GPU-Based Shear-Shear Correlation Calculation

Miguel Cárdenas-Montes[a], Miguel A. Vega-Rodríguez[b], Christopher Bonnett[c], Ignacio Sevilla Noarbe[a], Rafael Ponce[a], Eusebio Sánchez Alvaro[a], Juan José Rodríguez-Vázquez[a]

[a] *CIEMAT, Department of Fundamental Research, Avda. Complutense 40, 28040, Madrid, Spain.*
[b] *University of Extremadura, ARCO Research Group, Department Technologies of Computers and Communications, Escuela Politécnica, Campus Universitario s/n, 10003, Cáceres, Spain.*
[c] *Institut de Ciencies de lEspai, CSIC/IEEC, F. de Ciencies, Torre C5 par-2, Barcelona 08193, Spain*

Abstract

Light rays are deflected when traveling through a gravitational potential, this phenomenon is known as gravitational lensing. This causes the observed shapes of distant galaxies to be very slightly distorted by the intervening matter in the Universe, as their light travels towards us. This distortion is called *cosmic shear*. By measuring this component it is possible to derive the properties of the mass distribution causing the distortion. This in turn can lead to the measurement of the accelerated expansion of the Universe, as matter clumps together differently depending on its dynamics at each cosmological epoch. The measurement of the cosmic shear requires the statistical analysis of the ellipticities of millions of galaxies using very large astronomical surveys. In the past, due to the computational cost of the problem, this kind of analysis was performed by introducing simplifications in the estimation of such statistics. With the advent of scientific computing using Graphics Processing Units, the shear analysis without approximations can be addressed, even for very large surveys, while maintaining an affordable execution time. In this work we present the creation and optimization of such a Graphics Processing Unit code to compute the so-called shear-shear correlation function.

Keywords: Gravitational Weak Lensing, Shear-Shear Correlation Function, GPU Computing, Heterogeneous Computing, Optimization

1. Introduction

Current cosmological observations focus on surveying very large and deep regions of the sky, in order to be able to study the large scale structure of the Universe and its evolution.

Several observational probes [Albrecht et al., 2006] have been identified to tackle the study of the accelerated expansion of the Universe [Frieman et al., 2008]. Among them, one of the most promising turns out to be the analysis of the small deflections that large masses produce on the light traveling from distant galaxies. This phenomenon is known as gravitational lensing. Given the very small distortions involved, usually it is the statistical properties of the observed distribution of galaxy shapes which are studied [Fu et al., 2008]. The overall effect created by the gravitational lenses on these shapes is the *cosmic shear*. For many years this observational technique has been burdened by very large instrumental errors. Nevertheless first results were possible in the late 90s [Bacon et al., 2000], [Kaiser et al., 2000], [Van Waerbeke et al., 2000], [Wittman et al., 2000]. Only recently the first measurements in wide areas have been carried out delivering promising results for the future of the field [Heymans et al., 2012]. This has paved the way for present and future surveys, such as the Dark Energy Survey [Dark Energy Survey, 2005], the Kilo-Degree Survey [de Jong et al., 2012] and Euclid [Amendola et al., 2012], [Laureijs et al., 2011], to exploit this observational channel by increasing statistics by almost two orders of magnitude by the next decade as well as reducing systematic errors.

In this context cosmologists will have to deal with very large amounts of data (10^8 objects) to extract these measurements. In particular, the so-called shear-shear correlation function estimation requires computing the auto- and cross- correlation functions of the ellipticities of galaxies in different samples at varying redshifts. This algorithm has a high computational cost which goes with $O(N^2)$ in the case that one wants to achieve the best precision.

The calculation of correlation function estimators has already been addressed in the case of large scale structure studies, for the computation of the auto-correlation function of *positions* of galaxies and clusters (instead of shapes). In this case systematic errors play a lesser role but the probe in itself is less suited for determining cosmological parameters. Several codes have harnessed the computational power of Graphics Processing Units (GPUs) and other hardware platforms to carry out the job (Section 2). In the case of shear-shear correlations (involving the shape of the galaxies, Section 3)

several codes have been implemented using the kd-trees approach which simplifies the problem at the cost of precision (Section 2).

To the authors' knowledge, up to now, no code has taken advantage of the capabilities of GPUs to handle this specific problem. This calculation is of particular relevance in the short term with the rapid advent of larger datasets in the next few years. GPUs are able to deal with the shear correlation calculation of very large surveys of galaxies within a reasonable execution time.

Beyond the initial adaptation of the problem to the GPU platform and the subsequent verification of the results, the code passes through a series of optimization processes. Among other techniques, a more efficient use of on-chip memory with an increment of data locality, and the study of the compilation options modifying the use of the cache memory are checked. Finally a concurrent computing scenario (hybrid OpenMP-CUDA implementation) is presented.

This paper is organized as follows: Section 2 summarizes the related work and previous efforts. In Section 3.1 the concepts of gravitational lensing and shear are described, together with the equations to be implemented in the code. The statistical support to the analysis is described in Section 3.2. The hardware used in this work is presented in Section 3.3. Results are presented and analysed in Section 4. Finally, Section 5 contains the conclusions of this work.

2. Related Work

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 [Moore et al., 2000], 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 ex-

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

pense of a higher execution time (see Section 4 for a quantification of this effect). A similar approach is used in [Jarvis et al., 2004].

More generally, GPU computation is becoming increasingly used in Cosmology to deal with the analysis of the large scale structure of the Universe. Some examples include: the two-point angular correlation function [Ponce et al., 2012], [Roeh et al., 2009] and the aperture mass statistic (in which a filter is applied to shear maps, in order to detect mass structures) [Bard et al., 2012]. However, to the best of our knowledge, we present the first GPU-based shear-shear correlation function estimator implementation.

3. Methods and Materials

3.1. 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 [Bartelmann and Schneider, 2001]. The value of the shear field γ can be conveniently estimated from the ellipticity, e, of a particular galaxy. Here $|e|$ is defined as such that an ellipse with axes $a < b$:

$$|\epsilon| = \frac{a-b}{a+b}. \tag{1}$$

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

$$\epsilon = \epsilon_x + i\epsilon_y = |\epsilon| e^{2i\phi} \tag{2}$$

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 [Kilbinger et al., 2013] 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 the scope of this paper. The reader can consult [Bernstein and Jarvis, 2002] and [Hoekstra and Jain, 2008] for a detailed explanation of

systematic effects and the steps for the extraction of shear from observations. In the rest of the paper 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.

$$\vec{v_i} = (\cos\alpha_i \cdot \cos\delta_i, \sin\alpha_i \cdot \cos\delta_i, \sin\delta_i) \tag{3}$$

$$\cos(\theta) = \vec{v_1} \cdot \vec{v_2} \tag{4}$$

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 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:

$$\begin{aligned} \cos\Phi_1 &= \frac{\sin(\alpha_2 - \alpha_1)\cos\delta_2}{\sin\theta} \\ \sin\Phi_1 &= \frac{\cos\delta_2\sin\delta_1 - \sin\delta_2\cos\delta_1\cos(\alpha_2 - \alpha_1)}{\sin\theta} \end{aligned} \tag{5}$$

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:

$$\begin{aligned} \xi_+(\theta) &= \frac{\sum_{ij} w_i w_j(\gamma_t(\theta_i)\cdot\gamma_t(\theta_j) + \gamma_\times(\theta_i)\cdot\gamma_\times(\theta_j))}{\sum_{ij} w_i w_j} \\ \xi_-(\theta) &= \frac{\sum_{ij} w_i w_j(\gamma_t(\theta_i)\cdot\gamma_t(\theta_j) - \gamma_\times(\theta_i)\cdot\gamma_\times(\theta_j))}{\sum_{ij} w_i w_j} \\ \xi_\times(\theta) &= \frac{\sum_{ij} w_i w_j(\gamma_t(\theta_i)\cdot\gamma_\times(\theta_j))}{\sum_{ij} w_i w_j} \end{aligned} \tag{6}$$

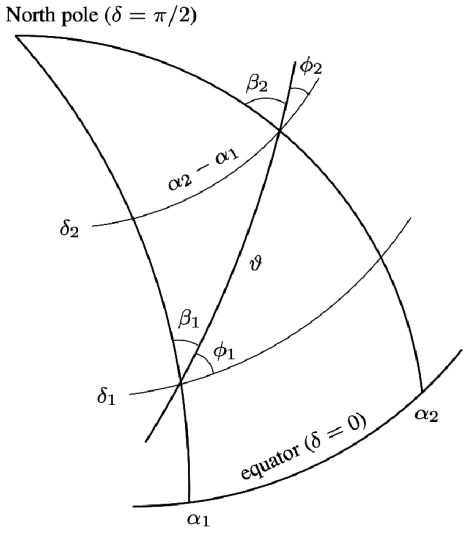


Figure 1: Angles and coordinates on a sphere for two galaxies i = (1,2) located at (α_i,δ_i). Figure taken from [Kilbinger et al., 2013], used with the author's permission.

where $w_{i,j}$ are the weights associated to the measurement of galaxy ellipticity. This takes into account measurement errors, see [Hoekstra et al., 2002]. These are the three correlation functions that are calculated by the code.

3.2. Statistics

In order to ascertain if the proposed modifications applied to the code improve the execution time, two different types of tests can be applied: parametric and non-parametric. The difference between both relies on the assumption that data is normally distributed for parametric tests, whereas non explicit conditions are assumed in non-parametric tests. For this reason, the latter is recommended when the statistical model of data is unknown [García et al., 2009B].

The Kruskal-Wallis test [Sheskin, 2004] 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.

The procedure when using multiple comparison (e.g. Kruskal-Wallis) to test when the null hypothesis is rejected implies the use of 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 [García et al., 2009].

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, that is, the behaviour —execution time in our study— of two codes before and after the modification.

On the other hand, the Bonferroni correction aims to control the Family-Wise Error Rate (FWER). 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.

3.3. Hardware

The creation and optimization process of the code, as well as the numerical experiments have been executed on an NVIDIA C2075 (Fermi architecture, see Appendix A). Otherwise, the CPU numerical experiments are executed on a computer with two Intel Xeon X5570 processors at 2.93 GHz and 8 GB of RAM.

4. Results and Analysis

4.1. Baseline Implementation

In Appendix A an overview of GPU Architecture and the CUDA programming model is presented. In the following section the CUDA baseline implementation of the shear-shear correlation code is described, while mentioning the technical aspects that affect the performance.

4.1.1. General Description of the Program Flow

The code consists in the calculation of the quantities: ξ_+, ξ_-, $\xi_\times$ (Eq. 6, 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.

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) and the sum of weights of all the pairs. For the calculation of the values and in order to avoid slow

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. 5);
        (angle with respect to line of equal declination) ;
        Calculate ξ+, ξ−, ξ× (Eq. 6) 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.
```

access to global memory, intermediate reusable values are stored on shared memory.

4.1.2. 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[2]. 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 led to the usage of atomic functions[3] 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 diminishing 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 partial-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 shared memory but their size is smaller. Data frequently accessed[4] 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 per-

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

[3]The atomic operation for float on shared memory is supported for compute capability 1.3 and higher [Sanders and Kandrot, 2010].

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

formed 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%. This is a limiting factor due to the double precision requirement[5].

Due to the fact that the correlation function needs to be studied at small separation scales double precision is required. 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.

4.1.3. Comparison with Athena Input Reference

For comparison purposes, ξ_+, ξ_- and $\xi_\times$ obtained with the GPU implementation and *ATHENA* version 1.54 are plotted in Fig. 2. 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.

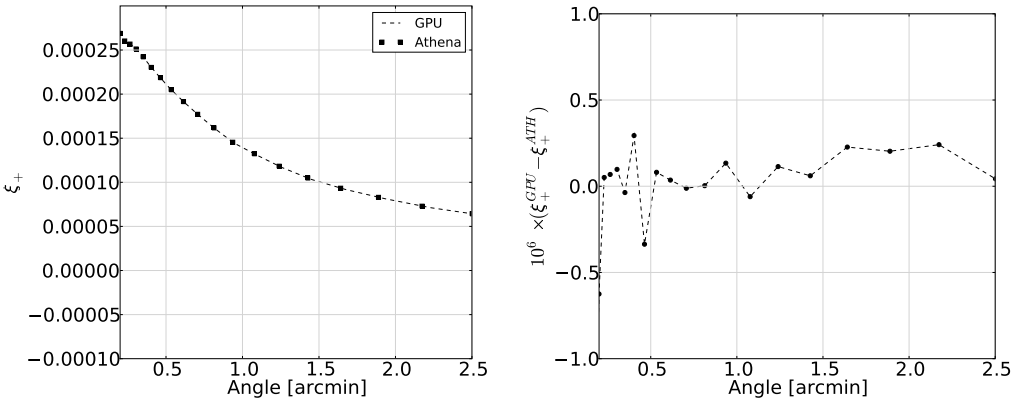
4.1.4. Comparison with 1 Million Galaxies Input Reference

Data from the Canada-France-Hawaii Lensing Survey [Heymans et al., 2012] is used, hereafter referred to as CFHTLenS. The CFHTLenS survey analysis combined weak lensing data processing with THELI [Erben et al., 2012], shear measurement with lensfit [Miller et al., 2013] and photometric redshift measurement with PSF-matched photometry [Hildebrandt et al., 2012]. A full systematic error analysis of the shear measurements in combination with the photometric redshifts is presented in [Heymans et al., 2012], with additional error analyses of the photometric redshift measurements presented in [Benjamin et al., 2013].

A query was done on the CFHTLenS catalogue query page[6] for right as-

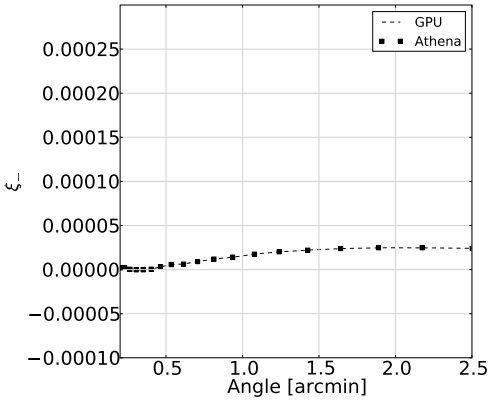
[5]When implementing all operations in single precision the execution time is significantly reduced. However, doing this prevents the computation of the shear-shear correlation at small angles (less than 0.1 arcminutes in our case). This is due to the lack of precision to cope with very small numbers.

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

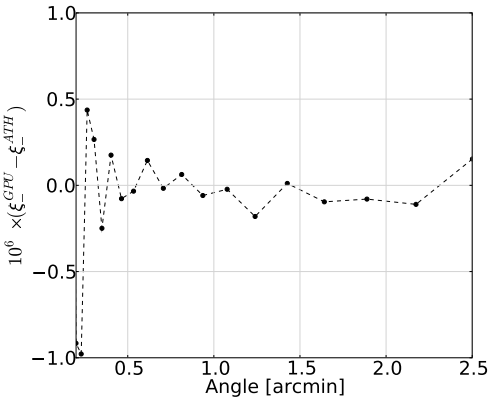


(a) ξ_+

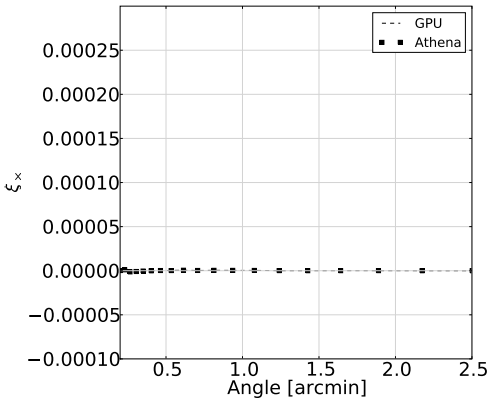
(b) $10^6 \times (\xi_+^{GPU} - \xi_+^{ATH})$



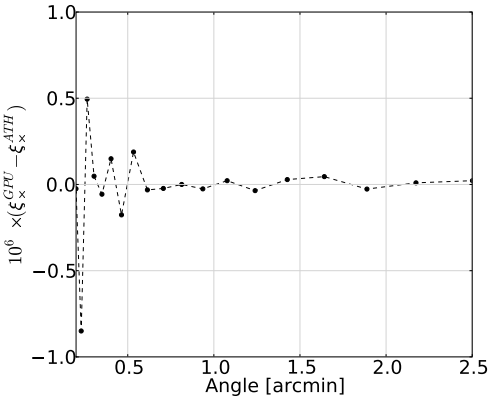
(c) ξ_-



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



(e) $\xi_\times$



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

Figure 2: Comparison of the results obtained with the GPU implementation and *ATHENA* v1.54 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) $10^6 \times (\xi_\times^{GPU} - \xi_\times^{ATH})$.

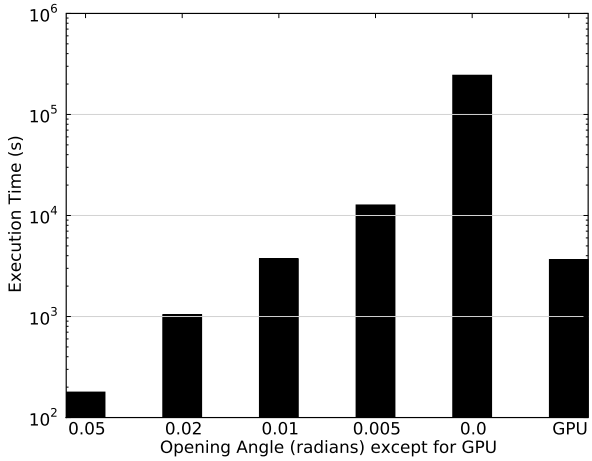


Figure 3: Mean execution time (s) for *ATHENA* code for various opening angles (radians) and GPU code. The execution time of the GPU code is roughly equivalent to the Athena code for an opening angle of 0.01 radians. For the same precision ('brute-force') the GPU implementation is a factor 68 faster than a CPU-based code such as *ATHENA*.

cension, 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 was tuned to obtain values for ξ_+, ξ_-, $\xi_\times$ at the exact same angle separation values θ. The results for execution time are shown in Fig. 3.

The GPU implementation takes $3650.0 \pm 1.4\,s$ to analyse the 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: $3723.3 \pm 8.4\,s$, (see Figure 3).

When the opening angle approaches zero radians the code makes fewer

approximations and becomes equivalent to a brute force method. Unfortunately the reduction of the opening angle leads to a critical increment in the execution time. For OA = 0.005 radians the execution time, $12688.7 \pm 122.7\, s$, this is 3.5 times slower than the GPU processing time. Finally for OA equal to zero the execution time increases to $247681\, s$ this is a factor 68 slower than the GPU code execution time.

Concerning precision, Figure 4 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 [Eifler et al., 2013]. 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.

4.2. Code Optimization

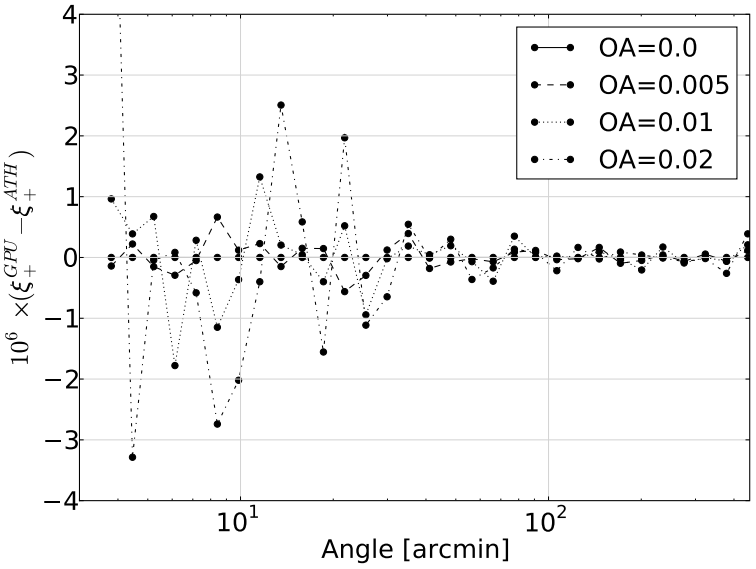
4.2.1. L1 Memory Optimization

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 was performed.

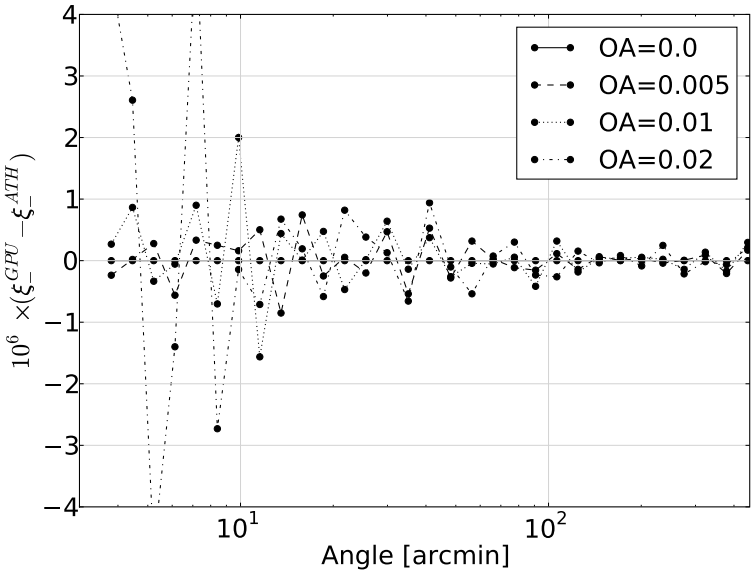
The most successful test corresponds to the modification of the L1 configuration. The Fermi architecture of the GPU distributes 64 KB between shared memory and L1 cache memory. Three configurations are possible: the default configuration with 48KB of shared memory and 16 KB of L1 cache memory a second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



(a) Deviations GPU-Athena of computed ξ_+.



(b) Deviations GPU-Athena of computed ξ_-.

Figure 4: 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 1: 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%)**

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 were tested in order to check potential reductions in the execution time. The numerical 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 1).

The statistical analysis of the execution times (Table 1) 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). This means that the differences are unlikely to have occurred by chance with a probability of 95%.

4.3. Heterogeneous Computing

In 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 a part of the computational resources are underused because during the kernel execution as 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

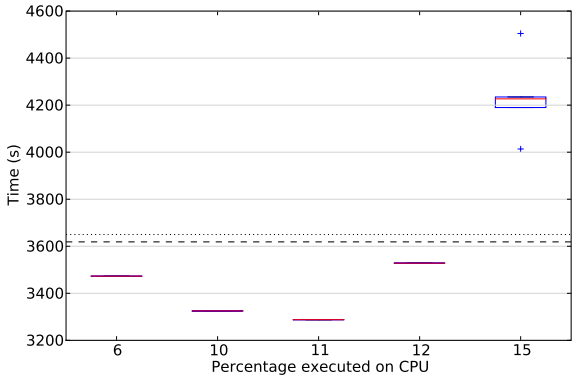


Figure 5: Execution time (s) for diverse CPU-processed percentages in the concurrent computing model. The dotted line is the reference to the baseline code execution time, whereas the dashed line is the execution time after L1 memory optimization.

Table 2: Mean execution time (s) for original code and 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%)**

take the same execution time, the choice of the chunk size is critical. Initially diverse chunk sizes were tested in order to select the most appropriate one for the reduction of the execution time (Fig. 5).

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 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 2), 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 significant also at a significance level of more than 95%. This result reinforces that the modifications applied produce a net improvement in the application productivity.

5. Conclusions

In this paper the first GPU code for the computation of the shear-shear correlation function is presented. In the past the computational cost of the problem has prevented a brute force implementation and approximations (kd-trees) were used aiming to reduce the execution time. By using GPU computing the shear-shear correlation function estimation without any simplifications can be achieved in a reasonable timescale. In this work an implementation is shown where a 68-fold improvement is reached with respect to the same algorithm running on a CPU while obtaining the same precision on the results (smaller than 0.001%). A kd-trees code taking the same amount of time by averaging the shears, will induce errors of the order of a few percent which are very relevant in the new era of increased statistics from very large surveys. A natural follow-up of this work is a Multi-GPU approach to be run on a GPU-farm to handle even larger datasets ($O(10^8)$). In addition to this an additional optimization process consisting of a GPU-CPU concurrent computing implementation achieves a reduction of 11% in execution time.

The code is being made available (http://wwwae.ciemat.es/cosmo/gp2pcf/) for cosmologists to integrate it within their portfolio of analysis codes.

Acknowledgements

The authors would like to thank the Spanish Ministry of Science and Innovation (MICINN) for funding support through grants AYA2009-13936-C06-03 and through the Consolider Ingenio-2010 program, under project CSD2007-00060.

CB is supported by the Spanish Science Ministry AYA2009-13936, Consolider-Ingenio CSD2007-00060, project 2009SGR1398 from Generalitat de Catalunya

and by the European Commission's Marie Curie Initial Training Network CosmoComp (PITN-GA-2009-238356).

IS would like to thank Tim Eifler for useful comments regarding the cosmology-related aspects of this work.

The CFHTLens data is based on observations obtained with MegaPrime/MegaCam, a joint project of CFHT and CEA/DAPNIA, at the Canada-France-Hawaii Telescope (CFHT) which is operated by the National Research Council (NRC) of Canada, the Institut National des Sciences de l'Univers of the Centre National de la Recherche Scientifique (CNRS) of France, and the University of Hawaii. This research used the facilities of the Canadian Astronomy Data Centre operated by the National Research Council of Canada with the support of the Canadian Space Agency. CFHTLenS data processing was made possible thanks to significant computing support from the NSERC Research Tools and Instruments grant program.

Appendix A. 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[7] 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. An other difference is that it is accessible by all the threads belonging to a block of threads[8].

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. 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

[7]Single Instruction, Multiple Data

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

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.

References

Albrecht, Andreas, Bernstein, Gary, Cahn, Robert, Freedman, Wendy L., Hewitt, Jacqueline, Hu, Wayne, Huth, John, Kamionkowski, Marc, Kolb, Edward W., Knox, Lloyd, Mather, John C., Staggs, Suzanne, and Suntzeff, Nicholas B.: Report of the Dark Energy Task Force, arXiv:astro-ph/0609591, 2006

Amendola,L. et al.: Cosmology and Fundamental Physics with the Euclid Satellite, arXiv:astro-ph/1206.1225, 2012

Bacon, David, Refregier, Alexandre, and Ellis, Richard: Detection of Weak Gravitational Lensing by Large-scale Structure, Monthly Notices of the Royal Astronomical Society 318(2) 625-640, 2000

Bard, Deborah, Bellis, Matthew, Allen, Mark T., Yepremyan, Hasmik, and Kratochvil, Jan M.: Cosmological Calculations on the GPU, arXiv:astro-ph/1208.3658, 2012

Bartelmann, Matthias, Schneider, Peter: Weak gravitational lensing, Physics Reports, Volume 340, Issues 4-5, pp. 291-472, 2001

Benjamin, Jonathan; Van Waerbeke, Ludovic; Heymans, Catherine; Kilbinger, Martin; Erben, Thomas; Hildebrandt, Hendrik; Hoekstra, Henk; Kitching, Thomas D.; Mellier, Yannick; Miller, Lance; Rowe, Barnaby; Schrabback, Tim; Simpson, Fergus; Coupon, Jean; Fu, Liping; Harnois-Déraps, Joachim; Hudson, Michael J.; Kuijken, Konrad; Semboloni, Elisabetta; Vafaei, Sanaz; Velander, Malin: CFHTLenS tomographic weak lensing: quantifying accurate redshift distributions, Monthly Notices of the Royal Astronomical Society, Volume 431, pp. 1547-1564, 2013

Bernstein, G. and Jarvis, M.: Shapes and Shears, Stars and Smears: Optimal Measurements for Weak Lensing, The Astronomical Journal, Volume 123, Issue 2, pp. 583-618, 2002

The Dark Energy Survey Collaboration: The Dark Energy Survey, arXiv:astro-ph/0510346, 2005

Eifler, Tim et al. In preparation (2013)

Erben, T.; Hildebrandt, H.; Miller, L.; van Waerbeke, L.; Heymans, C.; Hoekstra, H.; Kitching, T. D.; Mellier, Y.; Benjamin, J.; Blake, C.; Bonnett, C.; Cordes, O.; Coupon, J.; Fu, L.; Gavazzi, R.; Gillis, B.; Grocutt, E.; Gwyn, S. D. J.; Holhjem, K.; Hudson, M. J.; Kilbinger, M.; Kuijken, K.; Milkeraitis, M.; Rowe, B. T. P.; Schrabback, T.; Semboloni, E.; Simon, P.; Smit, M.; Toader, O.; Vafaei, S.; van Uitert, E.; Velander, M.: CFHTLenS: The Canada-France-Hawaii Telescope Lensing Survey - Imaging Data and Catalogue Products, arXiv:astro-ph/1210.8156, 2012

Frieman, Joshua, Turner, Michael, and Huterer, Dragan: Dark Energy and the Accelerating Universe, Annual Review of Astronomy and Astrophysics, Vol. 46, 385-432, 2008

Fu, L., Semboloni, E., Hoekst

García, Salvador, Fernández, Alberto, Luengo, Julián, and Herrera, Francisco: A study of statistical techniques and performance measures for genetics-based machine learning: accuracy and interpretability, Soft Comput. 13(10), 959-977, 2009

García, Salvador, Molina, Daniel, Lozano, Manuel, and Herrera, Francisco: 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

Heymans, C., Van Waerbeke, L., Miller, L., Erben, T., Hildebrt, H., Hoekstra, H., Kitching, T. D., Mellier, Y., Simon, P., Bonnett, C., Coupon, J., Fu, L., Harnois Déraps, J., Hudson, M. J., Kilbinger, M., Kuijken, K., Rowe, B., Schrabback, T., Semboloni, E., van Uitert, E., Vafaei, S., Veler, M.:CFHTLenS: the Canada-France-Hawaii Telescope Lensing Survey, Monthly Notices of the Royal Astronomical Society, Volume 427, Issue 1, pp. 146-166, 2012

Hildebrandt, H.; Erben, T.; Kuijken, K.; van Waerbeke, L.; Heymans, C.; Coupon, J.; Benjamin, J.; Bonnett, C.; Fu, L.; Hoekstra, H.; Kitching,

T. D.; Mellier, Y.; Miller, L.; Velander, M.; Hudson, M. J.; Rowe, B. T. P.; Schrabback, T.; Semboloni, E.; Benítez, N. : CFHTLenS: improving the quality of photometric redshifts with precision photometry, Monthly Notices of the Royal Astronomical Society, Volume 421, Issue 3, pp. 2355-2367

Hoekstra, H., Franx, M., Kuijken, K., VanDokkum,P. G., Monthly Notices of the Royal Astronomical Society, 333, 911-922, 2002

Hoekstra, Henk and Jain, Bhuvnesh: Weak Gravitational Lensing and its Cosmological Applications, Ann.Rev.Nucl.Part.Sci., 58, 99-123, 2008

Jarvis, Mike, Bernstein, Gary and Jain, Bhuvnesh: The Skewness of the Aperture Mass Statistic, Monthly Notices of the Royal Astronomical Society 352, 338-352, 2004

de Jong, Jelte T. A., Verdoes Kleijn, Gijs A., Kuijken, Konrad H., Valentijn, Edwin A.: The Kilo-Degree Survey, arXiv:astro-ph/1206.1254, 2012

Kaiser, Nick, Wilson, Gillian, and Luppino, Gerard A.: Large-Scale Cosmic Shear Measurements, arXiv:astro-ph/0003338, 2000

Kilbinger, Martin; Fu, Liping; Heymans, Catherine; Simpson, Fergus; Benjamin, Jonathan; Erben, Thomas; Harnois-Déraps, Joachim; Hoekstra, Henk; Hildebrandt, Hendrik; Kitching, Thomas D.; Mellier, Yannick; Miller, Lance; Van Waerbeke, Ludovic; Benabed, Karim; Bonnett, Christopher; Coupon, Jean; Hudson, Michael J.; Kuijken, Konrad; Rowe, Barnaby; Schrabback, Tim; Semboloni, Elisabetta; Vafaei, Sanaz; Velander, Malin: CFHTLenS: combined probe cosmological model comparison using 2D weak gravitational lensing, Monthly Notices of the Royal Astronomical Society, Volume 430, Issue 3, p.2200-2220, 2013

Laureijs, R. et al: Euclid Definition Study Report, Report number: ESA/SRE(2011)12, arXiv:astro-ph/1110.3193, 2011

Miller, L.; Heymans, C.; Kitching, T. D.; van Waerbeke, L.; Erben, T.; Hildebrandt, H.; Hoekstra, H.; Mellier, Y.; Rowe, B. T. P.; Coupon, J.; Dietrich, J. P.; Fu, L.; Harnois-Déraps, J.; Hudson, M. J.; Kilbinger, M.; Kuijken, K.; Schrabback, T.; Semboloni, E.; Vafaei, S.; Velander, M.: 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, Volume 429, Issue 4, p.2858-2880

Moore, Andrew, Connolly, Andy, Genovese, Chris, Gray, Alex, Grone, Larry, Kanidoris, Nick, Nichol, Robert, Schneider, Jeff, Szalay, Alex, Szapudi, Istvan, Wasserman, Larry: Fast Algorithms and Efficient Statistics: N-point Correlation Functions, ESO Astrophysics Symposia, 71-82, 2001

Ponce, R., Cárdenas-Montes, M., Rodríguez-Vázquez, J.J., Sanchez, E., and Sevilla, I.: Application of GPUs for the Calculation of Two Point Correlation Functions in Cosmology, Astronomical Data Analysis Software and Systems XXI. ASP Conference Series, Vol.461. Edited by P.Ballester, D.Egret and N.P.F.Lorente. Astronomical Society of the Pacific, 2012 p.73

Roeh, Dylan W., Kindratenko, Volodymyr V., and Brunner, Robert J.: Accelerating cosmological data analysis with graphics processors, Proceedings of 2nd Workshop on General Purpose Processing on Graphics Processing Units, ACM, 1-8, 2009

Sanders, Jason, and Kandrot, Edward: CUDA by Example: An Introduction to General-Purpose GPU Programming, Addison-Wesley Professional, July 2010

Sheskin, D.: Handbook of parametric and non-parametric statistical procedures, CRC Press, 2004

Van Waerbeke, L., Mellier, Y., Erben, T., Cuillandre, J. C., Bernardeau, F., Maoli, R., Bertin, E., Cracken, H. J. Mc, Le Fevre, O., Fort, B., Dantel-Fort, M., Jain, B., and Schneider, P.: Detection of correlated galaxy ellipticities on CFHT data: first evidence for gravitational lensing by large-scale structures, Astron. Astrophys. 358, 30-44, 2000

Wittman, David M., Tyson, J. Anthony, Kirkman, David, Dell'Antonio, Ian, and Bernstein, Gary: Detection of weak gravitational lensing distortions of distant galaxies by cosmic dark matter at large scales, Nature 405, 143-148, 2000