TESIS DOCTORAL

Aplicación de Computación Evolutiva y Paralelismo para la Resolución de Problemas de Astrofísica

Application of Evolutionary Computation and Parallelism for Solving Problems of Astrophysics

Miguel Cárdenas Montes

Departamento de Tecnología de los Computadores y de las Comunicaciones

Conformidad del Director:

Dr. Miguel Ángel Vega Rodríguez

A thesis submitted for the degree of

PhilosophiæDoctor (PhD)

Cáceres, 2014

Except where acknowledged in the customary manner, the material presented in this thesis is, to the best of my knowledge, original and has not been submitted in whole or part for a degree in any university.

Miguel Cárdenas Montes

_____

A mi familia.

The Cosmos is all that is or ever was or ever will be.

Carl Edward Sagan

Art. 13. El objeto del Gobierno es la felicidad de la Nación, puesto que el fin de toda sociedad política no es otro que el bienestar de los individuos que la componen.
CONSTITUCIÓN DE CÁDIZ DE 1812

Citations to Previously Published Works

The Chapter 3 has been constructed based, among others, on the publications:

> Cárdenas-Montes, Miguel, Vega-Rodríguez, Miguel A., and Gómez-Iglesias, Antonio: **Sensitiveness of Evolutionary Algorithms to the Random Number Generator**, ICANNGA (1), LNCS, Springer, 371-380, Eds: Dobnikar, Andrej, Lotric, Uros, and Ster, Branko, 2011
>
> Cárdenas-Montes, Miguel, Vega-Rodríguez, Miguel A., Gómez-Iglesias, Antonio, and Morales-Ramos, Enrique: **Exploration of the Conjecture of Bateman using Particle Swarm Optimization and Grid Computing**, 8th International Symposium on Parallel and Distributed Computing, ISPDC, IEEE Computer Society, 143-150, 2009
>
> Cárdenas-Montes, Miguel, Vega-Rodríguez, Miguel A., García Orellana, Carlos J., Rubio del Solar, Manuel, Gómez Pulido, Juan Antonio, González Velasco, Horacio M., Gómez-Iglesias, Antonio, Sánchez-Pérez, Juan Manuel, and Macías Macías, Miguel: **Volunteer Computing, an Interesting Option for Grid Computing: Extremadura as Case Study**, OTM Workshops (1), LNCS, Springer, 29-30, Eds: Meersman, Robert, Tari, Zahir, and Herrero, Pilar, 2007

Sections of the Chapter 4 can be also found in:

> Cárdenas-Montes, Miguel, Vega-Rodríguez, Miguel A.: **Effect of data layout in the evaluation time of non-separable functions on GPU**, Computing and Informatics, 1-21, ISSN: 1335-9150, 2015, Accepted (Impact Factor = 0,254 en 2012)
>
> Cárdenas-Montes, Miguel, Vega-Rodríguez, Miguel A., Rodríguez-Vázquez, Juan José, and Gómez-Iglesias, Antonio: **Effect of the Block Occupancy in GPGPU over the Performance of Particle Swarm Algorithm**, ICANNGA (1), LNCS, Springer, 310-319, Eds: Dobnikar, Andrej, Lotric, Uros, and Ster, Branko, 2011

Cárdenas-Montes, Miguel, Vega-Rodríguez, Miguel A., Rodríguez-Vázquez, Juan José, and Gómez-Iglesias, Antonio: **GPU-Based Evaluation to Accelerate Particle Swarm Algorithm**, EUROCAST (1), LNCS, Springer, 272-279, Eds: Moreno-Díaz, Roberto, Pichler, Franz, and Quesada-Arencibia, Alexis, 2011

Cárdenas-Montes, Miguel, Vega-Rodríguez, Miguel A., Rodríguez-Vázquez, Juan Jose, and Gómez-Iglesias, Antonio: **Accelerating Particle Swarm Algorithm with GPGPU**, 16th Euromicro International Conference on Parallel, Distributed and Network-Based Processing, PDP, IEEE Computer Society, 560-564, Eds: Cotronis, Yiannis, Danelutto, Marco, and Papadopoulos, George Angelos, 2011

Some parts of the Chapter 5 have been also published in:

Cárdenas-Montes, Miguel, Vega-Rodríguez, Miguel A., and Mollá, Mercedes: **Metaheuristics for Modelling Low-Resolution Galaxy Spectral Energy Distribution**, HAIS, LNAI, Springer, 490-501, Eds: Polycarpou, Marios M. et al., 2014

Cárdenas-Montes, Miguel, Vega-Rodríguez, Miguel A., and Mollá, Mercedes: **Metaoptimization of Differential Evolution by Using Productions of Low-Number of Cycles: The Fitting of Rotation Curves of Spiral Galaxies as Case Study**, HAIS, LNAI, Springer, 356-365, Eds: Pan, Jeng-Shyang, Polycarpou, Marios M., Wozniak, Michal, Carvalho, André C. P. L. F., Quintián-Pardo, Héctor, and Corchado, Emilio, 2013

Cárdenas-Montes, Miguel, Vega-Rodríguez, Miguel A., and Gómez-Iglesias, Antonio: **Real-World Problem for Checking the Sensitiveness of Evolutionary Algorithms to the Choice of the Random Number Generator**, HAIS (1), LNAI, Springer, 385-396, Eds: Corchado, Emilio, Snásel, Václav, Abraham, Ajith, Wozniak, Michal, Graña, Manuel, and Cho, Sung-Bae, 2012

Large portions of Chapter 6 have appeared in the following papers:

Cárdenas-Montes, Miguel, Vega-Rodríguez, Miguel A., Bonnett, Christopher, Sevilla-Noarbe, Ignacio, Ponce, Rafael, Sánchez Álvaro, Eusebio, Rodríguez-Vázquez, Juan José: **GPU-Based Shear-Shear Correlation Calculation**, Computer Physics Communications, 185(1):11-

18, ISSN: 0010-4655, 2014 (Impact Factor = 3,078 en 2012, Quartile Q1)

Cárdenas-Montes, Miguel, Rodríguez-Vázquez, Juan José, Vega-Rodríguez, Miguel A., Sevilla-Noarbe, Ignacio, Sánchez Álvaro, Eusebio: **Performance and precision of the histogram calculation on GPU: cosmological analysis as case study**, Computer Physics Communications, 1-27, ISSN: 0010-4655, Accepted (Impact Factor = 3,078 en 2012, Quartile Q1)

Cárdenas-Montes, Miguel, Vega-Rodríguez, Miguel A., Sevilla, Ignacio, Ponce, Rafael, Rodríguez-Vázquez, Juan José, Sánchez Álvaro, Eusebio: **Concurrent CPU-GPU Code Optimization: The Two-Point Angular Correlation Function as Case Study**, CAEPIA, LNAI, Springer, 209-218, Eds: Bielza, C.; Salmeron, A.; Alonso-Betanzos, A.; Hidalgo, J.I.; Martínez, L.; Troncoso, A.; Corchado, E.; Corchado, J.M., 2013

Cárdenas-Montes, Miguel, Rodríguez-Vázquez, Juan José, Sevilla, Ignacio, Sánchez Álvaro, Eusebio, Ponce, Rafael, Vega-Rodríguez, Miguel A., Bonnett, Christopher: **High-Performance Implementations for Shear-Shear Correlation Calculation**. Cluster, IEEE Computer Society, 2014. Submitted.

Acknowledgements

I warrant that I have obtained, where necessary, permission from the copyright owners to use any third-party copyright material reproduced in the Thesis, or to use any of my own published work in which the copyright is held by another party.

Agradecimientos

Los trabajos que requiere una tesis suponen un desafío para la persona que acomete la tarea, así como para su entorno familiar. Por ello, es necesario reconocer el esfuerzo que realiza el entorno familiar para soportar, sí éste es el verbo adecuado, soportar los cambios en el estado de ánimo del doctorando. Por ello el primer agradecimiento es para mi familia: mi mujer Irene, mi hija Ana, y mis padres Miguel y Manoli. Ellos han creado las condiciones de contorno adecuadas para la consecución con éxito de esta misión.

La relación que se establece entre un director de tesis y su doctorando guarda cierto parecido con una relación de noviazgo. En ella, para que la relación llegue a buen puerto es necesario que ambas personas sean compatibles. Sin Miguel Ángel Vega esta tesis no hubiera sido posible. Miguel Ángel ha sabido adaptarse a mis peculiaridades, permitiéndome desarrollar el trabajo en función de mis intereses y del tiempo disponible.

El tercer apoyo sin el cual esta tesis no habría sido posible es mi exjefe Nicanor Colino. Él me rescató, justo cuando nadie daba un duro por mí. Además, vio mis muy pequeñas virtudes frente a mis grandes defectos.

En estos años de trabajo en CIEMAT ha habido periodos más felices y otros no tanto. Durante uno de estos periodos oscuros, mi amigo Antonio Gómez se mantuvo a mi lado. Sin importarle las consecuencias, Antonio supo cual era la postura moral correcta, y no hay suficientes palabras en este escrito para agradecerle su actitud. Durante el tiempo compartido con Antonio he podido disfrutar de su brillantez, inteligencia, humor y fina ironía.

En estos años en CIEMAT he compartido una gran parte de mi trabajo, comidas y muchos cafés con mi amigo Juan José Rodríguez. Nuestras conversaciones son interminables, abarcando todos los ámbitos: política, educación, ciencia, informática, GPUs, etc. Todas enriquecedoras. Aunque es una palabra en desuso, en nuestra provincial natal se utiliza muy a menudo, Juanjo es una personal cabal como pocas.

Finalmente, agradecer a todos aquellos con los que he compartido grandes ratos, conversaciones y viajes, Francisco Castejón de quien copié su lema de 6 puntos para obtener buenos resultados profesionales: *trabajo, trabajo, trabajo, calidad, calidad y calidad*, a Antonio Delgado, Eusebio Sánchez, Nacho Sevilla, Rafa Ponce, Jorge Berenguer, Mercedes Mollá, Concha Braña, Manuel Aguilar, Chema y Ricardo.

Abstract

The scientific activity is destined to face the challenges of the epoch in which it develops. Nowadays the society, and therefore the science, face the challenge of adequately handling volumes of information without precedent in the history of the humankind.

In the area of astronomy, astrophysics and cosmology, the appearance of these enormous volumes of high-quality information has taken place as a consequence of the digitalization of the facilities. This increment is forcing to the practitioners to face new challenges associated with the management and the analysis of these data. This fact, still far from being temporary, will aggravate in the near future due to the improvements in the spatial and time resolution of the instruments. For this reason, in the next years this scientific area will need additional computational efforts to provide to the scientists of the necessary tools for the accomplishment of a scientific production of high quality.

Concerning the data analysis, both the parallelism techniques, to improve the processing speed; as well as the technologies related to evolutionary computation or data mining, to extract knowledge of large volumes of information are widely applicable to problems in the field of astronomy, astrophysics and cosmology.

In the area of parallel computing, the new developments must mitigate the penalty of the large execution times associated with the analysis of data sets continuously growing. Particularly interesting it is the case of the correlation functions used in cosmology. These functions have proved to be extremely useful for the study of the large-scale structure of the Universe. Nevertheless, their computational complexity, $O(N^m)$ where N it is the number of points of the sample and m the order of the correlation, make them computationally intensive. Without these developments in parallel computing and taking into account the volumes of information that the scientists are going to face, the analysis of the large-scale structure of the Universe will be penalized with unacceptable execution times for the scientific activity.

In this scenario, the computing based on GPGPUs (General Purpose Computation on Graphics Processing Unit) is an effective instrument for the reduction of the execution time. It is capable of delivering a great capacity of calculation, without penalizing the budget. This is suitable taking into account the size and the dispersion of the scientific groups in this field.

In this Thesis, diverse parallel implementations of the two-point angular correlation function and the shear-shear correlation function have been developed for evaluating their efficiency in reducing the execution time.

Beside increasing its volume, the scientific information is suffering from a strong increase in its complexity. As a consequence of this fact, the extraction of synthetic knowledge from the information will need the contribution of diverse technologies. Among others, it can be emphasized the use of evolutionary algorithms and the data mining techniques.

The evolutionary algorithms are widely used in science for the search of high-quality solutions in complex problems. Nevertheless in the area of astronomy, astrophysics and cosmology the examples of use are still scanty.

In this Thesis, diverse examples associated with the use of the evolutionary algorithms in problems of astrophysics have been evaluated. Among the examples, the study of the rotation curves of galaxies and the modelling of galactic spectra with simple stellar populations can be mentioned.

Additionally, other fundamental aspects of evolutionary algorithms, but essential for their later application, such as the impact of the random number generator in the performance of the evolutionary algorithms, or the strategies of the parallelization of evolutionary algorithms have been also studied.

The application of the algorithms developed in this Thesis has produced a net improvement of the scientific productivity if compared to the previous state-of-the-art.

Resumen

La actividad científica está destinada a enfrentarse a los retos de la época en que se desarrolla. Hoy día la sociedad, y por ende la ciencia, se enfrentan al reto de manejar adecuadamente volúmenes de datos sin precedente en la historia de la humanidad.

En el área de astronomía, astrofísica y cosmología, la aparición de estos volúmenes ingentes de datos de alta calidad se ha producido como consecuencia de la digitalización de los instrumentos de observación. Este incremento está obligando a los investigadores a encarar nuevos desafíos asociados con su gestión y análisis. Este hecho, aún lejos de ser pasajero, se agravará en el futuro por las mejoras en la resolución espacial y temporal de los instrumentos. Por ello, en los próximos años este área científica requerirá esfuerzos computacionales adicionales para proveer a los científicos de las herramientas necesarias para la realización de una producción científica de alta calidad.

Con respecto al análisis de datos, tanto las técnicas de paralelismo, para mejorar la velocidad de procesamiento; como las técnicas de computación evolutiva o la minería de datos, para extraer conocimiento sintético de grandes volúmenes de datos, son ampliamente aplicables a problemas en el campo de astronomía, astrofísica y cosmología.

En el campo de computación paralela, los nuevos desarrollos deben mitigar la penalización de los grandes tiempos de ejecución asociados al análisis de ficheros cada vez mayores. Particularmente interesante es el caso de las funciones de correlaciones usadas en cosmología. Estas funciones se muestran extremadamente útiles para el estudio de la estructura del Universo en escalas grandes. Sin embargo, la complejidad computacional de las mismas, del orden $O(N^m)$ donde N es el número de puntos de la muestra y m el orden de correlación, las hace computacionalmente intensivas. Sin los desarrollos en computación paralela adecuados y teniendo en cuenta los volúmenes de datos de que van a disponer los científicos, el análisis de la estructura de Universo en escalas grandes se verá penalizado con tiempos de ejecución inasumibles para la actividad científica.

En este escenario, la computación de propósito general basada en tarjetas de procesamiento gráfico, GPGPUs, es un instrumento eficaz para la reducción del tiempo de ejecución. La computación con GPGPU es capaz de entregar una gran capacidad de cálculo, sin requerir un elevado presupuesto. Esto es adecuado teniendo en cuenta el tamaño y dispersión de los grupos de investigación en estas disciplinas científicas.

En esta Tesis se han desarrollado diversas implementaciones paralelas de la función de correlación angular de dos puntos y de la función de correlación de la distorsión de la forma de la galaxia a fin de evaluar la eficiencia de las mismas para reducir los tiempos de ejecución.

Además del incremento en volumen, los datos están sufriendo un fuerte incremento en su complejidad. Como consecuencia de este hecho, la extracción de conocimiento sintético de los datos requerirá de la aportación de numerosas técnicas. Entre éstas se puede destacar el uso de los algoritmos evolutivos o la minería de datos.

Los algoritmos evolutivos son ampliamente utilizados en ciencia para la búsqueda de soluciones de alta calidad en problemas complejos, sin embargo en el área de astronomía, astrofísica y cosmología los ejemplos son aún escasos.

En esta Tesis se han evaluado aspectos asociados al uso de los algoritmos evolutivos en problemas de astrofísica, como por ejemplo el estudio de las curvas de rotación de galaxias o la modelización de espectros galácticos con poblaciones estelares simples; así como aspectos más fundamentales, pero esenciales para su posterior aplicación, como el impacto del generador de números aleatorios en el rendimiento del algoritmo evolutivo, o las estrategias para la paralelización de los algoritmos evolutivos.

La aplicación de los algoritmos desarrollados en esta Tesis ha producido una mejora neta de la productividad científica comparada con los desarrollos previos existentes.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This thesis is organised in four parts toughly linked between them. The research is articulated around these four parts as follows:

- Studies on evolutionary algorithms.
- Studies on GPGPU (for short GPU) computing.
- Evolutionary algorithms applied to astrophysics problems.
- Application of GPU computing to astrophysics problems.

These four parts are explained in the following exposition.

1.1 Motivation

During the last decades, most of the astronomical facilities, telescopes and detectors, have been digitalized. This fact has driven to a notable improvement of the quality of the information, as well as to a spectacular increment in the volume and the complexity of available information for the researchers. This increment forces to face new challenges associated with the management and the analysis of this information volume. Far from being temporary, this effect will aggravate in the future due to the improvements in the facilities resolution (space and time).

Concerning the previously mentioned information analysis processes, both parallelism techniques, to improve the processing speed; and evolutionary computation, to extract knowledge from large information volumes are widely applicable to problems in the area of astronomy, astrophysics and cosmology. For this reason, in the next years this scientific area will need some additional computational efforts to provide to

the scientists of the necessary tools for the accomplishment of a high-quality scientific production.

Two essential considerations emerge as reasons for this deluge of data. On the one hand, new storage hardware has become high-capacity, allowing composing large installations for storage purposes. On the other hand, the cost of this hardware has reduced enough to consider keeping everything, without considering to sift the data.

The general objective of this research is to investigate the suitability of parallelism techniques and evolutionary computing, as well as, to develop and optimize the codes to support a high-quality scientific production in the area of astronomy, astrophysics and cosmology.

In cosmology, the study of the large-scale structure of the Universe has suffered from this data deluge. Until now, this kind of studies has been burden by a lack of data and instrumental errors. However, in the last years, experiments such as: Dark Energy Survey (DES), Physics of the Accelerating Universe (PAU), Kilo-Degree Survey or Euclid are going to provide large collections of data.

The large-scale structure of the Universe can be studied by using the correlation functions. It should be underlined that the computational complexity of correlation functions, $O(N^m)$, is related to the number of points in the sample, N, and the order of correlation, m. In the past, the computational complexity of these analyses has been reduced by introducing approximations in the spatial treatment of the points, i.e. kd-trees. However, this kind of approximation might carry on losses in the accuracy of the physical results.

In order to avoid a hard penalization in the processing time when calculating correlation functions, the parallelism techniques are mandatory. To accomplish this reduction, diverse technologies, such as: MPI or OpenMP, and computational resources, such as: GPGPU cards, grid and clusters, have been tested.

Beside the increment of the volume, the scientific data are acquiring a high complexity. In order to extract, the maximum of knowledge from these data sets, practitioners are using tools coming from evolutionary computing, and data mining.

In particular, evolutionary algorithms such as: genetic algorithm, particle swarm algorithm, firefly algorithm or differential evolution have been applied in this Thesis to model the rotational curves of galaxies and to fit the galactic spectra with simple stellar populations.

The rotation curve of a galaxy is the relationship between the rotational velocity of stars as function of the radial distance to the galaxy centre. Its importance stems from the discrepancy between the observed velocity of the stars and the Newtonian-Keplerian prediction, in such way that masses derived from the rotational kinematics

and gravitational laws do not match. This discrepancy might be explained by the presence of dark matter. Therefore, the characterization of rotation curve in spiral galaxies is a measure of the amount of dark matter in the galaxy.

The modelling of the galactic spectra with simple stellar populations gives an idea of the age and the metallicity of the principal components of the galaxy. It is also possible to determine the star formation and the enrichment histories of the galaxy or of the galaxy region.

Besides the studies related with astrophysical and cosmological problems, exploratory studies have been necessary to pave this way. Among others, they include the impact of the choice of the random number generator in the performance of the evolutionary algorithms, or the most suitable layout for accelerating the evaluation of complex functions on GPU.

1.2 Objectives

The main aim of this research is the development and the implementation of the most suitable methodologies to improve the efficiency of the data analysis in the area of astronomy, astrophysics and cosmology. This motivation is associated to the challenge that supposes the increase in the complexity and the volume of the data accessible by the practitioners.

In order to fulfil this general objective, and by depending on the particular problem tackled, diverse techniques have been applied to find the most suitable one for the problem. Concerning the techniques employed, parallel computing in diverse platforms: distributed computing, accelerator cards and clusters have been applied. On the other hand, evolutionary computing has been also applied to extract knowledge from large and complex data sets. The main tasks developed are the following:

1. Study of the suitability of distributed computing paradigm for the resolution of complex problems.

2. Study of the suitability of evolutionary algorithms for the resolution of problems in the area of astronomy, astrophysics and cosmology.

3. Analysis of the sensitiveness of evolutionary algorithms to the choice of the random number generator.

4. Study and implementation of metaheuristics techniques for fitting the rotational curves of spiral galaxies, and for modelling low-resolution galaxy spectral energy distribution.

5. Review of the algorithms for the analysis of the spatial correlation of galaxies and their shapes.

6. Development of parallel algorithms for the analysis of spatial distribution of galaxies, and for the analysis of the distortion of the shape of the galaxies due to the presence of dark matter.

7. Evaluation of the results from computational and physical perspectives, and adaptation of the codes for its distribution.

In this Thesis, diverse areas of interest can be underlined:

1. From the computational point of view, two technical areas have been treated. On the one hand, metaheuristics techniques have been employed to extract synthetic knowledge from large data volumes. In some cases, previous exploratory studies have been mandatory to survey the scope of the approach. On the other hand, parallelism techniques are applied to reduce the execution time. As a result, more efficient analyses than previously have been implemented.

2. From a scientific perspective, it has to be underlined that the developed codes are going to be exploited in the analysis of data by international collaborations, such as: Dark Energy Survey (DES), Physics of the Accelerating Universe (PAU); or being promoted, such as Calar Alto Legacy Integral Field spectroscopy Area survey (CALIFA). In relation to the previous state-of-the-art, the new implementations involve net improvements, such as: larger capacity to extract knowledge from data volumes, and more efficiency for analysing without approximations with affordable execution times.

3. Regarding an economic position, the improvement in the efficiency of the applications reverberates into further improvements in the exploitation of the computational platforms: more jobs per unit of time can be executed, as well as a lower power consumption per application.

1.3 Organization of the Document

The organization of this Thesis is as follows: firstly a survey of the auxiliary methods used are presented in chapter 2. They include the statistical tests used in the analysis, data mining algorithms used for a concise presentation of the numerical results, or the hardware used for testing the approaches. Next, in chapter 3, some studies related with evolutionary computation are described. In this chapter, fundamental aspects

of the evolutionary algorithms are verified. Chapter 4 presents some practical studies of the application of GPU computing. These studies aim to check the suitability of diverse aspect of porting codes to GPU computing. The application of the evolutionary algorithms to the astrophysical problems is presented in chapter 5. In chapter 6, the parallelization of the correlation functions, two-point correlation function and shear-shear correlation function are described. A modification incrementing the accuracy on the most accepted strategy to construct histograms in GPU is also presented in chapter 6. Finally, chapter 7 summarizes the research carried out and outlines several future works.

Chapter 2

Methods and Hardware

2.1 Introduction

ALONG this thesis, diverse methods have been used with very different objectives. In most of the cases, the objective is to analyse the data in order to infer if two or more numerical results are significantly different. In other cases, the objective of the technique is to produce the best visual representation of the data.

Besides, the computational platforms and infrastructures employed in this thesis are described. The information presented in this chapter allows understanding the scope of the works performed and the improvements achieved.

2.2 Methods

2.2.1 Statistics

Statistical hypothesis testing is a fundamental method used at the data analysis stage of a comparative experiment. For this comparison, two kind of tests can be used: parametric and non-parametric. The main difference between parametric and non-parametric tests rely on the assumption of the distribution underlying the sample data. Given that the non-parametric tests do not require explicit conditions on the underlying sample data, they are recommended when the statistical model of data is unknown [37]. The Wilcoxon signed-rank test, Kruskal-Wallis test and the sign test belong to the category of non-parametric tests.

The Kruskal-Wallis test [89] is one of such non-parametric tests, which is used to compare three or more groups of sample data. For this test, the null hypothesis assumes that the samples are from identical populations; whereas the alternative hypothesis assumes that the samples come from different populations.

When the null hypothesis in a multiple comparison (e.g. Kruskal-Wallis) is rejected, it implies the use of a post-hoc test to determine which sample makes the difference. The most typical post-hoc test is the Wilcoxon signed-rank test with the Bonferroni or Holm correction [37].

The Wilcoxon signed-rank test also belongs to the non-parametric category. It is a pairwise test that aims to detect significant differences between two sample means [36, 37], for example the numerical results of two codes before and after a modification.

On the other hand, the Bonferroni correction aims to control the Family-Wise Error Rate (FWER hereafter). FWER is the cumulative error when more than one pairwise comparison (e.g. more than one Wilcoxon signed-rank test) is performed. Therefore, when multiple pairwise comparisons are performed, Bonferroni correction allows maintaining the control over the FWER.

In order to assess if one algorithm performs better than other when implementing a modification, the sign test can be used. The sign test is also a non-parametric test. For this test, the differences between both sets of results are calculated. Next, by counting the number of plus signs or minus signs, it can be stated if one particular implementation performs better than the other. For this scenario, the number of plus signs or minus signs must be lower than or equal to a critical value depending on the sample size.

2.2.2 Support Vector Machine

The use of Support Vector Machine (SVM hereafter) allows comprehensibly visualizing complex data. Furthermore, it permits the knowledge extraction from large numerical data sets. Through building a model with SVM, patterns in data can be inferred, being the model more comprehensible than the numerical output [40]. If the data are linearly separable, then LinearSVM is the simplest SVM classification model. In this Thesis, the SVM models have been produced by using scikit-learn API [74].

SVM is useful for distinguish areas with different behaviours. In the context of this Thesis, SVM has been employed in chapter 4 to show the configurations where a particular layout is the most suitable for evaluating individuals on GPU or on CPU.

2.2.3 Random Number Generator

In this thesis, two random number generators (RNG hereafter) have been used. On the one hand, most of the calculations have been performed by using a subroutine based on Mersenne Twister [59].

This pseudorandom number generator holds a high quality passing the most stringent tests of randomness. It was designed in 1997 by Matsumoto and Nishimura to rectify many of the flaws found in older RNGs. Its name derives from the fact that period length is chosen to be a Mersenne prime number.

The main features of the Mersenne Twister RNG are:

- It has a very long period of $2^{19937} - 1 \simeq 10^{6000}$. While a long period is not a guarantee of quality in a random number generator, short periods —common in many software packages— are problematic.

- It is k-distributed to 32-bit accuracy for every $1 \leq k \leq 623$ (Overlapping M-tuple test).

- It passes numerous tests for statistical randomness, including the *Diehard* tests. It passes most, but not all, of the even more stringent *TestU01 Crush* randomness tests.

2.2.3.1 Sensitiveness to the RNG

On the other hand, in the study of the sensitiveness of the evolutionary algorithm to the change of the RNG, the default RNG in the C and C++ languages is also employed, the rand() function.

Many platforms have poor-quality versions of the rand() function, however GNU platforms —*glibc*— implement a version with higher quality and broadly accepted as good quality RNG.

The main features of the GCC RNG are:

- The implementation of *glibc* corresponds to the category of Linear Congruential Generator [80].

- It has a period of $\simeq 2^{31} - 1$. This period is accepted in general as long but it is clearly shorter than the Mersenne Twister RNG.

Spite of the difference of period between both RNGs, the GNU rand() is accepted as good quality RNG, being used in many scientific and technical works. The question is if these differences affect to the final results of the optimization process when implementing in evolutionary algorithms (chapter 3).

2.3 Computational Infrastructure

Diverse computational infrastructures and hardware have been employed in this Thesis. In all cases, the purpose was to find the most suitable one for the problem treated. A brief description of them is presented in the following sections.

2.3.1 Grid

Distributed computing is nowadays a set of computational techniques that allow the scientists to approach problems with a high computational cost [53], specially when the units of simulation or analysis have not interaction among them. Diverse computational models have been used in these circumstances: from grid computing to volunteer or desktop computing. The grid computing allows the computers to be connected via a special software called middleware. The middleware exports and handles all the computer resources with the goal of providing a standard layer where the scientists can run their simulations and analyses.

The first concepts and technologies about grid computing were expressed by Foster and Kesselman in 1998 [49]. Before this, other publications started to express some ideas about the orchestration of wide-area distributed resources [90]. Now the grid is a consolidated discipline. However, the core components are under ongoing development in order to fulfil the requirements requested by the users.

Grid computing provides a huge volume of geographically distributed, dynamic and heterogeneous resources. The access to these resources is homogenized through the middleware.

In this Thesis, diverse infrastructures (EGEE: Enabling Grids for E-sciencE, ES-NGI: Spanish Grid Initiative) have been used, but in all cases, they implement the middleware gLite and GridWay as metascheduler.

2.3.2 Cluster Computing

In this Thesis and for comparison purpose, MPI and OpenMP implementations of diverse problems have be created and analysed. In these cases, the executions have been performed in Euler facility at CIEMAT. This cluster is composed by 144 nodes with two Quad-Core Xeon processors at 3.0 GHz with 8 GB at 667 MHz. The operating system is Red Hat Linux Enterprise, and the file system is Lustre.

2.3.3 GPU Resources

2.3.3.1 GPU Hardware Used in the 2PACF

The GPUs employed in this study have been two Fermi cards: C2050 and C2075, and a pre-Fermi card, the GTX295 —which contains two GPUs although only one is employed. The Tesla C2050 is fully IEEE-754-2008-compliant supporting single and double precision under the IEEE-754 standard. Otherwise, the GTX295 does not fully support the standard IEEE-754.

On the other hand, the OMP implementation, created for comparison purpose, is executed in a computer with two Intel Xeon X5570 processors at 2.93 GHz and 8 GB, whereas the CPU implementation is executed in a computer with two Intel Xeon E5520 processors running at 2.27 GHz and 6 GB of main memory.

2.3.3.2 GPU Hardware Used in the Shear-Shear Correlation Calculation and Other Works

The initial creation of the shear-shear correlation calculation code, all the optimization process, as well as, the comparisons with the previous effort done, have been executed on a machine with two Intel Xeon X5570 processors at 2.93 GHz and 8 GB of RAM, and a C2075 NVIDIA GPU card. CUDA release 5.0 and compute capability 2.0 have been used. The analysis and improvement of the precision of the histogram calculation on GPU have been also done by using this hardware.

Besides, the works in the searching of the most suitable data layout for accelerating the evaluation of non-separable functions has been executed in this facility.

On the other hand, the numerical experiments of the MPI-CUDA implementation of the shear-shear correlation calculation have been executed on a multi-core system at CETA-CIEMAT. The system is made up of two Quad Core Intel Xeon at 2.53 GHz with 12 physical cores and 24 GB of DDR3 RAM memory, which uses a motherboard Supermicro X8DTT-H and it is mounted on a bullx R424-E2. This node also has two M2075 GPUs with 448 CUDA cores and 6GB (5.375 GB with ECC enabled) of memory each one.

2.3.3.3 Overview of GPU Architecture and Programming Model

During the last two decades the semiconductor industry has followed two alternative paths to increase the performance of its products. On the one hand, the number of cores has grown evolving from a single core processor to two-core processor, four-core, etc. This has generated the multi-core architecture. On the other hand, the many-core

architecture follows a different strategy by implementing many small cores to profit from highly-parallel problems. NVIDIA GPU is an example of this kind of architecture.

The main differences between both types of architectures emerge from the purpose for which they are designed. Cores in multi-core architecture have to deal with a wide portfolio of sequential general-purpose codes. On the contrary, the many-core architecture comes form game industry where a massive number of floating-point calculations per time unit is required.

Scientific computing might benefit from this high capacity for simulation and analysis. For this purpose NVIDIA introduced the CUDA (Compute Unified Device Architecture) programming model. CUDA can be seen as a set of C extensions to handle code on GPU. A CUDA code embodies two differentiated parts: the sequential code which be executed in the CPU and the parallel code which be executed in the GPU. This piece of the code is termed the kernel. The compiler separates the two parts during the compilation.

From the architecture point-of-view, a GPU is composed of an array of highly-threaded streaming multiprocessors (SMs). Each SM is in turn composed of several streaming processors (SPs) which share control logic and instruction cache and are able to support many threads. This architecture is specially recommendable for SIMD[1] problems.

Concerning the data storage, the architecture implements diverse types of memories covering a wide range of capacities, latencies and bandwidths. Global memory is the main memory of the GPU card. Unfortunately it also has the lowest bandwidth and the largest latency. Data stored in global memory is accessible by all threads on the card. The register has the highest bandwidth and the lowest latency but its size is smaller. Registers are tightly bound to thread so that data in the registers are only accessible by the corresponding thread. A third type of memory is the termed shared memory. Regarding the latency and the bandwidth it is an intermediate case between the two previous types. Another difference is that it is accessible by all the threads belonging to a block of threads[2].

In spite of the fact that a CUDA kernel is executed correctly on any CUDA device, its performance will differ depending on the particular architecture and their code adaptations. For this reason it is necessary to know the particularities of the architecture to profit from the capabilities offered by the hardware.

In NVIDIA Tesla architectures each Streaming Multiprocessor (SM) had only 16k registers whereas in Fermi architecture this on-chip memory has grown up to 32k.

[1] Single Instruction, Multiple Data

[2] A block of threads is a logical group of threads which are executed on an SM.

Another feature that has been incremented in the Fermi architecture is the maximum number of threads per block from 512 to 1024.

NVIDIA Fermi architectures introduces a two-level transparent cache-memory hierarchy. Each SM has 64 KB of on-chip memory distributed between shared memory and L1 cache memory. Users can select diverse configurations of shared memory and L1.

When implementing coalesced or non-coalesced access to global memory the memory transaction segment size becomes an important factor in the final performance. In Tesla architecture the available memory transaction segment sizes are: 32, 64 and 128 bytes. The selected value depends on the amount of memory needed and the memory access pattern. The selection is automatic in order to avoid wasting bandwidth.

In the Fermi architecture the memory transaction segment size follows a different rule. When L1 cache memory is enabled, the hardware always issues transactions of 128 bytes (cache-line size); otherwise, 32 byte transactions are issued.

Chapter 3

Evolutionary Algorithms

3.1 Introduction

NOWADAYS, practitioners have a wide set of tools in order to afford their research activity: parallel techniques, evolutionary algorithms (EA hereafter), or data mining. The use of these techniques enhances the quality of the scientific production. As far as the treated problems become more and more complex, tools from different scientific disciplines are becoming mandatory and their interconnection necessary to face the analysis of data.

Besides the applied studies, exploratory studies are necessary to understand the scope and the impact of the techniques. For example, in this chapter some studies related to the efficiency of EAs or the impact of the choice of the RNG in the performance of the EAs are presented.

3.2 Related Work

3.2.1 Related Work for the Sensitiveness of Evolutionary Algorithms to the Random Number Generator

Many works have examined diverse aspects of the impact of the RNG over the final performance of EAs. However, relevant differences between these previous works and the study performed in this thesis have been found.

The first works in this area [61, 63] examined the impact of the RNG choice on the performance of Genetic Algorithm (GA), while applying to a collection of diverse well-known GA test functions. In this study, no statistical evidence of impact over the GA performance due to the RNG quality is found. However, the coarse-grained statistical analysis employed puts in quarantine the conclusions attained.

In a further study using finer statistics [62], no correlation between goodness on the RNG tests —Diehard suite— and good performance by the GA is obtained.

Other paper [12] has studied the sensitiveness of GA to the choice of RNG focusing on the components which are most affected by the RNG. The work presents an ablation experiment using two RNG and the true random number from an atmospheric noise source. The experiments show that the RNG used to initialize the population has a critical impact over the final performance; whereas the RNG used as input to other operations —crossover and mutation— do not significantly affect to the performance.

3.2.2 Related Work for the Analysis of Behaviour of Evolutionary Algorithms: Particle Swarm Algorithm as Case Study

In the study of the behaviour of the evolutionary algorithms and specially the behaviour of Particle Swarm Optimizer (PSO), most of the previous works are related with PSO itself [27, 48], its weaknesses [22, 72], and its variants: Inertial Weight [27] (IWPSO), Particle Swarm Optimization with Massive Extinction [101] (PSOME), Fitness Distance Ratio Based Particle Swarm Optimization [75] (FDRPSO), Dissipative Particle Swarm Optimization [31] (DPSO), A Diversity-Guide Particle Swarm Optimizer [83] (ARPSO), and Mean Particle Swarm Optimization [26] (MeanPSO).

However, previous publications with the aspects treated in this chapter: multipopulation PSO and benchmarking of PSO variants have not been found.

3.2.3 Related Work for the Application of Evolutionary Algorithms to the Resolution of Complex Problems: Bateman Conjecture as Case Study

No previous studies of the conjecture of Bateman have been found in the scientific literature. Only the scientific spreading book [98] refers to it.

3.3 Computational Platforms for Analysing Evolutionary Algorithms

In this section, some studies about the suitability of diverse computational platforms for scientific computing, and specially for analysing EAs, are performed.

3.3.1 Taxonomy of Grid Applications

The taxonomies provide useful information about the elements corresponding to each category, independently of the object classified. In this section, a revision of the taxonomies used for grid applications is presented. Although the categories composing

the taxonomies presented in this section are not perfectly independent, nor well settled in the community; they allow extracting important information from the application before its adaptation.

3.3.1.1 Taxonomy Based on the Application Type

One possible classification of the grid applications is based on their primary driving reasons for using the grid [8]. In the following, the three categories of this taxonomy are presented:

Megacomputing applications. These are problems that can be divided into a large number of independent parts. Each part is completely independent of the others. These problems are called *embarrassingly parallel*. In this category, it comes many problems biochemistry, high energy physics or fusion. All the applications based on Montecarlo simulations fall in this category.

Finally, within this category they are also applications referred to as sweeping parameters (*parameter sweep*). In these problems, the application is executed multiples times with different initial conditions or configurations.

Seamless data access applications. These are applications which use the grid to access and to integrate the use of multiples data sets and computational resources. In these cases, the grid is used for its ability to store large volume data and simultaneously to serve to a community around the world.

Today, excellent examples of this kind of applications are showed in the high energy physics and astronomy communities. Moreover, probably the evolution of the grid is going to attract many users communities with similar problems.

Loosely coupled applications. This case is functionally compound applications with interaction between the different parts of them. For example, applications where the outputs of some parts are used as input to other parts. As final result, a set of codes handles some information which suffers diverse manipulations.

This kind of applications is the most complex of the three categories and it is usually produced by scientific communities hardly exploiting the grid paradigm. They involve a strong implication of the community on the grid as fundamental computing model for their scientific activity.

In the research groups, a natural evolution from the *megacomputing applications* to *seamless data access applications*, and finally to *loosely coupled applications* is foreseen.

Usually the first contact of the research groups with the grid is articulated around the transposition of a problem of the *megacomputing applications* category. If the scientific area involved allows merging the needs and the interests of different groups, a transition towards *seamless data access applications* through the sharing of the data of the collaboration is envisaged. Finally, the collaboration might need complex visualisation, recovering data and simulation applications, evolving toward *loosely coupled applications*.

3.3.1.2 Extended Flynn Taxonomy

An alternative classification is to extend the taxonomy of Flynn [33] to the grid applications. Originally the taxonomy of Flynn is designed for parallel applications, however, it can be used as baseline to grid applications. From the original definitions, the concept *Instruction* has been changed by the concept *Program*; and the concept *Data* by the concept *File*.

SPSF. Single File Single Program. This case corresponds to applications with a single executable which produces a single file. An example of this kind of application is a montecarlo code.

SPMF. Single Program Multiple Files. In this case, a single application produces several output files. An example of this category is the production of some new images from a raw image.

MPSF. Multiple Programs Single File. In this case, the application is composed by more than one executable. These executables are applied consecutively to a file. As a consequence a work-flow is created. An example of this kind of application is a work-flow which manipulates an image applying diverse transformations in an established order.

MPMF. Multiple Programs Multiple Files. This case corresponds to the application of diverse executables, producing multiple output files. An example of this category could be a complex work-flow that creates some output files.

As general rule, as much as the application falls in the latest categories of this taxonomy, less suitable will be for the grid paradigm.

3.3.1.3 Scientific Community Taxonomy

An in-depth analysis of the applications running on grid shows the existence of a gap between the applications emerging from scientific communities well established around an international collaboration exploiting a scientific instrument (i.e. detector, satellite, sensor network); and the applications arising from independent small research groups. Unlike the taxonomies presented in the previous sections, where the criteria are exclusively technological; in this case, the criterion is environmental one.

In the case of a scientific community, there is a separation of roles between the developers of the applications and the users; while in the case of a independent group, all functions are overlapped.

Some of the features of each taxonomy are:

Applications supported by an *International Collaboration*. Examples of this category are the applications coming from the major international collaborations, mainly in high energy physics and astronomy. With the advent of international collaborations exploiting proteomic and genetic databases or digital repositories, it will produce a strong increase of cases in this category.

This category has the following features:

- Integration of different types of grid applications, including: portals, workflows, remote access to data, data handling, analysis, etc.
- Cooperative use of applications faced to more individual way to exploit the applications supported by an *Independent Group*.
- Separation of responsibilities between developers and users.
- Permanent operation running and production system on grid.
- The grid is used for computing and distributed storage, and in some cases for remote accessing to scientific instruments.
- High volume of computational resources.
- World wide geographic scope.

Applications supported by an *Independent Group*. This category includes a myriad of applications coming from research groups. Besides, the applications coming from small communities in the process of establishing an international collaboration might be included in this category.

This category has the following features:

- There is not integration of the applications.
- Individual use of applications.
- No separation of responsibilities between developers and users.
- Production in campaigns.
- The grid is normally only used for computing.
- Reduced volume of computational resources.
- Local geographic scope.

Both categories exposed in this section are suitable for the grid. The main difference will appear with the long term sustainability of the activity. The *International Collaboration* will sacrifice any point in order to fill the computational requirements imposed by the collaboration. Whereas, in the case of *Independent Group* there is an aspect of profiting technological opportunity. The technology is verified with some previous problems treated by the group.

3.3.1.4 Topological Taxonomy

Considering a grid application as a collection of executables to solve a scientific problem, then the topological structure of the executables can serve as a criterion for a taxonomy [32]. Oppositely to the previous taxonomies, this one will be only applicable for grid applications composed of several executables.

Under this taxonomy, the applications are classified in three categories:

Sequential Topology. In this category, the application is a collection of jobs that are sequentially executed in the same or different nodes of the grid infrastructure. The output serves as the input to the next job in the sequence.
In this case, the advantage of the execution of such applications in a grid environment is the ability to use the resources available on line as soon as they are released. Although the execution is sequential, not all the tasks which make up the application have to run on a single node of the grid. This strategy improves the scalability of the application by using as many resources as they are available.

Parallel Topology. In this category, the main body of the application is composed by diverse jobs which run in parallel on different nodes. Such applications are suitable for a grid environment, provided that an appropriate policy for booking resources is implemented.

Network Topology. This is the most complex case, in which the application is composed of some jobs which are executed in parallel and others which are sequentially executed. The execution of these applications in grid implies the existence of an efficient way of handling the synchronisation of the jobs and the transference of data files among them.

Due to the nature of the grid: geographical dispersion and heterogeneity, the applications belonging to the *network topology* category will be less suitable than the applications of the *sequential topology* category.

3.3.2 Volunteer Computing

Many regions around the world have low scientific funds. In these regions, the use of free software and *volunteer computing* is a very interesting option for scientific computing and for developing important research.

Volunteer computing is a variant of distributed computing which uses the computational resources of volunteers. They share their CPU-cycles with the projects that they are affiliated. In order to coordinate the download of the code project and the input data, the upload of the results, and the recollection of the statistic of use, a piece of software acting as middleware is necessary. The most popular middleware in volunteer computing is BOINC [4] (Berkeley Open Infrastructure for Network Computing).

For testing the suitability of this computational platform for scientific computing, three applications are deployed on the distributed infrastructure. This work was developed in collaboration with the ARCO research group for the Radio Network Design problem, the CAPI Research Group for the Microcalcification Clusters (MCCs) classification problem —both in the University of Extremadura—; and the CETA-CIEMAT for the virtualization problem.

The first example focuses on the telecommunications area. Many of the problems found in this area can be formulated as optimization tasks. In this case, the problem is the *Radio Network Design* (RND). This is an important problem in mobile telecommunications (for example in mobile/cellular technology), being also relevant in the rising area of sensor networks. When a set of geographically-dispersed terminals needs to be covered by transmission antennas (also called base transceiver stations -BTS-), the key subject will be to minimize the number and localizations of those antennas and to cover the biggest possible area. This is the RND, that is an NP-hard problem, and for this reason its resolution by means of volunteer computing is very appropriate.

The second research line is the optimization of neural networks classifiers by means of feature selection on input space, using Genetic Algorithms (GA). In many classification problems after the feature extraction step, the results can be improved by doing a feature selection step (so avoid the curse of dimensionality problem). However, the feature selection step is a very hard optimization problem with high computational requirements. In the past, we successfully applied GA for this selection in the cloud cover classification problem, using a 45 node Beowulf cluster. Now, we intend to apply this technique to Microcalcification Clusters (MCCs) classification in mammograms, for breast cancer diagnosis. This problem is considerably more complex than the former, and using only the cluster could be not enough. For this reason, the use of volunteer computing to solve it is proposed.

Finally, the third problem focuses on virtualization, where the goal is to provide a framework rapidly deployable to profit the best use of resources. Being the CETA-CIEMAT a centre devoted to grid computing, scenarios of under-use of resources appear. In these scenarios, to have the possibility to switch quickly the "flavour" of the machines, changing the real use dynamically, increases the maximum performance to the computers. As a result, BOINC clients completely virtualized has been successful employed in several projects, including these previously cited.

3.4 Sensitiveness of Evolutionary Algorithms to the Random Number Generator

There are numerous papers published every year in optimisation problems based on EAs which implement diverse high-quality RNGs. However, it is difficult to ascertain the role played by the RNGs in the final results. The intent of this section is to figure out the role of RNG in the final performance, and to assess if this element has any impact over the results obtained.

EAs techniques rely heavily on the use of RNG. From initial population generation, through the specific canonical operators applied to create new temporary population, the use of randomness is pervasive through EAs. Therefore, it is reasonable to wonder how RNG quality affects EAs performance.

In order to analyse the impact of the RNG over the performance of EA, two RNGs (Mersenne Twister and GCC rand()) (section 2.2.3) have been used to test their impact in the final performance of four EAs: Particle Swarm Algorithm (PSO), Differential Evolution (DE), Genetic Algorithm (GA), and Firefly Algorithm (FA). The two RNGs have been selected based on two following criteria:

- The RNG have to be frequently used in research papers.
- The RNG have to be considered as high-quality RNG.

In order to test our hypothesis about the impact of the RNG in the performance of these four EAs, a large number of experiments have to be executed to produce a high statistic. Besides, grid computing is employed to support the computational activity. This paradigm has made proof of be able to cover the requirements of a lot of scientific communities [49]. The computing capabilities delivered by this paradigm have increased the generation of new science. For this reason, a platform of grid computing (ES-NGI, section 2.3.1) has been selected for the present work in order to provide the necessary resources to produce the large data sets required.

3.4.1 Production Setup

The study is conducted using a set of benchmark functions (Table 3.1). These functions are selected in order that the set has a mixture of multimodal and monomodal, separable and non-separable functions. They have been extracted or inspired from *CEC 2010 and CEC 2008 Special Sessions and Competition on Large-Scale Global Optimization*; as well as other papers benchmarking EAs.

Regarding the features of the selected functions, they are as diverse as possible in order to cope with different scenarios of the RNG and EA. The functions termed *multimodal* present diverse minima in the interval of the variables, whereas the functions termed *monomodal* present only one minimum. Concerning the separability of the variables, if the variables involved in a function are *independent* of each other, the function is termed *separable*, otherwise *non-separable*.

In order to have a solid statistic, a total of 10^4 tries of each benchmark function has been executed. In this production, the powerful machinery of the grid was used to support the computational activity.

To manage the complexity of the problem —involving diverse benchmark functions— the grid jobs are created with 500 tries of a particular configuration. This structure assures the optimization of the execution time for the grid environment. Several runs are executed to reach the statistical relevance desired.

Each job is composed by a shell-script that handles the execution, and a tarball containing the source code (C++) of the program and the configuration files. When the job arrives to the Worker Node, it executes the instructions of the shell-script: to roll out the tarball, to compile the source code and to execute the 500 tries of the configuration for the benchmark function, and finally to create a tarball of the results files. When finishing the job, it recuperates the results tarball.

Table 3.1: Benchmark functions used in the study of the sensitiveness of evolutionary algorithms to the random number generator.

Expression	Optimum	Interval	Characteristics	
$f_1 = \sum_{i=1}^{D}[sin(x_i) + sin(\frac{2 \cdot x_i}{3})]$	$\simeq -1.21598 \cdot D$	[3, 13]	Multimodal	Separable
$f_2 = \sum_{i=1}^{D-1}[sin(x_i \cdot x_{i+1}) + sin(\frac{2 \cdot x_i \cdot x_{i+1}}{3})]$	$-2 \cdot D + 2$	[3, 13]	Multimodal	Full-non-separable
$f_3 = \sum_{i=1}^{D}[(x_i + 0.5)^2]$	0	[-100, 100]	Monomodal	Separable
$f_4 = \sum_{i=1}^{D}[(x_i)^2 - 10 \cdot cos(2\pi x_i) + 10]$	0	[-5.12, 5.12]	Monomodal	Separable
$f_5 = \sum_{i=1}^{D}[(x_i)^2]$	0	[-5.12, 5.12]	Monomodal	Separable
$f_6 = \sum_{i=1}^{D}[x_i \cdot sin(10 \cdot \pi \cdot x_i)]$	$\simeq -1.95 \cdot D$	[-1, 2]	Multimodal	Separable
$f_7 = 20 + 20 \cdot \exp(-20 \cdot \exp(-0.2\sqrt{\frac{\sum_{i=1}^{D} x_i^2}{D}})) - \exp(\sum_{i=1}^{D} \frac{cos(2\pi x_i)}{D})$	0	[-30, 30]	Monomodal	Separable
$f_8 = 418.9828 \cdot D - \sum_{i=1}^{D}[x_i \cdot sin(\sqrt{\lvert x_i \rvert})]$	0	[-500, 500]	Multimodal	Separable
$f_9 = \sum_{i=1}^{D-1}[100 \cdot (x_{i+1} - x_i^2)^2 + (x_i - 1)^2]$	0	[-5.12, 5.12]	Monomodal	Full-non-separable
$f_{10} = \sum_{i=1}^{D}[i \cdot (x_i)^2]$	0	[-5.12, 5.12]	Monomodal	Separable
$f_{11} = \sum_{i=1}^{D}[(x_i)^2] + [\sum_{i=1}^{D}(\frac{i}{2} \cdot x_i)]^2 + [\sum_{i=1}^{D}(\frac{i}{2} \cdot x_i)]^4$	0	[-5.12, 5.12]	Monomodal	Separable

Table 3.2: p-value for non-parametric hypothesis testing for each EA and fitness function.

Function	PSO	DE	GA	FA
f_1	$1.79 \cdot 10^{-6}$	nan	$8.31 \cdot 10^{-11}$	0.034
f_2	0.572	0.0	0.0	0.574
f_3	$3.40 \cdot 10^{-6}$	0.0	0.292	0.885
f_4	0.783	0.0	0.0	0.834
f_5	0.816	0.0	$2.78 \cdot 10^{-10}$	$5.89 \cdot 10^{-6}$
f_6	10^{-4}	$2.62 \cdot 10^{-12}$	0.0	0.013
f_7	0.222	0.0	0.0	0.558
f_8	0.130	0.0	0.0	0.522
f_9	0.640	0.0	0.003	0.014
f_{10}	0.013	0.0	0.646	0.355
f_{11}	$2 \cdot 10^{-4}$	0.0	0.002	0.135

Taking into account the total number of fitness functions, algorithms and runs, the whole production involved 1,760 jobs and 880,000 tries.

The configuration is selected in order to have the maximum number of calls to the corresponding RNG. Thus, the configuration involves 10,000 cycles, 100 particles/individuals as population size and a dimensionality of 100 for all fitness functions. The use of this configuration, independently of the EA employed, involves at least 10^8 calls to the RNG by execution, and 10^{12} calls by benchmark function.

3.4.2 Results and Analysis

The numerical results obtained with each RNG are analysed with the Wilcoxon signed-rank test. In Table 3.2, the p-value of Wilcoxon signed-rank test for each EA and fitness function is presented. In our study, the analysis of the sensitiveness of the EAs is based on these values.

As null hypothesis, the equal performance of both RNG is stated. The acceptance or rejection of the null hypothesis for a confidence level of 95% (p-value under 0.05), is based on whether the p-value at Table 3.2 is higher (acceptance) or lower (rejection) than $\alpha = 0.05$.

As it can be observed (Table 3.2), the null hypothesis ($H_0 : \ \mu_1 = \mu_2$) can be rejected in all cases for DE. Hence, DE algorithm is considered as very sensitive to the choice of RNG, producing results significantly different in relation to the RNG implemented. For f_1 the p-value can not be obtained due to all results for both RNGs are equal, being not possible to perform the Wilcoxon signed-rank test.

For PSO in 5 tests (f_1, f_3, f_6, f_{10}, f_{11}) the null hypothesis can be rejected; otherwise in 6 cases the null hypothesis can not be rejected (f_2, f_4, f_5, f_7, f_8, f_9). Consequently, PSO shows a low sensitiveness to the choice of the RNG. Furthermore for FA, the p-values obtained allow rejecting the null hypothesis in 4 cases (f_1, f_5, f_6, f_9), failing to reject in the 7 remaining cases; showing the lowest sensitiveness of the four EAs studied.

Between these two extreme cases —DE and FA— the p-values obtained for GA allow rejecting the null hypothesis in 9 cases, and only in 2 cases (f_3 and f_{10}) the null hypothesis can not be rejected.

Moreover, in relation to the main focus of this work, most of the cases analysed point that the use of a particular RNG affects to the final performance. This sensitiveness is particularly more explicit in DE, where for all cases the null hypothesis is rejected, than in PSO, where only in 5 cases, or FA, where only in 4 cases the null hypothesis can be rejected.

The analysis of these results allows building a scale of sensitiveness for the EAs tested: $DE > GA > PSO > FA$.

Based on the non-parametric analysis performed in this section, it can be concluded that the choice of the RNG affects to the final performance of the EA tested, although the level of sensitiveness shown is different among them. Unfortunately, this analysis does not allow establishing conclusions about the best performance of the EAs with the use of a particular RNG.

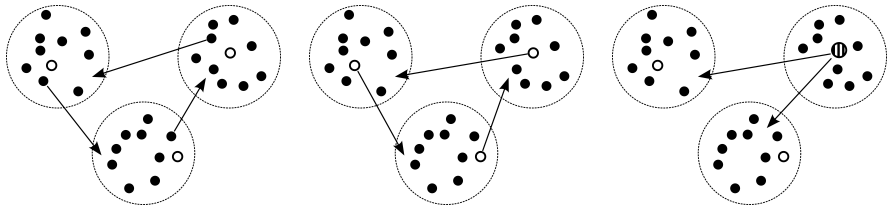
3.5 Analysis of Behaviour of Evolutionary Algorithms: Particle Swarm Algorithm as Case Study

3.5.1 Performance Improvement in Multipopulation Particle Swarm Algorithm

Particle Swarm Optimization (PSO) has demonstrated to be an efficient and fast optimizer (Eqs. 3.1, 3.2), with a wide applicability to very diverse scientific and technical problems. In spite of this efficiency, some disadvantages have appeared, mainly the premature convergence, as well as, the stagnation of the fitness improvement.

$$v_{id}(t+\delta t) \leftarrow v_{id}(t) + c_1 \cdot Rand() \cdot (x_{id}^{localbest} - x_{id}) + c_2 \cdot Rand() \cdot (x_{id}^{globalbest} - x_{id}) \quad (3.1)$$

$$x_{id}(t+\delta t) \leftarrow x_{id}(t) + v_{id} \quad (3.2)$$



(a) An individual randomly selected is copied to the neighbour population.

(b) The best individual of the population is copied to the neighbour population.

(c) The best individual of all the populations is copied to the other populations.

Figure 3.1: Schema of the three exchange patterns employed in this study.

The use of more than one population, with periodic interchange of the best individuals, is a technique widely employed in diverse EAs. The multipopulation technique has proved to be excellent to accelerate the convergence and to reduce the stagnation. However, in order to maximize the advantage of this approach, the individuals interchanged and the moment of the action have to be carefully selected. Otherwise, if an incorrect moment is selected, not major advantage will be given to the algorithm.

Two factors seem to be desirable for the individuals interchanged. First of all, they should represent as good solutions as the best individuals of the population where they are introduced. Second of all, they ought to increase the genetic diversity of the population.

In spite of the excellent performance of the original PSO, multitude of variants, improvements or alternatives have been proposed [16]. However, a complete characterization of the possible improvements on PSO requires a study of the behaviour of PSO in relation to a multipopulation approach.

The benchmark functions selected are presented in Table 3.1 as well as their characteristics. For supporting the production, the grid infrastructure described at section 2.3.1 has been employed.

3.5.1.1 Multipopulation Modifications in PSO

In order to characterize the capacity of the multipopulation approach to improve the performance, an in-depth study has been performed using a catalogue of configurations and fitness functions (Table 3.1). In all cases, three populations are implemented. This includes:

- Which individual is interchanged?
  - In the first configuration, an individual randomly selected is copied to the neighbour population (Fig. 3.1(a)). This configuration should not produce

better solutions that the equivalent –swarm size– configuration for a single population. A priory, it should represent the worst result of the three configurations.

  - In the second case, the best individual of one population is copied to the next population establishing a circular topology (Fig. 3.1(b)). A net improvement, in relation with the previous configuration, is expected.
  - In the third case, the best individual of any population is copied to the other populations (Fig. 3.1(c)). For this configuration, a net improvement is also expected. In relation to the previous configuration it is not obvious if the minor genetic diversity introduces a premature convergence.

- And, when is it more profitable to copy individuals from its original population to other population? For this characteristic three patterns have been established. when these percentages of cycles are reached, then the interchange is activated.
  - 33% - 66%
  - 25% - 50% - 75%
  - 20% - 40% - 60% - 80%

3.5.1.2 Results and Analysis

The results of the study are presented at Table 3.3. In this table, the configuration which obtains the best result for each fitness function is presented. If more than one configuration produces the same best result, none is represented.

The analysis of Table 3.3 shows a dominance of interchange patterns where best individuals are involved. Furthermore, a tendency toward more number of interchanges, 20%, is also remarked. This case corresponds to an activation of the interchange every 20% of cycles: 20%, 40%, 60%, 80%. This tendency will be more clearly drawn at Table 3.4. Spite of this trend, a certain dispersion of best results is observed. Even the worst a priory configuration (individuals randomly selected) obtains several best results.

Configurations, where only individuals randomly selected are interchanged, have the capacity to produce best results. This fact underlines the relevance of the genetic diversity of the individuals in the swarm. Stagnation in the convergence process is frequently due to a lack of genetically different individuals, able to explore areas far away of the local minima. Therefore, the individuals randomly selected provide genetic richness to the target swarm avoiding stagnation.

At Table 3.4, a digest of the data of Table 3.3 is shown. The analysis of data exposes that the performance of any interchange pattern involving best individuals has better

Table 3.3: Results of the benchmark functions for diverse interchange patterns. For each configuration (Dimension, Population size and number of Generations) and fitness function, the best exchange pattern is presented. The percentage indicates when the interchange is activated, i.e. 20% means 4 interchanges: 20%, 40%, 60%, and 80%.

D	P	G	Random			Circular Best			Global Best		
			33%	25%	20%	33%	25%	20%	33%	25%	20%
100	10	10^2						$f_3\ f_4$			
			f_8		f_7	f_6	$f_1\ f_9$	$f_5\ f_{10}\ f_{11}$			f_2
		10^3	f_{11}	f_7	f_8	f_1	f_2	$f_3\ f_4\ f_6\ f_{10}$		f_9	f_5
		10^4		$f_5\ f_9$			f_8	f_3	f_{11}	$f_4\ f_{10}$	$f_2\ f_6$
	100	10^2				$f_8\ f_9$		$f_1\ f_5$	$f_4\ f_{11}$	f_{10}	$f_2\ f_3\ f_6$
		10^3	f_8		f_{11}	$f_4\ f_7$	f_2	$f_3\ f_{10}$	f_9	f_5	f_6
		10^4		f_{11}		f_4	f_9	$f_2\ f_3$		f_5	$f_6\ f_{10}$
50	10	10^2		f_8	f_5		$f_6\ f_{10}$	$f_7\ f_9$	f_{11}	$f_1\ f_3\ f_4$	f_2
		10^3	$f_7\ f_8$			f_6	$f_2\ f_{11}$	$f_1\ f_3\ f_{10}$	f_9	f_4	f_5
		10^4	f_{11}			f_4			$f_2\ f_9$	$f_5\ f_6$	$f_3\ f_{10}$
	100	10^2		f_9		$f_4\ f_{11}$	f_8	f_2		$f_1\ f_6$	$f_3\ f_5\ f_{10}$
		10^3						$f_2\ f_{10}$		$f_4\ f_5\ f_{11}$	$f_3\ f_6\ f_9$
		10^4		f_{11}			f_6	$f_3\ f_9\ f_{10}$	f_4	$f_2\ f_5$	
20	10	10^2		f_9	f_{11}		f_5	$f_3\ f_4$	f_6	f_2	$f_1\ f_8\ f_{10}$
		10^3		f_2			$f_5\ f_9$	$f_3\ f_{10}$		f_6	$f_4\ f_{11}$
		10^4		f_{11}		f_3	f_2	f_4		$f_5\ f_6$ $f_8\ f_9\ f_{10}$	
	100	10^2			f_4	f_8	f_{10}			$f_3\ f_6\ f_9\ f_{11}$	$f_1\ f_2\ f_5$
		10^3			f_8	$f_3\ f_9\ f_{11}$	$f_2\ f_5$	$f_4\ f_{10}$			f_6
		10^4	f_{10}			$f_5\ f_8\ f_9\ f_{11}$				f_3	f_4
10	10	10^2				$f_5\ f_{11}$	f_8	f_9		$f_2\ f_3\ f_4\ f_6$	$f_1\ f_7\ f_{10}$
		10^3	f_4		f_3	f_5	f_8	$f_9\ f_{11}$		$f_2\ f_6$	f_{10}
		10^4		f_5	f_4			$f_2\ f_8\ f_9$		f_3	$f_6\ f_{10}\ f_{11}$
	100	10^2		f_9	f_3	f_4	f_{10}	f_8	$f_5\ f_{11}$	$f_1\ f_6$	f_2
		10^3			$f_2\ f_8$	$f_4\ f_5$	f_9	f_{11}			$f_3\ f_6\ f_{10}$
		10^4		$f_9\ f_{10}$		f_4		f_2	$f_3\ f_{11}$	f_5	f_8

Table 3.4: Number of best results in multipopulation PSO obtained for each configuration for each interchange pattern.

	Random	Circular Best	Global Best	
33%	8	27	14	**49**
25%	14	23	41	**78**
20%	12	43	42	**97**
	34	**93**	**97**	**Totals**

Table 3.5: Number of best results in multipopulation PSO obtained for each configuration in function of the character of fitness function.

	Multimodal functions				Monomodal functions			
	Random	Circular Best	Global Best		Random	Circular Best	Global Best	
11	3	6	2	33%	5	21	12	**38**
30	2	12	16	25%	12	11	25	**48**
33	4	10	19	20%	8	33	23	**64**
Totals	**9**	**28**	**37**		**25**	**65**	**60**	**Totals**

performance than patterns involving only randomly selected individuals. This consideration is obvious, however, the number of best results for randomly selected individuals are not negligible in relation to the other two selection modes. As a consequence, the importance of the genetic diversity is underlined.

Taking into consideration this argument, it is foreseeable that an algorithm interchanging best individuals and other randomly selected ought to reach higher level of convergence toward good solutions in relation to other interchanging only best solutions.

Regarding the number of exchanges, the Table 3.4 shows an improvement of the best results as much as the number of interchanges grows. However, this trend seems to have an asymptotic limit. The number of best results obtained augments from 25% to 20% more smoothly than from 33% to 25%.

At Table 3.5 an alternative digest of Table 3.3 is presented. In this case the analysis is performed in function of the character of fitness function: multimodal or monomodal. Similar considerations to those expressed in the analysis of the Table 3.4 can be applied to the analysis of this table.

3.5.2 Study of Performance of Particle Swarm Optimization Algorithms Using Grid Computing

The PSO is an efficient and fast optimizer. In spite of the efficiency demonstrated by the PSO, also some disadvantages have appeared, mainly the premature convergence which prevents the finding of optimal solutions.

Generally, when proposing enhancements to PSO, the original authors execute some benchmarks. These benchmarks include the most profitable configuration and function for the enhancements proposed, being, in general, a reduced number of tests. Besides, the benchmarks have been executed with different set of functions and configuration; and usually only between the standard algorithm and the new one. This impedes checking the results among the enhancements proposed.

In this section a survey of the performance of PSO variants is presented. For supporting the production, the grid infrastructure described at section 2.3.1 has been employed.

3.5.2.1 Weaknesses of Standard Particle Swarm Optimization

Diverse authors [22, 72] have demonstrated that the particles in PSO oscillate in damped sinusoidal waves until they converge to new positions. These new positions are between the global best position and their previous best position. During this oscillation, a position visited can have better fitness than its previous *local best*, reactivating the oscillation. This movement is continuously repeated by all particles until the convergence is reached or an end execution criterion is met.

However, in some cases, where the global optimum has not a direct path between current position and the local minimum already reached, the convergence is prevented. In this case, the efficiency of the algorithm diminishes. From the computational point of view, a lot of CPU-time is wasted exploring areas of suboptimal solutions already discovered.

In order to avoid this pernicious effect, diverse alternatives to PSO formulation have been proposed. Frequently, these enhancements are based on effects present in the nature, enforcing the image of the PSO algorithm as a bio-inspired algorithm.

3.5.2.2 Production Setup

The study is conducted using a set of benchmark functions (Table 3.1). For these functions, 400 tries of each configuration, each algorithm and each benchmark function have been executed.

To manage the complexity of the problem, involving several algorithms and benchmark functions, and a set of configurations for each of them; the grid jobs are created with 50 tries of each configuration of one specific algorithm and function. This structure assures the optimization of the time execution for the grid environment. A total of 8 runs per case are executed to reach the 400 tries.

Each job is composed with a shell-script that handles the execution, and a tarball containing the source code (C++) of the program and the configuration files. When the job arrives to the worker node, it executes the instructions of the shell-script: to roll out the tarball, to compile the source code, to execute the 50 tries of each configuration for a particular algorithm and benchmark function, and finally to resume in a tarball all the output files. When the job finishes, the middleware recuperates the output tarball containing the output files.

All PSO variants shares some common parameters, such as, $c_1 = c_2 = 1$ in Eq. 3.1, and the maximum velocity, $V_{max} = 5$; and configuration values of dimensionality (20, 100), population size (10, 100) and number of generations (100, 1000, 10000).

Finally, the production is composed by a total of 616 jobs, resulting from the 11 fitness functions and 7 PSO variants; and 8 runs per function and algorithm. As a consequence that each job has 400 tries, the number of total tries executed is 246,400. On the other hand, the mean CPU-time employed for run is 148.7 hours, then the total CPU-time for the eight runs is 1,189.6 hours.

3.5.2.3 Results and Analysis

At Tables 3.6 and 3.7, a resume of the best results obtained for each fitness function, and configuration is presented. For each fitness function and configuration, the PSO variants which obtains the best result are presented. In the case of several variants obtaining the same best results, all of them are presented.

From the results presented at Tables 3.6 and 3.7, it can be concluded that MeanPSO is the variant which more frequently outperforms to the other PSO implementations. It obtains the best result in 91 from the total 132 tests, the 69% of the tests, being the algorithm dominant for the functions f_3, f_4, f_6, f_{10} and f_{11}. Moreover, the MeanPSO produces the best results in 11 of the 12 configurations for f_5, and for f_7 it shares the best results with FDRPSO for all configurations. However, in the functions f_1 and f_2 MeanPSO does not obtain any best result.

The second best variant is FDRPSO, obtaining 19 from the total tests, the 14%. However, the best results for this algorithm are concentrated in the functions f_7 and f_8.

Table 3.6: Results of benchmarks of the PSO variants (Dimension, Population size and number of Generations) for the fitness functions f_1, f_2, f_3, f_4, f_5 and f_6 after 400 tries.

D	P	G	f_1	f_2	f_3	f_4	f_5	f_6
100	10	100	DPSO	DPSO	MeanPSO	MeanPSO	MeanPSO	MeanPSO
		1000	DPSO	PSOME	MeanPSO	MeanPSO	MeanPSO	MeanPSO
		10000	DPSO PSO PSOME	DPSO	MeanPSO	MeanPSO	MeanPSO	MeanPSO
	100	100	DPSO	PSO	MeanPSO	MeanPSO	MeanPSO	MeanPSO
		1000	PSO	PSO	MeanPSO	MeanPSO	MeanPSO	MeanPSO
		10000	DPSO PSO PSOME	DPSO	MeanPSO	MeanPSO	MeanPSO	MeanPSO
20	10	100	PSO	IWPSO	MeanPSO	MeanPSO	MeanPSO	MeanPSO
		1000	PSO	DPSO	MeanPSO	MeanPSO	MeanPSO	MeanPSO
		10000	DPSO PSO PSOME	DPSO	MeanPSO	MeanPSO	MeanPSO	MeanPSO
	100	100	DPSO	IWPSO	MeanPSO	MeanPSO	MeanPSO	MeanPSO
		1000	DPSO PSO IWPSO	PSOME	MeanPSO	MeanPSO	MeanPSO	MeanPSO
		10000	DPSO PSO PSOME IWPSO	DPSO	MeanPSO	MeanPSO	IWPSO	MeanPSO

Table 3.7: Results of benchmarks of the PSO variants (Dimension, Population size and number of Generations) for the functions f_7, f_8, f_9, f_{10} and f_{11} after 400 tries.

D	P	G	f_7	f_8	f_9	f_{10}	f_{11}
100	10	100	MeanPSO FDRPSO	PSOME	MeanPSO	MeanPSO	MeanPSO
		1000	MeanPSO FDRPSO	FDRPSO	MeanPSO	MeanPSO	MeanPSO
		10000	MeanPSO FDRPSO	FDRPSO	PSO	MeanPSO	MeanPSO
	100	100	MeanPSO FDRPSO	FDRPSO	MeanPSO	MeanPSO	MeanPSO
		1000	MeanPSO FDRPSO	FDRPSO	MeanPSO	MeanPSO	MeanPSO
		10000	MeanPSO FDRPSO	MeanPSO	PSO	MeanPSO	MeanPSO
20	10	100	MeanPSO FDRPSO	FDRPSO	MeanPSO	MeanPSO	MeanPSO
		1000	MeanPSO FDRPSO	DPSO	PSO	MeanPSO	MeanPSO
		10000	MeanPSO FDRPSO	FDRPSO	PSO	MeanPSO	MeanPSO
	100	100	MeanPSO FDRPSO	FDRPSO	MeanPSO	MeanPSO	MeanPSO
		1000	MeanPSO FDRPSO	DPSO	MeanPSO	MeanPSO	MeanPSO
		10000	MeanPSO FDRPSO DPSO	DPSO	PSO	MeanPSO	MeanPSO

For the third place, two variants obtain 17 best results (13%), they are PSO and DPSO. Specially significant is the fact that the original algorithm obtains 17 best results wining to other more complex variants.

At Tables 3.8, 3.9, 3.10 and 3.11 the previous results are digested by gathering by diverse criteria. As it can be appreciated, there are not major differences in the effectiveness of the variants face to different swarm size, cycles, dimensions or behaviour of the fitness function.

Table 3.8: Best results of the PSO variants in relation with the dimension.

D	PSO	IWPSO	PSOME	FDRPSO	DPSO	ARPSO	MeanPSO
20	8	5	3	9	11	0	44
100	7	0	2	10	8	0	47

Table 3.9: Best results of the PSO variants in relation with the swarm size.

P	PSO	IWPSO	PSOME	FDRPSO	DPSO	ARPSO	MeanPSO
10	7	1	3	10	9	0	45
100	8	4	2	9	10	0	46

Table 3.10: Best results of the PSO variants in relation with the number of cycles.

G	PSO	IWPSO	PSOME	FDRPSO	DPSO	ARPSO	MeanPSO
100	2	2	0	7	4	0	32
10,000	8	2	5	8	10	0	27

Table 3.11: Best results of the PSO variants in relation with the behaviour of the fitness function.

Behaviour	PSO	IWPSO	PSOME	FDRPSO	DPSO	ARPSO	MeanPSO
Monomodal	5	0	1	12	1	0	68
Multimodal	10	7	4	7	18	0	11

3.6 Application of Evolutionary Algorithms to the Resolution of Complex Problems: Bateman Conjecture as Case Study

The Bateman's Conjecture (Eq. 3.3) proposes how many coincidences of sum of powers of two prime numbers are [98]. Until now only one coincidence has been found (Eq. 3.4). No previous survey of the Conjecture has been published in the scientific literature, nor analytic demonstration has proved the existence of a finite or infinite number of coincidences. This section presents the works performed to search more coincidences in the Bateman's Conjecture.

$$\sum_{i=0}^{m} p_1^i = \sum_{i=0}^{n} p_2^i \tag{3.3}$$

$$\begin{cases} 31 = \sum_{m=0}^{4} 2^m = 1 + 2^1 + 2^2 + 2^3 + 2^4 \\ 31 = \sum_{n=0}^{2} 5^n = 1 + 5^1 + 5^2 \end{cases} \tag{3.4}$$

Taking into consideration the high computational cost of the conjecture's survey, the grid computing model is ideal to be used, independently if the brute force approach or an approach based on EA is executed. The grid computing is more suitable jobs completely independent, or with a very low level of coupling. The work units of Bateman problem are free of any liaison among them.

The scientific production of this section has been supported by the grid infrastructure described at section 2.3.1.

In order to obtain the list of prime numbers, the project Primer-Numbers.org is used. This project supplies files with lists of prime numbers.

3.6.1 Brute Force Approach

During the years 2007-2008, a systematic survey of the Conjecture of Bateman is performed (Fig. 3.2). The prime numbers are grouped in intervals of 100,000, except for 1 to 100,000 which is divided in two segments: from 1 to 50,000 and from 50,000 to 100,000. The purpose behind this fact is to produce jobs with an ideal CPU-time consumption for the grid computing. On the other hand, the maximum power reachable for the sums of powers series is established at 30. This survey covers all prime numbers lower than 15,000,000 and powers until 30; and the prime numbers until 100.000 for powers until 60.

New coincidences of the Conjecture of Bateman was not discovered during this brute force survey.

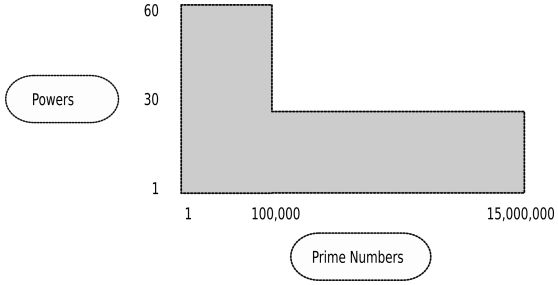


Figure 3.2: Solutions' space explored for the Bateman Conjecture with the brute force survey.

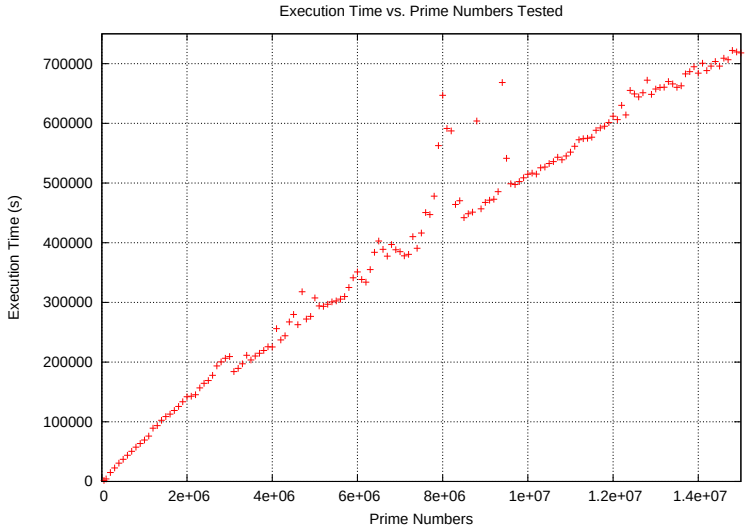


Figure 3.3: Growing up of aggregated execution time for the exploration of all prime numbers lower than a threshold.

This survey tackles several difficulties; for example, it shows a very high CPU-time consumption. This consumption increases with the power and the range of primes explored. Furthermore, the number of jobs and the CPU-time consumption necessary to finish a series, and therefore to close a threshold, also increase (Fig. 3.3).

In this scenario, the competence among the Bateman jobs with others to capture a free worker node in the grid produces a dispersion of total CPU-time. Finally, due to the higher number of jobs for each series; there is an extra risk of failure of a job which ruins the whole series.

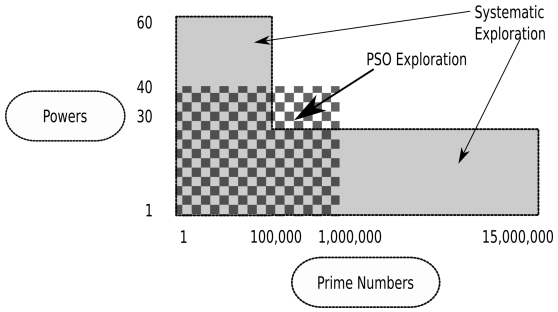


Figure 3.4: Solutions' space explored for the Bateman Conjecture with PSO and brute force approaches.

3.6.2 Particle Swarm Optimizer Approach

In order to overcome the barriers appeared in the systematic exploration of the Conjecture of Bateman, more intelligent methods should be utilised. As an alternative to the systematic exploration, an EA would provide optimal solutions avoiding the exploration of the whole solution space. A priory any kind of EA would be suitable.

Following the methodology of the PSO algorithm adapted to the Conjecture of Bateman, a initial population, randomly created, of particles (candidate solutions) is produced. Each particle of this population has four parameters: the two powers and the two primes. Additionally, for each parameter an associated velocity is settled.

Moreover, as fitness function, the absolute value of the difference between the two sums of power series is chosen, Eq. 3.5. The optimal solution will have a null fitness, nevertheless the suboptimal solutions can reach any other positive value of the fitness.

$$Fitness = |\sum_{i=0}^{m} p_1^i - \sum_{j=0}^{n} p_2^j| \tag{3.5}$$

Firstly, a production aimed to tuning the behavioural parameter of PSO is performed. Diverse values of population size, V_{max}, c_1 and c_2 are tested with 50 jobs. The most suitable values for those parameters are $c_1 = c_2 = 2$, $V_{max} = 5$ (Eq. 3.1) and 1,000 particles; and they are used in the final production.

During this final production, the area explored is the primes lower than 1,000,000 and the powers until 40; with 400 jobs and 345.88 CPU-hours invested. In Fig. 3.4, the whole solution space explored by the both techniques is shown. Unfortunately, no new coincidences was discovered during this exploration.

3.7 Conclusions

Along this chapter, specific aspects of the exploitation of EAs have been studied. For instance, an update of the impact of the choice of the RNG over the performance of some very popular EAs has been produced. From this study, it can be concluded that the performance of these EAs is affected in a diverse degree provided that high-quality RNGs are used.

On the other hand, studies about the use of multipopulation approach to improve the performance of the PSO have been executed. Besides, the relative performance of diverse variants of this algorithm has been also studied. Finally, an example of application of PSO to the resolution of a complex problem (the search of more coincidences in the conjecture of Bateman) has been presented.

Chapter 4

GPU Computing

4.1 Introduction

GPU computing has become a very popular platform for scientific computing due to its capacity to fulfil the computational requirement of mid-sized groups, at the same time that it maintains an affordable budget. However, the exploitation of GPU for scientific computing is burdened with a non-trivial learning curve for the practitioners and adaptation process of the code.

In this chapter some exploratory works in the area of GPU computing are presented. They include the study of optimum data layout for accelerating the evaluation of benchmark functions on GPU, or the adaptation of PSO to GPU.

4.2 Related Work

4.2.1 Related Work for the Implementation of Evolutionary Algorithms in GPU and Analysis of its Behaviour

During the last years, a plethora of works have covered diverse topics related with the adaptation of EAs to GPU architecture. Some few examples of this kind of works are: speeding-up the optimization of 0/1 knapsack problem with genetic algorithm [79], dealing with the mapping of the parallel island-based genetic algorithm [78], or an example of cellular genetic algorithm [96]; also there are examples in accelerating learning systems [34]; and specifically, examples of general-purpose parallel implementations of PSO in GPU [69, 70, 82, 103, 104]; and implementations of Differential Evolution [25, 30].

The study [103] is the closer one to the work presented in Section 4.3. In this article similar benchmark functions and the same EA (PSO) are employed. The main difference is the dimensionality employed. In this study the dimensionality ranges from

50 to 200; whereas in our study a higher dimensionality has been used —20,000— in order to check the behaviour for extremely large-scale problems.

Although the most frequent topics are the adaptation of EAs to GPU, other studies cover theoretical aspects of optimisation problems. An example of this kind of work is the study of the models of parallel EAs in GPU [55], where three basic models for adaptation of EAs to GPU hardware are presented.

However in the bibliography reviewed, there are not examples of analysis of the capacity of different implementation models to accelerate the execution of PSO and other EAs in GPU. The current work covers this kind of in-depth analysis of the occupancy of the blocks and its impact on the final performance.

4.2.2 Related Work for the Effect of Data Layout on GPU Evaluation Time

Evolutionary computing has taken a great advantage of the appearance of GPU. Many works are published every year by describing how evolutionary problems are accelerated by transferring partially or totally the execution to GPU. Generally these works focus on the problem and how to accelerate it. This purpose is achieved through a wide variety of improvements, but in most of the cases the data layout is fixed at the beginning of the problem without testing alternative layouts. Some few examples of this kind of works have been presented in the previous section.

Generally, authors modify the parameters of the algorithm, such as: population size, mutation or crossover rates in genetic algorithms, or maximum velocity in PSO; and GPU configuration parameters, such as: number of blocks or number of threads per block, or the distribution of data between global memory, shared memory and registers. However, the initial data layout remains mostly unaltered along the problem. Probably the initial data layout is the most intuitive for the authors. This avoids exploring other configurations. For this reason, works addressing similar issues have not been found.

4.3 Implementation of Evolutionary Algorithms in GPU: PSO as Case Study

Many scientific and technical problems have improved their performance through the use of GPU cards. GPU allows accelerating the execution of these problems, including those dealing with Evolutionary Algorithms (EAs) to optimize continuous functions.

This section presents a study of the improvements in performance when evaluating EAs in GPU[1]. This study analyses the variation of performance under diverse con-

[1] All the works in this section have been obtained by using a GTX295 (section 2.3.3.1).

figurations; such as: which functions are suitable to be evaluated in GPU and which are not, and variations of the problem size, population size and dimensionality of the individuals.

The study of high-dimensional problems has to face with long-execution times, and therefore, difficulties to reach high-statistics. Large-scale optimisation problems require powerful computing platforms in order to reduce the time consumption. Thus, accelerating with GPUs, more tries can be executed by time unit. As a consequence, the main drawback of high-dimensional optimization problems with EAs, the large increment of execution time is overcome.

4.3.1 GPU-Based Evaluation to Accelerate Particle Swarm Algorithm

4.3.1.1 Parallel Models of Evolutionary Algorithms

For non-trivial problems, to execute the reproductive cycle of a simple EA with long individuals and/or large populations requires high computational resources. In general, evaluating a fitness function for every individual is frequently the most costly operation of the EA.

In EAs, parallelism arises naturally when dealing with populations, since each of the individuals belonging to, it is an independent unit. Due to this, the performance of population-based algorithms is specially improved when running in parallel. Parallel Evolutionary Algorithms (PEAs) are naturally prone to parallelism, since most of the operations can be easily undertaken in parallel.

Basically, three major parallel models for EAs can be distinguished [2]: the island a/synchronous cooperative model, the parallel evaluation of the population and the distributed evaluation of a single solution.

The parallel evaluation of the population is recommended when the evaluation is the most time-consuming. This model has been selected in our adaptation of the application to GPU. The parallel evaluation follows a master-worker model. The master operation lies in CPU, and it is: the transformation of the population, as well as the generation of the initial random population.

Otherwise, the evaluation of population is performed in GPU (worker). When the particles need to be evaluated, the necessary data are transferred to GPU. After the evaluation, the results return back to CPU, and the CPU-code part regains the control. In the next cycle, the evaluation of the population is allocated again in GPU to be evaluated.

4.3.1.2 Production Setup

In order to stress the capacity of GPU, the functions Schwefel's Problem 1.2 (Eq. 4.1) and Rosenbrock function (Eq. 4.2) have been used. These functions have a global minimum at $\vec{0} = (0_1, 0_2, \ldots, 0_D)$. The main features of these functions are: both are fully-non-separable and the highest CPU-time consumption is the evaluation. They have been used in the editions of CEC 2008 and 2010 Special Session and Competition on Large-Scale Global Optimization (CEC competition) as benchmark functions [94, 95].

$$f_{Schwefel's\ \ Problem\ 1.2} = \sum_{i=1}^{D}(\sum_{j=1}^{i} x_j)^2 \tag{4.1}$$

$$f_{Rosenbrock} = \sum_{i=1}^{D-1} 100 \cdot [(x_i^2 - x_{i+1})^2 + (x_i - 1)^2] \tag{4.2}$$

Concerning the number of tries, in all cases 15 tries are executed. As pseudorandom number generator, a subroutine based on Mersenne Twister has been used [59].

4.3.1.3 Adaptation of PSO Algorithm

The invocation of the kernel is made with a bi-dimensional grid of blocks and allocating all threads of each block in a one-dimensional array. Regarding the grid of blocks, the dimension in y-axis represents the number of particles and the number of blocks in x-axis is made in such way that they can allocate all the dimensions of a particle. Taking into account that each block contains 512 threads for GTX295 (section 2.3.3.1), for particles with 1,000 variables two blocks are necessary, for particles with 5,000 variables 10 blocks are necessary, and so on. This distribution of data in the bi-dimensional grid of blocks eases the calculation of fitness, being possible a huge parallelization of the process.

This particular distribution of data has an important extra value. It is suitable for any population-based EA, easing the manipulation of the data and its coupling to any other population-based EA. The final result is an independent piece of software, the kernel where the function is evaluated, easily pluggable to any other population-based EAs.

4.3.1.4 Study of the Rosenbrock Function

A priory, it is foreseeable that any fully-non-separable function will be suitable for parallel evaluation in GPU. However the tests performed with the Rosenbrock function

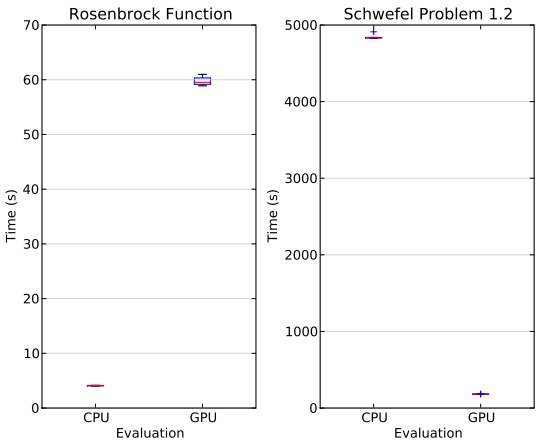


Figure 4.1: Comparative box plots —15 tries— for CPU and GPU codes of execution time for Rosenbrock function —left— and Schwefel Problem 1.2 —right—, and dimensionality 20,000 and 20 particles.

does not show any speedup, being the execution time for GPU version longer than the execution time for CPU version (Fig. 4.1 left). This is due to the low number of particles used.

In order to clarify the behaviour of both functions when executing the evaluation in CPU or in GPU, an in-depth analysis of the transfer and execution of the kernel is performed.

In the tests performed with both functions, the time of copying data from CPU to GPU memory, executing the kernel (evaluation) and copying back the data from GPU to CPU memory have been measured (Table 4.1). As it can be seen, for equal dimensionality the transfering-data time is similar for both functions. On the contrary, the kernel execution time is quite different, being much higher for Schwefel's Problem 1.2 than for Rosenbrock function. This demonstrates that the Rosenbrock function is to light to profit of the inherent parallelism of GPU architecture, at least for the configurations checked. Oppositely, the double sum that composes the Schwefel's Problem 1.2 obtains a definite speedup when executing in parallel.

Comparing the complete sequence of moving data between memories and the kernel execution in GPU with the execution time of the sequential evaluation (Table 4.2), remarkable differences arise. For the Rosenbrock function, the values are similar wherever the evaluation is performed, in CPU or in GPU, whereas for the Schwefel's Problem 1.2 the evaluation for CPU version takes much longer than for GPU version.

Table 4.1: Mean Execution Time ($10^{-6}s$) for transfer data between CPU and GPU memories and kernel execution in Schwefel Problem 1.2 and Rosenbrock function for dimensionalities 1,000 and 10,000; and 20 particles.

Function	Dimensionality	Transfer CPU to GPU	Kernel Execution	Transfer GPU to CPU
Schwefel's Problem 1.2	10^3	0.0888	0.555	0.1160
Rosenbrock	10^3	0.0905	0.0723	0.0917
Schwefel's Problem 1.2	10^4	0.4237	37.5213	0.4774
Rosenbrock	10^4	0.4414	0.3041	0.4665

Table 4.2: Mean Execution Time ($10^{-6}s$) for one evaluation in CPU and in GPU in Schwefel's Problem 1.2 and Rosenbrock function for dimensionalities 1,000 and 10,000; and 20 particles.

Function	Dimensionality	Evaluation on CPU	Evaluation on GPU
Schwefel Problem 1.2	10^3	18.976	0.761
Rosenbrock	10^3	0.085	0.255
Schwefel Problem 1.2	10^4	1,215.520	38.422
Rosenbrock	10^4	0.751	1.212

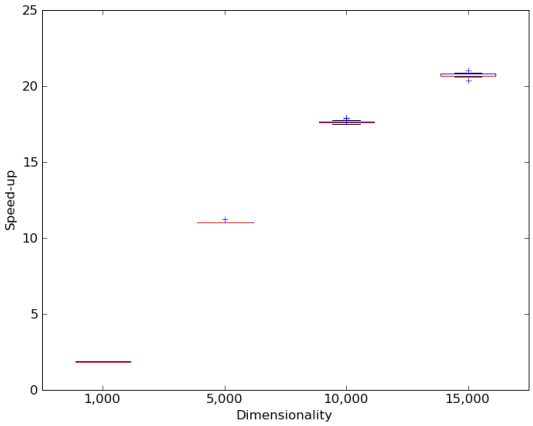


Figure 4.2: Comparative box plots of speedup and each dimensionality 1,000; 5,000; 10,000; 15,000 and 20,000, and 20 individuals for the Schwefel's Problem 1.2.

4.3.1.5 Study of Schwefel's Problem 1.2

The improvement in the execution time of a parallel application can be evaluated by using the speedup, defined as $S = \frac{T_{inCPU}}{T_{inGPU}}$, where T_{inCPU} is the execution time of the sequential version and T_{inGPU} is the execution time for the GPU version. In this section, the speedup achieved for diverse problem sizes is presented.

As it can be seen at Fig. 4.2, the speedup increases as much as the dimensionality of the problem increases. At Fig. 4.2 box plots are built with the values of speedup, corresponding to the dimensionality ranking from 1,000 to 20,000. The cases with higher dimensionality show a better harness of the parallelism capacity of GPU. As much as the number of variables grows, the capacity of GPU to map data to threads increases the parallel capacity of the hardware.

The application of the Kruskal-Wallis test and the Wilcoxon signed-rank test with Bonferroni correction indicates that all the differences in the speedup are significant —$\alpha = 0.05$—.

4.3.1.6 Varying Population

Focusing on Schwefel's Problem 1.2, the speedup when varying the population size is presented at Fig. 4.3. This corresponds to population sizes ranking from 10 to 50 individuals, and dimensionality in all cases of 10,000. As it can be appreciated, the speedup goes up when the population augments in the range from 10 to 30. However, for further increments, it does not produce any extra speedup.

In all population-based EAs, the parallelism arises naturally. For example, for PSO each particle is independent from the other particles. Therefore, when the population

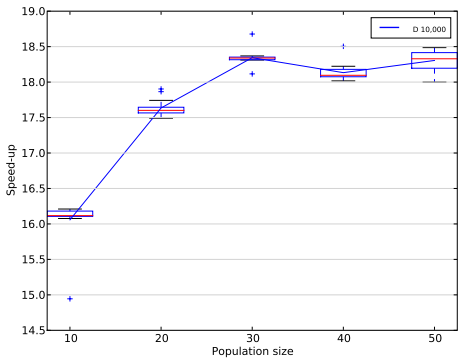


Figure 4.3: Comparative box plots of speedup for diverse population size and dimensionality 10,000 for the Schwefel's Problem 1.2.

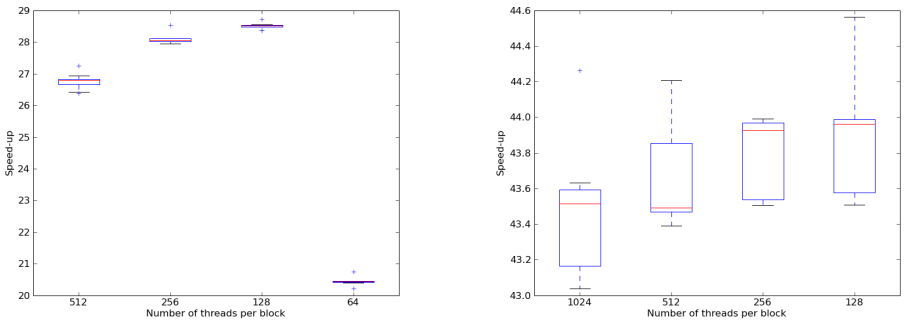
size is small, any increment will help to mask the latency of the memory access, through the allocation of other calculations without data dependencies. In this case, a more efficient usage of the parallelism of the hardware is produced. Oppositely, when the population size is larger, any increment should be serially scheduled, and as a consequence, it does not produce any improvement in the performance.

The application of the Kruskal-Wallis test to the data of the speedup when varying population size indicates that the medians of all of the groups are no-equal; and the Wilcoxon signed-rank test with Bonferroni correction demonstrates that all the differences are significant —$\alpha = 0.05$—; except for the differences between population sizes 30, 40 and 50.

4.4 Analysis of the Behaviour of Evolutionary Algorithms in GPU: PSO as Case Study

Many features in the GPU programming produce a reduction of the execution time. In this section, the effect of the occupancy when the PSO algorithm optimizes the Schwefel problem 1.2 (Eq. 4.1), is studied. Equally to the previous section, the works described in this section have been performed by using a GTX295 (section 2.3.3.1) and a C2050. This last card is similar to C2075 described at section 2.3.3.2 but it only incorporates 3 GB of global memory instead of the 6 GB of the C2075.

Concerning the adaptation process, it is important to underline the importance of the matching of data and parallel processing elements. So that, the same data layout that described in Section 4.3.1.3 is used in this study. The PEA model and the configuration of PSO is identical to those described in the previous section.



(a) Speed-up in GTX295

(b) Speed-up in TESLA C2050

Figure 4.4: Comparative box plots of speed-up in GTX295 and TESLA C2050 for diverse configurations of threads per block —100%, 50%, 25% and 12.5%— and 15 tries per configuration for the Schwefel's Problem 1.2.

4.4.1 Results and Analysis

The configuration employed in this study has been selected requiring: to stress the potential capabilities of the GPU cards, to stretch out as much as possible the data on the threads and blocks and to be representative of large-scale problems in continuous optimization. In all cases, the population size is 20 individuals, the dimensionality of the search space is 20,000 and the number of cycles is 1,000. Finally, the occupancies range from 100% to 12.5%. For each configuration, 15 tries are executed.

In Table 4.3 and Fig. 4.4(a), the speed-ups produced in the GTX295 by the reduction of the occupancy of the blocks —100%, 50%, 25% and 12.5%— are presented. The analysis of these data shows that the progressive decrement of the occupancy in the block produces an increment in the speed-up higher than 1.7 in percentage. This improvement becomes less relevant for further reduction of the occupancy: 1.3 from 100% to 50%, 0.5 from 50% to 25%. Nevertheless, this improvement disappears when the occupancy falls below 25%. In this case, a strong degradation of the performance for small occupancy emerges from the experimental data.

In Table 4.3 and Fig. 4.4(b), the speed-ups produced in the TESLA C2050 by the reduction of the occupancy of the blocks are presented. Alike to the GTX295, in the TESLA C2050 an increase of the speed-up appears when the occupancy of the blocks is reduced from 100% to 50%, and for further reductions. However, the increment in TESLA C2050 is less significant than the equivalent reduction observed in the GTX295 when the occupancy reduces.

On the contrary to GTX295, in TESLA C2050 a degradation of the performance is not observed when the occupancy diminishes below 25%. Nevertheless, the data are

Table 4.3: Mean speed-up and standard deviation —after 15 tries per configuration— in GTX295 and TESLA C2050 versus CPU codes for diverse number of threads per block for the Schwefel's Problem 1.2.

GTX295		Occupancy	TESLA C2050	
Threads per Block	Speed-up		Speed-up	Threads per Block
512	26.76±0.20	100%	43.43±0.30	1024
256	28.09±0.13	50%	43.64±0.23	512
128	28.54±0.14	25%	43.81±0.20	256
64	20.44±0.10	12.5%	43.90±0.26	128

so close that the significance of the difference has to be checked by statistical methods. The application of the Wilcoxon signed-rank test with the Bonferroni correction to the TESLA C2050 results demonstrates that the differences are not significant. On the contrary, the numerical results of the speed-up of GTX295 demonstrate that the differences are significant.

4.5 Effect of Data Layout on GPU Evaluation Time

GPUs are able to provide a tremendous computational power, but their optimal usage requires of the optimization of memory access. The many threads available can mitigate the long memory access latencies, but this usually demands a reorganization of the data and the algorithm to reach the performance peak. The addressed problem is to know which data layout produces a faster evaluation when dealing with population-based evolutionary algorithms optimizing non-separable functions. This knowledge will allow a more efficient design of evolutionary algorithms. Depending on the fitness function and the problem size, the most suitable layout can be implemented at the design phase of the algorithm, avoiding later costly code or data layout redesigns.

In this work, diverse non-separable functions, such as Rosenbrock and Rana functions, and data layouts are evaluated. The implemented layouts cover the main techniques to maximize the performance: coalesced access to global memory, intensive use of on-chip memory: shared memory and registers, and variable reuse to minimize the global memory transactions.

The works described in this section have been performed by using a GPU C2075 (section 2.3.3.2).

4.5.1 Strategies Tested

4.5.1.1 Strategy 1: Allocation of one Individual per Thread on Registers.

In the two first strategies (S1 and S2) presented, each individual is handled by a single thread. Both strategies exploit the fast access to on-chip memory. In S1, registers are used to accumulate the coordinates of the individuals, whereas in S2 shared memory is used instead of registers. In both cases the intermediate fitness values are accumulated on shared memory. Other tests, where the registers are used to accumulate the intermediate fitness values have been also performed. However, no-significant modifications in the execution times are produced.

Besides, in both strategies the input array is ranged as a sequence of individuals: firstly all the coordinates of the first individual, then all the coordinates of the second individual, and so on.

In S1, each single thread executes a for-loop sequentially over the coordinates of one individual. At the beginning of the sequence, some values are necessary to calculate the first partial fitness. Next, one coordinate is read ahead to produce a new part of the fitness which is gathered over the previous value of the accumulated partial fitness.

In each step, only a new coordinate is downloaded from global memory to on-chip memory (register for S1 and shared memory for S2). The remaining values, necessary to calculate the partial fitness, have been loaded and stored on the on-chip memory in the previous steps. This schema saves global memory accesses, replacing them by faster on-chip memory accesses. The number of saved accesses per cycle and individual depends on the number of necessary values to calculate a partial fitness value.

A priori this strategy would be more suitable for the evaluation of large populations, because each individual evaluation is independent of the others, and therefore, a large parallelism degree is reached. For small populations of high-dimensional individuals, this strategy is penalized because few threads are mobilized (as threads as individuals), and therefore, few streaming multiprocessors (SM) are active.

Regarding the limitations of the approach, it can be foreseen that this strategy is constrained by the total number of threads that the GPU can allocate. However, this number is high enough to allow fast evaluation of large size problems. A second limitation could be the consumption of on-chip memory when evaluating fitness functions requiring many coordinates of the individual. For example, Rosenbrock function (Eq. 4.2) only requires two coordinates, whereas the tailored functions require two (Eqs. 4.4 and 4.5) and four (Eqs. 4.6 and 4.7) respectively. In this case, the thread-block has to allocate a high number of registers to hold all of the necessary values to calculate the partial fitness.

Finally, an important drawback is envisaged for the S1 and S2 strategies. The conjunction between the data layout (a sequence of individuals) and the sequential access to the dimensions of each individual creates a stride pattern access to the global memory, which is pernicious for the performance.

Global memory is always read in chunks of 128 bytes (length of a cache line) by 32 consecutive threads[1]. If a part of a cache line reading involves no-necessary data for the calculations or simply not all necessary data for the calculation for the 32 threads fill the cache line, then global memory bandwidth is being wasted. As a consequence, the bus transactions are increased to complete the necessary data for the calculations.

In S1 and S2, when dimensionality is equal or higher than 32, due to the disposition of the individuals, only one float is valid (from the 128 bytes read) for the 32 threads involved. Therefore, to read the 32 dimensions for the 32 consecutive threads, 32 bus transactions are necessary. As can be appreciated, most of the memory bandwidth is misused.

When dimensionality is lower than 32, more that one float is valid in each bus transaction, but still a part of the global memory bandwidth is underused. If dimensionality is 16, then only 2 data are valid in the transaction; and, if dimensionality is 8, then only 4 data, and so on. This cache trashing severely degrades the performance. In the following strategies: S3 and S4, two alternatives to circumvent the stride access are presented[2].

In spite of the disadvantages drawn, it is important to emphasize the capacity of this strategy to cope with extremely large problem sizes. For this reason, and because it is one of the most intuitive layout when adapting evolutionary problems to GPU, this strategy is considered to be evaluated.

4.5.1.2 Strategy 2: Allocation of one Individual per Thread on Shared Memory.

This second strategy is very similar to the previous one but using shared memory to support the stencil instead of the registers. When implementing stencil in shared memory, some accesses to global memory are already saved. Similarly to S1, some necessary coordinates to calculate the partial fitness have been previously downloaded

[1] It exists a particular case in Fermi architecture which modifies this feature. In pre-Fermi architecture the available memory transaction segment sizes are: 32, 64 and 128 bytes. The selected value depends on the amount of memory needed and the memory access pattern. The selection is automatic in order to avoid wasting bandwidth. In Fermi architecture, the memory transaction segment size follows a different rule. When L1 cache memory is enable, the hardware always issues segment transactions of 128 bytes, the cache-line size; otherwise, 32 bytes segment transactions are issued. In our study, default configuration of L1 is enable in all numerical experiments.

[2] In Fermi architecture L1 and L2 cache memory can mitigate partially this penalization.

and stored from global memory into the shared memory. In order to hold these values on shared memory, some arrays have to be defined. These arrays have the same role as the variables defined in registers in S1.

Both S1 and S2 handle the individuals in parallel, but sequentially their dimensions. A priori this is more suitable for large populations than for small populations of high-dimensional individuals.

This strategy benefits from the absence of divergence in the warps, as well as a significant reduction of global memory transactions. On the contrary, it suffers from limitations steamed from the shared memory consumption, similarly to the register consumption in S1[1].

Finally, the most important drawback in S1 and S2 is the non-coalesced access to global memory. In the next strategies two alternative layouts are presented. They introduce the appropriate modifications in order to get coalesced access to global memory.

4.5.1.3 Strategy 3: Allocation of one Individual per Thread-Block on Share Memory with Coalesced Access to Global Memory and Atomic Operations.

In this third strategy, the main difference holds on how the individual is managed: each individual is handled by a single thread-block, instead of a single thread as in the previous strategies. This layout forces to select the number of threads per block equal to the dimensionality of the individual. This constraint marks a weakness in the strategy: the maximum threads per block allowed in the GPU[2] is the maximum dimensionality that this strategy can evaluate. Even though the highest allowed dimension for the individuals is the maximum threads per block, a reduction of this constraint is expected by the progressive increment of this value in future GPU architectures.

Although there exist techniques to deal with individuals with larger dimensionality than the maximum number of threads per block[3], these techniques are not considered in this work.

Since the individuals are disposed sequentially (as in S1 and S2), each thread accesses to some dimensions of an individual to calculate the partial fitness. These dimensions are accessed in a coalesced mode: consecutive threads access to consecutive

[1] In the numerical experiments, the configuration with maximal shared memory (48KB) is used.

[2] The maximum number of threads per block in pre-Fermi architecture is 512, whereas in Fermi architecture is 1024.

[3] The techniques are able to deal with larger individuals than the maximum number of threads per block are mainly two: to spread out the individual over more than a single thread-block including halo coordinates and later to use global memory to gather the partial fitness values of each individual, or to treat the individual in chunks of the maximum number of threads per block.

global memory directions. After getting the necessary data, each thread calculates a chunk of the fitness values in parallel, and stores it in the shared memory for later reduction.

Oppositely to the previous strategies, in S3 the calculation of the fitness function is executed in parallel, not only among the individuals, but also for the partial fitness of each individual. This is an advantage in relation to the previous strategies.

For the final reduction of the partial values of the fitness of the individual, two alternatives are considered:

- Folding of the array containing the partial fitness by the half successively, up to accumulate the addition of all the values over the first element of the array. This reduction technique has a relevant advantage in the high-degree of parallelism reached; however it also has an important drawback: it is only valid for 2^n dimensional individuals. Individuals with different dimensionality from 2^n have to be filled in with null values to reach the next 2^n value. Whereas the objective is to present the most general implementation with the widest applicability, this reduction technique is dismissed.

- Use of atomic operations, concretely *atomicAdd()*. Although the atomic operations should produce a degradation of the performance, this can be mitigated by parallelizing the reduction with atomic operations for the individuals on shared memory.

4.5.1.4 Strategy 4: Allocation of one Individual per Thread on Registers with Coalesced Access to Global Memory.

In this last strategy, the S1 strategy is modified to mitigate the penalty of the stride global memory access. In order to overcome the non-coalesced access, a transposition of the input data is performed previously to the transfer to global memory. This transposition modifies the disposition of the elements from a sequence of individuals to a sequence of the same coordinate of the individuals: firstly the first coordinate of all individuals, then the second coordinate of all individuals, and so on.

By using the present disposition, 32 consecutive threads access to 32 consecutive floats (128 bytes) to obtain the value of a particular dimension of 32 individuals. As a result, the access to global memory becomes coalesced and the length of the cache line is fully used.

Alike in S1, in S4 a single thread deals with a individual. Therefore, the individuals are handled in parallel but their dimensions are still sequentially managed. The other benefits and drawbacks of the S1 strategy are still valid for S4.

By comparing S3 and S4, two different ways to gain coalesced access to global memory are implemented. In S3 the calculations are modified to adapt them to the data layout; whereas, in S4 the modification is performed over the data layout.

4.5.1.5 Sequential Evaluation

For the sake of completeness, a purely sequential evaluation is also implemented, and its results are used in the comparisons. Considering the non-negligible cost of the data transfer between CPU and GPU, for reduced problem sizes, CPU evaluation is expected to be faster than the operations' set: transfer from CPU to GPU of population data, evaluation of individuals, and retrieval of the fitness data from GPU to CPU. Comparisons with sequential evaluation discern which problem sizes are more suitable for GPU or CPU evaluation.

4.5.2 Benchmark Functions

In general, the difficulty to find high-quality solutions and the evaluation time of the fitness functions in evolutionary computing increase with the dimensionality of the individuals and the population size. However, there are other relevant factors related with the *separability* of the fitness function. A function can be declared as *separable* if the variables are independent. This kind of problems is easier to solve by using evolutionary algorithms since a variable can be optimized while the rest is kept unchanged. By iteratively using this mechanism, all of the variables can be easily optimized. Even more, separable functions are often readily solved by local search methods. This is the main reason why some authors argue that they should not be incorporated to test suites [100].

On the contrary, the so-called *non-separable* fitness functions cannot implement this mechanism, since the variables are not independent. A non-separable function is called *m-non-separable* if at least m variables are not independent. In the extreme case, where neither variable is independent of the others, the function is called *fully-non-separable*.

In evolutionary computing, specially in the benchmark functions used in continuous optimization contest [94, 95], there exist well-known fully-non-separable functions as: Rosenbrock (Eq. 4.2) [88] or Rana function (Eq. 4.3).

$$\begin{aligned} f_{Rana} &= \sum_{i=1}^{D-1} (x_{i+1} + 1.0) \cdot cos(t_2) \cdot sin(t_1) \; + \; cos(t_1) \cdot sin(t_2) \cdot x_i \\ where\ t_1 &= \sqrt{|x_{i+1} + x_i + 1.0|}\ , and\ t_2 = \sqrt{|x_{i+1} - x_i + 1.0|} \end{aligned} \tag{4.3}$$

On the other hand, functions specially designed for this work are also employed. These functions allow a finer-grained control over the computational intensity of the benchmark function. For example, expensive and non-expensive calculation functions (Eqs. 4.4, 4.5) will allow an in-detail characterization of the behaviour of the data layouts.

These functions have been constructed simulating the Rosenbrock function: incorporating a fixed number of dimensions to calculate each term of the fitness function. The recommendations for constructing this type of functions have been followed in order to compose these new non-separable functions [100].

$$f_{2-light} = \sum_{i=1}^{D-1} (x_i + x_{i+1})^2 \tag{4.4}$$

$$f_{2-heavy} = \sum_{i=1}^{D-1} log\sqrt{log(\frac{1}{\sqrt{x_i}}) + log(\frac{1}{\sqrt{x_{i+1}}})} \tag{4.5}$$

$$f_{4-light} = \sum_{i=1}^{D-3} (x_i + x_{i+1} + x_{i+2} + x_{i+3})^2 \tag{4.6}$$

$$f_{4-heavy} = \sum_{i=1}^{D-3} log\sqrt{log(\frac{1}{\sqrt{x_i}}) + log(\frac{1}{\sqrt{x_{i+1}}}) + log(\frac{1}{\sqrt{x_{i+2}}}) + log(\frac{1}{\sqrt{x_{i+3}}})} \tag{4.7}$$

4.5.3 Results and Analysis

4.5.3.1 Rosenbrock Function

In this section, the performance achieved by each strategy when evaluating the Rosenbrock function is discussed. The execution times (mean and standard deviation) of the strategies for several configurations are presented in Table 4.4. The problem sizes have been selected for being representative of the state-of-the-art problem sizes, and mapping the transitions between the most suitable strategies. Most of the configurations have been selected with population equal to dimensionality. Where an unequal configuration is chosen, the objective is to map the transition between two best strategies, narrowing the uncertain border.

For each problem size, diverse threads-per-block configurations are executed. From the execution times obtained, the best results for each strategy are retained and presented. For example, for 100×100 configuration, two configurations of threads per block are tested: 32 and 64; whereas, for 2,000×2,000, four configurations are tested: 128, 256, 512, and 1024 threads per block. In order to fairly compare, for the sequential

strategy equal number of executions are produced, and then retaining the best one. Furthermore, for each case a total of 20 tries are executed.

On the other hand, input data are randomly generated in the range (0,1). Identical input files are used for all the executions of a particular configuration.

The results in Table 4.4 provide information about which strategy is the most suitable. Mildly speaking, sequential evaluation is the fastest strategy for the evaluation of the Rosenbrock function if the population and the dimensionality are up to 1,000. For these configurations, the cost of data transfer between CPU and GPU penalizes the GPU implementations. This penalization is not balanced by a parallel, and therefore faster, evaluation of the population.

However, when increasing problem sizes, the evaluation time of the sequential strategy dramatically grows, being no-longer the best option. For problem sizes equal or larger than 1,500×1,500, the penalization due to the data transfer between CPU and GPU is counteracted by a faster evaluation.

When dealing with large problem sizes, the S4 strategy outperforms all of the other strategies. In this strategy, the coalesced access to data in global memory produces an efficient usage of the bandwidth. On the other hand, the a priori flaw of the sequential treatment of the coordinates of each individual in S4 becomes a robust feature when evaluating very high-dimensional individuals. In this case the on-chip memory consumption is still moderated, because few variables per individual have to be simultaneously allocated on registers.

On the contrary, for these very high-dimensional individuals, the maximum number of threads per block becomes a major limiting factor in S3. This does not allow to evaluate individuals larger than 1024 in Fermi architecture (512 for pre-Fermi architecture). Even more 1,000×1,000 and higher configurations are not evaluable by S3 due to the depletion of the shared memory.

This depletion of the shared memory is also the reason why 8,000×2,000 configuration is not allocatable on S2 strategy. This demonstrates that strategies implementing registers as support for the intermediate data are more robust than strategies only implementing shared memory.

In Fig. 4.5, the results of Table 4.4 have been analysed using Linear Support Vector Machine[1] (LinearSVM) [24]. The use of SVM allows the knowledge extraction from large numerical data sets. Through building a model with SVM, patterns in data can be inferred, being the model more comprehensible than the numerical output [40]. If data are linearly separable, then LinearSVM is the simplest SVM classification model.

[1]It is assumed that the data are linearly separable.

Table 4.4: Mean execution time and standard deviation for Rosenbrock function depending on data layout.

Pop. × Dim.	S1	S2	S3	S4	Sequential
100×100	0.237±0.005	0.255±0.003	0.145±0.004	0.170±0.003	**0.026±0.001**
200×200	0.444±0.004	0.484±0.002	0.365±0.006	0.314±0.002	**0.098±0.003**
1,000×1,000	3.188±0.014	3.389±0.012	NAN	2.481±0.019	**2.432±0.041**
500×2,000	4.733±0.023	5.129±0.067	NAN	3.439±0.045	**2.422±0.029**
2,000×500	2.558±0.013	2.549±0.020	NAN	**1.962±0.071**	2.431±0.048
1,500×1,500	6.218±0.163	5.907±0.106	NAN	**4.651±0.018**	5.620±0.029
2,000×2,000	9.723±0.075	9.685±0.135	NAN	**7.533±0.110**	10.129±0.032
1,000×4,000	12.129±0.233	13.093±0.092	NAN	**9.539±0.253**	10.386±0.025
4,000×1,000	8.723±0.020	10.030±0.041	NAN	**6.633±0.067**	10.143±0.035
4,000×4,000	22.426±0.034	22.431±0.064	NAN	**22.425±0.049**	40.856±0.021
2,000×8,000	39.284±0.067	40.110±0.038	NAN	**30.369±0.101**	40.552±0.107
8,000×2,000	33.076±1.202	NAN	NAN	**26.601±0.069**	40.536±0.083

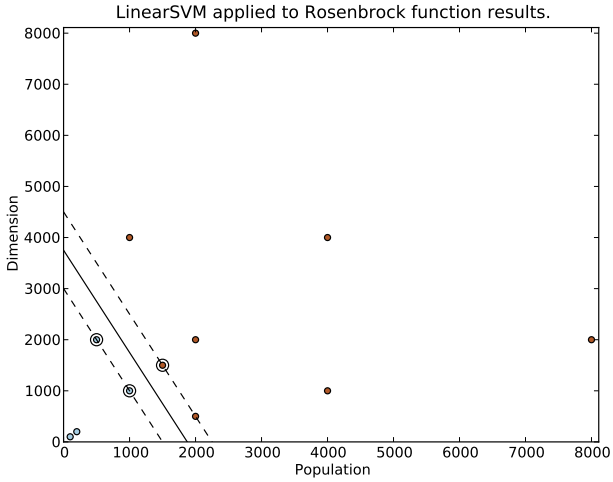


Figure 4.5: LinearSVM applied to the results of Rosenbrock function: the two data categories correspond to sequential evaluation for small and mid-size configurations, and S4 for large configurations. The support vectors for the first class (sequential evaluation) are 1,000×1,000 and 500×2,000; whereas for the second class (S4) is 1,500×1,500.

The application of LinearSVM to the numerical results allows showing the area (population size and dimensionality of individuals) where a particular data layout is the most suitable one, as well as the location of the borders between the areas.

Fig. 4.5 shows a clear distinction between the suitable configurations for sequential evaluation and the suitable configurations for GPU evaluation. The maximum-margin hyperplanes provide a visual estimation for the borders between the two data classes: sequential evaluation for small and mid-sized configurations, and S4 for large configurations. The support vectors pinpoint the configurations limiting the data categories: 1,000×1,000 and 500×2,000, for sequential strategy; and 1,500×1,500 for the S4 strategy.

Summarizing, decision-making about the most suitable evaluation strategy for the Rosenbrock function can be easily adopted by using Fig. 4.5.

4.5.3.2 F2-Light and F2-Heavy

Rosenbrock function constitutes an excellent example of non-separable function. However more functions are needed if the target is to characterize the behaviour of the problem based on how the data layouts are presented. For this reason, four more functions are constructed and their results analysed in this section and the following one.

The first function, $f_{2-light}$, is a Rosenbrock-like function with non-expensive calculation (Eq. 4.4), whereas the second one, $f_{2-heavy}$ (Eq. 4.5), is similar to the previous one, but replacing the very light inner calculations of the function by expensive operations from the computational point of view. With constructing these tailored functions, the computational intensity assigned to each thread is under-control. Similar to Rosenbrock, for fitness calculation both functions involve only two consecutive coordinates of the individuals.

The main difference between $f_{2-light}$ and $f_{2-heavy}$ is on the computational intensity supported by each thread to complete a partial fitness calculation. Because of this difference, each function results in a different relation between the access to data on global memory and the life-time of data on-chip memory. For the light version, the accesses to data in global memory per time unit are much higher than in the expensive version. For the expensive version, the calculation takes longer, and therefore, data should reside on the on-chip memory during a longer period. Consequently, the access to global memory per unit of time becomes more sparse.

The results of this study are presented using a LinearSVM plot: Fig. 4.6 for $f_{2-light}$, and Fig. 4.7 for $f_{2-heavy}$. The different results show the tendency for other non-separable functions when increasing its computational intensity.

The $f_{2-light}$ function has a lower computational intensity than Rosenbrock function, and for this reason, much larger problem sizes are the only suitable for GPU evaluation. The support vectors (Fig. 4.6) indicate that the limiting configurations are: 4,000×1,000 and 1,000×4,000 for the largest suitable configuration for CPU evaluation, and 4,000×4,000 for the lowest suitable configuration for GPU evaluation. Similarly to the Rosenbrock function, for the large-configurations case the appropriate strategy is S4.

By comparing the plots for $f_{2-light}$ and $f_{2-heavy}$, and both with Rosenbrock one, the most significant deviation appears for $f_{2-heavy}$. In this function, due to its high-computational intensity a strong reduction of the configurations where sequential evaluation is the most suitable option is observed. Only very small configurations, up to 16×16, are suitable for sequential evaluation. When increasing the problem size, the evaluation becomes more suitable for GPU in all cases. Moreover, since $f_{2-heavy}$ is more computationally intensive than $f_{2-light}$, the penalization of data transfer between CPU and GPU is largely counteracted by the faster evaluation of the population.

Considering only the GPU implementations, two dominance areas appear in the plot[1] (Fig. 4.7). In the inner one, the most suitable strategy is S3, whereas in the outer

[1]In the original implementation, SVM is only available for binary classification. Although there exist multiclass SVM implementations, in this work two simple and consecutive binary classifications

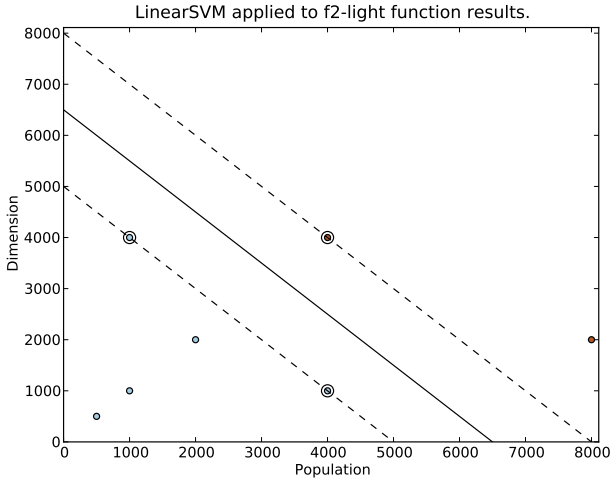


Figure 4.6: LinearSVM applied to the results of $f_{2-light}$ function: the two data categories correspond to sequential evaluation for small and mid-size configurations, and S4 for large configurations. The support vectors for the first class (sequential evaluation) are 4,000×1,000 and 1,000×4,000; whereas for the second class (S4) is 4,000×4,000.

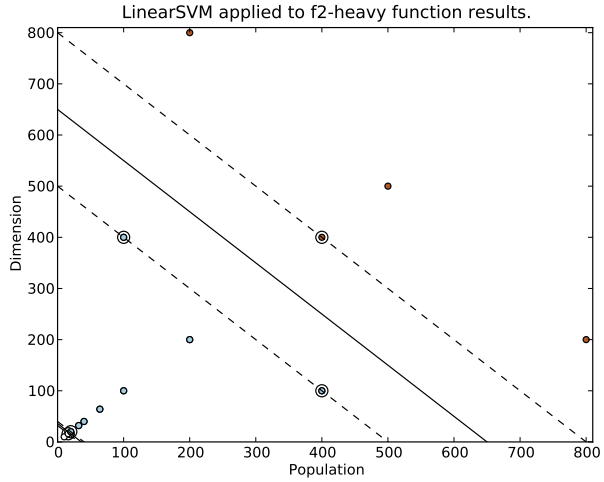


Figure 4.7: LinearSVM applied to the results of $f_{2-heavy}$ function: the three data categories correspond to sequential evaluation for smaller configurations, later when incrementing the problem size the best strategy becomes the S3 one, and finally S4 for larger configurations. The support vectors for the first class (sequential evaluation) are 16×16, and 20×20 for the lowest configuration of the second class (S3). This second class has as support vectors: from 20×20 to 100×400 and 400×100 for the largest configurations. And finally, for the third class (S4) the support vector is 400×400.

are executed with two consecutive classes.

is S4. Therefore, LinearSVM for $f_{2-heavy}$ presents three optimal strategies depending on the configuration. For the very small problem sizes the most suitable strategy is sequential evaluation, being the support vector 16×16. When increasing the problem size, the most suitable strategy becomes S3. In this case the support vector is 20×20 for the lowest configuration. On the other hand, S3 is the best strategy for a wide range of problem sizes. This class comprises from 20×20 to 100×400 and 400×100 as support vectors. For larger problem sizes (400×400 and larger ones) the most suitable strategy is S4.

All the functions tested until this point coincide granting a relevant role to the computational charge in order to be worthwhile the GPU-based evaluation. An intensive computational charge can be reached through an increment of the problem size, or by using fitness functions with high-computational intensity. The necessary threshold to be worthwhile a GPU-based evaluation can be reached with small configurations (20×20) for very expensive functions ($f_{2-heavy}$), or with very large problem sizes (4,000×4,000) for inexpensive functions ($f_{2-light}$). In all the cases, S4 becomes the most suitable strategy when dealing with a high computational charge.

4.5.3.3 F4-Light and F4-Heavy

In order to characterize the behaviour of the evaluation time of non-separable functions, the $f_{2-light}$ and $f_{2-heavy}$ are modified by incrementing the number of necessary dimensions to calculate each partial fitness value. With this modification an increment of the computational intensity over the functions is applied.

In Eq. 4.6 and Eq. 4.7 the new functions are presented. They are very similar to the functions used in the previous section but with more terms involved to calculate any partial fitness values. This modification puts an extra pressure over the on-chip memory consumption for the GPU-strategies. More resources have to be allocated in order to hold the extra variables.

With these new functions, the general tendency of the evaluation time of the non-separable functions is fully characterized. Decision-making about the most suitable layout for functions with similar morphology is eased with this broad study.

Similarly to $f_{2-light}$, for $f_{4-light}$ (Fig. 4.8) the most suitable strategy for small and mid-size problems is a sequential evaluation. Only for very large problem sizes (larger than 1,000×1,000 or 500×8,000) GPU-based evaluation (S4) becomes the most suitable choice.

Due to the higher computational intensity of $f_{4-light}$ compared with $f_{2-light}$, the area dominated by sequential evaluation shrinks. Whereas the support vectors for sequential evaluation in $f_{2-light}$ are 1,000×4,000 and 4,000×1,000, for $f_{4-light}$ are

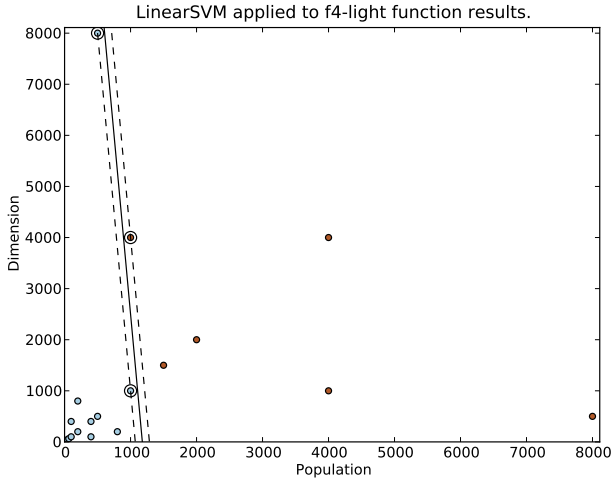


Figure 4.8: LinearSVM applied to the results of $f_{4-light}$ function: the two data categories correspond to sequential evaluation for small and mid-size configurations, and S4 for large configurations. The support vectors for the first class (sequential evaluation) are: 500×8,000 and 1,000×1,000; whereas for the second class (S4) is 1,000×4,000.

1,000×1,000 and 500×8,000. In conclusion, for lower configurations in $f_{4-light}$ than in $f_{2-light}$, the penalization of data transfer between CPU and GPU is balanced by a faster evaluation of a more expensive function.

The observed trend of reduction of dominant CPU-based evaluation between $f_{2-light}$ and $f_{4-light}$ is accentuated when comparing $f_{2-heavy}$ and $f_{4-heavy}$ (Fig. 4.9). When increasing the computational intensity of the expensive versions of the fitness function by adding more terms, the GPU-based evaluation area enlarges at the same time that the suitable problem sizes for sequential evaluation shrink.

In $f_{4-heavy}$, sequential evaluation is the fastest choice only for 10×10 configuration, whereas a slightly larger configuration, as 16×16 becomes faster when evaluating on GPU with S3 strategy. In $f_{2-heavy}$, for 16×16 the best choice is still CPU-based evaluation.

More comparisons between $f_{2-heavy}$ and $f_{4-heavy}$ show a similar pattern in both cases (Fig. 4.7 and Fig. 4.9). For low problem sizes, sequential evaluation is the fastest choice. However, due to the high-computational intensity of the function, any increment in the problem size carries out a switch from this choice to GPU-based evaluation choice.

When the problem reaches a size of 16×16, S3 becomes the most suitable strategy up to a size of 200×200. For larger problem sizes, the most suitable strategy is S4. As can be appreciated this pattern reproduces the schema of $f_{2-heavy}$ with a constriction

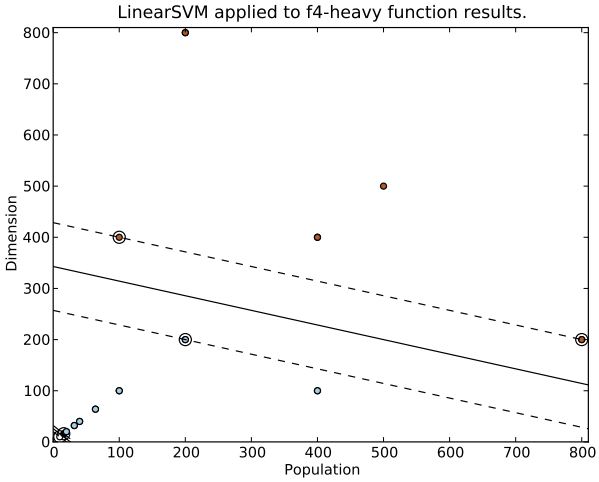


Figure 4.9: LinearSVM applied to the results of – $f_{4-heavy}$ function: the three data categories correspond to sequential evaluation for tiny configurations, later when incrementing the problem size the best strategy becomes the S3 one, and finally S4 for larger configurations. The support vector for the first class (sequential evaluation) is 10×10. For the second class (S3) are 16×16 for the lowest configuration and 200×200 for the largest one. Finally, for the third class (S4) are: 100×400 and 800×200

towards the lower problem sizes of the dominance areas of sequential and S3 strategy. S4 strategy is the fastest choice as much as the computational intensity of the function grows. For configurations such as: 100×400 or 800×200, and larger ones, S4 is the best option.

The tendency provides clues about the behaviour of the problem when progressively reducing the computational intensity of the fitness function: S3 dominance area disappears, while sequential and S4 strategies fill the previous S3 area.

Other remarkable result is the critical role of the coalesced access in the performance in all functions analysed. In the strategies where it is implemented, S3 and S4, they outperform the strategies where it is not implemented. The results suggest that coalesced access to global memory should be a major requirement in the design of evolutionary algorithms on GPU. Therefore, it can be stated that this option is mandatory to obtain an efficient implementation.

Furthermore, comparing with $f_{2-heavy}$, the pressure exerted in $f_{4-heavy}$ by the higher on-chip memory consumption produces a shared memory depletion for lower problem sizes in S2 strategy, as well as it happened in S3. Therefore, the experiments have demonstrated that implementations based on registers are more robust for very large problem sizes than others based on shared memory.

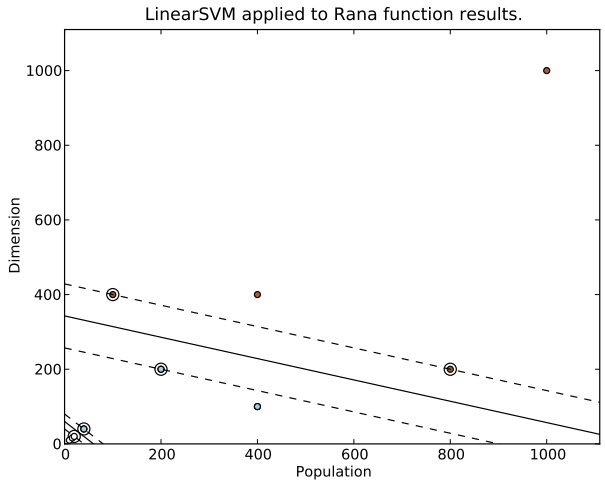


Figure 4.10: LinearSVM applied to the results of Rana function: the three data categories correspond to sequential evaluation for smaller configurations, later when incrementing the problem size the best strategy becomes the S3 one, and finally S4 for larger configurations. The support vectors for the first class (sequential evaluation) are 20×20, and 40×40 for the lowest configuration of the second class (S3). This second class has as support vectors: from 40×40 to 200×200 for the largest configurations. And finally, for the third class (S4) the support vector is 100×400 and 800×200.

4.5.3.4 Rana Function

Up to this point the proposed strategies have been tested against diverse non-separable functions. The SVM plots presented have a predictable capacity for other configurations of these particular fitness functions. However, this information is valuable not only for the functions already analysed, but also to predict the behaviour of the best strategies for other non-previously tested functions. To test the capacity of forecasting the best strategy to evaluate an arbitrary non-separable fitness function, the Rana function (Eq. 4.3) is used as benchmark.

Rana function has a computational intensity closer to the previous expensive functions than to the light ones. Therefore, depending on the problem size, three best strategies are expected: for the lowest range of problem sizes, sequential evaluation will be the most suitable strategy, for mid-range will be S3 strategy, and finally, S4 strategy will be the most suitable for the upper-range of problem sizes. In Fig. 4.10 the LinearSVM digesting the numerical results of Rana Function are shown. As can be appreciated the foreseen schema is roughly reproduced.

The results presented for f_{Rana} (Fig. 4.10) reproduce the schema of $f_{4-heavy}$ rather than $f_{2-heavy}$. Therefore, from the computational point of view f_{Rana} is close to

$f_{4-heavy}$. There are not differences for the support vectors of the largest suitable configuration for S3 (200×200), as well as for the lowest suitable configuration for S4 (100×400 and 800×200). However, there exist some differences in the largest suitable configuration for sequential evaluation: 20×20 for Rana function and 10×10 for $f_{4-heavy}$.

The fitting between the prediction and the numerical experiments is good enough to adopt it as predictive rule for other non-separable functions.

4.6 Conclusions

The works performed in this chapter have permitted an in-depth comprehension of the capacities of the GPU computing paradigm to accelerate evolutionary algorithms.

From the adaptation of the PSO algorithm to the most suitable layout to accelerate the evaluation of fitness functions, these developments have evolved in parallel to the maturity of the GPU computing paradigm and the capacities of CUDA language. By profiting of these capacities, more and more efficient implementations for evaluating fitness functions on GPU have been produced.

Chapter 5

Application of Evolutionary Algorithms to Astrophysics Problems

5.1 Introduction

The increment in the complexity of the data accessible in Astronomy, Astrophysics and Cosmology requires new procedures for their analysis. Evolutionary algorithms can provide to the practitioners the tools for analysing large and complex data sets, and by allowing the extraction of synthetic knowledge from large pieces of information. Works related with two astronomical problems have been developed in this area: the adjustment of rotational curves of spiral galaxy and the adjustment of the observational low-resolution spectral energy distribution. On the EAs side, the problems studied cover from the sensitiveness of the EAs to the choice of the RNG, to the reduction of the processing time in metaoptimization problems.

5.1.1 Rotational Curves of Spiral Galaxy

The rotation curve of a galaxy is defined as the relationship between the rotational velocity of stars as function of the radial distance to the galaxy centre. The relevance of this problem stems from the discrepancy between the observed velocity of the stars and the Newtonian-Keplerian prediction, in such way that masses derived from the rotational kinematics and gravitational laws do not match. Nowadays, this discrepancy is explained by the presence of dark matter, which is not emitting light. As a consequence, the characterization of rotation curve in spiral galaxies is a measure of the amount of dark matter in the galaxy.

Dark energy and dark matter have never directly been observed, and their nature remains unknown. Understanding the nature of the dark matter and the dark energy is one of the most important challenges of the current cosmology studies.

The velocity of the stars is characterized by an equation with physical meaning describing the four mass contributions to the rotation curve —bulge, disk, interstellar gas and halo— (Eq. 5.1).

$$v^2(r) \;=\; v_D^2(r) \;+\; v_B^2(r) \;+\; v_H^2(r) \;+\; v_G^2(r) \tag{5.1}$$

Except for the halo, the other three contributions are merged in a variable, whereas the halo contribution is modelled by Eq. 5.2. Therefore, the number of parameters to adjust is three: $v_D^2 + v_B^2 + v_G^2$, σ, and α .

$$v_H^2(r) \;=\; 2 \cdot \sigma^2 \cdot (1 - \left(\frac{r}{\alpha}\right) \;\cdot\; arctang\left(\frac{\alpha}{r}\right)) \tag{5.2}$$

5.1.2 Low-Resolution Galaxy Spectral Energy Distribution

The modelling techniques of spectra presently used in Astronomy and Cosmology are based on Single Stellar Populations[1] (SSP). The idea is to model a given galactic spectrum as a combination of spectra of SSPs defined by their different ages and metallicities[2]. Since the age and metallicity of the all components are obtained, it is also possible to determine the star formation and the enrichment histories of the galaxy or of the galaxy region. Due to the advances in the observed spectra, nowadays the researchers have a huge volume of accessible high-quality data. Theoretical stellar models have also greatly improved, mimicking to the real stellar evolution. At the same time, the fossil record methods based on spectral synthesis techniques, very demanded in the community to analyse, when possible automatically, the galaxy spectra, have emerged and matured in the last decade [21, 23, 58, 97].

5.2 Related Work

5.2.1 Related Work for the Fitting of the Rotational Curves of Spiral Galaxy

In the bibliographic search, few related studies have been found. It exists an old work which has inspired partially this survey. In this work, the author uses a genetic

[1]Simple Stellar Population consists of a set of stars born at the same time and having the same initial element composition.

[2]Metallicity (Z) is the proportion in mass of chemical elements without taking into account H and He, H+He+Z=1

algorithm to adjust the observational data of the spiral galaxy NGC 6946 [19]. Equation 5.2 is used for describing the four mass contributions to the rotation curve —bulge, disk, interstellar gas and halo— (Eq. 5.1).

5.2.2 Related Work for the Fitting of the Low-Resolution Galaxy Spectral Energy Distribution

Although it exists a wide list of works where the use of evolutionary computing is applied to problems in the area of Astronomy and Cosmology, cases where evolutionary algorithms are applied to the fitting of the low-resolution galaxy spectral energy distribution based on simple stellar populations have not been found in the literature.

By observing the techniques employed to adjust the galaxy spectra, the closest example is the use of an heuristic for the fitting process [20]. In that work, the exploration of the parameter space is made by a Metropolis algorithm. However, more differences exist between the mentioned and the present work than the simple change of an heuristic by an evolutionary algorithm. In [20] the complete emission spectrum is used in the adjustment process, whereas in the present work only 5 wavelengths —from 3650 Å to 8600 Å– are employed. This makes the fitting process essentially different.

5.2.3 Related Work for the Metaoptimization of Differential Evolution by Using Productions of Low-Number of Cycles.

Diverse works have examined aspects of parameter tuning in Evolutionary Algorithms. Early in the bibliography, the drawback associated with the large execution time in the parameter tuning is reported. In one of the pioneer studies in metaoptimization [60], it was already stated the large computational cost as limiting factor in metaoptimization processes.

A very popular strategy to overcome the large execution times of metaoptimization is *Racing* [56]. The aim of this strategy is to reduce the number of tests to estimate the utility of a behavioural parameters set. After an initial phase where all sets are equally estimated, the algorithm separates good and poor configurations, for later focussing on good ones by incrementing their number of evaluations.

The Racing strategy has suffered from modifications aimed to accelerate the discrimination of poor solutions. The variants differ on the criteria used to sift the poor configurations. For example, it can be mentioned: the use of a Gaussian distribution centred at the current best candidate to generate the next generation [102]; or *F-Race* where the Friedman test is used to promote or discard the candidates into the next

iteration. *F-Race* has been applied to Ant Colony Optimization for traveling salesman problem [11] and to *iterated local search* and *simulated annealing* for timetabling problem [10].

Other attempt of DE metaoptimization is presented at [73]. In this work, a suite of twelve fitness functions (separable and non-separable) are used as benchmark. The differences emerge in the general approach of the problem. In [73] the metaoptimization of DE is monolithic for the whole suite: the behavioural parameters are tuned for the suite; whereas in our work each case is treated independently. Finally, this work also underlines the disadvantage associated to the large execution time when evaluating the benchmark suite for the highest dimensionality (100 dimensions).

Finally, a review of the approaches for tuning the behavioural parameters of metaheuristics is presented in [10]. The review begins with the drawbacks of the *trial-and-error* approach, passing by a methodology based on *factorial design*; and finishing with *F-Race* approach. The time-consumption disadvantage when applying metaoptimization to industrial problems is also underlined. Other review of methods for parameter tuning can be found at [91]. Unfortunately, this work focusses only on one single separable function (Rastrigin function), which prevents any comparison process.

5.3 Application of Evolutionary Algorithm to Rotational Curves

5.3.1 Sensitiveness of Evolutionary Algorithms to the Choice of the Random Number Generator: Rotational Curves of Spiral Galaxies as Case Study

There are numerous scientific and technical disciplines which use random number sequences in their simulations. These disciplines are concerned about the randomness of the Random Number Generator (RNGs) employed. Fortunately, RNGs have become so close to real random number sequences that certain computational experiments are unable to distinguish between real and computational-generated random number sequences [12].

As soon as newer and longer period RNGs appear, articles studing the effect of its choice in the final performance of optimization problems are published. In spite of the continuous update of state-of-the-art, the majority of these publications use artificial problems.

These artificial problems are simplifications of more complex real problems. Therefore, the conclusions drawn in these studies should be put in quarantine and they should not be extrapolated to real problems.

In this study, a real problem —the fitting of observational data sets to a theoretical curve— is used in order to check the sensitiveness of several EAs to the choice of the RNG. The observational data sets employed are the orbital velocity of stars for four spiral galaxies: NGC2460, NGC3370, NGC4800 and NGC5394.

The data sets provide a variety of scenarios: measures with big errors and others with small ones, galaxies with a lot of points and other with very few ones, and, galaxies where the two arms fit the same curve and others where strong differences between the two arms have been observed.

For the theoretical curve, diverse series expansions are tested. Finally, a Legendre-polynomial serial expansion is selected due to its better adjustment to the experimental data sets.

Two RNGs (Mersenne Twister and GCC rand()) (section 2.2.3) have been used to test their impact over the final performance of three EAs: Particle Swarm Algorithm (PSO), Differential Evolution (DE) and Genetic Algorithm (GA). The two RNGs have been selected based on two criteria:

- The RNG has to be frequently used in research papers.
- The RNG has to be considered as high quality.

RNGs and the suites for testing their properties have hardly evolved in the last years. The most stringent suites for checking the randomness allow separating good RNGs from others. However, the validation of these tests does not suffice to deduce that all the RNGs will produce similar performances when coupling to EAs.

5.3.1.1 Production Setup

In order to check the sensitiveness of EAs to the choice of the RNG, the adjustment of observational data to a series has been used. The data correspond to the orbital velocity of stars around the centre of the galaxy.

As polynomial series, diverse series expansions are initially tested: Legendre, Bessel and polynomial series expansion. Legendre series obtains the best results reproducing better the experimental data sets. For the series expansion two degree of expansion —10 and 20— are employed. This parameter marks the dimensionality of the problem. The rest of the configuration used in this work is: 10 particles/individuals as population size and 1,000 cycles/generations.

In the PSO implementation, the c_1 and c_2 constants are established as $c_1 = c_2 = 1$ and the maximum velocity of particles $V_{max} = 2$. In the DE implementation, the mutation rate is established as $\mu = 0.5$ and the recombination rate as $C_r = 0.5$. In

the GA implementation, the mutation rate is established as $p_m = 0.01$. An one-point crossover operator is also implemented. From a randomly selected point in the two parents, two descendants are created by mixing the parents genetic information. The two best ones from the two parents and the two descendants are retained. Elitism is not implemented in the GA.

According to the usual practice in adjustment of experimental data to theoretical curve, the chi-square test —χ^2—- has been chosen as fitness function within this work [67]. The lower the χ^2 value is, the closer the solution is to the objective —the fitter the experimental data is to the theoretical curve. Therefore, the task becomes minimizing χ^2.

Considering the standard fitting problem, where one is given a discrete set of N data points with associated measured errors σ, and is asked to construct the best possible fit to these data using a specific functional form, the most appropriated fitness function is the merit function χ^2, Eq. 5.3 [80]. Therefore, independently of the specific functional form chosen, the fitness function used in this work is χ^2, Eq. 5.3.

$$\chi^2 = \sum_{all\ points} \left(\frac{y_{simulated} - y_{observed}}{\sigma}\right)^2 \tag{5.3}$$

For each case —each EA, galaxy rotation curve and polynomial degree— a total of 25 tries are executed in order to reach the statistical relevance desired.

In Fig. 5.1, the four galaxy rotation curves used in this work are shown. Particularly, these galaxy rotation curves are extracted from a larger astronomical data set, covering 56 galaxies. The criteria to select these curves were: the largest and the smallest data set, and two more randomly selected.

As can be appreciated the data are very diverse, providing different scenarios: curve with many points —more than 200— and with few points —less than 20—, big and small errors, duplicated values for the same x-axis coordinate —corresponding to the two arms of the spiral galaxy—. This diversity makes the task of adjustment very challenging and stressing; and providing different scenarios to test the EAs coupled to the RNGs.

In order to fit the points (Fig. 5.1) to a theoretical curve, a series expansion of Legendre Polynomial is used (Eq. 5.4). In this series expansion the unknown terms a_i have to be calculated to obtain the better adjust of function $F(x)$ to the points.

$$F(x) = \sum_{i=0}^{N} a_i \cdot LP_i(x) \tag{5.4}$$

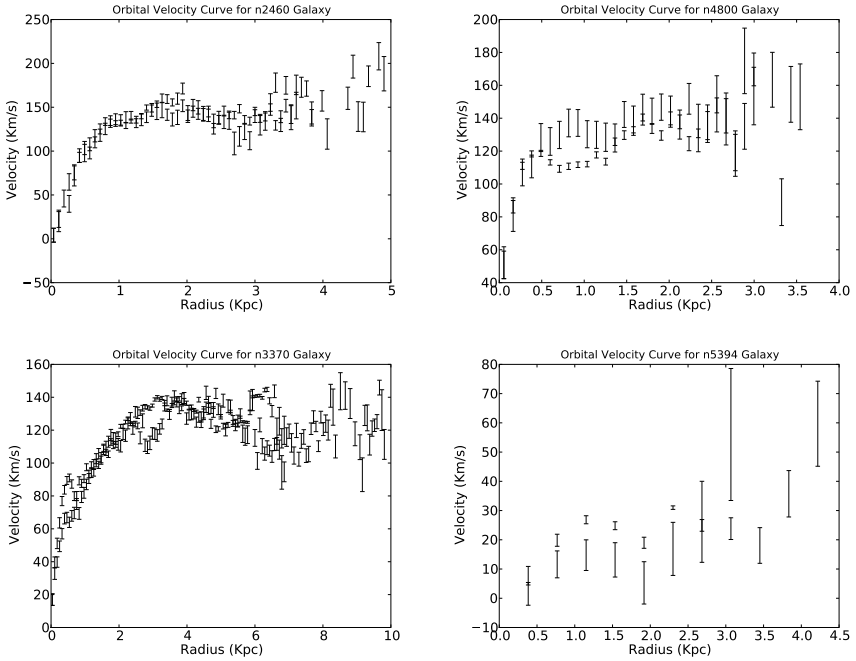


Figure 5.1: Galaxy rotation curves —experimental data sets— used in this survey: NGC2460, NGC4800, NGC3370 and NGC5394.

5.3.1.2 Results and Analysis

The most appropriated statistical test to assess the impact of the choice of the RNG over final performance of the EAs is the Wilcoxon signed-rank test. This is a non-parametric test used for statistical inference. In Table 5.1, the p-value of Wilcoxon signed-rank test for each EA, galaxy and polynomial degree is presented. In our study, the analysis of the sensitiveness of the EAs is based on these values. The significance level used has been $\alpha = 0.05$, which is the most usual in this kind of analysis.

When changing the RNG in the PSO algorithm, *null hypothesis* —$H_0 : \mu_1 = \mu_2$— can be rejected only in one case —NGC2460 and degree 10—. This is the single case where the PSO algorithm shows sensitiveness to the change of the RNG. By contrast, for the rest of the cases —a total of 7— the null hypothesis can not be rejected. For these cases the change of RNG has not any impact over the final performance of PSO.

Regarding the results of Wilcoxon signed-rank test for DE, the *null hypothesis* can be also rejected in a single case —NGC2460 and degree 20—. For the rest of the cases —a total of 7— the *null hypothesis* can not be rejected.

Finally for GA, the *null hypothesis* can not be rejected in any case. Therefore, the statement $H_0 : \mu_1 = \mu_2$ has to be accepted as true; leading to the conclusion that the

Table 5.1: p-value from Wilcoxon signed-rank test for non-parametric hypothesis testing for each evolutionary algorithm, galaxy and expansion degree.

Evolutionary Algorithm	Galaxy and Polynomial Degree							
	NGC2460		NGC3370		NGC4800		NGC5394	
	10	20	10	20	10	20	10	20
PSO	**0.026**	0.367	0.288	0.925	0.757	0.657	0.158	0.276
DE	0.443	**0.040**	0.098	0.638	0.619	0.135	0.058	0.946
GA	0.065	0.619	0.638	0.861	0.545	0.638	0.946	0.109

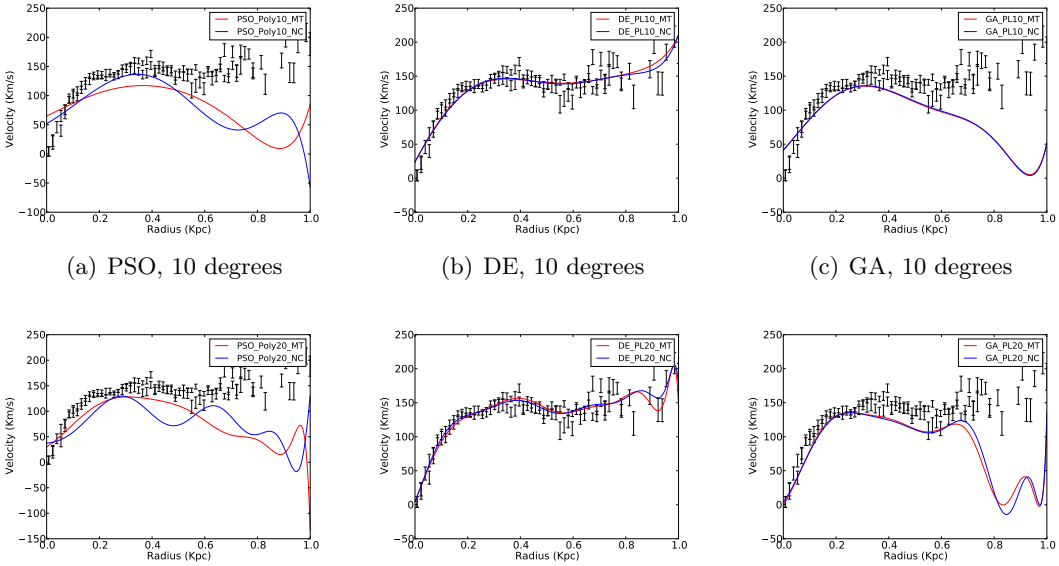
change of the RNG does not have any impact over the final performance of the GA.

Based on the non-parametric analysis performed, it can be concluded that the choice of the RNG has not any impact —for GA— or a very small impact —for PSO and DE— over the final performance of the problems treated in this section. Unfortunately, the Wilcoxon signed-rank test does not allow establishing conclusions about which particular RNG produces best performance when it is coupled to a EA, only if the results differ or not.

The analysis of the results allows building a scale of sensitiveness for the EAs tested: $DE = PSO > GA$. This scale coincides partially with the scale created with the work [15], in which only artificial functions were employed. In this work the scale of sensitiveness obtained was: $DE > GA > PSO > FA$. As can be appreciated, in both studies DE is the most sensitive algorithm to the choice of the RNG. Oppositely, both surveys alternate the positions in the scale of GA and PSO.

In order to evaluate if the EAs perform better when using a particular RNG, the sign test has been employed. For testing $H_0 : M_{NC} \geq M_{MT}$ against $H_1 : M_{NC} < M_{MT}$, H_0 is rejected if the number of plus signs is less than or equal to the critical value for this test. Taking into account that the number of tests is 25, the critical value is 7. When applying this, only one rejection is produced —PSO, NGC2460 and degree 10—. Therefore, only for this case, the algorithm performs better when using MT rather than NC can be stated. For the other case —DE, NGC2460 and degree 20—, the null hypothesis can not be rejected.

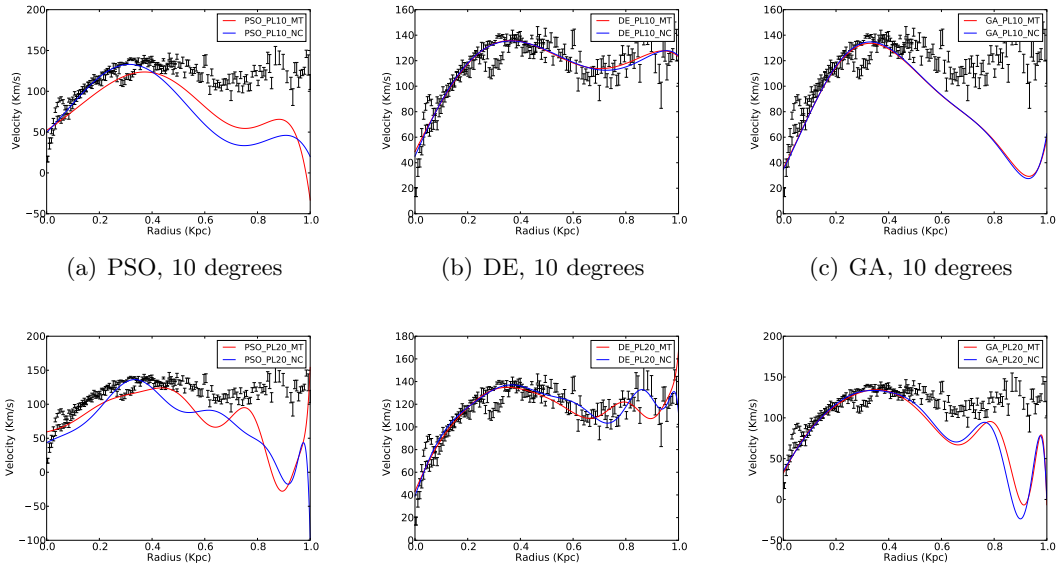
In Figs. 5.2, 5.3, 5.4 and 5.5 the best adjustments obtained for each case are presented. As observed in these figures, in most of the cases both curves well-conform the observational data, so the simple observation of them does not allow discerning if the RNGs differ in performance when coupling to different EAs. This endorses the results obtained by the Wilcoxon signed-rank test disabling in most of the cases to discern between both RNGs.



(a) PSO, 10 degrees (b) DE, 10 degrees (c) GA, 10 degrees

(d) PSO, 20 degrees (e) DE, 20 degrees (f) GA, 20 degrees

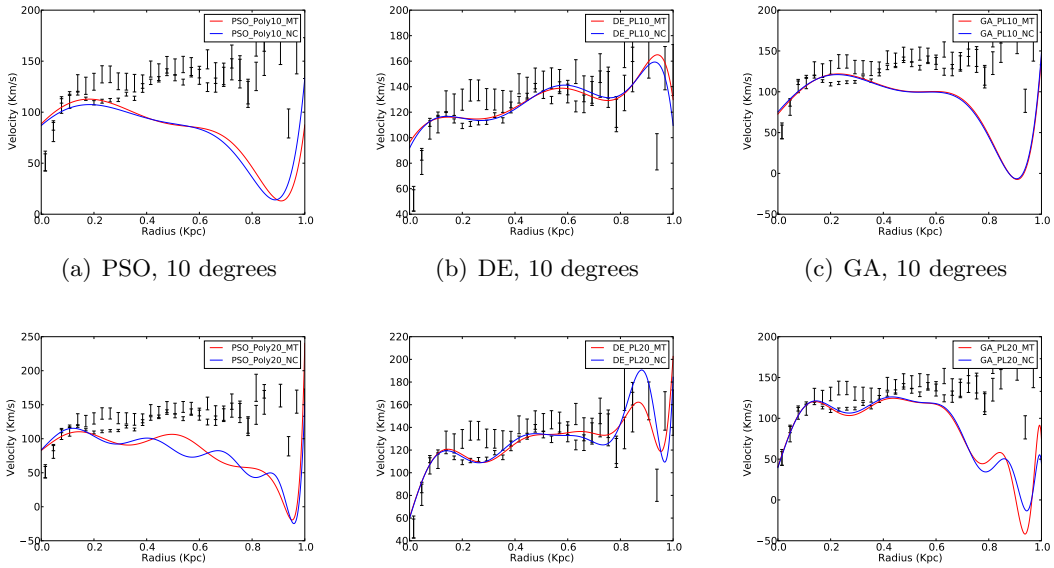
Figure 5.2: Comparison of the best adjustment obtained with the RNG tested for galaxy NGC2460.



(a) PSO, 10 degrees (b) DE, 10 degrees (c) GA, 10 degrees

(d) PSO, 20 degrees (e) DE, 20 degrees (f) GA, 20 degrees

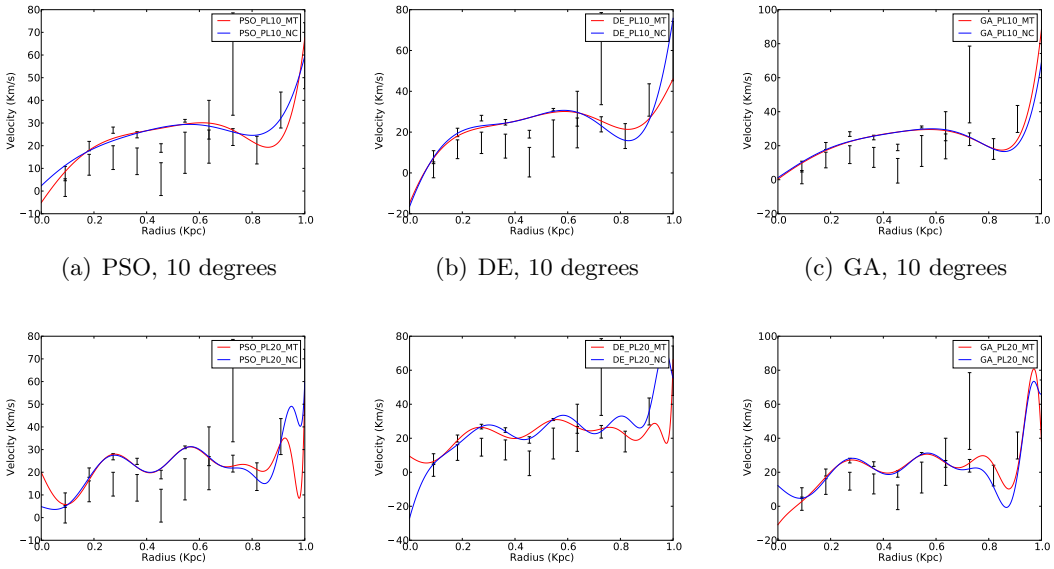
Figure 5.3: Comparison of the best adjustment obtained with the RNG tested for galaxy NGC3370.



(a) PSO, 10 degrees (b) DE, 10 degrees (c) GA, 10 degrees

(d) PSO, 20 degrees (e) DE, 20 degrees (f) GA, 20 degrees

Figure 5.4: Comparison of the best adjustment obtained with the RNG tested for galaxy NGC4800.



(a) PSO, 10 degrees (b) DE, 10 degrees (c) GA, 10 degrees

(d) PSO, 20 degrees (e) DE, 20 degrees (f) GA, 20 degrees

Figure 5.5: Comparison of the best adjustment obtained with the RNG tested for galaxy NGC5394.

5.3.2 Adjustment of Rotational Curves of Spiral Galaxy to Specific Functional Forms Using Particle Swarm Algorithm and Differential Evolution

This section focusses on the construction of a model for the rotational curves of spiral galaxies. For this, the observational data are normalized and merged, and next, fitted to physical meaningless functional forms. Due to the large search space, EAs are used to find sub-optimal high-quality solutions. PSO and DE are implemented to adjust a large observational data set —56 rotational curves of spiral galaxies— to functional forms.

PSO and DE are well-known EAs, widely adopted and suitable for the first approximation to any optimization problem. Regarding the functional form, Legendre polynomial and normal polynomial are considered to reproduce the essential information of the rotational curves.

5.3.2.1 Production Setup

Diverse serial expansions (Legendre and polynomial) are tested to fit the observational data to the theoretical physical-meaningless curve. In spite of the equal a priori capacity, the Legendre polynomial —50 degrees in all serial expansions— serial expansion showed a major sensitiveness to reproduce the data behaviour and produced the lowest values of the fitness function.

Similarly to other studies in this chapter, the chi-square test (Eq. 5.3) has been selected as fitness function.

For each case —each EA and type of polynomial— a total of 25 tests are executed in order to reach the desired statistical relevance. As pseudorandom number generator, a subroutine based on Mersenne Twister has been used [59].

In order to fairly compare the curves of the galaxies, a double normalization has been applied. First of all, the size of the galaxies has to be homogenized. For this normalization, the radius where the maximum velocity is reached, is settled —in arbitrary units— at 0.1 units. Consequently, all the radii measured for the galaxy under modification are conveniently scaled.

Second of all, the maximum velocity of each galaxy is settled at 1 —in arbitrary units—-. As a consequence, the rest of measured velocities are also appropriately scaled. Finally, resulting of the scaling in velocities, the velocity error must be rescaled proportionately to the velocity associated.

As result of this double normalization, all the curves have a common coordinate at (0.1, 1). Once the normalization process has proceeded, the extraction of a pattern

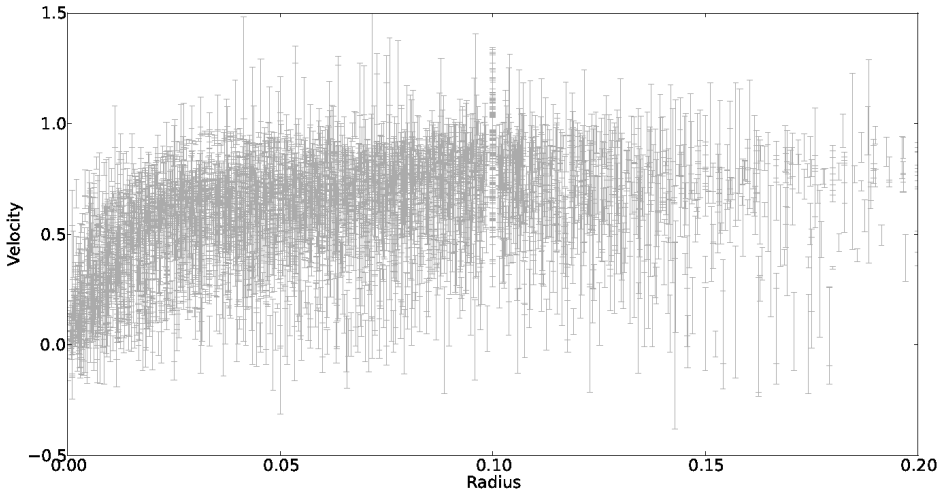


Figure 5.6: All rotation curves doubly normalized.

representing all the curves can be executed. In Fig. 5.6 the complete observational data set is presented. Particularly, the galaxy rotation curves used in this work cover 56 galaxies, being involved a total of 5051 points [57].

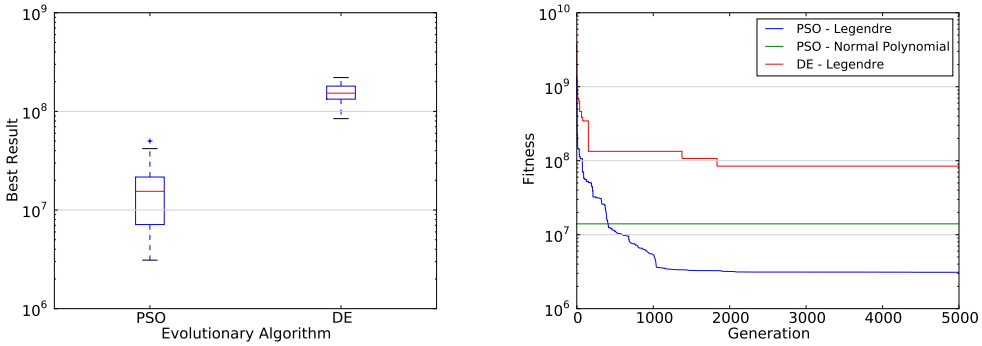
5.3.2.2 Results and Analysis

It is well known in evolutionary computing that it is not possible to know a priori which EA will perform the best for a particular problem. For this reason, the optimization problems are treated with a variety of techniques, retaining the best ones for further improvements.

In Fig. 5.7(a), the comparative box plots of the best results for the algorithms PSO and DE when using Legendre series are presented. As can be appreciated the PSO algorithm outperforms DE, in both: the absolute best result obtained after the 25 test, as well as the median of the samples. Therefore, the use of DE will be rejected for this problem.

The application of the Wilcoxon signed-rank test to the data shown in Fig. 5.7(a) indicates that the differences are significant from the statistical point of view for $\alpha = 0.05$.

In Fig. 5.7(b), the evolution of the best result for each case studied is presented. In this figure, the evolution of PSO with Legendre polynomial can distinguish from the other cases by the rapid evolution during the first thousand generations. However, for



(a) Comparative box plots

(b) Fitness evolution

Figure 5.7: Panel (a) shows the comparative box plots for the best results obtained for PSO and DE algorithms when using Legendre series, while panel (b) shows the fitness evolution for the best result of each case studied

the second half the fitness evolution stagnates. The other two algorithms show a lower ability to evolve along the generations.

Finally, in Fig. 5.8, the fittest solution to the observational data is presented. In the range [0, 0.2], where the bulk of data is concentrated, the adjustment is acceptable. However, far of this segment and due to the lack of data, the adjustment produced is far from the optimum.

5.3.3 Metaoptimization of Differential Evolution by Using Productions of Low-Number of Cycles: the Fitting of Rotation Curves of Spiral Galaxies as Case Study

In order to increase the efficiency of EAs, practitioners include improvements such as new operators, modifications of the canonical operators, or the hybridization with other EAs. However, an alternative to obtain high-quality solutions is: to tune the parameters which govern the behaviour of the algorithm to the specific problem to optimize. This parameters adjustment can be performed by using other EAs (metaoptimization). Unfortunately, metaoptimization leads to a critical increment in the execution time.

During the development of metaheuristics techniques, the optimizers require to fix the values of diverse behavioural parameters. In general, these parameters govern the behaviour of the algorithms, and therefore, they are key elements in its final efficiency.

In the past, approaches based on the *factorial design* have been followed to optimize the behavioural parameters. However, this procedure oversimplifies the problem, neglecting the potential relationships between the behavioural parameters. Neither, by-hand selection of the most suitable set of parameters is an affordable task.

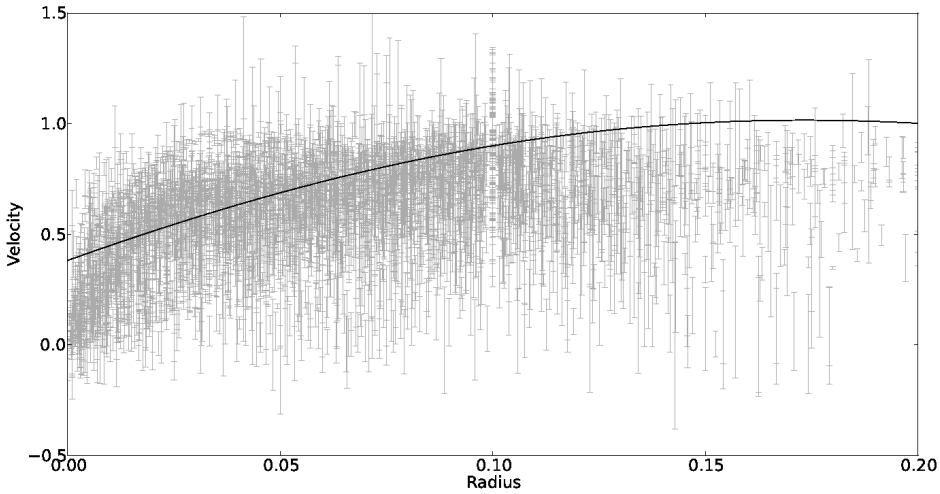


Figure 5.8: Absolute best result —the fittest adjustment to observational data— obtained. Configuration used PSO with configuration of 100 particles and 5,000 cycles, and a series of Legendre Polynomials of 50 degrees.

As any other complex problem, the adjustment of the behavioural parameters of an EA can be treated by other EA, termed *metaoptimizer* or *tuner*. This kind of optimization is termed *metaoptimization*.

Unfortunately, the metaoptimization carries out a relevant increment of the execution time. If the problem to optimize takes long, a high-number of cycles or a large population are required to obtain high-quality solutions, then the scenario aggravates. Therefore, it is necessary to evaluate if a lower number of cycles in the optimizer produces behavioural parameters of enough quality for the problem, and consequently, processing time can be saved; although this low-number of cycles of the optimizer is not producing so-high-quality solutions.

If the behavioural parameters used in the optimizer (algorithm which is optimized) exhibit its quality from the very initial cycles, then large executions can be avoided. Moreover, the number of cycles of the optimizer is a control mechanism over the execution time budget and, indirectly over the quality of the solutions of the tuner.

Particularly, this section focuses on tuning the behavioural parameters of DE [81, 92]. This election is based on the popularity of the algorithm, frequently used in optimization in artificial functions and real-world problems.

5.3.3.1 Implementation

In order to deal with a whole evolutionary algorithm, a python implementation is proposed for the tuner. Python election is based on its capacity to handle pieces of text, to compose files with these pieces, then to compile the source code, to execute it and to capture output information from the execution. By repeating this process, the behavioural parameters of the optimizer can evolve. In our work, both tuner and optimizer implement DE algorithm.

On the other hand, the EA which parameters are being optimized is codified in C language. C language election is based on the need of a fast execution for the problem under optimization. Additionally to the cited benefit, this different codification eases the identification of each part of the code while codifying.

One of the critical points of the metaoptimizer part is to capture the final fitness of the EA to be recorded as the fitness of the metaoptimizer individual. For this, the best fitness is recorded in a text file after executing the problem and captured by python from this file. So, synchronization operations during the writing and reading are required.

Both DE algorithms (tuner and optimizer) [81, 92] have been implemented with the schema DE/rand/1/bin [64]. Furthermore, in all numerical experiments, the configuration in the tuner is a population of 10 vectors and 10 cycles. Otherwise, in the optimizer, the population is composed by 10 vectors; and two configurations for the number of cycles: 10 and 1,000. In all cases, real-valued representation is used. The behavioural parameters of the tuner are fixed with values $\mu = CR = 0.5$.

As pseudorandom number generator, a subroutine based on Mersenne Twister [59] has been used in both implementations: python and C. The numerical experiments are executed in a single core of a computer with two Intel Xeon X5570 processors at 2.93 GHz and 8 GB of RAM. The C code has been compiled by using gcc version 4.4.5 with optimization level -O3.

5.3.3.2 Metaoptimization Production

In order to check the hypothesis of the capacity of the tuner to produce competitive behavioural parameters by using a reduced number of cycles (10) in the optimizer, a production composed of 25 executions is performed per case. Later, these behavioural parameters are compared with the behavioural parameters emerged from a production with high-number of cycles (1,000).

After each execution of DE tuner, a couple of values (μ, CR) are obtained as tuned behavioural parameters for the problem under optimization (Fig. 5.9). Additionally

to the scatter plot $\mu - CR$, at top and at right of each figure, the histograms with the frequency of each value are also plotted.

As can be appreciated, most of the tuned parameters are located in the upper-right quadrant. These results are slightly different, but congruent with the recommendation ($\mu = 0.8$ and $CR = 0.9$) of Prof. Storn[1] for the schema DE/rand/1/bin; although the original recommendation is stated for artificial separable functions. Next, the mid-point of the most populated division is employed to decide about the most suitable behavioural parameters (μ, CR) for the problem to optimize (Table 5.2).

By observing the values obtained for μ and CR (Fig. 5.9), it is appreciated the similarities in the values, independently of the number of cycles of the optimizer. This reinforces the hypothesis that the quality of the behavioural parameters can be extracted from the few initial cycles. The most suitable values for NGC2460 and 10 cycles are $\mu = CR = 0.95$ (Fig. 5.9(a)), whereas for 1,000 cycles are $\mu = 0.75$ and $CR = 0.65$ (Fig. 5.9(b)). For the galaxy NGC3370, the most suitable values for low-number of cycles are $\mu = 0.95$ and $CR = 0.85$ (Fig. 5.9(c)), whereas for high-number of cycles[2] are $\mu = 0.95$ and $CR = 0.75$ (Fig. 5.9(d)).

The next step is to verify if the efficiency of each set of behavioural parameters is significantly different.

5.3.3.3 Fitness Analysis

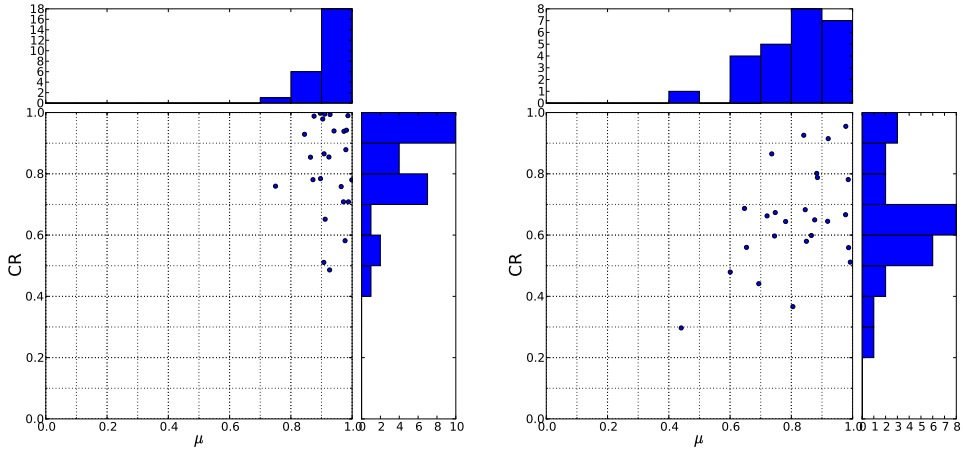
In order to discriminate if the tuned behavioural parameters of DE are more efficient when the tuning process has been performed with 10 or with 1,000 cycles; 25 runs of the optimizer are executed per case (Table 5.2).

Concerning the numerical results for the galaxy NGC2460, it can be observed that the tuned parameters with low-number of cycles ($\mu = CR = 0.95$) outperform the tuned parameters with high-number of cycles ($\mu = 0.75$, $CR = 0.65$) when both executing 1,000 cycles. For the galaxy NGC3370, the same comparison leads to both cases: low-number ($\mu = 0.95$, $CR = 0.85$) and high-number ($\mu = 0.95$, $CR = 0.75$) of cycles produce the identical mean fitness. As expected, whatever tuned behavioural parameters, independently of the number of cycles, outperform randomly selected behavioural parameters ($\mu = CR = 0.5$).

From the proposed experimental setup and the numerical results, it can be concluded that a reduction in the number of cycles of optimizer, at least, does not degrade

[1] http://www1.icsi.berkeley.edu/~storn/code.html

[2] In the previous cases —galaxy and number of cycles— the mid-point of the most populated division is selected to establish the most suitable values of μ and CR. However for the galaxy NGC3370 and 1,000 cycles configuration, neither division is populated with more than 2 points. Therefore, the most populated bin in the histogram is used as criterion to select the suitable behavioural parameters.



(a) NGC2460, 10 cycles

(b) NGC2460, 1,000 cycles

(c) NGC3370, 10 cycles

(d) NGC3370, 1,000 cycles

Figure 5.9: Results (μ and CR) of metaoptimizer after 25 executions for galaxies: NGC2460 and NGC3370, and for 10 and 1,000 cycles. Top and right: the histogram of the frequency of the values of the behavioural parameters.

Table 5.2: Best fitness (25 executions) for each galaxy and case. The numerical results labelled with: *random* have been obtained with $\mu = CR = 0.5$, those labelled with *optimized* by using μ and CR optimized with 10 or with 1,000 cycles. The numerical results without label correspond to the cases where μ and CR have been optimized with 10 cycles and the runs executed with 1,000 cycles.

Galaxy	μ	CR	Cycles	Mean fitness	Comment	Statistical Test (p-value)
NGC2460	0.50	0.50	10	57,518.6±16,848.4	Random	Wilcoxon signed-rank $1.2 \cdot 10^{-5}$
	0.95	0.95	10	2,938.3±1,966.7	Optimized	
	0.50	0.50	1,000	1,247.4±630.3	Random	Kruskal-Wallis $2.8 \cdot 10^{-12}$
	0.95	0.95	1,000	314.5±1.14e-13		
	0.75	0.65	1,000	375.2±222.1	Optimized	
NGC3370	0.50	0.50	10	353,444.0±61,195.0	Random	Wilcoxon signed-rank $1.1 \cdot 10^{-5}$
	0.95	0.85	10	28,741.8±16,472.1	Optimized	
	0.50	0.50	1,000	11,613.7±6,472.7	Random	Kruskal-Wallis $9.7 \cdot 10^{-14}$
	0.95	0.85	1,000	2,873.9±4e-13		
	0.95	0.75	1,000	2,873.9±4e-13	Optimized	

Table 5.3: Mean execution time of both tuner and optimizer for 10 and 1,000 cycles in the optimizer.

Galaxy	Cycles	Execution Time	Cycles	Execution Time	Reduction
Optimizer					
NGC2460	10	6.56 ms	1,000	412.24 ms	98.4%
NGC3370	10	12.24 ms	1,000	654.12 ms	98.1%
Tuner					
NGC2460	10	17.41 s	1,000	86.58 s	79.9%
NGC3370	10	18.23 s	1,000	146.10 s	87.5%

the quality of the behavioural parameters obtained in the metaoptimization process. Based only on the initial cycles of the optimizer, the tuner is able to capture enough information about the quality of the behavioural parameters to evaluate them.

5.3.3.4 Statistical Analysis

In order to check if the differences in the fitness (Table 5.2), when using behavioural parameters tuned with low-number and high-number of cycles in the optimizer, are significant, this production is statistically analysed.

The statistical analysis of data is performed by using Kruskal-Wallis test for multiple comparisons, and Wilcoxon signed-rank test for pair comparison. In all cases, non-parametric tests have been chosen because they do not require explicit conditions for data distribution.

Except for the case of NGC3370, 1,000 cycles and the two sets of tuned behavioural parameters —where identical numerical results are obtained—, the Kruskal-Wallis and Wilcoxon signed-rank tests indicate that the differences for the numerical results are significant for a confidence level of 95% (p-value under 0.05). This means that the differences are unlikely to have occurred by chance with a probability of 95%.

5.3.3.5 Execution Time

In the previous points, the analysis focussed on the values achieved by the tuned behavioural parameters of DE and on the numerical results obtained with these values. It has been proved that metaoptimization based on optimization process with low-number of cycles can produce high-quality behavioural parameters for DE algorithm. Once the numerical efficiency of the approach has been checked, the corresponding processing times are presented (Table 5.3).

As can be appreciated, Table 5.3 shows a significant reduction of the execution times for both: tuner and optimizer when a low-number of cycles are employed in the optimizer. The execution time reduction is higher than 98% for the optimizer, while ranging from 79.9% to 87.5% for the tuner.

Through optimizing behavioural parameters in this scenario, a saving of processing time is achieved, at the same time that high-quality solutions are produced. This is specially relevant for industrial applications where execution the time is as relevant as the fitness; and for the cases where an optimization process is applied successively to different data sets. By varying the number of cycles in the optimizer, the metaoptimization process is endowed of a control over the quality of the achieved solutions and over the processing time budget.

5.4 Metaheuristics for Modelling Low-Resolution Galaxy Spectral Energy Distribution

In the present section, the adjustment of the observational low-resolution Spectral Energy Distribution (SED, O_λ) of M110 galaxy to a set of SSPs is implemented and tested. Beyond of this immediate goal, the final purpose is to create and deliver to the community a code able to fit the SED of a galaxy using predefined SSPs.

Integral Field Units (IFU) facilities will enlarge the complexity of the problem since a complete spectrum will be produced per pixel of the image of the galaxy, instead of a single spectrum for the whole image of the galaxy. For this purpose, diverse EAs are tested against the problem in order to find the most suitable one. Therefore, the scope of this work is far from this particular galaxy; being this one simply used as benchmark to test the approach. The evolutionary algorithms tested in this work are: Genetic Algorithm (GA), Particle Swarm Optimizer (PSO) and some variants: Inertial Weight PSO (IWPSO), and MeanPSO; and Differential Evolution (DE).

The low resolution spectra for the galaxy M110 given by the magnitudes in several broad band filters[1] are used as input to the code. The fit will try to obtain the best combination of two stellar populations given by SSPs able to reproduce these data. The SSPs broad-band filter magnitudes are taken from the results of the evolutionary synthesis code PopStar [66], with which a set of models have been calculated for 6 different metallicities, Z, in the range [0.0001, 0.05], and 106 different stellar logarithmic

[1]Data have been extracted from NASA/IPAC Extragalactic Database; http://ned.ipac.caltech.edu/. These quantities have not necessarily been corrected for background extinction. This might introduce uncertainties which make difficult the fitting process.

ages ranging from 5.00 to 10.18. The final sample of SSPs employed consists in a set of 636 theoretical models.

By using SSP mechanism, some implicit considerations are usually assumed about the creation and evolution of the stars in the galaxy. In elliptical and spheroidal galaxies the conversion of an amount of gas in a set of stars is considered a very rapid process, and this way only two (or even one) SSPs may be valid to reproduce their spectra. Although this model can seem simple to describe correctly the spectrum of a galaxy —O_λ—, it is useful enough to show the general tendency of the galaxy, and therefore, to understand its evolutionary history.

5.4.1 Structure of the Candidate Solutions

By considering that the objective of this work involves the fitting of observational galaxy spectrum and, that this objective requires to weight the SSPs through coefficients representing the amount of stellar masses of the selected SSPs, the first step is to propose an adequate structure for the candidate solutions.

The structure of the proposed candidate solutions is composed by a sum of terms (Eq. 5.5), where each term has two factors:

$$O_\lambda = \sum_{types\ of\ SSPs} C_i \cdot SSP_{[Z,logage]} \tag{5.5}$$

- The first one is the amount of stars —measured in solar masses— of a particular SSP, C_i.
- Whereas, the second factor is the SSP itself. The spectrum of the SSP depends on two features of the stars: the age and the metallicity.

In adopting this structure for the solutions, two different types of optimizations have to be performed. On the one hand, the selection of the most suitable SSPs from the set of the SSPs constitutes a combinatorial optimization problem. On the other hand, the optimization of the weight previously mentioned constitutes a continuous optimization problem. Due to the nature of the problem and the structure of the candidate solutions, both optimizations should not be untied. This double faced problem makes the fitting process challenging and appropriate to hybridize different evolutionary techniques.

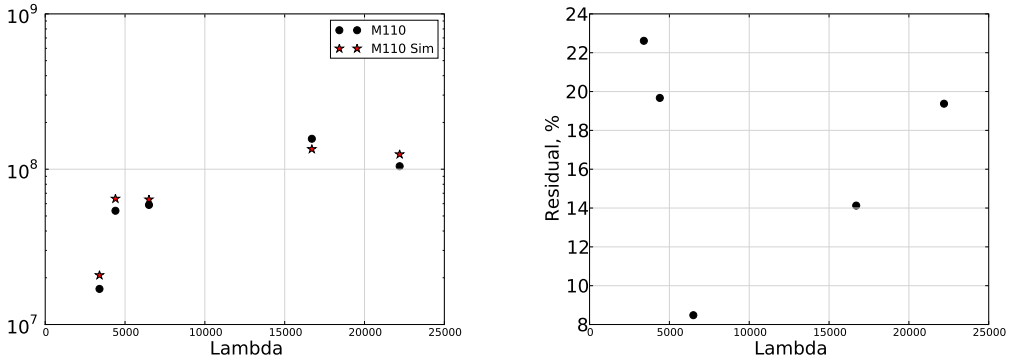
5.4.2 Results and Analysis

5.4.2.1 Mutation Operator for SSP

As first attempt, a GA —only with a mutation operator active— is implemented. GA holds numerous advantages, such as: flexibility and adaptability to many different types

Table 5.4: Mean fitness and deviation standard for 1,000 cycles 10 individuals and diverse mutation ratios when using only mutation operator over SSPs.

Mutation Ratio	0.01	0.05	0.10
Mean Fitness	14.3±39.4	0.8±0.7	7.8±14.2



(a) Observed and simulated SED

(b) Residual (relative error) between the observed SED and the simulated one.

Figure 5.10: Results when applying the mutation operator only to the SSPs.

of problem. Moreover, the reduced number of parameters and the low complexity of the GA allow a quick implementation in order to obtain in a short period of time the first tentative solutions.

Initially, the GA is applied to the combinatorial part of the problem —the selection of the SSPs—. In this part of the problem, the SSPs of the candidate solutions change, being governed this modification by a mutation ratio. The mutation operator randomly replaces one of the two SSPs which compose the candidate solution, by other randomly selected SSP. If the muted individual is better than the ascendant, then the ascendant is replaced by the muted individual; otherwise, the ascendant is kept and the descendant rejected. This simple mechanism allows a selection of the most suitable SSPs for a particular galaxy spectrum.

The results of this first strategy —when implementing a mutation ratios: 0.01, 0.05, and 0.10— are presented at Table 5.4. Although these results are promising for an initial attempt, the dispersion (deviation standard) is too much high. To have a low dispersion when repeating execution is considered as a valuable feature. Beyond the numerical results, the objective is to reproduce the SED of M110. For this reason, the observed and simulated SEDs for M110 galaxy are presented at Fig. 5.10(a).

In order to understand the goodness of the final results produced by the algorithm, a comparison between the observed SED of the galaxy M110, and the modelled SED by the final best individual of each execution is performed (Fig. 5.10). As can be observed the fitting of the SED for M110 is not optimum (Fig. 5.10(a)). This initial strategy is able to reproduce only the general tendency of the observed SED, however it still overestimates or underestimates some values. The relative differences, $(100 \times \frac{SED^{M100} - SED^{sim}}{SED^{M100}})$, reach up to 25% (Fig. 5.10(b)). Besides it is perceived the difficulty to fit values ranging up to 2 orders of magnitude. It is expected that more elaborated strategies will produce better adjustments between the SED observed and modelled. Better values of the fitness: lower mean and standard deviation, are expected when finding more suitable metaheuristics.

5.4.2.2 Mutation Operator for Coefficients

Beyond the initial approach focussed on the combinatorial part of the problem, an additional mutation operator is implemented for the coefficients, C_i. Until now the algorithm keeps frozen the coefficients[1] from the initial generation and along the whole execution. This impedes the evolution of the stellar mass of a selected SSP in the galaxy. By implementing this new operator, the algorithm will be able to make them evolve. It is expected that this new mechanism improves the overall performance of the algorithm; overcoming the inherent flaws of the first approach.

In order to keep the same number of evaluations in the algorithm and fairly compare with the previous implementation, the number of cycles is reduced to the half[2]. This is due that in each cycle the mutation operator over the SSP selection is applied and evaluated, and next, the mutation operator over the coefficients is also applied and evaluated. Consequently, the number of evaluations per cycle is the double that in the previous implementation.

Two variants of the mutation operator for the coefficients have been tested. Firstly, a flat mutation probability distribution in which the mutation ratio is fixed as identical to the mutation ratio for SSPs; and secondly, a Gaussian probability distribution in which the mean and the standard deviation of the probability distribution is calculated over the coefficient values in each generation (Table 5.5).

The application of mutation operators to make evolve the coefficients of the candidate solutions improves significantly the best results obtained until now (Table 5.5). However, better solutions (lower standard deviation and mean fitness) are expected if

[1] These coefficients are randomly created at the beginning of the algorithm.

[2] Hereafter, when two EA are hybridized, a similar reduction in the number of cycles is applied in order to keep constant the number of evaluations.

Table 5.5: Mean fitness and standard deviation for 1,000 cycles, 10 individuals and diverse mutation ratios when using mutation operator over SSPs selection and over the coefficients.

Mutation Ratio	0.01	0.05	0.10
Flat Mutation Operator			
Mean Fitness	110.2±334.1	0.27±0.06	0.25±0.03
Gaussian Mutation Operator			
Mean Fitness	16.1±33.9	0.33±0.11	0.25±0.05

more refinements are applied. Therefore, other evolutionary algorithms are tested in the followings.

5.4.2.3 PSO for Coefficients

In order to check efficiency of other evolutionary algorithms, PSO is tested for optimizing the coefficients. Besides the standard PSO implementation, two variants are also checked: MeanPSO and IWPSO. The production includes the three algorithms: PSO, MeanPSO and IWPSO; three maximum velocities: 10^7, 10^8, and 10^9, the parameters $c_1 = c_2 = 1$; and three mutation ratios for the SSPs: 0.01, 0.05 and 0.10. Unfortunately, the numerical results do not improve the previous results (mutation operator applied to the SSPs selection and to the coefficients), and for this reason, they have been omitted.

5.4.2.4 Differential Evolution for Coefficients

DE is other EA specially suitable for optimization of continuous problems, and it has a large portfolio of cases where it outperforms both GA and PSO. For this reason, it is considered to optimize the continuous part of the candidate solutions, C_i. In this case, DE follows the schema DE/rand/1/bin with $\mu = CR = 0.5$. The main features of this algorithm are its flexibility to deal with many different types of problems, and the speed in the implementation and in the execution, which allows quickly obtaining high-quality suboptimal solutions.

First of all, the numerical results of the DE algorithm (Table 5.6) are compared with the results obtained in the previous best implementations (Table 5.5). As can be observed, the DE algorithm outperforms the previous best implementation, i.e. mutation operator acting over the SSPs selection and Gaussian mutation operator over the coefficients. By comparing, it can be concluded that the hybridization of the mutation operator acting over the SSPs with the DE algorithm acting over the

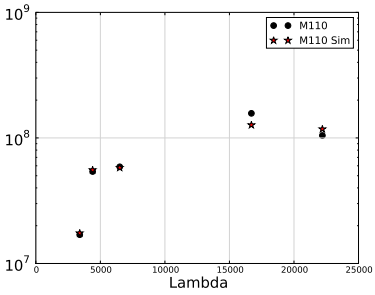
coefficients (stellar masses) produces better results than any of the previous strategies tested. As an extra value, a low standard deviation is also achieved.

Table 5.6: Mean fitness and standard deviation for 1,000 cycles, 10 individuals and diverse mutation ratios when using mutation operator over SSPs selection and DE over the coefficients.

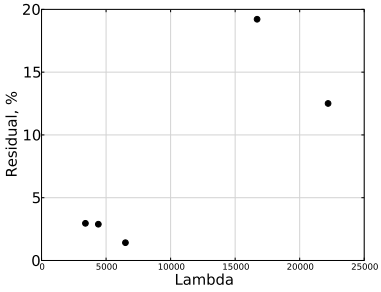
Mutation Ratio	0.01	0.05	0.10
Mean Fitness	0.32±0.19	0.24±0.03	0.226±0.009

The application of the Kruskal-Wallis test, $p-value = 8.6 \cdot 10^{-6}$ to these results indicates that the differences between the medians are significant[1]. The post-hoc analysis with the Wilcoxon signed-rank test with the Bonferroni correction shows that the differences between the cases with mutation ratio 0.01 and 0.05, $p-value = 0.0049$, are significant; whereas between the cases with mutation ratio 0.05 and 0.10, $p-value = 0.0926$, the differences are not significant.

Finally, in Fig. 5.11 the goodness of the numerical results are compared with the observed SED of the galaxy M110. By comparing Fig. 5.10 and Fig. 5.11, it can be appreciated the improvement in the fitting with the observational results. Considering that the data have not been corrected for background extinction, the adjustment can be considered as adequate for this phase of the work.



(a) Observed and simulated SED



(b) Residual (relative error) between the observed SED and the simulated SED.

Figure 5.11: Results when applying the mutation operator to the SSPs and DE to the coefficients.

[1] A confidence level of 95% (p-value under 0.05) is used in this analysis. This means that the differences are unlikely to have occurred by chance with a probability of 95%.

5.4.2.5 Larger Number of Cycles

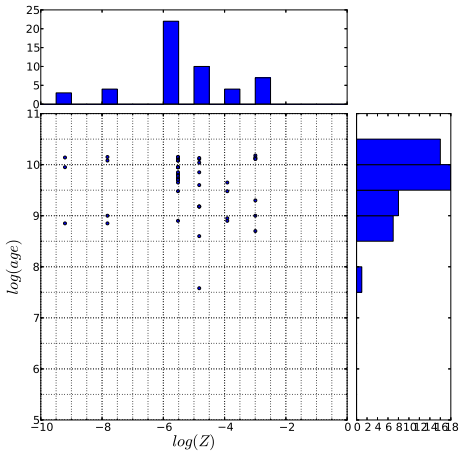
Until now the efforts have focussed on selecting the best EAs to produce high-quality suboptimal solutions. For comparison purposes, all the previous studies have been performed with 10^3 cycles. If the two SSPs obtained with the best solution of each execution are presented, a plot with some dispersion around the optimal SSPs appears (Fig. 5.12(a)). As much as the number of cycles grows up, a reduction of the dispersion is expected (Fig. 5.12). Progressively, for 10^4 cycles (Fig. 5.12(b)) and for 10^5 cycles (Fig. 5.12(c)), the dispersion diminishes but still it is appreciated. For 10^6 cycles, the plot has reduced significantly the dispersion until an acceptable limit (Fig. 5.12(d)); whereas for 10^7 cycles the scatter pattern completely disappears (Fig. 5.12(e)).

Furthermore, the increment in the number of cycles produces an improvement in the mean fitness (Table 5.7), as well as a reduction of the dispersion. This leads to select 10^6 cycles as an appropriate figure, which produces a low mean and standard deviation fitness, while keeping a limited processing time.

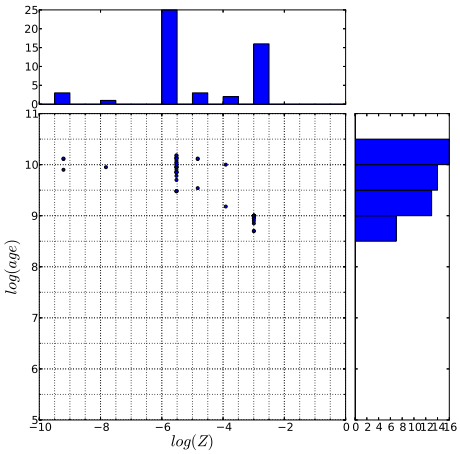
Table 5.7: Mean fitness and standard deviation from 10^3 to 10^7 cycles and 10 individuals, when using mutation operator over SSPs and DE over the coefficients.

Cycles	Mean Fitness
10^3	0.226±0.009
10^4	0.209±0.006
10^5	0.199±0.003
10^6	0.19548±0.00013
10^7	0.19535±0.00003

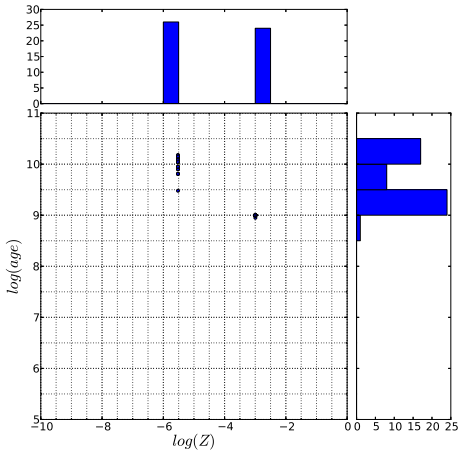
Alternatively, the dispersion can be analysed by clustering the points around centres. For clustering, k-means algorithm is used in this work [40]. The evolution of the coordinates of the two centres generated by k-means is presented at Table 5.8. It should be underlined that these centres might not coincide with a specific SSP. As can be appreciated, no evolution of the centres is produced when executing more than 10^5. Therefore, from the two last studies, it can be concluded that a number of cycles in the range from 10^5 to 10^6 seems appropriate to balance efficiency in the numerical solutions and a reduced processing time.



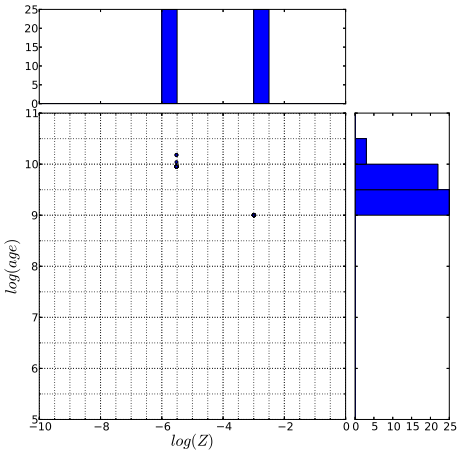
(a) 1K cycles



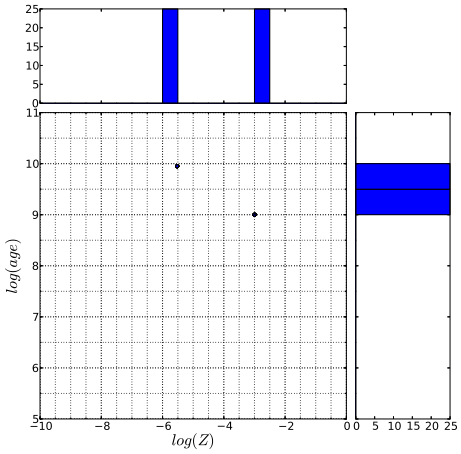
(b) 10K cycles



(c) 100K cycles



(d) 1M cycles



(e) 10M cycles

Figure 5.12: Scatter plots and histograms of SSPs (2 per solution) solutions (25 solutions) when applying the mutation operator to the SSPs selection and DE to the coefficients.

Table 5.8: Centres of the two clusters created by k-means algorithm for cycles from 10^3 to 10^7 and 10 individuals, when using mutation operator over SSPs selection and DE over the coefficients.

Cycles	(log(Z), log(age))
10^3	(-7.82, 9.52), (-3.00, 9.00)
10^4	(-5.52, 9.54), (-3.00, 8.94)
10^5, 10^6, 10^7	(-5.52, 9.95), (-3.00, 9.00)

5.5 Conclusions

The use of metaheuristics in the search of solutions for complex problems has allowed obtaining synthetic solutions from astrophysical problems when dealing with large data sets. The studies performed with rotational curves of spiral galaxies and the modelling of the low-resolution galaxy spectral energy distribution have proved the capacity of the evolutionary algorithms to solve this kind of problems.

Besides the astrophysical problems have served as benchmarks for other issues in evolutionary computing. They have been employed as real-world problems for testing the sensitiveness of the evolutionary algorithms to the choice of the random number generator, and as a test for reducing the processing time in metaoptimization processes.

Chapter 6

Application of GPU Computing to Astrophysics Problems

6.1 Introduction

RECENT progresses in observational cosmology have led to the development of the ΛCDM (Lambda Cold Dark Matter) model [35]. It describes a large amount of independent observations with a reduced number of free parameters. However, the model predicts that the energy density of the Universe is dominated by two unknown and mysterious components: the dark matter and the dark energy. These two components constitute the 96% of the total matter-energy density of the Universe.

Dark energy and dark matter have never directly been observed, and their nature remains unknown. Understanding the nature of the dark matter and the dark energy is one of the most important challenges of the current cosmology studies[1].

In most of the cases, the functions employed in the study of the dark matter distribution are computationally intensive. Taking into account that the astronomical surveys expected for the forthcoming years (Dark Energy Survey [1], the Kilo-Degree Survey [47] or Euclid [3, 54]) will enlarge from dozens of thousands up to hundreds of million galaxies and the number of accessible samples will also increase, then any improvement in the performance will be helpful to ameliorate the analysis capacity.

6.1.1 The Two-Point Angular Correlation Function

The distribution of galaxies in the Universe is one of the main probes of the ΛCDM cosmological model. One of the most important observable to study the statistical

[1]The quantification of the budget between ordinary and dark components in the Universe is a major issue as proven by the recognition of the Science magazine in 1998 and 2003 as "Scientific Breakthrough of the Year".

properties of this distribution is the Two-Point Angular Correlation Function (2PACF hereafter), which is a measure of the excess of probability, relative to a random distribution, of finding two galaxies separated by a given angular distance. By comparing different results in the correlation functions, implicit comparisons between cosmological models are made.

The 2PACF, $\omega(\theta)$, is a measure of the excess or lack of probability of finding a pair of galaxies under a certain angle with respect to a random distribution. In general, estimators 2PACF are built combining the following quantities (histograms):

- DD(θ) is the number of pairs of galaxies for a given angle θ chosen from the data catalogue D.
- RR(θ) is the number of pairs of galaxies for a given angle θ chosen from the random catalogue R.
- DR(θ) is the number of pairs of galaxies for a given angle θ taking one galaxy from the data catalogue D and another from the random catalogue R.

Although diverse estimators for 2PACF do exist, the estimator proposed by Landy and Szalay [52], (Eq. 6.1), is the most widely used by cosmologists due to its minimum variance.

$$\omega(\theta) = \left(\frac{N_{random}}{N_{real}}\right)^2 \cdot \frac{DD(\theta)}{RR(\theta)} \; - \; 2 \cdot \frac{N_{random}}{N_{real}} \cdot \frac{DR(\theta)}{RR(\theta)} \; + \; 1 \tag{6.1}$$

In Eq. 6.1, N_{real} and N_{random} are the number of galaxies in the data and random catalogues respectively.

A positive value of $\omega(\theta)$ —estimator of 2PACF— indicates that galaxies are more frequently found at angular separation of θ than expected for a randomly distributed set of galaxies. On the contrary, when $\omega(\theta)$ is negative, a lack of galaxies in this particular θ is found. Consequently $\omega(\theta) = 0$ means that the distribution of galaxies is purely random.

The calculation of 2PACF implies computing the angle among all pairs in a sample of N galaxies. As a consequence, the complexity of the calculation is $O(N^2)$.

6.1.2 The Three-Point Angular Correlation Function

The three-point angular correlation function (3PACF), $\zeta(\theta)$, is similar to the 2PACF but involving three galaxies instead of two. This modification increases the compu-

tational complexity to $O(N^3)$. In this case, the most accepted estimator is the one proposed by [93] (Eq. 6.2).

$$\zeta(\theta) = \left(\frac{N_{random}}{N_{real}}\right)^3 \cdot \frac{DDD}{RRR} - 3 \cdot \left(\frac{N_{random}}{N_{real}}\right)^2 \cdot \frac{DDR}{RRR} + 3 \cdot \frac{N_{random}}{N_{real}} \cdot \frac{DRR}{RRR} - 1 \quad (6.2)$$

where

- $DDD(\theta_1\theta_2\theta_3)$ denotes the number of triplets of galaxies for a given set of angles $\theta_1\theta_2\theta_3$, where the three galaxies are selected from the observational data catalogue, D.
- $RRR(\theta_1\theta_2\theta_3)$ denotes the number of triplets of galaxies for a given set of angles $\theta_1\theta_2\theta_3$, where the three galaxies are selected from the random data catalogue, R.
- $DDR(\theta_1\theta_2\theta_3)$ and $DRR(\theta_1\theta_2\theta_3)$ are similar to the previous ones taking two galaxies from one catalogue and the third one from the other catalogue.

6.1.3 The Shear-Shear Correlation Function

Cosmological information, such as dark matter distribution at different epochs, the amount of matter and the expansion history, is contained in the so-called shear-shear correlation function.

A thorough review on the topic of gravitational lensing can be found here [6]. The value of the shear field γ can be conveniently estimated from the ellipticity, ϵ, of a particular galaxy. Here $|\epsilon|$ is defined as such that an ellipse with axes $a < b$:

$$|\epsilon| = \frac{b-a}{a+b} \quad (6.3)$$

Given that each galaxy has an orientation ϕ with respect to the local coordinate frame, two ellipticity components can be defined.

$$\epsilon = \epsilon_x + i\epsilon_y = |\epsilon| e^{2i\phi} \quad (6.4)$$

The correlation function of these ellipticities, as a function of the separation angle between galaxies, encodes cosmological information about the mass distribution at different redshifts, see [50] for a recent interpretation of the shear correlation function. In reality, we need to extract shear *fields* averaging the ellipticities of many galaxies in every region. Just the computational problem of calculating the correlation functions is addressed here by assuming that the ellipticity ϵ has been measured to the best of our ability. The actual measurement of the shear from galaxy ellipticities is beyond

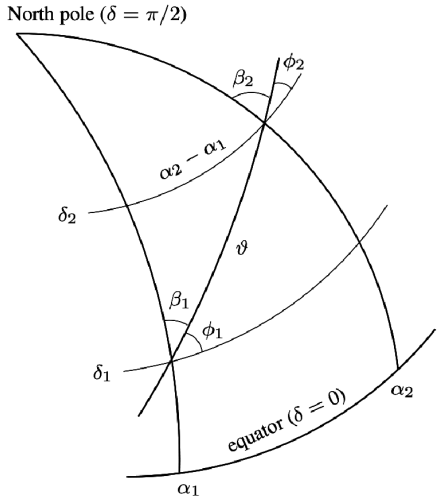


Figure 6.1: Angles and coordinates on a sphere for two galaxies i = (1,2) located at (α_i,δ_i). Figure taken from [50], used with the author's permission.

the scope of this Thesis. The reader can consult [9] and [44] for a detailed explanation of systematic effects and the steps for the extraction of shear from observations. In the rest of the document the shear notation and not the ellipticity notation is used, assuming that this process has already taken place.

The great circle distance θ between two galaxies i=(1,2), necessary for the correlation function binning, is calculated using the position vectors of both on a unit sphere, which are obtained from its spherical sky coordinates α_i,δ_i.

$$\vec{v_i} = (\cos\alpha_i \cdot \cos\delta_i, \sin\alpha_i \cdot \cos\delta_i, \sin\delta_i) \tag{6.5}$$

$$\cos(\theta) = \vec{v_1} \cdot \vec{v_2} \tag{6.6}$$

The shear at a particular galaxy position is defined in a local Cartesian coordinate system with the y-axis pointing towards the north pole and the x-axis going along the line of constant declination in a plane tangent to the sphere at the galaxy's position. Given a pair of galaxies $(1, 2)$, γ_{t_1} is the tangential projection of the shear of galaxy 1 along the geodesic that connects galaxy 1 and 2, and $\gamma_{\times_1}$ is the cross component. To be able to calculate these shear components, the angle β_1 (see Figure 6.1) must be known, this is the angle between the great circle at declination δ and the right ascension of the galaxy α_1. Then the angle (known as the *course angle*) that we need to use to project is given by $\Phi_1 = \pi/2 - \beta_1$. Using the sine and cosine rules on a sphere the calculation is done as follows:

$$\begin{aligned} \cos\Phi_1 &= \frac{\sin(\alpha_2-\alpha_1)\cos\delta_2}{\sin\theta} \\ \sin\Phi_1 &= \frac{\cos\delta_2\sin\delta_1-\sin\delta_2\cos\delta_1\cos(\alpha_2-\alpha_1)}{\sin\theta} \end{aligned} \tag{6.7}$$

The corresponding angle for Φ_2 can be found by exchanging the indices.

After projecting the measured shears (γ_x,γ_y) to (γ_t,$\gamma_\times$), the following correlation functions can be defined:

$$\begin{aligned} \xi_+(\theta) &= \frac{\sum_{ij} w_i w_j(\gamma_t(\theta_i)\cdot\gamma_t(\theta_j)+\gamma_\times(\theta_i)\cdot\gamma_\times(\theta_j))}{\sum_{ij} w_i w_j} \\ \xi_-(\theta) &= \frac{\sum_{ij} w_i w_j(\gamma_t(\theta_i)\cdot\gamma_t(\theta_j)-\gamma_\times(\theta_i)\cdot\gamma_\times(\theta_j))}{\sum_{ij} w_i w_j} \\ \xi_\times(\theta) &= \frac{\sum_{ij} w_i w_j(\gamma_t(\theta_i)\cdot\gamma_\times(\theta_j))}{\sum_{ij} w_i w_j} \end{aligned} \tag{6.8}$$

where $w_{i,j}$ are the weights associated to the measurement of galaxy ellipticity. These take into account measurement errors, see [43]. These are the three correlation functions that are calculated by the code.

6.2 Related Work

6.2.1 Related Work for the Two-Point Angular Correlation Function

The previous efforts done in the acceleration of the analysis of the distribution of galaxies can be classified in two categories. On the one hand, it can be mentioned the implementations of the 2PACF problem into more powerful computing platforms: FPGA [51], GPU [5, 84]. And, on the other hand, it can be cited the use of some tricky mechanism to reduce the complexity of the calculation without losing too much accuracy, i.e. k-trees [68] or pixelization [29].

Comparing with [84], the implementation of the kernel proposed in this Thesis is more compact —fewer number of lines—, and more versatile —easily adaptable to the study of other range of angles—. Whereas in [84] a water-fall-if-elseif structure is the core of the assignation of the angular coincidences to the bin, in our implementation is the atomicAdd() function. The water-fall-if-elseif structure implies less flexibility to modifications to the angular range of the supported histogram as well as diminishing the readability of the kernel due to the larger number of code-lines, making the code less compact.

Some more points in common appear in [5], such as: use of atomicAdd() function, and allocation of sub-histogram in shared memory; although also presents differences: no use of atomicAdd() for the final gathering of the sub-histograms in the final histogram. In this work, the gathering of the sub-histograms is done in two step, firstly the sub-histograms are gathered in global memory, and later the results from global memory are accumulated in CPU. This procedure stems from the division of the data for transferring to the kernel. In this work, the data are segmented and analysed by chunks, whereas in our implementation all the data are transferred and analysed to the GPU without dividing into chunks. This publication appears latter than our publications about the 2PACF.

Additionally, in [5] the aperture mass statistic is also calculated. This is an alternative approach to investigate the presence of dark matter.

6.2.2 Related Work for the Shear-Shear Correlation Function

Concerning the shear-shear correlation function, previous efforts implement some kind of mechanism to reduce the computational cost of the point-to-point correlation estimation, for example, the widely used *ATHENA* code[1]. This is a powerful tool based on kd-trees [68], which allows controlling the precision of the estimation by means of a parameter termed *opening angle* measured in radians (OA hereafter). This parameter regulates the minimum angle at which two kd-trees nodes must 'see' each other for the full point-to-point correlation to be estimated. If the nodes are far away from each other, an averaging of the values is performed and these averages are then correlated. Smaller opening angles make the required block size smaller, the precision achieved is higher at the expense of a higher execution time. A similar approach is used in [46].

6.2.3 Related Work for the Improvement in the Precision of Histogram on GPU

A standard and essential reference for histogram construction on GPU is the white paper from NVIDIA about this topic [76]. Although originally devoted to image processing and data mining, finally it has served as a guide for many other scientific areas. In this white paper, 64-bin and 256-bin histogram implementations are proposed and evaluated. Nowadays, the hardware has evolved enough to allow for larger histogram sizes.

[1] http://www2.iap.fr/users/kilbinge/athena/

The proposed implementation [76] is the "classical" one: per-block sub-histograms on shared memory. In this approach, the sub-histograms are created on shared memory, and later they are gathered on global memory to form the final histogram.

In this work, two more variants are proposed: per-thread and per-warp sub-histograms. Depending on the architecture of the GPU, the histogram size and the data size, these strategies might be a limiting factor in the performance and in the capacity to contain without overflowing the counts in the bins (hereafter bin containment).

In [87], the authors claim two new efficient methods for histogram calculation on GPU. This article uses the NVIDIA white paper as a starting point to propose improvements. However, from the two implementations for histogram construction presented in [76], the authors only cite the smallest one in capacity. In any case, this paper was written when compute capacity was 1.0, and no atomic operations on shared memory were available. Therefore, it can be considered an obsolete strategy.

In [86], other efficient implementation to compute image histogram on GPU is proposed. Among other considerations about grey-scale image manipulation, the authors make considerations about the precision and the bin containment. In this paper, an implementation which is unable to accumulate more than 256 counts per bin is compared with a new one. In order to mitigate the lack of containment, local histograms are created. However, this new implementation introduces errors when accumulating more than 2048 counts per bin. This is clearly insufficient for cosmological analysis such as: 2PACF, 3PACF and shear-shear correlation, as it will be later shown. Neither of these implementations expose the use of atomic functions nor shared memory to enhance the performance.

In the book [85], a very didactic and efficient implementation of per-block sub-histogram on shared memory is described. This implementation presents great advantages: it is easy to implement, widely applicable to many different disciplines, it has a great performance and bin containment. This implementation seems inspired in the per-block implementation of the white paper [76], but incorporating atomic functions and using shared memory for storing the intermediate sub-histograms and, hence, for improving the performance. This implementation has been used by the authors in previous works, such as: the analysis of the 2PACF [13, 17, 18, 77] and the shear-shear analysis [14]; at the same time, it is employed for comparison purposes in the current work.

In [39] a per-block sub-histogram implementation is proposed. The novelty of this approach relies on the multiple sub-histograms which are embodied in a single thread block. This approach is an extension of Nugteren et al. approach (1 sub-histogram per thread block) [71] to include multiple sub-histograms per thread block. As a final result,

the implementation is a hybrid between per-warp and per-block sub-histogram implementations. Given that this implementation gathers the sub-histograms in the final histogram on global memory, it will face difficulties with the largest number of counts per bin attainable for some number-representations. On the other hand, it will suffer from lack of precision when adding small and large numbers in float-representation.

The comparison between the methods proposed by Shams et al. [87] and by Nugteren et al. [71] shows that the different application areas (data mining and image processing) establish different requirements about the histogram size, larger for data mining than for image processing. The input sizes are also different: 8-bits are enough for image processing, whereas, 32-bits are typical in data mining. This restricts the performance study towards different objectives than in cosmology, and therefore, it forces to implement modifications.

In [71] two histogram implementations are proposed: a per-warp sub-histogram and a per-thread sub-histogram. When processing images, a successful and simple proposition to improve the performance is to shuffle the input data. This is useful for real images, however, it does not affect when processing cosmological input data. In real images, it is probable that close pixels will feed the same bin; and therefore, they will generate collisions (sequential updates of the number of counts in the same bin). Thus, this action will degrade the performance. However, for cosmological inputs, this a priori knowledge of the data structure does not exist, except for the case that the data has been previously ordered.

Neither of the mentioned approaches propose simultaneously modifications on both: the kernel and the CPU-part of the code, nor profit from double-representation accessible on the CPU-part of the code[1]. Furthermore, in all the previous implementations, after constructing the sub-histograms they are gathered in the final histogram on the GPU. However, in our implementation, the final gathering is performed on the CPU-part benefiting of the double-representation to improve the bin containment. Additionally, no considerations about the input size and how it impacts on the precision of the final result are presented in the works mentioned in this section.

In contrast to image processing, cosmological analysis requires trigonometric calculations before feeding the appropriate bin in the histogram. For this reason, performance comparison between the mentioned works and the current work for cosmological problems is not feasible.

Regarding the precision of floating point representation on GPU, a review of this issue is presented at [99]. In this work the inexactnesses associated with the use of float representation (operation accuracy and rounding), and how the programming affects

[1] AtomicAdd() is not available in double precision.

the final result is underlined. Furthermore, in [38] a complete description of the floating point format and its weaknesses are fully described.

Concerning other cosmological studies, in [5] the authors present an alternative approach to calculate the 2PACF. In the description of the implementation, the use of integer-representation for the histogram (256 bins and logarithmic binning), as well as the use of *atomicAdd()* function and shared memory for supporting the sub-histograms are enlightened. Although, they express that the final aim is to be able to process up to billions of galaxies with this code, the largest data set processed is composed of one million of galaxies. From the description of the implementation, it can be stemmed that similar difficulties for the bin containment as the float-based implementation will arise when increasing the size of the data set. The lack of precision in this implementation is mitigated because the data are processed in bunches, and then accumulated.

Other implementation of the 2PACF is presented in [84]. This is the most different approach. It uses an array in double-representation to hold the histograms, however it impedes to use *atomicAdd()* functions. Therefore, the selection of the bin has to be done in an alternative manner. In this case, it is performed by a water-fall-if-elseif structure. This strategy saves the problem of the lack of precision of other number-representations, but it severely penalizes the performance of the application.

6.3 Application of GPU Computing to the Two-Point Angular Correlation Function

One of the main objectives of this work is to produce an implementation for the 2PACF calculation, which can be executed on GPU hardware. The GPU election is based on the capability to accelerate the execution as well as the low price of the hardware. This makes high performance computing accessible for small-sized research groups.

6.3.1 GPU Implementation of 2PACF

First of all, the initial strategy for the GPU code pays special attention to the use of *shared memory*. For this reason, the dot product and the arc-cosine calculation for each pair of galaxies are implemented in this type of memory. This avoids the use of the *global memory* —much slower than *shared memory*— for any intermediate calculation which requires frequent read and write processes.

Other expected bottleneck is the construction of the histograms: DD(θ), RR(θ) and DR(θ). Following the sequence of the commands in the kernel, until this point a multithread calculation has operated over the pairs of galaxies, calculating the dot product, next the arc-cosine and, finally, the bin in the histogram where an account

ought to be incremented. But, due to the multithreaded nature of the kernel, simultaneous updates of the same bin in the histogram must be avoided in order to do not miss any count. This forces to use atomic functions to create the histogram or alternatively water-fall-if-elseif structures.

The use of water-fall-if-elseif structures implies strong drawbacks. It is less flexible to changes in the range of angles of the histogram, and forces to recompile after any modification. With implementing atomic functions, the range of angles for the histogram can be introduced during the invocation.

Besides, the water-fall-if-elseif structures produce less compact kernels —larger number of lines—, thus making more difficult the readiness of the code. By using atomic functions instead water-fall-if-elseif structures reduces significantly the number of lines of the code, so it eases its readiness and its maintenance.

The atomic functions for integers are supported by NVIDIA GPU for compute capability 1.1 and higher in *global memory*; and for compute capability 1.2 or higher in *shared memory* [85]. In our case, the appropriate function is *atomicAdd()*.

The use of atomic functions in *global memory* causes a major performance degradation. Therefore, the solution is to perform more atomic operations in parallel. For this reason, an alternative strategy has been followed. Each block of threads implements a histogram in *shared memory*; and in these histograms, angles are binned in parallel. Next, all shared-memory-built-histograms are reduced to a single one in *global memory*. This strategy produces a parallel treatment of the most critical operation, being the foundation of the success of the original GPU implementation.

On the other hand, the baseline code implements a coalesced pattern access to the global memory. This is achieved by disposing the x-coordinates of all galaxies in a single array, and similarly for y and z-coordinates. By implementing this data layout, adjacent threads in a block request contiguous data from global memory. Coalesced access maximizes global memory bandwidth usage by reducing the number of bus transactions.

Regarding the constrains, the histogram size is fixed at 64 degrees with 4 bins per degree. This size is selected as a mean of he recommendations of the final users. Slight variations on the histogram size or the data catalogues severely modify the overall performance of the algorithm, difficulting the comparison with other works. Indirectly this constrain eliminates the choice to optimize the code by reshaping the threadblock size (aiming to maximize the occupancy).

6.3.2 Initial Results

In order to test the capacity of the GPU implementation to accelerate the 2PACF, comparative tests are executed among CPU, GPU and multi-GPU —using 3 GPUs—

; and OpenMP, MPI and hybrid MPI-CUDA implementations. MPI and OpenMP implementations are executed at Euler cluster (section 2.3.2).

In the study of the 2PACF, two input files coming from simulations have been employed: MICE 0.35 (430,931 galaxies) and MICE 0.55 (654,094 galaxies)[1].

The results of the comparisons among the implementations are presented at Table 6.1. The analysis of these results shows the excellent speedup obtained by the GPU implementation —115.15±4.84— in opposition to the speedup obtained by the OpenMP implementation —10.58±0.48—. It should be underlined that the GPU implementation is fast enough to be competitive, and for this reason, to be used by small research groups lacking in more expensive equipment such as clusters or supercomputers.

Table 6.1: Mean execution time and speedup for CPU, GPU, Multi-GPU and OpenMP implementations for the 2PACF.

Implementation	Mean execution time (s)	Speedup
CPU	35,186.327 ± 1,452.419	
OpenMP	3,326.363 ± 28.498	10.580±0.478
GPU	305.570 ± 1.143	115.154±4.835
Multi-GPU (3 GPUs)	184.108 ± 2.127	191.174±8.897

Additionally to the previous implementations, an MPI one is also created and tested. In Table 6.2, the execution time and the speedups of the MPI implementation are shown.

As can be appreciated, the behaviour of the speedup is close to theoretical maximum speedup for small and mid-sized configurations (up to 32 cores). As far as the number of cores increases, the speedup deviates from the theoretical maximum obtaining a poor performance. This degradation of the performance occurs because communication time becomes relevant in comparison with computing time. Larger input files mitigate this degradation by incrementing the computing time in comparison with communication time. A similar effect will be observed later with the hybrid MPI-CUDA implementation (Table 6.3).

In any case, as the input size expected in the future is much higher that the sample file employed in this work as benchmark, the severity of this effect will be drifted towards larger number of nodes in both implementations.

[1]We acknowledge the use of data from the MICE simulations, publicly available at http://www.ice.cat/mice.

Table 6.2: Mean execution time and speedup for diverse numbers of cores in the MPI implementation for the 2PACF.

Cores	Mean execution time (s)	Speedup
1	23,712.281 ± 145.690	
2	11,652.683 ± 59.959	2.035 ± 0.014
4	6,327.005 ± 68.821	3.748 ± 0.047
8	3,162.222 ± 34.165	7.499 ± 0.084
16	1,577.899 ± 52.818	15.046 ± 0.562
32	799.457 ± 11.793	29.667 ± 0.455
64	403.937 ± 6.044	58.716 ± 0.965
128	205.008 ± 4.262	115.713 ± 2.423
256	104.040 ± 6.493	228.796 ± 14.210
512	71.292 ± 2.841	333.090 ± 12.301

Finally, taking into consideration the large input files expected (up to two orders of magnitude larger than the sample used in this work), a hybrid MPI-CUDA implementation is also created to cope with those executions. It has to be underlined that even the GPU implementation forecasts execution times of days for files of tens of millions of galaxies.

In Table 6.3, the execution times for two input files MICE 0.35 and MICE 0.55 are shown. This production has been executed on a cluster with 8 nodes, each one with a M2050 card (section 2.3.3.2). The different sizes of files will allow comparing the execution times when supplying different computational charge to each node.

The analysis of these results shows an excellent speedup in most of the cases, except for the MICE 0.35 file when executing on 8 nodes. In this case the communication time becomes relevant in comparison with the calculation time. Fortunately, this tiny degradation of the performance will become irrelevant when handling files with tens of millions of galaxies, as can be appreciated when analysing the file MICE 0.55. This second file is, mildly speaking, a 50% larger than the MICE 0.35. As an example, the comparative results for 8 nodes show a higher speedup for the analysis the file MICE 0.55 than for MICE 0.35. Essentially, for 8 nodes the execution time for each chunk in the later case is relevant in comparison to the communication time, where as for larger files is less relevant.

Table 6.3: Mean execution time (s) for the single precision MPI-CUDA implementation of the 2PACF for 1, 2, 4, and 8 nodes for MICE 0.35 and MICE 0.55.

	MICE 0.35		MICE 0.55	
Nodes	Execution Time	Speedup	Execution Time	Speedup
1	301.473±0.232		711.702±0.474	
2	151.622±0.498	1.988	349.279±0.506	2,037
4	78.907±0.461	3.821	181.331±0.073	3,925
8	43.273±0.037	6.967	94.274±0.035	7,549

6.3.3 Code Optimization

In order to improve the efficiency of the 2PACF code, an optimization process is performed. Diverse strategies are tested, but only the following ones present a positive impact in the execution time: the use of streams, reducing branching, and increment the occupancy and the data locality.

Use of Streams and Asynchronous Copy. In the sequential part of the code, the use of *streams* can be a key element to accelerate the code. A CUDA *stream* is a queue of GPU operations that are executed in a desired order. In this queue, operations such as data transfers between the host and the device, and kernel invocations can be gathered. By creating more than one *stream* and distributing appropriately the tasks among them, some extra speedup might be achieved. Simultaneously with the use of *streams*, the standard and more usual version of the *cudaMemcpy()* command should be replaced by *cudaMemcpyAsync()*. In executing this replacement, the transference of data between the host and the device changes from a synchronous process to an asynchronous one. It has to be remarked that kernel invocation is always an asynchronous process.

Reducing Branching. In reducing the thread divergence, the number of serialized threads is diminished, and an improvement of the performance is expected. This strategy leads to substitute the if-conditionals by a function calculating the minimum between the number involved and the number one. In CUDA, for the single precision implementation, the appropriate function is *fminf*; whereas for the double precision implementation is *fmin*.

Increasing the Active Blocks per Streaming Multiprocessor. Other aspect to take into account during the optimization is to overcome the factors limiting the

kernel execution. In our case the profiling in the GTX295 shows that the main limiting factor is the high number of registers used by thread. For this reason, the target proposed is to reduce the number of registers used per thread. Following this recommendation, some variables are removed, being this value calculated when necessary. Besides some variables are switched from shared memory to registers. With this change the stress over the on-chip memory is more correctly balanced.

Table 6.4: Mean execution time (ms), reduction of the execution time and speedup in comparison with original code of the 2PACF and when implementing all the positive strategies: the use of streams, reducing branching, increment the occupancy and the data locality.

Implementation		
Single Precision, Compute Capability 1.2, GTX295	Original Code	299,566.4±15.3 ms
	Positive Strategies	275,303.8±17.6 ms
	Reduction	**24,262.6**
	Speedup	**1.09**
Single Precision, Compute Capability 2.0, C2075	Original Code	314,346.8±199.6 ms
	Positive Strategies	269,969.5±69.8 ms
	Reduction	**44,377.3**
	Speedup	**1.16**
Double Precision, Compute Capability 2.0, C2075	Original Code	452,097.3±287.2 ms
	Positive Strategies	438,934.0±132.2 ms
	Reduction	**13,163.4**
	Speedup	**1.03**

The mentioned modifications differ in the degree of success (Table 6.4). As can be appreciated, for single precision the modifications have a higher impact over the execution time than for double precision.

Fortunately the use of double precision is only required when binning angles lower than 0.003 degrees. This happens when changing the histogram setup, and the researchers require a high detail level for very short range of angles, for instance from 0 to 4 degrees with 256 bins.

The optimization processes presented in this section aim to be generalist, in such way that the resulting code will be widely applicable. No optimization processes associated to the particularities of the input file, e.g. equality of the number of galaxies in the real and random data sets; or the region of the space covered have been applied until this point, e.g. simplify the arc-cosine calculation by a polynomial series expansion.

6.3.4 Concurrent Computing Optimization

Nowadays many computational systems are endowed of multi-cores in the main processor units, and one or more many-core cards. This makes possible the execution of codes on both computational resources concurrently. The challenge in this scenario is to correctly balance both execution paths. When the scenario is simple enough, by-hand optimization can be affordable, otherwise metaheuristics techniques are mandatory.

The maximization of the exploitation of the resources on an heterogeneous system, multi-core CPU and many-core GPU, requires an optimum balance between the execution time of the tasks assigned to CPU and to GPU. This forces to carefully select the amount of data analysed in each resource, which fairly balances both execution paths. Otherwise, an important penalization in the execution time might be produced.

In the initial version of this code, the amount of data analysed in CPU and GPU has been governed by a single parameter. This parameter governs the percentage of galaxies analysed in CPU, while the remaining galaxies are analysed in GPU. This single parameter is applied to the three histograms that have to be built for the calculation of the 2PACF.

The works described in this section have been performed by using a C2075 card (section 2.3.3.2).

6.3.4.1 Single Percentage Implementation

In the initial strategy, the percentage of data analysed in GPU and CPU is governed by a single parameter for the three histograms.

Concurrent computation is possible because the construction of each histogram can be split in partial histograms. Firstly, input data are split in two chunks. These chunks are assigned to both computational resources: CPU and GPU, where the corresponding partial histograms are constructed. At the end of the process, both partial histograms are merged in the CPU. In the CPU part, a parallel implementation based on OpenMP is performed, whereas in GPU is based on CUDA.

When using a single percentage for the concurrent execution of the histograms, the fitness landscape presents a minimum in the range from 9% to 11% (Fig. 6.2). Unfortunately, the percentages that exhibit the lowest values (11%) and the lowest median (10%) do not coincide. This indicates the difficulty of the decision-making process about the most suitable percentage to minimize the execution time.

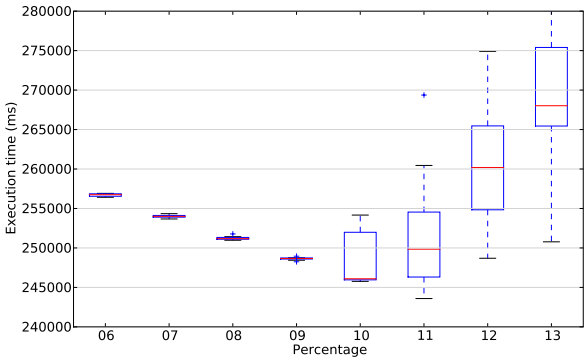


Figure 6.2: Execution time (ms) of 2PACF for concurrent execution by using a single percentage (12 executions per case).

6.3.4.2 Multiple Percentages Implementation

By measuring the execution time of each histogram construction when being executed completely in GPU, it has been proved that some of them take longer than others. These execution times are related to the nature of the data which the histogram analyses (positions of galaxies in the sky). The differences underlie on the data representation: on the one hand, galaxies randomly distributed on the sky, and on the other hand, galaxies distributed in clusters and superclusters following a particular cosmological model. These differences impact over the execution times through the construction of the histograms.

When histogramming galaxies with cosmological structure (clusters and superclusters of galaxies), most of the galaxies feed a reduced number of bins in the histogram. Then, the code must serialize a lot of increments in few bins. As a consequence, this produces an increment in the execution time. Oppositely, for random data the construction fairly distributes the counts in all bins, and therefore, less serialization is produced. In this case, the construction of the histogram operates with a higher degree of parallelism than in the previous case.

This scenario indicates that the strategy followed until this point —a single percentage for all the histograms— is quite naive. A single percentage does not balance correctly both execution paths in the histograms. Consequently, the most appropriate strategy is to propose independent percentages for each histogram. However, this increment in the number of parameters to optimize, in practice, impedes to fit the values by-hand, and makes necessary the use of evolutionary techniques for finding suitable values for them.

By dividing the previous single percentage into three percentages, one per concurrent part of each histogram, a most suitable matching between the execution times of both computational paths is expected, and finally an extra reduction in the execution time. Although, the three parameters can be fitted separately, by freezing two histograms constructions and optimizing the remaining one, the relatively narrow time slot assigned to the optimization forces to run the optimization as a whole. Thus, each finished run produces a potential solution for later verification process.

Due to its very simple and flexible implementation, Differential Evolution algorithm (DE) is usually proposed as first attempt to solve complex optimization problems. Moreover, DE is able to produce high-quality suboptimal solutions with a limited execution time budget.

Particularly, Python has been selected as programming language for the DE implementation (schema DE/rand/1/bin). Python allows manipulating pieces of text, composing files with these pieces; and then compiling the source code, executing it and capturing output information from the execution. So, the proposed Python-DE implementation is able to deal with pieces of text, which conform the hybrid CUDA-OpenMP source code.

Once the source code has been correctly assembled, Python implementation takes care about compiling the source code, to execute it, and finally, to capture the execution time. This last value corresponds to the fitness of each vector of the DE population. By repeating this process along the population and the generations, the code produces a set of optimized values for the three percentages.

Due that each 2PACF run takes around 250 seconds, the estimation of the execution time of the whole process (10 vectors and 10 generations) will increase two orders of magnitude in relation to a single 2PACF run.

When implementing the optimization process, each execution of the DE code produces a candidate solution composed of three percentages which minimize the execution time (Fig. 6.3). The analysis of the results indicates a tendency towards larger values of %DD than for the other %RR and %DR. This result is consistent with the fact that DD histogram construction should take more time due to the galaxy clustering around low angles.

Due that RR and DR histograms calculations involve random data, a lower number of serializations on the bins construction is expected. Therefore, these histograms should be faster than DD histogram calculation, and consequently to get lower percentage of data processed concurrently in the CPU.

By observing Fig. 6.3 and Table 6.5, it can be stated that a tendency towards graded values of the percentages: %DD > %RR >%DR is rawly achieved. The results

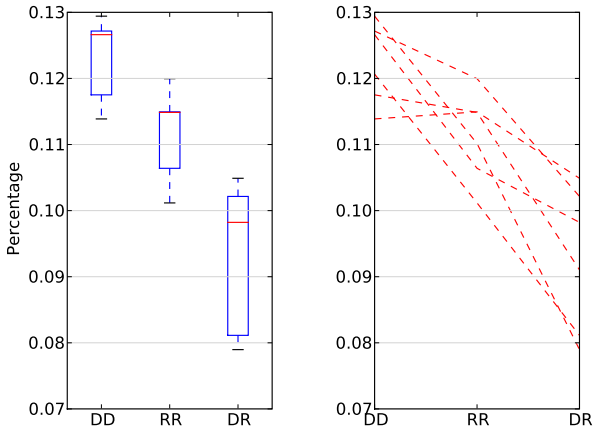


Figure 6.3: Panel (a, left) shows the comparative box plots for the percentages of DD, RR and DR, while panel (b, right) shows the lines endorsing each particular realization.

mostly reproduce the structure expected for the percentages. However, an in-depth insight to the individual results (Right Panel in Fig. 6.3) demonstrated that the current implementation does not always produce sets with this schema, otherwise other schemas as %DD <%RR appear.

In order to check the quality of the achieved solutions, a new production with each particular percentage values set of %DD, %DR, and %RR obtained as optimizer solutions, is performed with 12 executions per case. In the two last columns of Table 6.5, the mean execution time (fitness) achieved, and the speedup when comparing with the case of a single percentage with the lowest median (10%) are presented.

For some cases: R1, R3 and R4 the reduction of the execution time is relevant; whereas for other cases: R2 and R6 it is almost negligible. Finally, two cases: R5 and R7 do not produce any improvement in the execution time, even they take longer than the best case of single percentage (10%). These results underline the difficulty associated to obtain further reduction in the execution time of the optimized 2PACF code. Moreover, the success cases demonstrate that the use of three percentages with the appropriate values face to a single percentage improves the productivity of the code.

In order to produce higher-quality solutions, diverse actions can be applied to the optimizer. Probably an increment in the number of cycles or in the population size might mitigate the adverse scenario, but at the same time, it will increment critically the optimizer execution time. Therefore, at this point a balance between the quality of the suboptimal solution and the execution time is mandatory.

Table 6.5: Numerical results of the production with 10 vectors and 10 cycles: identifier, percentages of the best solution achieved, fitness (execution time of 2PACF in ms) of the best solution achieved, execution time of the optimizer run, mean and deviation standard of the fitness (execution time of 2PACF in ms) after 12 runs for this particular percentages set, and speedup (compared when using a single percentage, 10%).

Id.	%DD	%RR	%DR	Best Fitness	Optimizer Execution Time	Mean Fitness 12 executions	Speedup
R1	12.66	10.64	9.82	241,889	1,029m11.296s	242,762±1,600	1.022
R2	11.39	11.49	10.49	242,692	1,026m41.409s	247,506±4,697	1.003
R3	12.94	11.01	7.90	242,811	1,043m30.935s	244,369±2,467	1.016
R4	12.06	10.12	8.11	244,091	1,053m57.763s	244,410±151	1.016
R5	12.72	12.00	10.21	242,427	1,041m49.962s	250,555±5,854	0.991
R6	11.75	11.49	9.10	243,012	1,041m46.596s	246,572±2,575	1.007
R7	10.40	11.60	9.37	244,597	1,043m14.309s	249,483±5,513	0.995

Table 6.6: Numerical results of the production with 20 vectors and 10 cycles: identifier, percentages of the best solution achieved, fitness (execution time of 2PACF in ms) of the best solution achieved, and execution time of the optimizer run.

Id.	%DD	%RR	%DR	Best Fitness	Optimizer Execution Time
R1	12.56	10.56	11.38	241,399	2,074m29.396s
R2	12.76	9.04	9.91	242,704	2,082m44.008s

As part of the production, two runs are executed doubling the population (Table 6.6). This increment results in an extra improvement of the fitness achieved, but unfortunately, the execution time is doubled too. This increment makes unfeasible to proceed with more improvements in the fitness through incrementing the population or the number of cycles.

When statistically analysing the numerical results of each production with the Wilcoxon signed-rank test, only the productions R1, R3 and R4 state that the differences are significant for a confidence level of 95% (p-value under 0.05). This means that the differences are unlikely to have occurred by chance with a probability of 95%.

Moreover, in order to assess if the performance is better than when using the best case of single percentage (10%), the sign test can be used. The analysis indicates that for the percentages obtained at R1, R3 and R4, the 2PACF execution takes shorter than the previous best case, single percentage at 10%.

6.4 Application of GPU Computing to Shear-Shear Calculation

6.4.1 General Description of the Program Flow

The code consists in the calculation of the quantities: ξ_+, ξ_-, $\xi_\times$ (Eq. 6.8, Algorithm 1) as a function of the separation angle θ between the galaxies. This initial version of the code focuses on the calculation rather than on the performance. However it has been coded keeping in mind the most general recommendations for reaching the highest efficiency.

Algorithm 1 The shear-shear correlation algorithm pseudocode

```
foreach Pair of Galaxies do
    Calculate the separation angle θ between the galaxies on the sphere (dot product);
    if θ is in the user's range then
        Calculate all products of local components of the shears for both galaxies (g_[1,2][1,2] = γ_x;1,2 · γ_y;1,2);
        Calculate course angles for both galaxies (Φ1 and Φ2) (Eq. 6.7); (angle with respect to line of equal declination) ;
        Calculate ξ+, ξ−, ξ× (Eq. 6.8) after projecting the shears to the tangential and crossed components γt, γ×;
        Populate the histograms held in shared memory with the computed value of ξ+, ξ−, ξ×, number of pairs, and number of pairs with their corresponding weights;
    Load the histograms held in shared memory into global memory.
```

Once the separation angle θ of the galaxy pair is calculated and if it is within the histogram range (user-defined), five values have to be computed and incorporated in equal number of histograms. These values correspond to: the number of pairs of galaxies, ξ_+, ξ_-, $\xi_\times$ (Eq. 6.8) and the sum of weights of all the pairs. For the calculation of the values and in order to avoid slow access to global memory, intermediate reusable values are stored on shared memory, and the coordinates and ellipticities of the galaxies on registers.

6.4.2 Memory Management

The baseline code implements a coalesced pattern access to global memory. Input data are sorted by components rather than by galaxies: first the x-coordinates for all galaxies, and successively the y-coordinates, the z-coordinates, γ_x, γ_y (values of the measured shear field in the local reference frame) and the weight values. By implementing this

layout, adjacent threads in a block request contiguous data from global memory. Coalesced access maximizes global memory bandwidth by reducing the number of bus transactions.

Furthermore this baseline code pays special attention to making an intensive use of shared memory for intermediate calculations and for the construction of the correlation function histograms (Algorithm 1). The dot product and the arc-cosine calculation, necessary to get the angle subtended by each pair of galaxies, are executed on shared memory. The same is true for the calculation of the course angles for both galaxies and the projection of the shears to the tangential and crossed components. This avoids the use of global memory which is much slower than shared memory for any intermediate calculation which requires frequent read and write accesses.

An expected bottleneck is the construction of the histograms. Until this point a multithreaded calculation has operated over the pairs of galaxies calculating the dot product followed by the arc-cosine and finally the bin in the angle histogram where the value has to be incremented[1]. Due to the multithreaded nature of the kernel, simultaneous updates of the same bin in the histogram must be avoided in order not to miss any count. This leads to the usage of atomic functions[2] to create the histograms. Alternatives to atomic functions exist, for example water-fall-if-elseif structures. However they imply less flexibility to modifications to the angular range of the supported histogram as well as they diminish the readability of the kernel due to the larger number of code-lines, making the code less compact.

A known drawback of atomic operations is that when two threads are trying to update a value in the same bin, the operations are not parallel but sequential. Therefore if millions of threads are accessing at most a hundred bins then the serialization of the access will severely impact the performance. In order to overcome this bottleneck the histogram construction can be parallelized by means of constructing sub-histograms on shared memory and later gathering them on global memory. This mechanism increments the parallelism of the kernel and diminishes the impact of sequential operations on performance.

The initial description of the code focused on an intensive use of shared memory for intermediate operations. In order to avoid overloading it, registers are used to store relevant data for the calculation in process. Registers have a higher bandwidth than

[1]For the sake of simplicity, only the angle histogram is considered in this reasoning, but the solution is equally applied to the other histograms.

[2]The atomic operation for float on shared memory is supported for compute capability 1.3 and higher [85].

shared memory but their size is smaller. Data frequently accessed[1] for readout are stored in registers such as galaxy coordinates, ellipticities and the weight value.

Unfortunately this strategy is not exempt from drawbacks. The increment in the usage of registers can force the reduction of the occupancy, less streaming processors are active at the same time. Therefore the volume of information migrated towards the registers should be fitted carefully in order to avoid any harm to the performance of the code. Several tests were performed with incremental use of registers until the performance reached its optimum.

The baseline code has a consumption of 15.36 Kbytes of shared memory and 63 registers per thread achieving an occupancy of each multiprocessor of 25%.

Due to the fact that the correlation function needs to be studied at small separation scales double precision is used. The only exceptions are the quantities added to the histograms ξ_+, ξ_- and $\xi_\times$ because atomic operations in double precision in shared memory are still not supported. In all cases the numerical experiments have been performed with CUDA 5.0 release.

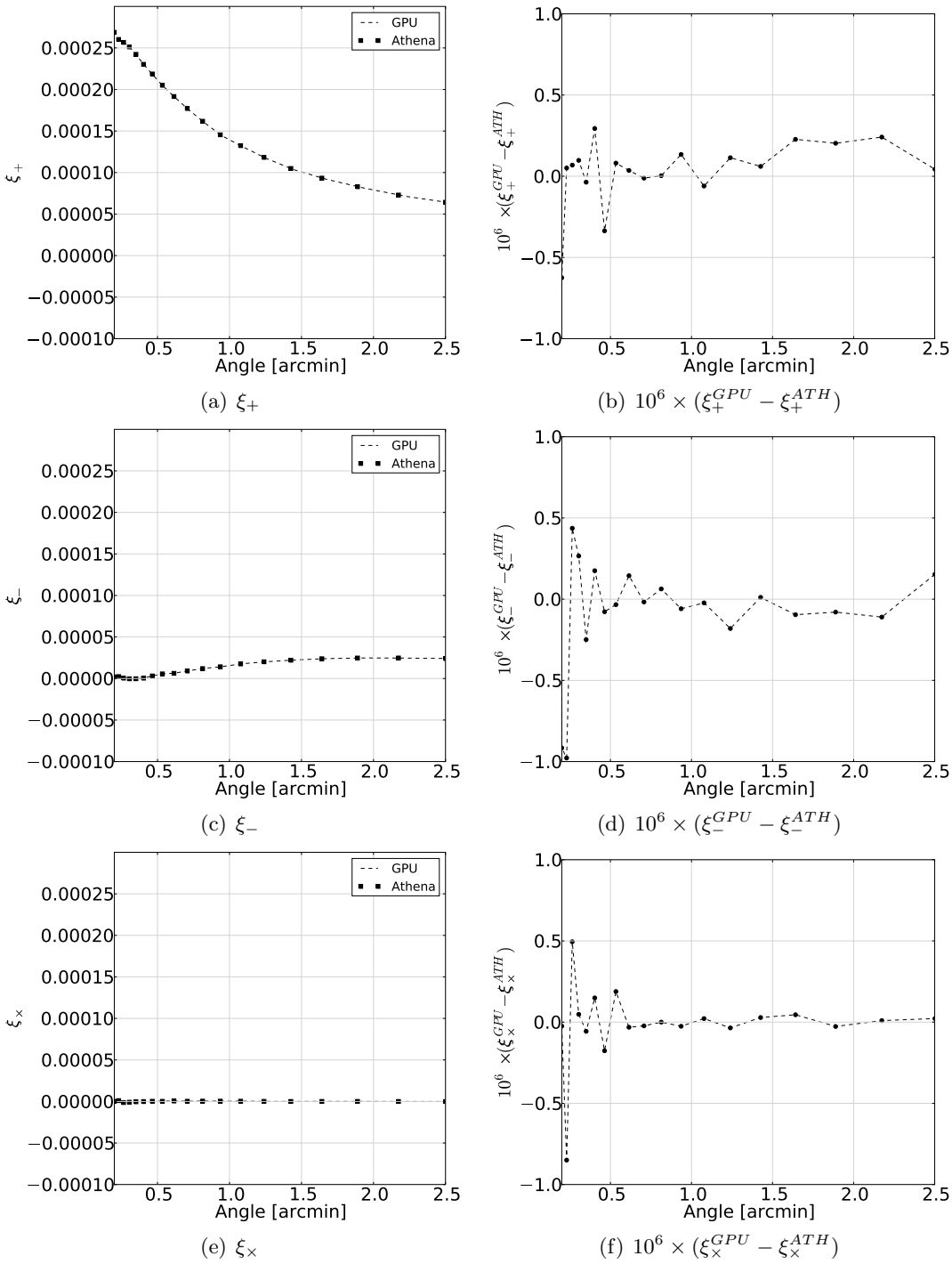
6.4.3 Comparison with Athena Input Reference

For comparison purposes, ξ_+, ξ_- and $\xi_\times$ obtained with the GPU implementation and *ATHENA* version 1.54 with OA = 0.02 are plotted in Fig. 6.4. Despite the slightly different binning schemes the results are in excellent agreement proving that the GPU-based code presented here is completely compatible with a standard analysis code used in cosmology. Unfortunately the sample *ATHENA* input file used as reference has only 40,546 galaxies. In order to obtain more realistic execution times a test with 1 million galaxies with real data is detailed in the next section.

6.4.4 Comparison with 1 Million Galaxies Input Reference

As 1 million galaxies input file, data from the Canada-France-Hawaii Lensing Survey [41] are used, hereafter referred to as CFHTLenS. The CFHTLenS survey analysis combined weak lensing data processing with THELI [28], shear measurement with lensfit [65] and photometric redshift measurement with PSF-matched photometry [42]. A full systematic error analysis of the shear measurements in combination with the photometric redshifts is presented in [41], with additional error analyses of the photometric redshift measurements presented in [7].

[1] This technique is termed *increment of data locality*. Data frequently used are stored locally to the thread.



(a) ξ_+ (b) $10^6 \times (\xi_+^{GPU} - \xi_+^{ATH})$

(c) ξ_- (d) $10^6 \times (\xi_-^{GPU} - \xi_-^{ATH})$

(e) $\xi_\times$ (f) $10^6 \times (\xi_\times^{GPU} - \xi_\times^{ATH})$

Figure 6.4: Comparison of the results obtained with the GPU implementation and *ATHENA* v1.54 OA=0.02 for the *ATHENA* input reference (40,546 galaxies): a) ξ_+, c) ξ_- and e) $\xi_\times$; and, b) $10^6 \times (\xi_+^{GPU} - \xi_+^{ATH})$, d) $10^6 \times (\xi_-^{GPU} - \xi_-^{ATH})$, and f) $10^6 \times (\xi_\times^{GPU} - \xi_\times^{ATH})$.

A query was done on the CFHTLenS catalogue query page[1] for right ascension, declination, the ellipticities (as proxies of the shear) and the weight without any selection cuts, except the requirement that the measured ellipticities are non-zero. The purpose here is to run the code on a catalogue with ellipticities even if they are not accurate or contains contamination from non-galaxy components. An area was selected from one of the four fields to hold exactly one million galaxies (randomly selected).

On the resulting catalogue, the GPU code is executed as well as *ATHENA* with varying opening angles to compare precision and execution time performance. For this purpose, the binning is tuned to obtain values for ξ_+, ξ_-, $\xi_\times$ at the exact same angle separation values θ. The results of execution times are shown in Fig. 6.5.

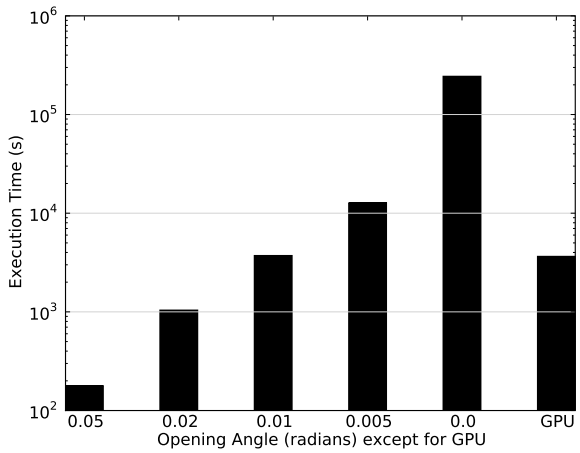


Figure 6.5: Mean execution time (s) for *ATHENA* code for various opening angles (radians) and GPU code for 1 million galaxies input reference (CFHTLenS). The execution time of the GPU code is roughly equivalent to the *ATHENA* code for an opening angle of 0.01 radians. For the same precision ('brute-force') the GPU implementation is a factor 68 faster than a CPU-based code such as *ATHENA*.

The GPU implementation takes $3,650.0 \pm 1.4$s to analyse this catalogue. It should be noted that execution times are tightly bound to the number of bins in the histogram. Variations in this will produce a different amount of sequential updates on the values stored in the bins, and consequently of the execution time. The GPU implementation speed is comparable to *ATHENA* when using an opening angle 0.01 radians for this dataset: $3,723.3 \pm 8.4$s, (see Figure 6.5).

When the opening angle approaches zero radians, the code makes fewer approximations and becomes equivalent to a brute force method. Unfortunately the reduction of

[1] http://www.cadc-ccda.hia-iha.nrc-cnrc.gc.ca/community/CFHTLens/query.html

the opening angle leads to a critical increment in the execution time. For OA = 0.005 radians the execution time, $12,688.7 \pm 122.7$s, this is 3.5 times slower than the GPU processing time. Finally for OA equal to zero the execution time increases to $247,681$s this is a factor 68 slower than the GPU code execution time.

Concerning the precision, Figure 6.6 shows how using an OA with an execution time equivalent to the GPU code (OA = 0.01) can induce large errors (a few percent in relative terms). The exact required precision will differ from survey to survey and is still a topic of debate in the cosmological community. On the other hand, the GPU code shows differences smaller than 0.001% with respect to the (much slower) OA=0.0 execution. It is worthwhile noting that for larger datasets in terms of angular range the required OAs to reach the same overall precision will be smaller thus pushing the case for a fast brute-force implementation to tackle future shear surveys.

6.4.5 Code Optimization

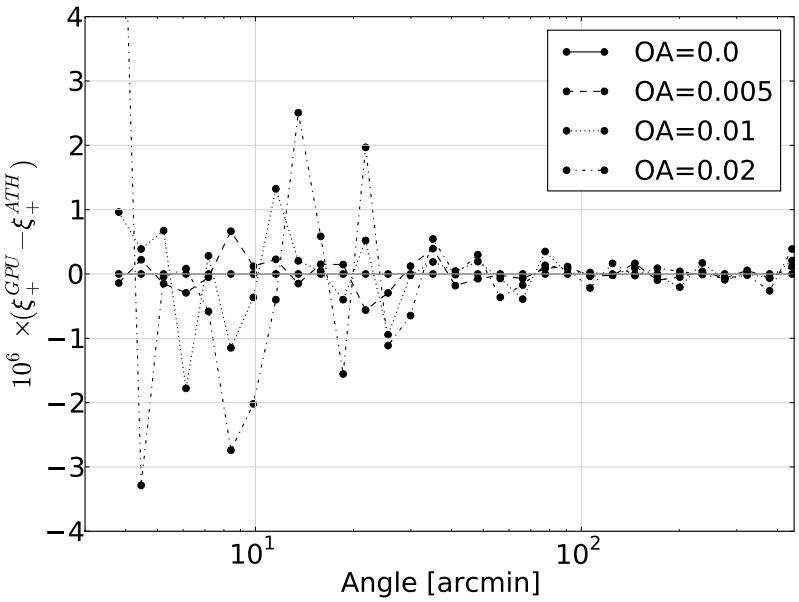
Once the performance has reached a satisfactory level and considering that most of the additional code modifications degrade the performance, a second phase of optimization based on the compiler options is performed.

The most successful test corresponds to the modification of the L1 configuration. The Fermi architecture of the GPU distributes 64 KB between shared memory and L1 cache memory. Three configurations are possible: the default configuration with 48KB of shared memory and 16 KB of L1 cache memory, a second configuration with 16KB of shared memory and 48 KB of L1 cache memory and finally it is possible to turn off L1 cache memory.

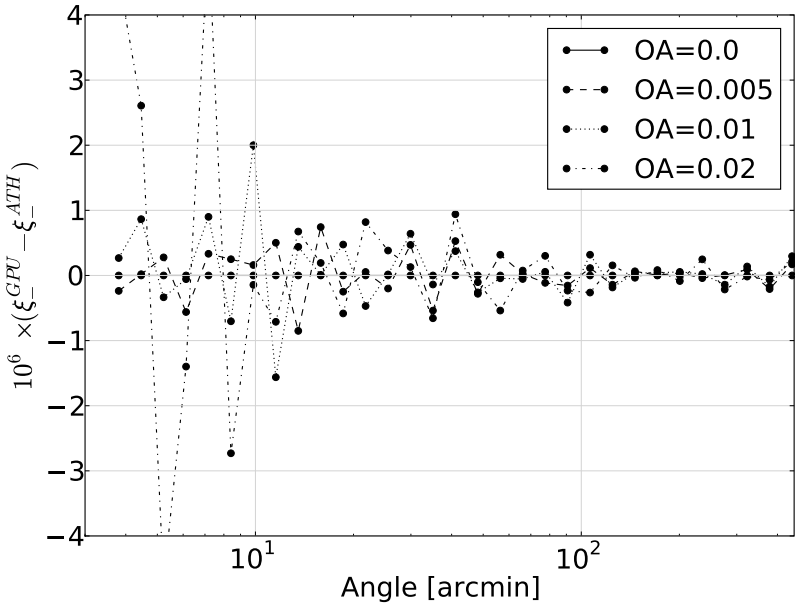
Both L1 and L2 cache are queried during the memory location process. First of all L1 is queried and only when memory location is not found L2 is queried. Finally if memory location is not found in any of the two caches then main memory is accessed. Through main memory accessing L1 and L2 caches are populated with memory addresses which might avoid future main memory accesses.

The configuration implementing 48KB as cache memory and 16 KB as shared memory is recommended when intense data reuse exists or one has a misaligned, unpredictable or irregular memory access pattern. If applications need to share data among the threads of the thread-block, 48KB as shared memory and 16KB as cache memory is recommended. If the kernel has a simple enough memory access pattern the explicit caching of global memory into shared memory through L1 turn-off may increment the performance of the code.

As the baseline code implements the L1 default configuration the two other options are tested in order to check potential reductions in the execution time. The numerical



(a) Deviations GPU-*ATHENA* of computed ξ_+.



(b) Deviations GPU-*ATHENA* of computed ξ_-.

Figure 6.6: Deviations of the GPU code results with respect to *ATHENA* results at different opening angle settings, for the computation of ξ_+ and ξ_-. As the OA becomes smaller, less approximations are made by the *ATHENA* implementation, and the result converges to the GPU computed values (to levels below 0.001%).

Table 6.7: Mean execution time (s) for original code and when applying L1 optimization.

Baseline Code	3,650.0±1.4
Base Code + *L1 turn off*	3,618.7±0.6
Reduction	**31.3**
Speedup	**1,009 (0.9%)**

experiments demonstrate that only the configuration with L1 turned-off improves the efficiency of code. By turning off L1 cache the achieved speedup is 1.009, which is equivalent to 0.9% of improvement (Table 6.7).

The statistical analysis of the execution times (Table 6.7) is performed using the Wilcoxon signed-rank test. The results of this test ($p-value = 8 \cdot 10^{-5}$), indicates that the differences are statistically significant for a confidence level of 95% (p-value under 0.05).

6.4.6 Heterogeneous Computing

The previous section optimization focused on the reduction of the execution time by modifying the code and the compilation options. The baseline code and the later L1 memory optimization has produced a competitive implementation where a high-accuracy and an affordable execution time are achieved.

However, parts of the computational resources are underused because during the kernel execution the CPU stays idle. In order to balance the computational load between GPU and CPU a concurrent computing scenario is proposed. Concurrent computing allows distributing tasks between the CPU and GPU. The tasks involved should not have dependencies between them. Concurrency is applied to the shear correlation calculation by dividing the input data into two chunks which are assigned to CPU and to GPU respectively. In the CPU part a parallel implementation based on OpenMP is applied. Due to the fact that the optimal scenario is when both executions take the same execution time, the choice of the chunk size is critical. Initially diverse chunk sizes are tested in order to select the most appropriate one for the reduction of the execution time (Fig. 6.7).

This naive trial-and-error method shows that the optimal chunk size is to compute 11% of the galaxies on CPU and the rest on GPU on our test setup. This may differ for different machines. For lower percentages the CPU-part analysis finishes faster than GPU-part. As a larger volume of data is supplied to CPU-part the execution time diminishes progressively up to where the minimum is reached. When a larger

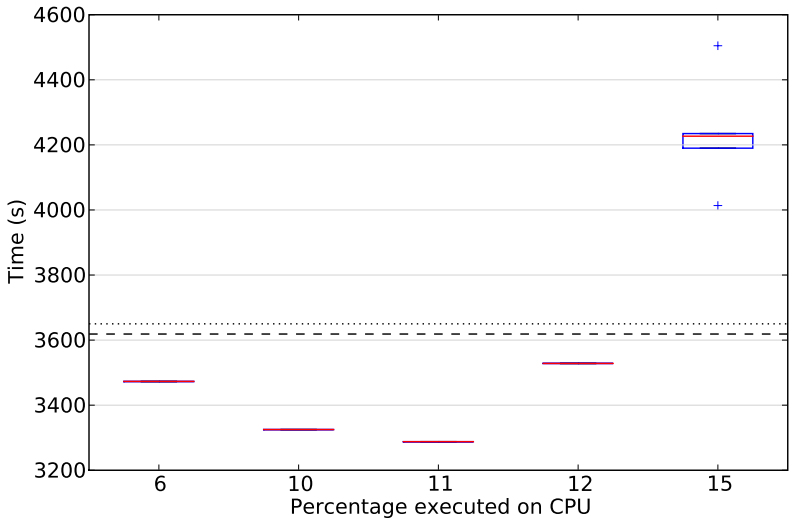


Figure 6.7: Execution time (s) for diverse CPU-processed percentages in the concurrent computing model. The dotted line is the reference to the baseline code execution time, whereas the dashed line is the execution time after L1 memory optimization.

Table 6.8: Mean execution time (s) for original code and for the previous improvements plus when implementing concurrent computing.

Baseline Code	3,650.0±1.4
Base Code + L1 turn off + *Concurrent Computing*	3,287.80±0.03
Reduction	**362.20**
Speedup	**1.11 (11%)**

than optimal amount of data is supplied for CPU processing the execution time grows significantly.

When balancing CPU and GPU processing, the code achieves an integrated speedup of 1.11, which means a reduction in the processing time of 11% in relation to the baseline execution time (Table 6.8), including the previous code optimization.

The statistical analysis with Kruskal-Wallis test of the successive versions of the code states that the differences in the execution time are significant at a significance level of more than 95%. The Wilcoxon signed-rank test with the Bonferroni correction indicates that the differences between the three sets of execution times (baseline, L1 memory optimization and concurrent implementation) are also significant at a significance level of more than 95%. This result reinforces that the modifications applied produce a net improvement in the application productivity.

Table 6.9: Mean execution time (s) for original code and for the previous improvements plus when implementing the loops reordered.

Baseline Code	3,650.0±1.4
Base Code + L1 turn off + *Reordering loops*	3,243.3±0.9
Reduction	**406.7**
Speedup	**1.13 (13%)**

6.4.7 Further Code Optimization

6.4.7.1 Reordering Loops

Although the OpenMP-CUDA implementation obtains a speedup of 1.11X, it is not fully satisfactory, specially taking into account that the goal is to be able to process very large files. So, before testing other hybridization such as MPI-CUDA, additional efforts in the kernel optimization have to be performed.

First of all, a modification of the sequence of the commands in the kernel is proposed. The kernel goes through the pairs of galaxies by using a double loop. Initially the sequence of the commands is a double loop with the commands *for-while*.

When implementing the double loop *for-while*, the coordinates of the most inner (in the loop) galaxy are loaded N^2 times. However, the alternative double loop *for-while* only makes N loads of the most inner coordinates to go through all the pairs of galaxies. In both cases, the coordinates in the outer loop are loaded once.

By profiling the both versions of the kernel, it can be checked that the number of global memory load requests are reduced to the half. This modification does not alter the values of the profiling such as, occupancy, shared memory consumption, or register consumption.

As a consequence of this modification, the mean execution time is reduced to 3,243.3±0.9 s (Table 6.9).

6.4.7.2 Vectorization

As second test, the vectorization of the loads of the galaxies coordinates and ellipticities is tested. The high computational capacity of the GPU hardware makes that many kernels are bandwidth bound. Independently of the calculation, an efficient management of the bandwidth is mandatory to achieve a high efficiency. In the baseline implementation, this efficient management of the bandwidth was considered pertinent from the initial steps of the porting process.

Table 6.10: Mean execution time (s) for original code and for the previous improvements plus when implementing vectorized loads.

Baseline Code	3,650.0±1.4
Base Code + L1 turn off + Reordering loops + *Vectorization*	3,184.2±0.7
Reduction	**465.8**
Speedup	**1.14 (14%)**

In addition to the previous considerations and in order to improve the performance of the GPU implementation, some of the load and store operations are replaced by vectorized instructions. By using vectorized instructions, the total number of load instructions and the latency are reduced, while improving the bandwidth exploitation. Unfortunately, the mandatory use of double precision limits the use of vectorized instructions. Only the vector type *double2* is available to implement the vectorized loads.

It should be underlined that the use of vectorized loads is not exempt of drawbacks: a higher consumption of register per thread is produced. For this reason, the conversion of some variables from *double* to *double2* has to be done step by step and by measuring if any reduction in the execution time is achieved. In our case, only the load of the coordinates of the galaxy and the ellipticities in the for-loop is worth to be converted from *double* to *double2*. After the vectorization process, the execution time becomes 3,184.2±0.7 s (Table 6.10).

When statistically analysing these execution times (baseline implementation, reordered loops implementation and vectorized implementation) with the non-parametric tests, it can be stated that the differences are significant for a confidence level of 95% (p-value under 0.05).

6.4.8 Hybrid MPI-CUDA Implementation

During a long period, the *ATHENA* code has been a reference of the shear-shear calculation. This implementation has been widely employed due to a successful strategy (kd-trees) to reduce the computation complexity. However, due to the increment expected for the forthcoming years in the volume of data accessible by cosmologists, new efforts in the acceleration of the analysis are mandatory.

After the long track of optimizations, the first objective is that the MPI-part of the code does not penalize the execution time. This can be verified through the execution time of the 1-node MPI-CUDA executions (Table 6.11). As can be appreciated, the 1-node execution time (3,325.39 s) is comparable with the mean execution time of the GPU-implementation (3,184.2 s), being only a 4.4% larger.

Table 6.11: Execution time and speedup (20 executions) for MPI-CUDA implementation and 1 million of galaxies of the shear-shear calculation for diverse number of nodes.

Nodes	Execution Time	Speedup Related to MPI-CUDA 1-node Implementation	Speedup Related to GPU Implementation	Speedup Related to *ATHENA* Code at OA=0.0
1	3325.39±0.66		0.96	73.4
2	1672.59±0.47	1.99	1.90	145.9
4	845.15±0.29	3.93	3.77	288.7
8	432.24±0.19	7.69	7.37	564.6
16	225.49±0.13	14.75	14.12	1082.2

Concerning the behaviour of the MPI-CUDA implementation for larger number of nodes (second column at Table 6.11), the speedup obtained when using 2 and 4 nodes is really close to the theoretical maximum speedup, whereas for 8 nodes the speedup diminishes slightly. Finally, when using 16 nodes the speedup degrades to 14.75. This degradation of the speedup for 16 nodes computation corresponds to the case when the time of data transfer becomes comparable with the time of data processing. However, the final aim of the GPU-CUDA implementation is to process files much larger than 1 million of galaxies. Therefore, a mitigation of this degradation is expected when executing these very large files. In these cases, the time of data transfer will loss relevance in relation to the processing time.

In addition to the comparison of the execution times with the 1-node execution time, other comparisons can be performed against the execution time of the GPU implementation (third column at Table 6.11). Obviously, the speedups obtained in this case are lower than in the previous cases. However, they provide a clue if the effort of porting the application to a hybrid platform, MPI-CUDA, is fruitful. As can be appreciated in the values shown in the third column at Table 6.11, the speedups are similar to those obtained when comparing with 1-node MPI-CUDA implementation. Therefore, it can be concluded that the MPI-CUDA implementation performs in a high level, with a small degradation.

The degradation reported for the largest number of nodes will be partially mitigated when analysing input files larger than 1 million galaxies. As an example, two tests are executed with an input file of 15 millions of galaxies. Although, no observational files with this size exist today, they are expected to reach this size in the forthcoming years.

When executing the analysis of 15 millions of galaxies in the GPU implementation, the execution time takes 169 hours, whereas when executing on the MPI-CUDA

implementation with 16 nodes it takes only 11 hours, achieving a speedup of 15.36. As foreseen, the larger input file mitigates the degradation when executing with large number of nodes. So that, by varying from 1 million to 15 millions of galaxies the speedup for 16 nodes executions grows from 14.12 to 15.36.

This last test is repeated by using 16 nodes with M2050 GPU cards instead of M2075. In this test, the execution time is identical, 11 hours. Both cards, equally perform for this implementation and input file. By mixing both resources: M2050 and M2075 cards, an execution with 32 nodes is also tested. In this case, the MPI-CUDA implementation takes 6 hours to analyse the 15 million of galaxies. For this test, the speedup achieves 28.17.

Finally, the execution times of the 1 million galaxies test can be compared with the execution time of *ATHENA* when using zero opening angle: 247,681 s (last column at Table 6.11). If this execution time is compared with the 16-nodes (225.49 s), an overall speedup of 1082.2 is obtained.

6.5 Improvement in the Precision of Histogram Calculation on GPU

Histogram calculation is an essential part of multitude of scientific analysis. In cosmology, they are frequently employed to study the large-scale structure of Universe through the statistical analysis (correlation function) of large amount of data. Among the most commonly used correlation functions are: the two-point and three-point correlation functions and the shear-shear correlation function. Therefore, the precision on the correct calculation of the counts in each bin is a key element in the final precision of the cosmological variables. In order to accelerate the analysis of large data sets, GPU computing is being widely employed. However, the recommended histogram calculation procedure suffers from lack of precision while analysing large data sets. In this section, an alternative implementation to those presented at Section 6.2.3 for histogram calculation which is able to tackle large data sets with high precision, while maintaining an affordable execution time, is proposed. This implementation is tested against three cosmological analyses by using observational data.

The work presented in this section has been performed in a C2075 card as it is described at section 2.3.3.2.

6.5.1 Weaknesses of Number-Representation

The floating point encoding allows representing numbers in the range from 1.18×10^{-38} to 3.40×10^{38}. However, this representation is done with some approximations. Some

numbers can be exactly represented, for example the number 1; whereas others are represented approximately, for example the number 0.1 or the fraction $\frac{2}{3}$. This incapability to represent exactly some numbers carries on inexactness of the result under certain circumstances, and as a consequence, the accuracy of the analysis can be affected. In [99], a selection of the peculiarities related with the precision of the float representation and their operations on GPU is presented. For a more extensive introduction about floating point representation, the following references can be consulted [38] [45].

Other aspect of floating point representation related with the current work is the termed as *epsilon comparison*, $boolIsEqual = fabs(f1 - f2) <= epsilon$, or in other words, when adding to a value a unit in float-representation and the result is equal to the initial value. The float-representation can not represent correctly all the integers above 16,777,216. If a unit is added to this number, the result is the same number. This is because the float-representation lacks of the accuracy to correctly represent all the integers in the range from 16,777,216 to 3.40×10^{38}.

When analysing cosmological data with correlation functions, the procedure is roughly as follows. After calculating the separation (linear or angular) between the galaxies, if the distance or the angle is in the range of the histogram, then a value is added to the corresponding bin. For 2PACF and 3PACF, the value is the unit, whereas for the shear-shear correlation this value depends on the shapes and the orientation of the galaxies, being a non-integer number.

Therefore, as a side effect of the lack of accuracy for representing some integers larger than 16,777,216, if a bin reaches this number of counts, then the additional coincidences of two galaxies for the angle or the distance corresponding to this bin will be lost. Thus, for the case of adding counts unit by unit in the bins, the number of counts will never grow above 16,777,216. And, consequently, the accuracy of the cosmological calculation will be affected. The capacity to correctly gather the number of counts assigned to a bin is termed *bin containment*.

The use of a number-representation based on integer will theoretically solve the lack of accuracy of float-representation. For example, integer representation is able to reproduce numbers up to $2.1 \cdot 10^9$, and unsigned-integer representation up to $4.3 \cdot 10^9$. However, even these limits are inadequate for cosmological studies when using large data sets[1]. On the other hand, it will be also demonstrated that unsigned-long-long-integer representation fulfils the requirements of accuracy, but it severely penalizes the performance.

[1]When analysing 1 million galaxies with the 2PACF or the shear-shear correlation codes, then the most populated bins reach the order of 10^{10} counts. For the 3PACF with a input of 10^4 galaxies, the most populated bins have a similar number of counts.

In the following section, these scenario are in-detail studied, and an alternative approach is proposed and evaluated.

6.5.2 Results and Analysis

6.5.2.1 Float-based Implementation

This first implementation follows the recommendation guidelines of the [76] and the procedure recommended at the book [85]. In this case, the implementation is based on float precision, and using shared memory for storing the sub-histograms and global memory for gathering the final histogram. This implementation has been widely used for cosmological analyses, for example, for the 2PACF [13, 17, 18, 77]; 3PACF and shear calculation [14].

The limit of the largest attainable number in float precision and the lack of precision will be more important in the 3PACF than in the 2PACF. This stems from the fact that if N galaxies are going to contribute to a single bin, then for the 3PACF this bin will accumulate N^3 counts; whereas for the 2PACF only N^2 counts. For this reason, the 3PACF is used in this section to stress the float-based implementation and the following ones.

During the execution, the threads go through all the triplets of galaxies. For each triplet, the bin where a count has to be added is calculated, and if the bin is in the range of the histogram, the addition is performed. This addition is performed over the sub-histograms held in *shared memory*, and for this operation, the *atomicAdd()* function is used. Since the kernel is multithreaded, simultaneous updates of the same bin might occur, so that the use of atomic functions is mandatory. Finally, all the sub-histograms are gathered over the final histogram on *global memory* by using again the *atomicAdd()* function. Presently, the *atomicAdd()* function can be implemented in float, integer, unsigned-integer and unsigned-long-long-integer precision.

For the float-based implementation, the lack of sensitiveness is the most relevant problem, because previously to reach the limit of the largest number in float, the mentioned addition through the *atomicAdd()* function will become not effective. When adding one unit to a bin where a large number of counts are, at least 16,777,216 counts, the final result will not show the correct result of the addition.

For illustrating this effect, some artificial input files have been created and processed with the 3PACF and the float-based implementation. In these files, all the galaxies (points) have the same coordinates, and therefore they will contribute to a single bin (the zero bin). The numerical results (Table 6.12) of this test show that float-based implementation handles correctly up to $8 \cdot 10^9$ counts per bin, which corresponds to

Table 6.12: Float-based implementation for the 3PACF: execution time and precision.

Input Size	Counts per bin	Correct result?	Kernel execution time
1k	10^9	Yes	2,861 ms
2k	$8 \cdot 10^9$	Yes	22,803 ms
3k	$2.70491 \cdot 10^{10}$	No	n/a
4k	$6.37629 \cdot 10^{10}$	No	n/a

an input size of $2 \cdot 10^3$ galaxies. Otherwise, when the input size is $4 \cdot 10^3$, the final result shows a deficit of counts[1]. This indicates the incapacity of this implementation to accurately contain $6.4 \cdot 10^{10}$ counts per bin.

For comparison purpose, when analysing observational data, for example the 1 million galaxies input file from CFHTLenS, the most populated bins reach around $1.2 \cdot 10^{10}$ counts. Therefore, the limitations presented in this section are very pertinent for obtaining accurate results.

Special attention has to be paid to the case of 3k input size (Table 6.12). In this case, no deficit of counts appears, but oppositely it is overestimated. This overestimation stems from the limited precision for representing some numbers when using float precision. When iteratively adding numbers, which during the representation process have been rounded up, the final result is slightly higher than expected[2].

In order to demonstrate this effect, a new execution is performed with 3k galaxies but corresponding to more than one single bin. Instead of having 3k identical single positions, 2k galaxies have the same coordinates and the remaining 1k, other different coordinates, but identical among them. When forming triplets between points selected from both subsets, then the bins corresponding to 1-degree angle of separation are fed. Otherwise, when the points are selected inside of each subset, only the zero degree bin is incremented. In this way, the overall $2.7 \cdot 10^{10}$ counts are distributed among few bins; oppositely to the scenario when all positions are identical, in which all the counts feed a single bin, the zero bin. When using as input data this 2k+1k file, the overall number of counts is correctly accumulated.

From the previous results, it can be concluded that the use of float representation as base of the histogram construction can produce two different types of errors: deficit or overestimation of the number of counts per bin. As a consequence, this implementation can induce errors in cosmological calculations.

[1] In the case where the implementation produces an incorrect number of counts per bin, the execution time is not shown, and as an alternative the string *n/a* is presented.

[2] The typical academic example of this underlying inexactness is the calculation of the number one by iteratively adding ten times the value 0.1 in float precision. The final result obtained is 1.0000001.

Table 6.13: Integer-based implementation: execution time and precision.

Input Size	Counts per bin	Correct result?	Kernel execution time
1k	10^9	Yes	3,319 ms
2k	-589934592	No	n/a

Table 6.14: Unsigned-integer-based implementation: execution time and precision.

Input Size	Counts per bin	Correct result?	Kernel execution time
1k	10^9	Yes	3,318 ms
2k	3705032704	No	n/a

At Table 6.12 and following ones for other implementations along this work, the execution times[1] of the kernel for diverse input sizes are presented, except for the cases where the implementation fails to calculate the correct result. Besides, the number of counts per bin is also presented.

In order to overcome the problems with the precision of the calculations, diverse modifications of the precision are tested. In the following, the other available number-representations for the *atomicAdd()* function are implemented and the results analysed.

6.5.2.2 Integer-based Implementation

In this second strategy, the previous case is modified by replacing float precision by integer precision. Unfortunately, this modification aggravates the final result (Table 6.13). Only the input with 1k points is correctly processed. Larger number of counts per bin is incorrectly processed due to the limit of the largest value that can be represented in integer precision.

6.5.2.3 Unsigned-Integer-based Implementation

By replacing integer precision by unsigned-integer precision, the problem is not solved (Table 6.14). Still the largest input file correctly processed is 1k galaxies. This new implementation does not handle correctly cases where $8 \cdot 10^9$ counts per bin ought to be accumulated.

[1] For measuring the kernel execution time, the CUDA API event has been used. This mechanism is more precise than CPU or operating system timers because it eliminates the sources of latency associated to the operating system.

Table 6.15: Unsigned-long-long-integer-based implementation: execution time and precision.

Input Size	Counts per bin	Correct result?	Kernel execution time
1k	10^9	Yes	9,778 ms
2k	$8 \cdot 10^9$	Yes	78,161 ms
3k	$27 \cdot 10^9$	Yes	263,690 ms
4k	$64 \cdot 10^9$	Yes	624,983 ms
5k	$125 \cdot 10^9$	Yes	1,220,516 ms
10k	10^{12}	Yes	9,762,523 ms

Table 6.16: Profile of the implementations.

Implementation	Static Shared Memory	Registers	Occupancy
Float	2.05KB	25	67%
Integer	2.05KB	28	67%
Unsigned integer	2.05KB	28	67%
Unsigned long long integer	4.10KB	28	67%

6.5.2.4 Unsigned-Long-Long-Integer-based Implementation

In order to improve the precision of the kernel calculating the histogram, in the previous implementation the unsigned-integer precision is replaced by unsigned-long-long-integer precision. The results of this implementation demonstrate that it calculates correctly up to 10^{12} counts per bin (Table 6.15), which corresponds to an input size of 10^4 points. As can be appreciated at Table 6.15 and taking into account the volume of observational data nowadays accessible, this implementation fulfils the precision requirements for cosmological calculations.

Unfortunately, the execution time takes much longer than for the previous implementations[1]. For example, for processing 10^3 points input (10^9 counts per bin), the unsigned-long-long-integer implementation takes more than three times (3.4) that the float-based implementation; whereas for $2 \cdot 10^3$ points input a similar ratio between both execution times is obtained.

In-depth analysis of the kernel (Table 6.16) unveils that the three integer-based implementations consume more registers than float-based implementation. On the other hand, the unsigned-long-long-integer implementation consumes more static shared mem-

[1]In order to fairly compare, the execution times correspond to the kernel execution, and they do not include any disturbing factors as the data copy between the CPU and GPU or the conversion of data from right ascension and declination to Euclidean coordinates.

ory than the remaining ones. In any case, neither of these variation modify the occupancy.

Since a trade-off between the precision and the performance of the implementation is the final goal, this implementation can not be considered for analysing large input files due to its large execution time.

6.5.2.5 Float-based Alternative Implementation

In order to overcome the difficulties associated with the lack of precision and the large execution time, a new implementation is proposed. Similarly to the previous implementations, the proposed implementation creates the sub-histograms on shared memory with float precision, but in contrast to the former ones, it does not accumulate the sub-histograms into the final histogram on global memory. In this new approach, the gathering of the sub-histograms into the final histogram is done in the CPU memory (RAM). This modification allows implementing the final histogram in double precision, whereas until now the previous implementations are limited to use single precision.

In the previous implementations, firstly the counts are added to the sub-histograms on shared memory by using the *atomicAdd()* function. Later the sub-histograms on shared memory are gathered in the final histogram on global memory by using, once again, *atomicAdd()* function. Currently the *atomicAdd()* function is limited to accumulate values with precision of integer, unsigned-integer, unsigned-long-long-integer, and float; but unfortunately it does not support double precision.

In the proposed implementation, the initial addition of the counts through *atomicAdd()* function to sub-histograms on shared memory is maintained. However, later the sub-histograms are not gathered on global memory. Contrary, all the sub-histograms are copied to global memory in such way that they are sequentially arranged. For this purpose, on global memory an array is created to hold all the sub-histograms sequentially ordered. In this way, all the sub-histograms are sent to the CPU memory where they are gathered in the final histogram. The key point of this new implementation is that this final histogram on CPU is defined with double precision, and therefore the precision is enhanced.

As a consequence, in the new algorithm the array holding the histograms on global memory does not have the size of the histogram, but the equivalent size to all the sub-histograms aligned sequentially.

This alternative does not avoid some penalties. Among others, the new array for holding all the sub-histograms increases the global memory consumption by a factor equal to the number of the thread blocks used during the kernel invocation. Moreover, this larger array takes longer during the copy of data between the CPU memory and the

Table 6.17: New algorithm implementation: execution time and precision.

Input Size	Counts	Correct result?	Execution time
1k	10^9	Yes	2,873 ms
2k	$8 \cdot 10^9$	Yes	22,701 ms
4k	$64 \cdot 10^9$	Yes	181,562 ms
5k	$125 \cdot 10^9$	Yes	353,717 ms
10k	10^{12}	Yes	2,819,102 ms

global memory. Finally, the final gathering in this new implementation is performed in a sequential manner on CPU, instead of a parallel way on GPU. All these considerations will slows down the execution.

To fairly compare, the execution time in this implementation is measured including the kernel execution and the gathering of the sub-histograms into the final histogram on the CPU; whereas for the previous cases, the execution time corresponds only to the kernel execution.

The numerical results (Table 6.17) demonstrate that the proposed algorithm is able to process correctly at least up to 10^{12} counts per bin[1], equally to unsigned-long-long-integer implementation, but with a reduction of the execution time by a factor of 3.4. When comparing with the float-based implementation, a similar execution time is produced. Therefore, the proposed implementation exhibits a capacity to deal with as large input size as unsigned-long-long-integer implementation, at the same time that it maintains a similar execution time as the fastest implementation, the float-based implementation.

It should be underlined that the precision of the proposed implementation depends on the parameters used during the kernel invocation. In all the cases, the number of threads per block has been considered only depending on the histogram structure and the current limitation of the hardware. The limitation of the number of threads per block on Fermi architecture (1024) forces to establish for the 3PACF a thread block structure of $8 \times 8 \times 8$ threads, 8 bins per each one of the three angles subtended by each triplet of points. So that, the number of thread blocks becomes relevant for the precision of the implementation[2]. More thread blocks mean that the counts will be distributed among a larger number of sub-histograms on shared memory, and therefore, less likely to reach the value 16,777,216 in any bin. If the number of thread blocks is reduced, then the problem associated with the lack of precision in float representation

[1] This number of counts per bin is high enough for the analysis of the current observational data.

[2] Further details of the Fermi architecture and its relation with the histogram construction can be found in [14]. This work focusses on the shear-shear analysis but many considerations can be extrapolated to other correlation functions.

Table 6.18: Precision of the results for diverse number of thread blocks and two large input sizes: 5k and 10k galaxies.

Thread blocks	Counts	Correct result?	Execution time
5k galaxies			
84	$115.956 \cdot 10^9$	No	n/a
98	$125 \cdot 10^9$	Yes	353,717 ms
10k galaxies			
210	$0.74 \cdot 10^{12}$	No	n/a
280	10^{12}	Yes	2,819,102 ms

emerges. Oppositely, if the kernel is invoked with a larger number of thread blocks, then this problem is mitigated. As an example, at Table 6.18, the number of thread blocks[1] required for correctly processing very large input files (5k and 10k galaxies) is presented.

The profiling of the kernel which implements the new algorithm does not differ from the values presented at Table 6.16. The register consumption is the same as in the float-based implementation: 25, as well as the static shared memory consumption: 2.05KB. The most significant difference emerges in the time that the new histogram-array takes to be copied to and from the *global memory*. In the previous implementations, it takes less than 2μs, whereas in the new algorithm takes much more. This transfer time depends on the number of thread blocks with which the kernel is invoked. For 280 thread blocks, the most unfavourable case, the copy to or from the global memory takes 47 ms. In despite of this penalization, the time for the two copies is tiny compared with the kernel execution time, 2,819,102 ms, having a very limited impact on the overall performance.

6.5.3 Real Cases

Until now the new algorithm has been tested with a real cosmological problem (3PACF) but by using artificial input files to fully control the tests. In order to complete the analysis, a new study is performed with observational data coming from the CFHTLenS survey [41]. For these tests, a set of 10^6 galaxies has been extracted from the survey.

In the previous sections, it has been proved that the new algorithm correctly reproduces at least up to 10^{12} counts per bin. In the next studies, it will be checked the differences between the new algorithm and the float-based implementation. Addition-

[1]For performance reasons, the number of thread blocks has been chosen as a factor of the number of streaming processors (14) on Fermi architecture.

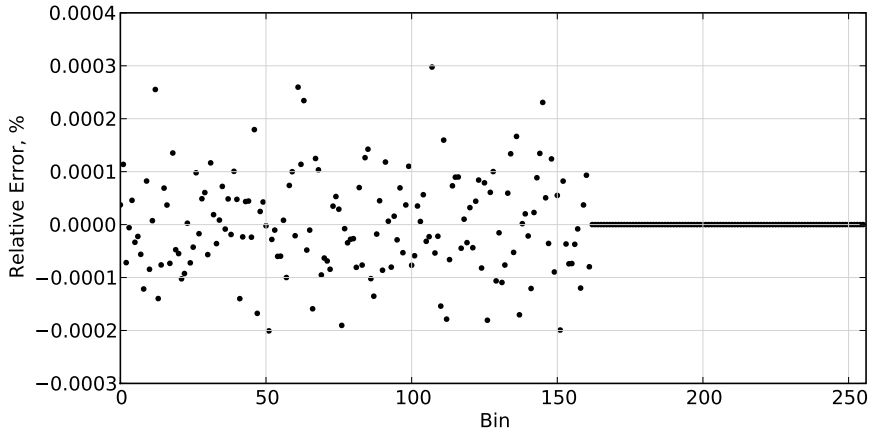


Figure 6.8: Relative error, $100 \cdot \frac{DD_{float-based} - DD_{new\ algorithm}}{DD_{new\ algorithm}}$ for the 2PACF between the float-based implementation and the new algorithm for the CFHTLenS input file (10^6 galaxies). The histogram is composed of 256 bins: 16 degrees with 16 bins per degree.

ally to the 3PACF, two other correlation functions: 2PACF and shear-shear correlation are also employed in the study.

It is expected that the new data set will distribute the counts along all the bins of the histogram. Due to this distribution, the observed deficit of counts in the bins in the previous studies will be partially mitigated. However, it can not be stemmed from the data set which bins will approach the float sensitive limit, and therefore, in which the deficit will appear.

6.5.3.1 Two-Point Angular Correlation Function

Considering that the new algorithm has demonstrated to correctly reproduce the number of counts per bin up to very large figures; then any deviation from these values will be attributed to the inexactness occurred in the float-based implementation. In order to underline this deviation, the relative error, $100 \cdot \frac{DD_{float-based} - DD_{new\ algorithm}}{DD_{new\ algorithm}}$, for the 2PACF is shown at Fig. 6.8.

As can be appreciated at Fig. 6.8, some differences appear between the number of counts calculated for both implementations[1]. In spite of the no-null values of the relative error, presently they are far from to be significant for the current cosmological calculations.

The float-based implementation reproduces acceptably well the number of counts per bin (Fig. 6.8), being the relative error below the threshold of the instrumental

[1]In this case, the histogram covers up to 16 degrees with 16 bins per degree. Due to the reduced area in the sky covered by the survey CFHTLenS, counts above the 160th bin does not exist.

errors. The relative error is low enough to validate the previous works performed with the float-based implementation [13, 18]. However, since the new algorithm does not increase the execution time, while improving the bin containment, it is recommended for the analysis of the 2PACF.

A collateral effect of the float-based implementation is that the repetition of the executions produces variations of one unit in the last significant digit of the counts accumulated in the bins. Again the origin of this effect is the float-representation. When gathering the counts accumulated in the sub-histograms, the order of the addition becomes relevant[1]. Depending on the order in which the *atomicAdd()* operations are executed, a difference of one unit in the last significant digit might occur. If the bins with large number of counts are firstly added, then when adding the bins with tiny number of counts, the lack of accuracy of the float-representation will emerge. However, if the bins with the tiny number of counts are firstly planned, then the float-sensitiveness will be enough to incorporate these values.

This effect is completely corrected in the new algorithm. At Fig. 6.9, the standard deviation for each bin after 10 executions is shown for both implementations. As can be appreciated, when using float-based implementation the values of the counts accumulated per bin have a no-null dispersion (Fig. 6.9(a)). On the contrary, no dispersion is drawn for the new algorithm (Fig. 6.9(b)).

6.5.3.2 Three-Point Angular Correlation Function

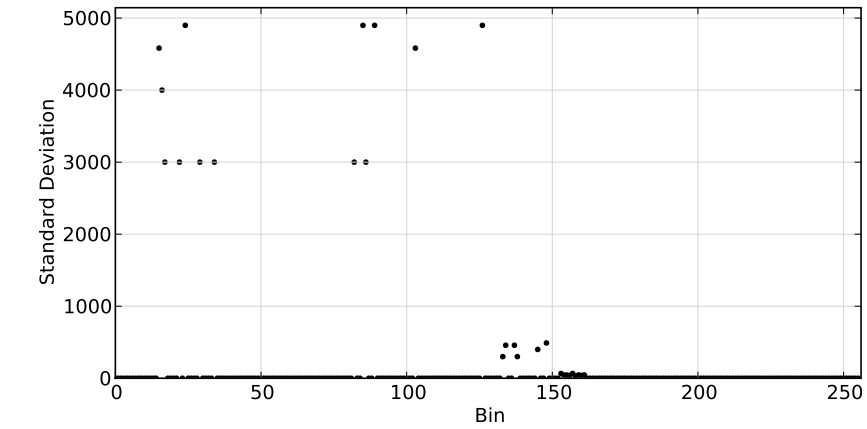
Similar tests to those performed in the previous section are executed, but instead of the 2PACF, by using the 3PACF. This second function increases the stress over the weaknesses of the float-based implementation, since more counts are accumulated per galaxy in the input file[2]. Due to the large execution time of the 3PACF in relation to the 2PACF, for this study a randomly-selected sample of 10^4 galaxies has been extracted from the CFHTLenS file.

In the case of the 3PACF (Fig. 6.10), the relative error for CFHTLenS data set is larger, one order of magnitude, than for the 2PACF, even if a smaller input file is employed. This increment stems from the higher computational intensity of the 3PACF in relation to the 2PACF.

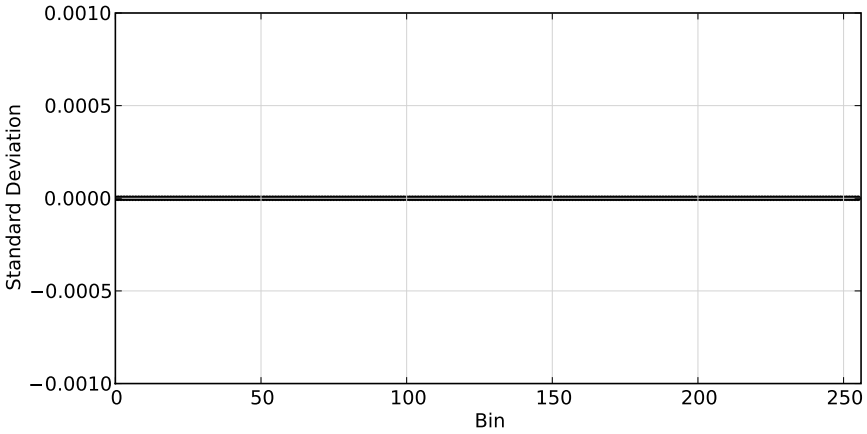
Alike to the 2PACF, for the 3PACF the dispersion of the counts accumulated per bin disappears when the new algorithm is used, instead of the float-based implementation

[1] In general, when using float-representation is advised to sum the numbers in order, being the smaller ones the first.

[2] If N identical points are processed with the 2PACF, then N^2 counts are accumulated in the bin zero. On the other hand, if this sample is processed with the 3PACF, then N^3 counts are expected.



(a) Float-based implementation



(b) New algorithm

Figure 6.9: Standard deviation for the 2PACF for 10 runs of the float-based implementation a) and the new algorithm b). The new algorithm reproduces always an identical result, therefore its standard deviation is null; whereas in the float-based implementation the last significant digit varies among the executions.

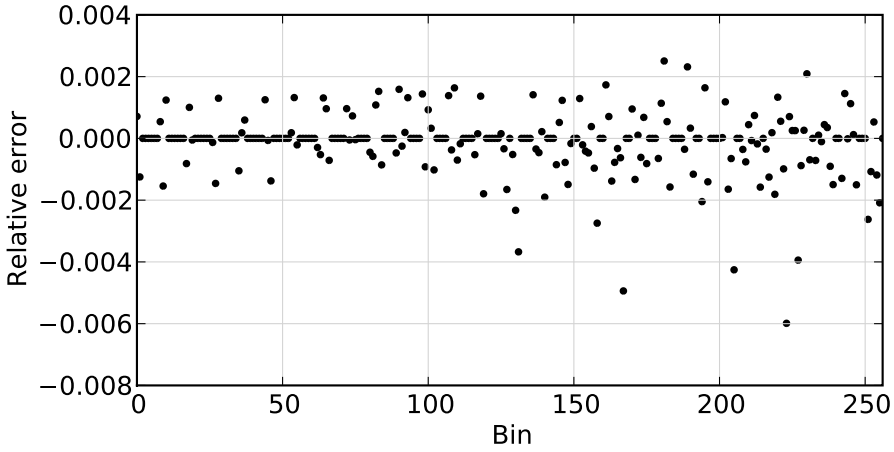


Figure 6.10: Relative error, $100 \cdot \frac{DD_{float-based} - DD_{new\ algorithm}}{DD_{new\ algorithm}}$ for the 3PACF between the float-based implementation and the new algorithm for a sub-set of the CFHTLenS input file (10^4 galaxies in this test). The histogram is composed of 256 bins: 1 bin per degree, with up to the 8th degree per angle in the triplets.

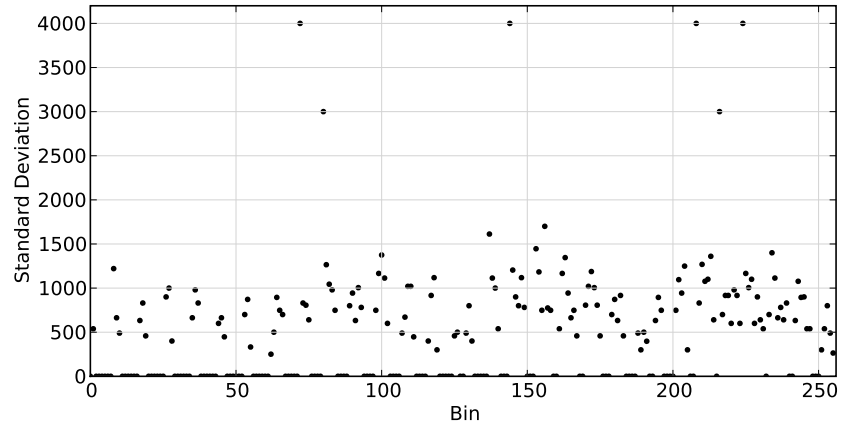
(Fig. 6.11). When using the float-based implementation (Fig. 6.11(a)), the counts per bin exhibit a clear dispersion of the values; whereas, this dispersion is not present when the new algorithm is employed (Fig. 6.11(b)).

The dispersion of results, stemmed from the inexactness of the float representation, disappears when using the proposed implementation. Both 2PACF and 3PACF produce more accurate results when using this new implementation, at the same time that it maintains a limited execution time.

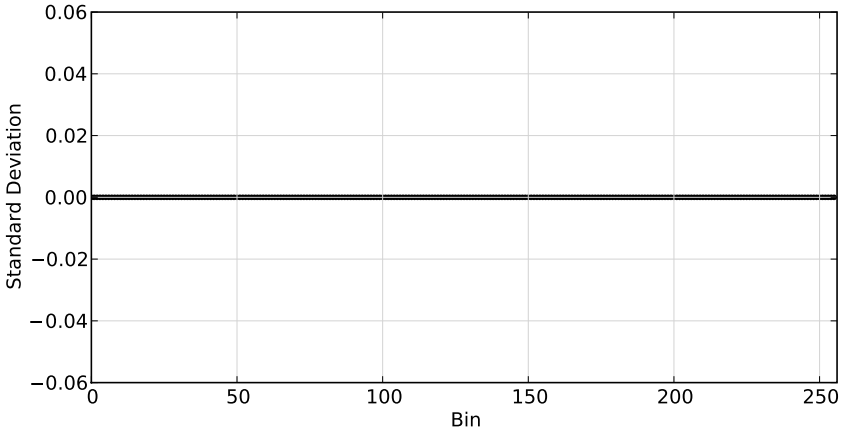
6.5.3.3 Shear-Shear Correlation Function

As a final analysis, the shear-shear correlation function is also studied by using both implementations: the float-based and the new algorithm. Three main quantities or histograms are calculated during the shear-shear analysis: the number of pairs, ξ_+, and ξ_-. Oppositely to the previous studies, where a unit is added to the bin for each coincidence, in the histograms of ξ_+, and ξ_- tiny non-integer values are added to the bins.

As can be appreciated at Fig. 6.12, for ξ_+, and ξ_- the float-based implementation produces different results to the new algorithm. Again the differences stem from the lack of accuracy of the float representation. Besides, similar to the previous studies, the relative error is low enough to still validate the cosmological analysis performed [14]. However, the proposed alternative is able to deal with larger data sets with higher



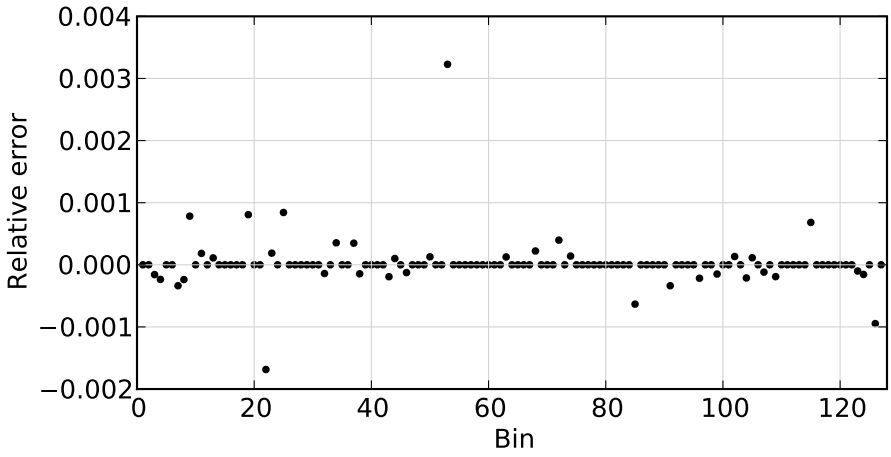
(a) Float-based implementation



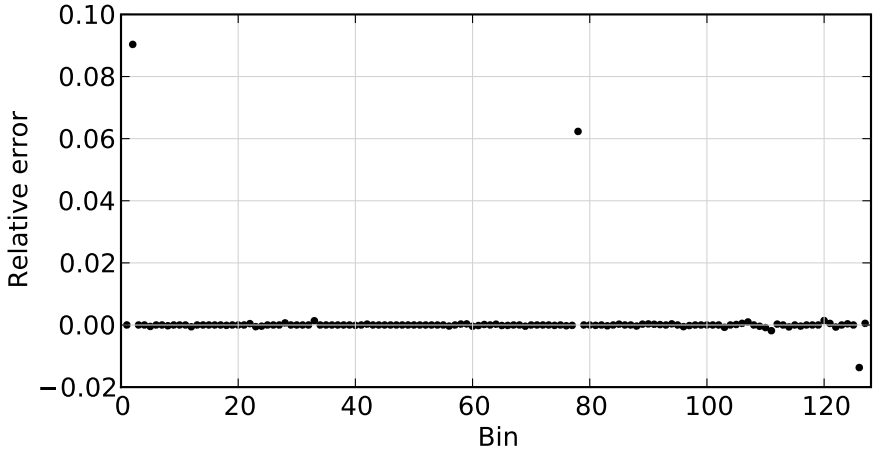
(b) New algorithm

Figure 6.11: Standard deviation for the 3PACF and a sub-set of 10^4 galaxies of the CFHTLenS data set for 10 runs of the float-based implementation a) and the new algorithm b). The new algorithm reproduces always an identical result, therefore its standard deviation is null; whereas in the float-based implementation the last significant digit varies among the executions.

precision by employing similar execution time, and for this reason, it is recommended for future analysis.



(a) ξ_+



(b) ξ_-

Figure 6.12: Relative error, $100 \cdot \frac{\xi_{float-based} - \xi_{new\ algorithm}}{\xi_{new\ algorithm}}$ for the parameters involved in the shear-shear correlation function (ξ_+, ξ_-) between the float-based implementation and the new algorithm for the CFHTLenS input file (10^6 galaxies), for: a) ξ_+, and b) ξ_-.

The study of the ξ_+ (Fig. 6.12(a)) shows a similar order of magnitude in the relative error as the 2PACF. However, the relative error for ξ_- (Fig. 6.12(b)) is much higher. The origin of this higher relative error is that ξ_- is very close to zero, so that the division of the relative error calculation magnifies its value.

Since in the previous studies the absence of dispersion in the final results, when implementing the new algorithm, has been demonstrated, this study has not been

repeated for the shear-shear correlation function.

6.6 Conclusions

The application of the GPU computing to the correlation functions: two-point correlation function and shear-shear correlation function has allowed accelerating these analyses. Faced to other parallel implementations or other approaches such as kd-trees, the GPU implementations for these problems achieve larger speedups.

Simultaneously, the weaknesses associated to the number representation and its impact in the accuracy of the histogram construction in GPU have been studied. The new algorithm proposed for the histogram construction on GPU allows circumventing these weaknesses, without degrading the execution time. Beyond the applicability to cosmological studies, this new algorithm can be applied to other disciplines where the construction of histograms is an essential part of the analysis, such as: image processing, data mining or statistical analysis.

As a consequence of the quality of the implementations, they are being adopted as analysis codes by international collaborations.

Chapter 7

Conclusions

7.1 Conclusions

This Thesis presents diverse studies focused on the improvement of the efficiency of the scientific computation in the areas of astrophysics, astronomy and cosmology. Firstly, exploratory studies have been performed to in-depth analyse particular objectives. These studies have been useful to gain the necessary expertise to evaluate the applicability of these techniques to problems in the scientific areas involved.

Some of the exploratory studies have evaluated and demonstrated how to accelerate the performance of Particle Swarm Optimizer when executing on GPU, the efficiency of the most popular variants of this algorithm, and the improvements when implementing a multipopulation approach for this algorithm.

Other studies have focussed on the impact of the choice of the random number generator on the efficiency of evolutionary algorithms, such as: particle swarm optimizer, differential evolution and genetic algorithm. The conclusions underline the low sensitiveness of the evolutionary algorithms when implementing high-quality random number generators. Furthermore, a technique to reduce the large execution times associated to metaoptimization processes has been also proposed.

Besides, an in-depth study of the most suitable layouts for accelerating the evaluation of non-separable functions on GPU has been performed. This study demonstrates how an appropriate layout reduces the processing time when evaluating this kind of functions on GPU. The lessons learned from these exploratory studies have been later applied to problems in the areas of astrophysics, astronomy and cosmology.

Concerning the searching of high-quality solutions for fitting problems in astrophysics, the applicability of metaheuristics to these problems has been widely demonstrated. The works performed in the rotational curve of the spiral galaxies and the

low-resolution galaxy spectral energy distribution have shown how the application of metaheuristics might improve the quality of the solutions achieved.

Finally, the porting and optimization of cosmological calculations, such as the two-point angular correlation function and the shear-shear correlation function, have allowed a net improvement in the efficiency of these analyses. The new implementations improve the quality of the scientific production by speeding up the analyses, while maintaining the accuracy.

The results obtained have been published in conferences on metaheuristics, parallelism and astrophysics, as well as in computational physics and computing journals. Furthermore, some computational developments are being adopted by international collaborations.

7.2 Future Work

Diverse lines of work are open beyond the results presented in this Thesis. In some cases, they represent further explorations of issues already treated. In these issues, the aim is to check if new refinements improve the efficiency of the solutions proposed. In other cases, they are new developments or new problems where the application of the lessons learned, might suppose an advance in relation to the state-of-the-art.

For instance, other analyses in cosmology might be ported and optimized to parallel infrastructures. The works performed on GPU have been specially successful. As ongoing work, it has been proposed the implementation of a two-point correlation function in three dimensions. A priory, this calculation is burdened with a high-computational charge due to the trigonometric calculations required. Similar improvements to those reached with the shear-shear calculation are expected.

On the other hand, evolutionary algorithms can be applied to the analysis of complex images and data sets in astrophysics, for example those coming from IFUs facilities. The analysis of data in this area faces two main difficulties: the complexity and the high volume. In these cases, the hybridization of evolutionary algorithms and techniques of parallelism is mandatory. International collaborations such as CALIFA or data repositories such as the Sloan Digital Sky Survey provide large data volumes which require the appropriate tools to accurately analyse the data with an affordable execution time budget.

Finally, the exploratory studies presented in this Thesis in evolutionary algorithms and parallelism, as well as in the hybridization of both techniques have inspired other ideas which have not been fully developed. For instance, improvements in the layouts

for accelerating the evaluation of non-separable functions and other complex fitness functions on GPU can be produced by implementing further refinements.

Appendix A

Publications

This section lists the publications obtained during the Thesis period.

A.1 JCR-indexed Journal Articles Arising from this Thesis

1. Cárdenas-Montes, Miguel, Vega-Rodríguez, Miguel A., Bonnett, Christopher, Sevilla-Noarbe, Ignacio, Ponce, Rafael, Sánchez Álvaro, Eusebio, Rodríguez-Vázquez, Juan José: **GPU-Based Shear-Shear Correlation Calculation**, Computer Physics Communications, 185(1):11-18, ISSN: 0010-4655, 2014 (Impact Factor = 3.078, 2012, Quartile Q1)

2. Cárdenas-Montes, Miguel, Rodríguez-Vázquez, Juan José, Vega-Rodríguez, Miguel A., Sevilla-Noarbe, Ignacio, Sánchez Álvaro, Eusebio: **Performance and precision of the histogram calculation on GPU: cosmological analysis as case study**, Computer Physics Communications, 1-27, ISSN: 0010-4655, Accepted (Impact Factor = 3.078, 2012, Quartile Q1)

3. Cárdenas-Montes, Miguel, Vega-Rodríguez, Miguel A.: **Effect of data layout in the evaluation time of non-separable functions on GPU**, Computing and Informatics, 1-21, ISSN: 1335-9150, Accepted at 2014, to publish at 2015, Accepted (Impact Factor = 0.254, 2012, Quartile Q4)

A.2 International Book Chapters Arising from this Thesis

1. Cárdenas-Montes, Miguel, Vega-Rodríguez, Miguel A., and Mollá, Mercedes: **Metaheuristics for Modelling Low-Resolution Galaxy Spectral Energy Distribution**, HAIS, LNAI, Springer, 490-501, Eds: Polycarpou, Marios M.,

Carvalho, André C. P. L. F., Pan, Jeng-Shyang, Wozniak, Michal, Quintián-Pardo, Héctor, and Corchado, Emilio, 2014

2. Cárdenas-Montes, Miguel, Vega-Rodríguez, Miguel A., and Mollá, Mercedes: **Metaoptimization of Differential Evolution by Using Productions of Low-Number of Cycles: The Fitting of Rotation Curves of Spiral Galaxies as Case Study**, HAIS, LNAI, Springer, 356-365, Eds: Pan, Jeng-Shyang, Polycarpou, Marios M., Wozniak, Michal, Carvalho, André C. P. L. F., Quintián-Pardo, Héctor, and Corchado, Emilio, 2013

3. Cárdenas-Montes, Miguel, Vega-Rodríguez, Miguel A., Sevilla, Ignacio, Ponce, Rafael, Rodríguez-Vázquez, Juan José, Sánchez Álvaro, Eusebio: **Concurrent CPU-GPU Code Optimization: The Two-Point Angular Correlation Function as Case Study**, CAEPIA, LNAI, Springer, 209-218, Eds: Bielza, C.; Salmeron, A.; Alonso-Betanzos, A.; Hidalgo, J.I.; Martínez, L.; Troncoso, A.; Corchado, E.; Corchado, J.M., 2013

4. Cárdenas-Montes, Miguel, Vega-Rodríguez, Miguel A., and Gómez-Iglesias, Antonio: **Real-World Problem for Checking the Sensitiveness of Evolutionary Algorithms to the Choice of the Random Number Generator**, HAIS (1), LNAI, Springer, 385-396, Eds: Corchado, Emilio, Snásel, Václav, Abraham, Ajith, Wozniak, Michal, Graña, Manuel, and Cho, Sung-Bae, 2012

5. Cárdenas-Montes, Miguel, Mollá, Mercedes, Vega-Rodríguez, Miguel A., Rodríguez-Vázquez, Juan José, and Gómez-Iglesias, Antonio: **Adjustment of Observational Data to Specific Functional Forms Using Particle Swarm Algorithm and Differential Evolution: Rotational Curves of Spiral Galaxy as Case Study**, Astrostatistics and Data Mining, Springer, 81-88, 2012

6. Ponce, Rafael, Cárdenas-Montes, Miguel, Rodríguez-Vázquez, Juan Jose, Sevilla, Ignacio: **Application of GPUs for the Calculation of Two Point Correlation Functions in Cosmology**, Astronomical Data Analysis Software and Systems XXI, Astronomical Society of the Pacific Conference Series, 461, 73-76, 2012

7. Cárdenas-Montes, Miguel, Vega-Rodríguez, Miguel A., and Gómez-Iglesias, Antonio: **Sensitiveness of Evolutionary Algorithms to the Random Number Generator**, ICANNGA (1), LNCS, Springer, 371-380, Eds: Dobnikar, Andrej, Lotric, Uros, and Ster, Branko, 2011

8. Cárdenas-Montes, Miguel, Vega-Rodríguez, Miguel A., Rodríguez-Vázquez, Juan José, and Gómez-Iglesias, Antonio: **Effect of the Block Occupancy in GPGPU over the Performance of Particle Swarm Algorithm**, ICANNGA (1), LNCS, Springer, 310-319, Eds: Dobnikar, Andrej, Lotric, Uros, and Ster, Branko, 2011

9. Cárdenas-Montes, Miguel, Vega-Rodríguez, Miguel A., Rodríguez-Vázquez, Juan José, and Gómez-Iglesias, Antonio: **GPU-Based Evaluation to Accelerate Particle Swarm Algorithm**, EUROCAST (1), LNCS, Springer, 272-279, Eds: Moreno-Díaz, Roberto, Pichler, Franz, and Quesada-Arencibia, Alexis, 2011

10. Cárdenas-Montes, Miguel, Vega-Rodríguez, Miguel A., Gómez-Iglesias, Antonio, and Morales-Ramos, Enrique: **Empirical Study of Performance of Particle Swarm Optimization Algorithms Using Grid Computing**, Nature Inspired Cooperative Strategies for Optimization, NICSO, Springer, 345-357, Eds: González, Juan Ramón, Pelta, David A., Cruz, Carlos, Terrazas, Germán, and Krasnogor, Natalio, 2010

11. Cárdenas-Montes, Miguel, Vega-Rodríguez, Miguel A., and Gómez-Iglesias, Antonio: **Performance Improvement in Multipopulation Particle Swarm Algorithm**, Distributed Computing and Artificial Intelligence: 7th International Symposium, DCAI, Springer, 533-540, Eds: Andre Ponce de Leon, F. de Carvalho, Sara Rodríguez-González, Juan F. De Paz, and Juan M. Corchado, 2010

12. Cárdenas-Montes, Miguel, Vega-Rodríguez, Miguel A., García Orellana, Carlos J., Rubio del Solar, Manuel, Gómez Pulido, Juan Antonio, González Velasco, Horacio M., Gómez-Iglesias, Antonio, Sánchez-Pérez, Juan Manuel, and Macías Macías, Miguel: **Volunteer Computing, an Interesting Option for Grid Computing: Extremadura as Case Study**, OTM Workshops (1), LNCS, Springer, 29-30, Eds: Meersman, Robert, Tari, Zahir, and Herrero, Pilar, 2007

A.3 International Conference Proceedings Arising from this Thesis

1. Cárdenas-Montes, Miguel, Rodríguez-Vázquez, Juan José, Vega-Rodríguez, Miguel A., Sevilla, Ignacio, Sánchez Álvaro, Eusebio, Ponce, Rafael, Bonnett, Christopher: **High-Performance Implementations for Shear-Shear Correlation Calculation**. Cluster, IEEE Computer Society, 2014. Submitted.

2. Cárdenas-Montes, Miguel, Vega-Rodríguez, Miguel A., Westerholm, Jan, Ponce, Rafael, Yurtesen, Evren, Sevilla, Ignacio, Sánchez Álvaro, Eusebio, Rodríguez-Vázquez, Juan José, Aspnäs, Mats, Timonen, Ville, Colino, Nicanor, and Gómez-Iglesias, Antonio: **Calculation of Two-Point Angular Correlation Function: Implementations on Many-Core and Multicore Processors**, Ibergrid, Editorial Universitat Politecnica de Valencia, 203-214, 2013

3. Cárdenas-Montes, Miguel, Rodríguez-Vázquez, Juan Jose, Ponce, Rafael, Sevilla, Ignacio, Sánchez Álvaro, Eusebio, Colino, Nicanor, and Vega-Rodríguez, Miguel A.: **New Computational Developments in Cosmology**, Ibergrid, Laboratorio de Instrumentaçao e Física Experimental de Partículas, 101-112, 2012

4. Cárdenas-Montes, Miguel, Vega-Rodríguez, Miguel A., Rodríguez-Vázquez, Juan Jose, and Gómez-Iglesias, Antonio: **Accelerating Particle Swarm Algorithm with GPGPU**, 16th Euromicro International Conference on Parallel, Distributed and Network-Based Processing, PDP, IEEE Computer Society, 560-564, Eds: Cotronis, Yiannis, Danelutto, Marco, and Papadopoulos, George Angelos, 2011

5. Cárdenas-Montes, Miguel, Vega-Rodríguez, Miguel A., Rodríguez-Vázquez, Juan José, and Gómez-Iglesias, Antonio: **GPGPU-based Evaluation of Particle Swarm Algorithm for High-Dimensional Problems**, EUROCAST 2011 Book of Abstracts, Universidad de Las Palmas de Gran Canaria, 252-253, Eds: Quesada-Arencibia, Alexis, Rodríguez-Rodríguez, José Carlos, Moreno-Díaz, Roberto jr., and Moreno-Díaz, Roberto, 2011

6. Cárdenas-Montes, Miguel, Vega-Rodríguez, Miguel A., Gómez-Iglesias, Antonio, and Morales-Ramos, Enrique: **Conjecture of Bateman Solver with Particle Swarm Optimisation and Grid Computing**, 3rd Iberian Grid Infrastructure Conference, Ibergrid, Netbiblo, 269-280, 2009

7. Cárdenas-Montes, Miguel, Vega-Rodríguez, Miguel A., Gómez-Iglesias, Antonio, Castejón-Magaña, Francisco, Arriero, Nicanor Colino, and Morales-Ramos, Enrique: **Grid Application Taxonomies and Models for its Adaptation**, 3rd Iberian Grid Infrastructure Conference, Ibergrid, Netbiblo, 420-431, 2009

8. Cárdenas-Montes, Miguel, Vega-Rodríguez, Miguel A., Gómez-Iglesias, Antonio, and Morales-Ramos, Enrique: **Exploration of the Conjecture of Bateman using Particle Swarm Optimization and Grid Computing**, 8th International Symposium on Parallel and Distributed Computing, ISPDC, IEEE Computer Society, 143-150, 2009

9. Cárdenas-Montes, Miguel, Vega-Rodríguez, Miguel A., García Orellana, Carlos J., Rubio del Solar, Manuel, Gómez-Pulido, Juan A., González Velasco, Horacio, Gómez-Iglesias, Antonio, Sánchez-Pérez, Juan Manuel, and Macías Macías, Miguel: **The Fruitful Application of Volunteer Computing to Regions with Low Scientific Funds: Extremadura as an Example**, 2nd Iberian Grid Infrastructure Conference, Ibergrid, Netbiblo, 129-140, 2008

10. Cárdenas-Montes, Miguel, Vega-Rodríguez, Miguel A., Rubio del Solar, Manuel, and Gómez-Iglesias, Antonio: **Bateman Conjecture's Exploration on Grid Computing**, 2nd Iberian Grid Infrastructure Conference, Ibergrid, Netbiblo, 371-377, 2008

A.4 Other Publications Arising from this Thesis

1. Cárdenas-Montes, Miguel, Vega-Rodríguez, Miguel A., Gómez-Iglesias, Antonio, and Morales-Ramos, Enrique: **Explorando la Conjetura de Bateman en un Entorno Grid**, XIX Jornadas de Paralelismo, Universitat Jaume I, 511-514, Sep 2008

A.5 Publications No-Related to PhD

1. Rodríguez-Vázquez, Juan José, Vázquez-Poletti, José Luis, Delgado-Méndez, Carlos José, and Cárdenas-Montes, Miguel: **Performance Evaluation of a Signal Extraction Algorithm for the Cherenkov Telescope Array's Real Time Analysis Pipeline** Cluster, IEEE Computer Society, 2014, Submitted

2. Cárdenas-Montes, Miguel: **Depth-based Outlier Detection Algorithm**, HAIS, LNAI, Springer, 122-132, Eds: Polycarpou, Marios M., Carvalho, André C. P. L. F., Pan, Jeng-Shyang, Wozniak, Michal, Quintián-Pardo, Héctor, and Corchado, Emilio, 2014

3. Rodríguez-Vázquez, Juan José, and Cárdenas-Montes, Miguel: **Performance Assessment of a Chaos-based Image Cipher on Multi-GPU**, Ibergrid, Editorial Universitat Politecnica de Valencia, 592-599, 2013

4. Rodríguez-Vázquez, Juan José, and Cárdenas-Montes, Miguel: **Speeding Up a Chaos-Based Image Encryption Algorithm Using GPGPU**, EUROCAST (1), LNCS, Springer, 272-279, Eds: Moreno-Díaz, Roberto, Pichler, Franz, and Quesada-Arencibia, Alexis, 2011

5. Cárdenas-Montes, Miguel, Franco Valiente, José Miguel, Cortés Fácila, Álvaro, Díaz Corchero, Miguel Ángel, Gómez-Tostón Gutiérrez, Carolina, Martínez Ramírez, Adrián, Suárez Ortega, César, and Gómez Pulido, Juan Antonio: **Population-Based Incremental Learning Algorithm to Search for Magic Squares**, 5th Iberian Grid Infrastructure Conference, Ibergrid, Netbiblo, 315-326, 2011

6. Gómez-Iglesias, Antonio, Vega-Rodríguez, Miguel A., Castejón, Francisco, and Cárdenas-Montes, Miguel: **Distributed and Asynchronous Bees Algorithm Applied to Nuclear Fusion Research**, PDP, IEEE Computer Society, 365-372, Eds: Cotronis, Yiannis, Danelutto, Marco, and Papadopoulos, George Angelos, 2011

7. Rodríguez-Vázquez, Juan José, Cárdenas-Montes, Miguel, and Romero-Sánchez, Sixto: **Speeding Up a Chaotic Image Encryption Algorithm on GPU**, EUROCAST 2011 Book of Abstracts, Universidad de Las Palmas de Gran Canaria, 132-133, Eds: Quesada-Arencibia, Alexis, Rodríguez-Rodríguez, José Carlos, Moreno-Díaz, Roberto jr., and Moreno-Díaz, Roberto, 2011

8. Gómez-Iglesias, Antonio, Vega-Rodríguez, Miguel A., Castejón, Francisco, and Cárdenas-Montes, Miguel: **Distributed and Asynchronous Bees Algorithm: an Efficient Model for Large Scale Problems Optimizations**, Distributed Computing and Artificial Intelligence: 7th International Symposium, DCAI, Springer, 381-388, 2010

9. Gómez-Iglesias, Antonio, Vega-Rodríguez, Miguel A., Castejón, Francisco, Cárdenas-Montes, Miguel, and Morales-Ramos, Enrique: **Artificial Bee Colony Inspired Algorithm Applied to Fusion Research in a Grid Computing Environment**, PDP, IEEE Computer Society, 508-512, Eds: Danelutto, Marco, Bourgeois, Julien, and Gross, Tom, 2010

10. Gómez-Iglesias, Antonio, Vega-Rodríguez, Miguel A., Castejón-Magaña, Francisco, and Cárdenas-Montes, Miguel: **Algoritmo de Abejas Asíncrono y Distribuido**, Actas de las XXI Jornadas de Paralelismo, Ibergarceta Publicaciones S.L., Valencia, España, 905-911. ISBN: 978-84-92812-49-3, 2010

11. Gómez-Iglesias, Antonio, Vega-Rodríguez, Miguel A., Castejón, Francisco, Cárdenas-Montes, Miguel, and Morales-Ramos, Enrique: **Evolutionary Computation and Grid Computing to Optimise Nuclear Fusion Devices**, Cluster Computing - The Journal of Networks, Software Tools and Applications 12(4),

Springer, 439-448, ISSN: 1386-7857, 2009 (Impact factor 0.695, 2009, Quartile Q3)

12. Gómez-Iglesias, Antonio, Vega-Rodríguez, Miguel A., Castejón, Francisco, Cárdenas-Montes, Miguel, Morales-Ramos, Enrique, and Reynolds, J. M.: **Grid-based Metaheuristics to Improve a Nuclear Fusion Device**, Concurrency and Computation: Practice and Experience 22(11), John Wiley & Sons, 1476-1493, ISSN:1532-0626, 2009 (Impact factor 1.004, 2009, Quartile Q3)

13. Gómez-Iglesias, Antonio, Vega-Rodríguez, Miguel A., Castejón-Magaña, Francisco, Cárdenas-Montes, Miguel, and Morales-Ramos, Enrique: **Grid-Enabled Mutation-Based Genetic Algorithm to Optimise Nuclear Fusion Devices**, EUROCAST, Springer, 809-816, Eds: Moreno-Díaz, Roberto, Pichler, Franz, and Quesada-Arencibia, Alexis, 2009

14. Gómez-Iglesias, Antonio, Vega-Rodríguez, Miguel A., Cárdenas-Montes, Miguel, Morales-Ramos, Enrique, and Castejón-Magaña, Francisco: **Grid-Oriented Scatter Search Algorithm**, ICANNGA, Springer, 193-202, Eds: Kolehmainen, Mikko, Toivanen, Pekka J., and Beliczynski, Bartlomiej, 2009

15. Gómez-Iglesias, Antonio, Vega-Rodríguez, Miguel A., Castejón-Magaña, Francisco, Cárdenas-Montes, Miguel, and Morales-Ramos, Enrique: **Scatter Search and Grid Computing to Improve Nuclear Fusion Devices**, Large-Scale Scientific Computing, LNCS, Springer, 483-490, 2009

16. Gómez-Iglesias, Antonio, Vega-Rodríguez, Miguel A., Castejón-Magaña, Francisco, Cárdenas-Montes, Miguel, and Morales-Ramos, Enrique: **A Grid-Oriented Crossover Genetic Algorithm to Optimise Nuclear Fusion Devices**, 3rd Iberian Grid Infrastructure Conference, Ibergrid, Netbiblo, 281-290, 2009

17. Gómez-Iglesias, Antonio, Vega-Rodríguez, Miguel A., Castejón-Magaña, Francisco, Cárdenas-Montes, Miguel, and Morales-Ramos, Enrique: **Mutation-based genetic algorithm and the grid to optimise nuclear fusion devices**, EUROCAST 2009 Book of Abstracts, Universidad de Las Palmas de Gran Canaria, 280-281, Eds: Rodríguez-Rodríguez, José Carlos, Moreno-Díaz, Roberto, Quesada-Arencibia, Alexis, Moreno-Díaz, Roberto jr.

18. Gómez-Iglesias, Antonio, Vega-Rodríguez, Miguel A., Castejón-Magaña, Francisco, Sánchez-Pérez, Juan M., Cárdenas-Montes, Miguel, and Morales-Ramos, Enrique: **Implementación grid de algoritmo Scatter Search. Aplicación**

a la optimización de dispositivos de fusión nuclear, XX Jornadas de Paralelismo, Servizo de Publicacións, Universidade da Coruña, A Coruña, Spain, 665-669. ISBN: 84-9749-346-8, 2009

19. Gómez-Iglesias, Antonio, Vega-Rodríguez, Miguel A., Castejón-Magaña, Francisco, Cárdenas-Montes, Miguel, and Morales-Ramos, Enrique: **Using a Genetic Algorithm and the Grid to Improve Transport Levels in the TJ-II Stellarator**, 7th International Symposium on Parallel and Distributed Computing, ISPDC, IEEE Computer Society, 81-88, 2008

20. Rubio-Solar, Manuel, Vega-Rodríguez, Miguel A., Sánchez-Pérez, Juan Manuel, Gómez-Iglesias, Antonio, and Cárdenas-Montes, Miguel: **A FPGA Optimization Tool Based on a Multi-island Genetic Algorithm Distributed over Grid Environments**, 8th IEEE International Symposium on Cluster Computing and the Grid, CCGrid, IEEE, 65-72, 2008

21. Fedak, Gilles, He, Haiwu, Lodygensky, Oleg, Balaton, Zoltan, Farkas, Zoltan, Gombas, Gabor, Kacsuk, Peter, Lovas, Robert, Marosi, Attila Csaba, Kelley, Ian, Taylor, Ian, Terstyanszky, Gabor, Kiss, Tamas, Cárdenas-Montes, Miguel, Emmen, Ad, and Araujo, Filipe: **EDGeS: A Bridge Between Desktop Grids and Service Grids**, Proceedings of The Third ChinaGrid Annual Conference (chinagrid 2008), IEEE Computer Society, 3-9, 2008

22. Balaton, Zoltán, Farkas, Zoltan, Gombás, Gabor, Kacsuk, Péter, Lovas, Róbert, Marosi, Csaba Attila, Emmen, Ad, Terstyánszky, Gábor, Kiss, Tamás, Kelley, Ian, Taylor, Ian, Lodygensky, Oleg, Cárdenas-Montes, Miguel, Fedak, Gilles, and Araujo, Filipe: **EdgES: the Common Boundary between Service and Desktop Grids**, Parallel Processing Letters 18(3), 433-445, ISSN: 0129-6264, 2008

23. Castrillo, Francisco Prieto, Cárdenas-Montes, Miguel, and Rubio-Solar, Manuel: **gPhase: A Grid-based technology for the analysis of phase space structure in nonlinear dynamics. Applications to rocking systems under armonic loading**, 2nd Iberian Grid Infrastructure Conference, Ibergrid, Netbiblo, 105-114, 2008

24. Castejón Magaña, Francisco, Cárdenas-Montes, Miguel, Gómez-Iglesias, Antonio, Morales-Ramos, Enrique, Guillerminet, Bernard, Coster, David P., Sonnendrücker, Eric, Campos-Plasencia, Isabel, Åström, Jan, Westerholm, Jan, Cela, José M., Kos, Leon, Smith, Lorna, Plociennik, Marcin, Hardt, Marcus, Aspnäs,

Mats, Strand, Pär, and Stotzka, Rainer: **EUFORIA: Grid and High Performance Computing at the Service of Fusion Modelling**, 2nd Iberian Grid Infrastructure Conference, Ibergrid, Netbiblo, 115-126, 2008

25. Cárdenas-Montes, Miguel, Emmen, Ad, Csaba Marosi, Attila, Araujo, Filipe, Gombás, Gábor, Terstyanszky, Gabor, Fedad, Gilles, Kelley, Ian, Taylor, Ian, Lodygensky, Oleg, Kacsuk, Péter, Lovas, Robert, Kiss, Tamas, Balaton, Zoltán, and Farkas, Zoltán: **EDGeS: bridging Desktop and Service Grids**, 2nd Iberian Grid Infrastructure Conference, Ibergrid, Netbiblo, 212-223, 2008

26. Gómez-Iglesias, Antonio, Vega-Rodríguez, Miguel A., Sánchez-Pérez, Juan Manuel, Cárdenas-Montes, Miguel, Rubio-Solar, Manuel, and Morales-Ramos, Enrique: **Grid Computing in Chess Endgames**, 2nd Iberian Grid Infrastructure Conference, Ibergrid, Netbiblo, 227-238, 2008

27. Lombraña González, Daniel, Fernández de Vega, Francisco, Trujillo, L., Olague, G., Cárdenas-Montes, Miguel, Araujo, L., Castillo, P., Sharman, K., and Silva, A.: **Interpreted Applications within BOINC Infrastructure**, 2nd Iberian Grid Infrastructure Conference, Ibergrid, Netbiblo, 261-272, 2008

28. Antoli, B., Benito, D., Cárdenas-Montes, Miguel, Castejón-Magaña, F., Castellanos, C., Castelo, V., de Alfonso, C., de Miguel, T., Gavela, R., Hernandez, V., Herraiz, L., Ibar, J., Jimenez, L., Ramos, R., Rivero, A., Saenz, J.F., Serrano, F., Rubio, M., Tarancón, A., and Vuillemin, P.: **CIVICO. Citizen Volunteer Infrastructure for Computing**, 2nd Iberian Grid Infrastructure Conference, Ibergrid, Netbiblo, 273-284, 2008

29. Morales-Ramos, Enrique, Vega-Rodríguez, Miguel A., Gómez-Iglesias, Antonio, Cárdenas-Montes, Miguel, Gómez Pulido, Juan Antonio, and Sanchez-Bajo, Florentino: **Peaks Detection in X-Ray Diffraction Profiles Using Grid Computing**, OTM Workshops, LNCS, Springer, 793-801, 2008

30. Gómez-Iglesias, Antonio, Vega-Rodríguez, Miguel A., Castejón-Magaña, Francisco, del Solar, Manuel Rubio, and Cárdenas-Montes, Miguel: **Grid Computing in Order to Implement a Three-Dimensional Magnetohydrodynamic Equilibrium Solver for Plasma Confinement**, PDP, IEEE Computer Society, 435-439, 2008

31. Gómez-Iglesias, Antonio, Vega-Rodríguez, Miguel A., Castejón-Magaña, Francisco, Cárdenas-Montes, Miguel, and Morales-Ramos, Enrique: **Algoritmo Genético para la Optimización del Transporte en un Stellarator Utilizando la Computación Grid**, XIX Jornadas de Paralelismo, Universitat Jaume I, 515-520, Sep 2008

32. Morales-Ramos, Enrique, Vega-Rodríguez, Miguel A., Gómez-Iglesias, Antonio, Cárdenas-Montes, Miguel, Gómez Pulido, Juan Antonio, and Sanchez-Bajo, Florentino: **Uso de la Computación Grid en el Reconocimiento de Picos en Perfiles de Difracción de Rayos X**, XIX Jornadas de Paralelismo, Universitat Jaume I, 538-543, 2008

33. Cárdenas-Montes, Miguel: **Desarrollo de Computación Grid Basada en Software Libre**, III Conferencia Internacional de Software Libre, 157-165, 2007

34. Cárdenas-Montes, Miguel, Perez-Griffo, Javier, Rubio del Solar, Manuel, and Ramos, Raul: **Management of a Grid Infrastructure in GLITE with Virtualization**, 1st Iberian Grid Infrastructure Conference, Ibergrid, Netbiblo, 313-321, 2007

35. Gómez-Iglesias, Antonio, Vega-Rodríguez, Miguel A., Castejón-Magaña, Francisco, Rubio-Solar, Manuel, and Cárdenas-Montes, Miguel: **Optimizing the configuration of magnetic confinement devices with evolutionary algorithms and grid computing**, 3rd EELA Conference, 507-514, 2007

36. Gómez-Iglesias, Antonio, Vega-Rodríguez, Miguel A., Rubio del Solar, Manuel, and Cárdenas-Montes, Miguel: **Algoritmo Evolutivo para Finales de Ajedrez en un Entorno Grid**, XVIII Jornadas de Paralelismo, II Congreso Español de Informática, Thomson-Paraninfo, Zaragoza, España, 507-514. ISBN: 978-84-9732-593-6, 2007

37. Rubio del Solar, Manuel, Perez-Griffo, Javier, and Cárdenas-Montes, Miguel: **BOINC Developments in the Extremadura Centre of Advanced Technologies**, Vértices, Ed. CIEMAT, ISSN: 1887-1461, 26-29, 2006.

Appendix B

Other Activities

This section lists the participation in teaching activities, research projects, program committee of international conferences, and reviewer of JCR-indexed journals.

B.1 Teaching

I collaborated in the following tutorials, training courses and academic activities:

- One session at Máster Física Teórica, "Física Experimental de Partículas y Cosmología", Universidad Complutense de Madrid, 2013-2014.
- Training activities, internal at CIEMAT (editions with duration varying from 12 to 25 hours). In most of the activities, I acted as proposer and coordinator of the training activity:
  - *Gráficas, estadística y minería de datos con Python*, 2013
  - *Técnicas de optimización aplicadas a problemas científico-técnicos*, (2010, 2012).
  - *Técnicas de programación en CUDA (GPU) aplicadas a problemas científico-técnicos*, (2010, 2011).
  - *Python para científicos*, (2 editions at 2010, 2011)
  - *Iniciación a la Computación Grid*, (2010)
- Máster de Computación Grid y Paralelismo (University of Extremadura). In the editions 2007-2008 and 2008-2009 I taught "Fundamentos de Computación Grid y Proyectos Actuales". In the editions from 2009-2010 to 2013-2014, I taught "Adaptación de Aplicaciones Grid para el Procesamiento de Imágenes". In all cases 3 credits was taught per year.

- Training course at CIEMAT, in April 14th of 2009 focused on the optimisation of large-scale problems with metaheuristics and the grid. Professor and Coordinator.
- Training —1 day— on CUDA programming during the Annual International Tokamak Modelling at Lisbon during September 2010.
- Training —1 day— on CUDA programming during the EFDA Goal Oriented Training in Theory (GOTiT) at the Max-Planck Institute for Plasma Physics (IPP) in Garching, Germany during October 18-29, 2010.
- Summer course: *Computación Inteligente Distribuida: La Naturaleza como Fuente de Inspiración*, 2008, Universidad de Extremadura.
- 16th EELA Tutorial for Grid Users, held in Badajoz, Spain, in November 12th to the 14th of 2007. Professor and Coordinator.
- 15th EELA Tutorial for Grid Users, held in Mérida, Spain, in November 5th to the 7th of 2007. Professor and Coordinator.
- 11th EELA Tutorial for Grid Users, held in Sevilla, Spain, in September 10th to 14th of 2007.
- 7th EELA Tutorial for Grid Users, held in Mérida, Spain, in November 7th to the 9th of 2007.
- Summer course: *Software Libre a Fondo*, Computación Grid con Software Libre, 2007, Universidad de Extremadura.
- Round table on grid computing, *Tendiendo Puentes en Computación Grid*, 2007, Universidad de Extremadura.
- Summer course: *Las Fronteras de la Computación: desde el Grid hasta la Computación Cuántica*, 2006, Universidad de Extremadura.

B.2 Participation as Program Committee of International Conferences

During the period of this Thesis, I have been member of the program committee for the following international conferences:

- International Workshop on Parallelism in Bioinformatics (PBio 2014), in Cluster (IEEE), Madrid, Spain, 2014.

- 7th, 8th Iberian Grid Infrastructure Conference, corresponding to Ibergrid'13 (Lisbon) and Ibergrid'14 (Madrid).
- International Workshop on Parallelism in Bioinformatics (PBio 2013), in EuroMPI (ACM), Madrid, Spain, 2013.
- 7th International Conference on Distributed and Parallel Systems (DAPSYS), in Springer, Debrecen, Hungary, 2008

B.3 Reviewer of JCR-indexed Journals

- Parallel Computing (PARCO), Elsevier, ISSN: 0167-8191, Impact Factor: 1.311, Quartile Q1. Special issue from PBio 2013.

B.4 Research Projects Participation

During the period of this Thesis, I have participated in the following research projects:

- **Contribución Del CIEMAT al Tier-2 Español de CMS**. FPA2005-07256-C2-02. Founded by Ministerio de Educación y Ciencia, Proyectos I+D, acciones estratégicas y eranets. From December 2005 to June 2008.
- **EELA (E-science grid facility for Europe and Latin America)**. RI-22379. Founded by European Commission, FP VII. From January 2006 to December 2007.
- **EGEE-II (Enabling Grids for E-sciencE II)**. RI-031688. Founded by European Commission. FP VI. From May 2006 to April 2008.
- **GRIDEX (Transformación de Códigos Científicos e Industriales a Entornos de Computación Distribuida)**. PRI06A223. Founded by III Plan Regional de Investigación, Desarrollo e Innovación (PRI+D+I, 2005-2008), Junta de Extremadura. From 2006 to 2008.
- **Desarrollo y operación de un Tier-2 federado para el experimento CMS (CIEMAT)**. FPA2007-66530-C02-02. Founded by Ministerio de Educación y Ciencia, Plan Nacional de I+D+i, Proyectos I+D Acciones Estratégicas y Eranets. From October 2007 to October 2010.
- **EGEE-III (Enabling Grids for EsciencE-III)**. RI-222667. Founded by European Commission. FP VII. From May 2008 to April 2010.

- **EDGeS (Enabling Desktop Grids for eScience)**. FP7-2007-211727. Founded by European Commission. FP VII. From January 2008 to December 2009.
- **EUFORIA (EU fusion for ITER Applications)**. FP7-2007-211804. Founded by European Comision. FP VII. From January 2008 to December 2010.
- **Métodos Cinéticos en Plasmas de Fusión**. ENE2008-06082. Founded by Ministerio de Educación y Ciencia, Programa Nacional de Proyectos de Investigación Fundamental en el Marco del Plan Nacional de I+D+i 2008-2011. From January 2009 to December 2011.
- **EGI-InSPIRE (European Grid Initiative: Integrated Sustainable Pan-European Infrastructure for Researchers in Europe)**. RI-261323. Founded by European Commission, FP VII. From May 2010 to December 2014.
- **Centro de Procesado de Datos de Colisiones Hadrónicas del LHC: TIER-2 Federado para el Experimento CMS**. FPA2010-21638-C02-02. Founded by Ministerio de Ciencia e Innovación, Plan Nacional de I+D+i, Programa Nacional de Proyectos de Investigación Fundamental. From 2011 to 2013.
- **AstroMadrid**. CAM S2009/ESP-1496. Founded by Comunidad de Madrid (programas de actividades de I+D entre grupos de investigación de la Comunidad de Madrid y convocatoria en tecnologías cofinanciada con Fondo Social Europeo). From January 2010 to December 2014.
- **VENUS-C (Virtual multidisciplinary EnviroNments USing Cloud infrastrutures)**. Founded by Engineering Ingegneria Informatica S.p.A., subcontracted, FP-VII. From January 2012 to December 2012.

At projects EELA, EDGES, and EUFORIA I participated in the elaboration of the proposal of the projects. At projects EELA and EUFORIA I led a work package.

B.5 Other Merits

- Chairman at sessions: "Hybrid Intelligent Systems for Data Mining and Applications" and "Data Mining and Knowledge Discovery" at HAIS 2014 (Springer), Salamanca.
- Presenter at session "GPGPU Integration and GPGPU User Application Support" at EGI Community Forum 2014, Helsinki.

- Convener and presenter at session "Bio-inspired Algorithms in Grid" at EGEE'09, Barcelona.
- Coauthor of the poster "Evolutionary Algorithm for Solving Chess Endgames in Grid Environments" at EGEE'07, Budapest.

Bibliography

[1] T. Abbott et al. The dark energy survey. *AIP Conf. Proc.*, **842**:989–991, 2005. 95

[2] Enrique Alba and Marco Tomassini. Parallelism and evolutionary algorithms. *IEEE Trans. Evolutionary Computation*, **6**(5):443–462, 2002. 43

[3] Luca Amendola and Euclid Theory Working Group. Cosmology and fundamental physics with the euclid satellite. *Living Reviews in Relativity*, **16**(6), 2013. 95

[4] David P. Anderson. Boinc: A system for public-resource computing and storage. In *Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing*, GRID '04, pages 4–10, Washington, DC, USA, 2004. IEEE Computer Society. 21

[5] D. Bard, M. Bellis, M.T. Allen, H. Yepremyan, and J.M. Kratochvil. Cosmological calculations on the GPU. *Astronomy and Computing*, **1**(0):17 – 22, 2013. 99, 100, 103

[6] Matthias Bartelmann and Peter Schneider. Weak gravitational lensing. *Physics Reports*, **340**(4–5):291 – 472, 2001. 97

[7] Jonathan Benjamin, Ludovic Van Waerbeke, Catherine Heymans, Martin Kilbinger, Thomas Erben, Hendrik Hildebrandt, Henk Hoekstra, Thomas D. Kitching, Yannick Mellier, Lance Miller, Barnaby Rowe, Tim Schrabback, Fergus Simpson, Jean Coupon, Liping Fu, Joachim Harnois-Draps, Michael J. Hudson, Konrad Kuijken, Elisabetta Semboloni, Sanaz Vafaei, and Malin Velander. CFHTLenS tomographic weak lensing: quantifying accurate redshift distributions. *Monthly Notices of the Royal Astronomical Society*, **431**(2):1547–1564, 2013. 116

[8] Fran Berman, Geoffrey Fox, and Anthony J. G. Hey. *Grid Computing: Making the Global Infrastructure a Reality*. John Wiley & Sons, Inc., New York, NY, USA, 2003. 17

[9] G. M. Bernstein and M. Jarvis. Shapes and shears, stars and smears: Optimal measurements for weak lensing. *The Astronomical Journal*, **123**(2):583–618, 2002. 98

[10] Mauro Birattari. *Tuning Metaheuristics - A Machine Learning Perspective*, **197** of *Studies in Computational Intelligence*. Springer, 2009. 70

[11] Mauro Birattari, Thomas Stützle, Luis Paquete, and Klaus Varrentrapp. A racing algorithm for configuring metaheuristics. In *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference, New York, USA, 9-13 July 2002*, pages 11–18. Morgan Kaufmann, 2002. 70

[12] Erick Cantú-Paz. On random numbers and the performance of genetic algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO*, pages 311–318. Morgan Kaufmann, 2002. 16, 70

[13] Miguel Cárdenas-Montes, Juan Jose Rodríguez-Vázquez, Rafael Ponce, Ignacio Sevilla, Eusebio Sánchez, Nicanor Colino, and Miguel A. Vega-Rodríguez. New computational developments in cosmology. In *6th Iberian Grid Infrastructure Conference Proceedings, Ibergrid*, pages 101–112, Lisbon. Portugal, 2012. Laboratório de Instrumentaçao e Física Experimental de Partículas. 101, 128, 136

[14] Miguel Cárdenas-Montes, Miguel A. Vega-Rodríguez, Christopher Bonnett, Ignacio Sevilla-Noarbe, Rafael Ponce, Eusebio Sánchez Álvaro, and Juan José Rodríguez-Vázquez. GPU-based shear-shear correlation calculation. *Computer Physics Communications*, **185**(1):11–18, 2014. 101, 128, 133, 138

[15] Miguel Cárdenas-Montes, Miguel A. Vega-Rodríguez, and Antonio Gómez-Iglesias. Sensitiveness of evolutionary algorithms to the random number generator. In Andrej Dobnikar, Uros Lotric, and Branko Ster, editors, *Adaptive and Natural Computing Algorithms - 10th International Conference, ICANNGA 2011, Ljubljana, Slovenia, April 14-16, 2011, Proceedings, Part I*, **6593** of *Lecture Notes in Computer Science*, pages 371–380. Springer, 2011. 74

[16] Miguel Cárdenas-Montes, Miguel A. Vega-Rodríguez, Antonio Gómez-Iglesias, and Enrique Morales-Ramos. Empirical study of performance of particle swarm optimization algorithms using grid computing. In *Nature Inspired Cooperative Strategies for Optimization, NICSO*, **284** of *Studies in Computational Intelligence*, pages 345–357. Springer, 2010. 27

[17] Miguel Cárdenas-Montes, Miguel A. Vega-Rodríguez, Ignacio Sevilla, Rafael Ponce, Juan José Rodríguez-Vázquez, and Eusebio Sánchez Álvaro. Concurrent CPU-GPU code optimization: The two-point angular correlation function as case study. In *Advances in Artificial Intelligence - 15th Conference of the Spanish Association for Artificial Intelligence, CAEPIA*, **8109** of *Lecture Notes in Computer Science*, pages 209–218. Springer, 2013. 101, 128

[18] Miguel Cárdenas-Montes, Miguel A. Vega-Rodríguez, Jan Westerholm, Rafael Ponce, Evren Yurtesen, Ignacio Sevilla, Eusebio Sánchez Alvaro, Juan José Rodríguez-Vázquez, Mats Aspnas, Ville Timonen, Nicanor Colino, and Antonio Gómez-Iglesias. Calculation of two-point angular correlation function: Implementations on many-core and multicore processors. In *7th Iberian Grid Infrastructure Conference Proceedings, Ibergrid*, pages 203–214, Madrid. Spain, 2013. Editorial Universitat Politecnica de Valencia. 101, 128, 136

[19] P Charbonneau. Genetic algorithms in astronomy and astrophysics. *The Astrophysical Journal Supplement Series*, **101**:309–334, 1995. 69

[20] R. Cid Fernandes, A. Mateus, L. Sodré, G. Stasińska, and J. M. Gomes. Semi-empirical analysis of Sloan Digital Sky Survey galaxies - I. Spectral synthesis method. *Monthly Notices of the Royal Astronomical Society*, **358**:363–378, April 2005. 69

[21] R. Cid Fernandes, E. Pérez, R. García Benito, R. M. González Delgado, A. L. de Amorim, S. F. Sánchez, B. Husemann, J. Falcón Barroso, P. Sánchez-Blázquez, C. J. Walcher, and D. Mast. Resolving galaxies in time and space: I: Applying starlight to califa data cubes. *Astronomy & Astrophysics*, **557**, Apr 2013. 68

[22] M. Clerc and J. Kennedy. The particle swarm - explosion, stability, and convergence in a multidimensional complex space. *Evolutionary Computation, IEEE Transactions on*, **6**(1):58–73, 2002. 16, 31

[23] Charlie Conroy. Modeling the panchromatic spectral energy distributions of galaxies. *Annual Review of Astronomy and Astrophysics*, **51**(1):393–455, 2013. 68

[24] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, **20**(3):273–297, 1995. 57

[25] Lucas de P. Veronese and Renato A. Krohling. Differential evolution algorithm on the GPU with C-CUDA. In *IEEE Congress on Evolutionary Computation, CEC*, pages 1–7. IEEE, 2010. 41

[26] Kusum Deep and Jagdish Chand Bansal. Mean particle swarm optimisation for function optimisation. *Int. J. Comput. Intell. Stud.*, **1**(1):72–92, May 2009. 16

[27] Russell C. Eberhart. *Computational Intelligence: Concepts to Implementations*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2007. 16

[28] T. Erben, H. Hildebrandt, L. Miller, L. van Waerbeke, C. Heymans, H. Hoekstra, T. D. Kitching, Y. Mellier, J. Benjamin, C. Blake, C. Bonnett, O. Cordes, J. Coupon, L. Fu, R. Gavazzi, B. Gillis, E. Grocutt, S. D. J. Gwyn, K. Holhjem, M. J. Hudson, M. Kilbinger, K. Kuijken, M. Milkeraitis, B. T. P. Rowe, T. Schrabback, E. Semboloni, P. Simon, M. Smit, O. Toader, S. Vafaei, E. van Uitert, and M. Velander. CFHTLenS: the Canada–France–Hawaii telescope lensing survey – imaging data and catalogue products. *Monthly Notices of the Royal Astronomical Society*, **433**(3):2545–2563, 2013. 116

[29] H. K. Eriksen, P. B. Lilje, A. J. Banday, and K. M. Górski. Estimating n-point correlation functions from pixelized sky maps. *The Astrophysical Journal Supplement Series*, **151**(1):1–11, 2004. 99

[30] Fábio Fabris and Renato A. Krohling. A co-evolutionary differential evolution algorithm for solving min-max optimization problems implemented on GPU using C-CUDA. *Expert Syst. Appl.*, **39**(12):10324–10333, 2012. 41

[31] Xiao feng Xie, Wen jun Zhang, and Zhi lian Yang. A dissipative particle swarm optimization. In *Congress on Evolutionary Computation*, pages 1456–1461. IEEE Computer Society, 2002. 16

[32] L. Ferreira. Grid services programming and application enablement. Technical report, IBM, 2004. 20

[33] Michael J. Flynn. Some computer organizations and their effectiveness. *IEEE Trans. Comput.*, **21**(9):948–960, September 1972. 18

[34] María A. Franco, Natalio Krasnogor, and Jaume Bacardit. Speeding up the evaluation of evolutionary learning systems using GPGPUs. In *GECCO '10: Proceedings of the 12th annual conference on Genetic and evolutionary computation*, pages 1039–1046, New York, NY, USA, 2010. ACM. 41

[35] Joshua Frieman, Michael Turner, and Dragan Huterer. Dark Energy and the Accelerating Universe. *Ann.Rev.Astron.Astrophys.*, **46**:385–432, 2008. 95

[36] Salvador García, Alberto Fernández, Julián Luengo, and Francisco Herrera. A study of statistical techniques and performance measures for genetics-based machine learning: accuracy and interpretability. *Soft Comput.*, **13**(10):959–977, 2009. 8

[37] Salvador García, Daniel Molina, Manuel Lozano, and Francisco Herrera. A study on the use of non-parametric tests for analyzing the evolutionary algorithms' behaviour: a case study on the CEC'2005 special session on real parameter optimization. *J. Heuristics*, **15**(6):617–644, 2009. 7, 8

[38] David Goldberg. What every computer scientist should know about floating-point arithmetic. *ACM Comput. Surv.*, **23**:5–48, March 1991. 103, 127

[39] Juan Gómez-Luna, José María González-Linares, José Ignacio Benavides, and Nicolas Guil. An optimized approach to histogram computation on GPU. *Mach. Vis. Appl.*, **24**(5):899–908, 2013. 101

[40] Jiawei Han and Micheline Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufmann, 2000. 8, 57, 92

[41] Catherine Heymans, Ludovic Van Waerbeke, Lance Miller, Thomas Erben, Hendrik Hildebrandt, Henk Hoekstra, Thomas D. Kitching, Yannick Mellier, Patrick Simon, Christopher Bonnett, Jean Coupon, Liping Fu, Joachim Harnois-Déraps, Michael J. Hudson, Martin Kilbinger, Koenraad Kuijken, Barnaby Rowe, Tim Schrabback, Elisabetta Semboloni, Edo van Uitert, Sanaz Vafaei, and Malin Velander. CFHTLenS: the Canada–France–Hawaii telescope lensing survey. *Monthly Notices of the Royal Astronomical Society*, **427**(1):146–166, 2012. 116, 134

[42] H. Hildebrandt, T. Erben, K. Kuijken, L. van Waerbeke, C. Heymans, J. Coupon, J. Benjamin, C. Bonnett, L. Fu, H. Hoekstra, T. D. Kitching, Y. Mellier, L. Miller, M. Velander, M. J. Hudson, B. T. P. Rowe, T. Schrabback, E. Semboloni, and N. Benítez. CFHTLenS: improving the quality of photometric redshifts with precision photometry. *Monthly Notices of the Royal Astronomical Society*, **421**(3):2355–2367, 2012. 116

[43] Henk Hoekstra, Marijn Franx, Konrad Kuijken, and Pieter G. Van Dokkum. HST large-field weak lensing analysis of ms 2053–04: study of the mass distribution and mass-to-light ratio of X-ray luminous clusters at 0.22<z <0.83. *Monthly Notices of the Royal Astronomical Society*, **333**(4):911–922, 2002. 99

[44] Henk Hoekstra and Bhuvnesh Jain. Weak gravitational lensing and its cosmological applications. *Annual Review of Nuclear and Particle Science*, **58**(1):99–123, 2008. 98

[45] IEEE Standard for Floating-Point Arithmetic. Technical report, Microprocessor Standards Committee of the IEEE Computer Society, August 2008. 127

[46] M. Jarvis, G. Bernstein, and B. Jain. The skewness of the aperture mass statistic. *Monthly Notices of the Royal Astronomical Society*, **352**(1):338–352, 2004. 100

[47] Jelte T.A. Jong, Gijs A. Verdoes Kleijn, Konrad H. Kuijken, and Edwin A. Valentijn. The kilo-degree survey. *Experimental Astronomy*, **35**(1-2):25–44, 2013. 95

[48] J. Kennedy and R. Eberhart. Particle swarm optimization. In *IEEE International Conference on Neural Networks*, **4**, pages 1942–1948 vol.4. IEEE, November 1995. 16

[49] Carl Kesselman and Ian Foster. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers, November 1998. 10, 23

[50] Martin Kilbinger, Liping Fu, Catherine Heymans, Fergus Simpson, Jonathan Benjamin, Thomas Erben, Joachim Harnois-Déraps, Henk Hoekstra, Hendrik Hildebrandt, Thomas D. Kitching, Yannick Mellier, Lance Miller, Ludovic Van Waerbeke, Karim Benabed, Christopher Bonnett, Jean Coupon, Michael J. Hudson, Konrad Kuijken, Barnaby Rowe, Tim Schrabback, Elisabetta Semboloni, Sanaz Vafaei,

and Malin Velander. CFHTLenS: combined probe cosmological model comparison using 2D weak gravitational lensing. *Monthly Notices of the Royal Astronomical Society*, **430**(3):2200–2220, 2013. xxi, 97, 98

[51] Volodymyr V. Kindratenko, Adam D. Myers, and Robert J. Brunner. Implementation of the two-point angular correlation function on a high-performance reconfigurable computer. *Sci. Program.*, **17**:247–259, August 2009. 99

[52] S. D. Landy and A. S. Szalay. Bias and variance of angular correlation functions. *American Journal of Physics*, **412**:64–71, July 1993. 96

[53] E. Laure, S.M. Fisher, A. Frohner, C. Grandi, and P. Kunszt. Programming the grid with glite. *Computational Methods in Science and Technology*, **12**(1):33–45, 2006. 10

[54] R. Laureijs, J. Amiaux, S. Arduini, J.-L. Augueres, J. Brinchmann, et al. Euclid Definition Study Report. *ESA report*, 2011. 95

[55] Thé Van Luong, Nouredine Melab, and El-Ghazali Talbi. GPU-based island model for evolutionary algorithms. In *Genetic and Evolutionary Computation Conference, GECCO*, pages 1089–1096. ACM, 2010. 42

[56] Oden Maron and Andrew W. Moore. The racing algorithm: Model selection for lazy learners. *Artif. Intell. Rev.*, **11**(1-5):193–225, February 1997. 69

[57] I. Marquez et al. Rotation curves and metallicity gradients from HII regions in spiral galaxies. *Astron. Astrophys.*, **393**:389–408, 2002. 78

[58] Abílio Mateus, Laerte Sodré, Roberto Cid Fernandes, Grazyna Stasińska, William Schoenell, and Jean M. Gomes. Semi-empirical analysis of sloan digital sky survey galaxies – ii. the bimodality of the galaxy population revisited. *Monthly Notices of the Royal Astronomical Society*, **370**(2):721–737, 2006. 68

[59] Makoto Matsumoto and Takuji Nishimura. Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Trans. Model. Comput. Simul.*, **8**(1):3–30, 1998. 8, 44, 77, 81

[60] R.E. Mercer and J.R. Sampson. Adaptive search using a reproductive meta-plan. *Kybernetes*, **7**:215–228, 1978. 69

[61] Mark M. Meysenburg and James A. Foster. The quality of pseudo-random number generations and simple genetic algorithm performance. In Thomas Bäck, editor, *Proceedings of the 7th International Conference on Genetic Algorithms, ICGA*, pages 276–282. Morgan Kaufmann, 1997. 15

[62] Mark M. Meysenburg and James A. Foster. Randomness and GA performance, revisited. In Wolfgang Banzhaf, Jason Daida, Agoston E. Eiben, Max H. Garzon, Vasant Honavar, Mark Jakiela, and Robert E. Smith, editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, **1**, pages 425–432, Orlando, Florida, USA, 13-17 July 1999. Morgan Kaufmann. 16

[63] Mark Mathew Meysenburg, James Foster, Gene Saghi, John Dickinson, Richard T Jacobsen, and Jean'ne M. Shreeve. *The Effect of Pseudo-Random Number Generator Quality on the Performance of a Simple Genetic Algorithm*. Master's thesis, University of Idaho, Idaho, 1997. 15

[64] Efrén Mezura-Montes, Jesús Velázquez-Reyes, and Carlos A. Coello Coello. A comparative study of differential evolution variants for global optimization. In *GECCO, Genetic and Evolutionary Computation Conference*, pages 485–492. ACM, 2006. 81

[65] L. Miller, C. Heymans, T. D. Kitching, L. van Waerbeke, T. Erben, H. Hildebrandt, H. Hoekstra, Y. Mellier, B. T. P. Rowe, J. Coupon, J. P. Dietrich, L. Fu, J. Harnois-Draps, M. J. Hudson, M. Kilbinger, K. Kuijken, T. Schrabback, E. Semboloni, S. Vafaei, and M. Velander. Bayesian galaxy shape measurement for weak lensing surveys – III. application to the Canada–France–Hawaii telescope lensing survey. *Monthly Notices of the Royal Astronomical Society*, **429**(4):2858–2880, 2013. 116

[66] M. Mollá, M. L. García-Vargas, and A. Bressan. PopStar I: evolutionary synthesis model description. *Monthly Notices of the Royal Astronomical Society*, **398**:451–470, September 2009. 86

[67] D.C. Montgomery and G.C. Runger. *Applied Statistics and Probability for Engineers*. John Wiley and Sons Ltd, New York, USA, 2002. 72

[68] Andrew W. Moore, Andy J. Connolly, Chris Genovese, Alex Gray, Larry Grone, Nick Kanidoris II, Robert C. Nichol, Jeff Schneider, Alex S. Szalay, Istvan Szapudi, and Larry Wasserman. Fast algorithms and efficient statistics: N-point correlation functions. In AnthonyJ. Banday,

Saleem Zaroubi, and Matthias Bartelmann, editors, *Mining the Sky*, ESO ASTROPHYSICS SYMPOSIA, pages 71–82. Springer Berlin Heidelberg, 2001. 99, 100

[69] Luca Mussi, Fabio Daolio, and Stefano Cagnoni. Evaluation of parallel particle swarm optimization algorithms within the CUDA architecture. *Inf. Sci.*, **181**(20):4642–4657, 2011. 41

[70] Luca Mussi, Youssef S. G. Nashed, and Stefano Cagnoni. GPU-based asynchronous particle swarm optimization. In *13th Annual Genetic and Evolutionary Computation Conference, GECCO*, pages 1555–1562. ACM, 2011. 41

[71] Cedric Nugteren, Gert-Jan van den Braak, Henk Corporaal, and Bart Mesman. High performance predictable histogramming on GPUs: exploring and evaluating algorithm trade-offs. In *Proceedings of 4th Workshop on General Purpose Processing on Graphics Processing Units, GPGPU*, page 1. ACM, 2011. 101, 102

[72] Ender Ozcan, Selmanipak Cad, Tophanelioglu Sok No, and Chilukuri K. Mohan. Particle swarm optimization: Surfing the waves. In *Proceedings of the Congress on Evolutionary Computation*, pages 6–9. IEEE Press, 1999. 16, 31

[73] Magnus Erik Hvass Pedersen. Good Parameters for Differential Evolution. Technical Report no. HL1002, Hvass Laboratories, University of Zurich, Department of Informatics, 2010. 70

[74] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, **12**:2825–2830, 2011. 8

[75] T. Peram, K. Veeramachaneni, and C. K. Mohan. Fitness-distance-ratio based particle swarm optimization. In *Proc. IEEE Swarm Intelligence Symposium*, pages 174–181, April 2003. 16

[76] Victor Podlozhnyuk. Histogram calculation in CUDA. *NVIDIA Corporation White Paper*, 2007. 100, 101, 128

[77] R. Ponce, M. Cárdenas-Montes, J. J. Rodríguez-Vázquez, E. Sánchez, and I. Sevilla. Application of GPUs for the Calculation of Two Point Correlation Functions in Cosmology. In P. Ballester, D. Egret, and N. P. F. Lorente, editors, *Astronomical Data Analysis Software and Systems XXI*, **461** of *Astronomical Society of the Pacific Conference Series*, pages 73–76, September 2012. 101, 128

[78] Petr Pospíchal, Jirí Jaros, and Josef Schwarz. Parallel genetic algorithm on the CUDA architecture. In *Applications of Evolutionary Computation, EvoApplications (1)*, **6024** of *Lecture Notes in Computer Science*, pages 442–451. Springer, 2010. 41

[79] Petr Pospíchal, Josef Schwarz, and Jirí Jaros. Parallel genetic algorithm solving 0/1 knapsack problem running on the GPU. In *16th International Conference on Soft Computing MENDEL*, pages 64–70. Brno University of Technology, 2010. 41

[80] William Press, Brian Flannery, Saul Teukolsky, and William Vetterling. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, 1992. 9, 72

[81] K. V. Price, R. Storn, and J. Lampinen. *Differential Evolution: A practical Approach to Global Optimization*. Springer-Verlag, Berlin, Germany, 2005. 80, 81

[82] Mikhail Rabinovich, Phillip Kainga, David Johnson, Brandon Shafer, Jaehwan John Lee, and Rusell Eberhart. Particle swarm optimization on a GPU. In *IEEE International Conference on Electro/Information Technology, EIT*, pages 1–6. IEEE, 2012. 41

[83] J. Riget and J.S. Vesterstroem. A diversity-guided particle swarm optimizer - the ARPSO. Technical Report no. 2002-02, Aarhus Universitet, 2002. 16

[84] Dylan W. Roeh, Volodymyr V. Kindratenko, and Robert J. Brunner. Accelerating cosmological data analysis with graphics processors. In *Proceedings of 2nd Workshop on General Purpose Processing on Graphics Processing Units*, GPGPU-2, pages 1–8, New York, NY, USA, 2009. ACM. 99, 103

[85] Jason Sanders and Edward Kandrot. *CUDA by Example: An Introduction to General-Purpose GPU Programming*. Addison-Wesley Professional, 1 edition, July 2010. 101, 104, 115, 128

[86] Thorsten Scheuermann and Justin Hensley. Efficient histogram generation using scattering on GPUs. In *Proceedings of the 2007 symposium on Interactive 3D graphics and games*, I3D '07, pages 33–37, New York, NY, USA, 2007. ACM. 101

[87] R. Shams and R. A. Kennedy. Efficient histogram algorithms for NVIDIA CUDA compatible devices. In *Proc. Int. Conf. on Signal Processing and Communications Systems (ICSPCS)*, pages 418–422, Gold Coast, Australia, December 2007. 101, 102

[88] Yun-Wei Shang and Yu-Huang Qiu. A note on the extended rosenbrock function. *Evolutionary Computation*, **14**(1):119–126, 2006. 55

[89] D. Sheskin. *Handbook of parametric and nonparametric statistical procedures*. CRC Press, fifth edition, 2011. 7

[90] L. Smarr and C. Catlett. Metacomputing. *Communication of the ACM*, **35**:44–52, 1992. 10

[91] Selmar K. Smit and A. E. Eiben. Comparing Parameter Tuning Methods for Evolutionary Algorithms. In *IEEE Congress on Evolutionary Computation (CEC)*, pages 399–406, May 2009. 70

[92] Rainer Storn and K. V. Price. Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. *J. of Global Optimization*, **11**(4):341–359, 1997. 80, 81

[93] István Szapudi and Alexander S. Szalay. A New Class of Estimators for the N-Point Correlations. *The Astrophysical Journal Letters*, **494**(1):41–44, 1998. 97

[94] K. Tang, X. Yao, P. N. Suganthan, C. MacNish, Y. P. Chen, C. M. Chen, and Z. Yang. Benchmark functions for the CEC 2008 special session and competition on large scale global optimization. Technical report, Nature Inspired Computation and Applications Laboratory, USTC, China, 2007. 44, 55

[95] Ke Tang, Xiaodong Li, Ponnuthurai N. Suganthan, Zhenyu Yang, and Thomas Weise. Benchmark functions for the CEC'2010 special session and competition on large-scale global optimization. Technical report, Nature Inspired Computation and Applications Laboratory (NICAL), School of Computer Science and Technology, University of Science and Technology of China (USTC), Electric

Building No. 2, Room 504, West Campus, Huangshan Road, Hefei 230027, Anhui, China, 2009. 44, 55

[96] Pablo Vidal and Enrique Alba. Cellular genetic algorithm on graphic processing units. In *Nature Inspired Cooperative Strategies for Optimization, NICSO*, **284** of *Studies in Computational Intelligence*, pages 223–232. Springer, 2010. 41

[97] C. J. Walcher, B. Groves, T. Budavari, and D. Dale. Fitting the integrated Spectral Energy Distributions of Galaxies. *Astrophysics and Space Science*, **331**(1):1–51, 2011. 68

[98] D. Wells. *Prime numbers: the most mysterious figures in math*. Wiley, 1992. 16, 36

[99] Nathan Whitehead and Alex Fit-Florea. Precision and Performance: Floating Point and IEEE 754 Compliance of NVIDIA GPUs. Technical report, NVIDIA, 2011. 102, 127

[100] L. Darrell Whitley, Keith E. Mathias, Soraya B. Rana, and John Dzubera. Building better test functions. In *Proceedings of the 6th International Conference on Genetic Algorithms, ICGA*, pages 239–247. Morgan Kaufmann, 1995. 55, 56

[101] Xiao-Feng Xie, Wen-Jun Zhang, and Zhi-Lian Yang. Hybrid particle swarm optimizer with mass extinction. In *Communications, Circuits and Systems and West Sino Expositions, IEEE 2002 International Conference on*, **2**, pages 1170–1173 vol.2, June 2002. 16

[102] B. Yuan and M. Gallagher. Combining Meta-EAs and Racing for Difficult EA Parameter Tuning Tasks. In F.G. Lobo, C.F. Lima, and Z. Michalewicz, editors, *Parameter Setting in Evolutionary Algorithms*, pages 121–142. Springer, 2007. 69

[103] You Zhou and Ying Tan. GPU-based parallel particle swarm optimization. In *IEEE Congress on Evolutionary Computation, CEC*, pages 1493–1500. IEEE, 2009. 41

[104] You Zhou and Ying Tan. Particle swarm optimization with triggered mutation and its implementation based on GPU. In *Genetic and Evolutionary Computation Conference, GECCO*, pages 1–8. ACM, 2010. 41