VNiVERSiDAD
Đ SALAMANCA

Facultad de Ciencias
Departamento de Física Aplicada

Spintronic Micromagnetic Simulations Using Parallel Computations

Doctor thesis

David Filipe Coelho de Almeida Aurélio

June 2013

D. Luis Torres Rincón profesor catedrático y **D. Eduardo Martínez Vecino** profesor Contratado Doctor, ambos miembros del Área de Electromagnetismo del Departamento de Física Aplicada de la Universidad de Salamanca,

CERTIFICAN

Que la presente Memoria, titulada "*Spintronic Micromagnetic Simulations Using Parallel Computations*" ha sido realizada bajo su dirección en el Área de Electromagnetismo del Departamento de Física Aplicada de la Universidad de Salamanca por **D. David Filipe Coelho de Almeida Aurélio**, y constituye su Tesis para optar al Grado de Doctor en Física.

En Salamanca, 20 de Junio de 2013

D. David Aurélio

D. Luis Torres Rincón
Profesor Catedrático

D. Eduardo Martínez Vecino
Profesor Contratado Doctor

Acknowledgments

It has finally arrived the day to defend my PhD thesis. It is an exciting time for me, especially when considering that I was not initially planning to do a PhD when I was about to finish my degree. However, for other reasons I found myself in the beautiful city of Salamanca and met the professors Luis Torres and Luis Lopez-Diaz, which offer me a one-year research project. Wanting to stay in Salamanca I accepted and it was the beginning of the venture that happily led me to this day. The support, guidance and friendship over that first year of both Luis Torres and Luis Lopez-Diaz, convince me to take the endeavor of making the PhD. For that reason my first thanks goes to them, thank you for everything. Shortly after, a new professor joined the micromagnetic research group, Eduardo Martinez, which together with Luis Torres became my thesis directors. I could not be more honored, because each of them taught me a lot, and made possible the PhD work that I am to defend. Thank you, it as has been an incredible learning experience. It is also hard not to mention everyone at the Electrodynamics hallway, since at one time or another they have help me and showed themselves to be wonderful people creating a really good working environment. In fact, a special thank you to all at the Applied Physics department, because everyone in it as contributed to the incredible work group in which I found myself. Also a special thank you to Prof. Pilar García Estevez, for all the help with the program and bureaucracies during the first years of the PhD.

Living away from your family and friends is not easy, however I have been lucky enough to have met people that made me fell at home during my stay in Salamanca. The first one was the rock climber Elzbieta, which was my work companion for most of my PhD. Through her I met cyclist Ana, another PhD student but working in electro stuff. Together we started a squash mini-club once a week, which was awesome. But to be true the sport was an excuse to the wonderful weekly dinners that started to be a tradition. It was not long for more people to join the "club", Javier the informatic, who I owe a great debt of gratitude has he became my CUDA teacher, fundamental to my thesis work, Erick the lawyer from Costa Rica, Diego the political scientist also from that beautiful country, Ana's also cycler brother Ignácio, the crazy Real Madrid fan José, and most recently physics-informatic Sergio and the French man Jeff, to all of you a thank you from the bottom of my heart. It is difficult to say how important your presence has been for me, since at one moment or another you

made me fell totally at home in Salamanca, and it is sad that due to different reasons some of you have left this city. I will always remember you no matter where you are or where I will be.

One of the cool things of making a PhD is that you get to travel to different places and in particular a have to single out Messina, Italy. It was there that I made my three months period abroad, under the tutelage of professor Giovanni Finocchio, and thus a thank you to him for taking me into the research group, which enriched not only my research abilities but also cultural ones, since I also got to learn some Italian. Also many thanks for making me feel welcomed there to Vito, Anna, Alesandro, Francesco, Ricardo and professor Azzerboni. It was a really good experience that enriched me in many ways. I also have to thank everyone at the tennis club Circoletto dei Laghi, which received me stupendously in friendship and support for my other passion that is tennis, with a special hug to my little brother Francesco. And of course a thank you also to my house companions Emanuele, Carmelo and Peppe.

Finely I would also like to thank to two different families that made me fell part of their own families. The Criado family that shares my affection for tennis, and the Blanco family, thank you for everything and continue to be the lovely people that you are. I also want to thank my family for all the support that they have given me throughout the years, especially during this period, because it is not easy being away, a huge hug to my parents and to my brother. Lastly to you Elena thank you, I have no words to describe everything that you mean to me.

David Aurélio

Salamanca, 2013

Abstract

The work described in this thesis discusses both the implementation of a simulation micromagnetic code that harnesses the power of parallel computing and the study of different phenomena in micromagnetic devices via simulations.

The implementation of the simulation code is performed by first studying the theoretical formalism of Micromagnetism and the numeric techniques that allow for the simulation of different devices, under that formalism. An existing sequential Fortran code is used as a basis to develop the parallel code, which is written in a recently developed language by NVIDIA named CUDA. This uses graphic processing units (GPUs) to perform highly parallel computations instead of the usual central processing units (CPUs), which allowed for speed-ups of up two orders of magnitude at a tenth of the cost of an equivalent super-computer CPU cluster. The developed parallel-GPU code includes the usual micromagnetic field contributions (exchange, anisotropy, magnetostatic, Zeeman, Oersted) as well as the thermal field and the spin-torque interaction on both, current perpendicular to plane (CPP) and current in-plane (CIP) devices. In particular for CPP devices the magnetization dynamics of both the usually pinned and free layers of a spin-vale or tunnel junction (MTJ) is considered, accounting for the spin-torque of both layers on each other (back-torque).

Different micromagnetic studies are presented, which involve the study of the magnetization switching in magnetic tunnel junctions, with and without the effect of temperature, which also show some of the limitations of sequential programming that lead to the will of developing a faster and more efficient parallel micromagnetic code. The developed parallel code, being able to tackle large temporal and/or large spatial simulations is used to rigorously study vortex oscillation frequencies in spin-valves (simulation times of 10^{-5} s), and to study the domain wall dynamics in long ferromagnetic stripes (in the order of 10^{6} computational cells).

KEYWORDS – Micromagnetics, spin-torque, MTJ, spin-valve, domain wall, parallel programming, GPU.

Resumen

En el trabajo descrito por esta tesis se incluye tanto la implementación de un código de simulación micromagnética que aprovecha la potencia de la computación en paralelo, como el estudio de diferentes fenómenos en dispositivos micromagnéticos.

La implementación del código de simulación se inició mediante el estudio del formalismo teórico del Micromagnetismo y de las técnicas numéricas que permiten la simulación de diferentes dispositivos. Un código secuencial de Fortran se utiliza como base para el desarrollo del código paralelo, escrito en un lenguaje recientemente desarrollado por NVIDIA (CUDA). Este sistema utiliza unidades de procesamiento gráfico (GPU) para realizar los cálculos en paralelo, en lugar de las habituales unidades centrales de procesamiento (CPU), permitiendo un incremento de la velocidad de simulación de hasta dos órdenes de magnitud, a una décima parte del costo de un sistema de supercomputación equivalente (cluster de CPUs). El código paralelo-GPU desarrollado incluye las contribuciones habituales micromagnéticas (intercambio, anisotropía, magnetostática, Zeeman, Oersted), así como el campo térmico y la interacción de par de espín, tanto para dispositivos con corrientes perpendiculares al plano (CPP) o corrientes en el plano (CIP). En particular para los dispositivos CPP se considera la dinámica de la magnetización de las capas fija y libre de una válvula de espín o unión de efecto túnel (MTJ). Se tiene en cuenta el par de espín de ambas capas (*back-torque*).

Se presentan diferentes estudios micromagnéticos, que implican el estudio de la inversión de la magnetización en uniones de efecto túnel, sin y con el efecto de la temperatura. Estos estudios muestran algunas de las limitaciones de la programación secuencial que llevaron a la idea de desarrollar un código micromagnético paralelo más eficiente. El código paralelo desarrollado, capaz de realizar simulaciones que implican grandes ventanas temporales y/o grandes dimensiones espaciales se utiliza para estudiar rigurosamente las frecuencias de oscilación de vórtices en válvulas de espín (tiempos de simulación del 10^{-5} s), y para estudiar la dinámica de paredes de dominio en largas tiras ferromagnéticos (del orden de 10^{6} células computacionales).

PALAVRAS CLAVE – Micromagnetismo, par de espín, MTJ, válvulas de espín, paredes de dominio, programación paralela, GPU.

Contents

List of abbreviations

DW	Domain wall
GPU	Graphic processing unit
CPU	Central processing unit
SI units	International system of units
LLG	Landau-Lifshitz-Gilbert equation
STT	Spin transfer torque
STO	Spin torque oscillators
CUDA	Compute Unified Device Architecture
RAM	Random access memory
FD	Finite difference
FE	Finite element
GMR	Giant magneto-Resistance
AMR	Anisotropic magneto-resistance
TMR	Tunnelling magneto-resistance.
MTJ	Magnetic tunnel junction
MRAM	Magneto-resistive random access memory
DW	Domain wall
STNO	Spin-torque nano-oscillator
CPP	Current perpendicular to plane
CIP	Current in-plane
FFT	Fast Fourier transform
ALU	Arithmetic logic unit
ID	Identification
CUFFT	CUDA Fast Fourier Transform library
CUDPP	CUDA Data Parallel Primitives library
μMAG	Micromagnetic Modeling Activity Group
Py	Permalloy
BC	Boundary conditions
NUCD	Non-uniform current distribution
UCD	Uniform current distribution
P	parallel
AP	anti-parallel

PL – pinned layer

FL – free layer

MSMT – Micromagnetic Spectral Mapping Technique

SR – sweep rate

PSTT – perpendicular spin-torque term

TW – transverse wall

BPW – Bloch-point wall

FWHM – Full width at half maximum

List of symbols

μ_B – Bohr's magneton

g – Landé factor

j – current density

η – spin-polarization factor

$\mathcal{P}$ – polarization function

M_S – magnetization of saturation

d – layer thickness

e – electron's charge

Γ_{STT} – spin transfer torque

γ_0 – gyromagnetic ratio

α – Gilbert damping parameter

β – non-adiabatic parameter

μ_0 – vacuum magnetic permeability

k_B – Boltzmann constant

T – temperature

A – exchange constant

K – anisotropy constant

fN – number of first neighbors

N – total number of cells

a – lattice constant

M_S – saturation magnetization

M_{PS} – saturation magnetization of the assumed polarizing pinned layer

R_P – parallel state resistance

R_{AP} – anti-parallel state resistance

f – frequency

T – temperature

T_{bath} – bath temperature

$\boldsymbol{v}_{sp\text{-}drift}$ – spin-drift velocity (maximum theoretical domain wall velocity)

Δt – time window

δt – time step

List of figures

List of tables

1 Introduction

The ongoing development in all fields of science, driven by Humanity's unquenchable thirst for knowledge, has brought incredible technological advancements that make not only Man's life easier and healthier but also opens the door to new and exciting branches of science. The work presented by this thesis tries to humbly leave its footstep in the never-ending marathon, which is Humanity's understanding of the wonderful universe it inhabits.

Micromagnetism is the particular branch of solid-state physics in which this work focuses on. It is usually defined as a mesoscopic formalism, since it uses numeric-physical models to describe the magnetization dynamics of ferromagnetic materials at the micro and nano-scale. In other words, it works at a scale larger than the atomic one, in which quantum considerations would be needed, but low enough that one can see the dynamics of domains formed by thousands, if not millions, magnetic spins. As a result, through this medium it is possible to gain a great insight into new fundamental physical effects, like the spin transfer torque (STT) and its applications into possible new devices like spin torque oscillators, and also into the most recent interest into the spin-Hall and spin-Seebeck effects. Actually, the previously mentioned phenomena gave birth to a new branch of research, which is known as spintronics (neologism for devices based on spin transport electronics).

The great driving force of this field of study is not only the insight it gives into fundamental physics, but also the fact that the study of those phenomena allows for the creation of new, more efficient and faster devices. One example of such applications comes directly from a strong consumer product, which is the magnetic hard drive. This device saw an incredible development due to the magneto-resistance study of magnetic materials, which allowed a memory density increase from 2,4Gb per sq-inch in 1997 up to 70Gb per sq-inch in 2007. Such development was driven by both the evolution of the miniaturization techniques and the understanding of the magneto-resistance phenomenon. This phenomenon is seen in thin film structures, such as spin-valves (Fig. 1), which are composed of alternating layers of ferromagnetic and non-magnetic conductor materials that "see" its electrical resistance change significantly under the application of an external magnetic field. The discovery of this phenomenon was recognized in 2007 through the Nobel Prize in physics given to Albert Fert and Peter Grünberg for the Giant Magneto-Resistance (GMR) discovery in 1988, [1],[2]. Around the year 2006 the spin-valve read heads of hard drives that used the GMR effect, started to be replaced by magnetic tunnel junctions (MTJs) due to its tunnelling magneto-resistance (TMR) effect. Unlike spin-valves, in a MTJ a thin insulator rather than a conductor separates the ferromagnetic materials. This allows for a smaller and more sensitive device to the magnetic changes present in hard drives, and thus lead to a further increase of the memory density.

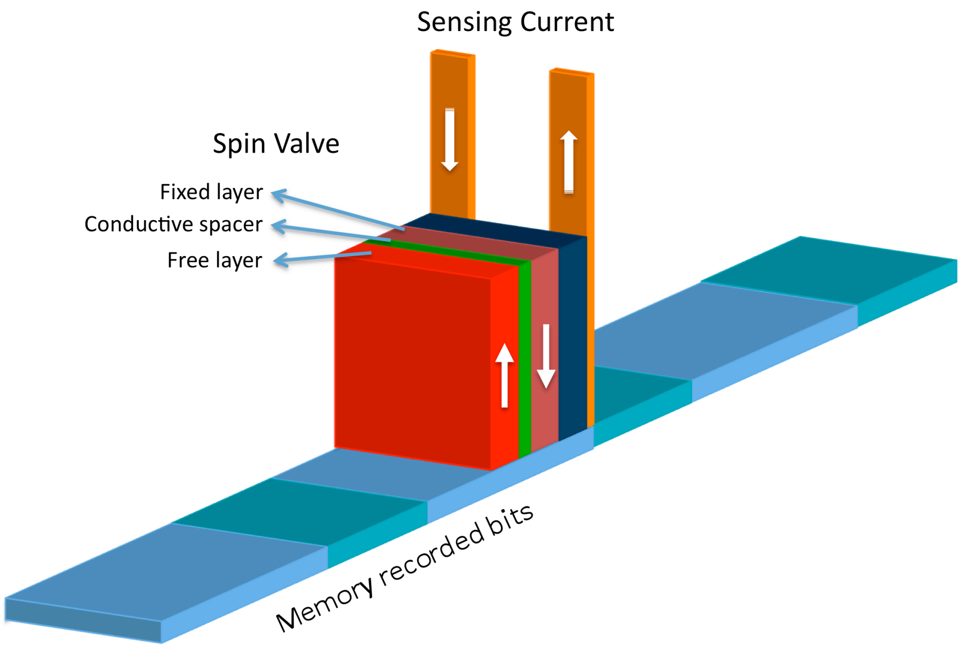

Fig. 1 – Spin-valve read head representation. When the fixed and free layer have their averaged magnetization in the anti-parallel configuration, as in the picture, the sensing current reads a high resistance state, whereas if they are in parallel it would read a low resistance state. This is the principle used in order to distinguish between the 0 and 1 bit logic values recorded in the memory track.

Another interesting finding that came from the study of fundamental micromagnetism was the concept of spin-transfer, as a result of the works of John Slonczewski and Luc Berger in 1996 [3],[4]. This phenomenon states that if a current passes through a thick enough magnetic material, its electrons become spin-polarized along the direction of the magnetization of that material. Such a current then exerts a torque onto a subsequent magnetic thin film by the transfer of spin angular momentum, which in turn influences its magnetic structure. This spin transfer torque effect immediately lead scientists to new ideas for future devices, due to the ability of manipulating the magnetization directly by using currents. Up to this point, in order to invert the magnetization, of for example the free layer of a spin-valve, large magnetic fields had to be generated in its vicinity. However with the STT effect one can invert the magnetization by applying the current directly through the device, which brought up the idea of current controlled magneto-resistive random access memory (ST-MRAM) (Fig. 2). This solution would not only decrease the power needed for its operation but also reduce the read and write times, which would make it a viable non-volatile memory that does not wear out. In fact in November of 2012, the company Everspin launched the first commercial ST-MRAM chips [5], bringing forth what will most probable be a booming new era of spin-torque memories.

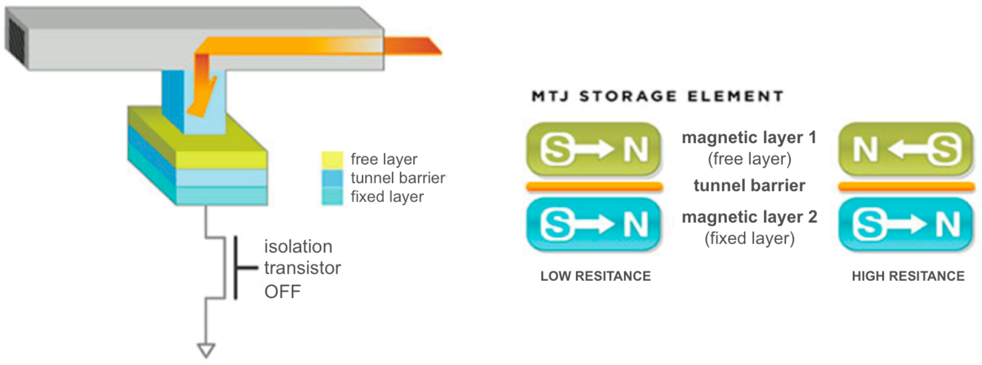

Fig. 2 – Magnetic tunnel junction MRAM working principle, [5]

More devices were thought up due to the STT, which are based on the propagation of spin currents by either conducting electrons or spin-waves. One of which was again in the area of magnetic storage, in the form of the racetrack memory (Fig. 3) proposed by Parkin [6] that again promises to be a more efficient and fast type of memory. This type of memory is based on moving domain walls (DWs), which is a region where the magnetization gradually changes from one given direction to another, in long magnetic strips [6]-[9]. Spin-torque nano-oscillators (STNOs) are also another interesting prospect of a device, since multiple devices that depend on oscillator principles could be further miniaturized. The idea for such a device came from the now well-established knowledge that a spin-polarized electric current injected into a ferromagnetic layer through a nano-contact exerts a torque on the magnetization, which eventually leads to microwave frequency precessions detectable through the magneto-resistance effect [10]-[12]. Since the amplitude of such spin-waves is small, researchers are trying to phase-lock a number of nano-contacts in order to get a higher signal. Until now this is proving to be a difficult task, in part due to finding the right combination of ferromagnetic materials that will allow for high amplitude microwave oscillations using a DC current.

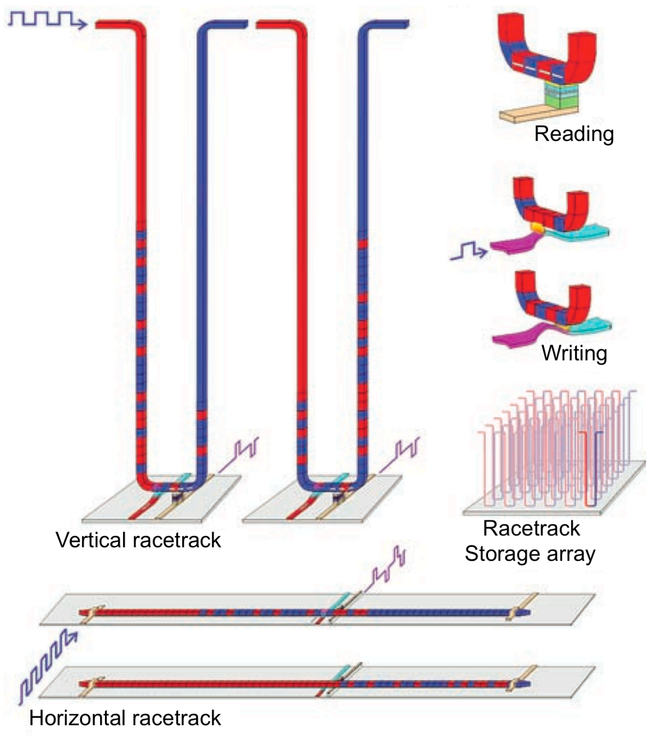

Fig. 3 – Racetrack memory working principle [6].

More recent phenomena have caught the eye of researches, like the spin-Hall and spin-Seebeck effects. It has been realized that spin-currents can be achieved through a magnetic insulator, like in the work reported by Kajiwara et al. [13], in which the spin-Hall effect is used to generate and detect spin-waves through an insulator. Also the temperature difference between two points in a ferromagnet, and even in non-magnetic materials, has shown a spin-current between them, which was denominated as the spin-Seebeck effect, [14],[15]. Both these new ways of approaching spin-currents may open the door to new devices besides a new look into fundamental physics.

In order to study the previously mentioned phenomena, from either the fundamental or experimental points of view, a bridge is needed in order to link the theoretical mathematical-physical models with the experimental devices. That link is achieved through the use of numeric simulations. This last type of investigation has been gaining more and more relevance in the last decades, as the research into different physical properties gets ever more complex. In order to justify the investment into expensive laboratory equipment that allows the scientific community to make the experimental research essential to the advancements in physics and other sciences, simulations are used in order to predict, discover and solve physical phenomena before an experiment is attempted. Simulations provide a unique link between the theoretical research and the experimental one, allowing both parties to see where they coincide and where the theories fail and need to be improved in order to better understand the experimental results. In the same way, micromagnetic simulations have gained an important role within its field because they have proved to be an efficient tool in the verification of the theoretical formalism [16]-[20], and subsequently in the interpretation and design of devices.

Prior to the beginning of the work described by this thesis, micromagnetic simulations based on the Landau-Lifshitz-Gilbert dynamic equation (17), were of a sequential type, written in computer languages like Fortran and C. Although several different advancements have been made using the traditional sequential programming it poses both temporal and spatial limitations, due to the continuum nature and characteristic scale of the micromagnetic mesoscopic formalism. The spatial limitation comes in evidence when trying to simulate either multilayer and/or geometrically large devices (typically in the micrometer range in either Cartesian direction). In order to perform the numeric simulations the atomic spins are grouped into nanometer cells, however these cells cannot be larger than the characteristic length, like the exchange length (defined later in (62)), which is typically of 5 nm or less for a ferromagnet. The spatial issue is then a memory problem, because when wishing to simulate the dynamics of a device like a spin-valve or a MTJ, which is composed of several layers, the number of computational cells can reach the hundreds of thousand or more. The number of cells also brings with it the temporal problem, since if many cells are to be calculated one by one, as they are in a sequential code, more cells means that more time is required to solve the problem under study. Also some complex magnetization structures like the one's in vortex dynamics and Bloch-Point domain walls, require the use of smaller cell sizes in order to avoid numerical errors, which also leads to the use of smaller time steps per iteration due to the stability criteria [21]. Another time issue comes in relevance when trying to accurately calculate the inherent oscillating frequencies of a given device, since the time step sets the maximum frequency one can detect and the duration of the simulation sets the frequency resolution. All of these spatial and time limitations present in micromagnetic simulations were asking for a better solution than the one provided by the sequential programming.

Objectives of this work

In order to try and overcome the previously mentioned limitations of the sequential programming, this work has focused on the development of the micromagnetic simulation computer tool. In particular it explores the advantages of parallel computation by using graphic processing units, GPUs, to solve the dynamic equation that governs the magnetization, by means of the finite difference method.

Once the parallel micromagnetic code is fully functional it is intended to use it towards tackling physical problems that involve large spatial and temporal simulations. These will account for the study of spin-valve dynamics that require large temporal simulations in order to precisely determine the magnetization oscillation frequency and other phenomena. And spatially challenging simulations, like the ones needed to study the dynamics of domain walls in long nano-strips, will also be performed using the new parallel micromagnetic code.

Thesis outline

Chapter 2 – In this section the theoretical fundamentals inherent to the micromagnetism formalism are given. Subjects like the magnetization as the variable of state of the system, the characteristic scale of micromagnetism, the dynamic equation and its different energy contributions, are covered in this chapter.

Chapter 3 – This is the most lengthy chapter, as it describes the computational micromagnetic basics and beyond. It starts by describing the finite difference method within the sequential code framework and how each energy contribution is managed numerically. This will show the limitations of the sequential type of programming, which will serve as a bridge to the second part of the chapter in which micromagnetic computations are approached through the use of GPU parallel computing. This section ends by comparing the results and efficiency between both methods of computation and the verification of the developed parallel code.

Chapter 4 - In this section results are shown of physical micromagnetic studies performed on different devices using both the sequential CPU and the newly developed parallel GPU computing methods.

Finally the thesis ends with the conclusions where the main achievements of this work are described.

2 The basics of Micromagnetism

2.1 Introduction

The magnetic properties of materials are of quantum nature [16]-[20]. However, in order to describe the magnetic properties starting from a quantum point of view, it is required to work from the atomic scale by considering a discrete system of spins. Although such formalism would be satisfactory, it is unviable due to large number of spins involved. Since the ferromagnetic materials that will be studied throughout this work are ranging from the tens of nanometers, up to the micrometer scale, a more practical formalism is needed. Such formalism is known as micromagnetism, and it has been proving to be a very useful tool in the description of the magnetization dynamics of ferromagnetic materials.

Micromagnetism is the theoretical formalism [16]-[20] that allows for the magnetization study at a scale larger than the atomic one, but still small enough to allow the visualization of the internal structure dynamics between magnetic domains (Fig. 4). This formalism is based on the assumption that the magnetization $\boldsymbol{M}(\boldsymbol{r})$, is a continuous vectorial function of the position within the material, and its modulus, called spontaneous magnetization M_S, remains constant. As a result, a ferromagnetic material can be idealized as a group of elements of volume dV with a constant magnetization per unit of volume M_S (Fig. 4). The direction of said magnetization is given by the unit vector $\boldsymbol{m}(\boldsymbol{r})=\boldsymbol{M}(\boldsymbol{r})/M_S$, which varies smoothly between each element of volume. Each element of volume has to be big enough so as to contain a large number of atoms that are responsible for the magnetic moment, *i.e.* much larger than the lattice constant a (Fig. 4). Nonetheless each element of volume has to be small in order to avoid the abrupt variation of the magnetization between each element of volume, in accordance to the continuous nature of the magnetization vectorial function.

Since the type of system that is intended to study is a discrete one, the continuous approximation requires a justification. This is based on the fact that the exchange interaction is the dominant one at short distances, forcing the magnetic dipoles (or the elements of volume dV in the micromagnetic description) to be parallel between each other. With this in mind, all other forces can be seen as a small perturbation to the parallel orientation between first neighbors. Therefore it is reasonable to say that $\boldsymbol{M}(\boldsymbol{r})$

is a continuous function of the position, since the magnetization varies slightly between each surrounding element.

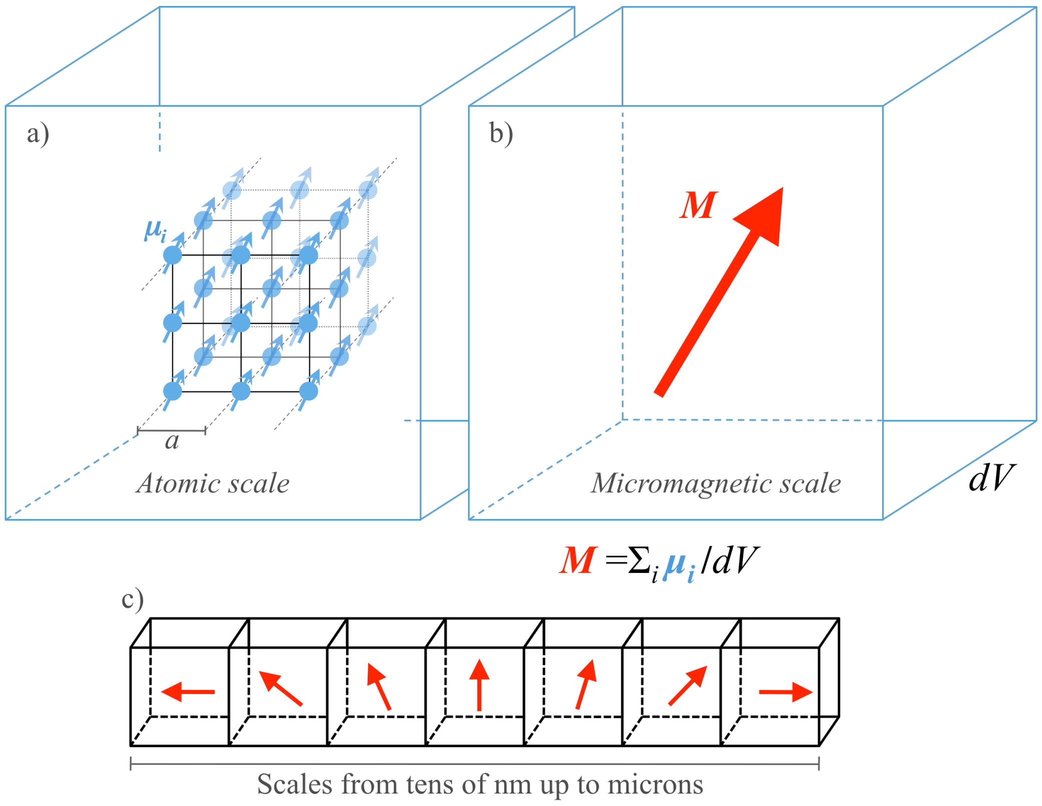
a)
μ_i
a
Atomic scale
b)
M
Micromagnetic scale
dV
$M = \Sigma_i \mu_i / dV$
c)
Scales from tens of nm up to microns

Fig. 4 – Scale in micromagnetism. a) Atomic scale representation of individual magnetic moments $\boldsymbol{\mu}_i$, where a is the lattice constant. b) Micromagnetic scale, representing the magnetization vector $\boldsymbol{M}$ as the sum of all magnetic moments $\boldsymbol{\mu}_i$ inside the volume dV, c) Micromagnetism is in an intermediate scale larger than the atomic one, but small enough to "see" the transition region between magnetic domains.

Although the exchange interaction is the main one in ferromagnetic materials, others are needed in order to fully describe the behavior of such materials, as the ones derived from Maxwell equations of electromagnetism. In the following sections the equation of motion and the different energy contributions relevant to this work, will be discussed.

2.2 The equilibrium and dynamic equations

In order to determine the equilibrium equation, all the energy contributions that act on the magnetization have to be considered. Such expression may be obtained using Hamilton's variational principle [22], so as to solve the time evolution of the system,

$$\int_{t_1}^{t_2} L\,dt = \int_{t_1}^{t_2} \left(\int_V l_V\,dV + \int_S u_S\,dS \right) dt \tag{1}$$

where, L is the Lagrangian functional to the continuous vectorial field in a tridimensional space $\boldsymbol{m}(\boldsymbol{r},t)$, and $l_V = k_V + u_v$ is the Lagrangian density per unit of volume with k_V and u_v being the kinetic and potential energy densities per unit of volume respectively. In the second part of (1) u_S represent the potential energy density per unit of surface. After some algebra and considering the static equilibrium state, it is possible to reach the following equilibrium equations for both volume (2) and surface (3) points respectively,

$$\frac{\delta u_V}{\delta \boldsymbol{m}(\boldsymbol{r},t)} = 0 \tag{2}$$

$$\left[\frac{\delta u_S}{\delta \boldsymbol{m}(\boldsymbol{r},t)} + \frac{\delta u_V}{\delta(\nabla \boldsymbol{m}(\boldsymbol{r},t))} \cdot \boldsymbol{n}\right] = 0 \tag{3}$$

Working in S.I. units it is possible to define a vector with the dimensions of a magnetic field from the functional derivative of the energy density per unit of volume given by (2) as,

$$\boldsymbol{H}_{eff} = -\frac{1}{\mu_0 M_S}\frac{\delta u_V}{\delta \boldsymbol{m}} \tag{4}$$

where the functional derivative is given by,

$$\frac{\delta}{\delta \boldsymbol{m}} \equiv \frac{\partial}{\partial \boldsymbol{m}} - \nabla . \frac{\partial}{\partial(\nabla \boldsymbol{m})} \tag{5}$$

The quantity $\boldsymbol{H}_{eff}$ is called the effective field, whose different energy contributions will be determined in the next section. Thus by multiplying (2) by $\boldsymbol{m}\times$[1] one can express the volume equilibrium condition in terms of $\boldsymbol{H}_{eff,}$ as,

$$\boldsymbol{m} \times \frac{\delta u_V}{\delta \boldsymbol{m}} = 0 \rightarrow \boldsymbol{m} \times \boldsymbol{H}_{eff} = 0 \tag{6}$$

The last expression represents the state when the torque between the magnetization $\boldsymbol{M}$ and the $\boldsymbol{H}_{eff}$ is zero in each element of volume of the material. Equations (3) and (6) are known as Brown's equations [23]. From (3) it is also possible to write a torque of the magnetization with a surface effective field $\boldsymbol{H}_{eff,S}$, however this contribution will not be taken into account and thus will not be discussed further.

Suppose now a ferromagnetic sample with a magnetization $\boldsymbol{M}$ per unit of volume under the influence of a magnetic field $\boldsymbol{H}_{eff}$, in S.I. units the induction field $\boldsymbol{B}_{eff}$, can be expressed by $\boldsymbol{B}_{eff}=\mu_0(\boldsymbol{M}+\boldsymbol{H}_{eff})$. Under such field each element of volume of the system experiments a torque $\boldsymbol{\tau}$ given by,

[1] Note that throughout the micromagnetic framework described here $\boldsymbol{m}\equiv\boldsymbol{m}(\boldsymbol{r},t)$ or $\boldsymbol{M}\equiv\boldsymbol{M}(\boldsymbol{r},t)$, $\boldsymbol{m}$ or $\boldsymbol{M}$ is written for the sake of simplicity.

$$\boldsymbol{\tau} = \boldsymbol{M} \times \boldsymbol{B}_{eff} = \mu_0 \boldsymbol{M} \times \boldsymbol{H}_{eff} \tag{7}$$

where μ_0 is the vacuum magnetic permeability. From (7) it can be deduced that the torque $\boldsymbol{\tau}$ tends to rotate each magnetization element of volume in a given direction, set by the local $\boldsymbol{H}_{eff}$.

The expression for the magnetization dynamics can be obtained from the magnetic dipole $\boldsymbol{\mu}$ equation of motion under the influence of a field $\boldsymbol{B}_{eff}$, by using Newton's second law of motion. The torque that acts upon the magnetic dipole is equal to the variation of angular momentum $\boldsymbol{J}$, of the magnetic dipole,

$$\frac{d\boldsymbol{J}}{dt} = \boldsymbol{\tau} = \mu_0 \boldsymbol{\mu} \times \boldsymbol{H}_{eff} \tag{8}$$

In most ferromagnetic materials the angular momentum $\boldsymbol{J}$, is mainly due to the electron's spin and thus $\boldsymbol{J}=\boldsymbol{L}+\boldsymbol{S}\approx\boldsymbol{S}$, [24]-[26]. The spin $\boldsymbol{S}$ and the magnetic dipole $\boldsymbol{\mu}$ are related through the gyromagnetic ratio γ by,

$$\boldsymbol{\mu} = \gamma \boldsymbol{S} \tag{9}$$

where γ is given by,

$$\gamma = \frac{g q_e}{2 m_e} = \frac{g \mu_B}{\hbar} < 0 \tag{10}$$

where, g is the so-called Landé factor, which for a free electron is approximately equal to 2. The other factors are the electron's charge q_e and mass m_e, the Bohr magneton μ_B, and the reduced Planck's constant $\hbar$. The magnetic dipole dynamic equation can then be written as,

$$\frac{d\boldsymbol{\mu}}{dt} = \gamma \mu_0 \boldsymbol{\mu} \times \boldsymbol{H}_{eff} \tag{11}$$

Since in the micromagnetic formalism, as was seen in the previous section, instead of individual magnetic dipoles a large group of them is considered in each element of volume dV (Fig. 4), the dynamic equation of the magnetization can be described as,

$$\frac{d\boldsymbol{M}}{dt} = -\gamma_0 \boldsymbol{M} \times \boldsymbol{H}_{eff} \tag{12}$$

where γ_0 is defined as,

$$\gamma_0 = \mu_0 \frac{g|\mu_B|}{\hbar} = -\mu_0 \gamma \tag{13}$$

The dynamics expressed by (12) describes that in the presence of a constant magnetic field the magnetization $\boldsymbol{M}$ would rotate indefinitely, never achieving the equilibrium state[2]. However, from experience after a given amount of time the magnetization will align itself with the field, meaning that the system will tend to an equilibrium state through some mechanism of dissipative nature. There are several processes that may contribute to the dissipation of energy, like lattice interactions, scattering, eddy currents, etc. All of which are really difficult to describe, thus in micromagnetism all of these dissipative processes are included in a phenomenological way, by adding a dissipative term into the dynamic equation (12).

The simplest way to add dissipation to the dynamic equation is to modify the effective field $\boldsymbol{H}_{eff}$ in order to include an Ohmic type of dissipation [16], (Fig. 5 (b)),

$$\boldsymbol{H}_{eff} \rightarrow \boldsymbol{H}_{eff} - \frac{\alpha}{\gamma_0 M_S} \frac{d\boldsymbol{M}}{dt} \tag{14}$$

where α is the dimensionless phenomenological damping parameter. Using the modified effective field from (14) in equation (12), the Gilbert equation is obtained,

$$\frac{d\boldsymbol{M}}{dt} = -\gamma_0 \boldsymbol{M} \times \boldsymbol{H}_{eff} + \frac{\alpha}{M_S} \boldsymbol{M} \times \frac{d\boldsymbol{M}}{dt} \tag{15}$$

From the previous equation it can be seen that the system will continuously lose energy until it reaches the equilibrium state, where the magnetization $\boldsymbol{M}$ is parallel to the effective field $\boldsymbol{H}_{eff}$. Also, if $d\boldsymbol{M}/dt=0$ the stationary state of the dynamic equation (15), is reduced to the equilibrium equation (6).

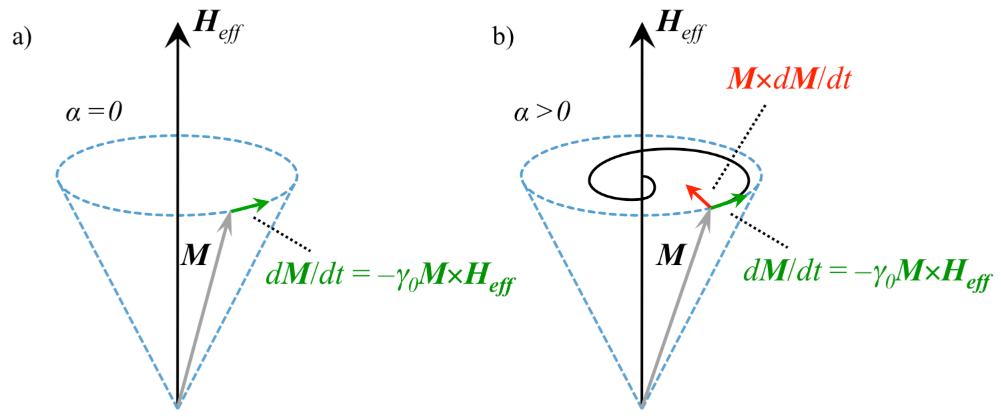
a)
$\boldsymbol{H}_{eff}$
$\alpha = 0$
$\boldsymbol{M}$
$d\boldsymbol{M}/dt = -\gamma_0 \boldsymbol{M}\times\boldsymbol{H}_{eff}$
b)
$\boldsymbol{H}_{eff}$
$\boldsymbol{M}\times d\boldsymbol{M}/dt$
$\alpha > 0$
$\boldsymbol{M}$
$d\boldsymbol{M}/dt = -\gamma_0 \boldsymbol{M}\times\boldsymbol{H}_{eff}$

Fig. 5 – Magnetization $\boldsymbol{M}$ dynamics in the presence of a magnetic field $\boldsymbol{H}_{eff}$. a) Without dissipation α=0, the magnetization rotates around the field with frequency $-\gamma_0 \boldsymbol{H}_{eff}$. With damping α>0, after a certain amount of time the magnetization will precess until it aligns itself with the field, due to the dissipative term $\boldsymbol{M}\times d\boldsymbol{M}/dt$.

Working with the Gilbert equation is not a simple task, since the time derivative of the magnetization is present in both sides of equation (15). However, a different way

[2] In truth the rotation of $\boldsymbol{M}$ around $\boldsymbol{H}_{eff}$ would in fact radiate energy, however this dissipative contribution is depreciable in this context.

of introducing the phenomenological dissipation term in (12) was proposed by Landau-Lifshitz. In this proposal the dissipative term is added in such a way that it is perpendicular to both the magnetization vector $\boldsymbol{M}$ and the precession of $\boldsymbol{M}\times\boldsymbol{H}_{eff}$,

$$\frac{d\boldsymbol{M}}{dt} = -\gamma_0 \boldsymbol{M}\times\boldsymbol{H}_{eff} - \frac{\gamma_0\alpha}{M_S}\boldsymbol{M}\times\left(\boldsymbol{M}\times\boldsymbol{H}_{eff}\right) \tag{16}$$

Unlike the Gilbert equation (15), in (16) the dissipation and precession terms are uncoupled and thus can be easily solved numerically. It is possible to obtain an equation which is formally equal to the one described by Landau-Lifshitz (16), and that expresses the same dynamics of Gilbert's equation (15). Multiplying both sides of (15) by $\boldsymbol{M}\times$ and using the property $\boldsymbol{a}\times(\boldsymbol{b}\times\boldsymbol{c})=(\boldsymbol{a}.\boldsymbol{c})\boldsymbol{b}-(\boldsymbol{a}.\boldsymbol{b})\boldsymbol{c}$, and $\boldsymbol{M}.\boldsymbol{M}= M_S^2$, one reaches the following expression,

$$\frac{d\boldsymbol{M}}{dt} = -\frac{\gamma_0}{1+\alpha^2}\boldsymbol{M}\times\boldsymbol{H}_{eff} - \frac{\gamma_0}{1+\alpha^2}\frac{\alpha}{M_S}\boldsymbol{M}\times\left(\boldsymbol{M}\times\boldsymbol{H}_{eff}\right) \tag{17}$$

The previous equation is formally known as the Landau-Lifshitz-Gilbert (LLG) equation and it describes the same physical properties as (15). Equation (17) represents the dynamic equation used to solve the dynamics of the magnetization throughout this work. However it should be kept in mind that in order to take into account the effect of spin transfer torque, some terms need to be added to (17). These terms will be discussed further along this chapter. Sometimes it is also useful to express the previous expression in the form,

$$(1+\alpha^2)\frac{d\boldsymbol{m}}{dt} = -\gamma_0\left[\boldsymbol{m}\times\boldsymbol{H}_{eff} - \alpha\,\boldsymbol{m}\times\left(\boldsymbol{m}\times\boldsymbol{H}_{eff}\right)\right] \tag{18}$$

with, $\boldsymbol{m} = \dfrac{\boldsymbol{M}}{M_S}$

2.3 Energy contributions and the effective field

In the following sub-sections all of the energy contributions to the effective field (4) are discussed, so as to use them to calculate the LLG dynamic equation (17). These contributions are of both quantum nature like the exchange and anisotropy interactions, and of classical nature like the magnetostatic, Oersted and Zeeman interactions. The effects of a thermal field added to the $\boldsymbol{H}_{eff}$ are also discussed so as to account for the effect of temperature.

2.3.1 Exchange energy

As it was said before, the exchange energy is the dominant interaction at short distances and thus it is responsible for the parallel alignment between neighboring spins. Its origin is of quantum mechanical nature and it is due to Pauli's exclusion principle. The Heisenberg Hamiltonian that describes the exchange interaction is usually written as [23],

$$\mathcal{H}_{exch} = -\sum_{i,j}^{fN} 2J_{ij}\hat{\boldsymbol{S}}_i.\hat{\boldsymbol{S}}_j \tag{19}$$

The sum is among first neighbors fN, $\hat{\boldsymbol{S}}_{ij}$ represent the spin operators and J_{ij} is the exchange integral, whose value decreases rapidly with the distance between spins and thus is only noticeable among first neighbors. This means that one can simply write J instead of J_{ij}.

In order to adapt the Heisenberg Hamiltonian to the continuous description used here, the spin operators are replaced by the vectors $\boldsymbol{S}_{i,j}=S_{i,j}\boldsymbol{s}_{i,j}$, (with $S_i=S_j=S$), and the dot product in (19) is rewritten so as to obtain the following exchange energy expression,

$$\mathcal{U}_{exch} = -JS^2\sum_{i,j}\cos\theta_{ij} \tag{20}$$

where θ_{ij} is the angle between spins $\boldsymbol{s}_i$ and $\boldsymbol{s}_j$ and j represents the first neighboring spins of i (Fig. 6).

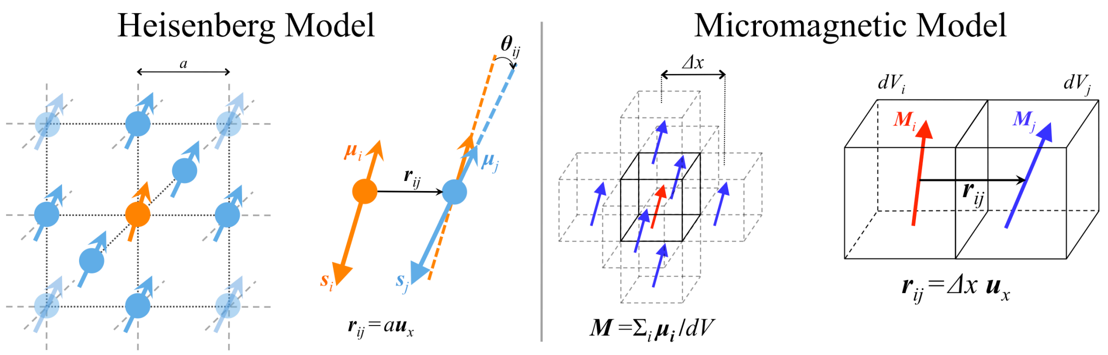

Fig. 6 – Exchange interaction representation between first neighbors. In the atomic Heisenberg model representation, a is the lattice constant and $\boldsymbol{r}_{ij}$ the position vector between the spins i and j. In the micromagnetic model, each magnetization $\boldsymbol{M}$ volume element is at a Δx distance from the first neighbors where $\boldsymbol{r}_{ij}$ is the position vector connecting them.

Assuming that the angle between the spins θ_{ij} is very small it is possible to use the mathematical property $\cos\theta_{ij} \approx 1-\theta^2_{ij}/2$ in (20), plus considering that the sum is for each pair of first neighbors then,

$$\mathcal{U}_{exch} = JS^2 \sum_i \sum_j \theta_{ij}^2 \tag{21}$$

In the micromagnetic description the unit vector in the direction of the spin $\boldsymbol{s}_i$ of each dipole $\boldsymbol{\mu}_i$ present in the lattice is replaced by the unit vector $\boldsymbol{m}_i = \boldsymbol{M}_i / M_S$, along the direction of the magnetization in each element of volume $\boldsymbol{M}_i = \Sigma_i \boldsymbol{\mu}_i / dV$, (Fig. 6). Using the dimensionless and continuous variable of the magnetization $\boldsymbol{m}$, it is possible to write for small angles that,

$$\left|\theta_{ij}\right| \approx \left|\boldsymbol{m}_i - \boldsymbol{m}_j\right| \approx \left|\left(\boldsymbol{r}_{ij}.\nabla\right)\boldsymbol{m}\right| \tag{22}$$

where, $\boldsymbol{r}_{ij}$ is the position vector between *i* and *j* (Fig. 6). Thus the exchange energy can now be written as,

$$\mathcal{U}_{exch} = JS^2 \sum_i \sum_{\boldsymbol{r}_{ij}} \left[\left(\boldsymbol{r}_{ij}.\nabla\right)\boldsymbol{m}\right]^2 \tag{23}$$

In the continuous representation the sum in *i* is replaced by an integral over the volume *V* of the ferromagnetic sample under study. Thus the exchange energy takes the form of,

$$\mathcal{U}_{exch} = \int_V u_{ex} dV = \int_V A(\nabla \boldsymbol{m})^2 dV \tag{24}$$

where *A* is the exchange constant in J/m, which is given by,

$$A = \frac{JS^2 z}{a} \tag{25}$$

where *a* is distance between first neighbors and *z* is equal to 1, 2 or 4 depending if the lattice is simple cubic, face-centered cubic or body-centered cubic, respectively. The exchange energy described by (24) can also be shown as valid for other types of lattice like the hexagonal one, [23]. Equation (24) is considered to be valid at any temperature *T*, and that *A* varies with *T*.

Finally by using (4) and (24) the expression for the exchange field $\boldsymbol{H}_{exch}$ at each element of volume can be written as,

$$\boldsymbol{H}_{exch}(\boldsymbol{r}) = \frac{2A}{\mu_0 M_S}\left(\nabla^2 \boldsymbol{m}(\boldsymbol{r})\right) \tag{26}$$

2.3.2 Anisotropy energy

From different experiments it is well know that magnetic materials are in general not isotropic and thus have preferred directions. Such directions are called easy directions, they are easier to magnetize and are related to the symmetric directions of the crystal. The anisotropy energy is then defined as the excess energy needed to magnetize a material in a certain direction in respect to an easy direction. Its origin comes from spin-orbit interactions at the atomic level. As a result, it is very complicated to obtain an expression for it starting from a microscopic model and thus a phenomenological approach is used [16].

Since throughout this work the focus was on materials with uniaxial anisotropy, this will be the only one described. Taking the first term of the Taylor development, the uniaxial magneto-crystalline energy is equal to,

$$\mathcal{U}_{an.u} = \int_V u_{an,u} dV = \int_V K\left(1 - (\boldsymbol{m}.\boldsymbol{u}_K)^2\right) dV \tag{27}$$

where K (J/m^3) is the anisotropy constant and $\boldsymbol{u}_K$ the unit vector along the anisotropy direction. If K>0 the direction of $\boldsymbol{u}_K$ is an easy direction, on the contrary if it K<0 it is a hard direction. In the first case, the uniaxial magneto-crystalline anisotropy energy is at a minimum when the magnetization is parallel to easy axis $\boldsymbol{u}_K$. On the other hand, when K<0, the anisotropy energy is at a maximum when the magnetization is parallel to the hard axis $\boldsymbol{u}_K$. As a result, in this case the plane perpendicular to the hard axis $\boldsymbol{u}_K$ is called easy plane, since all the directions parallel to that plane are easy directions being energy minimums of the uniaxial magneto-crystalline energy.

From the energy equation (27) and using (4), one determines the uniaxial anisotropy effective field $\boldsymbol{H}_{an,u}$ for each element of volume as,

$$\boldsymbol{H}_{an,u}(\boldsymbol{r}) = \frac{2K}{\mu_0 M_S}(\boldsymbol{m}.\boldsymbol{u}_K)\boldsymbol{u}_K \tag{28}$$

2.3.3 Magnetostatic energy

The magnetostatic energy is the energy associated to the magneto-static interaction between the lattice magnetic dipoles. The field created by the dipoles is called magnetostatic or sometimes demagnetizing field, since when looking from inside the material this field tends to demagnetize the sample, both names are used throughout. This is one of the classical fields derived from Maxwell equations (29)-(32), whose contribution is essential to the micromagnetic formalism, and it is also the most challenging field to compute, as it will be shown in chapter 3.

$$\nabla.\boldsymbol{D}(\boldsymbol{r},t) = \rho(\boldsymbol{r},t) \tag{29}$$

$$\nabla.\boldsymbol{B}(\boldsymbol{r},t) = 0 \tag{30}$$

$$\nabla \times \boldsymbol{E}(\boldsymbol{r},t) = -\frac{\partial \boldsymbol{B}(\boldsymbol{r},t)}{\partial t} \tag{31}$$

$$\nabla \times \boldsymbol{H}(\boldsymbol{r},t) = \boldsymbol{j}(\boldsymbol{r},t) - \frac{\partial \boldsymbol{D}(\boldsymbol{r},t)}{\partial t} \tag{32}$$

where, $\boldsymbol{D}(\boldsymbol{r},t)$ is the electrical displacement vector field, $\rho(\boldsymbol{r},t)$ is the charge density per unit of volume, $\boldsymbol{B}(\boldsymbol{r},t)$ is the magnetic induction field, $\boldsymbol{E}(\boldsymbol{r},t)$ is the electrical field, $\boldsymbol{H}(\boldsymbol{r},t)$ is the magnetic field, and $\boldsymbol{j}(\boldsymbol{r},t)$ is the current density per unit of volume.

It is well know that a uniformly magnetized ellipsoid along one of his axis experiences a magnetic field that opposes the magnetization per unit of volume, hence the name demagnetizing field, $\boldsymbol{H}_{dmg}$[3]. Naturally the same effect is seen in all geometries of different magnetic materials with magnetizations pointing in different directions.

Considering the case in which the magnetostatic field is the only field present, *i.e.* in the absence of electrical fields, electrical currents, or any other magnetic field, the aforementioned Maxwell equations are reduced to,

$$\nabla.\boldsymbol{B}(\boldsymbol{r}) = 0 \tag{33}$$

$$\nabla \times \boldsymbol{H}(\boldsymbol{r}) = 0 \tag{34}$$

Knowing that $\boldsymbol{B}=\mu_0(\boldsymbol{M}+\boldsymbol{H})$ where now the magnetic field $\boldsymbol{H}$ represents the magnetostatic field $\boldsymbol{H}_{dmg}$, the previous equations can be written as,

$$\nabla.\mu_0\left(\boldsymbol{M}(\boldsymbol{r}) + \boldsymbol{H}_{dmg}(\boldsymbol{r})\right) = 0 \quad \Leftrightarrow \quad \nabla.\boldsymbol{H}_{dmg}(\boldsymbol{r}) = \nabla.\boldsymbol{M}(\boldsymbol{r}) \tag{35}$$

$$\nabla \times \boldsymbol{H}_{dmg}(\boldsymbol{r}) = 0 \tag{36}$$

Calculating the magnetostatic field $\boldsymbol{H}_{dmg}(\boldsymbol{r})$ is analogous to calculating the electrical field $\boldsymbol{E}(\boldsymbol{r})$ created by a distribution of electrical charges $\rho(\boldsymbol{r})$ inside a volume V surrounded by a surface S (Fig. 7). Therefore from equations (35) and (36), and keeping in mind the surface boundary conditions that $\boldsymbol{H}_{dmg}(\boldsymbol{r})$ must obey, it is possible to define fictitious volume and surface magnetic charge densities, respectively as $\rho_m(\boldsymbol{r})$ and $\sigma_m(\boldsymbol{r})$. Naturally magnetic charges do not exist, the previous quantities are thus defined in order to aid in the calculation of the field $\boldsymbol{H}_{dmg}(\boldsymbol{r})$, in the same way as the electrical field is calculated.

[3] Note that although it is named as $\boldsymbol{H}_{dmg}$, this field represents the magnetostatic field generated by the sample in all points inside and outside the sample. The subscript $_{\text{dmg}}$ is used throughout to refer to the magnetostatic field.

$$\rho_m(\boldsymbol{r}) = -\nabla.\boldsymbol{M}(\boldsymbol{r}) \tag{37}$$

$$\sigma_m(\boldsymbol{r}) = \boldsymbol{M}(\boldsymbol{r}).\boldsymbol{n} \tag{38}$$

Where ***n*** is the unit vector perpendicular to the surface *S*. From the previous equations it is possible to deduce the following expression for the magnetostatic field $\boldsymbol{H}_{dmg}(\boldsymbol{r})$ at each point ***r***, of the sample,

$$\boldsymbol{H}_{dmg}(\boldsymbol{r}) = \frac{1}{4\pi}\left[\int_{V'} \frac{(\boldsymbol{r}-\boldsymbol{r}')\rho_m}{|\boldsymbol{r}-\boldsymbol{r}'|^3} dV' + \int_{S'} \frac{(\boldsymbol{r}-\boldsymbol{r}')\sigma_m}{|\boldsymbol{r}-\boldsymbol{r}'|^3} dS'\right] \tag{39}$$

where |***r-r'***| is the distance between the point of the field being calculated (***r***) and all other field creating magnetic moments (at ***r'***). The magnetostatic energy can now be written as,

$$\mathcal{U}_{dmg} = \int_V u_{dmg} dV = -\frac{1}{2}\mu_0 \int_V \boldsymbol{H}_{dmg}(\boldsymbol{r}).\boldsymbol{M}(\boldsymbol{r}) dV \tag{40}$$

where the factor ½ is added because the source of the magnetostatic field $\boldsymbol{H}_{dmg}(\boldsymbol{r})$ is the volume magnetization distribution ***M***(***r***) of the sample.

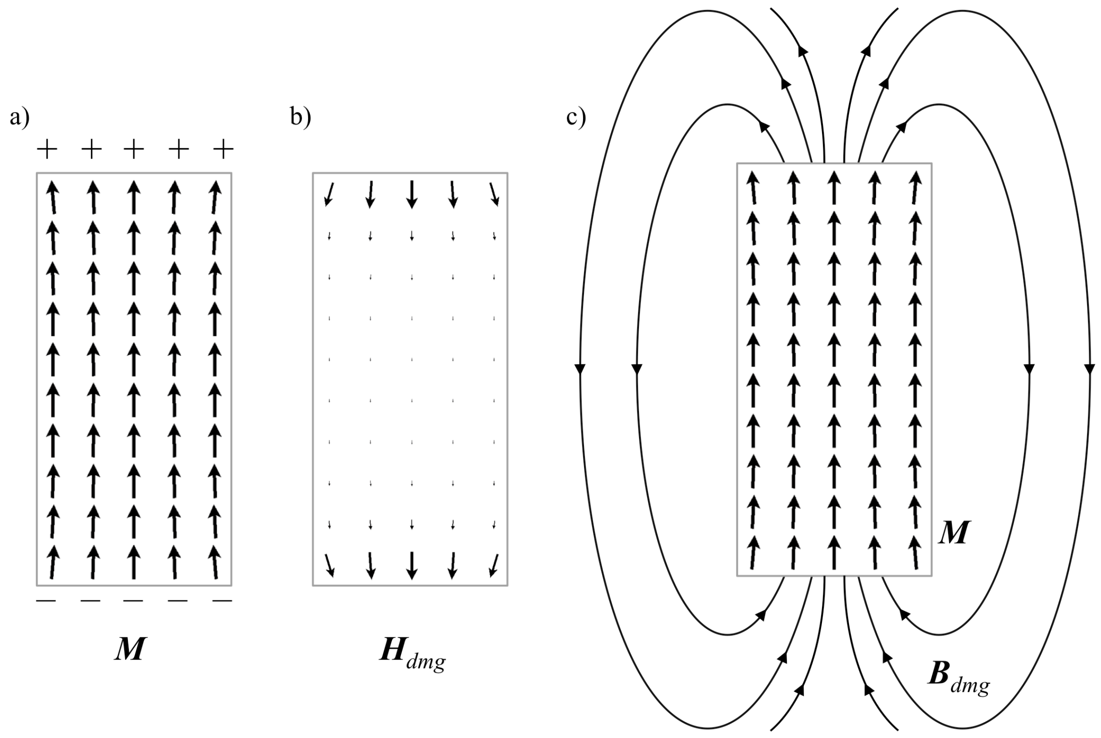

Fig. 7 – Representation of the magnetostatic field $\boldsymbol{H}_{dmg}$, in a rectangular prism. a) Magnetization ***M*** of the sample as if there were "magnetic charges". b) The magnetization ***M*** is responsible for the creation of a magnetostatic field $\boldsymbol{H}_{dmg}$ inside the sample and it also induces a magnetostatic field $\boldsymbol{B}_{dmg}$ in the regions outside the sample c). Outside the sample ***M***=0 and thus $\boldsymbol{B}_{dmg}=\mu_0\boldsymbol{H}_{dmg}$.

2.3.4 Zeeman energy

The Zeeman field is referred in micromagnetism as the externally applied field, which might be used to magnetize the sample in any given direction. This field, for example, is commonly used to see the typical hysteresis curve of a given material, among other different applications. Within the micromagnetic formalism this field is usually uniform throughout the sample under study, (however it can be made to vary in both space and time). This is in general a good approximation since in principle, as in typical experiments, the micro-sized sample is immersed in an external field $\boldsymbol{H}_{ext}(\boldsymbol{r})$, which is considered uniform.

Because the Zeeman field $\boldsymbol{H}_{ext}(\boldsymbol{r})$ is considered an external variable to the system, and in this formalism represents a vectorial field whose module and direction are known beforehand, the Zeeman energy [23] density can be written as,

$$\mathcal{U}_{Zee} = \int_V u_{Zee} dV = -\mu_0 \int_V \boldsymbol{H}_{ext}(\boldsymbol{r}).\boldsymbol{M}(\boldsymbol{r}) dV \tag{41}$$

2.3.5 The Oersted field

The Oersted field can also be regarded as an external field since its source is the external current density $\boldsymbol{j}(\boldsymbol{r})$ that flows through each element of volume dV. Thus the energy density can be described as,

$$\mathcal{U}_{Oe} = \int_V u_{Oe} dV = -\mu_0 \int_V \boldsymbol{H}_{Oe}(\boldsymbol{r}).\boldsymbol{M}(\boldsymbol{r}) dV \tag{42}$$

and the field $\boldsymbol{H}_{Oe}(\boldsymbol{r})$[4] is determined from Maxwell equations (29)–(32) as,

$$\boldsymbol{H}_{Oe}(\boldsymbol{r}) = \frac{1}{4\pi}\left[\int_{V'} \boldsymbol{j}(\boldsymbol{r}') \times \frac{\boldsymbol{r}-\boldsymbol{r}'}{|\boldsymbol{r}-\boldsymbol{r}'|^3} dV'\right] \tag{43}$$

2.3.6 The Thermal field

Since it would be a very difficult task to construct a thermal theory starting from the particle level up to the mesoscopic scale of micromagnetism, the temperature effect is usually included by adding a random noise thermal field $\boldsymbol{H}_{th}$, to the deterministic dynamic equation (17). The dynamic equation is thus converted into a stochastic one, which is usually referred to as the Langevin equation [27]. In 1963 Brown applied this procedure to single domain particles, where he showed what the statistical properties [27],[29] had to be in order to correctly reproduce the equilibrium thermodynamics [28]. It is quite difficult to apply that methodology to the

[4] The current density may also vary in time, $\boldsymbol{j}(\boldsymbol{r},t)$, hence so will the Oersted field $\boldsymbol{H}_{Oe}(\boldsymbol{r},t)$.

continuous formalism of micromagnetism, however once it is discretized[5] the problem is formally equivalent to an ensemble of single domain particles. Since once discretization is achieved each cell corresponds to a single magnetic particle, thus the same procedure can be used in order to include thermal fluctuations to the micromagnetic simulations. As a result the thermal random field $\boldsymbol{H}_{th}$ can be added to the effective field $\boldsymbol{H}_{eff}$ (4) acting on the magnetization of each cell.

$$\boldsymbol{H}_{eff} \rightarrow \boldsymbol{H}_{eff} + \boldsymbol{H}_{th} \tag{44}$$

The Cartesian components of $\boldsymbol{H}_{th}$ are independent Gaussian distributed random numbers with the following statistical properties,

$$\left\langle \boldsymbol{H}_{th,\alpha,i}(t) \right\rangle = 0 \tag{45}$$

$$\left\langle \boldsymbol{H}_{th,\alpha,i}(t)\, \boldsymbol{H}_{th,\beta,j}(t') \right\rangle = 2D\delta_{ij}\delta_{\alpha\beta}\delta(t-t') \tag{46}$$

where i and j are the cell counters, α, $\beta = x,y,z$ refer to the Cartesian components of the field and the brackets represent the statistical average. Each Kronecker delta has a different meaning; the first, δ_{ij}, implies that the fluctuating term of different cells are independent from each other; the second one, $\delta_{\alpha\beta}$, means that the three Cartesian terms are independent from each other; the Dirac delta, $\delta(t–t')$, indicates that the noise is uncorrelated in time, *i.e.* it is a white noise. The coefficient D is obtained so as to satisfy Maxwell-Boltzmann statistics when thermodynamic equilibrium is reached. This is achieved through the stationary solution of the Fokker-Planck equation [27],[29], which is constructed from Langevin's equation and governs the probability distribution dynamics of the magnetization, leading to,

$$D = \frac{\alpha k_B T}{\left(1+\alpha^2\right)\gamma_0 \mu_0 M_S V} \tag{47}$$

where k_B is the Boltzmann constant, T the temperature and V is the volume of each individual cell. Therefore, the fluctuating thermal field that will be added to the effective field $\boldsymbol{H}_{eff}$ (4), at each cell, within the micromagnetic formalism is given by,

[5] The discretization of the micromagnetism theory is executed in chapter 3, where the numerical integration that will allow for the simulation of the magnetization dynamics of different magnetic materials is described.

$$\boldsymbol{H}_{th,i}(t) = \boldsymbol{\eta}_i(t)\sqrt{\frac{2\alpha k_B T}{\left(1+\alpha^2\right)\gamma_0 \mu_0 M_S V \Delta t}} \tag{48}$$

where $\boldsymbol{\eta}_i(t)$ is a stochastic vector whose components are zero-mean, standard normal-distributed random numbers and Δt is the time step used in the micromagnetic simulations.

2.3.7 The spin-transfer-torque

As it was previously mentioned, the phenomenon of spin transfer torque (STT) opens the possibility to manipulate the magnetization of a material by using currents instead of fields. Therefore, it opens the possibility to design different new devices, which are not only potentially much faster but also more energy efficient.

The STT effect arises whenever the flow of spin-angular momentum through a sample is not constant, but has sources or sinks. This torque transfer is “seen”, for example, whenever a spin-current flowing through a magnetic material whose magnetic moment is not collinear with that of the spin-current (Fig. 8). Changes to the spin-angular momentum flow also occur when spin-polarized currents pass through a magnetic domain wall or any other non-uniform magnetization pattern such as vortexes, etc. In this process, the spin conducting electrons have their spins rotate in the direction of the local magnetization, and thus the angular momentum spin vector flow changes as a function of the position. Therefore, the magnetization $\boldsymbol{m}$ of a ferromagnet influences the flow of spin-angular momentum of the conducting electrons due to the exchange interaction between them, by exerting a torque on the incoming spins reorienting them in the process. Due to Newton’s third law of motion, the flowing electrons also must exert an equal and opposite torque onto the local magnetization of the ferromagnet. This exerted torque by the non-equilibrium conduction electrons onto the ferromagnet is what is commonly known as the STT.

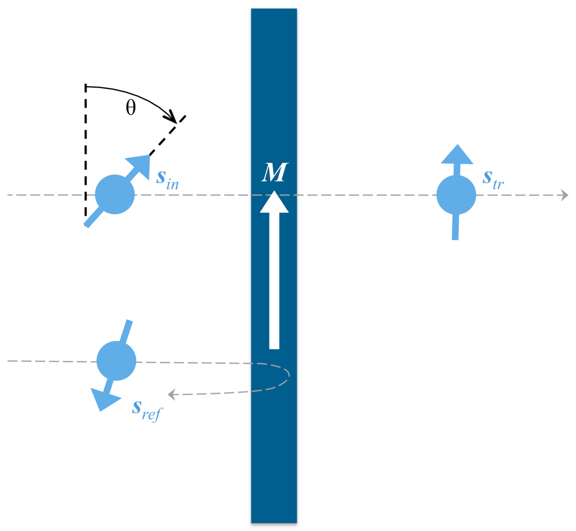

Fig. 8 – Representation of a single spin of angular momentum $\boldsymbol{s}_{in}$, that suffers a torque exerted by the magnetization $\boldsymbol{M}$ of the magnetic thin film, which in turn gets reoriented by the direction of $\boldsymbol{M}$ and is transmitted through the thin film with angular momentum $\boldsymbol{s}_{tr}$. The amount of spin that gets transmitted depends on the material properties and many are reflected with moment $\boldsymbol{s}_{ref}$., which are mainly constituted by the spins of opposite polarization in regard to the transmitted ones.

During the development of this work two types of STT were used, one for devices in which the Current flow is Perpendicular to Plane (CPP), like in spin-valves (Fig. 9) or MTJs, and the other when the Current is In-Plane (CIP), as is the case in domain wall (DW) dynamics along ferromagnetic strips.

The STT contribution enters the equation describing the magnetization dynamics by including additional torques to the LLG dynamic equation (18). This will be shown for each type of torque below.

Current Perpendicular to Plane (CPP).

In the case where two magnetic layers are separated by a non-magnetic metal spacer, as in a spin-valve (Fig. 9), the STT that the thin magnetic layer experiences was given by Slonczewski [3] and it reads,

$$\Gamma_{ST-CPP} = \frac{d\boldsymbol{m}}{dt} = -\frac{g\mu_B \boldsymbol{j}\mathcal{P}(\boldsymbol{m}.\boldsymbol{p})}{2M_S d|q_e|}\boldsymbol{m}\times(\boldsymbol{m}\times\boldsymbol{p}) \tag{49}$$

The previous equation is written in its dimensionless form, where $\boldsymbol{p}=\boldsymbol{M}_P/\boldsymbol{M}_{PS}$ is the unit vector magnetization of the assumed thick polarizing layer with saturation magnetization $\boldsymbol{M}_{PS}$, and $\boldsymbol{m}=\boldsymbol{M}/\boldsymbol{M}_S$ is the unit vector magnetization of the assumed thin free layer, $\boldsymbol{j}$ is the current density, d the thickness of the layer undergoing the STT effect, q_e the electron's charge, and $\mathcal{P}(\boldsymbol{m}.\boldsymbol{p})$ is a polarization function, which depends on the relative orientation between the assumed thin free layer $\boldsymbol{m}$, and the thick pinned layer $\boldsymbol{p}$, magnetizations [3],

$$\mathcal{P}(\boldsymbol{m}.\boldsymbol{p}) = \frac{1}{-4 + (1+\eta)^3 \frac{(3+(\boldsymbol{m}.\boldsymbol{p}))}{4\eta^{3/2}}} \tag{50}$$

where η is the spin polarization factor of the magnetic material under study. Other polarization functions can also be used for spin-valves, which also take into account the giant-magneto resistance asymmetry χ, besides the polarization factor and it is given by [30],

$$\mathcal{P}(\boldsymbol{m}.\boldsymbol{p}) = \frac{0.5\eta(\chi+1)}{2+\chi(1-(\boldsymbol{m}.\boldsymbol{p}))} \tag{51}$$

The previous expressions are valid for spin-valve devices, however if considering a MTJ in which an insulator spacer is used instead of a non-magnetic metal, the polarization function reads [31],

$$\mathcal{P}(\boldsymbol{m}.\boldsymbol{p}) = \frac{1}{2}\eta\frac{1}{1+\eta^2(\boldsymbol{m}.\boldsymbol{p})} \tag{52}$$

Adding the spin torque term (49) to the dynamic equation (18), allows for the writing of the LLG equation with the STT effect for CPP devices as,

$$\begin{aligned}\left(1+\alpha^2\right)\frac{d\boldsymbol{m}}{dt} &= -\gamma_0\left[\boldsymbol{m}\times\boldsymbol{H}_{eff} + \alpha\boldsymbol{m}\times\left(\boldsymbol{m}\times\boldsymbol{H}_{eff}\right)\right] \\ &\quad - \frac{g\mu_B \boldsymbol{j}\mathcal{P}(\boldsymbol{m}.\boldsymbol{p})}{2M_S d|q_e|}\left[\boldsymbol{m}\times(\boldsymbol{m}\times\boldsymbol{p}) - \alpha(\boldsymbol{m}\times\boldsymbol{p})\right]\end{aligned} \tag{53}$$

Usually it is considered that the pinned layer magnetization $\boldsymbol{p}$, does not suffer the effects of the spin-torque interaction since it is considered to be a pinned or fixed layer. Although this is a good approximation for many experiments it is not generally true, and in some cases the dynamics of the called pinned layer is relevant, as is the case of coupled device modes involving vortex oscillations in both layers (section 4.2.1 and [32]). In such cases, both layers $\boldsymbol{m}$ and $\boldsymbol{p}$ in Fig. 9 are considered free to move and both polarize the conducting electrons depending on the direction of the current. Thus when developing a micromagnetic simulation code in which both layer magnetizations are considered dynamic, the names pinned and free are usually changed to thick and thin. (Nonetheless in Fig. 9 the usual names pinned and free layer were kept).

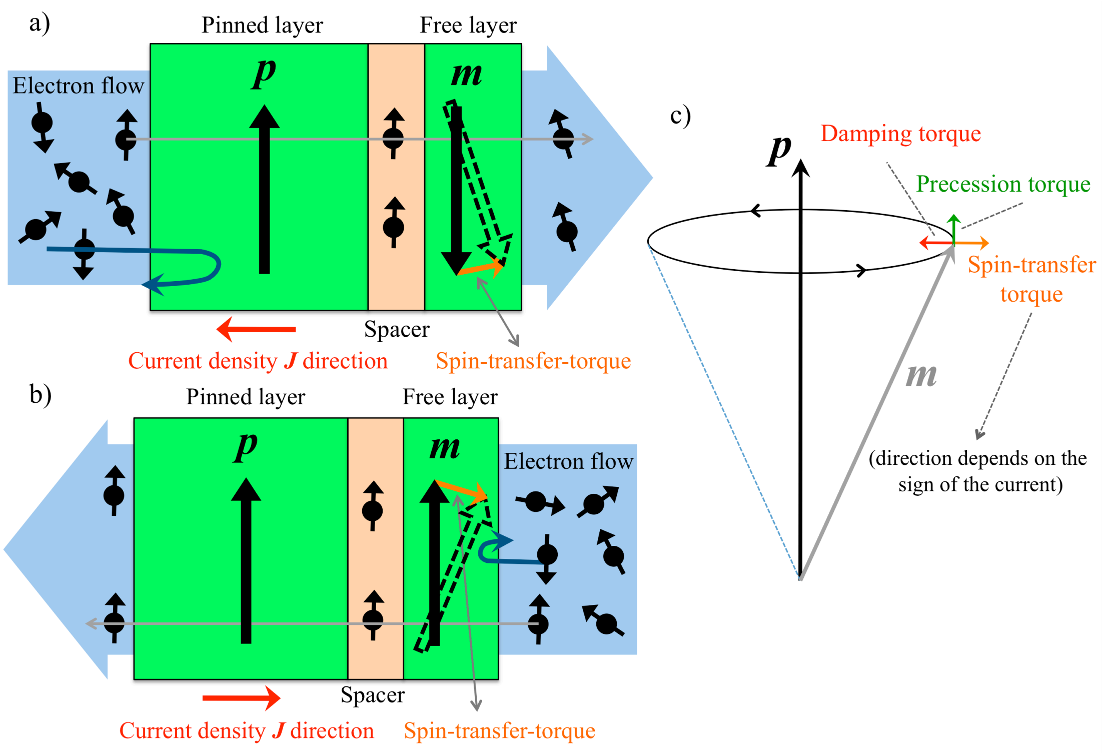

Fig. 9 – Representation of the STT acting on a spin-valve device. a) When the magnetizations of both layers are anti-parallel, the STT that acts on the free layer ***m*****,** comes from the transmitted electrons that have been polarized by the pinned layer ***p***. b) When magnetizations of both layers are parallel, the STT that acts on the free layer comes from the reflecting electrons that flow through it. c) Direction of all the torques being applied to the magnetization ***p*** in the presence of both the ***H****eff* and spin polarized current.

When considering that both layers are dynamic the dynamic equation used (53) is the same but now as to be evaluated twice, once for each layer, taking the care that the thin layer is acting in one of the expressions as the polarizing layer and on the other as the one suffering the effects of the spin-torque, and vice-versa for the thick layer. Meaning that when evaluating, at each time step, the dynamics of the thin layer it is used the equation as in (53), whereas for the thick layer it would read,

$$\begin{aligned}\left(1+\alpha^2\right)\frac{d\boldsymbol{p}}{dt} &= -\gamma_0\left[\boldsymbol{p}\times\boldsymbol{H}_{eff}+\alpha\,\boldsymbol{p}\times\left(\boldsymbol{p}\times\boldsymbol{H}_{eff}\right)\right] \\ &\quad -\frac{g\mu_B\boldsymbol{j}\mathcal{P}(\boldsymbol{p}.\boldsymbol{m})}{2M_S d|q_e|}\left[\boldsymbol{p}\times\left(\boldsymbol{p}\times\boldsymbol{m}\right)-\alpha\left(\boldsymbol{p}\times\boldsymbol{m}\right)\right]\end{aligned} \tag{54}$$

This gives rise to what is usually called back-torque due the fact that at each time step the STT effect of one layer over the other is evaluated.

Current In-Plane (CIP).

Although the nature of STT is the same in either CPP or CIP, that is due to exchange interaction between the non-collinear spin carrying electrons with the local magnetization, the torque description is different. In order to study the magnetization dynamics involving in-plane currents, like in the study of DWs, two sources of torque are required to describe the dynamics, one adiabatic and the other non-adiabatic.

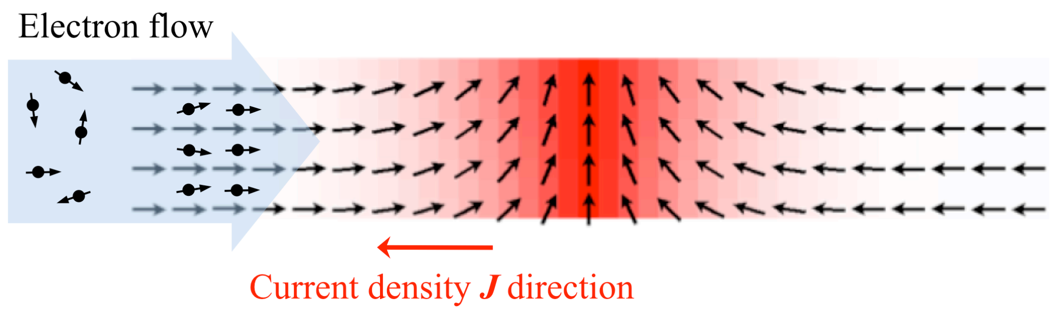

Fig. 10 – Schematic representation of the electron's spin polarization when using CIP in a long stripe with the presence of a domain wall.

A current flowing through a metallic ferromagnet is naturally polarized by it (Fig. 10), and when said current encounters a DW the relevant microscopic interaction is again the exchange interaction between the local magnetization $\boldsymbol{m}$ and the spin carrying electrons. This interaction affects the DW in two different ways. One is the momentum transfer, or force, and the other the spin transfer, or torque.

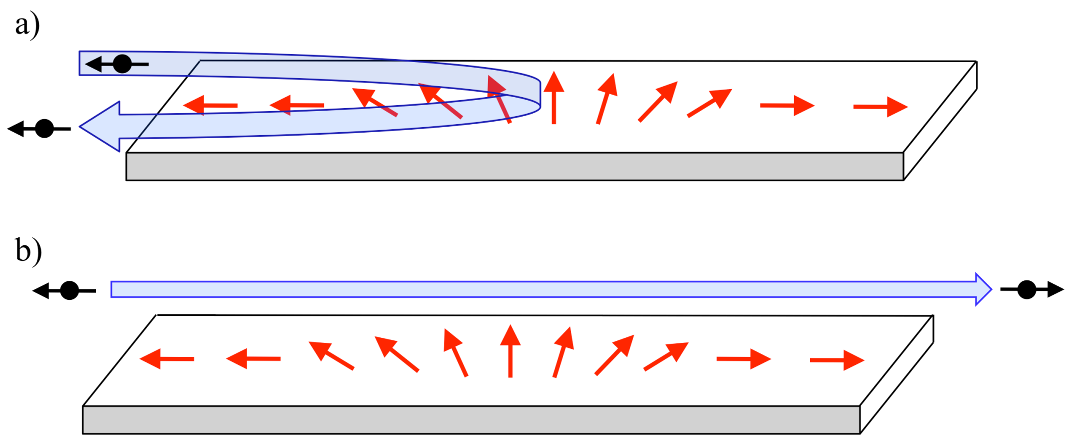

Fig. 11 – Two possible effects when a electric current encounters a domain wall due to the exchange interaction between the conduction electrons and the local magnetization. a) A reflected electron has transferred linear momentum to the domain wall. b) An adiabatically transmitted electron has transferred spin angular momentum to the domain wall.

Consider a metallic ferromagnet containing a single DW, and suppose there is an electron flowing from left to right (Fig. 11), if the electron is reflected by the DW its momentum is changed. This process acts as a force on the DW by transferring linear momentum from the electron to the DW (Fig. 11 (a)). On the other hand, if the electron is transmitted through the DW adiabatically, namely, by keeping its spin direction closely parallel to the local magnetization, the spin angular momentum of the electron is changed (Fig. 11 (b)). This process acts as a torque on the domain wall by transferring the spin angular momentum from the electron to the DW. In other words, this change of the electron spin should be absorbed by the magnetization, which might lead to the translational motion of the DW. In thick DWs the reflection probability is very small and thus the spin-transfer effect will be the dominant driving mechanism. This leads to what is denominated as the adiabatic torque τ_{adi}, which is given by [4],

$$\boldsymbol{\tau}_{adi} = \frac{d\boldsymbol{m}}{dt} = -\frac{g\mu_B\eta}{2M_S|q_e|}(\boldsymbol{j}.\nabla)\boldsymbol{m} \tag{55}$$

where in this case η is a spin polarization factor, which basically represents which percentage of the electrons is spin polarized along the direction of the local magnetization. The quantity $\boldsymbol{v}_{sp\text{-}drift}=\boldsymbol{j}g\mu_B\eta/2M_S|q_e|$ is generally called the spin-drift velocity and is actually the maximum velocity that the DW can reach in the adiabatic limit.

However since the theories developed to describe the STT in the adiabatic limit were not able to reproduce some experimental results (namely for thin DW), it was suggested that the spin transfer was more complicated and that some non-adiabatic contributions must be present. This non-adiabatic torque was first introduced by Zhang *et al.* [33] and Thiaville *et al.* [34] and it is given by,

$$\boldsymbol{\tau}_{non-adi} = \frac{d\boldsymbol{m}}{dt} = -\beta\boldsymbol{m} \times \left[\frac{g\mu_B\eta}{2M_S|q_e|}(\boldsymbol{j}.\nabla)\boldsymbol{m}\right] \tag{56}$$

where β is the dimensionless non-adiabatic parameter [35],[36].

With the two previous torque contributions, adiabatic and non-adiabatic, it is possible to write the total spin-torque when applying CIP through a ferromagnetic material as,

$$\Gamma_{ST-CIP} = \frac{d\boldsymbol{m}}{dt} = \frac{g\mu_B\eta}{2M_S|q_e|}\left[-(\boldsymbol{j}.\nabla)\boldsymbol{m} + \beta\boldsymbol{m} \times (\boldsymbol{j}.\nabla)\boldsymbol{m}\right] \tag{57}$$

Adding the spin torque term (57) to the dynamic equation (18), allows for the writing of the LLG equation that takes into account both adiabatic and non-adiabatic spin-torque terms for CIP,

$$\left(1+\alpha^2\right)\frac{d\boldsymbol{m}}{dt} = -\gamma_0\left[\boldsymbol{m}\times\boldsymbol{H}_{eff} + \alpha\boldsymbol{m}\times\left(\boldsymbol{m}\times\boldsymbol{H}_{eff}\right)\right] \\ -\frac{g\mu_B\eta}{2M_S|q_e|}\left[(1+\beta\alpha)(\boldsymbol{j}.\nabla)\boldsymbol{m} - (\beta-\alpha)\boldsymbol{m}\times(\boldsymbol{j}.\nabla)\boldsymbol{m}\right] \tag{58}$$

2.4 Recapitulation

During this chapter all of the relevant contributions to the magnetization dynamics that will be treated throughout this work, were discussed. Before continuing to the description on how each contribution is discretized, a small recapitulation is presented in order to recall all of the effective field $\boldsymbol{H}_{eff}$ components and the dynamic equation that is to be solved numerically.

Effective field

$$\boldsymbol{H}_{eff} = \boldsymbol{H}_{exch} + \boldsymbol{H}_{an,u} + \boldsymbol{H}_{dmg} + \boldsymbol{H}_{Ext} + \boldsymbol{H}_{Oe} + \boldsymbol{H}_{th} \tag{59}$$

were each component of the field is given by the previously seen equations.

Since when actually performing the numeric computations it is usually more practical to use the dimensionless form of the LLG dynamic equation they are written below for both CPP and CIP devices.

Dynamic equation for CPP devices

$$\left(1+\alpha^2\right)\frac{d\boldsymbol{m}}{d\tau} = -\left[\boldsymbol{m}\times\boldsymbol{h}_{eff} + \alpha\boldsymbol{m}\times\left(\boldsymbol{m}\times\boldsymbol{h}_{eff}\right)\right] \\ -\frac{1}{\gamma_0 M_S}\frac{g\mu_B\boldsymbol{j}\mathcal{P}(\boldsymbol{m}.\boldsymbol{p})}{2M_S d|q_e|}\left[\boldsymbol{m}\times\left(\boldsymbol{m}\times\boldsymbol{p}\right) - \alpha\left(\boldsymbol{m}\times\boldsymbol{p}\right)\right] \tag{60}$$

Note that if the simulating both the assumed magnetic free thin layer $\boldsymbol{m}$ and thick polarizing layer $\boldsymbol{p}$, care as to be taken while writing the dimensionless equation (60) since the saturation magnetization of each material M_S might be different.

Dynamic equation for CIP devices

$$\left(1+\alpha^2\right)\frac{d\boldsymbol{m}}{d\tau} = -\gamma_0\left[\boldsymbol{m}\times\boldsymbol{h}_{eff} + \alpha\boldsymbol{m}\times\left(\boldsymbol{m}\times\boldsymbol{h}_{eff}\right)\right] \\ -\frac{1}{\gamma_0 M_S}\frac{g\mu_B\eta}{2M_S|q_e|}\left[(1+\beta\alpha)(\boldsymbol{j}.\nabla)\boldsymbol{m} - (\beta-\alpha)\boldsymbol{m}\times(\boldsymbol{j}.\nabla)\boldsymbol{m}\right] \tag{61}$$

where in both previous equations it was considered that,

$$\tau = \gamma_0 M_S t \qquad \boldsymbol{m} = \frac{\boldsymbol{M}}{M_S} \qquad \boldsymbol{h}_{eff} = \frac{\boldsymbol{H}_{eff}}{M_S} \qquad \boldsymbol{p} = \frac{\boldsymbol{M}_P}{M_{PS}}$$

3 Micromagnetism numeric modelling

3.1 Introduction

This section is dedicated to the description on how the micromagnetism numeric modeling is performed, in order to study different interesting phenomena of the magnetization dynamics, from either the point of view of fundamental physics or from experimental devices.

As it has been seen thus far, micromagnetism is a semi-classical formalism, which allows for the study of the behavior of magnetic materials at the nano-scale. That behavior is described through the use of non-lineal partial differential equations, which are significantly complicated to solve and in general do not have analytical solutions. There are some analytical solutions but these are only for very simple geometries at low dimensions. That said, in micromagnetism there are two approximations, which can be seen as having opposite nature. On one hand, the discrete atomic system of magnetic moments is approximated by a continuous system as it was described in chapter 2. On the other hand, one has to discretize the previously obtained continuous expressions in order to solve the problem numerically.

The first part of this chapter is dedicated to the discretization of the geometry and of the dynamic equation, as well as all the different effective field contributions, using the finite difference method. In the second part, the numerical issues pertaining to the sequential type of programming are discussed, and its limitations will serve as a bridge to the last part of the chapter, where the spatial and temporal limitations are overcome by developing a parallel micromagnetic code using graphic processing units (GPUs). The chapter finishes by presenting comparative results that show the accuracy and efficiency of the developed parallel code.

3.2 Spatial discretization

As it was previously described, micromagnetism requires solving a set of non-lineal integral differential equations, which involve long-range (magnetostatic, Oersted), short-range (exchange, spin-torque) and local (anisotropy, external fields) interactions. These do not have analytical solutions except for very idealized cases [23],[37] and thus have to be solved numerically. In order to do that, one first has to spatially discretize the sample of the magnetic material. By doing so, the framework goes from a continuous system where the variable is the vectorial field $\boldsymbol{M}(\boldsymbol{r})=M_S\,\boldsymbol{m}(\boldsymbol{r})$, to a discrete system where now the variables are the values of the magnetization at each cell of the computational mesh, $\boldsymbol{M}(i,j,k)=M_S\,\boldsymbol{m}(i,j,k)$. There are two discretization approaches usually used in micromagnetics; the finite elements (*FE*) and finite difference (*FD*) methods. The *FE* approach [38] is based on interpolating the magnetization using linear basis functions on a non-uniform typically tetrahedral mesh. The *FD,* on the other hand, uses a uniform rectangular mesh, as it is shown in Fig. 12. The main disadvantage of the *FD* approach is that sampling curved boundaries with a rectangular mesh results in a staircase type approximation to the geometry, which introduces spurious effects. However in most simulations these effects are small enough to be ignored, and in the ones in which they are relevant some *ad hoc* techniques have been developed to hinder them [39]-[41]. The *FD* method is in general easier to implement, in particular when regarding the meshing of the sample, and it was the one adopted in this work.

Once the spatial discretization is achieved some considerations have to be taken into account about the size of each cell of volume ΔV (Fig. 12). Essentially, it has to be larger than the atomic scale, but not too large, so as to comply with the mesoscopic formalism of micromagnetics. In other words, it has to contain a sufficient number of magnetic spins in order to consider that the module of the magnetization M_S is constant in each cell, but small enough so that the magnetization can be considered as continuous function of the position inside the material. This leads to what is called the *characteristic length*, which in micromagnetism is the scale at which the magnetization varies significantly. Therefore, when numerically solving a problem it has to be ensured that the size of each computational cell is small enough when compared to the *characteristic length*. There are at least two characteristic scale lengths that are usually used in micromagnetics, and although an adequate cell size has to be chosen depending on the problem to solve these two serve as guide. They are the exchange length l_{ex} and the wall width l_w, and are given respectively by,

$$l_{ex} = \sqrt{\frac{2A}{\mu_0 M_S^2}} \tag{62}$$

and,

$$l_w = \sqrt{\frac{A}{K}} \tag{63}$$

where A is the exchange constant and K is the magneto-crystalline anisotropy constant. Roughly speaking the exchange length l_{ex} is the relevant scale when magneto-static dipolar interaction dominants over the magneto-crystalline anisotropy, as is the case when working with soft magnetic materials, whereas when considering materials with large anisotropy, the wall width l_w becomes the relevant scale.

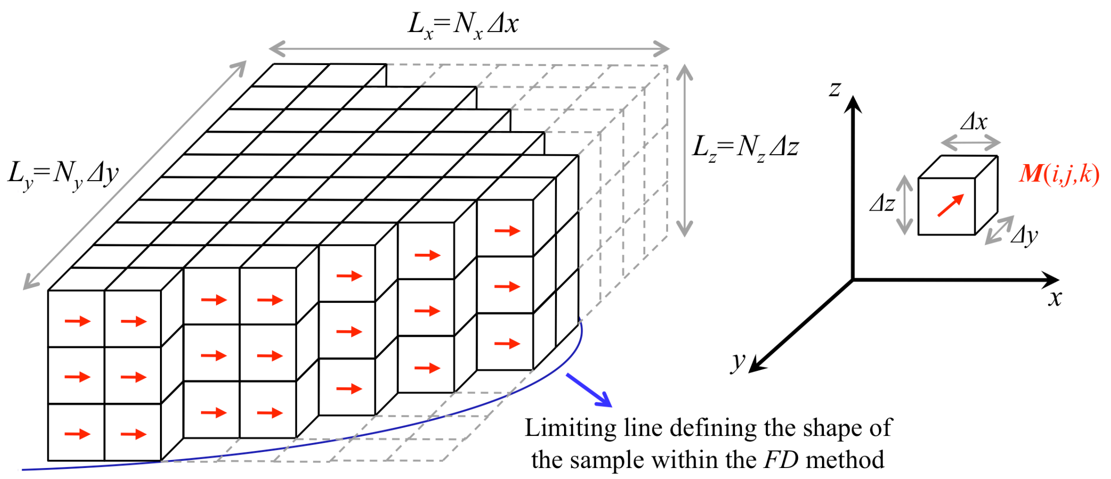

Fig. 12 – Representation of the discretization of the magnetic sample into a mesh of individual cells of volume $\Delta V=\Delta x\Delta y\Delta z$. Each cell is assumed to be uniformly magnetized with its magnetization equal to $\boldsymbol{M}(i,j,k) = M_S\boldsymbol{m}(i,j,k)$, where $i=1,\ldots,N_x$, $j=1,\ldots,N_y$, and $k=1,\ldots,N_z$ with N_x, N_y, and N_z representing the total number of cells in each Cartesian direction. Note that the total number of cells includes non-magnetic cells while discretizing the sample (empty cubes). These nonmagnetic cells are used in order to "draw" more complex structures like curvatures (in figure), notches, bumps, etc, when using the finite difference method. The dimensions of the sample are given by $L_n=N_n\Delta n$ (where $n\equiv x,y,z$).

In the most frequently used ferromagnetic materials the scales of both l_{ex} and l_w usually range between 4 nm and 8 nm, which sets some bounds to the cell size that is to be used in the simulations. Nonetheless this may vary depending on the type of problem to solve, since in more complex magnetic structures like Bloch-point walls ([42] and section 4.2.2) and others, the numeric precision can force the use of smaller cell sizes, which in turn leads to longer simulation times.

3.3 Discretization of the micromagnetic equations

Once the spatial discretization is achieved, the equilibrium (6) and dynamic (18) equations with and without the influence of the STT ((53) or (58)), have to be discretized as well in order to solve them numerically. Hence, this section is dedicated to finding the discrete counterparts to the continuous functions that describe all the contributions to the local effective field $\boldsymbol{H}_{eff}$, in each computational cell.

3.3.1 Exchange interaction discretization

From the continuous exchange energy density function (24) it is possible to express it for each computational cell of the mesh (i,j,k) as,

$$u_{exch}(i,j,k)=A\left[\left(\nabla m_x(i,j,k)\right)^2+\left(\nabla m_y(i,j,k)\right)^2+\left(\nabla m_z(i,j,k)\right)^2\right] \tag{64}$$

where,

$$\left(\nabla m_n\right)^2=\left(\frac{\partial m_n}{\partial x}\right)^2+\left(\frac{\partial m_n}{\partial y}\right)^2+\left(\frac{\partial m_n}{\partial z}\right)^2 \tag{65}$$

with *n*=*x*,*y*,*z*.

In the *FD* approximation the derivatives are replaced by ratios at the center of each cell of the mesh, thus the previous expression can be written as,

$$\left(\nabla m_n\right)^2\equiv\left(\frac{\Delta_x m_n}{\Delta x}\right)^2+\left(\frac{\Delta_y m_n}{\Delta y}\right)^2+\left(\frac{\Delta_z m_n}{\Delta z}\right)^2 \tag{66}$$

where Δx, Δy and Δz are the spatial distances in the three dimensions between each cell of the mesh, whereas Δ_x, Δ_y and Δ_z are the *FD* operators in each respective spatial direction. Considering now the unit magnetization vector $\boldsymbol{m}(i,j,k)$ at the point (i,j,k) and the first neighbor in the $+x$ direction, $(i+1,j,k)$ with magnetization $\boldsymbol{m}(i+1,j,k)$, the first term of (66) can be written as,

$$\begin{aligned}\left(\frac{\Delta_x m_x}{\Delta x}\right)^2&=\left(\frac{m_x(i+1,j,k)-m_x(i,j,k)}{\Delta x}\right)^2=\\&=\frac{m_x^2(i+1,j,k)-2m_x(i+1,j,k)m_x(i,j,k)+m_x^2(i,j,k)}{\Delta x}\end{aligned} \tag{67}$$

The second and third terms of (66) can be obtained in a similar manner. Taking into account that $|\boldsymbol{m}(i,j,k)|=1$ for all cells, equation (67) becomes,

$$\left(\frac{\Delta_x m_x}{\Delta x}\right)^2=\frac{1}{\Delta x^2}\left[2-2m_x(i+1,j,k)m_x(i,j,k)\right] \tag{68}$$

An analogous process can be made in order to obtain from (66) the first neighbor components for the positive *y* and *z* directions.

Naturally, the first neighbors in the negative directions of each Cartesian component have also to be considered, and thus from (66) (with *n*=*x*),

$$\begin{aligned}\left(\nabla m_x\right)^2=&\frac{1}{\Delta x^2}\left[\left(2-2m_x(i+1,j,k)m_x(i,j,k)\right)+\left(2-2m_x(i-1,j,k)m_x(i,j,k)\right)\right]+\\&+\frac{1}{\Delta y^2}\left[\left(2-2m_x(i,j+1,k)m_x(i,j,k)\right)+\left(2-2m_x(i,j-1,k)m_x(i,j,k)\right)\right]+\\&+\frac{1}{\Delta z^2}\left[\left(2-2m_x(i,j,k+1)m_x(i,j,k)\right)+\left(2-2m_x(i,j,k-1)m_x(i,j,k)\right)\right]+\end{aligned} \tag{69}$$

and similarly for $(\nabla m_y)^2$ and $(\nabla m_z)^2$. From the equation (69) the exchange energy density at each point of the mesh (64) can be written as,

$$\begin{aligned} u_{exch}(i,j,k) = {} & \frac{2A}{\Delta x^2}\sum_{i'j'k'}^{fN}\left[1-\boldsymbol{m}(i,j,k).\boldsymbol{m}(i',j',k')\right]+ \\ & +\frac{2A}{\Delta y^2}\sum_{i'j'k'}^{fN}\left[1-\boldsymbol{m}(i,j,k).\boldsymbol{m}(i',j',k')\right]+ \\ & +\frac{2A}{\Delta z^2}\sum_{i'j'k'}^{fN}\left[1-\boldsymbol{m}(i,j,k).\boldsymbol{m}(i',j',k')\right] \end{aligned} \tag{70}$$

where *fN* is the number of first neighbors in each Cartesian direction.

The total exchange energy (24) can now be approximated by,

$$\mathcal{U}_{exch} = \int_V u_{exch} dV \approx \sum_{i,j,k}^{N} u_{exch}(i,j,k)\Delta V \tag{71}$$

where $N=N_xN_yN_z$ is the total number of cells within the discretized sample.

In the discrete representation the functional derivative ($\delta/\delta\boldsymbol{m}\equiv\partial/\partial\boldsymbol{m}-\nabla.\partial/\partial(\nabla\boldsymbol{m})$) is converted into the ordinary derivative ($\partial/\partial\boldsymbol{m}$), which is valid when considering that the angles between the magnetization of a particular cell and its neighbors are small. Therefore the effective field contribution due to the exchange interaction can be calculated from,

$$\boldsymbol{H}_{exch}(i,j,k) = -\frac{1}{\mu_0 M_S}\frac{\partial u_{ex}(i,j,k)}{\partial \boldsymbol{m}(i,j,k)} \tag{72}$$

to give,

$$\begin{aligned} \boldsymbol{H}_{exch}(i,j,k) = \frac{2A}{\mu_0 M_S}\Bigg[& \frac{\boldsymbol{m}(i+1,j,k)+\boldsymbol{m}(i-1,j,k)}{\Delta x^2}+\frac{\boldsymbol{m}(i,j+1,k)+\boldsymbol{m}(i,j-1,k)}{\Delta y^2} \\ & +\frac{\boldsymbol{m}(i,j,k+1)+\boldsymbol{m}(i,j,k-1)}{\Delta z^2}\Bigg] \end{aligned} \tag{73}$$

Another important aspect arises from the variation of the exchange energy besides the abovementioned effective field contribution, which are the boundary conditions (BC) at the surface of the sample. The BC arising from the discontinuity of the exchange interaction at the surface is referred to as “free” BC, and it can be written as mentioned in the different references [23],[43],[44]. However, since throughout this work the surface anisotropy and interlayer exchange are not considered, the BC used are simply reduced to,

$$\frac{\partial \boldsymbol{m}}{\partial \boldsymbol{n}} = 0 \tag{74}$$

3.3.2 Anisotropy interaction discretization

The local interaction term of the anisotropy is trivially discretized from the uniaxial anisotropy energy density function (27) as,

$$u_{an,u}(i,j,k) = K\left[1 - \left(\boldsymbol{m}(i,j,k).\boldsymbol{u}_k\right)^2\right] \tag{75}$$

where $\boldsymbol{u}_k$ is the unit vector in the direction of the anisotropy, and thus the total anisotropy energy is given by,

$$\mathcal{U}_{an,u} = \int_V u_{an,u} dV \approx \sum_{i,j,k}^{N} u_{an,u}(i,j,k)\Delta V \tag{76}$$

As before, the effective field contribution of the anisotropy can be determined from the functional derivative of the uniaxial anisotropy energy density (75),

$$\boldsymbol{H}_{an,u}(i,j,k) = -\frac{1}{\mu_0 M_S}\frac{\partial u_{an,u}(i,j,k)}{\partial \boldsymbol{m}(i,j,k)} = \frac{2K}{\mu_0 M_S}\left(\boldsymbol{m}(i,j,k).\boldsymbol{u}_k\right)\boldsymbol{u}_k \tag{77}$$

3.3.3 Magnetostatic interaction discretization

As a non-local term, in order to make the discretization of the magnetostatic field $\boldsymbol{H}_{dmg}(\boldsymbol{r})$, it is necessary to take into account that this field at the point $\boldsymbol{r}(i,j,k)$ depends on the magnetization $\boldsymbol{m}(\boldsymbol{r'})$ of all other points $\boldsymbol{r'}(i',j',k')$ within the sample's volume. Assuming that the magnetization $\boldsymbol{m}(i,j,k)$ at each cell (i,j,k) of the mesh is uniform, the averaged magnetostatic field at each cell can be shown to be equal to [45],[46],

$$\boldsymbol{H}_{dmg}(i,j,k) = -M_S \sum_{\beta}^{(x,y,z)} \sum_{i'j'k'}^{N} \mathcal{N}_{\alpha\beta}\left(i-i',j-j',k-k'\right).\boldsymbol{m}_{\beta}(i',j',k') \tag{78}$$

where the sum is over the total number of cells N, and $\mathcal{N}_{\alpha\beta}\,(\boldsymbol{r}-\boldsymbol{r'})$ is a 3×3 symmetric tensor usually called the demagnetizing tensor and it is given by,

$$\mathcal{N}_{\alpha\beta}\left(\boldsymbol{r}-\boldsymbol{r}'\right) = \frac{1}{4\pi}\oint_{S_r}\oint_{S_{r'}} \frac{d\boldsymbol{S}_r d\boldsymbol{S}_{r'}}{\left|\boldsymbol{r}-\boldsymbol{r}'\right|} \tag{79}$$

where S_r and $S_{r'}$ are the surfaces of the cells at the positions $\boldsymbol{r}$ and $\boldsymbol{r}'$, whereas $\boldsymbol{S}_r$ and $\boldsymbol{S}_{r'}$ are the corresponding surface vectors $\boldsymbol{S}_r = S_r\,\boldsymbol{n}$ and $\boldsymbol{S}_{r'} = S_{r'}\,\boldsymbol{n}$, with $\boldsymbol{n}$ being the normal unit vector. As it is stated by (78) the evaluation of the magnetostatic field $\boldsymbol{H}_{dmg}$, requires the summation of all cells within the sample, which means that it involves a total of N^2 operations (note that this number of operations will be multiplied by three since each Cartesian component of the field is evaluated separately). All the other terms that contribute to the dynamic equation require only N operations, since they are local interactions or at the most short range as is the case of the exchange interaction. As a result, the simulations of samples composed by a large number of cells become prohibitive if trying to evaluate (78) directly. However, since the demagnetizing tensor $\mathcal{N}_{\alpha\beta}$ only depends on the relative position between the cells, ($\boldsymbol{r}$–$\boldsymbol{r}'$), and on the geometry of the sample, (78) can be recognized as a discrete convolution of $\mathcal{N}_{\alpha\beta}$ with $\boldsymbol{m}$. On one hand this means that $\mathcal{N}_{\alpha\beta}$ only needs to be calculated once at the beginning of the simulation, and on the other hand, the magnetostatic field can be computed more efficiently using Fast Fourier Transform (FFT) techniques [47],[48]. The reason why it is more efficient to use FFTs is that the convolution is converted into a simple product in the Fourier space, which will be briefly discussed below.

Each Cartesian coordinate of the magnetostatic field (78) $H_{dmg,n}$ ($n \equiv x,y,z$), has the form of a tridimensional discrete convolution in the coordinate space, which accordingly to the convolution theorem allows for the following product in the Fourier space,

$$\begin{aligned}\mathcal{F}\left[H_{dmg,x}(i,j,k)\right] \equiv \tilde{H}_{dmg,x}(k_x,k_y,k_z) = \mathcal{F}\left[\mathcal{N}_{\beta}(i,j,k)\right]\mathcal{F}\left[m_{\beta}(i,j,k)\right] = \\ = \tilde{\mathcal{N}}_{\alpha\beta}(k_x,k_y,k_z)\,\tilde{m}_{\beta}(k_x,k_y,k_z)\end{aligned} \tag{80}$$

where (k_x,k_y,k_z) are the coordinates in the Fourier space. The components $H_{dmg,y}$ and $H_{dmg,z}$ are calculated in a similar manner. Therefore, in order to calculate the magnetostatic field $\boldsymbol{H}_{dmg}$, it is first necessary to determine the demagnetizing tensor $\mathcal{N}_{\alpha\beta}$, perform its FFT and store it in memory. Once the magnetization at each point of the mesh is known, the FFT is also performed for it, followed by the product in Fourier space as described in (80). To retrieve the value of the magnetostatic field back into the "real" space one simply has to perform the inverse FFT of (80),

$$H_{dmg,x}(i,j,k) \equiv \mathcal{F}^{-1}\left[\tilde{H}_{dmg,x}(k_x,k_y,k_z)\right] \tag{81}$$

All of the previous operations amount to a total of six FFT (three direct for each Cartesian component of $\boldsymbol{m}$ to the Fourier space, and three inverse for each of them to bring the result of $\boldsymbol{H}_{dmg}$ back to the "real" space), plus three products for each component of $\boldsymbol{H}_{dmg}$ (80). This means that by using the FFT method the number of operations needed to calculate $\boldsymbol{H}_{dmg}$ is about $Nlog_2N$ for each FFT, which is a considerable improvement when compared to the N^2 operations that result when directly evaluating (78) (Fig. 13). Nonetheless some care has to be taken when using this technique, because the convolution carried by the FFT method assumes that the magnetization data present in the computational region repeats itself periodically in an infinite space. In order to avoid the artifacts derived from this, the *zero-padding* technique [50] is applied to the physical region where the magnetization data resides in the Fourier space. This technique consists in adding computational cells that surround the physical region with zero magnetization, in such a way that the total number of cells is at least the double of that in the physical region in each direction, *i.e.* $K_x \geq 2N_x$, $K_y \geq 2N_y$ and $K_z \geq 2N_z$ within the Fourier space (Fig. 14). By doing this, it is ensured that the periodicity of the augmented region does not affect the physical one when evaluating the FFTs.

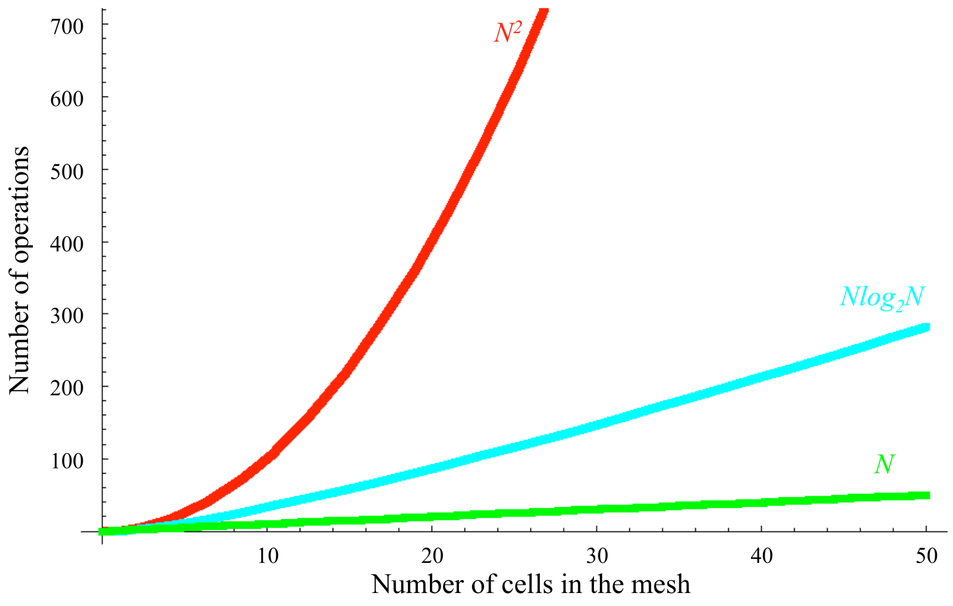

Fig. 13 – Comparing the number of operations needed in a small 50 cell mesh sample between: directly evaluating (78) to calculate $\boldsymbol{H}_{dmg}$, N^2 operations; using the FFT to calculate $\boldsymbol{H}_{dmg}$ (81) $Nlog_2N$; and local terms of the $\boldsymbol{H}_{eff}$, N.

The magnetostatic energy can be calculated numerically from (40) knowing that at each cell the energy density is,

$$u_{dmg}(i,j,k) = -\frac{1}{2}\mu_0 M_S\left[\boldsymbol{H}_{dmg}(i,jk).\boldsymbol{m}(i.j\,k)\right] \tag{82}$$

and thus the magnetostatic energy of the entire system is given by,

$$\mathcal{U}_{dmg} = \int_V u_{dmg}(\boldsymbol{r})\,dV \approx \sum_{i,j,k}^{N} u_{dmg}(i,j,k)\,\Delta V \tag{83}$$

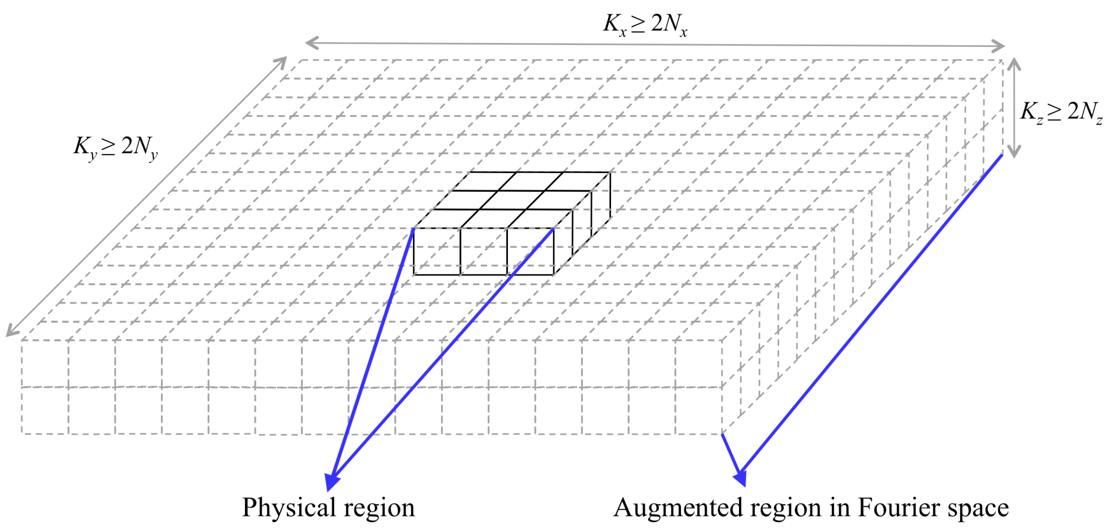

Fig. 14 – Representation of the physical space of size $N=N_xN_yN_z$ where the magnetization is solved in contrast with the augmented Fourier space where the demagnetization field $\boldsymbol{H}_{dmg}$ is calculated using the *zero-padding* technique. The magnetization is zero in the augmented region.

3.3.4 Zeeman interaction discretization

As the anisotropy interaction, the Zeeman one due to an external magnetic field is local, and thus the discretization of both its energy density and effective field contribution are trivially obtained. Therefore the Zeeman energy density of the system can be calculated numerically from (41) as,

$$\mathcal{U}_{Zee} = \int_V u_{Zee}(\boldsymbol{r})\,dV \approx -\mu_0 M_S \sum_{i,j,k}^{N} \boldsymbol{H}_{Ext}(i,j,k).\boldsymbol{m}(i,j,k) \tag{84}$$

where the modulus of the field $\boldsymbol{H}_{Ext}$ is chosen by the user and given as an initial parameter.

3.3.5 Oersted interaction discretization

Like the magnetostatic field, the Oersted field $\boldsymbol{H}_{Oe}$ generated by a certain flowing current density $\boldsymbol{j}$, is also dependent of all cells and on the geometry of the magnetic sample. However, unlike the magnetostatic field it does not depend on the magnetization $\boldsymbol{m}$, just on the value of the current density $\boldsymbol{j}(\boldsymbol{r},t)$ that runs through each individual cell. Therefore the expression for the Oersted field (43), can be discretized in a similar manner as it was done for the magnetostatic field in 3.3.3 to give,

$$\boldsymbol{H}_{Oe}(i,j,k) = -M_S \sum_{\beta}^{(x,y,z)} \sum_{i,j,k}^{N} \mathcal{M}_{\alpha\beta}(i-i',j-j',k-k') \times \boldsymbol{j}_{\beta}(i',j',k') \tag{85}$$

In this case $\mathcal{M}_{\alpha\beta}$ is an anti-symmetrical tensor that depends on the relative position between, the center of the cell in which the field is being calculated and the source cell generating the field [49]. Since equation (85) is a discrete convolution it can be more efficiently evaluated by using the FFT method in order to determine the value of $\boldsymbol{H}_{Oe}$ at each cell, in the same manner as it was discussed in section 3.3.3 for the magnetostatic field.

Similarly the energy density of this field can be calculated from (42) for each cell as,

$$u_{Oe}(\boldsymbol{r}) = -\frac{1}{2}\mu_0 M_S \sum_{i,j,k}^{N} \boldsymbol{H}_{Oe}(i,j,k).\boldsymbol{m}(i,j,k) \tag{86}$$

and the total energy,

$$\mathcal{U}_{Oe} = \int_V u_{Oe}(\boldsymbol{r})\,dV \approx \sum_{i,j,k}^{N} u_{Oe}(i,j,k)\,\Delta V \tag{87}$$

3.3.6 Thermal interaction discretization and inherent issues

The discretization of the thermal field described by equation (48) is pretty straightforward, since it already represents the thermal fluctuation for an individual cell, as it was proposed by Brown [28]. Therefore when performing stochastic simulations one simply has to calculate the value of (48) for each cell, since the dynamic equation (53) or (58), usually referred to as the Langevin equation, already includes in the effective field the $\boldsymbol{H}_{th}$ contribution. Nonetheless, something has to be said about the validity of the Langevin formalism when applied to micromagnetism, since there might be some debate to if expanding the single domain result obtained by Brown to the micromagnetic formalism is adequate.

From the theoretical point of view it bears the question of whether the correlation properties of the noise derived by Brown for a single magnetic moment [28], are applicable to a system in which the magnetic moments strongly interact with each other. In particular, there is the question of whether the noise should remain uncorrelated in space and time between the cells, and if not, how should both space and time correlations, as well as the strength of the noise, be determined [51],[52]. It has also been pointed out that the model produces errors at high temperatures, in particular it fails in reproducing the Curie temperature of a magnetic sample [53],[54]. Another shortcoming is that when thermal fluctuations are included the results strongly depend on the discretization of the sample, even for sizes smaller than the exchange length l_{ex} (62), or the wall width l_w (63) [54],[55]. All of the mentioned issues are not independent form each other and a model that solves all of them has yet to be achieved.

However despite the shortcomings of the model it does reproduce many results rigorously. The fundamental issue is that as long as it reproduces Maxwell-Boltzmann statistics in thermodynamic equilibrium, which is the condition imposed to determine the strength of the thermal field (47), it is considered satisfactory. The verification of such properties is described in the published work [56].

3.4 Numerically solving the dynamic LLG equation

In order to numerically solve the LLG equation including the spin-torque effect and all effective field components, (53) or (58), different ordinary differential solver algorithms can be implemented [50]. This section is thus dedicated to the discussion of some algorithms that show how to update the dynamics of the magnetization based on the discrete LLG equation, from the time t to $t+\delta t$.

3.4.1 Predictor-Corrector algorithm

The second order Predictor-Corrector uses as a basis the Euler method [50],

$$\begin{aligned} y_{n+1} &= y_n + h f(x_n, y_n) \\ \Rightarrow \quad \boldsymbol{m}(t+\delta t) &= \boldsymbol{m}(t) + \delta t\, f(t, m(t)) \end{aligned} \tag{88}$$

where $f(t,\boldsymbol{m}(t))$[6], is equal to,

$$\begin{aligned} f(t,\boldsymbol{m}(t)) = \frac{d\boldsymbol{m}(t)}{dt} &= -\frac{\gamma_0}{\left(1+\alpha^2\right)}\left[\boldsymbol{m}(t)\times\boldsymbol{H}_{eff}(t)+\alpha\boldsymbol{m}(t)\times\left(\boldsymbol{m}(t)\times\boldsymbol{H}_{eff}(t)\right)\right] \\ &- \frac{g\mu_B \boldsymbol{j}(t)\mathcal{P}(\boldsymbol{m}(t).\boldsymbol{p}(t))}{\left(1+\alpha^2\right)2M_S d|e|}\left[\boldsymbol{m}(t)\times\left(\boldsymbol{m}(t)\times\boldsymbol{p}(t)\right)-\alpha\left(\boldsymbol{m}(t)\times\boldsymbol{p}(t)\right)\right] \end{aligned} \tag{89}$$

Equation (88) is used to predict the value of the magnetization $\boldsymbol{m}$ at the instant $(t+\delta t)$, thus for the first step the Predictor is defined as,

$$\begin{aligned} f_{Pre}(t,\boldsymbol{m}(t)) &= -\frac{\gamma_0}{\left(1+\alpha^2\right)}\left[\boldsymbol{m}(t)\times\boldsymbol{H}_{eff}(t)+\alpha\boldsymbol{m}(t)\times\left(\boldsymbol{m}(t)\times\boldsymbol{H}_{eff}(t)\right)\right] \\ &- \frac{g\mu_B \boldsymbol{j}(t)\mathcal{P}(\boldsymbol{m}(t).\boldsymbol{p}(t))}{\left(1+\alpha^2\right)2M_S d|e|}\left[\boldsymbol{m}(t)\times\left(\boldsymbol{m}(t)\times\boldsymbol{p}(t)\right)-\alpha\left(\boldsymbol{m}(t)\times\boldsymbol{p}(t)\right)\right] \end{aligned} \tag{90}$$

The Predictor $f_{Pre}(t,\boldsymbol{m}(t))$ is used to predict magnetization at the instant $\boldsymbol{m}(t+\delta t)$ as described in (88). Said value of the magnetization has to be normalized before the

[6] Note that the mention expression is for the CPP spin-torque term and thus should be changed to the CIP spin-torque (57) accordingly to what one wishes to calculate.

next step, since the module is not conserved in the operation, and afterwards it is used to calculate the effective field $\boldsymbol{H}_{eff}(t+\delta t)$. Recall that the $\boldsymbol{H}_{eff}$ is a function of the magnetization and thus has to be evaluated every time the magnetization changes. The Corrector $f_{Cor}(t+\delta t,\boldsymbol{m}(t+\delta t))$ is then evaluated from $\boldsymbol{m}(t+\delta t)$, which was calculated from the Predictor step and $\boldsymbol{H}_{eff}(t+\delta t)$ as,

$$\begin{aligned} f_{Cor}\left(t+\delta t,\boldsymbol{m}(t+\delta t)\right) = &-\frac{\gamma_0}{\left(1+\alpha^2\right)}\Big[\boldsymbol{m}(t+\delta t)\times\boldsymbol{H}_{eff}(t+\delta t) \\ &+\alpha\boldsymbol{m}(t+\delta t)\times\left(\boldsymbol{m}(t+\delta t)\times\boldsymbol{H}_{eff}(t+\delta t)\right)\Big] \\ &-\frac{g\mu_B\boldsymbol{j}(t+\delta t)\mathcal{P}(\boldsymbol{m}(t+\delta t).\boldsymbol{p}(t+\delta t))}{\left(1+\alpha^2\right)2M_S d|e|}\Big[\boldsymbol{m}(t+\delta t)\times\left(\boldsymbol{m}(t+\delta t)\times\boldsymbol{p}(t+\delta t)\right) \\ &-\alpha\left(\boldsymbol{m}(t+\delta t)\times\boldsymbol{p}(t+\delta t)\right)\Big] \end{aligned} \tag{91}$$

Once both Predictor and Corrector terms have been calculated, the final value of the magnetization at $\boldsymbol{m}(t+\delta t)$ is given by,

$$\boldsymbol{m}(t+\delta t)=\boldsymbol{m}(t)+\frac{\delta t}{2}\left(f_{Pre}(t,m(t))+f_{Cor}(t+\delta t,m(t+\delta t))\right) \tag{92}$$

which again has to be renormalized since the method does not preserve the modulus of the magnetization.

3.4.2 Runge-Kutta algorithm

One of the most used algorithms to solve differential equations is the forth order Runge-Kutta method, which basically states that at each step the derivative is evaluated four times, and has the following general form [50],

$$\begin{aligned} k_1 &= hf(x_n,y_n) \\ k_2 &= hf\left(x_n+\frac{h}{2},y_n+\frac{k_1}{2}\right) \\ k_3 &= hf\left(x_n+\frac{h}{2},y_n+\frac{k_2}{2}\right) \\ k_4 &= hf(x_n+h,y_n+k_3) \\ y_{n+1} &= y_n+\frac{k_1}{6}+\frac{k_2}{3}+\frac{k_3}{3}+\frac{k_4}{6}+O(h^5) \end{aligned} \tag{93}$$

where $f(x_n,y_n)$ is the differential equation to solve, in this case (53) or (58), h is the size of the step that takes the function from y_n to y_{n+1}, and $O(h^5)$ is the error term. Therefore, and using (53) as an example for the differential equation $f(x_n,y_n)$, the numerical algorithm (93) can be written accordingly to the micromagnetic formalism as,

$$k_1 = \delta t\, f\big(t, \boldsymbol{m}(t)\big)$$

$$k_2 = \delta t\, f\left(t + \frac{\delta t}{2}, \boldsymbol{m}(t) + \frac{k_1}{2}\right)$$

$$k_3 = \delta t\, f\left(t + \frac{\delta t}{2}, \boldsymbol{m}(t) + \frac{k_2}{2}\right)$$

$$k_4 = \delta t\, f\big(t + \delta t, \boldsymbol{m}(t) + k_3\big)$$

$$\boldsymbol{m}(t + \delta t) = \boldsymbol{m}(t) + \frac{k_1}{6} + \frac{k_2}{3} + \frac{k_3}{3} + \frac{k_4}{6} + O(h^5) \tag{94}$$

where $f(t,\boldsymbol{m}(t))$ is the same as in (89).

Notice that the effective field $\boldsymbol{H}_{eff}(t)$, is also a function of $\boldsymbol{m}(t)$, due to the exchange, anisotropy and magnetostatic field interactions, which means that at each step k_n the $\boldsymbol{H}_{eff}$ as to be calculated before evaluating $f(t,\boldsymbol{m}(t))$ (89). Also it has to be taken into account the fact that the Runge-Kutta method does not conserve the modulus of the magnetization, and thus the magnetization has to be renormalized at each step k_n. Solving equation (94) for every cell of the discretized sample will give the numeric solution to the dynamic equation (53) or (58) at each time step δt.

Higher order Runge-Kutta methods can also be implemented through the general expression [57],

$$k_1 = h\, f(x_n, y_n)$$

$$k_i = h\, f(x_n + c_i h,\, y_n + h\sum_{j=1}^{i-1} a_{ij} k_j) \qquad i = 2, \ldots, s$$

$$y_{n+1} = y_n + \sum_{i=1}^{s} b_i k_i \tag{95}$$

where the term s is the order or stage number of the method, and the parameters c_i, b_i and a_{ij}, are determined according to a set of constraints [57] that depend on the order of the method. The sixth order method was also implemented using the general expression in (95) [57], and applying it to the micromagnetic formalism the same way as described above for the forth order method.

3.5 A look into sequential programming

For mainly historic reasons, the sequential language mostly used within the scientific media has been Fortran. IBM first developed it in the 1950s and it rapidly became the dominant programming language not only within the scientific community, but also in the engineering one. The main reason for its continued success within those communities is in particular attributed to its great efficiency in array programming, as well as other features involving its original goal of being a ***For**mula* ***Tran**slating* system. Such properties kept this language very much alive until recent times, even when languages such as C, which appeared in the early 1970s, came offering a more efficient mapping of typical machine instructions, and a more general-purpose programming. Nonetheless, Fortran stayed strong due to its facility and efficiency when solving arrays involving different numeric equations. For those reasons the sequential micromagnetic code, from which this work was based on, was written in Fortran.

This section is dedicated to briefly exposing how the sequential code is set in order to solve the magnetization dynamics ((53) or (58)), followed by the limitations inherent to it, which led to the will of developing a faster and more efficient parallel code.

3.5.1 Stages of the sequential micromagnetic code

As the name suggests, in a sequential micromagnetic code each of the discretized cells, which form the magnetic sample under study, are solved one by one sequentially. Therefore the dynamic equation ((53) or (58)) has to be solved at each time step δt once for each cell of the sample. As an example, if trying to solve the dynamics in a Permalloy parallelepiped with dimensions L_x=200 nm, L_y=60 nm and L_z=8 nm discretized in 4×4×4 nm^3 cells, the dynamic equation has to be solved 1500 times for each time step δt. The total number of evaluations of the dynamic equation is much larger than that since the dynamic equation is solved once for each Cartesian component of the magnetization $\boldsymbol{m}$, and either solver method presented in 3.4 implies a much larger number of evaluations per time step.

The working principle of the sequential programming is pretty straightforward and its basic running process can be seen in the simplified flow diagram of Fig. 15. It is clear from the diagram that once all the simulation parameters are defined it should be decided whether if the simulation to perform is static or dynamic.

The static simulation is usually used to determine the equilibrium state, and it is governed by a final equilibrium criterion. The criterion is that the cross product $(\boldsymbol{m}\times\boldsymbol{H}_{eff})\leq\mathcal{Er}$, where $\mathcal{Er}$ is the maximum value of misalignment between $\boldsymbol{m}$ and $\boldsymbol{H}_{eff}$ allowed, so as to say that the system is in equilibrium. In this case the simulation will run as many iterations as it needs in order to reach that equilibrium condition or until it reaches the maximum number of iterations allowed by the user. On the other hand, the dynamic simulation will run until the allotted time window Δt, for it is reached by sequentially increasing the time step δt, at the end of each solver algorithm evaluation.

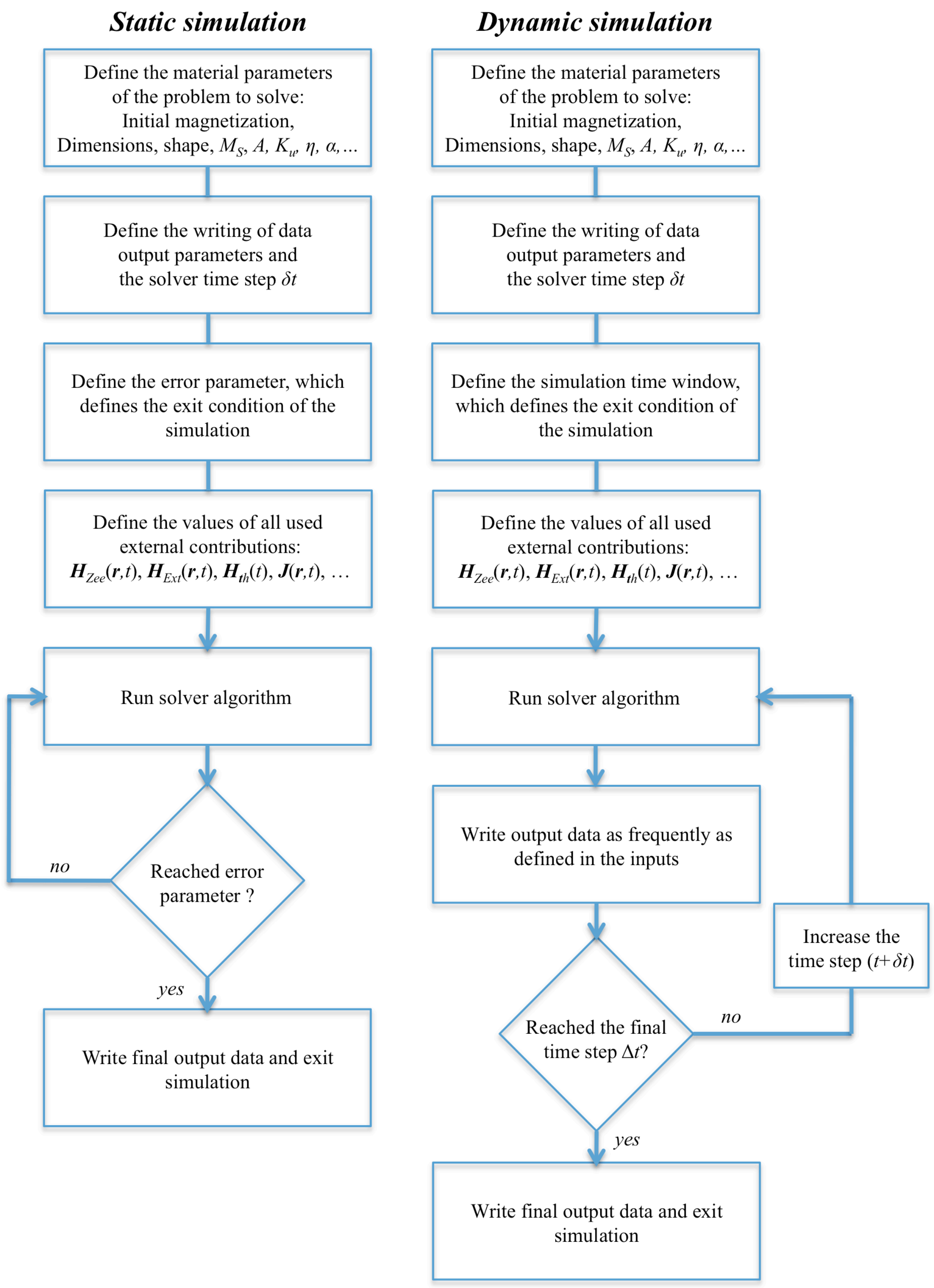
Static simulation
Dynamic simulation
Define the material parameters of the problem to solve: Initial magnetization, Dimensions, shape, M_S, A, K_u, η, α, ...
Define the writing of data output parameters and the solver time step δt
Define the error parameter, which defines the exit condition of the simulation
Define the simulation time window, which defines the exit condition of the simulation
Define the values of all used external contributions: $\boldsymbol{H}_{Zee}(\boldsymbol{r},t)$, $\boldsymbol{H}_{Ext}(\boldsymbol{r},t)$, $\boldsymbol{H}_{th}(t)$, $\boldsymbol{J}(\boldsymbol{r},t)$, …
Run solver algorithm
Reached error parameter ?
no
yes
Write final output data and exit simulation
Write output data as frequently as defined in the inputs
Reached the final time step Δt?
Increase the time step ($t+\delta t$)

Fig. 15 – Flux diagram for a typical micromagnetic sequential code.

Now a more detailed look at the stages that the sequential code runs through in an example of a dynamic simulation is given:

1. Define all material parameters, sample dimensions, cell size, shape of the sample, initial magnetization, solver time step δt parameter and final time Δt, and data output writing parameters.
2. Start the micromagnetic simulation, by reading all of the defined parameters into proper simulation variables.
3. Calculate the demagnetizing tensor $\mathcal{N}_{\alpha\beta}$ (and anti-symmetrical tensor $\mathcal{M}_{\alpha\beta}$ if the Oersted field is to be consider).
4. Call the desired algorithm to solve the dynamic equation.
5. Within the solver calculate the effective field components before each step of the algorithm. This has to be done because the effective field is a function of the magnetization due to the magnetostatic, anisotropy and exchange interactions.
6. Write the output data as frequently as defined by the user in the inputs.
7. Exit the simulation once the pre-defined time window is reached.

Notice, for example, that in a sequential code step 5 is done for each cell of the discretized magnetic sample through the use of a cycle, which for a sample composed of many cells takes a considerable amount of time to calculate.

3.5.2 Advantages and limitations of sequential programming

The code from the research group where this work was developed was in sequential Fortran, and while it is debatable to say if it is more efficient than other sequential languages such as C one thing is for sure, programming sequentially is much simpler than in parallel. However it does bring important limitations when compared to the parallel one.

The great limitation nowadays has to do with the ever-rising demand for computer power in order to solve not only a numerical problem faster but also with larger geometry. Even though the rise in computational power continues as it is suggested by Fig. 16, a technological foreseen limitation for central processing units (CPUs) became evident in the first decade of the 21st century. Until then the computational power of any given machine, was basically measured by the clock speed of the processor. This was continuously increased up to sensitively ≈4.5 GHz (larger clock speeds can be achieved but using non-conventional cooling systems like water cooled or liquid nitrogen), by increasing the number of transistors per unit of area, driven by the technological advancements in the miniaturization of such crucial components, down to the nano-scale. Of course the problem here is the amount of heat generated by such densely packed transistors, which when trying to reach higher clock speeds basically melted the device. The way to continue, not only the computational power demand but also power efficiency, was to start producing multi-core processors. However, like PC clusters that were already being used in super-computation, native programming languages like Fortran or C do not naturally run in multi-cores, they have to be programmed that way. Thus nowadays multi-core processors as power efficient and computationally evolved as they are, they are more oriented for multi-tasking not high performance computing. Also, the generation of multi-core processors demanded new operating systems that could take advantage of the multi-core capabilities, which imply new compatible language compilers. Therefore in order to try and have higher performances in numeric computations a different approach should be considered.

3.6 Mircromagnetics using Parallel programming in GPUs

3.6.1 Why parallel computing in GPU's and not PC clusters?

Computer clusters have existed for some time now and they basically consist of the desired number of individual computer units connected by fast local networks, forming nodes that can be seen as a single system.

This actually was the first idea proposed by the research group in which this work was developed, in order to reduce the time spent on each micromagnetic simulation. Although this system would have, in principle, the advantage of not having to significantly change the original micromagnetic code developed by the group, it had other important inconveniences. First and most importantly the cost, at the beginning of the work in 2009 the price of a computer cluster with the same computational power of a top of the line NVIDIA GPU [58] was more or less 10 times higher. And then there were the running costs and physical space issues needed to accommodate a computer cluster. Not to mention that although it would be theoretically possible to continue program in Fortran, this language is not natively prepared to run in concurrent nodes and thus accelerate each simulation in a true parallel way. To do that it is necessary to write the code in a true parallel fashion between the cluster CPUs, which is something that none of the members of the research group was familiar with.

The group thus turned its attention to NVIDIA's GPU Compute Unified Device Architecture (CUDA) [59], which is a parallel computing platform and programming model that enables impressive increases in computing performance by harnessing the power of said GPUs (Fig. 16). One of the main advantages of using CUDA is of course the price per gigaflop of computational power and the fact that it is natively a general purpose parallel language, being a mere extension of the C/C++ language with special functions called kernels. These instruct the computer to execute the desired algorithms within the GPU instead of the CPU. Details on how this actually works will be given in the next section. Another advantage is regarding the compiler, since more recent versions of compilers are usually not free, but in the case of the CUDA language it is freely distributed by NVIDIA.

The programming language CUDA is not without its drawbacks, among them it can be pointed out the fact that it is restricted to NVIDIA's graphic cards and the problem to solve has to fit inside the memory of the graphic card (if needed this can be overcome by programming an hybrid system, although it is considerably more complicated for newcomers, since it implies the need to balance the CPU computational power with the GPU, concurrently). Nonetheless, the advantages far compensate the limitations in the case of micromagnetic simulations (as well as in many other research fields [60]), since it allowed for speed-ups roughly two orders of magnitude higher than a CPU-based code, as it will be shown further.

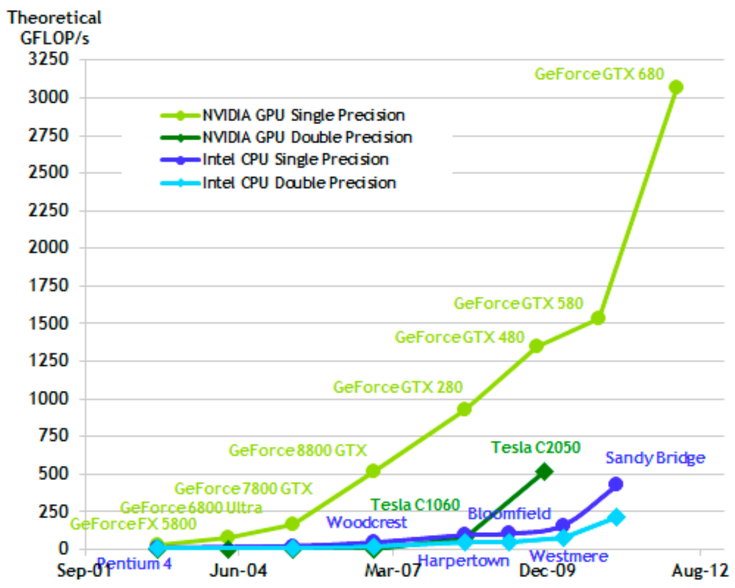

Fig. 16 – Floating-point operations per second evolution for the CPU and GPU over the last decade [61].

3.6.2 *Some considerations prior to CUDA parallel programming*

Before going into the challenges of implementing a parallel micromagnetic code in CUDA, some considerations should be made about what is needed in order to develop it. As a first consideration, it is recommended to be familiar with the C/C++ programming language, since CUDA is built from it and thus will facilitate the transition to this new language.

The next consideration is an essential one, which is to get a CUDA capable graphic card, since it is an exclusive NVIDIA technology. However, NVIDIA is mainly a chip manufacture and thus many other companies assemble CUDA capable graphic cards, especially in the GeForce line of cards. This line of cards is mostly known due to the original goal of this product, highly parallel computations for graphic rendering in video games and other applications. Although for a few years now all NVIDIA's graphic cards are CUDA capable, the Tesla line is the recommended one for numerical computations. The reason for it is that unlike the GeForce and Quadro lines, which are respectively designed for consumer graphics and professional visualization, the Tesla line was especially designed for parallel computing and programming, and thus offers exclusive high performance computing features. Nonetheless these cards are more expensive than the GeForce line, and thus it is up to the user/programmer to decide if his work requires the advantages of one line over the other, since high performance can also be achieved using the other lines of cards.

3.6.3 GPU hardware

The main reason for the recent application of GPU computing outside graphic applications, is that it has proven a powerful computation platform in many different areas such as, general signal processing, physics simulations, finance, computer biology, medicine, etc [60]. The reason for this tendency is that GPUs are specialized for compute-intensive, highly parallel computations and therefore when compared to a CPU, more transistors are devoted to data processing rather than caching and flow control, (Fig. 17). In contrast, GPU's typically have hundreds of floating-point execution units and large context switch information storage space, resulting in a small area remaining for cache. The cache is a small, very fast memory that stores copies of the most frequently used data from the main memory locations.

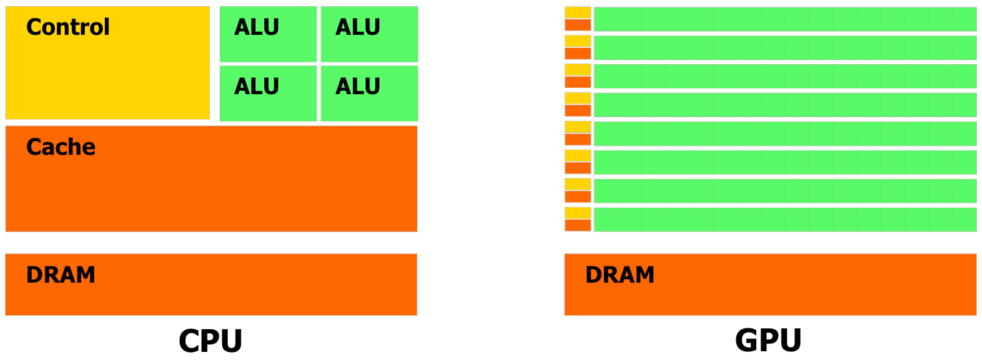

Fig. 17 – Schematic representation of a CPU and a GPU. The GPU devotes more transistors to data processing than the CPU, (ALU stands for Arithmetic Logic Unit) [61].

State-of the-art CPUs (at the writing of this thesis), such as the Intel i7-3970X, can only run 12 threads per core (double that if using HyperThreading), whereas one of the NVIDIA's cards that was used, the Tesla C2070 GPU has 448 cores to run the threads (the most recent Kepler GPU family has up to 2688 cores). Moreover, execution of concurrent threads in a CPU is generally time consuming and complicated to perform, since the operating system must swap threads on and off host execution channels in order to provide multithreading capability. Context switches (when two threads are swapped) are therefore slow and expensive. In contrast, threads on a GPU are extremely lightweight. In a typical system, thousands of threads are queued up for work (in 32 thread wraps), and if the device must wait on one wrap of threads, it simply begins executing work on another. Because separate registers are allocated to all active threads, no swapping of registers or state need occur between device threads. Resources stay allocated to each thread until execution is completed.

Since going into deeper understanding of the working hardware properties of GPUs and CPUs starts going outside the goals of the thesis, which is focused on programming a parallel solution for the micromagnetic formalism, it is recommended to consult the manufacture references [58] for more details. Nonetheless, the main idea is that each instruction or thread is handled by the GPU faster that in a CPU, since many more cores are available in a GPU to perform said instructions simultaneously.

3.6.4 CUDA programming model

As it was said before, CUDA is NVIDIA's parallel computing architecture that manages computation on its GPUs, and it does it by providing a simple interface for the programmer, based on the industry standard C/C++ with a few extensions. When programming in CUDA, the GPU is viewed as a *device* capable of executing a very large number of threads in parallel, and it operates as a coprocessor to the main CPU, which in turn is designated as the *host*. The *host* CPU manages the flow of the program by selecting the compute-intensive portions of applications that are to be performed in parallel, and it off-loads them from the *host* to the *device* memory. In other words the portion of an application that is executed many times (typically performed in a loop cycle in a sequential code), but is independent on different data, can be isolated into a function that is executed simultaneously on the *device* (GPU) in many different threads. To that effect, such a function is compiled accordingly to the instruction set of the *device*, and the resulting function, called a kernel in CUDA, is downloaded to the *device*.

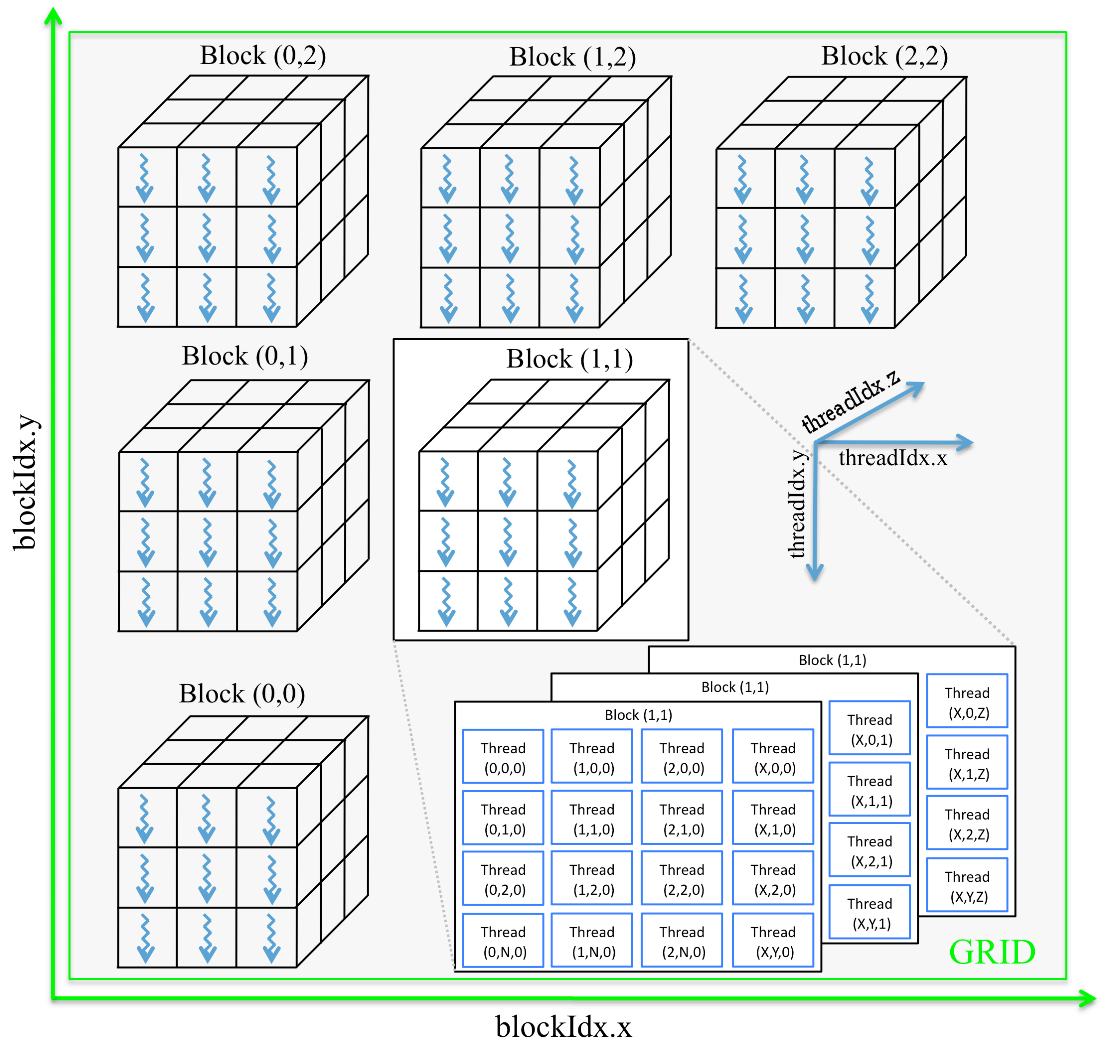

Fig. 18 – Schematic representation of the indexation of the thread blocks inside a grid. Note that each block can be three-dimensional and thus indexed by the three-component vector *threadIdx.x, .y, .z*. In this case the grid is two-dimensional and thread blocks within it are addressed by the two-component vector, *blockIdx.x, .y*.

When a kernel is called, the function is executed N times in parallel by N CUDA threads, which is different from a regular C function (or Fortran or any other sequential language) that is executed only once at a time. A kernel is defined using the `__global__` declaration specifier (there are also others but this is the general one [61]). Code that is written to be executed in both the *host* and the *device* can be contained in a single source file with a ".cu" extension. The code compilation is executed using the *nvcc* CUDA C-compiler, in which the resulting executable file coordinates the execution of the *host* and *device* components accordingly to the C runtime for CUDA [62].

As shown in Fig. 18 when a kernel is called, the threads within it are organized into blocks, who in turn are organized into grids, whose dimensions are specified when using <<<…>>>, a CUDA execution configuration syntax (see section 3.6.5 for more details). Each thread has a unique identification (ID), which is only accessible within a kernel through the *threadIdx* variable, which is a three-component vector (Fig. 18). Therefore, threads can be identified using either one-dimensional, two-dimensional, or three-dimensional thread indexes, which in turn may form one-dimensional, two-dimensional or three-dimensional thread blocks (Fig. 18). There is a limit to the number of threads per block, since all threads within one block are executed in the same processor core and must share the limited memory resources. This limit depends on the GPU architecture being used [61].

Blocks are organized into a grid of thread blocks, (Fig. 18, Fig.19). This grid may also be one-dimensional, two-dimensional or three-dimensional. The number of thread blocks within a grid is usually indicated by the size of the data being processed or by the number of processors in the system (if there is a lot of data to be processed there would be one block per processor core). Each block within the grid can also be identified by a one-dimensional, two-dimensional or three-dimensional index accessible within the kernel through the *blockIdx* variable. The dimensions of the thread block are accessible within the kernel through the *blockDim* variable [61].

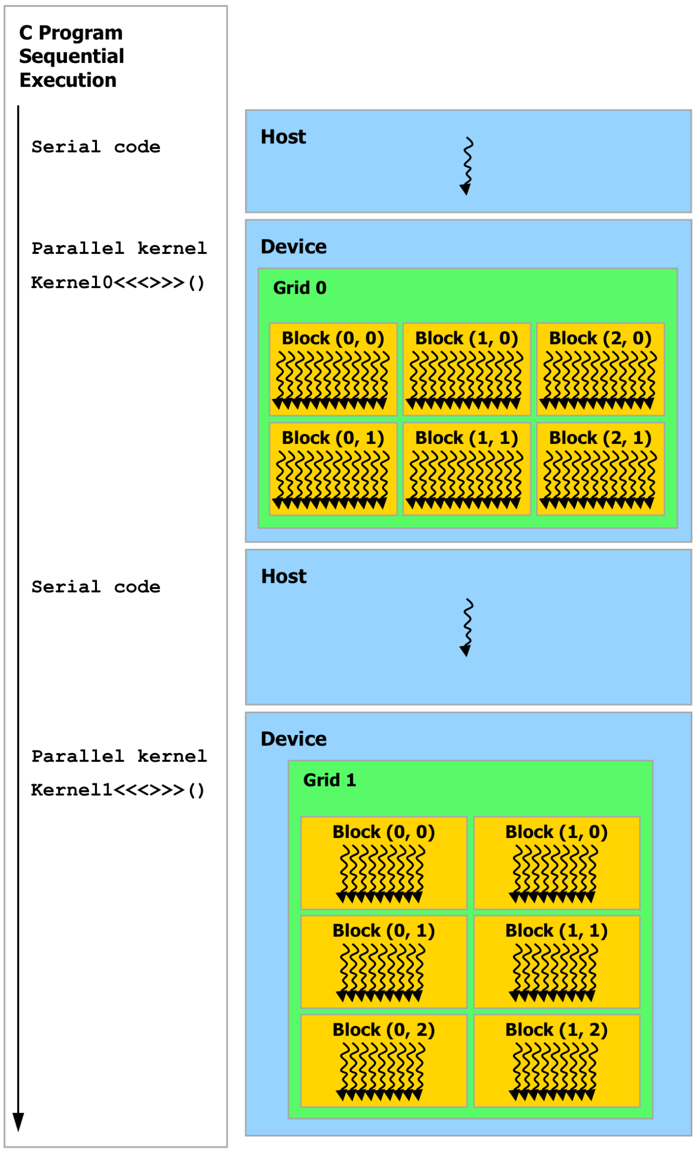

Fig. 19 – Schematic representation of the CUDA programming model, serial code executes on the *host* while parallel code executes on the *device*. The host issues a succession of kernel invocations to the device. Each kernel is executed as a batch of threads organized as a grid of thread blocks. In this case, both the blocks and grids are two-dimensional [61].

The CUDA memory hierarchy is depicted in Fig. 20. Unlike in the *host*, where the random access memory (RAM) is generally equally accessible to all code (within the limitations enforced by the operating system), on the *device* RAM is divided virtually and physically into different types, each of which has a special purpose. These memory spaces include global, local, shared, constant, texture and registers [61]. Among these memory spaces, the global and texture memories are the most abundant. Global, local and texture memory have the greatest latency, followed by constant memory, registers and shared memory. Because it is on-chip, shared memory is much faster than the local and global memories, so it is often used as a scratch pad to store intermediate results of computations, buffering reads and writes in order to achieve optimal memory access patterns and to provide inter-thread communication within a block. Local memory is so named because its scope is local to the thread, not because of its physical location. In fact, local memory is off-chip and access to it is as expensive as the global memory [63].

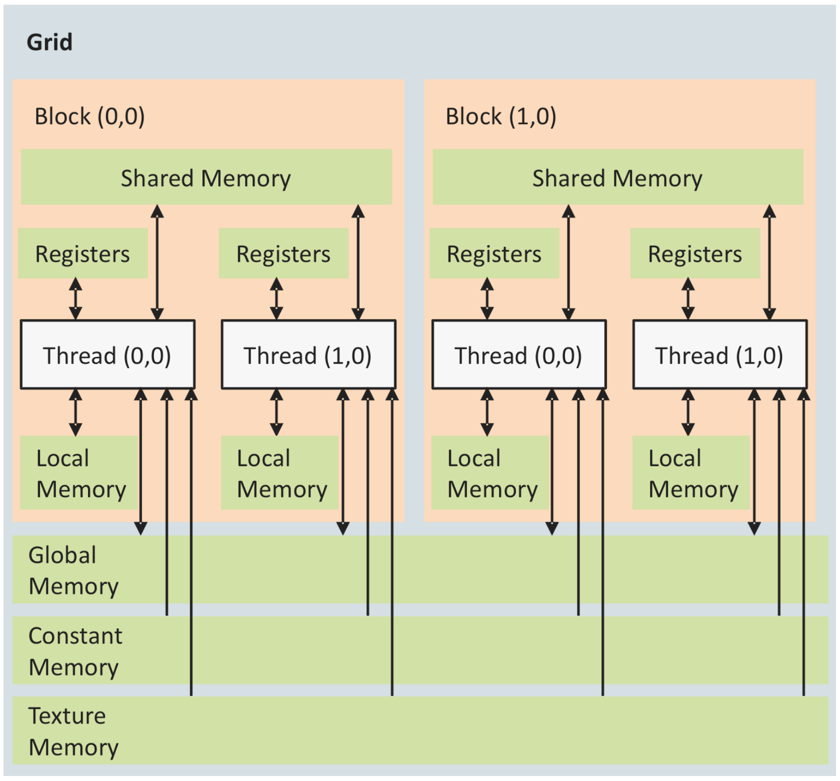

Fig. 20 – CUDA memory model. A thread has access to the device's DRAM and on-chip memory through a set of memory spaces of various scopes [61].

As in any other programming language, memory optimization is a very important aspect for performance increase, and particularly in CUDA a lot of care has to be given to the essential memory transfers between *host* and *device*. The maximum off-chip memory bandwidth (depends on the graphic card; 144 GB/s for the C2070 GPU [58]) inside the graphic card is much higher than the maximum bandwidth between the *host* memory and the *device* memory (8 GB/s on PCIe x16 Gen2 [63]). Consequently in order to achieve the best overall application performance, it is critical to minimize data transfer between the *host* and the *device*, even if that means running kernels on the *device* that do not demonstrate any speed-up compared with running them on the *host* [63]. These data transfers depend on the nature of the code to parallelize, or in other words on the granularity of the problem. In parallel computing, granularity is a qualitative measure of the ratio between the computation and communication. For a coarse (fine) granularity problem, relatively large (small) amounts of computational work are performed between communication events.

3.6.5 Some examples of CUDA programming

After the previous section it is possible that the reader, even if being familiar with typical programming architectures, to be a little confused on how exactly the CUDA language works when actually trying to parallelize a code. Therefore, this section is dedicated to explaining a few examples involving typical operations between arrays, emphasizing as much as possible the differences between typical sequential C language and parallel CUDA.

⇒ *Example 1: Adding two arrays and saving the result into a different array*

This example shows how different are the steps to take between the C and CUDA languages, in order to make a simple add operation between the corresponding elements of two arrays *a*[n] and *b*[n], where in this example the number of elements is n=5. The example will be divided in distinctive sections so as to better show the differences between the two languages (Check Appendix A for full code).

The first part of any code is usually the declaration of variables and, if necessary, their memory reservation and initialization. As can be seen in Fig. 21 in C it is only needed to declare the variables to use once, whereas in CUDA besides the variables one would create in C, the *host* variables defined has `h_a`, `h_b`, etc, it is also needed to create those same variables within the memory of graphic card, that is the *device* variables `dev_a`, `dev_b`, etc. In the CUDA case, the *host* variables are usually where the variables are initialized, and then they are uploaded to the *device* memory, to be changed accordingly to a certain function that is to be performed in parallel. A good example for a *host* variable within the micromagnetic formalism is the initial magnetization of the sample, which after it is read or initialized is sent to the corresponding variable that was created within the *device* memory. Once the magnetization information is in a *device* variable it is possible to solve the dynamics of the individual magnetic spins of the sample in parallel, using the discretized LLG equation, which is also solved within the *device* memory.

```
{
...
//Creating all the needed host variables and initializing them
   int ndim=5;
   float *a;   //
   float *b;   //Creating the arrays
   float *c;   //
   a = new float[ndim];                //Defining the dimensions of the array
   memset(a,0,sizeof(float)*ndim); //Initializing the array at 0
   b = new float[ndim];                //
   memset(b,0,sizeof(float)*ndim); //
   c = new float[ndim];                //
   memset(c,0,sizeof(float)*ndim); //
...
}                                                                        IN C
```

```
{                                                                        IN CUDA
...
/*Creating the variables on the host*/
   int ndim=5;
   float *h_a;
   float *h_b;
   float *h_c;
   h_a = new float[ndim];                //Defining the dimensions of the array
   memset(h_a,0,sizeof(float)*ndim); //Setting the array's memory at 0
   h_b = new float[ndim];
   memset(h_b,0,sizeof(float)*ndim);
   h_c = new float[ndim];
   memset(h_c,0,sizeof(float)*ndim);

/*Creating the variables on the device*/
   float *dev_a;
   float *dev_b;
   float *dev_c;
   //Reserving the array's memory within the device
   cudaMalloc((void**)&dev_a,sizeof(float)*ndim);
   //Setting the array's memory within the device at 0
   cudaMemset(dev_a,0,sizeof(float)*ndim);

   cudaMalloc((void**)&dev_b,sizeof(float)*ndim);
   cudaMemset(dev_b,0,sizeof(float)*ndim);
   cudaMalloc((void**)&dev_c,sizeof(float)*ndim);
   cudaMemset(dev_c,0,sizeof(float)*ndim);

/*Defining the size of the block and grid*/
   dim3 dimBlock(5);                  //Number of threads per block
   dim3 dimGrid(ndim/dimBlock.x); //Number of blocks within the grid
...
}
```

Fig. 21 – Creation of variables needed to perform an add operation between the element of two arrays of 5 elements *a*[] and *b*[], in both C language and CUDA. In CUDA more variables are required since the same variables need to be created inside the *device* memory in order to perform the calculation in parallel.

Also shown in Fig. 21, is the initialization of the variables that define the number of threads in each block, `dimBlock` in this example, and how many blocks are inside the grid, `dimGrid`. These two variables can also be defined at the moment of the kernel call the following way <<<`dimGrid,dimBlock`>>>. However, it is a good practice to define it before, keeping in mind that these are *dim3* type variables, *i.e.* they have *x, y* and *z* integer dimensions. If only one of those dimensions is defined, as it is shown in Fig. 21, all the others are initialized at 1 by default [61].

```
{
...
//Loop to set the values within each array and to perform the add operation
 for(int i=0 ;i<ndim; i++){
        a[i]=1+i;
        b[i]=3;
        c[i]=a[i]+b[i];
 }
//Loop to print the results on the screen
 for(int i=0 ;i<ndim; i++){
        printf("a[%d]=%1.1f   b[%d]=%1.1f    c[%d]=%1.1f\n",
                i,a[i],i,b[i],i,c[i]);
 }
...
}                                                                     IN C
```

```
{                                                                     IN CUDA
...
/*Loop to set the values of each array within the host memory*/
    for(int i=0 ;i<ndim; i++){
        h_a[i]=1+i;
        h_b[i]=3;
    }
/*Copying both arrays to the GPU device variables dev_a and dev_b*/
    cudaMemcpy(dev_a, h_a, sizeof(float)*ndim, cudaMemcpyHostToDevice);
    cudaMemcpy(dev_b, h_b, sizeof(float)*ndim, cudaMemcpyHostToDevice);

/* Kernel call - this function is performed within the device*/
    KernelName<<<dimGrid,dimBlock>>>(dev_a,dev_b,dev_c,ndim);

/*Returning the result of the kernel operation back to a host variable*/
    cudaMemcpy(h_c, dev_c, sizeof(float)*ndim, cudaMemcpyDeviceToHost);

//Loop to print the results on the screen
    for(int i=0 ;i<ndim; i++){
        printf("a[%d]=%1.1f   b[%d]=%1.1f   c[%d]=%1.1f\n",
                i,h_a[i],i,h_b[i],i,h_c[i]);
    }
...
}
```

Fig. 22 – In the C part, it is shown how to make the simple add operation between the arrays a[] and b[] within the same cycle *for* in which they are defined, followed by the print on screen instruction in order to verify the results. In the CUDA part, it is shown that first it is required to initialize the arrays with the desired values (*host* memory), and then they are copied to the *device* variables. Then the kernel function call can be made to perform the add operation using the *device* variables. When the kernel finishes it is necessary to bring the result from the *device* variable `dev_c` back to the *host* one `h_c`, and then perform the print on screen instruction so as to validate the result.

In C, after the declaration of variables it is possible to immediately perform the add operation using a *for* cycle. In this example, as it is shown in Fig. 22, it was chosen a cycle from 0 to 4 accounting for the 5 elements of the array, and the add operation saved into array c[] within the same *for* cycle in which the values of a[] and b[] are defined, for the sake of brevity. After the add operation in C the results can simply be printed on screen with the corresponding instruction in order to verify them. However, in CUDA more steps are required (Fig. 22), first as in C the *host* arrays are set with the desired values, afterwards those values have to be copied from the *host* to *device* memory, in order to perform the add operation in parallel. This is done by using the `cudaMemcpy(...,cudaMemcpyHostToDevice);` instruction shown in Fig. 22. To perform the device calculation the CUDA kernel function can now be called through the instruction `KernelName<<<...>>>(...);` (Fig. 22), which indicates the *host* to make the *device* function call described in Fig. 23, and also assigns the needed variables to perform such operation. The add operation is then calculated in parallel by assigning to each thread of the block an index (Fig. 23), which in the case of this example corresponds exactly to each coordinate of the array. The `if()` instruction in Fig. 23, is not actually necessary for this example, it is put there in order to not calculate more threads per block than the necessary ones, and to show that all the typical C language operators are valid inside a kernel.

```
/*Kernel function of type global with the corresponding variables declaration*/
__global__ void KernelName(float *dev_a,float *dev_b,float *dev_c, int ndim){

//Thread index variable used to address each element of the array to a thread
    int index = blockIdx.x * blockDim.x + threadIdx.x;
//Performing the add operation in parallel
    if(index < ndim){
        dev_c[index] = dev_a[index] + dev_b[index];
    }
//Instruction to guarantee the synchronization of threads within a block
 __syncthreads();
}
```

IN CUDA

Fig. 23 – Kernel function with the `__global__` qualifier that is executed on the *device*, which is callable from the *host*, and showing the declaration of the thread variable and the add operation between each coordinate of the arrays a[] and b[]. Also shown is the `__syncthreads()` instruction, which is used to coordinate communication between the threads of a same block that help in avoiding read-after-write, write-after-read, or write-after-write hazards, when some of the threads within a block access the same addresses in a shared or global memory.

Once the kernel finishes, in order to print the results of the performed operation it is needed to first transfer the results from the *device* back to the *host* memory as it is shown in Fig. 22 with the `cudaMemcpy(...,cudaMemcpyDeviceToHost);` instruction. Since it is a good practice to free the allocated memory of the variables that are no longer being used in Fig. 24 it is shown how to do it in both the C and CUDA examples.

From the previous example one can already have an idea on how the implementation of the dynamic equation ((53) or (58)) can be achieved in order to take advantage of the parallel computation capabilities of the NVIDIA's GPUs. Basically every time a parallelizable part of the micromagnetic code is identified (calculation of $\boldsymbol{H}_{eff}$ components, spin-torque, LLG equation, etc.), adequate *host* and *device* variables are created. Then a kernel function is associated to each calculation that is to be performed in parallel in the same way as it was shown in the example above (or see Appendix A for full code).

Other explicit examples could be given with more complex calculations between arrays, however this one already focuses the essential part on how to work with the parallel scheme of a kernel function in CUDA. Therefore, for more details on the programming techniques and syntax, it is recommended to consult the CUDA programming guide [61] and best practices guide [63], as well as a C/C++ guide.

```
{
...
    delete a;                   //
    delete b;                   // freeing the allocated memory of the arrays
    delete c;                   //
...
}
```

IN C

IN CUDA

```
{
...
/*Freeing the allocated host memory of the arrays*/
    delete h_a;
    delete h_b;
    delete h_c;
/*Freeing the allocated device memory of the arrays*/
    cudaFree(dev_a);
    cudaFree(dev_b);
    cudaFree(dev_c);
...
}
```

Fig. 24 – Freeing the allocated memory. In CUDA you should not only free the memory of the host variables but also the ones on the device.

⇒ *<u>Example 2</u>: Using the CUDA Fast Fourier Transform library CUFFT*

The parallel programming language CUDA offers a FFT library called CUFFT, which is modeled after one of the most popular and efficient CPU-based FFT libraries the FFTW [64]. The CUFFT provides a configuration mechanism called a *plan,* which pre-configures internal building blocks such that the execution time of the transform is as low as possible for the given configuration and particular GPU hardware selected. Therefore, when the execution function is called the actual transform takes place following the plan of execution. Accordingly to NVIDIA, this approach brings the advantage that once the user creates a *plan* the library retains whatever state is needed to execute it multiple times without recalculating the configuration. There are three types of FFTs supported by the CUFFT library in either single or double precision and they are: complex-to-complex, real-to-complex and complex to real [64].

From the results obtained during the development of the parallel code it was verified that the CUFFT is very effective, especially if compared to the sequential Fortran-based code, as can be asserted in section 3.7. However, in order to get the most out of it the developer should try to restrict the size along all dimensions, while performing the FFT, so that size can be factored as $2^a*3^b*5^c*7^d$, (where *a,b,c,d*= 0,1,2,3,…) [64], since accordingly to the Cooley-Tukey algorithm used by the CUFFT library those are the sizes for which the transform is highly optimized, (this can be verified in Fig. 33). Ideally the size along any dimension should be a multiple of 2, 3, 5 or 7, since for example a transform of size 3^n will likely be faster than one of size 2^a*3^b, even if the latter is slightly smaller [64].

The following description details how to perform a FFT using the CUFFT parallel library, in either the forward or inverse direction of a given array.

The simplest example is in computing a certain number (batch) of one-dimensional discrete transforms of size `nSize`, which using the CUFFT typically looks like the description in Fig. 25. There, it is shown that the first step needed is the creation of a *plan* variable using the `cufftHandle` declaration, which is followed by the definition of the *plan* using the `cufftPlan1d()` instruction that has the following configuration:

```
cufftPlan1d(cufftHandle *plan, int nSize, cufftType type, int batch);
```

where:

⇒ `plan`	is a pointer to the `cufftHandle` object
⇒ `nSize`	is the transform size, *i.e.* number of elements of the array to transform
⇒ `type`	is the transform data type, which is described in Table 1.
⇒ `batch`	is the number of transforms of size `nSize`.

Table 1 – Fast Fourier transform types within the CUFFT library

Types of FFTs, `cufftType`	Description of the type of transform
`CUFFT_R2C`	Real to complex
`CUFFT_C2R`	Complex to real
`CUFFT_C2C`	Complex to complex
`CUFFT_D2Z`	Double to double-complex
`CUFFT_Z2D`	Double-complex to real
`CUFFT_Z2Z`	Double-complex to double-complex

```
{
...
// Define the handle variable that allows access to CUFFT plans.
    cufftHandle plan;

// Create a 1-dimensional CUFFT plan of a float Complex to float Complex.
    cufftPlan1d(&plan, nSize, CUFFT_C2C,batch);

// Transform the data to the inArray to the outArray in the forward
// direction by execution the previously defined plan.
    cufftExecC2C(plan, inArray, outArray, CUFFT_FORWARD);

// Destroy the CUFFT plan.
    cufftDestroy(plan);
...
}
```

Fig. 25 – Typical steps when using the CUFFT library in order to perform a batch of one-dimensional transforms.

Once the *plan* is set, it can be executed to perform the desired transform using the `cufftExecC2C()` instruction (note that this instruction depends on the type of transform [64]), which has the following configuration:

```
cufftExecC2C(cufftHandle *plan, cufftComplex *indata, cufftComplex
            *outdata, int direction);
```

where:

⇒ `plan`	is the `cufftHandle` object for the plan to update
⇒ `indata`	is the pointer to the single-precision complex input data, in the GPU memory, to transform
⇒ `outdata`	is the pointer to the single-precision complex output data, in the GPU memory, to transform
⇒ `direction`	is the transform direction: `CUFFT_FORWARD` or `CUFFT_INVERSE.`

The last step of the transform execution should be to destroy the *plan*, so as to release the allocated resources of the *plan* using the `cufftDestroy()` instruction.

Different return values can be used in order to check if each command was successfully performed. However, since these are more advanced options of CUDA and although some of them were used in the developed code, for the sake of simplicity these were left out and the reader should consult the CUDA references [61]-[65] for more information and examples.

Since all the arrays that are used to compute the previously mentioned contributions to the dynamic equation ((53) or (58)), which are more efficiently calculated using FFTs, are three-dimensional arrays a more practical example is given next. (The full code of the example is shown in Appendix B).

```
{
...
/*Initializing the arrays*/
       for(int i=0 ;i<ndim; i++){
        h_a[i]= 1+i;
        h_b[i]= 3;
   }
/*Copying the values of h_a and h_b to the GPU device variables dev_a and dev_b*/
    cudaMemcpy(dev_a, h_a, sizeof(float)*ndim, cudaMemcpyHostToDevice);
    cudaMemcpy(dev_b, h_b, (sizeof(float)*ndim), cudaMemcpyHostToDevice);
/*Kernel call*/
    KernelName<<<dimGrid,dimBlock>>>(dev_a,dev_b,dev_c,nx,ny,nz);
...
}
...
{
...
__global__ void KernelName(float *dev_a, float *dev_b, cuFloatComplex *dev_c,
                             int nx, int ny, int nz){
    int index = blockIdx.x * blockDim.x + threadIdx.x;

     if(index < (2*nx*2*ny*2*nz)){
       if(index < (nx*ny*nz)){
          dev_c[index] = make_cuFloatComplex(dev_a[index] + dev_b[index], 0.0e0);
       }else{
          dev_c[index] = make_cuFloatComplex(0.0e0, 0.0e0);
       }
     } __syncthreads();
...
}
```

Fig. 26 – Reusing example 1 with the idea of performing the CUFFT on array `dev_c[]`, which contains the result of the add operation of arrays `h_a[]` and `h_b[]`. Note that since the transform to perform is of complex-to-complex `dev_c[]`, is declared as a float complex and inside the kernel the add operation in realized inside the real part of the `make_cuFloatComplex(real,imaginary)` instruction, so as to adapt the complex `dev_c[]`, using the floats `h_a[]` and `h_b[]`. Also note that the first *if* runs 8 times more than the total dimension of the problem `nx*ny*nz`. This is done in order to zero-pad the `dev_c[]` array as it is described in the figure.

As in *example 1*, in this example a simple add operation is also done between each coordinate of the arrays `h_a[]` and `h_b[]`. However in this case the array in which the operation is saved, array `dev_c[]`, is of float complex type (see Appendix B) in order to subsequently perform the Fourier transform. Therefore and since arrays `h_a[]` and `h_b[]` are of type float, inside the kernel besides the add operation, the adaptation to the complex type variable of array `dev_c[]` is also performed, as described in Fig. 26. Also note that inside the kernel shown in Fig. 26 the array `dev_c[]` is also being zero-padded in the regions outside the problem's data as described in section 3.3.3, in order to avoid the numeric artifacts of using FFTs. Once the array that is to be transformed using the CUFFT library is set with the wished values, it is possible to start creating the aforementioned *plan* of the transform. To do that it is useful to create a function that when called from the *host* code simply takes the dimensions, of the *in-array* to transform and of the *out-array* where the results of the transform are saved, as is described in Fig. 27.

As suggested by Fig. 27 the function prototype created, `cuFFT3DF()`, can be called multiple times to make the transform without having to create a new *handle plan*. This is very useful since several FFT calls are made during a micromagnetic simulation. When the function `cuFFT3DF()` is called, a *plan* is created for the three-dimensional complex-to-complex transform through the instruction `cufftPlan3d()`, followed by its execution with the `cufftExecC2C()` instruction in the forward direction (Fig. 27). Once that is done the `cufftDestroy(plan)` is executed in order to release the allocated resources. When finally exiting the `cuFFT3DF()` function, the desired values in Fourier space within the `outArray` (`fftdev_c` array in Appendix B) are ready to be used in further calculations within the GPU memory.

```
// Function prototype for the 3-dimensional FFT
void cuFFT3DF (int xSize, int ySize, int zSize, cuFloatComplex *inArray,
        cuFloatComplex *outArray);
{
...
// Calling the function to perform the FFT on array dev_c and
//save it to array fftdev_c
 cuFFT3DF (2*nx,2*ny,2*nz, dev_c, fftdev_c);
...
}
...
void cuFFT3DF (int xSize, int ySize, int zSize, cuFloatComplex *inArray,
        cuFloatComplex *outArray){
// Define the handle variable that allows access to CUFFT plans
    cufftHandle plan;
// Create a 3-dimensional CUFFT plan of a float Complex to float Complex array
    cufftPlan3d(&plan, xSize, ySize, zSize, CUFFT_C2C);
// Transform the data in the inArray to the outArray in the forward direction
    cufftExecC2C(plan, inArray, outArray, CUFFT_FORWARD);
// Destroy the CUFFT plan
    cufftDestroy(plan);
}
```

Fig. 27 – Steps needed to perform a three-dimensional FFT using the CUFFT library. The `cuFFT3DF()` function prototype shown defines the order in which each Cartesian variable should be placed as well as the *in* and *out* arrays. When the function is called the three-dimensional FFT is performed accordingly to the defined *plan*, which in this case is a complex-to-complex forward transform.

⇒ *<u>Example 3:</u> Using the independently developed CUDA library CUDPP*

As it was said before in order to parallelize a certain part of a code, said part cannot depend on other data. So how can an operation like a summation between each element of an array, be made without having to resource to a cycle? This would effectively mean one operation per element of the array and thus be in a sequential way. Operations like the mentioned summation are possible to perform in a more parallel, or in other words less sequential way, by the use of an independently developed library.

The CUDA Data Parallel Primitives Library (CUDPP) [65] is defined as a library of data-parallel algorithm primitives such as parallel prefix-sum (“scan”), parallel sort and parallel reduction. Primitives such as these are important building blocks for a wide variety of data-parallel algorithms, including sorting, stream compacting, and building data structures such as trees and summed-area tables. Although this library was independently developed by different contributors [66], and was initially developed to test the algorithms developed in C for CUDA, it is now freely available to anyone who whishes to use it with the CUDA runtime application interface. In order to use it however, care has to be taken between the versions of the CUDDP library and the CUDA version, since they may not be compatible.

The CUDPP library was used on one hand, to determine the average between the elements of an array, and on the other hand to determine the element of an array whose value is maximum.

In order to determine the summation of the elements within an array it is useful to create a function that will return the value of the sum by simply taking the array whose elements one wishes to sum and its dimensions. Since the magnetization variable to which this was applied is of type *float*, the function type created was of the same type, as it is shown in Fig. 28.

The first steps to take when using the CUDPP library is to set the configuration of the task to perform using the structure reference `CUDPPConfiguration` and giving it a name (`config` in Fig. 28). This structure is used to specify the algorithm, data type, operator and options of the task. Obviously depending on the configuration chosen it is possible to perform other operations with this library. Each component of the configuration is as follows [66];

⇒ `CUDPPAlgorithm`	`algorithm;`	is the algorithm to be used.
⇒ `CUDPPOperator`	`op;`	is the numerical operator to be applied.
⇒ `CUDPPDatatype`	`datatype;`	is the data type of the input arrays.
⇒ `Unsigned int`	`options;`	are the options to configure the algorithm.

```
// Function that returns the sum value of all the float elements
//within the array inArray
float Sum(float *inArray, size_t xSize, size_t ySize, size_t zSize){

// Setting up the configuration of the task to perform using the CUDPP library
    CUDPPConfiguration config;
    config.algorithm = CUDPP_SCAN;
    config.op = CUDPP_ADD;
    config.datatype = CUDPP_FLOAT;
    config.options = CUDPP_OPTION_BACKWARD | CUDPP_OPTION_INCLUSIVE;
// Since the successive summations have to be saved in a different array, a new
//array has to be created within the device memory
    static float *outArray;
    cudaMalloc((void**)&outArray,sizeof(float)*xSize*ySize*zSize);
    cudaMemset(outArray,0,sizeof(float)*xSize*ySize*zSize);
// Creating the handle variables needed to set up the plan
    CUDPPHandle cudppLibrary;
    cudppCreate(&cudppLibrary);
    CUDPPHandle scanPlan;
// Defining and executing the plan accordingly to the configuration set
    cudppPlan(cudppLibrary,&scanPlan,config,xSize*ySize*zSize,1,0);
    cudppScan(scanPlan,outArray,inArray,xSize*ySize*zSize);
// Destroy the plan of the handle variable scanPlan and all associated internal
//storage
    cudppDestroyPlan(scanPlan);
// Destroys the CUDPP library instance releasing allocated memory
    cudppDestroy(cudppLibrary);

    float sum;
// Copy the value of the summation stored in the first element of outArray to the
//host variable sum
    cudaMemcpy(&sum,&outArry[0],sizeof(float),cudaMemcpyDeviceToHost);
    cudaFree(outArray);      // Freeing the device memory allocated to outArray
// Return the value of the float function value sum
    return sum;
}
```

Fig. 28 – Defining a function that sums all of the elements of `inArray` and returns that value. The summation is done between two elements in a prefix-sum or cumulative sum as $y_0=x_0$, $y_1=x_0+x_1$, $y_2=x_0+x_1+x_2$,… and each y_n saved in a `outArray` in such a way that the first element of the array as the summation of all elements of `inArray`.

Following the description in Fig. 28, the algorithm chosen was the `CUDPP_SCAN`, which allows the scan of the elements of the array or to make a cumulative-sum as $y_0=x_0$, $y_1=x_0+x_1$, $y_2=x_0+x_1+x_2$, etc. The operator chosen was the `CUDPP_ADD`, which adds two operands and the data type `CUDPP_FLOAT`, since the array is of type float in this example. Finally the chosen options of the algorithm were, `CUDPP_OPTION_BACKWARD` that instructs the algorithm to operate backwards from the end to the start of the array, and `CUDPP_OPTION_INCLUSIVE`, which makes the scan include all the elements up to and including the current element. The option backwards was chosen so as to save the cumulative sum of the `inArray` components onto the first element of the `outArray`. Since an out array is needed to save the result of the operation defined in the configuration, the `outArray` is created within the memory of the device, as indicated in Fig. 28 after the configuration. Then two `CUDPPHandle` variables have to be created, the first one defined is followed by the `cudppCreate()` instruction, in order to create an instance of the CUDPP library which in turn returns a handle type, and this must be called before any other CUDPP function [65]. The second `CUDPPHandle` variable is used to set up the *plan* of the operation to perform. Once that is done, the plan can be defined in order to perform the sum, as it was set in the configuration, by using the `cudppPlan()` instruction, which has the following configuration [65]:

```
   cudppPlan(const CUDPPHandle cudppHandle, CUDPPHandle *planHandle,
CUDPPConfiguration config, size_t numElements, size_t numRows, size_t
rowPitch);
```

where,

⇒ `cudppHandle`	is a handle to an instance of the CUDPP library used for resource management.
⇒ `planHandle`	is a pointer to an opaque handle to the internal plan.
⇒ `config`	is the configuration structure specifying the algorithm and options.
⇒ `numElements`	is the maximum number of elements to be processed.
⇒ `numRows`	is the number of rows (2D operations) to be processed
⇒ `rowPitch`	is the pitch of the rows of input data, in elements

After this scan *plan* is defined it can be executed by using the `cudppScan()` instruction with the configuration [65]:

```
   cudppScan(const CUDPPHandle planHandle, void* outArray, void* inArray,
size_t numElements);
```

where,

⇒ `planHandle`	is a handle to plan for this scan.
⇒ `outArray`	is where the output scan is saved in GPU memory.
⇒ `inArray`	is the input to scan in GPU memory.
⇒ `numElements`	is the number of elements to scan.

Once the `cudppScan()` is completed both `CUDPPHandle` variables should be destroyed in order to properly release the resources allocated for each of them, as indicated in Fig. 28. The final part of the function is devoted to retrieving the desired sum of all the elements within `inArray`, which was saved to the first element of the `outArray` as defined in the configuration set up. This retrieval is described in the last part of Fig. 28 followed by the release of the no longer used memory, and then ending with the return of the value `sum` back to where the function was originally called.

The CUDPP library was also used to determine the element of an array whose value is maximum. This is useful, for example, to compare the cell within a sample with the maximum $\boldsymbol{m}\times\boldsymbol{H}_{eff}$ with the error parameter $\mathcal{Er}$, in equilibrium simulations, as mention in section 3.5.1.

The process to determine the maximum element of an array using the parallel CUDPP library is practically the same as the sum operation seen previously in Fig. 28. The only difference is in the configuration setup in which instead of defining an add operation (`config.op=CUDPP_ADD`) a search for the maximum between two operands (`config.op=CUDPP_MAX`) is defined, as shown in Fig. 29.

```
{
...
// Setting up the configuration of scan for maximum using the CUDPP library
    CUDPPConfiguration config;
    config.algorithm = CUDPP_SCAN;
    config.op = CUDPP_MAX;
    config.datatype = CUDPP_FLOAT;
    config.options = CUDPP_OPTION_BACKWARD | CUDPP_OPTION_INCLUSIVE;
...
}
```

Fig. 29 – Setting up the configuration in order to find the maximum element of an array, using the CUDPP library.

The next section is dedicated to the description of the micromagnetic parallel code implementation using the seen CUDA GPU programming language. Since the fundamentals of the language were already mentioned in this section there will be several references to these examples in the following section.

3.6.6 Making the parallel micromagnetic code

Here it is given a detailed explanation on how each part of the micromagnetic code is achieved when writing it in the CUDA parallel language, which was briefly introduced in the previous section. The description includes the difference between both spin-torques, meaning for when the excitation is with either CPP or CIP. The main differences between the torques are in the calculation of the spin-torque components, where the calculation of the CIP is pretty straightforward using (57) since just one material is involved. However, in the case of CPP devices more care has to be taken, since it has to be considered the dynamics of possibly two different magnetic materials, plus the back-torque interaction (section 2.3.7) between them.

The first thing to do when making a parallel code is to identify which parts of it can be done in parallel. In the case of the micromagnetic code developed this can be asserted by looking at the flux diagram in Fig. 30. There it can be seen that the sequential part controlled by the CPU, usually designated as the *host* in CUDA, manages the flow of the program by reading the input data and exporting it to the *device* accordingly to the parts of the code that are to be performed in parallel, as suggested in Fig. 19. (Note: Once the data is loaded into the device memory it stays there until freed). In micromagnetics the parts that naturally come to mind to parallelize are all of the calculations performed for each cell of the discretized magnetic sample. In other words all of the discretized expressions seen in 3.3 (in short all of the components of the $\boldsymbol{H}_{eff}$) and the dynamic equation using either of the algorithms seen in 3.4 with or without the spin-torque effect (2.3.7).

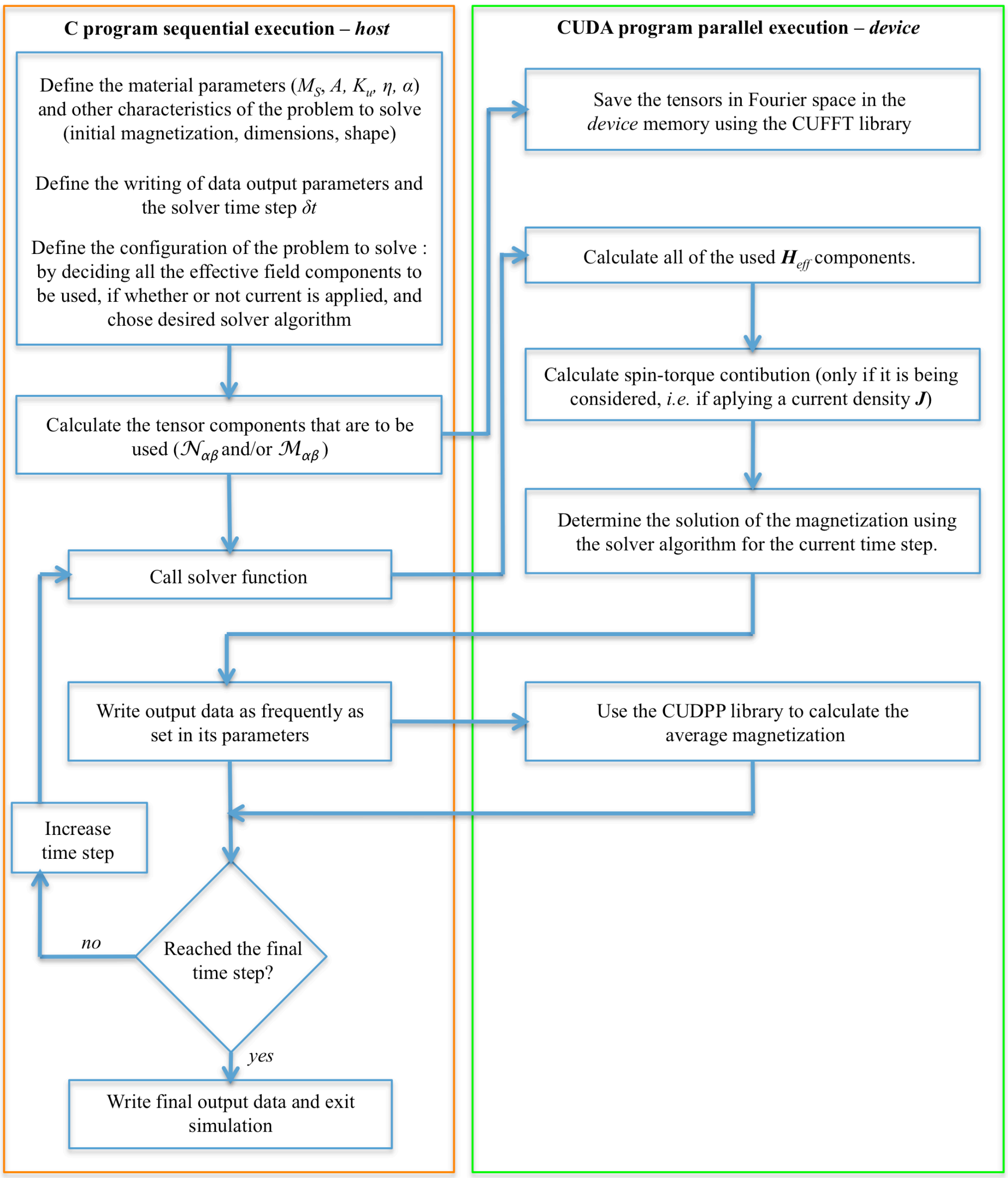
C program sequential execution – *host*
Define the material parameters (M_S, A, K_u, η, α) and other characteristics of the problem to solve (initial magnetization, dimensions, shape)
Define the writing of data output parameters and the solver time step δt
Define the configuration of the problem to solve : by deciding all the effective field components to be used, if whether or not current is applied, and chose desired solver algorithm
Calculate the tensor components that are to be used ($\mathcal{N}_{\alpha\beta}$ and/or $\mathcal{M}_{\alpha\beta}$)
Call solver function
Write output data as frequently as set in its parameters
Increase time step
Reached the final time step?
no
yes
Write final output data and exit simulation
CUDA program parallel execution – *device*
Save the tensors in Fourier space in the *device* memory using the CUFFT library
Calculate all of the used $\boldsymbol{H}_{eff}$ components.
Calculate spin-torque contibution (only if it is being considered, *i.e.* if aplying a current density $\boldsymbol{J}$)
Determine the solution of the magnetization using the solver algorithm for the current time step.
Use the CUDPP library to calculate the average magnetization

Fig. 30 – Flux diagram for the parallel CUDA code (dynamic simulation case), illustrating that all of the operations that are done for each cell of the sample are calculated in parallel within the *device*, and followed by the *host*, as it is suggested in Fig. 19.

Following the flow of the code described in Fig. 30, after the *host* reads all of the input data, without forgetting to properly allocate the *host* and *device* variables, the first calculations performed are the calculations of the components of the demagnetizing tensor (79) and/or the anti-symmetric tensor needed for the $\boldsymbol{H}_{Oe}$ (85). As it was previously mentioned, these tensors only depend on the relative position between the cells $\boldsymbol{r}_i$-$\boldsymbol{r}_j$ and thus need only to be calculated once at the beginning of the simulation. Once the tensors are calculated they are saved in the Fourier space by using the parallel CUFFT library, as described in example 2 of section 3.6.5, and thus stored within the *device* memory for further calculations. The next step will be to "tell" the *host* to instruct the *device* to start running the chosen solver algorithm, where the first part of it is the determination of all the contributions to the effective field $\boldsymbol{H}_{eff}$. Each field contributing to the $\boldsymbol{H}_{eff}$ is calculated through the use of a kernel, as the one described in example 1 of section 3.6.5, in order to calculate in parallel the value of the field in each computational cell by using the corresponding discrete field equation seen in 3.3. Once each individual field is calculated, the total $\boldsymbol{H}_{eff}$ is determined by the use of another kernel that calculates the total field sum in each cell in parallel.

Since the calculation of the magnetostatic field $\boldsymbol{H}_{dmg}$ is the most time consuming aspect of micromagnetic simulations a more detailed look will be given to this particular field. As it was seen in 3.3.3 the calculation of this field is a simple multiplication between the demagnetizing tensor and the magnetization in the Fourier space, given by (80). Since the demagnetizing tensor in Fourier space has already been previously saved within the *device* memory, the magnetization $\boldsymbol{m}$ also has to be transferred to Fourier space in a similar manner as in example 2 of section 3.6.5. First the kernel call `kernel_M_Calc<<<…>>>` is done so as to transform the float variable $\boldsymbol{m}$ into complex-float type, followed by the `CUFFT3F()` function calls that transform the data to the Fourier space (Fig. 31). Now that both the tensor and magnetization are in Fourier space, equation (80) can be solved in parallel by using the kernel call `kernel_H_Calc<<<…>>>`, where each corresponding cell of the tensor array `demagTensor.cuSD`*nn* (where *nn*=*xx, yy, zz, xy, xz* or *yz*) is multiplied with each corresponding cell of the array `hdmg.cuM`*n* (where *n*=*x, y* or *z*) and saved to array `hdmg.cuH`*n* (where *n*=*x, y* or *z*).

```
{
...
// Defining the block and grid sizes
    dim3 dimBlock(BLOCK_SIZE);
    dim3 dimGrid((Dim.Kx*Dim.Ky*Dim.Kz)/dimBlock.x +
                ((Dim.Kx*Dim.Ky*Dim.Kz)%dimBlock.x == 0?0:1));

// Kernel used to convert the magnetization from float to Complex float
    Kernel_M_Calc<<<dimGrid,dimBlock>>>(hdmg.cuMx,hdmg.cuMy,hdmg.cuMz,Dim.Kx,
            Dim.Ky,Dim.Kz,m.cuma,m.cumb,m.cumc,m.sizeX,m.sizeY,m.sizeZ);

// Functions that send the zero-padded dimensions and arrays to perform the forward
//CUFFT of the magnetization for each Cartesian component
    cuDFFT3F (Dim.Kx, Dim.Ky, Dim.Kz, hdmg.cuMx);
    cuDFFT3F (Dim.Kx, Dim.Ky, Dim.Kz, hdmg.cuMy);
    cuDFFT3F (Dim.Kx, Dim.Ky, Dim.Kz, hdmg.cuMz);

// Kernel where the magnetostatic field Hdmg is calculated in the Fourier space
    Kernel_H_Calc<<<dimGrid,dimBlock>>>(hdmg.cuHx,hdmg.cuHy,hdmg.cuHz,hdmg.cuMx,
            hdmg.cuMy,hdmg.cuMz,Dim.Kx,Dim.Ky,Dim.Kz,demagTensor.cuSDxx,
            demagTensor.cuSDyy,demagTensor.cuSDzz,demagTensor.cuSDxy,
            demagTensor.cuSDxz,demagTensor.cuSDyz);

//Functions that send the zero-padded dimensions and arrays to perform the inverse
CUFFT of the magnetostatic field for each Cartesian component
    cuDFFT3B (Dim.Kx, Dim.Ky, Dim.Kz, hdmg.cuHx);
    cuDFFT3B (Dim.Kx, Dim.Ky, Dim.Kz, hdmg.cuHy);
    cuDFFT3B (Dim.Kx, Dim.Ky, Dim.Kz, hdmg.cuHz);

//Kernel where the real part of the field is obtained
    Kernel_hdmg_Calc<<<dimGrid,dimBlock>>>(hdmg.cuhdmga,hdmg.cuhdmgb,hdmg.cuhdmgc,
            hdmg.sizeX,hdmg.sizeY,hdmg.sizeZ,hdmg.cuHx,hdmg.cuHy,hdmg.cuHz,
            Dim.Kx,Dim.Ky,Dim.Kz,inv.Kxyz);
...
}
```

Fig. 31 – Steps needed to calculate the $\boldsymbol{H}_{dmg}$ in parallel within the GPU. First the sizes of the block and grid are defined. Note that here the data is zero-padded as described in 3.3.3. Then the kernel `kernel_M_Calc<<<…>>>` is called to convert the magnetization from a float to a complex value. Once that is done the forward transform is performed by use a function created by the programmer that sends all of the information needed to perform the FFT as described in example 2 of section 3.6.5. This is followed by the kernel, `kernel_H_Calc<<<…>>>`, where the magnetization and demagnetizing tensor in Fourier space are multiplied component by component in parallel to give $\boldsymbol{H}_{dmg}$ in Fourier space. The last part is to perform the inverse transform of the calculated value followed by a kernel that extracts only the real part of $\boldsymbol{H}_{dmg}$.

Once the value of the magnetostatic field $\boldsymbol{H}_{dmg}$ is calculated in Fourier space the inverse transform of each component is performed by again using the CUFFT library (as described in example 2 in section 3.6.5). This is followed by the kernel call `kernel_hdmg_Calc<<<…>>>` function in which the field is converted from complex to float, and back to the real dimensions of the sample. The calculation of the coefficients of the demagnetizing tensor is performed in double precision and then the result of the field $\boldsymbol{H}_{dmg}$ truncated to single-precision. This is done on one hand to avoid the loss of precision due to the numerical behavior of the analytical formulas used to calculate said coefficients [67], and on the other hand to keep using, for the most part, single-precision variables since the CUDA code runs significantly faster with them, as suggested in Fig. 16. In terms of numerical “correctness” it was found sufficient enough to only calculate the demagnetizing tensor coefficients in double precision when performing the micromagnetic simulations. This can be asserted in the code validation section 3.7 where the developed CUDA parallel code is compared to the full double-precision sequential Fortran-based code.

When developing the code for CPP devices in which the dynamics of two different ferromagnetic materials has to be solved, (one viewed as the thick and the other as the thin layer of the device), the steps taken to calculate the $\boldsymbol{H}_{dmg}$ are the same. The difference here is that care has to be taken about the kernel’s index reference, in order to properly normalize the magnetization $\boldsymbol{m}$ being used in the calculation of the field in each layer, since the saturation magnetization M_S of each material might be different. Naturally the magnetostatic field interaction of one layer over the other is also being taken into account here, when calculating the magnetostatic field in each layer at each time step.

Continuing with the description of the flow diagram present in Fig. 30, if applying a certain current density through the sample the next step is to calculate the spin-torque effect. In either case of spin-torque, CPP or CIP, a kernel function is created in order to first calculate the spin-torque dependent parts of the dynamics equations (53) or (58). In the case of CPP device geometry some care has again to be taken with the kernel’s index reference, in order to multiply each ferromagnetic layer by the appropriate material parameters, and most importantly to avoid non-coalesced access to the *device* global memory when addressing the indexes to each layer of different material. One of the most single important performance considerations in CUDA architecture is to ensure that global memory accesses are coalesced whenever possible [61],[63]. Another important aspect to take into account regards to the range of the STT interaction, which is considered to be limited only to the adjacent layers of the ferromagnets with the spacer. Therefore more care is needed with the addressing of the indexes, since only the spacer adjacent layer of each ferromagnet is under the STT effect. (As an example if the thick layer is of 16 nm and the cell discretization is 4 nm in the *z*-axis direction, then the thick layer has 4 layers. The STT contribution is only considered for the layer adjacent to the spacer and not the other 3). Two kernels are used for the CPP device, one to calculate the STT on the thick ferromagnet and another for the thin ferromagnet. This allows for more configuration options when running a simulation since it is possible to have only the thick or the thin ferromagnets being solved dynamically as well as both of them.

In the case of the CIP device geometry, since there is only one ferromagnetic material whose dynamics has to be solved, the spin-torque contribution is simply calculated within the designated kernel by solving (57).

Once the spin-torque is calculated for each cell the dynamic equation (53) or (58) can be evaluated. Naturally this is done accordingly to either of the numeric algorithms described in section 3.4, where each stage of the algorithm progression is handled by a corresponding kernel function so as to perform it in parallel.

The full process is repeated until the total time window designated for the simulation is reached, writing the desired output data like the averaged magnetization ***m***, as often as decided by the user. Note that writing the output data involves retrieving data from the *device* memory back to the *host* memory, which as was mentioned previously is the most time consuming aspect of the CUDA parallel programming language. Therefore this process should only be done at these desired points of the simulation, avoiding as much as possible unnecessary memory transfers.

In the next section the validation of the developed parallel micromagnetic code using NVIDIA's CUDA is presented, in order to test its accuracy and efficiency by comparing it to CPU-based sequential solvers like OOMMF [68] and the Fortran code from which it was based on.

3.7 Parallel GPU micromagnetic code validation and performance

With the aim of testing the accuracy and efficiency of the developed GPU parallel code the Micromagnetic Modeling Activity Group (μMAG) [69] standard problem #4 will be presented, as well as a general performance test.

3.7.1 Standard problem #4

This standard problem focuses on the dynamic aspects of micromagnetic simulations. Accordingly to the information posted in μMAG website, about the standard problem #4, a rectangular elongated ferromagnetic sample of length L=500nm, width w=125nm and thickness t=3nm of typical Permalloy material parameters (A=1.3×10^{-11} Jm^{-1}, M_S=8.0×10^{5} Am^{-1}, K=0 Jm^{-3}) is considered. The initial magnetization used is an s-state obtained by saturating the sample along the [1, 1, 1] direction and then slowly removing the field. The interactions involved in this simulation are the magnetostatic, exchange and external fields, where a damping parameter of α=0.02 is considered. Two different external fields are applied, a field 1 of 25 mT oriented 170º counter-clockwise from the positive *x*-axis, and a field 2 of 36 mT directed 190º also counter-clockwise from the positive *x*-axis. The results are shown in Fig. 32 for the developed GPU parallel micromagnetic code. When comparing the solutions reported by the different groups [69], (one of which is from the research group in which this work was developed), with the results obtained using the developed parallel GPU code, the verdict is that it is completely satisfactory.

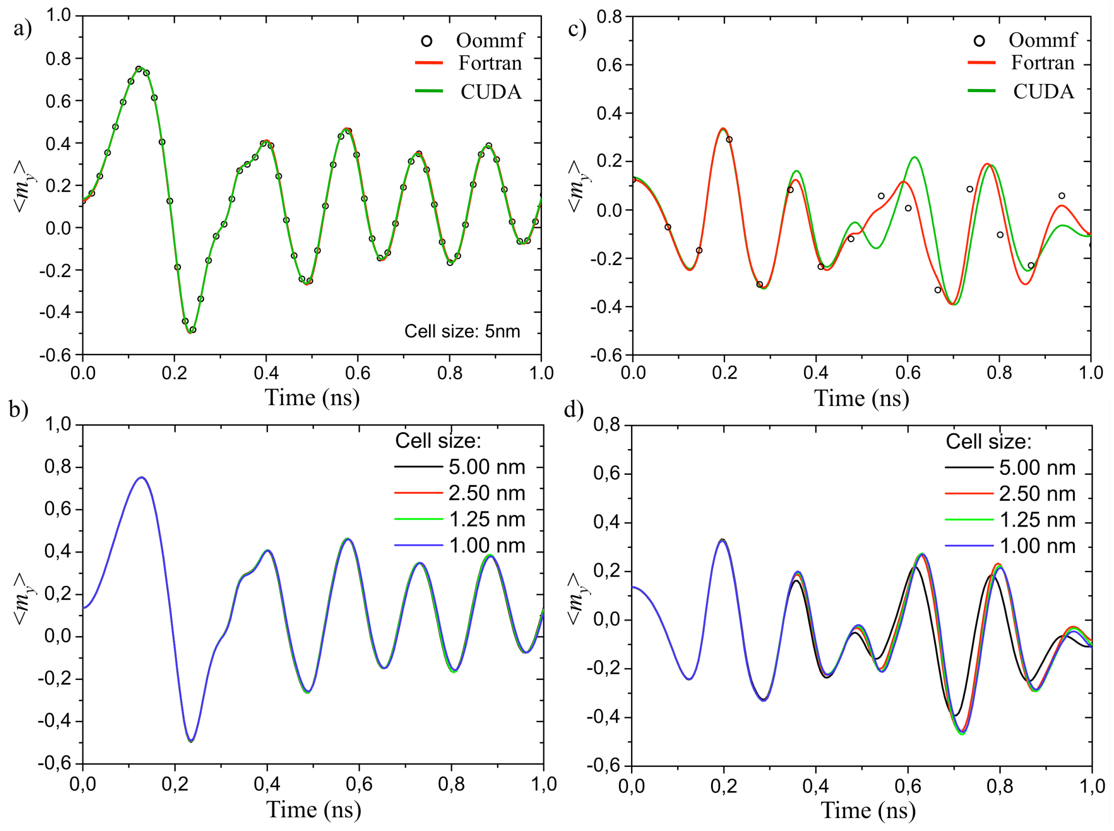

Fig. 32 – Standard problem #4 computed using the developed GPU parallel micromagnetic code: a) Comparing the time evolution of the averaged magnetization along the *y*-axis under the influence of field 1 (25mT, 170º counter-clockwise of +*x*-axis) between Oommf, Fortran and CUDA. b) time evolution of the averaged magnetization along the *y*-axis under the influence of field 1 for four different discretizations. c) Comparing the time evolution of the averaged magnetization along the *y*-axis under the influence of field 2 (36mT, 190º counter-clockwise of +*x*-axis) between Oommf, Fortran and CUDA. d) time evolution of the average magnetization along the *y*-axis under the influence of field 2 for four different discretizations.

Although the solutions reported by the different groups were all obtained using cell sizes smaller than the exchange length (l_{ex}=5.69 nm), the presented results in Fig. 32 show an even smaller discretization in order to also check the robustness of the solution as the cell size goes down. For each field, four discretizations in the *xy* plane Δ_x=Δ_y= 5.0, 2.5, 1.25 and 1 nm were considered, all of them computed using the same time step of δt = 50 fs (in the *z* direction, Δ_z=3 nm was used in all). As can be seen in Fig. 32, for the case of field 1 the solution is virtually independent of the discretization whereas in field 2 some discrepancies appear roughly after 0.35 ns. This divergence of results is expected due to the physical nature of the problem under field 2 and is also seen for other simulations, as reported in the μMAG website [69]. However the solutions clearly converge as the cell size decreases.

Besides concluding that the developed GPU parallel code accurately reproduces the expected results of standard problem #4, of note is the fact that more general simulations with a mesh size of Δ_x=Δ_y=1.0 nm are hardly feasible with a standard CPU sequential code, in particular of longer time windows, whereas with the GPU parallel solver such problems are easily tackled. Since the validity has been checked, the next section is dedicated to the performance of the developed GPU parallel code.

3.7.2 Performance test

Now that the accuracy of the developed parallel code has been verified in the previous section, a performance test is shown by comparing it to the Fortran sequential code in which it was based on. The hardware used for the test was; for the Fortran CPU simulation an Intel Core 2 Quad Q9300 processor (2.5 GHz and 6 MB of L2 cache memory), whereas for the developed GPU parallel code a NVIDIA Tesla 2070 GPU (1.15 GHz and 6GB of RAM memory).

The comparison is made in terms of the time required to perform a single time step when using a second order predictor-corrector method, which at least involves the calculation of all the components of the effective field twice, as was previously discussed in 3.4.1. The results are shown in Fig. 33 for a two-dimensional problem of $N\times N$ cells, where N was systematically increased up to 2000×2000 cells. (Note: in Fortran it was not possible to go much over 500×500 cells, without crashing). From those results it can be asserted that the developed code is considerably faster than the sequential Fortran code from which it was based on, being of up two orders of magnitude faster for meshes with a larger number of cells. Also shown in Fig. 33 is the difference of using or not the optimal size of the sample in the Fourier space when performing the FFT using the CUFFT library, as it was discussed in example 2 of section 3.6.5. By carefully zero-padding the sample in Fourier space it is possible to reduce the simulation time of less favorable geometries, as it is shown in Fig. 33 from the CUDA-1 to the CUDA-2 case.

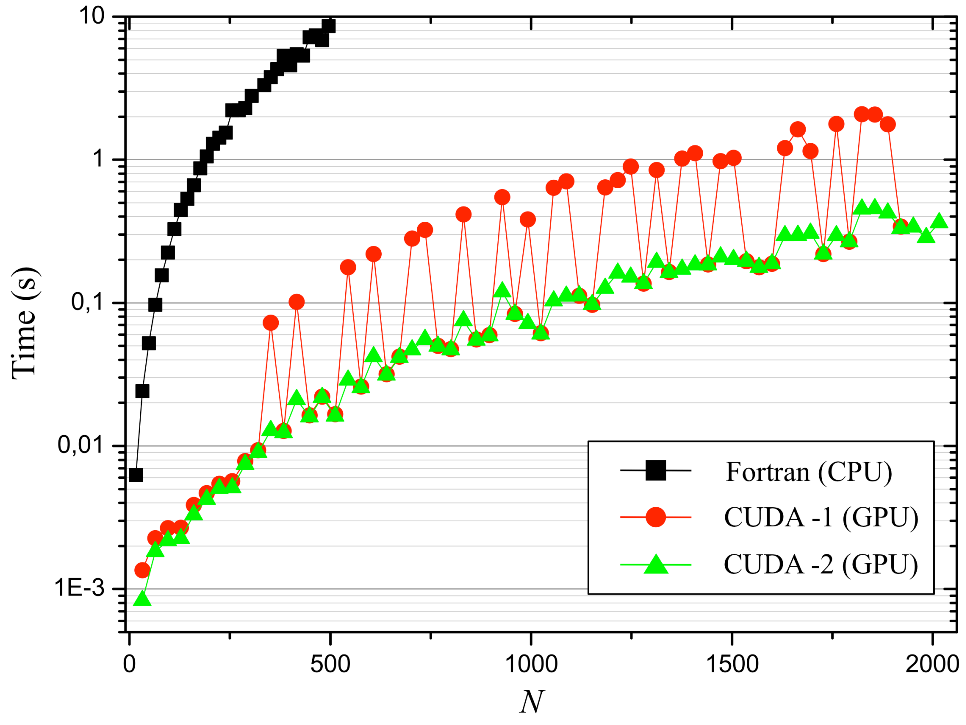

Fig. 33 – Time required performing one time step using the 2nd order predictor-corrector solver algorithm for a two-dimensional problem of *N*×*N* cells. The time it takes for the sequential Fortran code to perform each time step becomes significantly larger than the CUDA GPU parallel code, as the number of cells increases. The difference between CUDA-2 and CUDA-1 comes, respectively, from using or not a zero-padding technique to round up to the optimal dimensions in Fourier space, as discussed in example 2 of section 3.6.5.

The results discussed in section 3.7, among others, were published in the paper titled “*Micromagnetic simulations using Graphic Processing Units*” [56], in which the version of the developed code used was the commercial tailored one developed in conjunction with GoParallel S. L. [70].

4 Different investigations using both sequential-CPU and parallel-GPU micromagnetic codes

This section presents the published works that resulted from the PhD period that this thesis describes. These investigations involve the study of different phenomena on distinct devices such as MTJs, spin-valves and long ferromagnetic stripes, using both sequential-CPU and parallel-GPU codes that were used and developed during the period of this work.

The first part is dedicated to the investigations done using the Fortran sequential code. These were the very first studies done by the author during the initial period of learning and experience gaining on micromagnetics, from both the theoretical and computational points of view. These works put in evidence some of the already discussed limitations (3.5.2) of sequential simulations, which lead to the development of the parallel code. The second part of this chapter is then dedicated to the studies done using the CUDA parallel code, which allowed for a more efficient study of large temporal simulations, as vortex oscillations, and large spatial simulations involving long ferromagnetic strips.

4.1 Studies made using the sequential-CPU micromagnetic code

Both studies shown here are focused on the magnetization dynamics in MTJs. The first investigates the STT induced magnetization switching [71] when using a non-uniform current density distribution (NUCD) based on the fact that the resistance throughout the MTJ is not the same, and compare it to the usual uniform current distribution (UCD) model. The second one studies the influence of thermal activation on STT induced magnetization switching [72].

4.1.1 Magnetization switching driven by spin-transfer-torque in high-TMR magnetic tunnel junctions

Introduction

As it was said in the introduction of this thesis the study of STT devices opens the possibility of different and new device designs, in particular the use of MTJs as ST-MRAMs. With that in mind a micromagnetic study of the magnetization switching in high-TMR (Tunnelling magneto-resistance) MTJs that takes into account the non-uniform current density (NUCD) distribution was performed. This NUCD is implemented by employing a parallel resistance channel model, under the hypothesis that the current flows perpendicularly to the sample plane. In standard studies the current distribution is considered uniform throughout the device. However, this is not a realistic approximation, in particular in high-TMR MTJs. This can be visualized by considering a spatial magnetization configuration that possesses domains in which the magnetization orientation differs by 180º. In the region where the magnetization is parallel (*P*) to the reference magnetization of the pinned layer, the resistance (current density) is smaller (larger) when compared to that of the anti-parallel (*AP*) region.

The voltage dependence of the TMR was not considered in these simulations, which in principle overestimates the NUCD effect. However, comparing the results with numeric computations using the uniform current density (UCD) distribution, gives the possibility to fix a working limit to the model application.

The studied device was a CoFe(8nm)/MgO(0.8nm)/Py(4nm) nanopillar of elliptical 90 nm × 35 nm cross-section. The CoFe is exchange biased and acts as the pinned layer (PL), whereas the Permalloy, represents the free-layer (FL).

Simulation details

Here the particularity of the previously non-discussed variable current density is described and shown how it is taken into account in the dynamic equation (53).

The magnetoresistance R of the MTJ depends on the relative orientation of the normalized magnetization vector of both the FL ($\boldsymbol{m}$) and the PL ($\boldsymbol{p}$) and can be approximated within the macro-spin approach [20] by,

$$R = R_P + \frac{\Delta R}{2}\left[1 - \boldsymbol{m}.\boldsymbol{p}\right] \tag{96}$$

where $\Delta R = R_{AP} - R_P$, with R_{AP} and R_P representing resistances of the AP and P state respectively (note, it is considered throughout this study that the magnetization of the PL $\boldsymbol{p}$**,** is fixed and points along the $+x$ direction). Assuming that the current flows perpendicular to the sample plane $\boldsymbol{j}=j_z(x,y)\boldsymbol{u}_z$ and since it depends on the value of R (96) when making the 2D numerical discretization, the $j_z(x,y)$ is computed for each cell becoming a state-dependent function $j_z(x,y,\boldsymbol{m})$. In other words it is a nonlocal term, since it depends on the spatial configuration of the FL magnetization. The FL is discretized in 2.5×2.5×4 nm^3 cells (area ΔS =6.25 nm^2). Let N be the total number of cells, the total current I_0 is then given by,

$$I_0 = \sum_{i,j}^{N} \Delta S\, j_z(i,j,\boldsymbol{m}) = S\sum_{i,j}^{N} j_z(i,j,\boldsymbol{m}) \tag{97}$$

where S is the total area of the FL, and the indexes (i, j) are the cell coordinates of the 2D discretization of the FL. Considering N parallel channels (Fig. 34), one for each cell, and the macro-spin approximation (96)*,* it is possible to say that the resistance of each channel is then given by $r(i,j)=R_P+(\Delta R/2)[1-\boldsymbol{m}(i,j).\boldsymbol{p}(i,j)]$, plus considering that $J_0=I_0/\mathrm{S}$, the current density distribution for each channel can be computed using simple circuit theory considerations as,

$$j_z(i,j,\boldsymbol{m}) = \frac{J_0}{r(i,j)\sum_{i,j}^{N}\frac{1}{r(i,j)}} \tag{98}$$

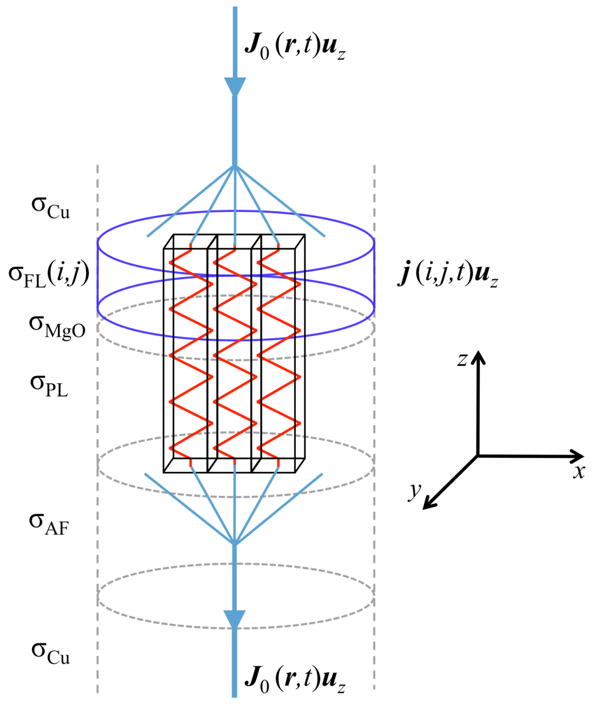
$\boldsymbol{J}_0(\boldsymbol{r},t)\boldsymbol{u}_z$
σ_{Cu}
$\sigma_{FL}(i,j)$
$\boldsymbol{j}(i,j,t)\boldsymbol{u}_z$
σ_{MgO}
σ_{PL}
z
x
y
σ_{AF}
σ_{Cu}
$\boldsymbol{J}_0(\boldsymbol{r},t)\boldsymbol{u}_z$

Fig. 34 – Sketch of the simulated MTJ showing the parallel channel resistance model, as well as the conductivity symbols for each layer.

With expression (98) it is now possible to represent the time dependent NUCD method implemented in the micromagnetic framework, which is no more than replacing $\boldsymbol{j}$ in the dynamic equation (53) by the one expressed in (98).

In the effective field $\boldsymbol{H}_{eff}$, it was taken into account the standard micromagnetic contributions (external, anisotropy, magnetostatic and exchange fields), the magnetostatic coupling between the PL and FL and the Oersted field, which differently from previous studies [73],[75], depends on the magnetization state since

$\boldsymbol{H}_{Oe}[j_z(\boldsymbol{m})]$. In the performed simulations the polarization function considered was the one described by (52), and the other parameters used were the following: external field $\mu_0 H$=50 mT (applied along the positive easy axis direction so as to compensate the magnetostatic coupling with the PL), M_S=6.44×10^5 Am^{-1}; M_{PS}= 1.15×10^6 Am^{-1}; α=0.01; η=0.7 [76], A=1.3×10^{11}Jm^{-1}, R_P=100 Ω; R_{AP}=200 Ω, which are typical experimental parameters [77],[79]. The time step used was of 28 fs.

In order to check the validity of the hypothesis (current perpendicular to the sample plane), a finite element commercial software (MagNet [80]) was used to calculate the spatial current density distribution. This software uses a 3D worksheet that allows for the drawing of the desired three-dimensional MTJ geometry, from which the software automatically generates the finite element mesh. The following conductivities of each layer were introduced in order to perform the computation, (Fig. 34 and Fig. 35; σ_{Cu}=5.77×10^7 Sm^{-1}; σ_{AF}=σ_{PL}=$\sigma_{\mathrm{MgObarrier}}$=1.111×10^3 Sm^{-1}; $\sigma_{\mathrm{CoFe(-16\%)}}$=1.029×10^3 Sm^{-1}; $\sigma_{\mathrm{CoFe(+8\%)}}$=1.322×10^3 Sm^{-1}). The variation to the conductivity of the FL is introduced so as to view the current density behavior, and assert if the tangential component of the current density is of significance, especially in the zone where conductivity changes. The conductivity in the central area was decreased by 16% and increased in the outer regions by 8% of the nominal value. This is done so as to resemble a magnetic domain with higher magneto-resistance in the central region and lower on the outer regions. The spatial distribution of the conductivity can be visualized in Fig. 35, in which the lighter central area represents where the conductivity was decreased by 16%, and the darker lateral regions increased by 8%. After performing the computation, the spatial distribution of the current density was evaluated.

Analyzing the results of the MagNet simulation (Fig. 35), it was observed that the current density $\boldsymbol{j}$ mainly lays in the normal direction (z), representing in the worst case 95% of the total $\boldsymbol{j}$. Looking at the variation of the normal component of $\boldsymbol{j}$ (Fig. 35 (a)), there are two distinct zones in the FL, each corresponding to the different values of the conductivity, where the most significant variations of j normal are observed at the border of these two regions. It is possible to crosscheck this result with the tangential component of $\boldsymbol{j}$ shown in Fig. 35 (b), which yields a gradient of current density towards the border of the conductivity variation (equivalent to a domain wall). The tangential component of $\boldsymbol{j}$ is always less than 2% of J normal, which confirms the hypothesis made for the micromangetic simulations using the parallel resistance model (Fig. 34) that ***j*** is mainly perpendicular to the sample plane, as can also be seen in Fig. 35 (c).

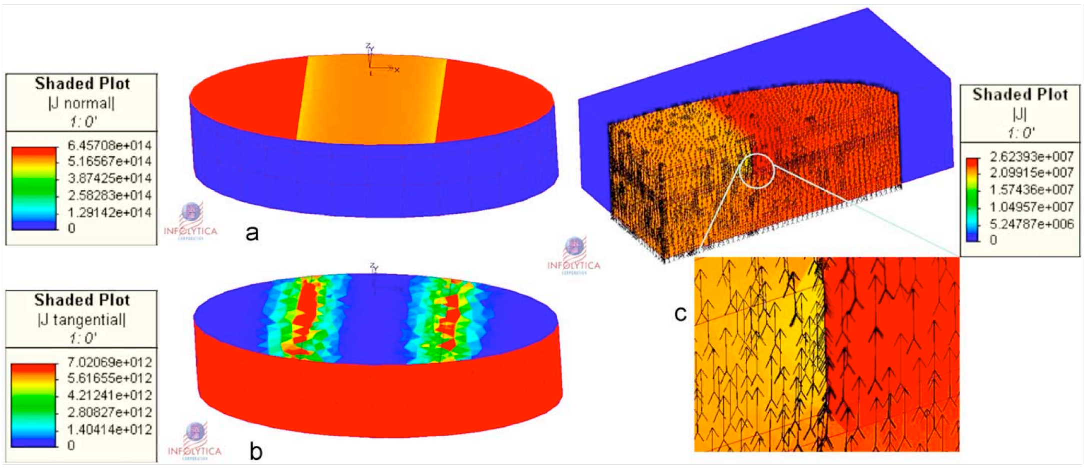

Fig. 35 – Results of the current density ***j*** computed using MagNet to check which percentage of ***j*** flows perpendicularly to the sample plane within the FL, (color gradient: ***j*** in Am^{-2}). a) Spatial distribution of the normal component of ***j***. b) Spatial distribution of the tangential component of ***j***. c) Horizontal cross-section of the FL in which the arrows represent the vector ***j*** with zoom in. In the worst case the normal component of ***j*** accounts for 95% of the total current density.

Results and discussion

Earlier micromagnetic studies of some aspects of the magnetization switching [81] and persistent dynamics [79],[82] in MTJs have been performed using an UCD distribution model. Here the NUCD model is introduced and its results compared to the previous one. Considering the resistance in each state as R_{AP}=200 Ω and R_P=100 Ω for the NUCD model (simulations performed considering R_{AP} =250 Ω gave qualitatively the same results), the switching dynamics were studied in two different regimes for both transitions (P-AP and AP-P). The first performed by applying current pulses through the sample, whereas the second was done with an increasing ramp-like current.

i) Magnetization switching with a current pulse: modal analysis and phase diagrams

This section presents a detailed study of the magnetization switching when applying current pulses in MTJs for both current distribution models. This kind of study is important from both technological (writing mechanism in MRAM) and fundamental points of view, since it provides information about the stability of intermediate states and the way in which energy is pumped into the system.

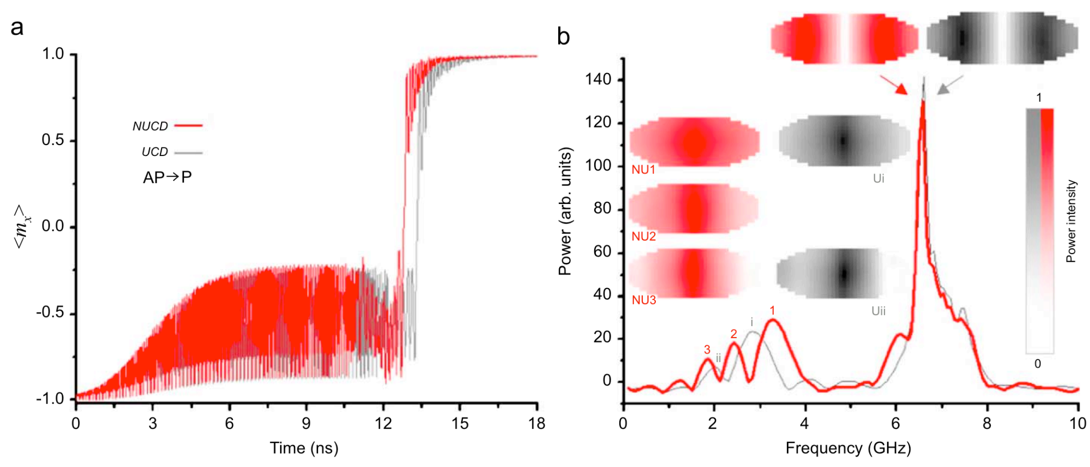

Fig. 36 – AP to P transition. a) Normalized '*x*' component of the magnetization vs. time for the UCD (gray) and NUCD model (red). The height of the current density pulse applied was of 4.5×10^6 Acm^{-2}, in a 14 ns pulse duration with rising and descending times of 100 ps. b) Frequency spectrum for the UCD (gray thin line) and NUCD model (red thick line) of the pre-switching oscillations from 0 to 12.8 ns. Insets: 2D power density plots produced by each computational cell at the FL with the corresponding frequency mode indicated for each model, (darker means larger power).

An example of a magnetization switch is shown in Fig. 36 (a), where the *x*-component of the normalized averaged magnetization ($<m_x>$) is displayed. A pulse of amplitude $J_0=4.5\times10^6$ Acm^{-2} is applied during 14 ns with rising and descending times of 100 ps. Comparing the results of both models, it is observed that the NUCD promotes the AP-P transition and the switching process begins by means of preliminary oscillations, on both models. In order to study how these pre-switching oscillations might influence the magnetization spatial distribution, a detailed analysis within the frequency domain was performed by applying a micromagnetic spectral mapping technique (MSMT) [83],[85]. The technique consists on first individually calculating the Fourier transform for each cell within the computational mesh and then extracting the corresponding frequency. This assures that the off-phase oscillations in different parts of the sample do not cancel, which would be the case when calculating the transform only for the average magnetization.

Using the MSMT technique, applied from the initial instant up to 12.8 ns, allows for the computation of the excited pre-switching oscillation modes. Looking at the frequency domain spectra shown in Fig. 36 (b), the main mode present ($f = 6.5$ GHz) is basically the same for both the UCD and NUCD models. However, there are differences in the lower frequency modes, *i.e.* more modes and with larger power are exhibited in the NUCD case. To gain a better understanding of these modes, 2D density plots (spatial distribution) were also computed that show the power intensity of the excited modes [83],[84] (insets of Fig. 36 (b) and Fig. 37 (b)). This determines which parts of the sample oscillate for a given mode. Fig. 36 (b) shows two types of pre-switching oscillation modes during the AP-P transition. The main mode is localized at the lateral sides of the sample, which was defined as an ‘‘edge’’ mode, while the low power modes in the central area of the sample, were defined as ‘‘central’’ modes. Comparing the spatial distribution of the oscillation modes (insets of Fig. 36 (b)), one concludes that both models describe the switching similarly, with the difference that the NUCD model presents one more central mode. The faster transition in the NUCD model might emerge due to both the presence of this extra central mode and the fact that these secondary modes are in general more intense. In this case the increase in power of the central modes could be explained as follows: when the magnetization in the central region begins to oscillate, the resistance starts to decrease leading to a current density increase in that area, which augments the spin-torque and promotes the oscillations. On the other hand, the oscillations begin at the edges (see main mode at 6.5GHz in Fig. 36 (b), and inset (v) of Fig. 39), but before the switching takes place the symmetry has to be broken, implying that the central region has to be destabilized from its static configuration (along the easy axis). In the NUCD case this process happens faster (for this AP to P transition) because the current is initially localized at the edges of the FL, thus promoting the oscillations.

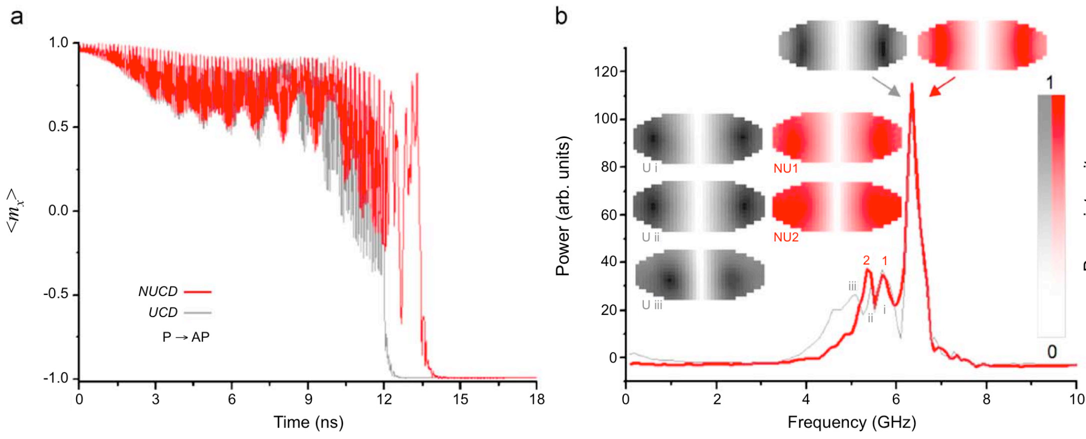

Fig. 37 – P to AP transition: a) Normalized ‘x’ component of the magnetization vs. time for the UCD (gray) and NUCD model (red). The height of the current density pulse applied was of 1.05×10^{7} Acm^{-2}, in a 14 ns pulse duration with rising and descending times of 100 ps. b) Frequency spectrum for the UCD (gray thin line) and NUCD model (red thick line) of the pre-switching oscillations from 0 to 12.0 ns. Insets: 2D power density plots produced by each computational cell at the FL with the corresponding frequency mode indicated for each model, (darker means larger power).

The P to AP switching was also analyzed with the MSMT (Fig. 37 with J_0=1.05×10^{7} Acm^{-1}), whose results show a main pre-switching mode (6.5GHz) equal to the AP to P transition and different edge modes around 5.5 and 5.8 GHz (NUCD).

Comparing the models with respect to this transition, the switching process was found to be practically the same, having both similar pre-switching oscillation modes. However now it is the UCD distribution that presents an extra oscillation mode and switches faster. This effect is due to the distribution of the current density in the NUCD model that for this transition gives rise to less oscillation modes. As in the previous transition the oscillations start at the edges but in this case it makes the resistance higher in those regions, thus initially localizing the current in the center of the FL where the oscillations are more strongly damped, retarding the switching.

In general, it seems that the switching is triggered earlier when more low frequency oscillation modes are excited (since the main high frequency mode remains the same for both models). The number of modes appearing in the NUCD model can be directly linked to the way in which the current is distributed throughout the FL.

To extend the analysis of pulsed current assisted switching, a systematic study was performed in order to compile comparative phase diagrams (Fig. 38) as function of the height (amplitude of ***j***) and duration (time interval) of the current density pulse. The current density was increased linearly from zero up to its maximum value in 100 ps, and resolution of 3×10^{5} Acm^{-2} was considered for the current density between different simulation points in all phase diagrams (Fig. 38).

As a general trend it is observed (Fig. 38): for the NUCD model the boundary between switched and non-switched states is smoother, without much of the ‘‘teeth’’ behavior seen in the UCD model. However, a fully smooth frontier was not obtained. For example in the P to AP transition with the external field perfectly aligned along the easy axis and for current density pulses of 1.14×10^{7} Acm^{-2}, a ‘‘teeth’’ region is present. A detailed analysis of the switching in that region shows that the intermediate states of the switching are constantly changing, including cases of vortex and anti-vortex configurations [86]. It is due to these configurations (influenced by intrinsically non-uniform effective fields like the Oersted field) that the system does not switch for certain current pulses, even though it is achieved for smaller current densities (black ‘‘teeth’’ in Fig. 38).

Comparing both models according to Fig.38, it is possible to say that the NUCD distribution gives rise to less ‘‘teeth’’ structures. As in the particular cases of Fig. 36 and Fig. 37, these diagrams also show that the NUCD favors the AP to P and hinders the P to AP transitions. In particular for the P to AP transition the current density needed for switching in the NUCD model is significantly larger than in the UCD model. Therefore the NUCD model is more asymmetric and its current distribution contributes to the intrinsic asymmetry of the switching observed in real devices.

Former works in this kind of nanostructures, both experimental [87],[88] and theoretical [81], demonstrated that a small misalignment of the external field in respect to the easy axis, gives rise to more uniform magnetization dynamics and lower critical currents. To evaluate this phase diagrams where the external field was tilted by 3º were also computed. As expected, Fig. 38 shows how the critical current densities decrease and the frontiers become smoother as the result of a more uniform behavior of the magnetization dynamics. This is due to the hard axis component of the external field whose torque helps in pulling the average magnetization away from the easy axis equilibrium position.

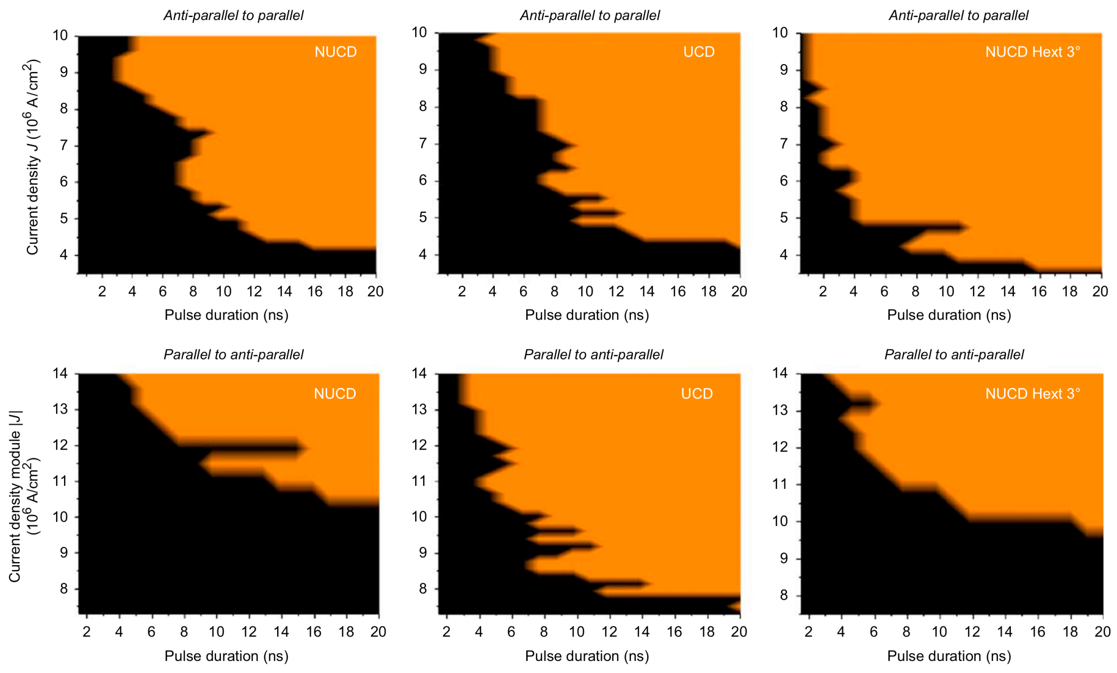

Fig. 38 – Phase diagrams of the current density pulsed excitation switching for both the AP-P (top) and P-AP (bottom) transitions. From left to right, NUCD and UCD model with a 50 mT external field applied along the easy axis, with the last NUCD diagram where the external field applied has a 3º misalignment in respect to the easy axis. Color area means that the system has switched and black the opposite.

Performing these phase diagrams is an extremely laborious task since it involves simulating a large number of different case parameters. This is an aspect, which is greatly improved using the developed parallel code when performing similar diagram analysis, due to the significant simulation speed increase discussed in the previous chapter.

ii) Ramped current hysteresis loops

This section describes the effects of applying a linearly increasing current through the nano-pillar until the switching critical value.

Depicted in Fig. 39 are the ramped current density magneto-resistance hysteresis loops computed using both models. Looking at the hysteresis loops, one sees that the NUCD model favors the switching going from the AP to P state, and on the other hand it hinders the reverse transition P to AP, in respect to the UCD model. These results are in agreement with the discussion of the previous section, but here presented as a function of the critical current density.

Analyzing both switching processes (for a sweep rate (SR) of 10^{13} A/(cm^{2}s)), again it is seen that they are preceded by initial oscillations of the FL magnetization. In insets (ii) for the P-AP transition and (v) for the AP-P case, it is shown that the non-uniform oscillations start at the lateral regions (Fig. 39). As the amplitude of these oscillations increase, they break the magnetization symmetry and lead to several complex states (insets (iii) and (vi)) before the switching is achieved.

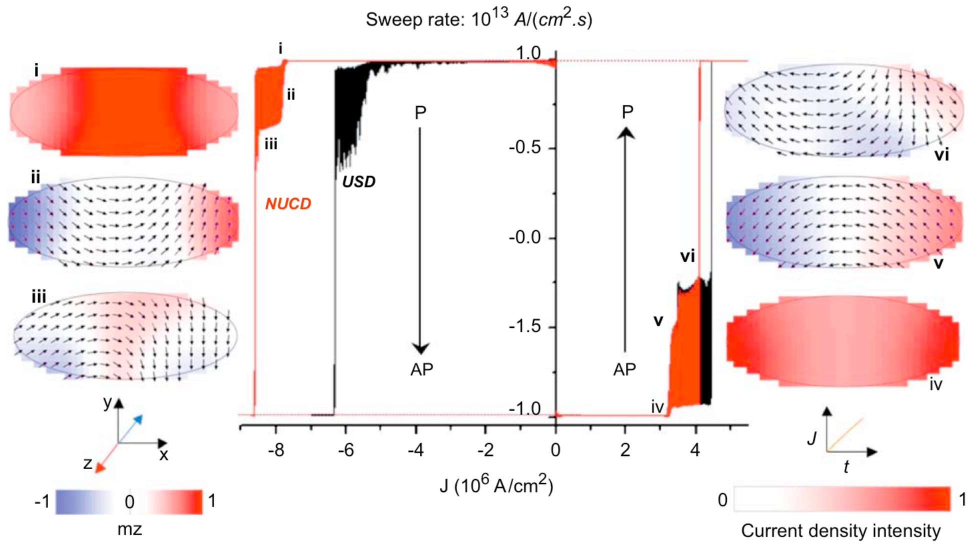

Fig. 39 – Magnetization vs. critical current density hysteresis loops. In red the results using the NUCD and in black using UCD for a sweep rate 10^{13} A/(cm^{2}s). Insets (i) and (iv) represent the spatial distribution of the current density (see the coordinate in the graph). Insets (ii) and (v) show a magnetization snapshot at the beginning of the magnetization switching, whereas (iii) and (vi) show the magnetization snapshot just before the switching is achieved.

Focusing now on the introduction of the NUCD model, it is again analyzed how it leads to opposite effects depending on the examined transition. In the AP to P transition, the oscillations at the lateral regions produce a decrease of the resistance in those regions, thus the application of the NUCD model results in a higher current injection in said lateral areas (inset (iv) of Fig. 39), promoting larger oscillations and triggering the switching. On the other hand, for the P-AP reversal the oscillations in the outer regions imply an increase of the resistance and thus less current is injected in those regions (inset (i) Fig. 39). The oscillations are damped and in order to achieve the switching more current is needed. Of course this is just a simplified description of the process and indeed the magnetization dynamics is rather non-linear. Plotting the critical current densities as a function of the SR (Fig. 40), it is seen that these curves are non-monotonic and as reported in other numerical studies [73],[75], the trend is related to a trade-off between the effective field contributions (in particular the Oersted field) and the STT. Although the observed behavior is not linear, the general trends of applying the NUCD model follow the aforementioned discussion (Fig. 39).

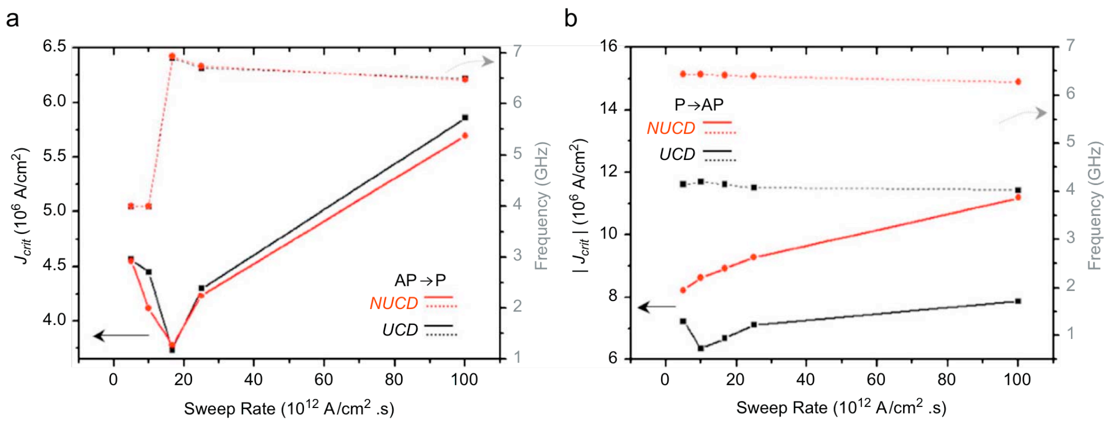

Fig. 40 – Both AP to P, a), and P to AP, b), reversals comparing the critical current density between the NUCD (red line) and the UCD (black line) models in function of the sweep rate. On the right scale of each graph, it is represented the main pre-switching oscillation frequency mode for both models and transitions at each sweep rate tested, (doted line).

Looking at Fig. 40 it also shows the frequency of the main (larger power) pre-switching oscillation mode as a function of the SR, for each transition and model. Again depending on the transition analyzed, different results are observed between the models. In the AP to P transition (Fig. 40 (a)), the critical current density and frequency of the main mode as a function of the SR are more or less independent of the spatial current density distribution. On the other hand, the critical current density and frequency of the main mode in the P to AP transition are clearly affected by the NUCD model. In order to better understand the physical mechanisms that give rise to this behavior, the MSMT (Fig. 41) was performed, similarly to the ones shown in Fig. 36 and Fig. 37. Nevertheless, it has to be stressed that a direct comparison with the pulsed current case is not possible due to the different type of excitation. The current is applied linearly (with a slope equal to SR) implying that when the switching is achieved the spin-torque has been acting and pumping energy for a much longer time.

For example, in the AP to P case for the slowest SR (Fig. 41 (a), SR=5×10^{12} A/cm^{2}s) when the switching is achieved the current has been pumping energy for 610 ns. For the same transition in the current pulsed case analyzed in Fig. 36, the current was applied only for 13 ns. Again this gives an idea on how long the sequential simulations might take in real time and how the developed parallel code benefited this aspect, since simulations of hundreds of nanoseconds take weeks to solve, even in a relatively small sample like the one studied here. In the following paragraphs the discussion focuses on the modes excited in both transitions.

As it was previously stated, in the AP to P transition (Fig. 40 (a)) the critical current density and frequency of the main mode in function of the SR are practically independent of the current density spatial distribution. This occurs because the excited modes are the same for both the UCD and NUCD models, giving rise to practically identical nucleation processes of the switching (Fig. 41 (a) and (b)).

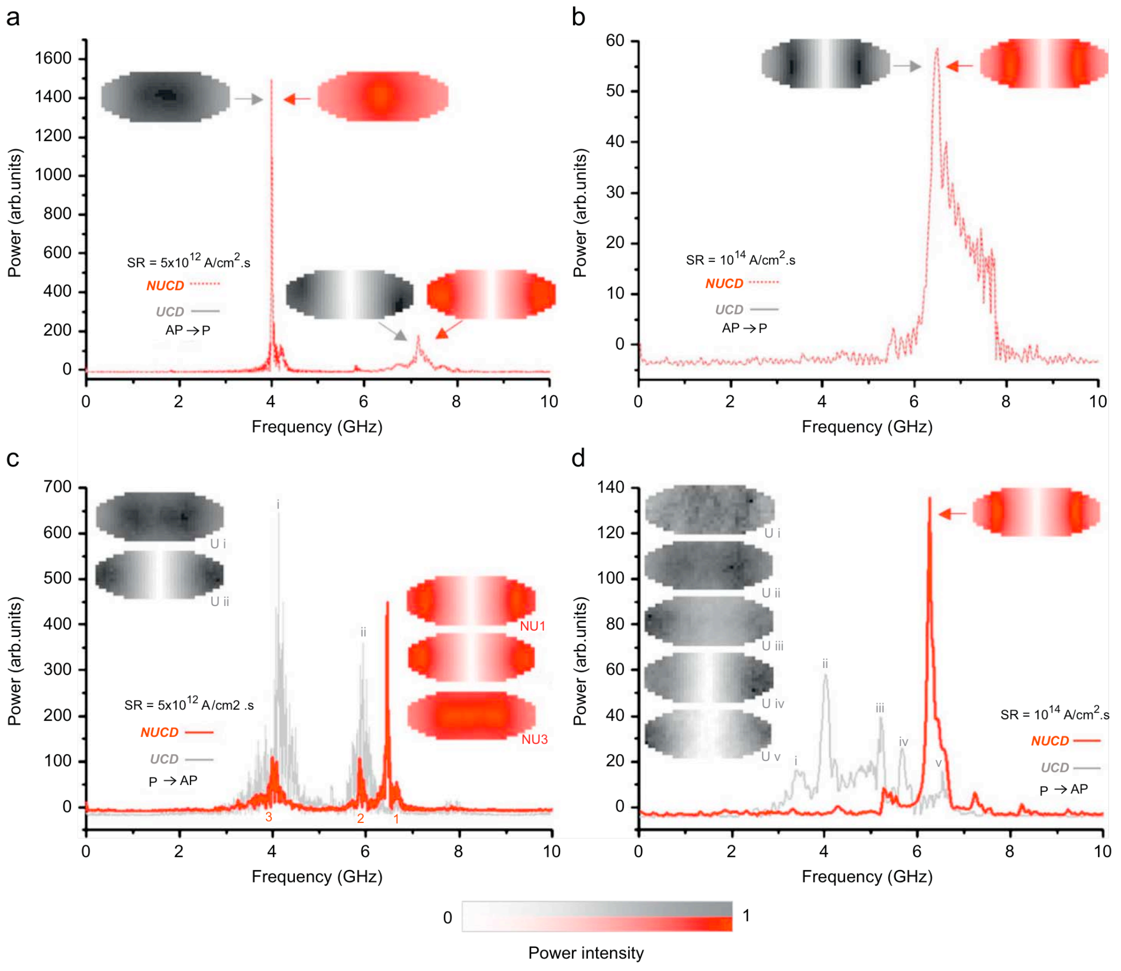

Fig. 41 – Frequency spectrum for both models and transitions in respect to the lowest and highest sweep rates (SR) of Fig. 40: a) and b) pre-switching oscillation modes of both AP to P transitions, show that the modes generated are equal between models and that for lower sweep rates (lower than the one of minimum critical current density of Fig. 40 (a)) the main mode turns into a central one (see

insets). In c) and d) pre-switching oscillation modes of both P to AP transitions, show that for the UCD model more oscillation modes are generated being the main one central, while for the NUCD the main mode is an edge one, (cause of the difference in frequency seen in Fig. 40 (b)).

For the slowest SR analyzed (Fig. 41 (a) SR=5×10^{12} A/cm^2 s) the main mode is a central lower frequency one, in accord with the slower pumping of energy. On the other hand, for the fastest SR analyzed (Fig. 41 (b) SR=10^{14} A/cm^2 s) the edge mode is the principal one, corresponding to the faster way of pumping energy. In other words, the difference in the excitation velocity (slope of the current density ramp, *i.e.* SR) leads to a different formation of oscillation modes within the sample. Between the minimum and maximum SR an intermediate behavior is found, with a minimum in the critical current density where both edge and central modes have similar importance. The jump in frequency seen in Fig. 40 (a) after the minimum in critical current density $J_{\min}$ (SR=1.67×10^{13}A/cm^{2}s) reveals a transition from the edge predominant to the central predominant mode dynamics. Also observed is that this mechanism is related to a trade-off between the Oersted field and the spin-torque, given that when the Oersted field effect is removed from the simulations the minimum is not present. This happens because in the absence of Oersted field the predominant mode is always a central one around 4 GHz (similar to the one of Fig. 41 (a)). This is understandable since the non-uniform Oersted field, more intense near the boundaries of the sample, clearly promotes the edge modes.

Finally it is worth to comment that for the studied AP to P transitions, even though the NUCD model slightly diminishes both the switching time (seen in the pulsed cases of Fig. 36 and Fig. 37) and critical current density (seen in the ramped current loops of Fig. 39 to Fig. 41), the effect is in both cases very weak. Consequently it can be concluded that the NUCD model weakly affects the AP to P transition.

Analyzing now the P to AP reversal (Fig. 40 (b)), it can be seen that the frequency of the main mode arising from both the UCD and NUCD models is clearly different, but pratically independent of the SR. This can be understood by looking at the modal analysis of Fig. 41 (c) and (d), where for the UCD model the predominant mode is a central one for all the SRs, whereas for the NUCD model the predominant mode in the dynamics is an edge one. Again, as expected due to the way in which the energy is pumped into the system, the central modes are more intense for slower SRs, while the edge modes gain intensity for faster SRs. Nevertheless, the predominance of one or the other does not change in the simulated SR range. It was also observed that in this case the dynamics is not affected at all by the suppression of the Oersted field, pointing to a more spin-torque dependent excitation of modes.

Comparing both models in terms of critical current densities for these transitions (P to AP), it is seen that a higher current density is needed to switch the system in the NUCD model, which is in agreement with discussions of the previous section of results and Fig. 39.

Summing up the results of both models and transitions, it was seen that the P to AP transition is more affected by the use of the NUCD model. Regarding the NUCD model, bringing together the simple discussion of Fig. 39 and all the modal analysis performed, it is concluded that after the initial oscillations in the edges (due mainly to the magnetostatic coupling with the fixed layer) the concentration of the current in the central area does not significantly promote the central modes (P to AP case). Its main effect seems to be a frequency reduction of the edge mode with respect to the AP to P transition (compare Fig. 36 with Fig. 37 or Fig. 41 (a) and (b) with Fig. 41 (c) and (d)).

Since the effect of the NUCD model is clearly different for AP to P and P to AP transitions, in real devices the effect of the NUCD distribution could therefore be taken as an additional source for the intrinsic asymmetry in the switching process.

Conclusion

In summary, it was performed a micromagnetic study of the magnetization switching driven by spin-polarized current in a high-TMR MTJs, studying the effects of uniform and non-uniform current density distributions by using current pulse excitations and an increasing ramp-like current density up to the transition point. The results show that the NUCD distribution is a source of asymmetry between both transitions in terms of critical current density, verified in both pulsed and ramped excitations. This asymmetry is highlighted in the numerical experiments using an increasing current density ramped excitation, which showed that the AP to P switching is only marginally affected by the current density distribution while its effect on the P to AP transition is more significant in terms of critical current density and generated oscillation modes.

In the analysis within the frequency domain for current density pulsed excitations, it was seen that the pre-switching oscillations are characterized by ‘‘edge’’ and ‘‘central’’ modes, coming the transition faster when more oscillation modes are generated. For ramped current hysteresis loops, the modal analysis also shows effects on the critical current density that can be explained based on the predominance of these central or edge modes within the dynamics. In this respect less current is needed when the predominant mode is a central one. These types of modes have been experimentally detected [89].

Making an extensive study using both models in the pulsed excitation regime, the NUCD distribution presents smoother transitions in a current density vs. pulse duration diagram (Fig. 38), showing as well the aforementioned higher asymmetry between transitions and that a small misalignment of the external field with respect to the easy axis reduces the critical current density. From that phase diagrams (Fig. 38) and the slow SR ramped current density simulations a limitation of the sequential micromagnetic code becomes evident, since such graphs and simulations take a considerable amount of time to complete. These types of lengthily studies are greatly beneficiated by the use of the developed parallel GPU micromagnetic code.

4.1.2 Thermal effects on spin-transfer-driven switching in high-tunneling-magnetoresistance magnetic tunnel junctions

Introduction

As the work described in 4.1.1, here it is also discussed the STT dynamics in MTJ devices, however in this case the role of the thermal field to the switching process is investigated. A particular point of the switching process is also investigated and it refers to the single shot time domain measurements [77],[90],[91] in MTJs of either rectangular [77] 300×100 nm^2 or elliptical [90] cross-section 170×60 nm^2, which show the presence of an "incubation delay" or a "nonreactive time" during the reversal process of the free layer. It has been demonstrated theoretically that this incubation delay arises from the presence of the field-like STT term [92]. Moreover, highly sensitive measurements of a CoFe(0.5 nm)/CoFeB(3.4 nm) [91] free layer MTJ device of elliptical cross-section 130×65 nm^2, indicate the existence of a temporal coherence of pre-switching oscillations.

In order to add some insight into the mechanisms governing the switching processes, the thermal field influence on the magnetization switching dynamics in high-TMR MTJs was investigated by means of micromagnetic simulations.

Simulation details

The simulated MTJ is of 90×35 nm^2 elliptical cross-section with the following structure, CoFe(8 nm)/MgO(0.8 nm)/Py(4 nm). The CoFe is the exchange biased pinned layer and the Py corresponds to the free layer. A Cartesian coordinate system was chosen with the *x* direction along the easy axis. The micromagnetic dynamic equation (53) was considered with $\boldsymbol{j}$ as a function of the magnetization as described in the previous study 4.1.1 (NUCD model). Another modification to equation (53) was considered by adding the dimensionless parameter ξ, so as to study the influence of the magnitude of the perpendicular spin-torque term (PSTT), which according to the works [76],[95] and [96], is larger than the one that naturally arises from (53). Thus the dynamic equation (53) is written as,

$$\begin{aligned}\left(1+\alpha^2\right)\frac{d\boldsymbol{m}}{dt} &= -\gamma_0\left[\boldsymbol{m}\times\boldsymbol{H}_{eff}+\alpha\boldsymbol{m}\times\left(\boldsymbol{m}\times\boldsymbol{H}_{eff}\right)\right] \\ &\quad -\frac{g\mu_B\boldsymbol{j}\mathcal{P}(\boldsymbol{m}.\boldsymbol{p})}{2M_S d|q_e|}\left[\boldsymbol{m}\times\left(\boldsymbol{m}\times\boldsymbol{p}\right)-\xi\alpha\left(\boldsymbol{m}\times\boldsymbol{p}\right)\right]\end{aligned} \tag{99}$$

The effective field accounts for all standard micromagnetic contributions: external, anisotropy, magnetostatic, exchange, and Oersted fields. Additionally, thermal fluctuations are incorporated as a stochastic field with Gaussian distribution and zero-mean statistical properties, as discussed in section 2.3.6. The polarization $\boldsymbol{p}$ is fixed along the $+x$ direction and the other simulation parameters read as follows: $\mu_0 H$=50 mT applied along the easy axis direction so as to compensate for the magnetostatic coupling with the pinned layer; M_S=6.44×10^5 Am^{-1}; M_{SP}=1.15×10^6 Am^{-1} pinned layer saturation magnetization; α=0,01 damping parameter; η=0.7 polarization factor; A=1.3×10^{-11} Jm^{-1} exchange constant. The resistance in the parallel P state is considered to be of 100 Ω and of 200 Ω for the anti-parallel AP state. Only the dynamics of the free layer, discretized into a 2.5×2.5×4 nm^3 mesh, is resolved using in the simulations a time step of 28 fs. Critical current densities at T=0 K were found to be of Jc=3.0×10^6 Acm^{-2} and Jc=-6.1×10^6 Acm^{-2}, for the AP to P and P to AP transitions, respectively. The sample temperature T, was calculated using the following expression [93],

$$T = \sqrt{T_{bath}^2 + \varepsilon I^2} \tag{100}$$

where T_{bath} is the bath temperature in which the sample is immersed, I the applied current, and ε is a parameter that depends on specific material factors and sample geometry. It includes the Joule heating generated by the current.

Results and discussion

With the goal of studying the effects triggered by the thermal fluctuations, the switching process was first characterized in the spin-torque switching regime, [94] with the stochastic study of 1200 simulations for each transition (Fig. 42). Again this type of study shows the limitation of sequential simulations, since in order to perform such quantity of simulations a few months were needed using several CPUs, even though the discretized sample was small (504 cells).

In Fig. 42 it is shown the switching time distribution for the AP to P and P to AP transitions when respectively applying a current density of J=4.5×10^6 Acm^{-2} and J=−1.05×10^7 Acm^{-2}. Although the same sample temperature would be larger for the P to AP transition due to the larger applied current, the same temperature T=411 K was considered for both transitions so as to have the same thermal field interaction in both transitions.

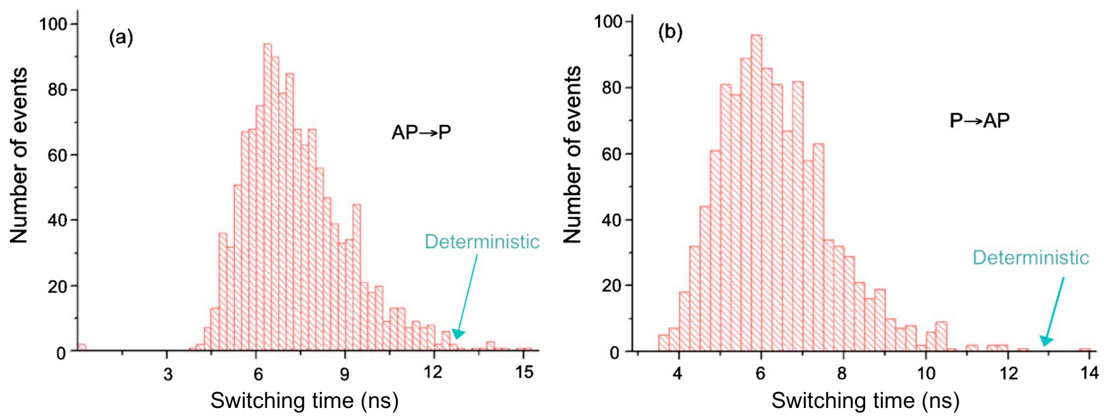

Fig. 42 – Stochastic simulations at T_S=411 K. Switching time distribution for a set of 1200 simulations using a current pulse of: t_{pulse}=14 ns and J=4.5×10^6 Acm^{-2} for the AP to P transition (a) and of t_{pulse}=14 ns and J=-1.05×10^7 Acm^{-2} for the P to AP transition (b).

The Gaussian-like distribution of switching times shown in Fig. 42, indicate that the most probable switching time is roughly 50% lower than at T= 0 K (deterministic 12.3 ns and 12.5 ns for the AP to P and P to AP, respectively). The observation that thermal fluctuations assist the switching process is a well-known phenomenon, however, the exact mechanisms governing it are still not clear. Therefore, in order to gain some insight into the switching dynamics, it was first calculated the magnetization average of the *x*-component displayed in Fig. 43 (a) and Fig. 43 (b), for the AP to P and P to AP transitions, respectively. This averaging procedure only accounts for the results of the simulations with the most probable switching time shown in Fig. 42, with the maximal error between switching times of 25 ps. For these averages, due to the difference in oscillation phase, the pre- and post-switching dynamics are practically absent. Hence, it seems that the thermal field does not introduce any relevant dynamics and the main driving torque is due to the STT.

However, coherent pre- and post-switching oscillations Fig. 43 (c) and Fig. 43 (d) do appear when each simulation is first shifted to the same switching point and then averaged. The resulting averages were obtained from 1100 simulations for each transition. These results, resembling the experimental ones reported in [77] and [91], indicates that a small difference in the pre- and post-switching oscillation phase during the average procedure can hinder the oscillation dynamics. As it will be discussed later, these results are quantitatively independent of the perpendicular torque.

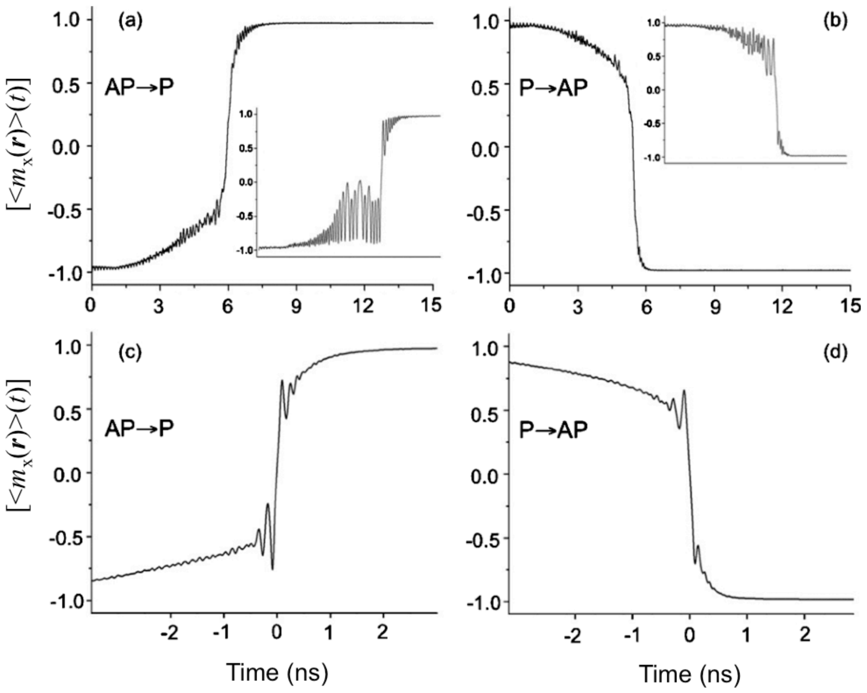

Fig. 43 – Average magnetization of the *x*-component at 411 K, showing the difference between averaging close events in the time frame, (a) and (b), and averaging all events via numerically transposing each simulation to the same switching point, (c) and (d). Average of 24 events for the AP to P transition with switching times of 6.725 ns ± 25 ps, with a single shot transition inset; t_{pulse}=14 ns and J=4.5×10^6 Acm^{-2} (a). Average of 23 events for the P→AP transition with switching time of 5.825 ns ± 25 ps, with a single shot transition inset; t_{pulse}=14 ns and J=−1.05×10^7 Acm^{-2} (b). Average magnetization extracted from 1100 simulations of equal switching time, whereby numerically translating the points of each simulation to the same switching point, (c) and (d).

Unlike the macrospin simulations performed in [92], micromagnetically no incubation delay (absence of pre-switching oscillations, see Fig. 43 (a) and Fig. 43 (b)) is observed, since the pre- and post-switching oscillations originate from the excitation of non-uniform modes, which are beyond the macrospin approximation and are discussed in detail below.

The analysis of the oscillations (Fig. 43) in the frequency domain is carried out by the previously mentioned MSMT. [83],[85]. This spectral analysis reveals the existence of central and edge [71],[89] oscillation modes of similar power intensity Fig. 44. The interpretation can be twofold; either the thermal field promotes both modes in the same manner and the qualitative difference arises from the STT contribution, or the current density is still too high and dominates over the thermal effect. To determine the real origin of these modes, simulations using current densities below the critical current density obtained at T=0K for both transitions, were performed ($J_{c\text{AP}\rightarrow\text{P}}$=3.0×10^6 Acm^{-2} and $J_{c\text{P}\rightarrow\text{AP}}$=-6.1×10^6 Acm^{-2}). This way the switching is expected to be more temperature rather than STT dependent, and therefore, a division between thermally activated (|*J*|<|*Jc*|) and STT-driven switching (|*J*|>|*Jc*|) can be set clearly for the simulated device.

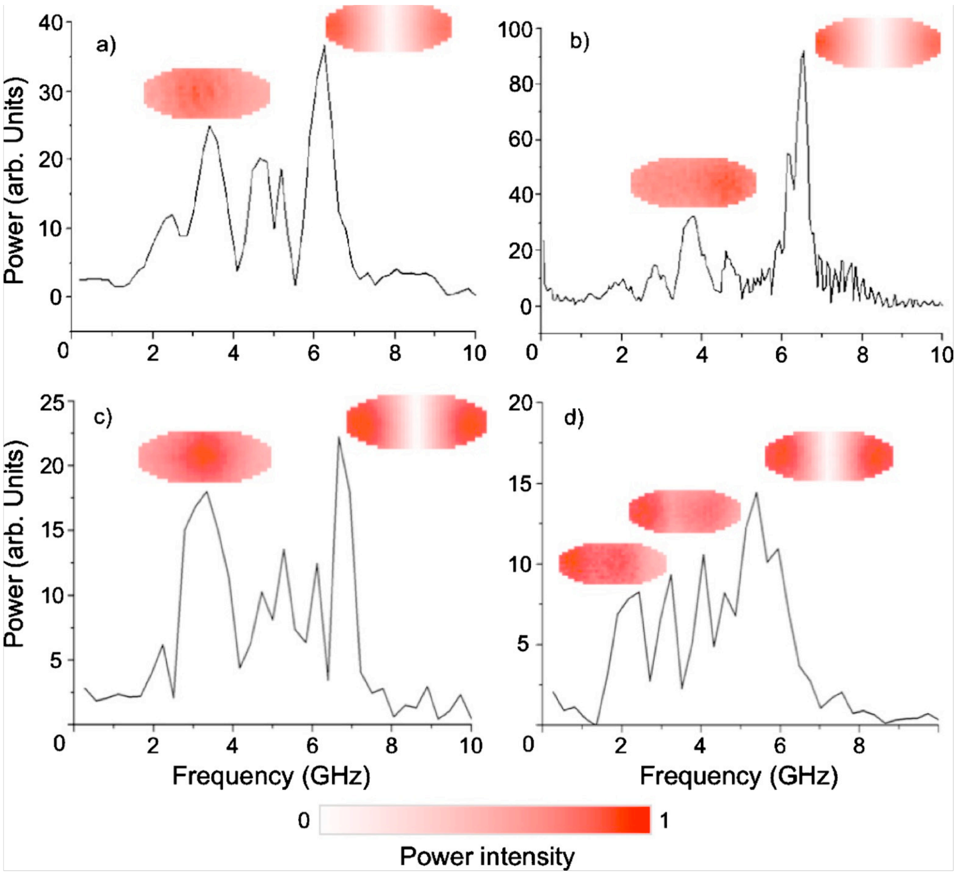

Fig. 44 – Frequency analysis of the pre-switching oscillations (from Fig. 43) exhibiting both central and edge modes as seen in the insets. It suggests that either the thermal fluctuations excite similar modes for both transitions or the current density is still too high so that the STT effect is dominant. Oscillation modes present in the transition of Fig. 43 (a) a); Fig. 43 (b) b); Fig. 43 (c) c); Fig. 43 (d) d).

The results of the reversal study using current densities below the 0 K critical ones, are plotted in Fig. 45. A transpose average of 100 distinct simulations showing pre- and post-switching oscillations is presented for each transition. The results of using the MSMT for these average time traces identify a central mode with a frequency of roughly 4.3 GHz to be the dominant one for both transitions. This leads to the conclusion that the thermal fluctuations assist the switching processes mainly by promoting oscillations at the center of the sample. Since the magnetization starts oscillating at the edges, [71] and then the oscillations slowly propagate to the entire sample until the switching is achieved, the role of the thermal fluctuations is to accelerate this propagation by destabilizing the magnetization in the central area sooner.

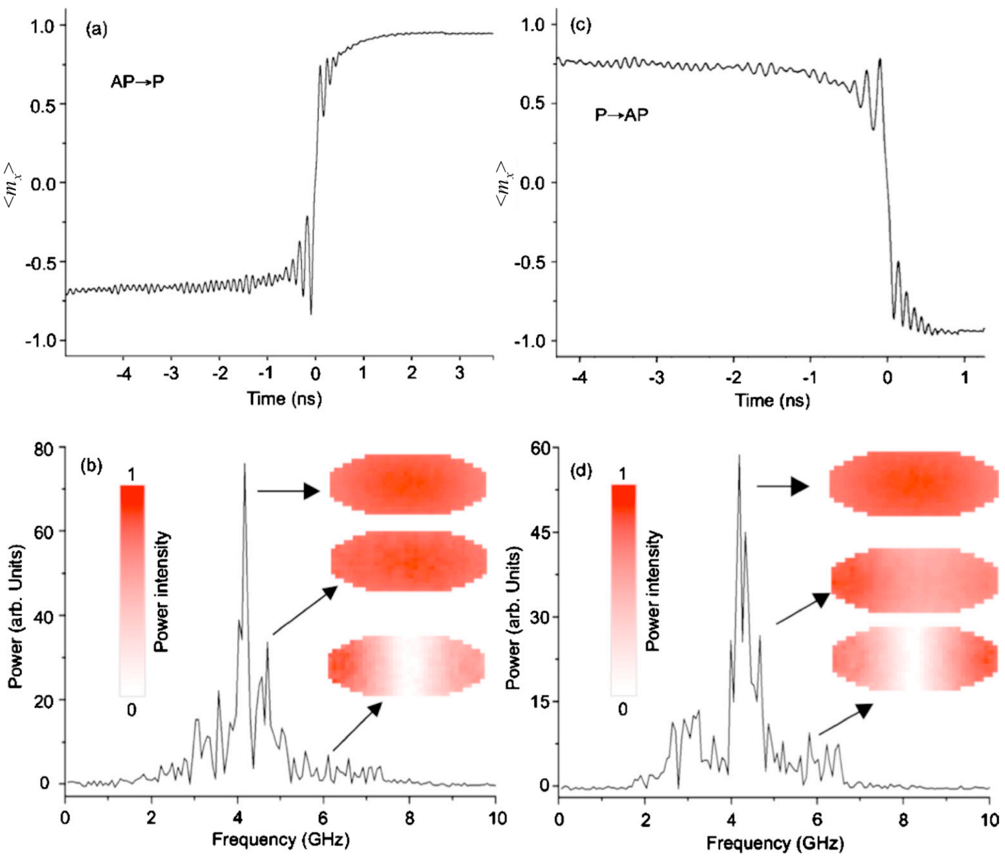

Fig. 45 – Reversal study at 411 K with current densities below the critical one at T=0K ($J_{c\mathrm{AP}\rightarrow\mathrm{P}}$=3.0×10^6 Acm^{-2}; $J_{c\mathrm{P}\rightarrow\mathrm{AP}}$=-6.1×10^6 Acm^{-2}). Transpose average of 100 simulations for the AP to P transition (J=2.9×10^6 Acm^{-2}) (a), and P to AP transition (J =-6.0×10^6 Acm^{-2}) (c). Frequency analysis of (a) and (c), show the dominant central mode at roughly 4.3 GHz, (b) and (d), respectively.

The influence of the PSTT to the thermal oscillations observed in Fig. 45 (a) and (b) was also studied by increasing its magnitude up to 20% (Fig. 46 (a) and Fig. 46 (b)) and 30% (Fig. 46 (c) and Fig. 46 (d)), of the total STT term by attributing the adequate value to ξ in equation (99), as proposed in [76],[95],[96]. Considering the switching processes and respective frequency analysis, as presented in Fig. 46, it is concluded that there is little change to the pre- and post-switching oscillations shown in Fig. 45 (a), since the main modes are still central ones of roughly the same frequency.

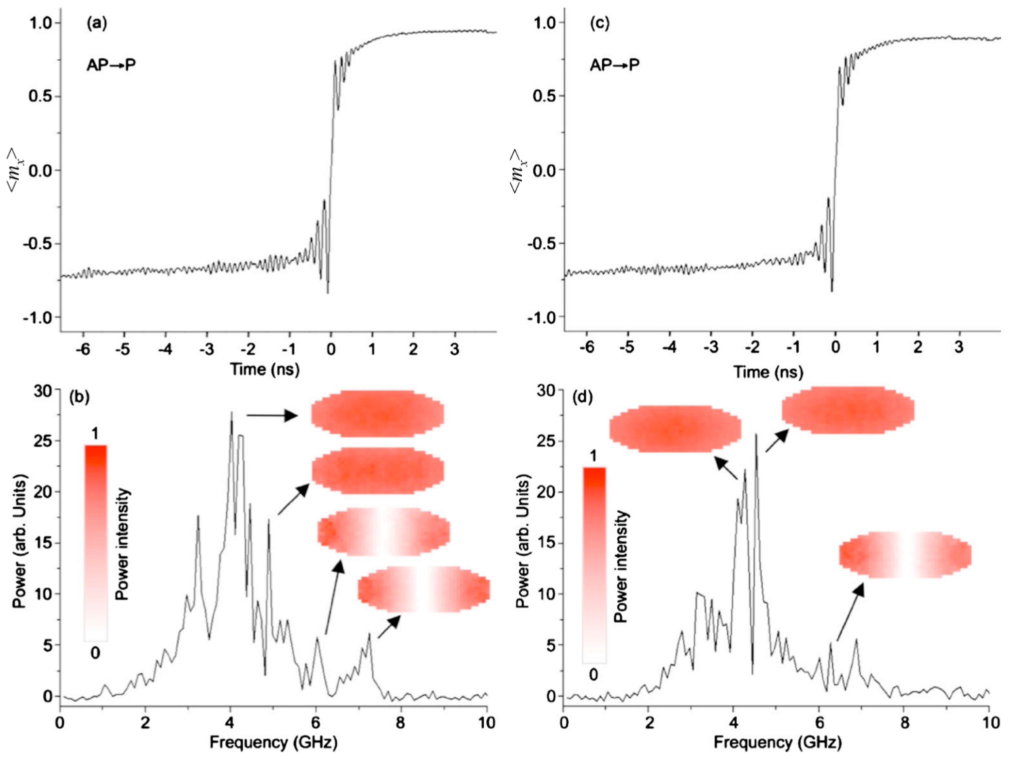

Fig. 46 – Influence of the PSTT on the thermal oscillations observed in Fig. 45 (a). Averaged magnetization for the AP→P transition with a PSTT of 20% of the total STT term a) and a PSTT of 30% of the total STT term c). Frequency analysis of a) and c) both showing the dominant modes as central ones between 4.0 and 4.5 GHz, b) and d), respectively.

Conclusion

In summary, a micromagnetic study of the reversal process in MTJs in the presence of thermal fluctuations has been performed. Unlike previously reported works [77],[92] no incubation delay is observed. Moreover, the pre- and post-switching dynamics of non-uniform modes (edge and central modes [71],[89]) was found to be similar to the one reported experimentally and theoretically [91]. It was also found that if even a very small phase misalignment is present during the averaging procedure, it could result in the disappearance of these dynamics. Furthermore, two different switching regimes are identified for currents above ($|J|>|Jc|$) and below ($|J|<|Jc|$) the critical current at 0 K. In the thermally assisted switching regime ($|J|>|Jc|$), the main effect of the thermal field is the reduction of the switching time, with the STT being the main driving force for the reversal process. In the thermally induced switching regime ($|J|<|Jc|$) however, the thermal field becomes the main driving force for the switching through the promotion of central modes.

Also from this study it is patent the necessity of having a faster simulation code, like the parallel GPU one that was developed during the work discussed in this thesis, since the months long stochastic simulations although feasible for this study of a small sample (504 cells), becomes prohibitive for large samples or longer simulation times.

4.2 Studies made using the CUDA parallel-GPU micromagnetic code

This part is dedicated to the description of two studies made by using the developed parallel GPU micromagnetic code. The first study is on vortex self-oscillations in spin-valves, where both pinned and free layer are simulated dynamically in the presence of STT from one layer over the other simultaneously (back-torque). This simulation study would be extremely difficult to perform in a sequential CPU-based code, not only because there are more layers that have to be computed dynamically, but also because it is very expensive in terms of the computational times that are needed to obtain high resolution in the frequency spectrum (10 microseconds).

The second study focuses on different types of domain wall current-driven dynamics in long ferromagnetic wires with squared cross-section in the presence of Oersted field. The main advantage to this study, besides the very good simulation time performance of the developed parallel code, refers to large number of cells involved in these dynamic computations (in the order of 10^6 cells), which are practically impossible to perform with a sequential code. (In particular the Fortran code, from which the parallel developed one was based on, could not handle more than half a million cells without crashing).

Both the aforementioned works are on final publishing procedures in the IEEE Transactions on Magnetics journal [113], at the writing of this thesis.

4.2.1 Intrinsic and thermal linewidths of spin-transfer-driven vortex self-oscillations

Introduction

In a previous work of the research group, in which this thesis was developed, [97] it was described by means of micromagnetic simulations the spin-transfer-driven vortex self-oscillations experimentally found in Py/Cu/Py elliptical spin valves [98]. The experimental linewidths range from 0.3 MHz to 60 MHz thus, as it was stated in [97], “the linewidth computed numerically cannot be compared to the experimental data because the 5×10^{-8} seconds simulation time limits the resolution to 20 MHz”.

In this work the developed parallel GPU micromagnetic code was used to perform very long micromagnetic simulations up to 10^{-5} seconds, which allows a spectral resolution of 100 kHz. The longest simulation is carried out in a standard GPU in more or less 60 hours. Therefore, a systematic analysis can be carried out using a server with several GPUs.

Spin-Transfer Nano-Oscillators (STNOs) based on magnetic vortex self-oscillations have demonstrated stable dynamics in the gigahertz and sub-gigahertz frequency range at room temperature experiments [98], and are thus promising candidates for viable STNOs. This detailed micromagnetic study will help gain an insight into the intrinsic and thermal linewidths that characterize the oscillation modes.

The analysis of the spectrum of non-linear STNOs needs to be confronted with the introduction of thermal noise, taking into account all the difficulties inherent to the fitting of noisy linewidths [99]. The linewidth strongly depends on the temperature, going through different regimes due to the effect of the phase fluctuations and spatial inhomogeneities [99]-[107]. In detail, the long simulations performed at T=0 K show the intrinsic linewidth of the oscillation modes as function of the bias current. On the other hand, the computations for 0 K < T < 300 K show the influence of the thermal noise to such modes.

Simulation details

The studied device was the one experimentally measured in reference [98]: Py(5nm)/Cu(40nm)/Py(60nm) with elliptical cross sectional area (160 nm × 75nm, with cell discretization of 5×5×5 nm^3) as shown in Fig. 47. The dynamic equation used is the one described by (53) with the polarization function given by (51), which in this study is calculated once for each layer (thick and thin), so as to reproduce, qualitatively, the experimental behavior of the torque induced by both layers on their counterparts (back torque; section 2.3.7 and 3.6.6).

The magnetostatic field has been calculated by solving the magnetostatic problem for the entire spin-valve. A Cartesian coordinate system is considered such that the x-axis is the easy axis of the ellipse whereas the y-axis is the hard in-plane axis. Positive current polarity (+z-axis) corresponds to an electron flow from the thinner to the thicker layer of the spin valve (Fig. 47).

The parameters used in simulations were the same as in [96]: M_S=M_{SP}=6.5×10^5 Am^{-1};

χ=1.5 and η=0.38; A=1.3×10^{-11} Jm^{-1}; α=0.01. A perpendicular field (along the *z*-axis) of 160 mT is applied throughout all the presented simulations. The time window is extended to 10^{-6} seconds for most simulations and 10^{-5} seconds in the cases where more resolution is needed, and using a time step of 500 fs with the 6th order Runge-Kutta algorithm. Note that such temporal windows cannot be studied using the sequential micromagnetic code in a reasonable amount of time. Magnetoresistance is computed over all ballistic channels and using a cosine angular dependence [97]. The thermal activation is introduced as it was previously discussed in section 2.3.6.

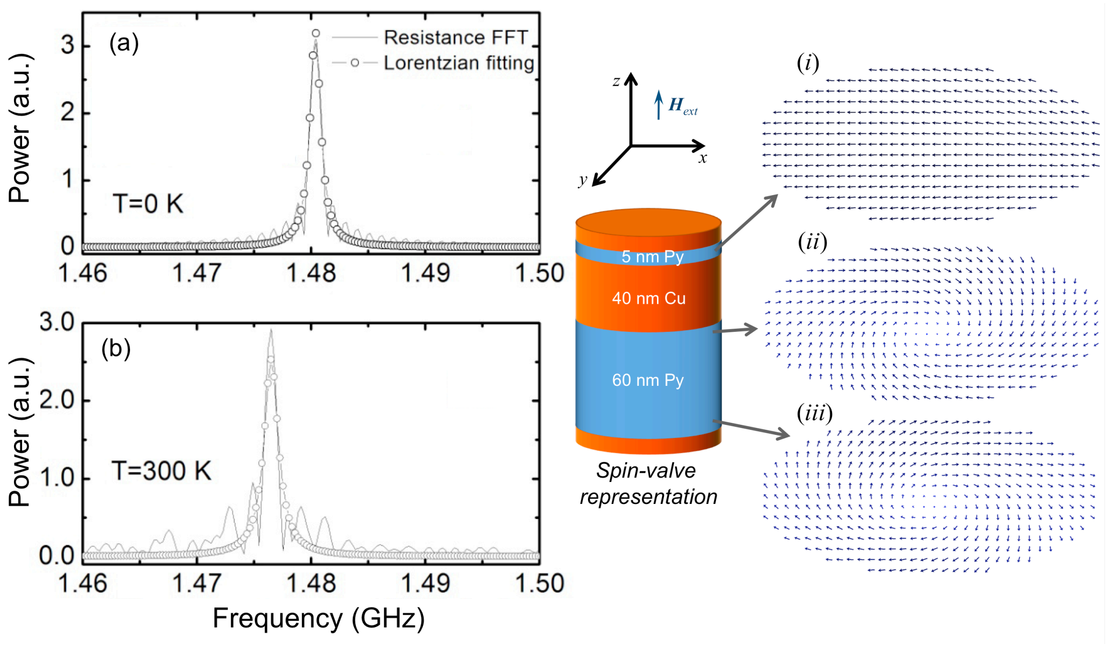

Fig. 47 – Sketch of the spin-valve and linewidth (solid line) for J=10×10^7 Acm^{-2} together with lorentzian fitting (dotted line) at (a) T=0 K and (b) T=300 K. Representation of the initial magnetizations in the thin Py layer (*i*) and in the top (*ii*) and bottom (*iii*) slices of the thick Py layer showing the vortices.

Results and discussion

i) Intrinsic and thermal linewidths

In Fig. 47 (a) the frequency spectrum around the main peak, for an applied current density of 10×10^7 Acm^{-2} at T=0 K, is displayed. The extremely narrow and clean peak is fitted by a lorentzian function resulting in a linewidth of about 1.1 MHz. This low linewidth is produced by coherent rotations of the vortex formed in the thick layer of the spin valve, as can be observed in the M_x versus M_y trajectory shown in Fig. 48 (a). The STT excites the vortex rotation, which from its initial state reaches a clearly defined stationary orbit. The thin layer presents non-uniform configurations but still following quasi-periodical trajectories like the one shown in Fig. 48 (b).

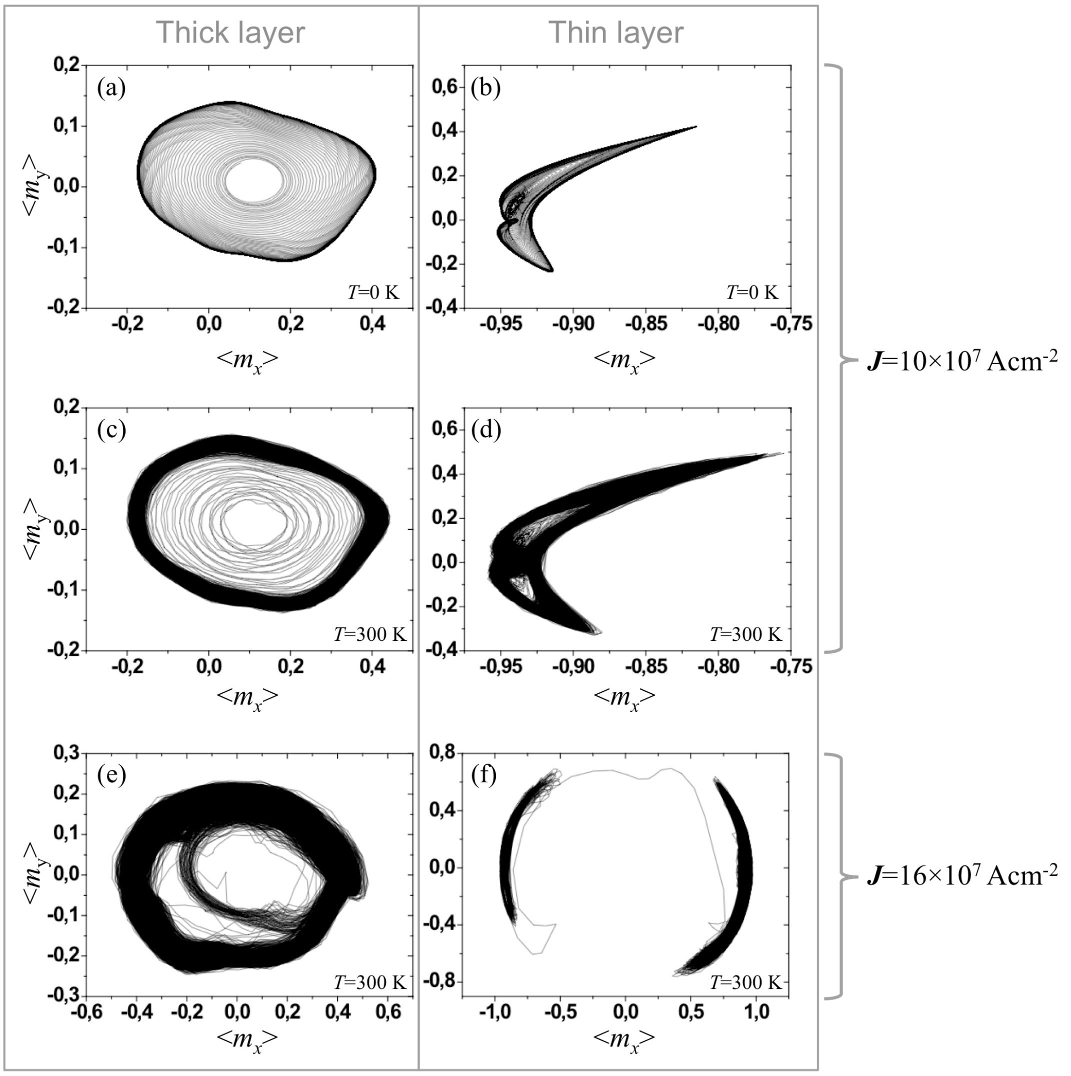

Fig. 48 – Average magnetization trajectories in each layer: $<m_x>$ versus $<m_y>$ trajectories for the thick (a, c, e) and thin (b, d, f) Py layers. Current density is J=10×10^7 Acm^{-2} in (a-d), T= 0 K in (a-b) and T= 300 K in (c-d). Current density is J=16×10^7 Acm^{-2} and T= 300 K in (e-f).

In Fig. 47 (b) the spectrum computed for J=10×10^7 Acm^{-2} at T=300 K is shown. As expected for a non-linear STNO, the thermal noise generates a linewidth increase as well as a small frequency shift. Comparing Fig. 48 (a) and Fig. 48 (c), it can be observed that the orbits of the average magnetization are preserved in spite of the thermal activation so that the linewidth remains below 1.5 MHz as observed in experiments [98]. The vortex oscillator within the spin-valve shows itself as a robust oscillator against thermal noise for a certain range of current. The magnetization trajectory within the thin layer at T=300 K (Fig. 48 (d)) differs significantly with respect to the T=0 K case (Fig. 48 (b)). In particular, the averaged $<m_x>$ component for T=300 K expands towards positive (less negative) $<m_x>$ values due to the thermal activation. This aspect seems to indicate the possibility of more a complicated behavior of the device, as it will be addressed later when explaining the general dependence on current and temperature.

The linewidth and peak frequency dependences on the applied current and temperature are presented in Fig. 49. In Fig. 49 (a), when increasing the current a slight increase in the linewidth is observed up to currents around 16×10^{7} Acm^{-2}. This increase is due to the modification of the dynamics in a similar way to what was commented in regard to Fig. 48 (a-d). Hence the general characteristics of the dynamics are preserved, although the orbits are noisier.

For larger currents ($J \geq 16\times10^{7}$ Acm^{-2}), a more noticeable increase in the linewidth is detected (Fig. 49 (a)). Having a look at the dynamics within this range (see Fig. 48 (e-f)) a change in their general features is detected. The vortex within the thick layer (Fig. 48 (e)) rotates now in a larger and noisier orbit, changing also the sense of rotation of the vortex. This orbit jump is motivated by a change in the thin layer configuration which goes from its originally *"$-m_x$"* state to a *"$+m_x$"* magnetization state (Fig. 48 (f)). This change in orientation of the magnetization within the thin layer implies a change to the sense of rotation of the vortex in the thick layer, since the torque experienced by the vortex in the different regions ($+m_x$, $-m_x$) changes sign. The polarity of the vortex remains always positive since a perpendicular applied field of +160 $\boldsymbol{u}_z$ mT is always present.

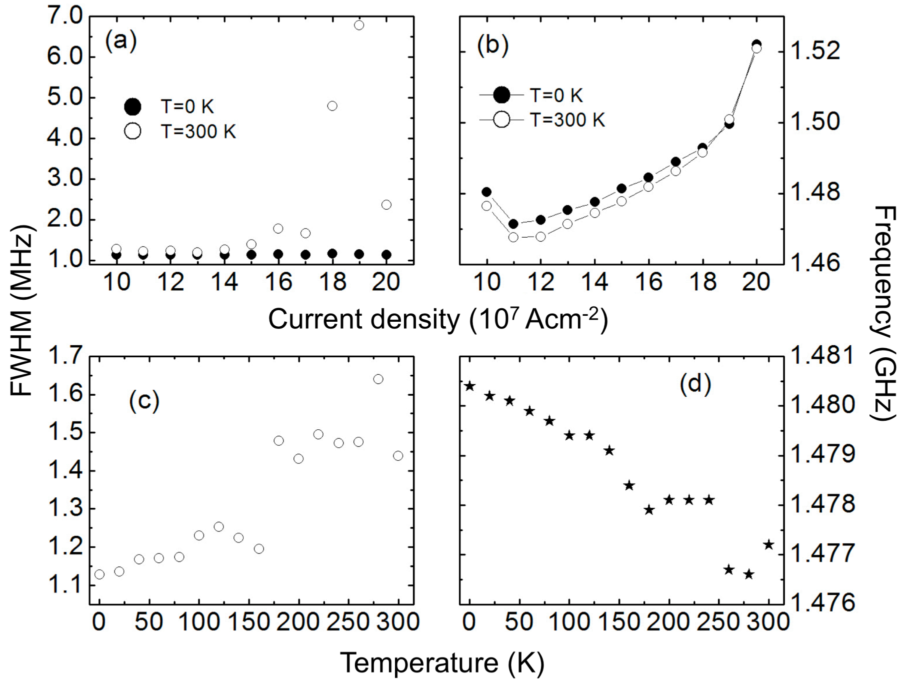

Fig. 49 – Linewidth and frequency dependence on temperature and current density. Linewidth (a) and frequency (b) for J ranging from 10×10^{7} Acm^{-2} up to 20×10^{7} Acm^{-2} at T=0 K and T=300 K. Linewidth (c) and frequency (d) for J=10×10^{7} Acm^{-2} for a temperature range from T=0 K up to T=300 K.

Therefore, there are two oscillation modes present, which appear as a result of the complete analysis of the whole device and the use of the back-torque. They are not thick-layer or thin-layer isolated modes but *"coupled device modes"* [32]. The jump to the second mode depends on the thermal noise (different computational realizations can give rise or not to the jump) and also on the used computational time window. The jump time scale is the same as the one measured by Pribiag *et al.* [108], in devices at zero bias field. The temporal window used in the computations was of 10^{-6} seconds for the fitting, always choosing realizations where the two modes were present. In a real experiment the temporal window is very large, which is why one can reason that both modes are always present. In any case, for J>15×10^{7} Acm^{-2}, an analysis of the thermal noise contribution to linewidth generation cannot be carried out in terms of analytical theories [99]-[107].

The frequency of the vortex oscillation is shown in Fig. 49 (b). Always the same mode was followed and the frequency exhibits a blue-shift (increase in frequency), which is in agreement with experimental observations [98]. The difference between the temperatures T=0 K and T=300 K diminishes with the increase in current, since the magnitude of the thermal noise becomes less important regarding the larger intensity of the STT.

The temperature dependence on the linewidth and frequency for J=10×10^{7} Acm^{-2} is shown in Fig. 49 (c-d), with no change in mode observed for any of the temperatures considered.

In a previous work performed by the research group [99] it was found that micromagnetic simulations gave a qualitative agreement with the theoretical work of Tiberkevich *et al.* [103] regarding STNOs. In this theoretical work, two different regimes can be observed depending on the temperature. The low temperature regime is characterized by a linear dependence of the linewidth with T, whereas the high temperature regime presents $T^{1/2}$ dependence [103]. In the mentioned work [99], micromagnetic simulations on a different spin-valve system certainly found this type of behavior and it was ascribed to small temporal and spatial inhomogeneities of the uniform magnetization model [99]. Since in those spin-valves just small deviations from the uniform magnetization states were present, the theoretical analysis of [103] was proved to be adequate.

In the present work, however, highly non-uniform vortex configurations are being dealt with, so that analytical theories are not even valid to describe single mode dynamics, like in the case of Fig. 49 (a-b). Nevertheless, linear behavior seems to be present in Fig. 49 (a) for T<140 K and also a lowering in frequency with temperature is detected as in other linewidth temperature studies [99]-[107].

ii) Higher harmonics analysis

Finally, the second and third harmonics of the frequency spectra of the vortex STNO was analyzed in order to check the recent study of Quinsat *et al.* [109]. In a non-linear resonator, there is a relation between the corresponding linewidth of the n^{th} harmonic and the linewidth of the main mode. In particular, in a non-isochronous oscillator the relation for the linewidths of higher harmonics are smaller than predicted in isochronous oscillators. In fact, non-isochronous auto-oscillators like STNOs are expected to show an increase in the linewidth with the harmonic order [109]. This increase for the n^{th} harmonic (Δfn) for a non-isochronous STNO is shown to be in the range of $n\Delta f_1 < \Delta f < n^2\Delta f_1$, [109].

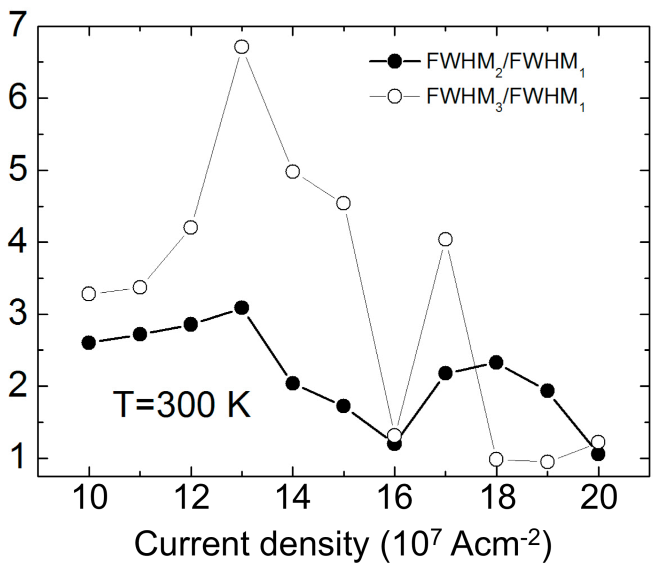

Fig. 50 – Ratio between the linewidth of second and third harmonic at T = 300 K for a J ranging from 10×10^7 Acm^{-2} to 20×10^7 Acm^{-2}.

The obtained results of the vortex STNO for the second and third harmonic are shown in Fig. 50. At lower currents (J<14×10^7 Acm^{-2}) it was observed an agreement with the theory, $2\Delta f_1<\Delta f_2<4\Delta f_1$ and $3\Delta f_1<\Delta f_3<9\Delta f_1$. At larger currents, again, not even a qualitative agreement is found. This fact confirms that the presence of other possible coupled oscillation modes inhibits a simple linewidth dependence that could be described by analytical theories.

Conclusion

In summary, a detailed analysis of spin-valve vortex STNOs was carried out, showing that several device coupled modes can be present depending on the temperature and applied current [107]. When just one mode is excited, the STNO shows linear linewidth dependence at low temperatures. In these cases, the second and third harmonics show a linewidth compatible with recent non-isochronous auto-oscillator theories [109].

The use of the developed parallel GPU micromagnetic code was fundamental in this study, since it allowed for large temporal window simulations that in turn give high resolution when analyzing the oscillations in the frequency spectrum. The used time window of 10^{-6} seconds is completely unpractical when using sequential codes, since a single simulation would take many months.

4.2.2 The role of the Oersted field on the current-driven domain wall dynamics along wires with square cross section

Introduction

As mentioned in chapter 1, a very interesting finding in recent years has been the behavior of domain wall dynamics in magnetic strips at the nano-scale. In these long structures the magnetization gradually changes from one domain to another through a DW structure like for example, a typical head to head or tail to tail transverse wall (TW) configuration, or even more complex types of DW like the Bloch-point wall (BPW) [7]. Domain wall dynamics in nano-wires or nano-strips, can be induced by either external magnetic fields or by injecting spin-polarized electrical currents [3],[4],[8],[110]. This discovery could have significant consequences on the development of new magnetic storage devices in which data is moved electronically rather than mechanically as it is done in today's computer hard disk drives. A new type of memory based on current-driven DW dynamics, the racetrack memory, has been proposed by Parkin [6].

In this work, it was first studied the dependence of the type of DW nucleated in respect to the squared section size of the strip. Then, the dynamics was investigated for each type of wall TW and BPW (Fig. 51), under different values of in-plane spin-polarized currents and observed if the DWs experience the Walker breakdown [7], with and without the influence of the Oersted field. The developed parallel GPU micromagnetic code was used so as to manage the large structure of the ferromagnetic strip, which ascends to the order of 10^6 cells, in a reasonable amount of time.

Simulation details

Since the applied current density is in-plane with the ferromagnetic strip, the dynamic equation used in this study was the one described by (58), thus including the effect of both the adiabatic and non-adiabatic spin-torques. All the usual magnetic fields are considered for the effective field $\boldsymbol{H}_{eff}$ (magnetostatic, exchange), including also the Oersted field (2.3.5 and 3.3.5), generated by the in-plane current that runs along the length of the wire. No external field is applied, hence the DW dynamics is driven only by the in-plane spin-polarized current density $\boldsymbol{j}=j\boldsymbol{u}_x$. The convention used for the current density is that for positive values, the electrons flow along the positive direction of the wire x-axis (Fig. 51 (a)). The current was also considered to be DC, applied from the beginning to the end of the simulation and uniformly distributed along the nano-strip. Typical Permalloy parameters were considered: $M_S=8.6\times10^5$

Am^{-1}; exchange constant A= 1.3×10^{-11} Jm^{-1}; damping parameter α=0.02; and polarization factor η=0.4.

Regarding the geometry of the simulated wire, it was considered an 800 nm long strip, in which the squared section lateral size L was varied from 10 nm up to 100 nm in order to assert which type of DW is the most stable for different values of L. Also, in order to avoid the demagnetizing field effect of the borders upon the DW, the wire was made "infinite" by computational manipulation [114]. This was achieved by first calculating the demagnetizing field generated by the artificially imposed "magnetic charges", of the first and last cells of the 800 nm long computational area, and then subtracting this demagnetizing field to the demagnetizing field of the entire sample. In all simulations the initial magnetization configuration was head-to-head, for both types of DW, as show in Fig. 51. The numerical solver used was the forth order Runge-Kutta (3.4.2), with a time step of 63 fs, and the mesh composed of 2×2×2 nm^3 cells.

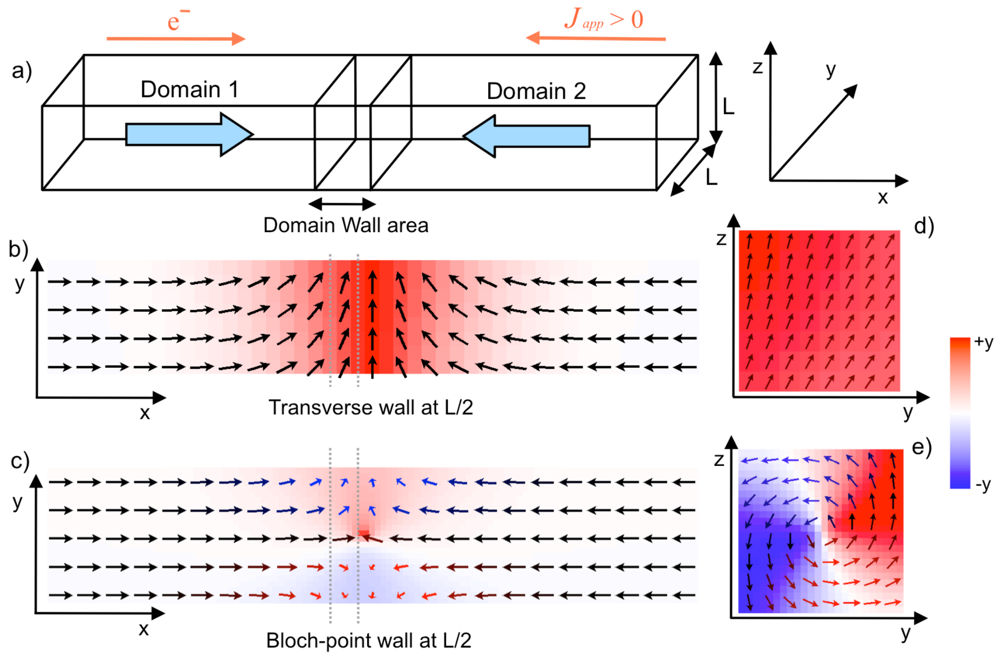

Fig. 51 – (a) Representation of the nano-wire showing the head to head configuration used in this work. (b) Transverse wall at $L/2$ from the z-axis viewpoint, the dotted lines represent the slice of the nano-wire depicturing in (d) its transversal view. (c) Bloch-point wall at $L/2$ from the z-axis viewpoint, the dotted lines represent the slice of the nano-wire depicturing in (e) its transversal view.

Results and discussion

i) Equilibrium DW configurations

The first part of this study was focused on investigating which DW configurations, either the TW or the BPW, are the most stable for different values of L. This was achieved by varying L from 10 nm to 100 nm, starting from a magnetic configuration that favored either the TW or the BPW and then letting the system reach its equilibrium state. The results showed that for lateral sizes L<30 nm of the wire the most stable DW configuration is the TW, whereas for values of L>40 nm the BPW is the most energetically favored state. For values of L between the previously mentioned ones, the equilibrium configuration is a complex one, appearing to be a mixture of the two types of DW. (When current was applied to these complex DW structures, the wall continuously transformed into several different wall structures).

These results are in good agreement with the ones reported in [7]. Due to the physical nature of each DW type and its dependence on the lateral size L, it was chosen to focus the study on two specific values of the squared cross-section. The chosen sizes for the detailed study of the dynamics were, L=16 nm for the TW and L=48 nm for the BPW.

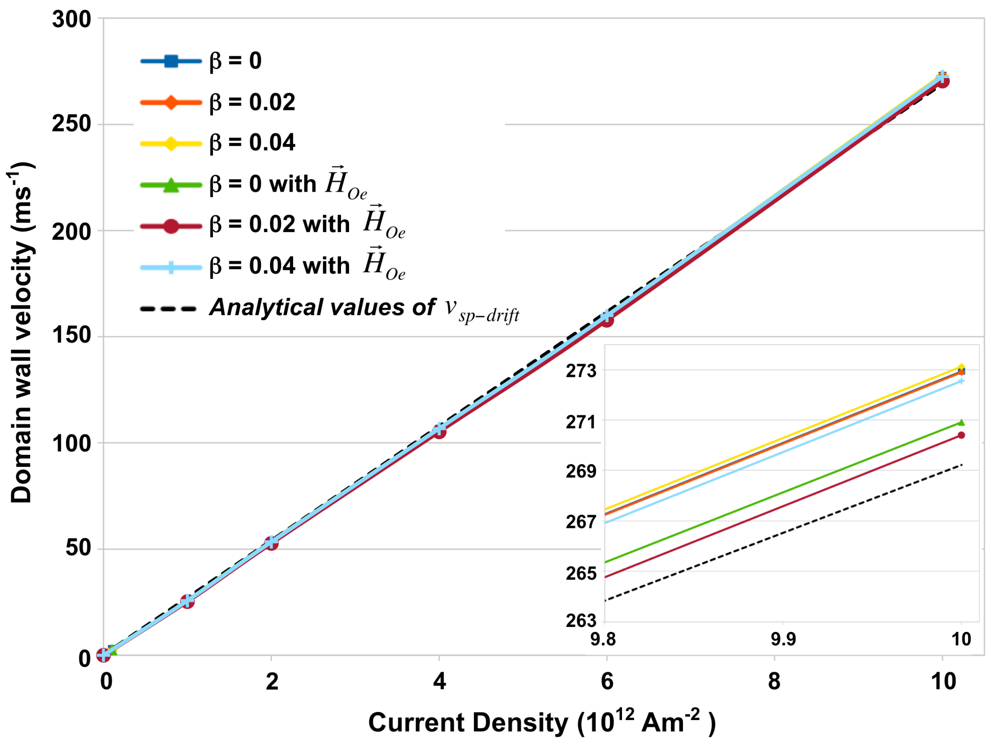

Fig. 52 – Transverse wall velocity *versus* DC current densities for a nano-wire of L=16 nm. The DW velocity increases linearly with the current and it is practically unaffected by the presence of the Oersted field or the variation of the non-adiabatic parameter β, inset: zoom of the last part of the graph. The dashed line represents the analytical value of the maximum DW velocity $\boldsymbol{v}_{sp\text{-}drift}$ (see 2.3.7)

ii) Transverse DW dynamics

Analyzing the results of the TW dynamics in Fig. 52, it shows that the velocity of the DW displacement rises linearly with the different applied DC current densities. Also seen in the inset of Fig. 52 is the fact that the DW motion is practically unaffected by the non-adiabatic parameter *β* or the presence of the Oersted field, which points to in this case, that the DW dynamics is dominated by the adiabatic spin-torque term. The effect of the torques applied on the DW generated by the effective field including the Oersted field versus the spin torque is shown in Fig. 53. There, it is seen that for each cell along the *x* direction the torque caused by the effective field (scattered points) is in module several times smaller than the spin-torque effect (solid lines). This difference in torque represents why the presence of the Oersted field has so little effect to the TW dynamics.

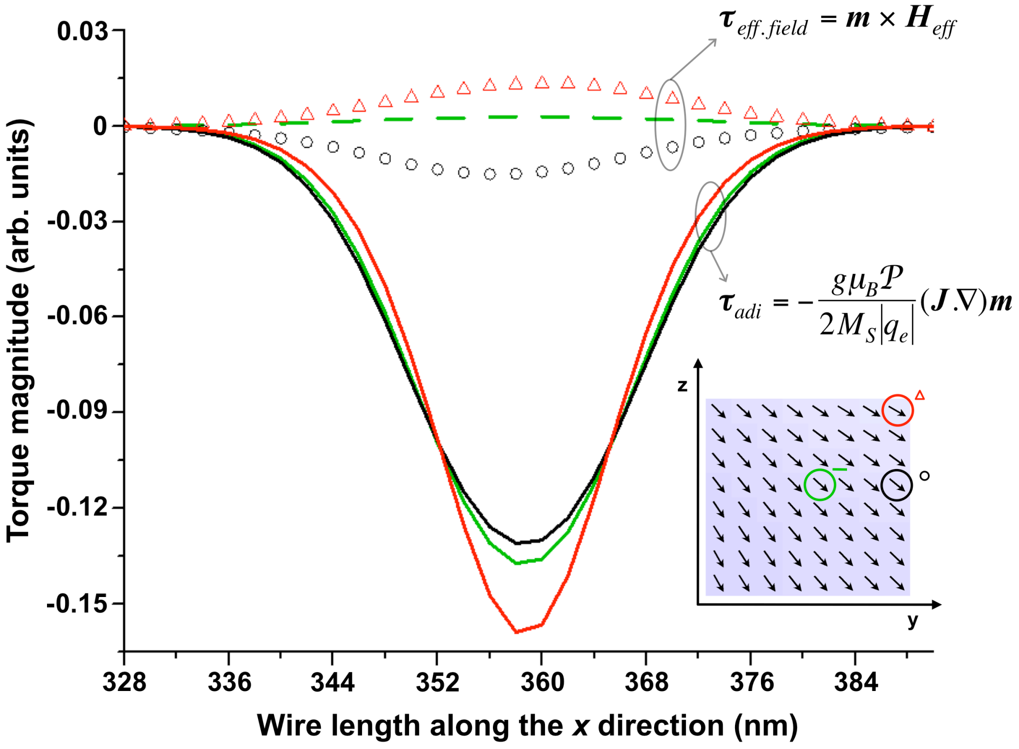

Fig. 53 – Comparing the torques magnitude between the effective field (with Oersted field) (scattered/dashed points) and the spin-torque (solid lines), in the perfect adiabatic case *β*=0, along *x*-component of the wire and in different points of the cross-section (see inset). The spin-torque is several times larger than the effective field torque in the DW area. The inset shows a slice of the wire in the *z*0*y* plane and from which cells along the *x*-axis the plotted data was taken (J=10^{13} Am^{-1}).

Another result is displayed in Fig. 54, where it is shown that while the DW is moving along the strip it rotates clockwise for *β*<*α* (Fig. 54 (a)) and counter-clockwise if *β*>*α* (Fig. 54 (b)). Not shown is the case when *β*=*α* in which the DW moves without rotating. These results are in accordance with the results obtained by M.Yan *et.al.* [9] for cylindrical wires. In that work, they adopted an analytical model that allowed them to understand that the linear velocity is independent of *β* and that the angular velocity responsible for the rotation of the DW depends on (*β*–*α*). This description of the DW dynamics, associated with the symmetry of the cylindrical wire permitted them to describe that the DW beats the Walker breakdown and its intrinsic pinning, by being massless. Döring introduced the concept of DW mass in 1948 [111]. He discovered that the structure of a Bloch wall moving with velocity $\boldsymbol{v}$ differs from that of a static one, and that its energy increase is porpotinal $\boldsymbol{v}^2$. This allows for the definition of a kinetic energy and a mass. However, the DW does not have real mass, since there is no actual material displacement, what happens is that the translation of the DW profile contains some inertia because, in a first step the spins need to be rotated to the wall plane, which costs energy [18]. Moving DWs in thin magnetic strips have displayed particle-like behavior such as momentum and inertia [112]. The connection between DW mass and the Walker limit is given by the dynamic modification of the DW structure during its montion and the resulting energy density increase. This increase in energy continues until it reaches a limit where the micromagnetic structure collapses. Therefore the speed limit of the DWs is related to the kinetic energy and is inevitable for massive walls. However, if like in the results observed here the DW does not change its structure during the motion, thus moving rigidly, such DW is massless and with zero kinetic energy, which results in the absence of a Walker-type speed limit. The results of the simulated squared-section wires showed all of these same properties, like in [9], and thus it can be said that for DWs in wires with squared-section L<30 nm, the wall's dynamics is practically the same as to the ones seen for cylindrical wires where the shape anisotropy is zero.

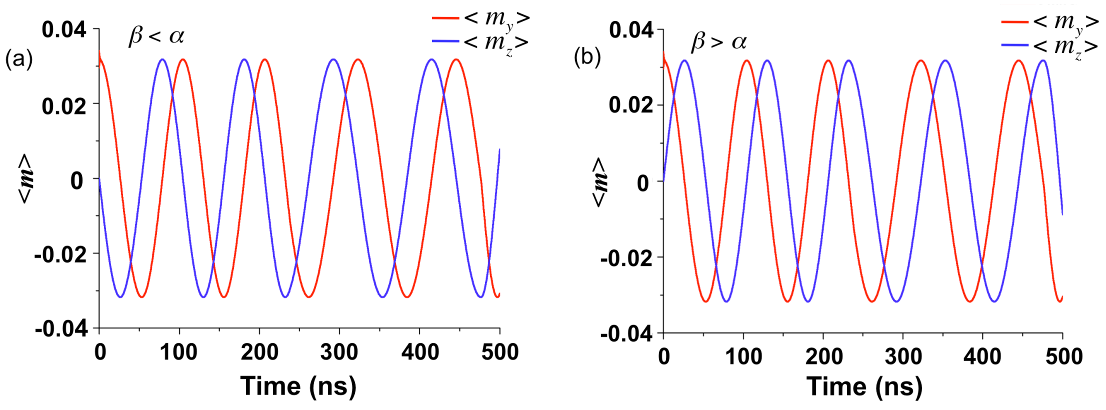

Fig. 54 – Average magnetization during the transverse wall displacement. Here it is seen that for values of the non-adiabatic parameter, β, smaller than the damping parameter, α, the DW rotates clockwise (a), whereas for values of β larger than α the wall rotates counter-clockwise (b). Not shown is when β equals α for which the wall moves rigidly without precessing.

iii) Bloch-point DW dynamics

The dynamics found for BPW in squared-section wires was substantially different. From Fig. 55 it can be seen that there is a threshold current that has to be overcome in order to move the DW, and that it depends on the value of *β*. This might open the possibility to roughly determine the value of *β* in real experiments with strips where BPW are present. Also independently of *β,* the BPW always moves rigidly, unlike the previously seen TW, which would rotate clockwise or counter-clockwise depending on the value of *β*. Analyzing the dependence on *β*, it is seen that this is a main feature in order to get the BPW moving, since for the perfect adiabatic case *β*=0 there is no DW displacement for any value of current, and that as the value of *β* increases so does the velocity (Fig. 55). For current densities above 10^{13} Am^{-2}, the DW transforms into complex structures that are continuously changing in a disordered way. Also seen in Fig. 55, for *β*=0.02 and *β*=0.04, is the existence of steps/plateaus, in which the DW moves for different applied current densities at practically the same velocity without significantly changing its structure. Further study is needed to better understand these different phenomena.

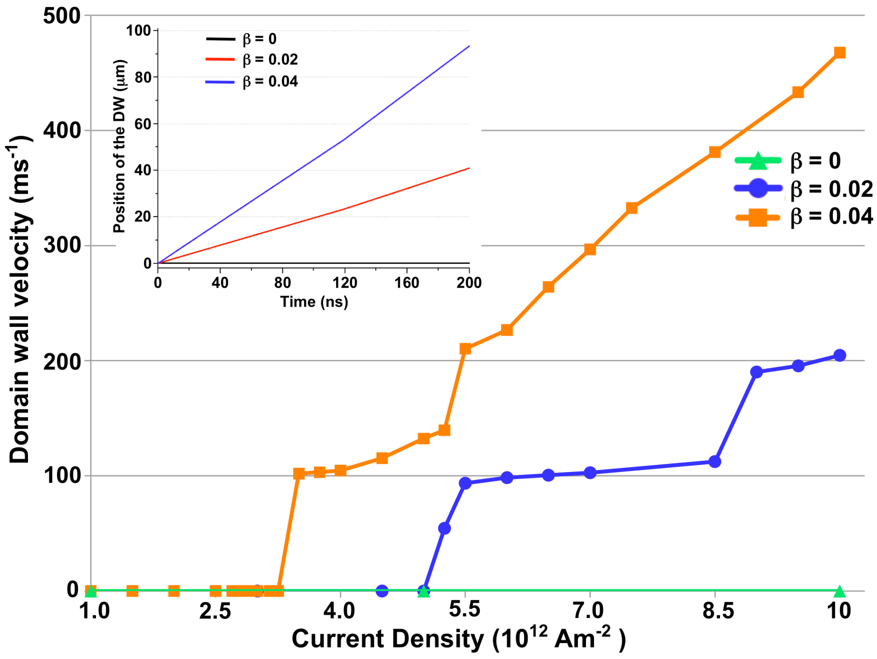

Fig. 55 – Bloch-point wall velocity *versus* current density for a nano-wire of *L*=48 nm. No Oersted field applied. It can seen that there is a threshold current in order to move the DW. For this type of DW the velocity varies non-linearly showing fast increases or plateaus for certain values of the current density. The inset shows how the movement of the DW strongly depends on the value of the non-adiabatic parameter *β*, for an applied current density of 10^{13} Am^{-2}.

The influence of the Oersted field on this type of wall differs from the one seen in TWs. Its presence contributes, in general, with an increase of the DW velocity (Fig. 56). However, this phenomena was only seen in the *β*=0.04 case, since when including the Oersted field for *β*=0.02, no DW movement is seen. In Fig. 56 it is also shown that the DW movement is independent of the chirality of the BPW in the absence of the Oersted field. (The chirality in this case refers to direction of the magnetization in the DW area, clockwise or counter-clockwise, as seen from +*x*). However, when applying the Oersted field the counter-clockwise chirality is favored. It is not totally clear why one chirality favours the DW dynamics more than the other in the presence of the Oersted field.

The BPW case requires further study since there are some phenomena that are still not well understood. Apart from the previously mentioned existence of steps and chirality dependence in the presence of the Oersted field, there is also the fact that the DW for some positive values of the current density moves from left to right and for other values of ***J*** from right to left. And why is it that when applying the Oersted field it hinders the movement of the DW for β=0.02 and below.

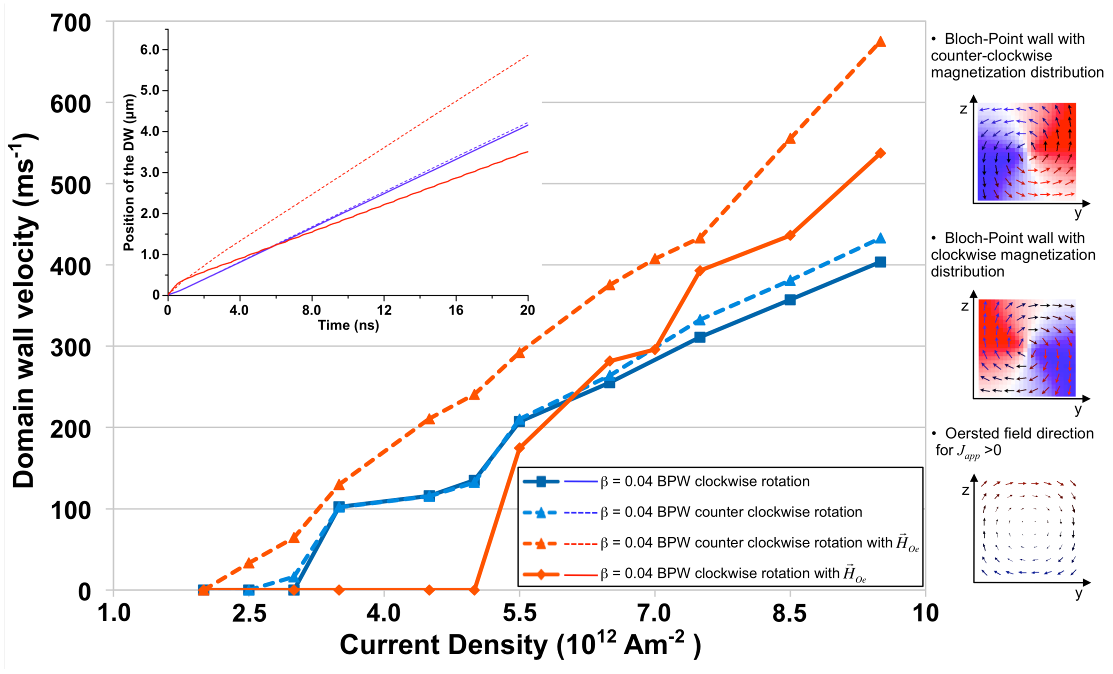

Fig. 56 – Bloch-point wall velocity *versus* current density for a nano-wire of *L*=48 nm for the same value of the non-adiabatic parameter, β, with and without Oersted field. The DW velocity varies non-linearly and in the absence of the Oersted field the DW dynamics is practically unaffected by the difference in the direction of the magnetization. In the presence of the Oersted field the counter clockwise magnetization (as viewed from +*x*) of the BPW favors the wall's movement. In the inset it is shown the DW position in function of the elapsed time, for an applied current density of 5.5×10^{12} Am^{-2}.

Conclusion

In summary, it was studied the dynamics of DWs in long squared cross-section Permalloy strips. It was found that the type of stable DW depends on the size of the cross-section *L*, in which for *L*<30 nm the stable wall is a TW, whereas for *L*>40 nm the BPW is the most stable. For TW configurations the DW velocity increases linearly from zero and it is practically independent of the value of β and the presence of the

Oersted field, which points to more spin-torque dependent dynamics. The TW dynamics are also the same as the ones seen in cylindrical wires and there is no Walker breakdown. In the BPW case, there is a threshold current in order to displace the DW. There is a range of current density values in which the wall moves rigidly without being destroyed, however the rise in velocity with the current is not linear. The effect of the Oersted field on the BPW requires further study but in general and for some values of β, it appears to increase the DW velocity and there is also a dependence to the DW dynamics between the Oersted field and the BPW chirality.

The simulations of large structures as the wires studied here benefit greatly from the developed parallel GPU micromagnetic code, since the number of cells involved (in the order 10^6 cells), are very difficultly handled by a CPU sequential code, requiring months to solve a single simulation.

5 Conclusions

The main goal of the work described by this thesis, was on the development of a new parallel GPU micromagnetic simulation code so as to overcome the temporal and spatial limitations of a sequential CPU-based code. These types of limitations arise when trying to make rigorous studies of: stochastic simulations, frequency oscillators, dynamic multilayer analysis, domain wall dynamics in long ferromagnetic wires, etc. Such limitations are resumed to the fact that they become impractical to perform in real time due to amount of computational cells that are needed to solve or the nature of the problem under study, which might lead to months of simulation time for a single run.

The developed parallel GPU micromagnetic code programmed in the NVIDIA's C-based CUDA language, was successfully implemented allowing for a performance speed up of roughly two orders of magnitude when compared to the sequential CPU Fortran code from which it was based on. Writing in the CUDA parallel language is challenging and it does have a couple of limitations, as the fact that it is restricted to NVIDIA chip-based graphic cards and the size of the problem to solve has to be limited to the card's RAM memory. Nonetheless, even for the large memory demanding simulations performed in micromagnetics, memory issues are seldom a problem, and the price per Gflop of performance obtain far compensate the effort of implementation.

Different interactions were implemented in the developed code, which include the magnetostatic, exchange, anisotropy, Zeeman, Oersted and thermal field contributions to the effective field, as well as the spin-torque effect for both current perpendicular to plane and in-plane devices. These were successfully tested against the well-established Oommf and the Fortran micromagnetic code developed by the research group posted on the Micromagnetic Modeling Activity Group webpage.

In order to study the dynamics of two different types of device, the developed parallel GPU code focused on two parts. On one hand, a two layer dynamic part was developed, so as to simulate the dynamic properties of both the assumed "pinned" layer and "free" layers. Of course in this case there is no real pinned layer since at each time step all the effective field contributions are calculated in both layers as well as the spin-torque effect of one over the other and vice-versa (back-torque). This code was used to rigorously study the vortex oscillation frequencies in spin-valves. On the other hand, a different part of the code focused on the study of domain wall dynamics in long ferromagnetic strips was also developed, which also allowed for the rigorous study of large ferromagnetic samples composed of a great number of computational cells (order of 10^6 cells).

Developing such a micromagnetic parallel code required not only the learning of the demanding CUDA parallel language, but also a profound understanding of the physics involved in the micromagnetic formalism. The resulting programming work served as a basis to a commercial parallel micromagnetic code by the name of GPMagnet, which was further developed in conjunction with GoParallel S.L. This is a spin-off company from which the author of this work is co-founder along with the other members of the research group.

This new micromagnetic simulation tool can now be used to continue to perform systematic and massive micromagnetic simulations, like the ones studied in this work involving dual-layer vortex oscillations and DW dynamics in long ferromagnetic strips. This because it has proven to be a very efficient tool in the study of several physical processes related to the understanding and control of the magnetization dynamics at the nano-scale.

Published works during the development of the PhD thesis

D. Aurelio, L. Torres and G. Finocchio, "Magnetization switching by spin-transfer-torque in high-TMR magnetic tunnel junctions" *J. Magn. Mag. Mater.* **321**, 3913–3920, (2009)

D. Aurélio, L. Torres and G. Finocchio, "Thermal effects on spin-torque-driven switching in high-tunneling-magnetoresistance magnetic tunnel junctions", *J. Appl. Phys.*, **108**, 083911, (2010)

L. Lopez-Diaz, D. Aurelio, L. Torres, E. Martinez, M. A. Hernandez-Lopez, J. Gomez, O. Alejos, M. Carpentieri, G. Finocchio. and G. Consolo, "Micromagnetic Simulations using Graphic Processing Units", *J. Phys. D: Appl. Phys.*, **45**, 323001 (17pp), (2012).

Works accepted for publishing and on final production stages

L. Torres, M. Carpentieri, E. Martinez, L. Lopez-Diaz, A. Hernandez-Lopez, D. Aurelio and G. Finocchio, "Intrinsic and Thermal Linewidths of Spin-Transfer-Driven Vortex Self-oscillations", *IEEE Trans. Magn.*

D. Aurelio, A. Giordano, L. Torres, G. Finocchio, E. Martinez, "The role of the Oersted field on the current-driven domain wall dynamics along wires with square cross section", *IEEE Trans. Magn.*

6 Appendix A

In this appendix the code transcripts of the add operation between the elements of two different arrays used in example 1 of section 3.6.5 is presented. The first is the one using the sequential C code, and the second the CUDA parallel one.

```
/*
 * File:  main.cpp
 * Example of adding the elements of two arrays and saves the result
 * to a third array, in normal C
 */

#include <iostream>
#include <fstream>
#include <stdlib.h>
#include <math.h>

using namespace std;

int main(int argc, char** argv) {

// Declaring the variables
    int ndim=5;
    float *a;
    float *b;
    float *c;
    a = new float[ndim];
    memset(a,0,sizeof(float)*ndim);
    b = new float[ndim];
    memset(b,0,sizeof(float)*ndim);
    c = new float[ndim];
    memset(c,0,sizeof(float)*ndim);

// Loop that sets the array values and performs the add operation
    for(int i=0 ;i<ndim; i++){
        a[i]=1+i;
        b[i]=3;

        c[i]=a[i]+b[i];
    }
// Loop that prints the results on screen
    for(int i=0 ;i<ndim; i++){
        printf("a[%d]=%1.1f   b[%d]=%1.1f    c[%d]=%1.1f\n",
                i,a[i],i,b[i],i,c[i]);
    }

    delete a;                   //
    delete b;                   // freeing the memory
    delete c;                   //

    return (EXIT_SUCCESS);
}
```

```
/*
 * File:   newmain.cu
 * Example of adding the elements of two arrays and saves the result
 * to a third array, in parallel CUDA
 */

#include <iostream>

using namespace std;

/* Kernel prototype*/
__global__ void KernelName(float *dev_a,float *dev_b,float *dev_c, int
ndim);

int main(int argc, char** argv) {

/*Creating the variables on the host*/
    int ndim=5;
    float *h_a;
    float *h_b;
    float *h_c;
    h_a = new float[ndim];
    memset(h_a,0,sizeof(float)*ndim);
    h_b = new float[ndim];
    memset(h_b,0,sizeof(float)*ndim);
    h_c = new float[ndim];
    memset(h_c,0,sizeof(float)*ndim);

/*Creating the variables on the device*/
    float *dev_a;
    float *dev_b;
    float *dev_c;
    cudaMalloc((void**)&dev_a,sizeof(float)*ndim);
    cudaMemset(dev_a,0,sizeof(float)*ndim);
    cudaMalloc((void**)&dev_b,sizeof(float)*ndim);
    cudaMemset(dev_b,0,sizeof(float)*ndim);
    cudaMalloc((void**)&dev_c,sizeof(float)*ndim);
    cudaMemset(dev_c,0,sizeof(float)*ndim);

/*Defining the size of the block and grid*/
    dim3 dimBlock(5);
    dim3 dimGrid(ndim/dimBlock.x);

    printf("block %d   grid %d \n",dimBlock.x,dimGrid.x);

/*Initializing the vectors*/
    for(int i=0 ;i<ndim; i++){
        h_a[i]=1+i;
        h_b[i]=3;
    }
/*Printing the initial values before the add operation*/
     for(int i=0 ;i<ndim; i++){
        printf("a[%d]=%1.1f   b[%d]=%1.1f   c[%d]=%1.1f\n",
                i,h_a[i],i,h_b[i],i,h_c[i]);
    }printf("\n");
/*Copying the values of a and b to the GPU device variables dev_a and
dev_b*/
    cudaMemcpy(dev_a, h_a, sizeof(float)*ndim, cudaMemcpyHostToDevice);
    cudaMemcpy(dev_b, h_b, (sizeof(float)*ndim), cudaMemcpyHostToDevice);

/* Kernel call*/
    KernelName<<<dimGrid,dimBlock>>>(dev_a,dev_b,dev_c,ndim);

/*Returning the result of the kernel operation to the host*/
    cudaMemcpy(h_c,dev_c,sizeof(float)*ndim,cudaMemcpyDeviceToHost);

    for(int i=0 ;i<ndim; i++){
        printf("a[%d]=%1.1f   b[%d]=%1.1f   c[%d]=%1.1f\n",
                i,h_a[i],i,h_b[i],i,h_c[i]);
    }

/*Freeing memory*/
    delete h_a;
    delete h_b;
    delete h_c;
    cudaFree(dev_a);
    cudaFree(dev_b);
    cudaFree(dev_c);

return (EXIT_SUCCESS);
}

/*Kernel*/
__global__ void KernelName(float *dev_a,float *dev_b,float *dev_c, int
ndim){

    int index = blockIdx.x * blockDim.x + threadIdx.x;

    if(index < ndim){
        dev_c[index] = dev_a[index] + dev_b[index];
    }

 __syncthreads();
}
```

7 Appendix B

In this appendix it is shown the code transcript of the CUFFT library usage described in example 2 of section 3.6.5.

```
/*
 * File:   newmain.cu
 * Example of using the CUFFT library
 */

#include <iostream>
#include "exFFT.h"

using namespace std;

/* Kernel prototype*/
__global__ void KernelName(float *dev_a,float *dev_b,cuFloatComplex *dev_c,
        int nx, int ny, int nz);

int main(int argc, char** argv) {

/*Creating the variables on the host*/

    int nx=400;
    int ny=30;
    int nz=2;
    int ndim=nx*ny*nz;
    float *h_a;
    float *h_b;
    cuFloatComplex *h_c;
    cuFloatComplex *ffth_c;
    cuFloatComplex *Iffth_c;
    h_a = new float[ndim];
    memset(h_a,0,sizeof(float)*ndim);
    h_b = new float[ndim];
    memset(h_b,0,sizeof(float)*ndim);
    h_c = new cuFloatComplex[2*nx*2*ny*2*nz];
    memset(h_c,0,sizeof(cuFloatComplex)*2*nx*2*ny*2*nz);
    ffth_c = new cuFloatComplex[2*nx*2*ny*2*nz];
    memset(ffth_c,0,sizeof(cuFloatComplex)*2*nx*2*ny*2*nz);
    Iffth_c = new cuFloatComplex[2*nx*2*ny*2*nz];
    memset(Iffth_c,0,sizeof(cuFloatComplex)*2*nx*2*ny*2*nz);

/*Creating the variables on the device*/
    float *dev_a;
    float *dev_b;
    cuFloatComplex *dev_c;
    cuFloatComplex *fftdev_c;
    cuFloatComplex *Ifftdev_c;
    cudaMalloc((void**)&dev_a,sizeof(float)*ndim);
    cudaMemset(dev_a,0,sizeof(float)*ndim);
    cudaMalloc((void**)&dev_b,sizeof(float)*ndim);
    cudaMemset(dev_b,0,sizeof(float)*ndim);
    cudaMalloc((void**)&dev_c,sizeof(cuFloatComplex)*2*nx*2*ny*2*nz);
    cudaMemset(dev_c,0,sizeof(cuFloatComplex)*2*nx*2*ny*2*nz);
    cudaMalloc((void**)&fftdev_c,sizeof(cuFloatComplex)*2*2*nx*2*ny*2*nz);
    cudaMemset(fftdev_c,0,sizeof(cuFloatComplex)*2*2*nx*2*ny*2*nz);
    cudaMalloc((void**)&Ifftdev_c,sizeof(cuFloatComplex)*2*2*nx*2*ny*2*nz);
    cudaMemset(Ifftdev_c,0,sizeof(cuFloatComplex)*2*2*nx*2*ny*2*nz);

/*Defining the size of the block and grid*/
    dim3 dimBlock(512);
    dim3 dimGrid(ndim/dimBlock.x + (ndim%dimBlock.x==0?0:1));

    printf("block %d   Blocks in grid %d \n",dimBlock.x,dimGrid.x);

/*Initializing the vectors*/
    for(int i=0 ;i<ndim; i++){
        h_a[i]= 1+i;
        h_b[i]= 3;
    }

    /*Copying the values of a and b to the GPU device variables dev_a and
dev_b*/
    cudaMemcpy(dev_a, h_a, sizeof(float)*ndim, cudaMemcpyHostToDevice);
    cudaMemcpy(dev_b, h_b, (sizeof(float)*ndim), cudaMemcpyHostToDevice);

/* Kernel call*/
    KernelName<<<dimGrid,dimBlock>>>(dev_a,dev_b,dev_c,nx,ny,nz);

    cuFFT3DF (2*nx,2*ny,2*nz, dev_c, fftdev_c);
    cuFFT3DI (2*nx,2*ny,2*nz, fftdev_c, Ifftdev_c);

/*Returning the result of the kernel operation to the host*/
    cudaMemcpy(h_c,dev_c,sizeof(cuFloatComplex)*2*nx*2*ny*2*nz,
cudaMemcpyDeviceToHost);
    cudaMemcpy(ffth_c,fftdev_c,sizeof(cuFloatComplex)*2*nx*2*ny*2*nz,
cudaMemcpyDeviceToHost);
    cudaMemcpy(Iffth_c,Ifftdev_c,sizeof(cuFloatComplex)*2*nx*2*ny*2*nz,
cudaMemcpyDeviceToHost);

/*Freeing memory*/
    delete h_a;
    delete h_b;
    delete h_c;
    delete ffth_c;
    delete Iffth_c;
    cudaFree(dev_a);
    cudaFree(dev_b);
    cudaFree(dev_c);
    cudaFree(fftdev_c);
    cudaFree(Ifftdev_c);

return (EXIT_SUCCESS);
}

/*Kernel*/
__global__ void KernelName(float *dev_a, float *dev_b,cuFloatComplex
*dev_c, int nx, int ny, int nz){

    int index = blockIdx.x * blockDim.x + threadIdx.x;

    if(index < (2*nx*2*ny*2*nz)){
        if(index < (nx*ny*nz)){
          dev_c[index] = make_cuFloatComplex(dev_a[index] + dev_b[index],
0.0e0);
```

```
        }else{
          dev_c[index] = make_cuFloatComplex(0.0e0, 0.0e0);
        }
    }
 __syncthreads();
}


/*
 * File: exFFT.cu
 * Example of using the CUFFT library
 * File containing the CUFFT functions plan
 *
 */

#include "exFFT.h"

void cuFFT3DF (int xSize, int ySize, int zSize, cuFloatComplex *inArray,
        cuFloatComplex *outArray){
// Define the handle variable that allows access to CUFFT plans
    cufftHandle plan;

// Create a 1-dimensional CUFFT plan of a float Complex to float Complex
//variables
    cufftPlan3d(&plan, xSize, ySize, zSize, CUFFT_C2C);

// Transform the data to the same array in the forward direction
    cufftExecC2C(plan, inArray, outArray, CUFFT_FORWARD);

// Destroy the CUFFT plan
    cufftDestroy(plan);
}

void cuFFT3DI (int xSize, int ySize, int zSize, cuFloatComplex *inArray,
        cuFloatComplex *outArray){
// Define the handle variable that allows access to CUFFT plans
    cufftHandle plan;

// Create a 1-dimensional CUFFT plan of a float Complex to float Complex
//variables
    cufftPlan3d(&plan, xSize, ySize, zSize, CUFFT_C2C);

// Transform the data to the same array in the forward direction
    cufftExecC2C(plan, inArray, outArray, CUFFT_INVERSE);

// Destroy the CUFFT plan
    cufftDestroy(plan);
}


/*
 * File:   exFFT.h
 * Example of using the CUFFT library
 * Header file containing the CUFFT function prototypes
 */

#ifndef EXFFT_H
```

```
#define      EXFFT_H

#include <cufft.h>      // CUDA library to calculate FFT

void cuFFT3DF (int xSize, int ySize, int zSize, cuFloatComplex *inArray,
        cuFloatComplex *outArray);

void cuFFT3DI (int xSize, int ySize, int zSize, cuFloatComplex *inArray,
        cuFloatComplex *outArray);

#endif /* EXFFT_H */
```

8 Bibliography

[1] M. N. Baibich, J. M. Broto, A. Fert, F. Nguyen-Van-Dan, F. Petroff, P. Etienne, G. Creuzet, A. Friedrich and J. Chazelas, *Phys. Rev. Lett.* **61**, 2472 (1988).

[2] G. Binasch, P. Grünberg, F. Saurenbach and W. Zinn, *Phys. Rev. B* **39**, 4828 (1989).

[3] J. Slonczewski, *J. Magn. Magn. Mater.*, **159**, L1-L7, (1996).

[4] L. Berger, *Phys. Rev. B*, **54**, no. 13, pp. 9353-9358, (1996).

[5] www.everspin.com; ST-MRAM_Technical_Brief

[6] S.S.P. Parkin, M. Hayashi and L. Thomas, *Science* **320**, 190, (2008)

[7] A. Thiaville and Y. Nakatani, 2006, "Domain-Wall Dynamics in Nanowires and Nanostrips, "*Spin dynamics in Confined Magnetic Structures III*", ed. B. Hillebrands and A Thiaville (Berlin:Springer).

[8] A. Thiaville, J.M. Garcia and J. Miltat, "Domain wall dynamics in nanowires", *J. Magn. Magn. Mater*, Vol. **242–245**, 1061–1063, Apr. 2002.

[9] M. Yan, A. Kákay, S. Gliga and R. Hertel, "Beating the Walker Limit with Massless Domain Walls in Cylindrical Nanowires", *Phys. Rev. Lett.*, **104**, 057201, Feb. 2010.

[10] V.E. Demidov, S. Urazhdin, S. Demokritov, *Nat. Mat*, Vol. **9**, 984-988, (2010).

[11] S. I. Kiselev, J. C. Sankey, I. N. Krivorotov, N. C. Emley, R. J. Schoelkopf, R. A. Buhrman, and D. C. Ralph, *Nature*, **425**, 380-383, (2003).

[12] W. H. Rippard, M. R. Pufall, S. Kaka, S. E. Russek, and T. J. Silva, *Phys. Rev. Lett.,* **92**, 146803, (2004).

[13] Kajiwara Y., Harii K., Takahashi S., Ohe J., Uchida K., Mizuguchi M., Umezawa H., Kawai H., Ando K., Takanashi K., Maekawa S., Saitoh E., *Nat. Let.* **464**, 262-266 (2010).

[14] K. Uchida, S. Takahashi, K. Harii, J. Ieda, W. Koshibae, K. Ando, S. Maekawa, and E. Saitoh, *Nature* **455**, 778-781 (2008).

[15] C.M. Jaworski, R.C. Myers, E. Johnston-Haperin and J.P. Heremans, *Nature*, **487**, 210–213, (2012).

[16] G. Bertotti, "Hysteresis in Magnetism - For Physicists, Materials Scientists and Engineers", Academic Press 1998.

[17] B. Hillebrands, A. Thiaville (Eds.), "Spin Dynamics in Confined Magnetic Structures III", 101, Topics in Applied Physics, Springer:Berlin 2006

[18] B. Hillebrands, K. Ounadjela (Eds.), "Spin Dynamics in Confined Magnetic Structures I", 83m Topics in Applied Physics, Springer:Berlin 2002

[19] D. Stancil, A. Prabhakar, "Spin Waves - Theory and Applications", Springer 2009.

[20] Teruya Shinjo (edited), "Nanomagnetism and Spintronics", Elsevier B.V., 2009.

[21] E. Martinez, l. Lopez-Diaz, L. Torres and C.J. Garcia-Cervera, *J. Phys. D: Appl. Phys,* **40**, 942-948, (2007).

[22] H. Goldstein, “Mecánica Clásica”, Ed. Reverté (1990).

[23] W.F. Brown Jr., “Micromagnetics”. Ed. John Wiley and Sons (1978).

[24] B.D. Cullity, “Introduction to Magnetic Materials”, Ed. Addison-Wesley (1972).

[25] H.M. Rosenberg, “El estado Sólido”, Ed. Alianza Editorial (1991).

[26] Kittel, “Introducción a la Física del Estado Sólido”, Ed. Reverte (1993).

[27] Coffey W T, Kalmykov Y P and Waldrom J T, “The Langevin Equation, with applications to stochastic problems in Physics, Chemistry and Electrical Engineering”, Singapore: World Scientific, (1996).

[28] W.F. Brown Jr., *Phys. Rev. B*, **130,** 1677–86, (1963).

[29] N.G. Van Kampen, “Stochastic Processes in Physics and Chemistry”, Amsterdam: North-Holland, (1981).

[30] J. Slonczewski, *J. Magn. Magn. Mater*. **247,** 324, (2002).

[31] J. Slonczewski, *Phys. Rev B*, **71**, 024411, (2005).

[32] A. A. Awad, A. Lara, V. Metlushko, K. Y. Guslienko, and F. G. Aliev “Broadband probing magnetization dynamics of the coupled vortex state permalloy layers in nanopillars” *Appl. Phys. Lett*. vol. **100**, 26240, (2012)

[33] S. Zhang and Z. Li, *Phys. Rev. Lett.* **93,** 127204, (2004)

[34] A . Thiaville, Y. Nakatani, J. Miltat and Y. Suzuki, *Europhys. Lett.* **69,** 990–6, (2005)

[35] Martinez E., Lopez-Diaz L., Alejos O. and Torres L., *Phys. Rev. B*, **77** 144417, (2008).

[36] O. Boulle, G. Malinowski, M Klaui, *Mater. Sci. Eng. R*, (2011), doi:10.1016/j.mser.2011.04.001

[37] A. Aharoni, “Introduction to the theory of ferromagnetism”, Claredon Press: Oxford, (1996).

[38] C. Johnson, “Numerical Solution of Partial Differential Equations by the Finite Element Method”, New York: Cambridge University Press, (1990).

[39] C.J. García-Cervera and E. Weinan, *IEEE Trans. Magn.,* **39,** 1766–70, (2003).

[40] C.J. Parker, C. Cerjan and D.W. Hewett, *J. Magn. Magn. Mater.,* **214,** 130–8, (2000).

[41] M.J. Donahue and R.D. McMichael, *IEEE Trans. Magn.* **43,** 2878–80, (2007).

[42] A. Thiaville and Y. Nakatani, 2006, “Domain-Wall Dynamics in Nanowires and Nanostrips, (Spin dynamics in Confined Magnetic Structures III)”, ed. B. Hillebrands and A Thiaville (Berlin:Springer).

[43] J. Miltat and M. Donahue, “Numerical Micromagnetics: Finite Difference Methods (Handbook of Magnetism and Advanced Magnetic Materials)”, vol 2 ed. H. Kronmuller and S. Parkin, Chichester: Wiley-Interscience, (2007).

[44] M. Labrune and J. Miltat, *J. Magn. Magn. Mater.,* **151,** 231–45, (1995).

[45] A.J. Newell, W. Williams and D.J. Dunlop, *J. Geophys. Res.,* **98**, 9551–5, (1993).

[46] J. Miltat, M. Donahue, "Numerical Micromagnetics: Finite Difference Methods", Handbook of Magnetism and Advanced Magnetic Materials, edited by H. Kronmüller and S. Parkin, Wiley-Interscience, Chichester, Vol2, pp 742-764, (Sep 2007).

[47] D.J. Stockhan, *Joint. Comput. Conf. Proc.,* **28,** 229–33, (1966).

[48] M. Mansuripur and R. Giles, *IEEE Trans. Magn.,* **24,** 2326–8, (1998).

[49] E. Martinez, L. Torres, *Member*, *IEEE,* and L. Lopez-Diaz, *IEEE Trans. Magn.*, **40**, 5, (2004)

[50] Press W. H., Teukolky S. A., Watterling W. T. and Flannery B. P. "Numerical Recipes in Fortran: The Art of Scientific Computing", New York: Cambridge University Press, (1996).

[51] Berkov D. V. and Gorn N. L., *J. Magn. Magn. Mater.,* **290,** 442–8, (2005).

[52] Berkov D. V., "Magnetization Dynamics Including Thermal Fluctuations: Basic Phenomenology, Fast Remagnetization Processes and Transitions Over High-energy Barriers (Handbook of Magnetism and Advanced Magnetic Materials)", vol 2, ed H. Kronmuller and S. Parkin, Chichester: Wiley-Interscience, (2007).

[53] Grinstein G. and Koch R .H., *Phys. Rev. Lett.,* **90,** 207201, (2003).

[54] Martinez E., Lopez-Diaz L. and Torres L., *IEEE Trans. Magn.,* **39,** 2522–4, (2003).

[55] Tsiantos V., Scholz W., Suess D., Schrefl T. and Fidler J., *J. Magn. Magn. Mater.,* **242–245** 999–1001, 2002

[56] L. Lopez-Diaz, D. Aurelio, L. Torres, E. Martinez, M. A. Hernandez-Lopez, J. Gomez, O. Alejos, M. Carpentieri, G. Finocchio. and G. Consolo, *J. Phys. D: Appl. Phys.*, **45,** 323001 (17pp), (2012).

[57] D.W. Zingg, T.T. Chisholm, *Applied Numerical Mathematics,* **31,** 227–238, (1999)

[58] NVIDIA Corporation http://www.nvidia.com

[59] What is CUDA?, http://www.nvidia.co.uk/object/what_is_cuda_new_uk.html

[60] CUDA in action, http://www.nvidia.co.uk/object/cuda_in_action_uk.html

[61] CUDA C Programming Guide, http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html

[62] NVIDIA CUDA runtime API, http://docs.nvidia.com/cuda/cuda-runtime-api/index.html

[63] CUDA C Best Practices Guide http://docs.nvidia.com/cuda/cuda-c-best-practices-guide/index.html

[64] CUFFT library http://docs.nvidia.com/cuda/cufft/index.html

[65] CUDPP library http://code.google.com/p/cudpp/

[66] http://gpgpu.org/developer/cudpp#high_1

[67] Donahue M. J. 2007, *6th International Symposium on Hysteresis Modeling and Micromagnetics, HMM-2007* (presentation available at http://math.nist.gov/MDonahue/talks.html).

[68] http://math.nist.gov/oommf/

[69] http://www.ctcms.nist.gov/~rdm/mumag.org.html

[70] http://www.goparallel.net/

[71] D. Aurelio, L. Torres and G. Finocchio, *J. Magn. Mag. Mater.* **321**, 3913–3920, (2009)

[72] D. Aurélio, L. Torres and G. Finocchio, *J. Appl. Phys.,* **108**, 083911, (2010)

[73] G. Finocchio, M. Carpentieri, B. Azzerboni, L. Torres, E. Martinez, L. Lopez-Diaz, *J. Appl. Phys.* **99**, 08G522, (2006)

[74] G. Finocchio, M. Carpentieri, B. Azzerboni, L. Torres, L. Lopez-Diaz, E. Martinez, *Physica B* **372**, 294, (2006)

[75] D. Apalkov, M. Pakala, Y. Huai, *J. Appl. Phys.* **99**, 08B907, (2006)

[76] J.C. Sankey, Y. -Tao Cui, J.Z. Sun, J.C. Slonczewski, R.A. Buhrman, D.C. Ralph, *Nat. Phys.* **4,** 67, (2008)

[77] T. Devolder, J. Hayakawa, K. Ito, H. Takahashi, S. Ikeda, P. Crozat, N. Zerounian, Joo-Von Kim, C. Chappert, H. Ohno, *Phys. Rev. Lett.* **100**, 057206, (2008)

[78] T. Devolder, J. Hayakawa, K. Ito, H. Takahashi, S. Ikeda, J.A. Katine, M.J. Carey, P. Crozat, J.V. Kim, C. Chappert, H. Ohno, *J. Appl. Phys.* **103**, 07A723, (2008)

[79] M. Iwayama, T. Kai, M. Nakayama, H. Aikawa, Y. Asao, T. Kajiyama, S. Ikegawa, H. Yoda, A. Nitayama, *J. Appl. Phys.* **103**, 07A720, (2008)

[80] http://www.infolytica.com/en/products/magnet/

[81] G. Finocchio, B. Azzerboni, G.D. Fuchs, R.A. Buhrman, L. Torres, *J. Appl. Phys.* **101**, 063914, (2007)

[82] G. Finocchio, G. Consolo, M. Carpentieri, A. Romeo, B. Azzerboni, L. Torres, *J. Appl. Phys*., **101**, 09A508, (2007)

[83] R.D. McMichael, M.D. Stiles, *J. Appl. Phys.* **97**, 10J901, (2005)

[84] G. Finocchio, I. Krivorotov, M. Carpentieri, G. Consolo, B. Azzerboni, L. Torres, E. Martinez, L. Lopez-Diaz, *J. Appl. Phys.* **99**, 08G507, (2006)

[85] L. Torres, L. Lopez-Diaz, E. Martinez, G. Finocchio, M. Carpentieri, B. Azzerboni, *J. Appl. Phys.* **101**, 053914, (2007)

[86] Y. Acremann, J.P. Strachan, V. Chembrolu, S.D. Andrews, T. Tyliszczak, J.A. Katine, M.J. Carey, B.M. Clemens, H.C. Siegmann, J. Stohr*, Phys. Rev. Lett.* **96**, 217202, (2006)

[87] K. Ito, T. Devolder, C. Chappert, M.J. Carey, J.A. Katine, *J. Phys. D: Appl. Phys.* **40**, 1261–1267, (2007)

[88] Y. Saito, T. Inokuchi, H. Sugiyama, K. Inomata, *Eur. Phys. J. B* **59**, 463–469, (2007)

[89] A.M. Deac, A. Fukushima, H. Kubota, H. Maehara, Y. Suzuki, S. Yuasa, Y. Nagamine, K. Tsunekawa, D.D. Djayaprawira, N. Watanabe, *Nat. Phys.* **4**, 803, (2008)

[90] H. Tomita, K. Konishi, T. Nozaki, H. Kubota, A. Fukushima, K. Yakushiji, S. Yuasa, Y. Nakatani, T. Shinjo, M. Shiraishi, and Y. Suzuki, *Appl. Phys. Express* **1**, 061303, (2008).

[91] Y. Cui, G. Finocchio, C. Wang, J. A. Katine, R. A. Buhrman, and D. C. Ralph, *Phys. Rev. Lett.* **104**, 097201, (2010).

[92] S. Garzon, Y. Bazaliy, R. A. Webb, M. Covington, S. Kaka, and T. M. Crawford, *Phys. Rev. B* **79**, 100402, (2009).

[93] G. D. Fuchs, I. N. Krivorotov, P. M. Braganca, N. C. Emley, A. G. F. Garcia, D. C. Ralph, and R. A. Buhrman, *Appl. Phys. Lett.* **86**, 152509, (2005).

[94] N. C. Emley, I. N. Krivorotov, O. Ozatay, A. G. F. Garcia, J. C. Sankey, D. C. Ralph, and R. A. Buhrman, *Phys. Rev. Lett*. **96**, 247204, (2006).

[95] C. Wang, Y.T. Cui, J. Z. Sun, J. A. Katine, R. A. Buhrman, and D. C. Ralph, *Phys. Rev. B* **79**, 224416, (2009).

[96] M. H. Jung, S. Park, C.Y. You, and S. Yuasa, *Phys. Rev. B* **81**, 134419, (2010).

[97] G. Finocchio, V.S. Pribiag, L. Torres, R. A. Buhrman and B. Azzerboni. "Spin-torque driven magnetic vortex self-oscillations in perpendicular magnetic fields" *Appl. Phys. Lett*. vol. **96**, 102508, (2010).

[98] V. S. Pribiag, I. N. Krivorotov, G. D. Fuchs, P.M. Braganza, O. Ozatay, J. C. Sankey, D. C. Ralph and R. A. Buhrman. "Magnetic vortex oscillator driven by d.c. spin-polarized current*" Nat. Phys.* vol. **3**, pp. 498-503, (2007)

[99] M. Carpentieri, L. Torres and E. Martinez. "Temperature Dependence of Microwave Nano-Oscillator Linewidths Driven by Spin-Polarized Currents: A Micromagnetic Analysis". *IEEE Trans. Mag*. vol **45** (10), pp. 3426-3429, (2009)

[100] J. C. Sankey, I. N. Krivorotov, S. I. Kiselev, P. M. Braganca, N. C. Emley, R. A. Buhrman, and D. C. Ralph, "Mechanisms limiting the coherence time of spontaneous magnetic oscillations driven by dc spin- polarized currents," *Phys. Rev. B*, *Condens. Matter,* vol. **72**, 224427, (2005)

[101] J. Kim, Q. Mistral, C. Chappert, V. S. Tiberkevich, and A. N. Slavin, "Lineshape distortion in a nonlinear auto-oscillator near generation threshold: Application to spin-torque nano-oscillators," *Phys. Rev. Lett.,* vol. **100**, 167201, (2008)

[102] J. Kim, V. S. Tiberkevich, and A. N. Slavin, "Generation linewidth of an auto-oscillator with a nonlinear frequency shift: Spin-torque nanooscillators*," Phys. Rev. Lett.*, vol. **100,** 017207, (2008)

[103] V. S. Tiberkevich, A. N. Slavin, and J. Kim, "Temperature dependence of nonlinear auto-oscillator linewidths: Application to spin-torque nano- oscillators," *Phys. Rev. B, Condens. Matter*, vol. **78**, 092401, (2008)

[104] C. Boone, J. A. Katine, J. R. Childress, J. Zhu, X. Cheng, and I. N. Krivorotov, "Experimental test of an analytic theory of spin-torque oscillator dynamics*" Phys. Rev. B, Condens. Matter*, vol. **97**, 140404(R), (2009)

[105] M. L. Schneider, W. H. Rippard, M. R. Pufall, T. Cecil, T. J. Silva, and S. E. Russek "Temperature dependence of spin-torque-driven self- oscillations" Phys. *Rev. B, Condens. Matter*, vol. **80**, 144412, (2009)

[106] P. Bortolotti, A. Dussaux, J. Grollier, V. Cros, A. Fukushima, H. Kubota, K. Yakushiji, S. Yuasa, K. Ando, and A. Fert "Temperature dependence of microwave voltage emission associated to spin-transfer induced vortex oscillation in magnetic tunnel junction" *Appl. Phys. Lett*. vol. **100**, 042408, (2012)

[107] P. K. Muduli, O. G. Heinonen, and Johan A° kerman "Temperature dependence of linewidth in nanocontact based spin torque oscillators: Effect of multiple oscillatory modes" *Phys. Rev. B, Condens. Matter* vol. **86**, 174408, (2012)

[108] V. S. Pribiag, G. Finocchio, B. J. Williams, D. C. Ralph, R. A. Buhrman, "Long timescale fluctuations in zero-field magnetic vortex oscillations driven by dc spin-polarized current", *Phys. Rev. B, Condens. Matter* vol. **80**, 180411(R), (2009)

[109] M. Quinsat, V. Tiberkevich, D. Gusakova, A. Slavin, J. F. Sierra, U. Ebels, L. D. Buda-Prejbeanu, B. Dieny, M.-C. Cyrille, A. Zelster, and J. A. Katine. "Linewidth of higher harmonics in a nonisochronous auto- oscillator: Application to spin-torque nano-oscillators" *Phys. Rev. B, Condens. Matter,* vol. **86**, 104418, (2012)

[110] J.H. Ai, B.F. Miao, L. Sun, B. You, An Hu, and H.F. Ding, "Current-induced domain wall motion in permalloy nanowires witha rectangular cross-section", *J. Appl. Phys.* Vol. **110**, 093913, (Nov. 2011).

[111] W. Döring, *Z. Naturforsh*, **3a**, 373, (1948)

[112] M. Kläui, *J. Phys: Condens. Matt.*, **20**, 313001, (2008)

[113] IEEE Transactions on Magnetics journal webpage: http://ieeexplore.ieee.org/xpl/RecentIssue.jsp?punumber=20

[114] E. Martinez, L. Lopez-Diaz, L. Torres, C. Tristan and O. Alejos, *Phys. Rev. B* **75**, 174409 (2007)