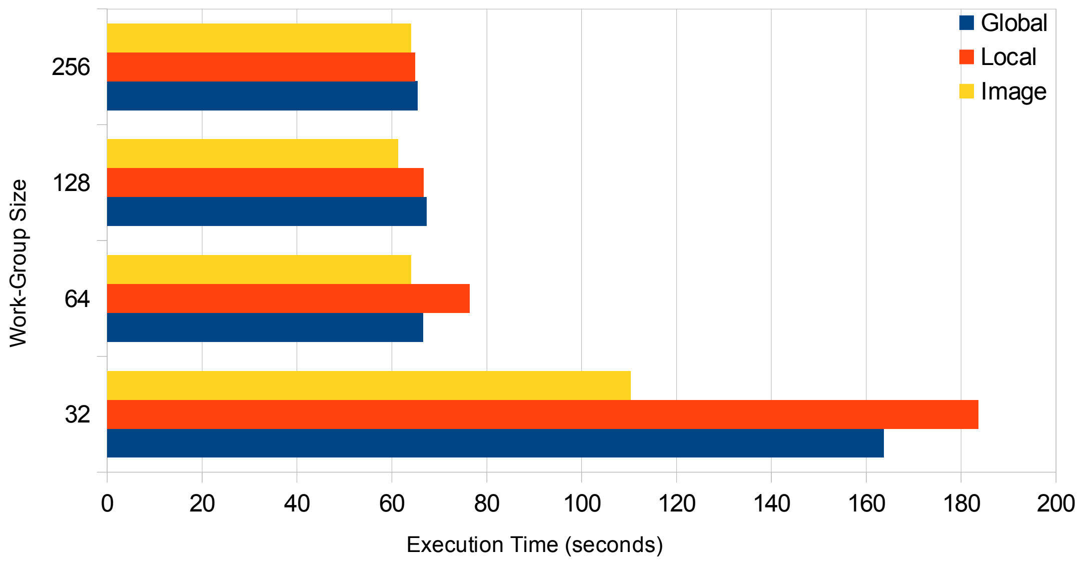


Figure 5.11: Work-group size influence in a Nvidia device (Tesla k40).

Another annotation that can be derived from these assessments is the selection of the work-group size. As reported before, all the devices are characterized by a minimum amount that must be scheduled in order to take advantage of the hardware. For example, Nvidia devices are designed to have at least 32 threads running in parallel per compute unit, although it is a best practice to feed the CUs with a number of threads larger than

32. On the other hand, on AMD devices these number of threads is 64. For that reason figures 5.10 and 5.11 show the effect of multiple work-group sizes in the execution time of the conceived kernels.

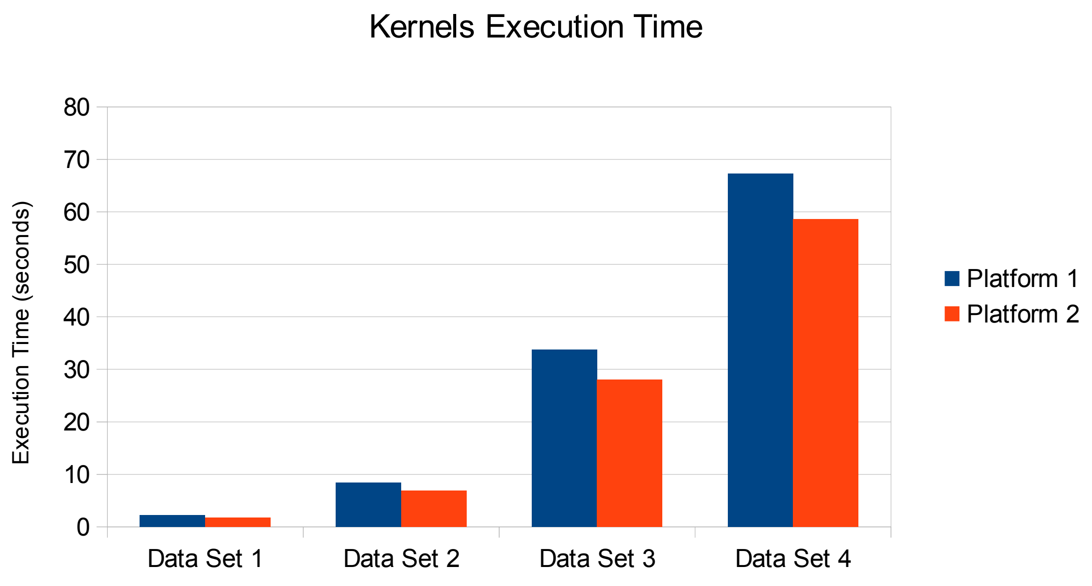


Figure 5.12: Kernels execution time (effective medium model), for each platform.

Another interesting analysis to conduct is the performance of OpenCL programs running on different devices. Because Nvidia has its on programming framework (CUDA), OpenCL programs may not take full advantage of the available hardware. For AMD devices this is not the case. They have adopted OpenCL as the programming framework, so that the hardware and drivers are optimized for OpenCL solutions. Hence, platform 2 performs better than platform 1 for all the simulated data sets. From all the performance metrics, the version that produces a better execution time is the texture version, although if the selection of the work-group shape is meticulously tunned, all three versions provide close execution times. Moreover, if we focus on the performance between the two models, it is seen that the microscopic model is the fastest approach. This is the result of less arithmetic operations being performed per node, as can be proved by the update equations for each model (3.6, 3.7, 3.8 and 3.9).

5.3.2 Overall performance

The differences between the Central Processing Unit (CPU) and GPU, as already depicted, have a substantial impact in the throughput performance of compute intensive tasks. Thereby, to prove these differences, figure 5.13 exposes the contrast between the two architectures. For this test was used the CPU and the GPU (AMD R9 280x) from platform 2.

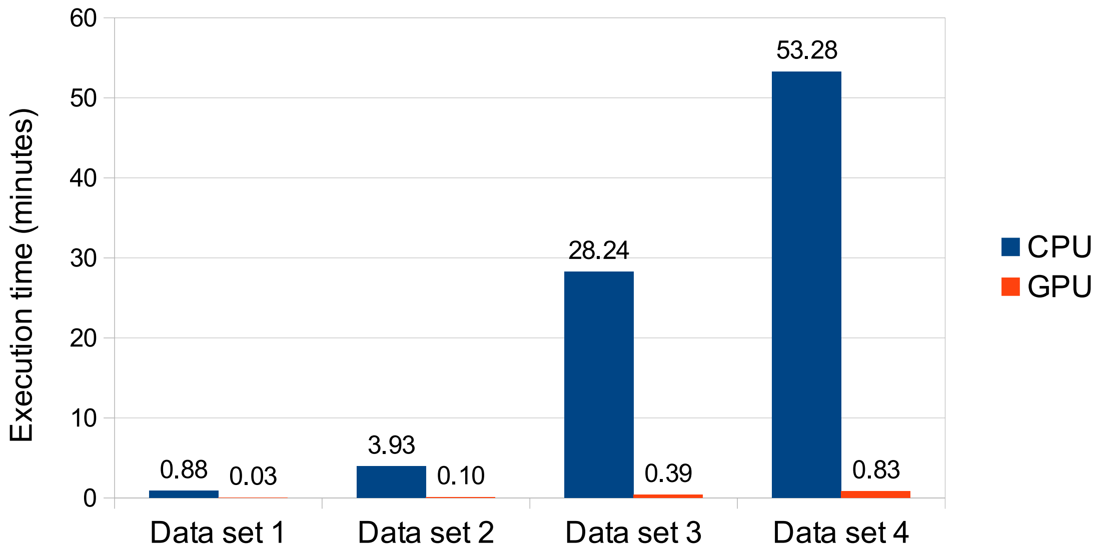


Figure 5.13: Throughput performance comparison between the CPU and GPU.

To conclude the performance evaluation, it is compared the devised solution based on the OpenCL standard with the equivalent Matlab and Mathematica versions.

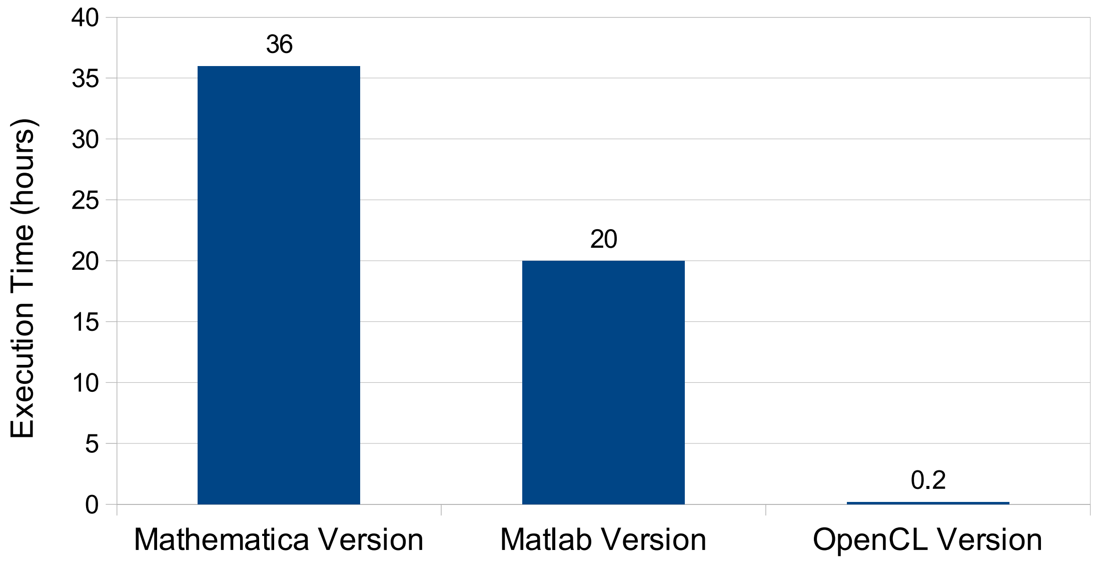


Figure 5.14: Overall performance compared between the three programing platforms.

OpenCL clearly demonstrates the ability to perform simulations at incredible speeds. The speed-up is 180x in comparison with Mathematica simulations and 100x when compared with Matlab simulations. It is also important to elucidate that these results comprehend the total length of a simulation, namely the execution of the setup stage and update stage. Also, regarding the update stage, there were considered store operations in disk of the wave function at each 1000 time iterations. This leads to throughput performance downgrade because there are memory transfers between the device and host. Furthermore, write operations to disk represent a major contribution in the degradation of the throughput performance.

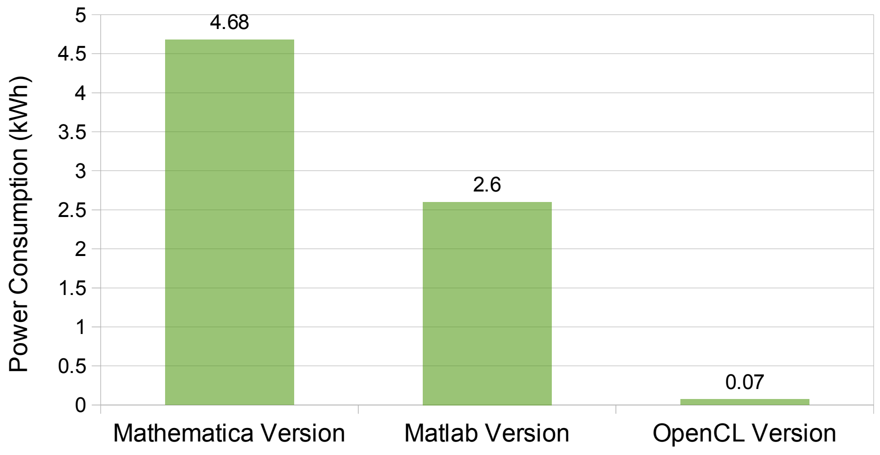


Figure 5.15: Overall performance compared between the three programing platforms.

Nowadays, energy consumption awareness is an increasingly issue at global scale. Therefore, the search for efficient solutions should be considered. This work also represents improvements in this field, as can be emphasized by figure 5.15. The solution devised represents energy savings on the order of 67x compared with Mathematica and 38x comparing with matlab version.

5.4 Summary

In this chapter, a detailed review of the proposed solution was presented and discussed. This analysis was conducted in terms of the simulation results obtained and performance metrics. Consequently, a significant reduction in execution time was achieved in all the considered scenarios. As for the simulation expected results, the implemented kernels proved to be a useful tool to study the behavior of the electron wave propagation. Also, an inspection on how the different versions of the kernels, resulting from the distinct memory hierarchy utilization, was conducted in order to assess the performance impact on the execution time of the kernels. From this assessment it is clear that the execution time of a kernel is influenced by the work-group selection and type of memory spaces used. A multi-device implementation for accelerating the two models was also a target of analysis when compared with single-device implementation providing major improvements in the execution time, because it enables the execution of both models in parallel.

6 Conclusions

Contents

This thesis purpose was to accelerate a recently proposed method to study the time evolution of electron waves in graphene superlattices, by designing an optimized and potable parallel solution. This was achieved by using the Open Computing Language (OpenCL) framework.

The starting point was the realization of a Matlab version, while the theoretical background was assimilated. From this study two approaches were identified: Microscopic and Effective Medium approach. The Matlab implemented version also provided a base for performance evaluation. The parallelization strategy was devised based on the study of the available data and task-parallelism opportunities. The newly proposed solution helped developing the proposed method by enabling the simulation of extremely large meshes and for a high number of iterations.

Exploring the inherent parallelism in algorithms has proven to be a powerful tool to develop and accelerate the rate that science is produced. OpenCL gives the programmer the right amount of resources to explore and design fully functional programs ready to adapt and take advantages of the available processors/hardware. It is seen that combining the right tools and programming techniques the expected result is surpassed.

6.1 Future Work

The developed solution proves to be a useful and powerful tool to simulated electron wave propagation problems on graphene superlattices. Although outstanding performance improvements have already been observed, there are other aspects that can be further explored:

- Test the developed kernels in other devices such as the Field-Programmable Gate Array (FPGA), and lower power processors, and study the impact on the energy-consumption performance compared with mainstream CPUs and GPUs;

- Another base for improvements comprehends the simulation analysis procedure. When dealing with compute-intensive simulations, typically these are performed and data is stored in disk. Just then, the analysis of the data is carried out. Thus, there is the possibility of integrating the Open Graphics Library (OpenGL) Application Programming Interface (API), giving feedback in real-time to the user performing the simulation, by showing all the metrics, figures and charts necessary for a comprehensive data inspection.

Bibliography

[1] A. H. Castro Neto, F. Guinea, N. M. R. Peres, K. S. Novoselov, and A. K. Geim, "The electronic properties of graphene," Rev. Mod. Phys., vol. 81, pp. 109–162, Jan 2009.

[2] A. K. Geim and K. S. Novoselov, "The rise of graphene," Nature Materials, vol. 6, no. 3, pp. 183–191, 2007.

[3] D. Fernandes, M. Rodrigues, G. Falcão, and M. G. Silveirinha, "Time evolution of the electron wave in graphene supperlattices," Physical Review B, 2015 (Submited).

[4] G. Moore, "Cramming more components onto integrated circuits," Proceedings of the IEEE, vol. 86, pp. 82–85, Jan 1998.

[5] J. G. Revilla, E. F. Barry, P. R. Marchand, and G. G. Pechanek, "Methods and apparatus to dynamically reconfigure the instruction pipeline of an indirect very long instruction word scalable processor," Apr. 10 2001. US Patent 6,216,223.

[6] O. Aciiçmez, Ç. K. Koç, and J.-P. Seifert, "On the power of simple branch prediction analysis," in Proceedings of the 2nd ACM symposium on Information, computer and communications security, pp. 312–320, ACM, 2007.

[7] V. Popescu, M. A. Schultz, G. A. Gibson, J. E. Spracklen, and B. D. Lightner, "Processor architecture having out-of-order execution, speculative branching, and giving priority to instructions which affect a condition code," Apr. 29 1997. US Patent 5,625,837.

[8] M. Kowarschik and C. Weiß, "An overview of cache optimization techniques and cache-aware numerical algorithms.," in Algorithms for Memory Hierarchies (U. Meyer, P. Sanders, and J. F. Sibeyn, eds.), vol. 2625 of Lecture Notes in Computer Science, pp. 213–232, Springer, 2002.

[9] R. Viswanath, V. Wakharkar, A. Watwe, V. Lebonheur, M. Group, and I. Corp, "Thermal performance challenges from silicon to systems," 2000.

[10] L. Gwennap, "Sandy bridge spans generations," Microprocessor Report, vol. 9, no. 27, pp. 1–8, 2010.

[11] C. Cantrell, "Arcade space invaders." `http://http://www.computerarcheology.com/wiki/wiki/Arcade/SpaceInvaders/`, January 2014. Accessed: February 2015.

[12] D. Luebke and G. Humphreys, "How gpus work," Computer, vol. 40, pp. 96–100, Feb 2007.

[13] J. Owens, "Gpu architecture overview," in ACM SIGGRAPH, vol. 1, pp. 5–9, 2007.

[14] J. Nickolls and W. Dally, "The gpu computing era," Micro, IEEE, vol. 30, pp. 56–69, March 2010.

[15] M. Garland, S. Le Grand, J. Nickolls, J. Anderson, J. Hardwick, S. Morton, E. Phillips, Y. Zhang, and V. Volkov, "Parallel computing experiences with cuda," Micro, IEEE, vol. 28, pp. 13–27, July 2008.

[16] W.-M. Hwu, C. Rodrigues, S. Ryoo, and J. Stratton, "Compute unified device architecture application suitability," Computing in Science and Engineering, vol. 11, no. 3, pp. 16–26, 2009.

[17] J. D. Owens, M. Houston, D. Luebke, S. Green, J. E. Stone, and J. C. Phillips, "Gpu computing," Proceedings of the IEEE, vol. 96, no. 5, pp. 879–899, 2008.

[18] "NVIDIA's Next Generation CUDA Compute Architecture: Kepler GK110," 2012.

[19] A. R. G. Technology, "AMD Graphics Cores Next (GCN) Architecture White Paper," June 2012.

[20] NVIDIA, NVIDIA CUDA Programming Guide 6.5. August 2014.

[21] AMD, AMD Accelerated Parallel Processing OpenCL User Guide. August 2013.

[22] M. Harris, S. Sengupta, and J. D. Owens, "Parallel prefix sum (scan) with cuda," GPU gems, vol. 3, no. 39, pp. 851–876, 2007.

[23] "Khronos group." `https://www.khronos.org/opencl/`. Accessed: December 2014.

[24] J. E. Stone, D. Gohara, and G. Shi, "Opencl: A parallel programming standard for heterogeneous computing systems," Computing in science & engineering, vol. 12, no. 1-3, pp. 66–73, 2010.

[25] G. Falcao, V. Silva, L. Sousa, and J. Andrade, "Portable ldpc decoding on multicores using opencl [applications corner]," Signal Processing Magazine, IEEE, vol. 29, no. 4, pp. 81–109, 2012.

[26] P. Du, R. Weber, P. Luszczek, S. Tomov, G. Peterson, and J. Dongarra, "From cuda to opencl: Towards a performance-portable solution for multi-platform gpu programming," Parallel Computing, vol. 38, no. 8, pp. 391–407, 2012.

[27] K. Karimi, N. G. Dickson, and F. Hamze, "A performance comparison of cuda and opencl," arXiv preprint arXiv:1005.2581, 2010.

[28] K. O. W. Group., The OpenCL Specification version 1.1 revision 44. 2011.

[29] C. de la Lama, P. Toharia, J. Bosque, and O. Robles, "Static multi-device load balancing for opencl," in Parallel and Distributed Processing with Applications (ISPA), 2012 IEEE 10th International Symposium on, pp. 675–682, July 2012.

[30] L. Chen, O. Villa, S. Krishnamoorthy, and G. Gao, "Dynamic load balancing on single- and multi-gpu systems," in Parallel Distributed Processing (IPDPS), 2010 IEEE International Symposium on, pp. 1–12, April 2010.

[31] A. F. Oskooi, D. Roundy, M. Ibanescu, P. Bermel, J. D. Joannopoulos, and S. G. Johnson, "Meep: A flexible free-software package for electromagnetic simulations by the fdtd method," Computer Physics Communications, vol. 181, no. 3, pp. 687–702, 2010.

[32] D. M. Sullivan, Electromagnetic simulation using the FDTD method. John Wiley & Sons, 2013.

[33] C. Dean, A. Young, I. Meric, C. Lee, L. Wang, S. Sorgenfrei, K. Watanabe, T. Taniguchi, P. Kim, K. Shepard, et al., "Boron nitride substrates for high-quality graphene electronics," Nature nanotechnology, vol. 5, no. 10, pp. 722–726, 2010.

[34] Y. P. Bliokh, V. Freilikher, S. Savel'ev, and F. Nori, "Transport and localization in periodic and disordered graphene superlattices," Physical Review B, vol. 79, no. 7, p. 075123, 2009.

[35] M. G. Silveirinha and N. Engheta, "Effective medium approach to electron waves: Graphene superlattices," Phys. Rev. B, vol. 85, p. 195413, May 2012.

A

Appendix A

Contents

A.1 Propagation of stationary electron waves

This section shows the update equations used in the simulations of the propagation of stationary electron waves for both models. The only difference, from equations (3.6, 3.7, 3.8 and 3.9) is the addition of the term $v_F \Delta_t j_{1,p,q}$ in each equation.

A.1.1 Microscopic Model

$$\begin{aligned}\Psi^{n+1}_{1,p,q}\left(1-\frac{V_{p,q}}{2\hbar i}\Delta_t\right) &= \Psi^{n}_{1,p,q}\left(1+\frac{V_{p,q}}{2\hbar i}\Delta_t\right)+v_F\Delta_t j_{1,p,q}\\ &-v_F\Delta_t\left[\left(\frac{1}{2\Delta_x}-i\frac{1}{2\Delta_y}\right)\Psi^{n+\frac{1}{2}}_{2,p+\frac{1}{2},q+\frac{1}{2}}-\left(\frac{1}{2\Delta_x}+i\frac{1}{2\Delta_y}\right)\Psi^{n+\frac{1}{2}}_{2,p-\frac{1}{2},q+\frac{1}{2}}+\right.\\ &\left.+\left(\frac{1}{2\Delta_x}+i\frac{1}{2\Delta_y}\right)\Psi^{n+\frac{1}{2}}_{2,p+\frac{1}{2},q-\frac{1}{2}}-\left(\frac{1}{2\Delta_x}-i\frac{1}{2\Delta_y}\right)\Psi^{n+\frac{1}{2}}_{2,p-\frac{1}{2},q-\frac{1}{2}}\right]\end{aligned} \tag{A.1}$$

$$\begin{aligned}\Psi^{n+1}_{2,p+\frac{1}{2},q+\frac{1}{2}}\left(1-\frac{V_{p+\frac{1}{2},q+\frac{1}{2}}}{2\hbar i}\Delta_t\right) &= \Psi^{n}_{2,p+\frac{1}{2},q+\frac{1}{2}}\left(1+\frac{V_{p+\frac{1}{2},q+\frac{1}{2}}}{2\hbar i}\Delta_t\right)+v_F\Delta_t j_{2,p,q}\\ &-v_F\Delta_t\left[\left(\frac{1}{2\Delta_x}+i\frac{1}{2\Delta_y}\right)\Psi^{n}_{1,p+1,q+1}-\left(\frac{1}{2\Delta_x}-i\frac{1}{2\Delta_y}\right)\Psi^{n}_{1,p,q+1}+\right.\\ &\left.+\left(\frac{1}{2\Delta_x}-i\frac{1}{2\Delta_y}\right)\Psi^{n}_{1,p+1,q}-\left(\frac{1}{2\Delta_x}+i\frac{1}{2\Delta_y}\right)\Psi^{n}_{1,p,q}\right]\end{aligned} \tag{A.2}$$

A.1.2 Effective medium Model

$$\begin{aligned}\Psi^{n+1}_{1,p,q}\left(1-\frac{V_{ef,p,q}}{2\hbar i}\Delta_t\right) &= \Psi^{n}_{1,p,q}\left(1+\frac{V_{ef,p,q}}{2\hbar i}\Delta_t\right)+v_F\Delta_t j_{1,p,q}\\ &-v_F\Delta_t\left[\left(\frac{1}{2\Delta_x}-i\frac{\frac{\chi_{p,q}}{2}+\frac{\chi_{p+\frac{1}{2},q+\frac{1}{2}}}{2}}{2\Delta_y}\right)\Psi^{n+\frac{1}{2}}_{2,p+\frac{1}{2},q+\frac{1}{2}}-\left(\frac{1}{2\Delta_x}+i\frac{\frac{\chi_{p,q}}{2}+\frac{\chi_{p-\frac{1}{2},q+\frac{1}{2}}}{2}}{2\Delta_y}\right)\Psi^{n+\frac{1}{2}}_{2,p-\frac{1}{2},q+\frac{1}{2}}+\right.\\ &\left.+\left(\frac{1}{2\Delta_x}+i\frac{\frac{\chi_{p,q}}{2}+\frac{\chi_{p+\frac{1}{2},q-\frac{1}{2}}}{2}}{2\Delta_y}\right)\Psi^{n+\frac{1}{2}}_{2,p+\frac{1}{2},q-\frac{1}{2}}-\left(\frac{1}{2\Delta_x}-i\frac{\frac{\chi_{p,q}}{2}+\frac{\chi_{p-\frac{1}{2},q-\frac{1}{2}}}{2}}{2\Delta_y}\right)\Psi^{n+\frac{1}{2}}_{2,p-\frac{1}{2},q-\frac{1}{2}}\right]\end{aligned} \tag{A.3}$$

$$\Psi^{n+1}_{2,p+\frac{1}{2},q+\frac{1}{2}}\left(1-\frac{V_{ef,p+\frac{1}{2},q+\frac{1}{2}}}{2\hbar i}\Delta_t\right)=\Psi^{n}_{2,p+\frac{1}{2},q+\frac{1}{2}}\left(1+\frac{V_{ef,p+\frac{1}{2},q+\frac{1}{2}}}{2\hbar i}\Delta_t\right)+v_F\Delta_t j_{2,p,q}$$
$$-v_F\Delta_t\left[\left(\frac{1}{2\Delta_x}+i\frac{\frac{\chi_{p+\frac{1}{2},q+\frac{1}{2}}}{2}+\frac{\chi_{p+1,q+1}}{2}}{2\Delta_y}\right)\Psi^{n}_{1,p+1,q+1}-\left(\frac{1}{2\Delta_x}-i\frac{\frac{\chi_{p+\frac{1}{2},q+\frac{1}{2}}}{2}+\frac{\chi_{p,q+1}}{2}}{2\Delta_y}\right)\Psi^{n}_{1,p,q+1}+\right.$$
$$\left.+\left(\frac{1}{2\Delta_x}-i\frac{\frac{\chi_{p+\frac{1}{2},q+\frac{1}{2}}}{2}+\frac{\chi_{p+1,q}}{2}}{2\Delta_y}\right)\Psi^{n}_{1,p+1,q}-\left(\frac{1}{2\Delta_x}+i\frac{\frac{\chi_{p+\frac{1}{2},q+\frac{1}{2}}}{2}+\frac{\chi_{p,q}}{2}}{2\Delta_y}\right)\Psi^{n}_{1,p,q}\right] \tag{A.4}$$

The term j is the source of the form:

$$j=j_0e^{ik_yy}e^{-i\omega_0t} \tag{A.5}$$