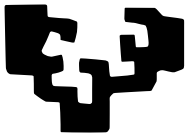
eman ta zabal zazu



Universidad del País Vasco
Euskal Herriko Unibertsitatea

Konputagailuen Arkitektura eta Tecnologa Saila
Departamento de Arquitectura y Tecnología de Computadores

Contributions to the Efficient Use of General Purpose Coprocessors: Kernel Density Estimation as Case Study

by

Unai Lopez Novoa

Supervised by Jose Miguel-Alonso and Alexander Mendiburu

Dissertation submitted to the Department of Computer Architecture and Technology of the University of the Basque Country (UPV/EHU) as partial fulfilment of the requirements for the PhD degree in Computer Science

Donostia - San Sebastián, June 2015

It's not about how much power you have, it's about how much power you can use!

James May.

Acknowledgments

This dissertation would not have been possible without the support of many people. The list is so long that I hope I do not to forget anybody. First of all, I want to acknowledge my advisors Alex and Jose, for their guidance during all of these years. Their encouragement and patience have been fundamental to finish this work. I want to also mention the invaluable support of Jon Sáenz in the part related to KDE and climatology.

I am grateful to the Basque Government for the financial support, and to the funding institutions. In addition, during this time I have been an affiliated PhD student member of the European Network of Excellence on High Performance and Embedded Architecture and Compilation (HIPEAC).

I want to acknowledge all present and past colleagues in the Intelligent Systems Group, with whom I have shared this phase: Carlos P., Jose, Josu, Eneko, Juan, Ekhiñe, Jonathan, Carlos E., Leti, Momo, Aritz, Usue, Ibai, Ari, Roberto, Borja, Iñaki and Jose A. L. I have learnt a lot from all of you, and I'm proud to consider you not only work mates, but also friends. I want to extend this acknowledgement to the PhD candidates in other research groups at the School of Computer Science of San Sebastian, who have made life in the faculty extraordinary. I want to specially acknowledge Borja Gamecho for all the *M-Index* conversations, Itzi for the great *Pisi* adventures, and Josune and Amparo for making every visit to Paco's inspiring and revitalizing.

As part of my PhD candidate years, I made an internship at the APT Group of the University of Manchester from May until August 2014. I want to thank Javier Navaridas, Mikel Lujan, and all of the members of APT and other groups in the School of Computer Science for making me feel at home (and for the Ducie Arms visits on Friday afternoons). I want to mention, as well, my Opal Gardens family (Nuria, Isabel, Cora, Rita, Inma, Uxue and Javi) - Without them, those months wouldn't have been so amazing.

I also want to mention the group of the *Lords*: Ander, Pablo, Andoni, Inazio and David, with whom I have been sharing more than a hobby (sometimes a lifestyle) for so many years. Thank you for all the trips, for the endless discussions about clubs, vinyls, and hard and soft compressors, and for all the unforgetable night life adventures.

There is also another group of people that have supported me over the last years: my colleagues at the University of Deusto. Thank you for always being there, waiting for me on the other side of the Basque Country. I want to specially mention two persons: Oihane, thank you for your unconditional support, for so many good and bad moments, and for the ones that are yet to come; and also Itsaso, thank you for being by my side during the last months of this work: your support has been essential to write the final words.

I want to devote these last lines to give the most relevant acknowledgement to my family, for making me proud of my surnames, and specially to my parents: *Gracias Javi y Maria por la educación que me habeís dado, por vuestra paciencia, por vuestros ánimos y apoyo incondicional, y por además, ser los mejores padres del mundo.*

Abstract

The high performance computing landscape is shifting from assemblies of homogeneous nodes towards heterogeneous systems, in which nodes consist of a combination of traditional out-of-order execution cores and accelerator devices. Accelerators, built around GPUs, many-core chips, or FPGAs, are used to offload compute-intensive tasks. These devices provide superior theoretical performance compared to traditional multi-core CPUs, but not every application fits into the programming model they impose, and exploiting their computing power remains a challenging task.

This dissertation discusses the issues that arise when trying to efficiently use general purpose accelerators. As a contribution to aid in this task, we present a thorough survey of performance modeling techniques and tools for general purpose coprocessors. Then we use as case study the statistical technique Kernel Density Estimation (KDE). KDE is a memory bound application that poses several challenges for its adaptation to the accelerator-based model. We present a novel algorithm for the computation of KDE that reduces considerably its computational complexity, called S-KDE. Furthermore, we have carried out two parallel implementations of S-KDE, one for multi and many-core processors, and another one for accelerators. The latter has been implemented in OpenCL in order to make it portable across a wide range of devices. We have evaluated the performance of each implementation of S-KDE in a variety of architectures, trying to highlight the bottlenecks and the limits that the code reaches in each device. Finally, we present an application of our S-KDE algorithm in the field of climatology: a novel methodology for the evaluation of environmental models.

Contents

Part I

Summary of Contributions

1

Introduction

High Performance Computing (HPC) is the branch of computer science related to the use of supercomputers and parallel architectures to solve complex computational problems. These problems are deployed as parallel applications that split their workload among the available computing resources in order to minimize running time. These applications are usually scientific and technical codes that try to solve problems from different research fields such as chemistry, earth sciences or bioinformatics [1].

Traditional HPC environments were built around single or multi-core Central Processing Units (CPUs), but in the last years these systems include general purpose coprocessors used as accelerators for offloading computationally intensive tasks. This shift has defined a change from homogeneous (just CPUs) to heterogeneous (CPUs and accelerators) systems. This is reflected in the last edition of Top500, the list that contains the 500 most powerful supercomputers in the world[1]. In the November 2014 edition, five machines out of the top ten were built as combinations of CPUs and general purpose coprocessors.

Accelerator devices are hardware pieces designed for the efficient computation of specific tasks or subroutines, and hold important architectural differences with respect to CPUs: the number of computing cores, the instruction sets and the memory hierarchy are completely different. Nowadays most common accelerators in HPC are Graphics Processing Units (GPUs) and many-core coprocessors. These devices usually work attached to a CPU which controls the offloading of software fragments and manages the copying and retrieval of the manipulated data [2].

The main strength of state-of-the-art coprocessors is the higher theoretical performance they provide compared to multi-core CPUs, e.g. recent GPUs and many-cores provide up to 1 TFLOP/s of theoretical double precision performance, while a 16-core AMD CPU can reach up to 250 GFLOP/s. However, effectively exploiting that performance remains a challenging task. Accelerators have been designed to work with data intensive applications with high data locality access, and not every code fits into this computing pattern. We can find highly successful cases of application porting [3], which may compel programmers to jump onto the accelerator bandwagon. However, metrics about the effort required for the porting are not that common. Besides, the way of measuring the degree of success can be misleading [4][5].

Modifying an application to use an accelerator means extensive program rewriting, only to achieve a preliminary, not really efficient implementation. This process often requires the use of new programming paradigms and tools that sometimes are not as polished as the ones for multi-core CPUs. In these cases, the optimization of the code relies almost entirely on the hands of the developer [6]. In addition, we can find a huge amount of literature about optimizing

[1] http://www.top500.org/

accelerator-based applications, but in many cases it can be messy, overwhelming the developer [7].

This dissertation discusses some of the issues that arise when trying to efficiently use general purpose coprocessors. In this process, we use as case study the statistical technique Kernel Density Estimation (KDE), which is a common building block in many HPC applications. The main motivation is to provide a piece of handy literature that aids developers when using modern coprocessors, including detailed performance evaluations in different scenarios.

First we present a deep survey that analyzes the literature on techniques for performance modeling on general purpose coprocessors. Second, we present the design of a novel algorithm to compute KDE called S-KDE. Then, we present the implementation and performance evaluation of S-KDE, first in multi-core and many-core processors, and then in a wider range of general purpose coprocessors. Last, we present an application of this improved KDE algorithm in the domain of environmental sciences.

1.1 Organization of this Dissertation

This dissertation consists of a compilation of research articles that describe the mentioned contributions. It is organized in two parts. Part I gives an introduction to the topic and presents a summary of the contributions in the following manner: Chapter 1 motivates the research and Chapter 2 provides a brief background on general purpose coprocessors and KDE. Chapter 3 is devoted to summarizing the contributions of the dissertation and Chapter 4 provides some global conclusions and a brief description of the publications included in this thesis. This part ends with a list of references.

Part II is composed of four Chapters (5 to 8), each one being a scientific paper published by the author of this dissertation. In these chapters, the contributions described in Part I are explained with a higher level of detail. Each Chapter (publication) includes its own list of references, which complements those given in Part I.

2

Background

In this chapter we provide the background and terminology used in this dissertation. First we briefly describe the hardware devices on which we have focused: general purpose coprocessors. Then we provide a description of the statistical technique Kernel Density Estimation (KDE), which is used as case study in several parts of this work.

2.1 General Purpose Coprocessors

This section is devoted to briefly describing the most common coprocessors used in HPC environments nowadays: Graphics Processing Units (GPUs), many-core processors and hybrid chips. We have depicted in Figure 2.1 a time line of the main hardware devices developed by different manufacturers, where a dashed line indicates the absence of a product family and a shortened line indicates the discontinuation of a product. We include as well a brief description of the development environments used for those devices.

2.1.1 Graphics Processing Units

GPUs are hardware devices designed for the efficient manipulation of computer images [8]. In the early 2000s, the HPC community began using GPUs as accelerators for general purpose computations [9], coining the term *General Purpose Computing on GPUs* (GPGPU) [10]. Since then, each new generation of devices arrives with significantly improved horsepower in terms of FLOP/s [11]. Currently the main GPU manufacturers developing HPC-class products are NVIDIA and AMD (who bought ATI in 2006). Figure 2.1 shows the evolution of their GPU product families. Both AMD and NVIDIA provide desktop and server versions of their cards. Desktop cards are marketed as graphic accelerators that can also be used for GPGPU, running punctual workloads at full performance. Server cards are marketed as GPU-based accelerators, designed to run intensive workloads in an uninterrupted way.

In essence, a GPU is an autonomous compute system composed of a set of processing cores and a memory hierarchy. A global memory space is accessible to all the cores, which can be exclusive (this is the common case if the GPU is a discrete accelerator plugged into a PCI-Express slot) or shared with other processing elements in the system, including the CPU (which is the common case when using a heterogeneous, multi-core chip). Discrete NVIDIA and AMD server-class accelerators (Tesla [12] and FireStream systems [13] respectively) include up to 8 GB of global memory.

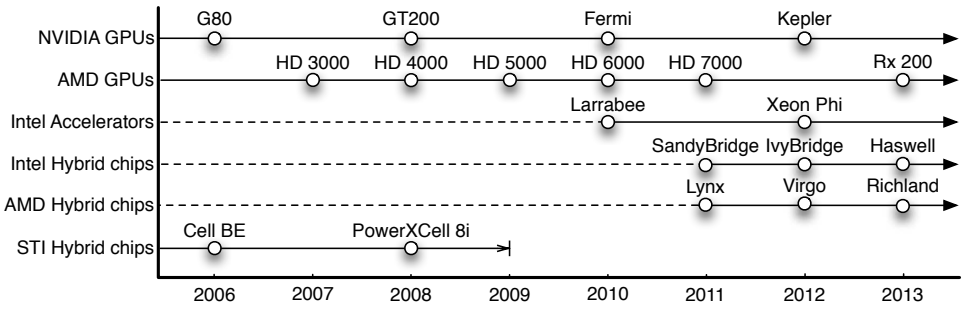


Fig. 2.1: Timeline of accelerator/hybrid devices

The strong point of GPUs is the way they handle thousands of in-flight active threads, and make context switching among them in a lightweight way. Running threads may stall when trying to access global memory, a relatively expensive operation. GPUs hide these latencies by rapidly context-switching stalled threads (actually, groups of threads) with active ones.

2.1.2 Many-core coprocessors

The main many-core architecture in HPC is the Intel MIC (Many Integrated Core), which is nowadays implemented in the Xeon Phi range of coprocessors. These devices have become heavyweight players in the accelerator arena, as demonstrated by the prominent positions held by Phi-based supercomputers in the Top500 list.

A device with a preliminary version of the MIC architecture was Larrabee, a PCIe accelerator composed of 32 x86 cores. It was announced in 2008 [14], and some prototype cards were shipped in 2010. In 2012, Intel commercially launched the Xeon Phi Coprocessor (codenamed Knights Corner) as an actual implementation of the MIC architecture, becoming an immediate success [15].

A Xeon Phi coprocessor of series 3100 houses 57 cores running at 1.1 GHz (1003 GFLOP/s of peak double precision performance), together with 6 GB of RAM, and can be attached to a host computer through PCIe. Series 7100 devices have better specifications: 61 cores at 1.238 GHz (1208 GFLOP/s) and 16 GB of RAM. The x86 cores used in these devices are less powerful than those in state-of-the-art Xeon processors, but they are still more general-purpose than those used in GPUs. They support 4-way Simultaneous Multithreading (SMT) and integrate Single Instruction, Multiple Data (SIMD) units. Interconnection among them and with the memory is via a ring bus.

2.1.3 Hybrid architectures

In recent years, manufacturers have marketed hybrid chips combining different types of cores in the same die. A prototypical example is AMD's family of Accelerated Processing Units (APUs) [16], which combine several x86 cores with a GPU composed of several SIMD cores. Similarly, Intel has its own family of processors with integrated graphics [17]. The target market of these hybrid processors is low cost, low power computers, with the focus on mobility. A previous implementation of this class of hybrid chips was the Cell Broadband Engine [18].

GPUs integrated into hybrid chips are not very different from those used in discrete accelerators. They may provide reduced performance (fewer cores), due to cost and power limitations,

thus being less attractive for HPC. However, they have an important advantage when compared to discrete accelerators: memory spaces for GPU and CPU are not separated; in fact, memory is shared. This drastically reduces the overheads associated with CPU–GPU communication and synchronization, because data movement through PCIe (required with discrete accelerators) is not necessary. Studies comparing the performance of the AMD Fusion platform against a discrete CPU+GPU system [19, 20] conclude that APUs are more compute efficient than a discrete GPU for applications involving massive data movements through the system bus.

2.1.4 Development environments

Applications for accelerators are generally written using software development tools provided by the device manufacturers. They can consist of manufacturer-specific tool chains, or can be implementations of standard APIs. OpenCL [21] is a standardized, vendor-neutral framework for programming all classes of accelerators, defining a hardware model and an API. CUDA [22] implements similar concepts, but it is specific for NVIDIA GPUs and uses a slightly different terminology. Both models are widely used for GPGPU.

Compared to GPUs, the Xeon Phi is a relative newcomer to the accelerator arena. However, it has inherited from Intel's vast experience with multi-core processing, and the result is the availability of a diverse collection of tools that makes the learning curve of the developer less steep. Among other APIs, Phi developers can use OpenMP, OpenCL and MPI [15].

In order to simplify programming, debugging and tuning accelerator-based applications, a diversity of higher-level tools and language constructions have appeared in the last years, which try to hide hardware complexities and to offer programmers a higher-level view of accelerators. Some of these approaches include the use of optimized libraries (for example, clMath [23]), programming with directives (such as OpenACC [24] or version 4 of OpenMP [25]), or relying on the parallelizing capabilities of the compiler (using, for example, the Par4All [26] open-source compiler workbench).

2.2 Kernel Density Estimation

KDE is a statistical technique used to estimate the probability density function of a sample set with unknown density function. It was first introduced in the 1960s for univariate data and, due its widespread adoption, multivariate estimators appeared in subsequent years. It is considered as a fundamental data smoothing problem and it is widely used due to its properties (smoothness, continuity) in contrast to other common density estimation techniques, such as histograms [27].

KDE is a common tool in many research areas, used for a variety of purposes. For example, in [28] authors use density estimates to forecast weather and other factors as part of a model for optimizing maize production. In the same field, it has been applied to evaluate the signature of climate change in the frequency of weather regimes [29]. In [30], the effectiveness of a particular medical treatment is determined by means of KDE. In computer vision [31], it is applied for image segmentation and tracking. An extensive list of application fields of KDE can be found in [32].

Given several observed data points (samples) from a random variable X, with unknown density function f, KDE is used to create an estimated density function $\hat{f}$ from the observed data. One of the most common techniques for density estimation of a continuous variable is the *histogram*, which is a representation of the frequencies of the data over discrete intervals (bins). Its main drawback is the lack of continuity. The KDE technique relies on assigning a kernel function K to each sample (observation in the dataset), and then summing all the kernels to

obtain the estimate. In contrast to the histogram, KDE constructs a smooth probability density function, which may reflect more accurately the real distribution of the data. We now describe the KDE technique in more detail.

Given a multivariate sample set $(\boldsymbol{x}_1, \boldsymbol{x}_2 ... \boldsymbol{x}_n)$ where $\boldsymbol{x}_i$ is the i-th sample value from an unknown density f, KDE builds an estimation the following way:

$$\hat{f}(\boldsymbol{x}) = \frac{1}{n} \sum_{i=1}^{n} K_H \left(\boldsymbol{x} - \boldsymbol{x}_i\right) \tag{2.1}$$

where:

- n denotes the cardinality of the sample set.
- $K_H(\boldsymbol{x}) = |H|^{-1/2} K(H^{-1/2}\boldsymbol{x})$.
- $K(\boldsymbol{x})$ refers to the kernel function used to define the weight or influence of each sample.
- H is a *dxd* dimensional diagonal matrix containing the *bandwidth* or *smoothing factor* value for each dimension. If the bandwidth is the same in all dimensions, we refer to it as a scalar parameter h.

Intuitively, the kernel estimator is a sum of "bumps" placed at the sample points. The kernel function K determines the shape of the bumps, while the smoothing factor h determines their width. As an example of KDE, Figure 2.2 depicts a simple estimator for a one dimensional space with three sample points. Each of the sample points is surrounded by a kernel depicted as a bell and the estimator is depicted as the thick line over the kernels. Note that, even though a density estimation is a continuous function, KDE programs generate as output the values of the estimation in the discretized space requested by the user.

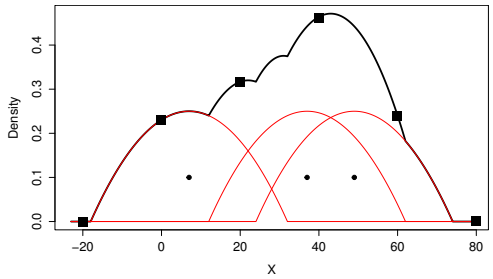


Fig. 2.2: Example of KDE for 1D data. Small dots represent the samples, and the red lines are the kernels around them. The estimated density is computed at the indicated points (square dots) in a discretized grid with a step of 20

The bandwidth parameter h controls the influence area and smoothness of kernels. It is used to reduce the noise in the density estimation, and it must be carefully selected. To describe its effect, we depict in Figure 2.3 a simulated random sample from the Gaussian distribution in 1D, and different estimations of the density derived from those samples. The solid curve depicts the real density of the data, while the other lines depict (kernel-based) density estimations using different values of h. The dashed line corresponds to a large h, and results in an over-smoothed estimator. In contrast, the dotted line corresponds to a small h and results in an exceedingly sharp estimator. None of the cases reflect accurately the actual density of the samples. Several techniques to aid in the selection of the optimum bandwidth are detailed elsewhere [27] [33].

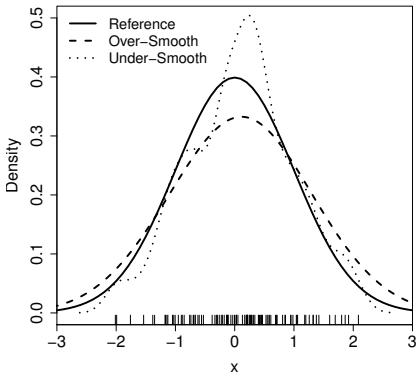


Fig. 2.3: Effect of the smoothing parameter when estimating data sampled from a normal distribution

A kernel function K is a symmetric but not necessarily positive function that integrates to one. Kernels can be classified into two groups: *bounded* and *unbounded*, depending on their area of influence. Two widely used kernel functions are *Epanechnikov* (which is bounded) and *Gaussian* (which is unbounded). They are defined in Equations 2.2 and 2.3 respectively. In Equation 2.2, c_d is the volume of the unit d-dimensional sphere: $c_1 = 2, c_2 = \pi, c_3 = 4\pi/3$, etc. For the sake of clarity, we have depicted in Figures 2.4 and 2.5 the Epanechnikov and Gaussian kernels for 1D spaces. Note how bounded kernels only affect those points in the space at a limited distance (1 in the figure), while the influence area of unbounded kernels spans infinitely in both directions. The choice of the particular kernel to apply is up to the KDE user, taking into account the problem at hand.

$$K(\boldsymbol{x}) = \begin{cases} 1/2 c_d^{-1}(d+2)(1-\boldsymbol{x}^T\boldsymbol{x}) & \text{if } \boldsymbol{x}^T\boldsymbol{x} < 1 \\ 0 & otherwise \end{cases} \tag{2.2}$$

$$K(\boldsymbol{x}) = \frac{1}{\sqrt{(2\pi)^d}} exp\left(-\frac{1}{2}\boldsymbol{x}^T\boldsymbol{x}\right) \tag{2.3}$$

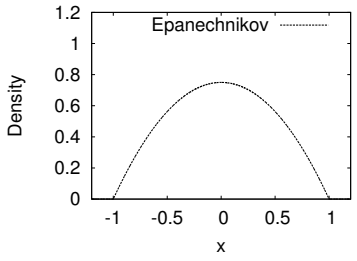


Fig. 2.4: Shape of a Epanechnikov kernel in 1D

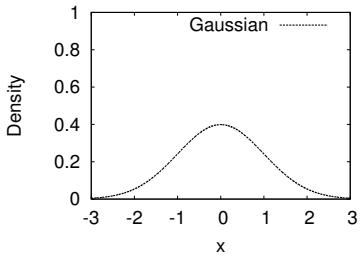


Fig. 2.5: Shape of a Gaussian kernel in 1D

3

Contributions

This dissertation aims at providing insights about how a developer can get the most out of general purpose coprocessors. As described in the previous chapter, accelerators offer the potential of huge processing power, but not every code can be suitably adapted to the computing model they impose. Along this chapter we will study different tools, techniques and methods to analyze and optimize massively parallel codes. To that end, we will use Kernel Density Estimation (KDE), also described in the previous chapter, as a case study.

3.1 A Survey of Performance Modeling and Simulation Techniques for Accelerators

A common pitfall when developing programs for accelerators is to carry out code implementation and tuning using a trial-and-error methodology, without appropriate feedback and guidance from performance tools. Unfortunately, in the field of accelerator-based computing there is no outstanding tool or model that can be considered as the reference instrument for performance prediction and tuning. There is, though, an extensive body of literature related to this (relatively) novel area. A major contribution of this work is a thorough review of the literature about this topic.

As support for the developer, tools such as profilers and debuggers, together with best practices manuals written by the manufacturers of accelerators, are of great help during the process of porting and tuning an application. However, they lack prediction abilities: they cannot estimate performance on a different platform, or for a new application. Additionally, these tools can overwhelm the programmer with excessive information, making it difficult to filter out what is actually limiting the performance.

We initially set our focus on models of parallel applications running on accelerators, that can be used to analyze (and predict) their combined performance. These models require input data (the characteristics of the target application and platform), and may provide different sets of output data, such as performance estimations or a list of bottlenecks.

A performance model can be seen as a system representation that provides output values based on a given set of input parameters. Depending on the characteristics of the model and the goal it has been designed for, the input and output datasets can be notably different. In addition, they can be classified according to different criteria. In this work we propose a classification criterion based on the output generated by the model, that is, the information it provides about a particular hardware/software combination. We have identified models designed to:

- predict the execution time of a target application on a target hardware platform,

- identify performance bottlenecks and propose code modifications to avoid them,
- provide estimations of power consumption, and
- provide detailed, step-by-step information of resource usage, based on simulation

The first group contains models for the estimation of the execution time of an application (or a portion of an application), that can be useful in situations such as making a decision about acquiring new hardware or testing the application for scalability. After a review of performance models aiming to predict the execution time of an accelerator-based parallel application, we have organized the different works taking into account whether they propose general models applicable to any program/kernel, or they are specific for a particular application or pattern. We have also taken into account the input information required by the model: actual kernel codes, or pseudo-codes/patterns derived from the structure of the program.

The second group contains models developed with the aim of helping developers in off-line code optimization tasks. While a program is being ported and fine-tuned, it is necessary to carefully analyze its behavior at run time on the target platform, looking for potential resource bottlenecks and, if possible, finding alternatives to eliminate or mitigate them. Toolkits for programming accelerators commonly include a profiler (e.g. NVIDIA Visual Profiler for the CUDA platform, CodeXL for the AMD OpenCL platform or VTune for the Intel Xeon Phi platform), which should be the first tools to use when optimizing a kernel. They analyze code execution, spotting bottlenecks, and can even make recommendations to the programmer about code changes or compiler flags. However, they cannot predict the performance benefits associated to these changes. Models in this second group aim to be complementary to the use of profilers, going a step further in several aspects. They are discussed in two subgroups: some of them create, given an application, an entire dissection under different criteria such as execution time or resource consumption, while others that try to find specific bottlenecks in the code.

The third group contains models that estimate the energy required to run a particular code on a particular device. A trending topic in the HPC field is improving the power efficiency of computers. To that extent, significant effort is being devoted to modeling the power characteristics of systems. These models consider not only code and device properties, but also program input and some other run-time characteristics. We have reviewed a collection of proposals aiming to model power efficiency. We found two types of models in this group: standalone, and tied to simulators.

In the last group we have reviewed simulators. A simulator is a system representation (model) able to mimic, step-by-step, the behavior of the target (real) system. Simulators are widely used to carry out performance studies of existing hardware and software platforms, and also to analyze platforms that either do not exist, or are not available. The accuracy of the output information provided by a simulator depends on many factors, among them the level of detail with which the system has been modeled and the quality and detail of the workloads provided to feed the model.

We surveyed 29 performance models targeting modern coprocessors and classified them following the described taxonomy. For each one we extracted the following features:

- Input requirements, such as running μBenchmarks or analyzing intermediate codes.
- Limitations, such as lack of detail in memory modeling or inaccuracies for a particular type of application.
- Highlights and benefits beyond other models, such as ease of use or extendability.

The conducted survey allowed us to identify a few performance models outstanding over the remaining ones: *MWP-CWP* by Hong et al. [34] would be the model of choice to predict execution times in GPUs, due to its ease of extension to upcoming architectures and GPUs of

different vendors. The *roofline model* [35] is useful to determine the factor that limits program performance in parallel systems (including accelerators), due to its ease of use and visual output to guide optimizations. Finally, GPGPU-Sim [36] seems the simulator of choice for GPU-based accelerators due to its rich set of features, and its capabilities to be enhanced with a power model.

In addition, we concluded that all currently available models and tools have limitations, that should be overcome in the future:

- There is no accurate model valid for a wide set of architectures. Each model finds a different trade-off between being more device-specific and therefore more accurate, or being more general purpose at the cost of losing accuracy. Furthermore, the fast pace at which manufacturers market new products, with new or improved features, makes models obsolete in a very short time.
- A majority of the discussed models have been designed for CUDA, the most mature development environment for GPGPU. However the vendor neutrality of OpenCL and its availability for non-GPU accelerators is increasing its adoption by HPC programmers.
- Society is becoming aware of the great monetary and environmental costs derived from the high energy consumption of computing systems. The challenge is not only to squeeze the maximum performance out of a system, but also to do it with the minimum power. As often these two requirements cannot be optimized simultaneously, good trade-offs have to be found. Power models can help to solve this bi-objective optimization problem.
- Reviewed models focus on offloading compute-intensive tasks to accelerator devices, leaving the CPU idle while the accelerator is busily crunching numbers. It is possible, however, to make both work simultaneously, to increase system efficiency. There are proposals dealing with this workload distribution, using static or dynamic scheduling techniques. Energy can also be included into the equation, making power-aware load balancing across heterogeneous systems.

The complete survey of performance modeling tools can be found in Chapter 5. This chapter also includes strong background information about the main benchmark suites used for performance analysis; some of them will be used later in this dissertation.

3.2 S-KDE: An Efficient Algorithm for Kernel Density Estimation

A collaborative work with another research group of the UPV/EHU made us focus our attention on a problem in the climatology field with huge computational requirements: the evaluation of environmental models. The design and optimization of novel methodologies for climate model evaluation is a challenging task, that requires extensive use of KDE algorithms. Any acceleration in the execution of KDE translates immediately in enabling faster, more accurate model evaluation methods.

We soon understood that high levels of acceleration cannot be achieved by simply adapting the available, serial code, to run in a parallel computing system. In fact, the first thing we did was understanding the basics and constraints of the problem in order to propose a *novel algorithm* with greatly reduced complexity. The achieved performance gains are visible even with a serial implementation. Only after this was done, we proceed to further accelerate the program using parallel computing. We describe in this section the algorithmic changes made to the reference implementation of KDE, and discuss the parallel implementations afterwards.

KDE generates an estimation of the probability density function of a sample set. This estimation is computed in a user defined *evaluation space* that should include all the observation points in the sample set. In the example of Figure 2.2, the evaluation space spans from -20

to 80, covering the three samples plus some "extra" space at the boundaries in order to fully accommodate the domain where the kernel is defined.

KDE implementations usually discretize this space, computing the density in equally separated *grid points*. This way, the evaluation space can be represented as a multi-dimensional matrix, the *evaluation grid*. The separation between grid points is defined by a user-provided *evaluation step* or *grid step*, that can be different in each dimension. In Figure 2.2, the density function will be computed at six evaluation points (-20, 0, 20, 40, 60 and 80), which are equally separated at distance 20.

A common way to compute KDE is described in Algorithm 1. For each evaluation point, the density that all the samples generate on it is computed, which depends on the kernel of choice. This computation requires measuring the distance between the evaluation point and each sample. The combined density at the evaluation point is the sum of all the partial densities. We call this approach *evaluation point-wise KDE* or *EP-KDE* for short.

Algorithm 1 Evaluation point-wise KDE (EP-KDE)

```
for each Evaluation Point e do
  value(e) = 0
  for each Sample s do
    dist = computeDistance(e,s)
    value(e) += computeDensity(dist)
  end for
end for
```

The computational complexity of EP-KDE is $O(k_d mn)$, where k_d is a constant related to the dimensionality of the dataset, m is the number of evaluation points, and n the number of samples. Note that m is proportional to the size of the evaluation space and the grid step. For large, multi-dimensional spaces or/and tight grid steps, m can be *huge*.

The EP-KDE approach is valid for both unbounded (e.g. Gaussian) and bounded (e.g. Epanechnikov) kernels. In the first case, all the samples affect all the evaluation points. In contrast, with bounded kernels, samples only contribute to the density in those evaluation points within its influence area. Therefore, when using EP-KDE with bounded kernels, in most cases the *computeDistance* function of Algorithm 1 will return a value outside the bounds of the kernel and, therefore, function *computeDensity* will return 0.

As EP-KDE with bounded kernels can lead to a huge amount of worthless computations (that would depend on the size of evaluation space and on the dispersion of the samples), we have developed a more efficient algorithm for this group of bounded kernels. It is described in Algorithm 2, and we call it *sample-wise KDE* or *S-KDE* for short. Instead of focusing one by one on each evaluation point and the influences of all samples over it, S-KDE focuses one by one on each sample, computing its influence on the evaluation points surrounding it. As the kernel is bounded, the area of influence is confined within a *bounding box*, which is computed in advance. This bounding box is a hyperrectangle whose dimensions are determined by the maximum per-dimension distances of influence of the kernel. Note that this is kernel-dependent, but not sample-dependent. Thus, the size and shape of the bounding box $\boldsymbol{b}$ is computed just once. Then, for each sample the bounding box must be aligned to the evaluation grid, defining the per-sample bounding box $\boldsymbol{b_s}$.

With S-KDE, computations of distance and summations of influences are greatly reduced: for each sample most evaluation points fall outside the influence area of the kernel, and are not

Algorithm 2 Sample-wise KDE (S-KDE)

```
for each Evaluation Point e do
  value(e) = 0
end for
b = computeBoundingBox
for each Sample s do
  b_s = adjustBoundingBox(b,s)
  for each Evaluation Point e in b_s do
    dist = computeDistance(e,s)
    value(e) += computeDensity(dist)
  end for
end for
```

considered in subsequent computations. The complexity of S-KDE is $O(k_d np)$, where k_d is a constant related to the dimensionality of the dataset, n is the number of samples and p is the size of the bounding box, which is normally *much smaller* than the total number of points in the evaluation grid m.

The described S-KDE approach requires the computation of the bounding box b that surrounds the area of influence of each sample. We will now describe how b can be computed, assuming that an Epanechnikov bounded kernel is used. The area of influence of a sample using this kernel has an elliptic shape in 2D spaces, and a d-dimensional ellipsoid shape for spaces of higher dimensions. The challenge is to find a box bounding the ellipsoid, and to align it to the discretized grid that defines the evaluation space. To do so, we propose an approach based on the eigenvalues of the covariance matrix of the dataset; this is the natural choice when, as in our case, a multidimensional Fukunaga estimator with the Mahalanobis distance is used [37]. In Figure 3.1 we depict an example: a sample s in position *(6.75, 4.5)*, its area of influence (the ellipse), and the bounding, aligned rectangle (dashed line).

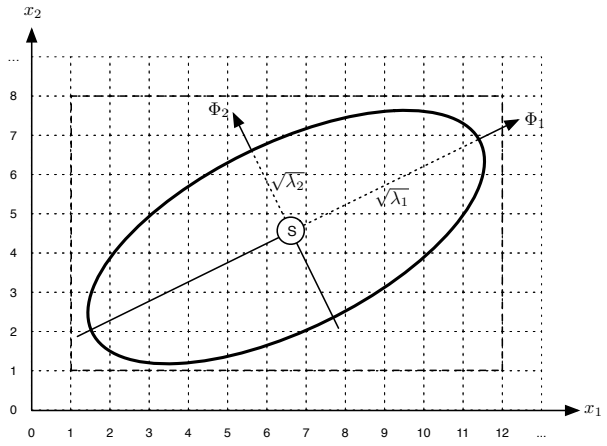


Fig. 3.1: Area of influence of a sample and the corresponding bounding box

The evaluation points influenced by the sample are *only* those within the ellipsoid, which means that the bounding box contains evaluation points whose contribution to the density is zero. In fact, depending on the shape and angle of the ellipsoid, the evaluation points outside the ellipsoid could be more numerous than those inside it. Motivated by this fact, we define a way to make a tighter delimitation of bounding boxes. The first idea is to reduce the dimensionality of the problem, splitting (chopping) the original d-dimensional box into several non-overlapping $d-1$-dimensional boxes. Each of them is again chopped into $d-2$-dimensional boxes, repeating the process until a collection of 2D rectangles (slices) is obtained. Then, each rectangle is cropped until reducing it to a smaller one minimizing the number of evaluation points not influenced by the sample. This two-step procedure is represented in Figures 3.2 and 3.3.

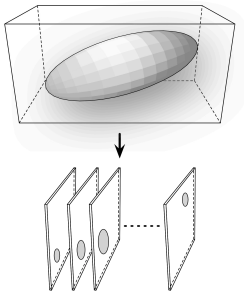


Fig. 3.2: Chopping a 3D bounding box into 2D slices

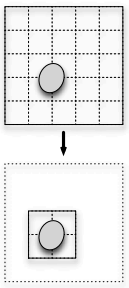


Fig. 3.3: Cropping a 2D slice to obtain a minimum-size bounding rectangle

We provide some example figures to illustrate the efficiency of our approach. We will assume a 3D dataset with 500k samples, and a evaluation grid with 194.81 million points. The traditional EP-KDE approach that traverses every evaluation point of the grid would execute $9.74 * 10^{13}$ sample-evaluation point operations. In contrast, a rectangular 3D bounding box around each sample in the mentioned scenario contains on average 102461 points, and using the sample-wise KDE approach would require $5.12 * 10^{10}$ computations. If we apply the *Chop & Crop* technique, the number of evaluation points per bounding box is further reduced to 53511 on average, and the resulting total number of computations is $2.67 * 10^{10}$. S-KDE, thus improves KDE efficiency by several orders of magnitude. A more extensive description of the S-KDE algorithm can be found in Chapter 6.

3.3 S-KDE in Multi and Many-Core Processors

In the field of climatology, KDE is used with large datasets and in an iterative manner. Although S-KDE boosts the speed of KDE computations, any additional acceleration would be welcome. We tackled this issue using the traditional HPC approach: through parallel computing. Therefore, our next step was to implement S-KDE as a data-parallel program, initially targeting multi-core and many-core processors.

The workflow of our parallel S-KDE code is depicted in Figure 3.4. Each thread is in charge of computing the influence of a set of samples over the evaluation grid. For each sample, the thread first adjusts its bounding box. If the evaluation space is of dimensionality three or higher, the *Chop & Crop* procedure is recursively applied to reduce the computation to 2D slices. Then,

for each 2D bounding rectangle the density that the sample creates in each evaluation point is computed. The parallel code has been developed in ANSI-C, making use of OpenMP and Intel compiler directives.

One of the drawbacks of performing sample-wise computations is the memory contention that may appear when two or more different threads, managing samples whose influence area overlap, have to add partial density values into the memory position that represent the same evaluation point (this may happen in the *consolidation* step, the last one in Figure 3.4). To reduce this harmful effect, each thread calculates every row of the slice in its private memory, and adds it into main memory using the *atomic* OpenMP pragma. This way, we ensure data write consistency. There is a cost to pay, though: atomic operations causes overheads due to the serialization of memory write operations.

We have tested our S-KDE code in two different hardware platforms: an Intel Core i7 3820 multi-core CPU (4 cores @ 3.60 Ghz) and an Intel Xeon Phi 3120A many-core coprocessor (57 cores @ 1.1 Ghz), with datasets of diverse dimensionality and size. Here we focus on a single 3D dataset with 500k samples. The evaluation is conducted varying the grid step in order to increase or reduce the problem size.

In an initial comparison, we measured the execution times of our S-KDE implementation against two public implementations of KDE: *ks-kde* from the R statistical software, linked with the Intel MKL library to perform multi-core computations [38], and GPUML, a GPU based one [39]. The tests of the latter were run on a NVIDIA GTX 650 GPU. In addition, we implemented a parallel version of EP-KDE with OpenMP and run it in the same platforms used to test

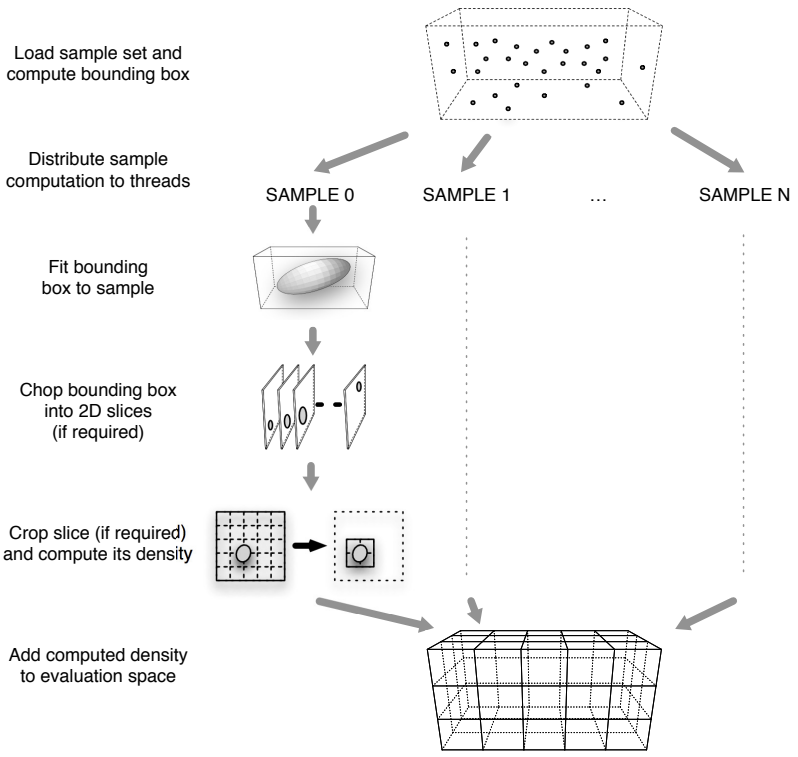


Fig. 3.4: Workflow of our S-KDE implementation

S-KDE. Results are depicted in Figure 3.5. Our S-KDE implementation running in the Xeon Phi obtained speed-ups over 460x when compared against the *ks-kde* implementation. Note that this acceleration includes the combined effects of the algorithmic changes and the OpenMP parallelization.

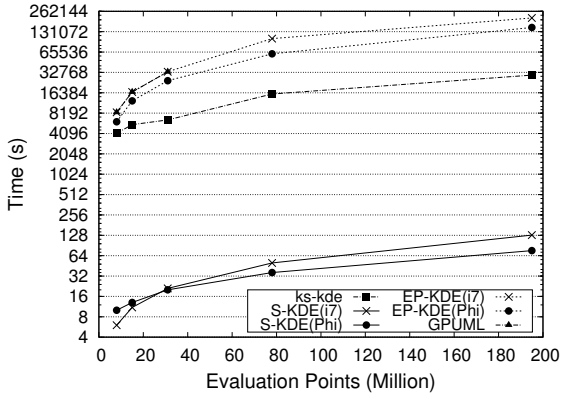


Fig. 3.5: Execution times of the KDE implementations under test, for 3D datasets with 500k samples

We conducted additional tests to analyze different features of the S-KDE implementation. In one of the tests we run S-KDE with *Chop & Crop* enabled and disabled to assess the performance gains derived from the use of this technique. Some results for a 3D dataset in the Xeon Phi are depicted in Figure 3.6. We measured that S-KDE runs 60.4% faster in the Core i7, and 49.2% in the Xeon Phi, due to the removal of useless computations (and the corresponding memory accesses). In other tests, we tried to dig and find the bottlenecks of the code. We conducted a dissection of the execution times to find the portion of time devoted to the different stages of the code and found the atomic memory writes used for results consolidation to be the main bottleneck. Figure 3.7 depicts a dissection of the execution for a 3D dataset in both architectures, showing the large portion of time devoted to memory writes.

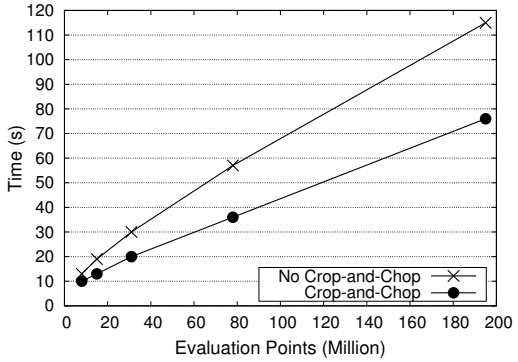


Fig. 3.6: Execution times for 3D dataset 1 with crop-and-chop enabled/disabled in the Xeon Phi

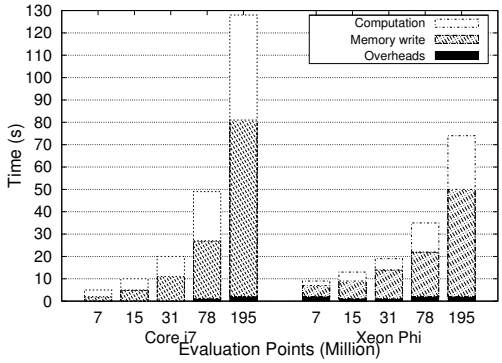


Fig. 3.7: Dissection of execution times of S-KDE for a 3D dataset in the Core i7 and the Xeon Phi

We concluded from the experiments that:

- The current implementation of S-KDE has shown that, compared with available, state-of-the-art implementations of KDE, it provides, *by far*, the best performance, even when running in a modest i7 processor. This performance can be boosted if a many-core coprocessor is available.
- Although the overall performance of the code is satisfactory, we have detected several bottlenecks (the main one being memory write contention) that require further exploration. This is left as future work.

The complete description of the evaluation and the results, and further discussion on potential improvements can be found in the Chapter 6 of this dissertation.

3.4 S-KDE in General Purpose Coprocessors

The S-KDE algorithm presents an important advance in the computation of KDE, and its parallel implementation in multi-cores and many-cores has proven to be very efficient when compared against competitors. However, the described implementation targets a limited set of coprocessors. We have already discussed in Chapter 2 the wide spectrum of coprocessors available in the HPC landscape. Thus, we felt that our S-KDE approach should be available for a larger set of accelerators.

Porting a code to run in accelerators usually requires a major recoding effort, as it has to be adapted to a massive data parallel processing model. In order to target the widest possible set of accelerators, we have chosen OpenCL as development platform. We have structured the accelerator-based implementation of S-KDE as a sequence of steps, depicted in Figure 3.8. Steps between parentheses are additional operations required in the accelerator-based implementation and not required in the serial or OpenMP codes:

1. *Initialization:* In a first step, the entire sample dataset is copied into the accelerator, along with the required support structures. We assume that all the dataset fits in the memory of the accelerator (note that this is not the evaluation grid). In this step we also compute the size of the generic bounding box. This is host code (executed in the CPU).
2. *Box fit and Chop:* For every sample, its bounding box is fitted to the grid and chopping is applied. At this point, the problem has been reduced to a collection of 2D slices. This is implemented as an OpenCL kernel (executed in the accelerator in a data parallel way).
3. *Crop:* Cropping is applied to reduce the number of evaluation points in each slice. Additional information about each slice is computed, such as its coordinates in the evaluation space and the number of evaluation points it contains. This is an OpenCL kernel.
4. *PrefixSum:* A PrefixSum is applied to the vector that contains the number of evaluation points per slice. This support computation is required by the next kernel for its threads to make ordered stores. This is an OpenCL kernel.
5. *Density Computation:* Each thread calculates the influence created by a sample on an evaluation point of a given slice. The resulting densities are stored in an auxiliary vector and not consolidated into the global PDF structure. This is an OpenCL kernel.
6. *Densities Transfer:* The resulting vector of partial densities is transferred through PCI-Express from the accelerator to host memory. This is managed by the host.
7. *Consolidation:* The host reads the vector of partial densities and accumulates them into the evaluation space. This is host code.

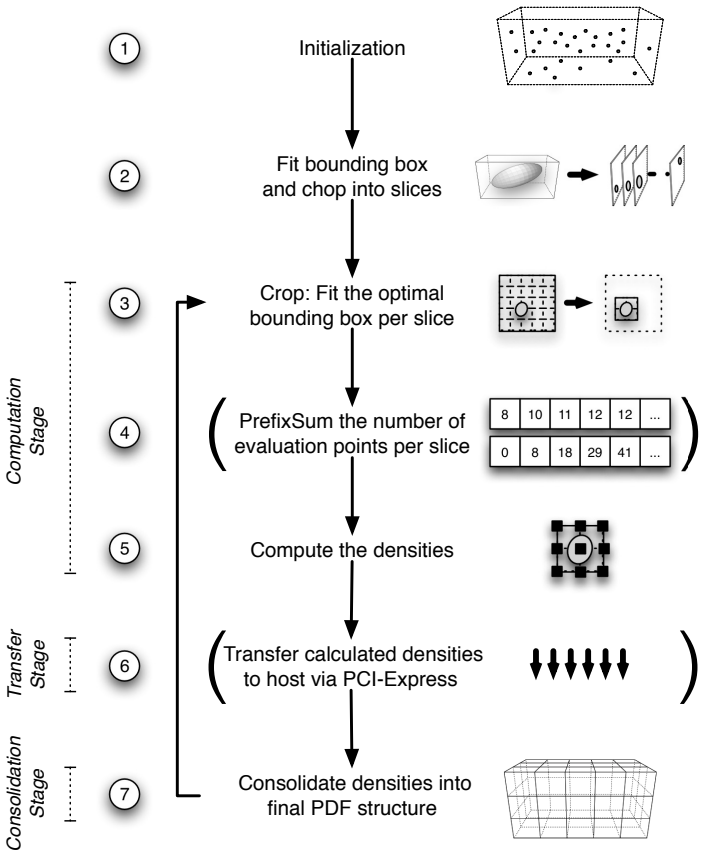


Fig. 3.8: Workflow of OpenCL KDE implementation

The main program is iterative after the second step. Each iteration consists of processing a "chunk" of the problem, which is nothing more than a subset of the samples. The size of this chunk is computed on a per-device and per-problem basis, to guarantee a high degree of parallelism and a size of intermediate data structures that fit into the accelerator memory. Then the program operates in a data-parallel fashion, using a chain of OpenCL kernels (*Computation* stage in Figure 3.8). The per-chunk intermediate results are stored in the accelerator and, later, transferred (*Transfer* stage in Figure 3.8) to the CPU for consolidation into the main densities matrix (*Consolidation* stage in Figure 3.8).

The arrangement in stages has enabled the implementation of a pipeline of operations: while the device is busy in the Computation and Transfer stages for a chunk, the CPU is consolidating the results of the previous chunk. An example of the pipeline is depicted in Figure 3.9. A deeper pipeline would be possible (simultaneous Computation, Transfer and Consolidation) but, given the duration of the Consolidation phase, it does not result in performance benefits.

We have carried out a performance analysis of the OpenCL code in three modern accelerators: an AMD Radeon HD 6950 GPU, a NVIDIA GTX 650 GPU and a Intel Xeon Phi 3120A coprocessor. Our aim was to conduct the analysis using device independent resources instead of manufacturer specific tools (e.g. profilers). To that extent, we have relied on some of the performance models and tools presented in Chapter 5 (summarized in Section 3.1). We report here

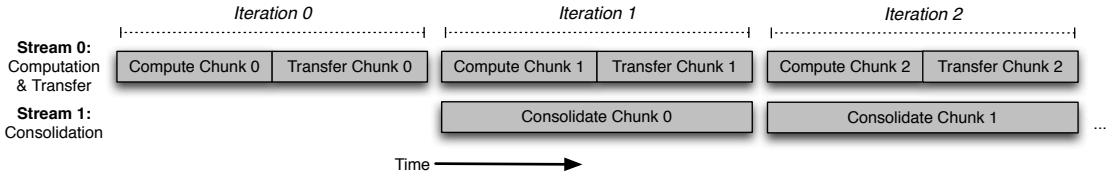


Fig. 3.9: Pipelined execution of the OpenCL implementation of S-KDE

just the results obtained with the 3D, 1M samples dataset, varying the step to increase/decrease the number of evaluation points (the problem size).

We followed a top-down approach, beginning from global performance measurements to lower level details. To get an initial assessment of the performance of our non-pipelined OpenCL S-KDE, we compared its total execution time against that of the serial version, for the three target devices, see Figure 3.10. The OpenCL code runs significantly faster than the serial code, in the three accelerators: speed-ups are 3.47x for the AMD GPU, 3.31x for the NVIDIA GPU and 4.27x for the Intel Xeon Phi, for the most complex of the tested problems. With the simultaneous operation of Computation+Transfer and Consolidation, speedups improve: 4.42x for the AMD GPU, 5.74x for the NVIDIA GPU and 5.67x for the Intel Xeon Phi, for the same problem.

After this black-box assessment, we went deeper into finer grain details. We first computed the accumulated time spent in each of the stages executed per program iteration. Results for the largest problem size are depicted in Figure 3.11. We identified the Consolidation stage as the main performance bottleneck: it takes longer that the other two stages together, determining in the pipelined program the total execution time. These times also explain the lack of additional benefit derived from a deeper pipeline.

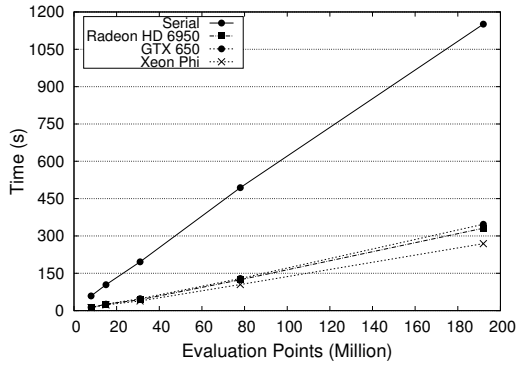


Fig. 3.10: Execution times of the OpenCL KDE implementations

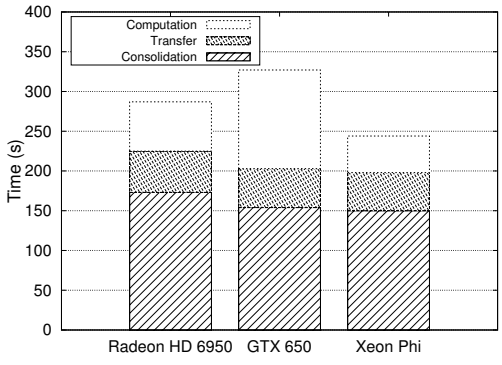


Fig. 3.11: Dissected execution time of OpenCL S-KDE

To better understand the performance of the code running in the accelerators (the kernels in the Computation stage), we first characterized several aspects of the devices, such as their computational efficiency (using the Roofline model) and the PCI-Express transfer capabilities (with the public SHOC benchmark suite). We depict here as an example the visual output of these evaluations for the AMD Radeon HD 6950 GPU, see Figures 3.12 and 3.13 respectively.

We concluded that our OpenCL S-KDE code reaches acceptable efficiency levels given the characteristics of the algorithm: it has (relatively) low complexity, but it is also memory-bound.

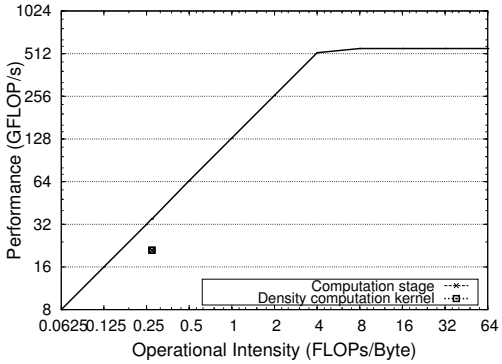


Fig. 3.12: Roofline of a AMD HD Radeon 6950 GPU

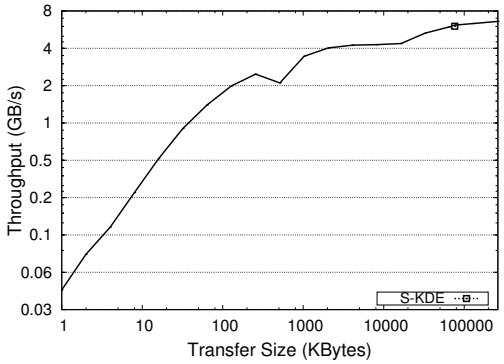


Fig. 3.13: SHOC PCI-Express Test on a AMD Radeon 6950 GPU

Therefore, it does not fit in the accelerator model in a straightforward way. However, we managed to get close to the limits of the accelerators.

Additional implementation details and discussions about design and implementation decisions, together with a more thorough evaluation of the OpenCL S-KDE code, can be found in Chapter 7.

3.5 A Methodology for Environmental Model Evaluation based on S-KDE

As discussed before, our motivating scenario was in the climatology field, in a problem in which KDE was extensively used: the evaluation of environmental models. In this section we propose a novel methodology for this evaluation, which is enabled by the availability of S-KDE, a fast KDE algorithm.

Climate models are mathematical representations of a climate system, based on physical, chemical and biological principles [40]. They usually include complex equations that represent these laws, that are solved numerically. Climate models provide discrete results in space and time, whose accuracy depend on the resolution of the model. Nowadays climate models are used by scientists to understand climatic changes from a dynamic point of view and to give quantitative answers to questions about future climate.

In order to measure the confidence of the results that climate model produce, they must must be evaluated against different observations [41] or paleoclimate data [42]. The rationale behind this hypothesis is that models that are the best simulating current climate are expected to be also the best simulating future climate. Some climate model evaluation exercises are based principally on seasonal and annual time scales. However, monthly or seasonal averages can hide biases or ordinal errors that are identifiable in daily data. In addition, climate on time scales of days has a direct impact on human health and activities (e.g. agriculture). Thus, more recent model evaluation exercises tend to provide estimates on time scales of days [43].

Many climate model evaluations are based on averaged output values for the target variables, such as temperature, air pressure and sea level. However, it has been proven that this approach is not the most accurate one, as a mean value, even when it is precise, does not represent with fidelity the distribution of a variable. For this reason, recent climate model evaluations are based on Probability Density Functions (PDFs) of the target variables [44]. These PDFs can

be generated using KDE techniques, and result in higher computational demands during the evaluation process. An advantage of using PDFs in climate models is that they are capable of evaluating the probability corresponding to situations that are currently rare but may become common in the future [45].

We propose a novel methodology for the evaluation of climate models by means of PDFs, in the time scale of days. Part of this methodology is based in the work by Perkins et al. [45]. They propose the index for the evaluation of daily climate models based on PDFs limited to the evaluation of unidimensional PDFs: they can evaluate the performance of a single output variable, e.g. the temperature. However, in some scenarios it is useful to evaluate several variables in a single step. This is precisely what our methodology achieves. It works as follows:

1. *Identify the optimal bandwidth to be used by the estimation of multidimensional PDFs:* Our methodology requires comparing the PDFs generated by the climate model and the PDFs built from the real world observations. KDE is used to build these PDFs – in particular, one of our fast S-KDE implementations. As described in Section 2.2, the bandwidth parameter is critical in KDE computations. In this first step the optimal bandwidth for the PDFs is found using a bootstrap-based cross-validation technique.
2. *Compute the multidimensional PDFs:* Using the optimal bandwidth values from the previous step, compute (using, again, S-KDE) the PDFs for the estimations generated by the climate model, and for the observations.
3. *Compute the similarity score:* Once the PDFs of the model and the observations are available, obtain a similarity score (in the range 0..1), that represents to what extent the estimations produced by the models match the observations. A perfect model would have a score 1. This score is computed extending the index by Perkins et al. to multiple dimensions.

S-KDE is essential in this methodology, because the most costly steps (the first two) require multiple estimations of PDFs. In particular, the bootstrap used in the first step requires the computation of multiple realizations of the PDFs for the same datasets, each set of realizations using a different bandwidth value, until the optimal bandwidth is identified. A slow KDE algorithm would make this step, and the whole methodology, unfeasible in terms of computation time.

The proposed methodology has been applied to two different case studies. The first one corresponds to a realistic application of climate model evaluation [46]. The second one requires working with data of higher dimensionality, and consists of assessing the performance of a coupled atmosphere-ocean reanalysis in reproducing the global scale Sea Surface Temperature and Sea Surface Height.

A more detailed explanation of the methodology, and its complete evaluation can be found in Chapter 8 of this dissertation.

4

Conclusions

In this chapter we summarize the conclusions of this dissertation. More specific conclusions are listed in the previous chapter, and in Part II. We also present the publications included in Part II of this dissertation and the future paths to extend the presented work.

4.1 Conclusions

This dissertation deals with the problem of efficiently using state-of-the-art general purpose coprocessors. These devices have been widely adopted in HPC environments due to their high theoretical performance compared to multi-core CPUs, and many developers have adopted them as computing platforms [8]. However, achieving that theoretical peak remains challenging for two main reasons: first, not every application fits in the massively data parallel model that accelerators are designed for, and second, despite the huge amount of literature and toolsets around these devices, there is a lack of standardized ways of development and use of performance tools.

In this work we have presented several contributions towards an efficient use of accelerators. Next, a short description of the main contributions of this dissertation is listed.

- We have conducted an extensive survey on performance models and tools targeting accelerators. We have used a taxonomy to arrange the different proposals of tools and methodologies, and analyzed their main features and limitations.
- We have proposed a novel algorithm to compute Kernel Density Estimation (KDE) using a sample-wise approach, instead of the common evaluation point-based approach. The complexity of S-KDE is $O(np)$ instead of $O(mn)$, where m is the size of the evaluation grid, n is the size of the sample dataset and p is the size of the bounding box. S-KDE is very efficient because p is normally much smaller than m. Furthermore, we propose a *Chop & Crop* technique to further reduce the size of p.
- We have implemented and evaluated S-KDE as a parallel application targeting multi-core and many-core processors using OpenMP. The code, compared against state-of-the-art implementations of KDE, has an excellent performance, due to the combination of a low-complexity algorithm and an effective parallelization.
- We have also implemented S-KDE as an OpenCL application targeting modern accelerators, purposefully avoiding device-specific optimizations in order to guarantee portability. Its performance has been evaluated on two GPUs and a many-core processor, yielding speed-up values around 4x in all the tested devices. The evaluation included an analysis of application-dependent and platform-dependent factors that hinders further acceleration.

- We have designed a novel methodology for the evaluation of environmental models, that uses extensively KDE computations and that is only feasible in a reasonable time using our efficient S-KDE implementations.

4.2 Future Work

This work has left open several lines of work and research that the author plans to tackle in the future:

- The performance of both parallel implementations of S-KDE is greatly affected by the mechanisms chosen to avoid memory write contention. We plan to rework these codes in order to reduce this bottleneck.
- We plan to create a framework to characterize accelerators and accelerator-based applications to carry out performance predictions. We miss a unified, portable way to characterize accelerator based environments, in contrast with what is available for single and multi-core CPUs [47].
- Related to this, we plan to extend the analysis work conducted with the OpenCL S-KDE code to propose a device-independent methodology to assess accelerator-based applications. This would complement the current literature on code optimization, which is in most cases too specific for a class of devices and / or applications.
- The use case presented in this dissertation uses 3D datasets of climate variables after reducing the dimensionality of the model results. However, recent publications from the Coupled Model Intercomparison Project (CMIP), part of the World Climate Research Programme (WCRP)[1], could be analyzed with a better diagnostic strategy (from the point of view of physics of the problem) with higher dimensionality (up to ten), even after a dimensionality reduction. Even with S-KDE, dealing with those datasets (that do not fit in main memory) poses a significant storage and processing challenge that we plan to undertake.

4.3 List of Publications

As a result of this research work several papers have been published in different journals and conferences. In this section we introduce the papers included in Part II of this dissertation, and their associated references.

Chapter 5: A Survey of Performance Modeling and Simulation Techniques for Accelerator-Based Computing

- This work presents a review of the state-of-the-art in performance tools for heterogeneous computing, focusing on the most popular families of accelerators: GPUs and the Intel Xeon Phi. It describes current heterogeneous systems along with their associated development frameworks and tools. The core of this work is a review of the performance models and tools, including simulators, proposed in the literature for these platforms. It extends the contributions described in Section 3.1. Published as:

  Unai Lopez-Novoa, Alexander Mendiburu, and Jose Miguel-Alonso. A survey of performance

[1] http://cmip-pcmdi.llnl.gov/

modeling and simulation techniques for accelerator-based computing. *IEEE Transactions on Parallel and Distributed Systems*, 26(1):272 - 281, Jan 2015.

Note: this work was published as a main paper and a separate supplementary file. We have included both parts in Chapter 5.

Chapter 6: An Efficient Implementation of Kernel Density Estimation for Multi-core and Many-core Architectures

- This work presents a novel strategy to compute Kernel Density Estimation (KDE) using bounded kernels, trying to minimize memory accesses, and its implementation as a parallel program targeting multi-core and many-core processors. The implementation is evaluated with different datasets, obtaining impressive levels of acceleration when taking as reference alternative, state-of-the-art KDE implementations. It extends the contributions described in Sections 3.2 and 3.3. Published as:

  Unai Lopez-Novoa, Jon Sáenz, Alexander Mendiburu, and Jose Miguel-Alonso. An efficient implementation of kernel density estimation for multi-core and many-core architectures. *International Journal of High Performance Computing Applications*, first published on March 16, 2015.

Chapter 7: Kernel Density Estimation in Accelerators: Implementation and Performance Evaluation

- This work presents an accelerator-based implementation of the statistical technique Kernel Density Estimation (KDE) and its performance evaluation. The code is based on an efficient algorithm to compute KDE and with the aim of being portable across different state-of-the-art accelerators. The work describes first the process of adapting the algorithm to the accelerator paradigm using OpenCL, and its evaluation in three modern coprocessors. In this evaluation the behavior of the code is studied and its limits in every device are found. It extends the contribution described in Section 3.4. To be published as:

  Unai Lopez-Novoa, Alexander Mendiburu, and Jose Miguel-Alonso. Kernel Density Estimation in Accelerators: Implementation and Performance Evaluation. *To be submitted to Parallel Computing.*

Chapter 8: Multi-objective Environmental Model Evaluation by Means of Multidimensional Kernel Density Estimators: Efficient and Multi-core Implementations

- This work presents an extension to multiple dimensions of the univariate index of agreement between Probability Density Functions (PDFs) used in climate studies. A set of high-performance programs targeted both to single and multi-core processors are presented as well. They compute multivariate PDFs by means of kernels, the optimal bandwidth using smoothed bootstrap and the index of agreement between multidimensional PDFs. Their use is illustrated with two case-studies. Results show that the proposed methodology is robust to variations in the optimal bandwidth used. It extends the contribution described in Section 3.5. Published as:

Unai Lopez-Novoa, Jon Sáenz, Alexander Mendiburu, Jose Miguel-Alonso, Iñigo Errasti, Ganix Esnaola, Agustín Ezcurra, and Gabriel Ibarra-Berastegi. Multi-objective environmental model evaluation by means of multidimensional kernel density estimators: Efficient and multi-core implementations. *Environmental Modelling & Software*, 63(0):123 - 136, 2015.

References

[1] C. Severance and K. Dowd, *High Performance Computing*. OpenStax CNX, 2010.

[2] D. B. Kirk and W.-m. W. Hwu, *Programming Massively Parallel Processors: A Hands-on Approach*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1st ed., 2010.

[3] S. Ryoo, C. Rodrigues, S. Baghsorkhi, S. Stone, D. Kirk, and W. Hwu, "Optimization principles and application performance evaluation of a multithreaded GPU using CUDA," in *Proceedings of the 13th ACM SIGPLAN Symposium on Principles and practice of parallel programming*, pp. 73–82, 2008.

[4] V. W. Lee, C. Kim, J. Chhugani, M. Deisher, D. Kim, A. D. Nguyen, N. Satish, M. Smelyanskiy, S. Chennupaty, P. Hammarlund, R. Singhal, and P. Dubey, "Debunking the 100x GPU vs. CPU myth: An evaluation of throughput computing on cpu and gpu," in *Proceedings of the 37th Annual International Symposium on Computer Architecture*, ISCA '10, (New York, NY, USA), pp. 451–460, ACM, 2010.

[5] C. Pérez-Miguel, J. Miguel-Alonso, and A. Mendiburu, "Porting estimation of distribution algorithms to the cell broadband engine," *Parallel Computing*, vol. 36, no. 10, pp. 618–634, 2010.

[6] J. Diaz, C. Muñoz-Caro, and A. Niño, "A survey of parallel programming models and tools in the multi and many-core era," *IEEE Transactions on Parallel and Distributed Systems*, vol. 23, no. 8, pp. 1369–1386, 2012.

[7] U. Lopez-Novoa, A. Mendiburu, and J. Miguel-Alonso, "A survey of performance modeling and simulation techniques for accelerator-based computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, pp. 272–281, Jan 2015.

[8] J. Nickolls and W. Dally, "The GPU computing era," *IEEE Micro*, vol. 30, no. 2, pp. 56–69, 2010.

[9] J. D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Krüger, A. E. Lefohn, and T. J. Purcell, "A survey of general-purpose computation on graphics hardware," *Computer Graphics Forum*, vol. 26, no. 1, pp. 80–113, 2007.

[10] GPGPU Community. `http://www.gpgpu.org`, May 2015.

[11] C. Wittenbrink, E. Kilgariff, and A. Prabhu, "Fermi GF100 GPU architecture," *IEEE Micro*, vol. 31, no. 2, pp. 50–59, 2011.

[12] E. Lindholm, J. Nickolls, S. Oberman, and J. Montrym, "NVIDIA Tesla: A unified graphics and computing architecture," *IEEE Micro*, vol. 28, no. 2, pp. 39–55, 2008.

[13] A. Bayoumi, M. Chu, Y. Hanafy, P. Harrell, and G. Refai-Ahmed, "Scientific and engineering computing using ATI Stream technology," *Computing in Science and Engineering*, vol. 11, no. 6, pp. 92–97, 2009.

[14] L. Seiler, D. Carmean, E. Sprangle, T. Forsyth, M. Abrash, P. Dubey, S. Junkins, A. Lake, J. Sugerman, R. Cavin, R. Espasa, E. Grochowski, T. Juan, and P. Hanrahan, "Larrabee: a many-core x86 architecture for visual computing," *ACM Transactions on Graphics*, vol. 27, no. 3, pp. 18:1–18:15, 2008.

[15] J. Jeffers and J. Reinders, *Intel Xeon Phi coprocessor high performance programming*. Morgan Kaufman, 2013.

[16] A. Branover, D. Foley, and M. Steinman, "AMD Fusion APU: Llano," *IEEE Micro*, vol. 32, no. 2, pp. 28–37, 2012.

[17] E. Rotem, A. Naveh, D. Rajwan, A. Ananthakrishnan, and E. Weissmann, "Power-management architecture of the intel microarchitecture code-named sandy bridge," *IEEE Micro*, vol. 32, no. 2, pp. 20–27, 2012.

[18] J. Kahle, M. Day, H. Hofstee, C. Johns, T. Maeurer, and D. Shippy, "Introduction to the cell multiprocessor," *IBM journal of Research and Development*, vol. 49, no. 4.5, pp. 589–604, 2005.

[19] M. Daga, A. Aji, and W. Feng, "On the efficacy of a fused CPU + GPU processor (or APU) for parallel computing," in *IEEE Symposium on Application Accelerators in High-Performance Computing (SAAHPC), 2011*, pp. 141–149, 2011.

[20] K. L. Spafford, J. S. Meredith, S. Lee, D. Li, P. C. Roth, and J. S. Vetter, "The tradeoffs of fused memory hierarchies in heterogeneous computing architectures," in *Proceedings of the 9th Conference on Computing Frontiers*, CF '12, pp. 103–112, 2012.

[21] Khronos Group, *The OpenCL Specification*, 2008.

[22] NVIDIA, "CUDA Home Page." `http://developer.nvidia.com/cuda`, Nov. 2013.

[23] AccelerEyes, "clMath: An Open Source BLAS and FFT Library for OpenCL." `http://www.accelereyes.com/clmath`, Nov. 2013.

[24] M. Wolfe, "The OpenACC application programming interface, version 2.0," 2013.

[25] OpenMP Architecture Review Board, "OpenMP application program interface version 4.0," July 2013.

[26] M. Amini, B. Creusillet, S. Even, R. Keryell, O. Goubier, S. Guelton, J. McMahon, F. Pasquier, G. Péan, and P. Villalon, "Par4all: From convex array regions to heterogeneous computing," in *2nd International Workshop on Polyhedral Compilation Techniques (Impact), 2012*, 2012.

[27] B. W. Silverman, *Density estimation for statistics and data analysis*, vol. 26. Chapman & Hall/CRC, 1986.

[28] R. Andrés Ferreyra, G. P. Podestá, C. D. Messina, D. Letson, J. Dardanelli, E. Guevara, and S. Meira, "A linked-modeling framework to estimate maize production risk associated with enso-related climate variability in argentina," *Agricultural and Forest Meteorology*, vol. 107, no. 3, pp. 177–192, 2001.

[29] S. Corti, F. Molteni, and T. Palmer, "Signature of recent climate change in frequencies of natural atmospheric circulation regimes," *Nature*, vol. 398, no. 6730, pp. 799–802, 1999.

[30] R. Weissbach, "A general kernel functional estimator with general bandwidth-strong consistency and applications," *Journal of Nonparametric Statistics*, vol. 18, no. 1, pp. 1–12, 2006.

[31] A. Elgammal, R. Duraiswami, and L. Davis, "Efficient kernel density estimation using the fast gauss transform with applications to color modeling and tracking," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 25, no. 11, pp. 1499–1504, 2003.

[32] S. J. Sheather, "Density estimation," *Statistical Science*, pp. 588–597, 2004.

[33] D. W. Scott, *Multivariate density estimation: theory, practice, and visualization*, vol. 383. Wiley, 2009.

[34] S. Hong and H. Kim, "An analytical model for a GPU architecture with memory-level and thread-level parallelism awareness," *ACM SIGARCH Computer Architecture News*, vol. 37, no. 3, pp. 152–163, 2009.

[35] S. Williams, A. Waterman, and D. Patterson, "Roofline: an insightful visual performance model for multicore architectures," *Communications of the ACM*, vol. 52, no. 4, pp. 65–76, 2009.

[36] "GPGPU-Sim Online Manual." `http://gpgpu-sim.org/manual`, May 2015.

[37] K. Fukunaga, *Introduction to statistical pattern recognition (2nd ed.)*. San Diego, CA, USA: Academic Press Professional, Inc., 1990.

[38] T. Duong, "ks: Kernel density estimation and kernel discriminant analysis for multivariate data in r," *Journal of Statistical Software*, vol. 21, no. 7, pp. 1–16, 2007.

[39] B. V. Srinivasan, Q. Hu, and R. Duraiswami, "Gpuml: Graphical processors for speeding up kernel machines," in *Workshop on High Performance Analytics-Algorithms, Implementations, and Applications*, (Columbus, USA), May 2010.

[40] H. Goosse, P. Barriat, W. Lefebvre, M. Loutre, and V. Zunz, *Introduction Climate Dynamics and Climate Modelling*. Université catholique de Louvain, 2008.

[41] A. Otto, F. E. Otto, O. Boucher, J. Church, G. Hegerl, P. M. Forster, N. P. Gillett, J. Gregory, G. C. Johnson, R. Knutti, *et al.*, "Energy budget constraints on climate response," *Nature Geoscience*, vol. 6, no. 6, pp. 415–416, 2013.

[42] P. Braconnot, S. P. Harrison, M. Kageyama, P. J. Bartlein, V. Masson-Delmotte, A. Abe-Ouchi, B. Otto-Bliesner, and Y. Zhao, "Evaluation of climate models using palaeoclimatic data," *Nature Climate Change*, vol. 2, no. 6, pp. 417–424, 2012.

[43] Y. Sun, S. Solomon, A. Dai, and R. W. Portmann, "How often does it rain?," *Journal of Climate*, vol. 19, no. 6, pp. 916–934, 2006.

[44] S. Brands, J. M. Gutiérrez, S. Herrera, and A. Cofiño, "On the use of reanalysis data for downscaling," *Journal of Climate*, vol. 25, no. 7, pp. 2517–2526, 2012.

[45] S. Perkins, A. Pitman, N. Holbrook, and J. McAneney, "Evaluation of the ar4 climate models' simulated daily maximum temperature, minimum temperature, and precipitation over australia using probability density functions," *Journal of climate*, vol. 20, no. 17, pp. 4356–4376, 2007.

[46] I. Errasti, A. Ezcurra, J. Sáenz, G. Ibarra-Berastegi, and E. Zorita, "Comparison of the main characteristics of the daily zonally averaged surface air temperature as represented by reanalysis and seven cmip3 models," *Theoretical and Applied Climatology*, vol. 114, no. 3-4, pp. 417–436, 2013.

[47] A. Danalis, P. Luszczek, G. Marin, J. S. Vetter, and J. Dongarra, "Blackjackbench: Portable hardware characterization with automated results analysis," *The Computer Journal*, 2013.

Part II

Publications

5

A Survey of Performance Modeling and Simulation Techniques for Accelerator-Based Computing

A Survey of Performance Modeling and Simulation Techniques for Accelerator-Based Computing

Unai Lopez-Novoa, Alexander Mendiburu, and Jose Miguel-Alonso, *Member, IEEE Computer Society*

Abstract—The high performance computing landscape is shifting from collections of homogeneous nodes towards heterogeneous systems, in which nodes consist of a combination of traditional out-of-order execution cores and accelerator devices. Accelerators, built around GPUs, many-core chips, FPGAs or DSPs, are used to offload compute-intensive tasks. The advent of this type of systems has brought about a wide and diverse ecosystem of development platforms, optimization tools and performance analysis frameworks. This is a review of the state-of-the-art in performance tools for heterogeneous computing, focusing on the most popular families of accelerators: GPUs and Intel's Xeon Phi. We describe current heterogeneous systems and the development frameworks and tools that can be used for developing for them. The core of this survey is a review of the performance models and tools, including simulators, proposed in the literature for these platforms.

Index Terms—Accelerator-based computing, heterogeneous systems, GPGPU, performance modeling

1 Introduction

Accelerator devices are hardware pieces designed for the efficient computation of specific tasks or subroutines. These devices are attached to a Central Processing Unit (CPU) which controls the offloading of software fragments and manages the copying and retrieval of the data manipulated in the accelerator. Most accelerators show important architectural differences with respect to CPUs: the number of computing cores, the instruction sets and the memory hierarchy are completely different, making accelerators suitable for specific classes of computation.

In the early 2000s, the High Performance Computing (HPC) community began using Graphics Processing Units (GPUs) as accelerators for general purpose computations [1], coining the term *General Purpose Computing on GPUs* (GPGPU) [2]. GPUs are designed for the efficient manipulation of computer images, and each new generation of devices arrives with significantly higher horsepower in terms of FLOP/s [3], [4]. The HPC community soon became aware of this potential, devising smart tricks to disguise scientific computations as graphics manipulations. GPU manufacturers noticed this trend and provided Application Programming Interfaces (APIs) and Software Development Kits (SDKs) to allow a more direct programming of their devices for non-graphics tasks, while simultaneously designing the newer GPUs taking into consideration the needs of this growing market. Discrete accelerators built around GPU chips but without video connectors were produced and rapidly adopted by the HPC community. Soon, other hardware manufacturers started building dedicated accelerator devices, shipping them with programming environments similar to those developed initially for GPGPU in order to ease its adoption by HPC developers.

Effectively exploiting the theoretical performance of an accelerator is a challenging task, which often requires the use of new programming paradigms and tools. Porting an application to an accelerator means extensive program rewriting, only to achieve a preliminary, not really efficient implementation. Optimization is even more complex. This complexity is exacerbated when simultaneously trying to use the aggregated performance of a processor and the accelerator(s) attached to it, not to mention massively parallel systems with thousands of hybrid nodes. Several works expound that it is hard to efficiently use (homogeneous) massively parallel computing systems [5], and even harder to deal with heterogeneous, accelerator-based supercomputers [6].

An added difficulty is that porting codes to use accelerators could be useless if the target application does not fit into a massively data parallel model. We can find highly successful porting cases [7], which may compel programmers to jump onto the accelerator bandwagon. However, metrics about the effort required for the porting are not that common. Besides, the way of measuring the degree of success can be misleading. In particular, it is a common practice to measure speedup against a serial version of the code. Works such as [8] claim that achieved speedups are not that large if the yardsticks are multi-core fine-tuned applications.

A common pitfall when developing for accelerators is to carry out code implementation and tuning using a trial-and-error methodology, without appropriate feedback and

• *The authors are with the Intelligent Systems Group, Department of Computer Architecture and Technology, University of the Basque Country UPV/EHU. P. Manuel Lardizabal 1, 20018 Donostia-San Sebastian, Spain. E-mail: {unai.lopez, alexander.mendiburu, j.miguel}@ehu.es.*

Manuscript received 20 May 2013; revised 19 Dec. 2013; accepted 30 Dec. 2013. Date of publication 24 Feb. 2014; date of current version 5 Dec. 2014. Recommended for acceptance by S. Aluru.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TPDS.2014.2308216

guidance from performance tools. Unfortunately, in the field of accelerator-based computing there is no outstanding tool or model that can be considered as the reference instrument for performance prediction and tuning. There is, though, an extensive body of literature related to this (relatively) novel area. The main objective of this survey is precisely to compile, organize and analyze this literature. We are not aware of a work similar to this in terms of breadth. We are convinced that this work will be useful for developers that want to extract the best possible performance (or performance/power tradeoff) of the widely available accelerator-based platforms.

The remainder of this paper is organized as follows. In Section 2 we provide some background on hardware, development tools and modeling methodologies for accelerator-based computing. Section 3 is devoted to the review of the literature on performance and power models and simulators targeting accelerators. Finally, in Section 4 we provide some conclusions and discuss future research lines in this field.

2 Background

In this section we provide a summary of the background information required to understand the analysis of performance models carried out in Section 3, which is the core part of this survey. The reader is referred to Appendices A to E of the Supplementary File, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2014.2308216, for more details and additional references.

2.1 Accelerators and Heterogeneous Architectures

This survey focuses on heterogeneous hardware, in the form of (1) hybrid chips integrating several cores of different characteristics, including specific-purpose accelerators, and (2) traditional computing platforms to which a discrete accelerator is attached using, for example, PCIe. Nowadays, two classes of accelerators are mainstream: GPUs and many-cores such as Intel's Xeon Phi coprocessor. Other possibilities do exist, such as Field Programmable Gate Arrays (FPGAs) and Digital Signal Processors (DSPs).

Most of the literature on performance analysis for accelerator-based computing is devoted to analyze discrete GPUs and many-cores (specifically, the Xeon Phi), and therefore we do the same in this survey. Devices targeting GPGPU have been marketed since mid-2000s, while the Phi reached the market in 2012. This fact justifies the bias of the literature towards GPUs.

A GPU is an autonomous computing system composed of a set of processing cores and a memory hierarchy. A global memory space is accessible to all the cores, which can be exclusive (this is the common case if the GPU is a discrete accelerator plugged into a PCIe slot) or shared with other processing elements in the system, including the CPU (which is the common case when using a heterogeneous, multi-core chip). The strong point of GPUs is the way they handle thousands of in-flight active threads, and make context switching among them in a lightweight way. Running threads may stall when trying to access global memory, a relatively expensive operation. GPUs hide these latencies by rapidly context-switching stalled threads (actually, groups of threads) with active ones.

The Intel Xeon Phi is an accelerator that attaches to a host device using PCIe. It includes a many-core processor and memory. Current implementations incorporate up to 61×86 cores supporting four-way Simultaneous Multi-threading (SMT) and Single Instruction, Multiple Data (SIMD) capabilities.

Regarding heterogeneous chips, we pay special attention to those integrating a traditional multi-core processor and a GPU that, in addition to be useful for graphics, can be used for offloading compute-intensive program sections. The main advantage of these chips is that memory is shared by GPU and CPU, therefore avoiding the use of expensive PCIe transfers.

2.2 Development Tools for Accelerators

Applications for accelerators are generally written using software development tools provided by the device manufacturers. They can consist of manufacturer-specific tool chains, or can be implementations of standard APIs. OpenCL [9] is a standardized, vendor-neutral framework for programming all classes of accelerators, defining a hardware model and an API. CUDA [10] implements similar concepts, but it is specific for NVIDIA GPUs and uses a slightly different terminology. Both models are widely used for GPGPU.

Compared to GPUs, the Xeon Phi is a relative newcomer to the accelerator arena. However, it has inherited from Intel's vast experience with multi-core processing, and the result is the availability of a diverse collection of tools that makes the learning curve of the developer less steep. Among other APIs, Phi developers can use OpenMP, OpenCL and MPI [11].

It is important to stress that the use of standard APIs may provide code portability between different platforms, but this does not translate into performance portability: currently, device specific, performance-oriented fine-tuning of an application is required in order to fully exploit the capabilities of an accelerator.

2.3 Tools and Techniques for Performance Modeling

Tools such as profilers and debuggers, together with best practices manuals written by the manufacturers of accelerators, are of great help during the process of porting and tuning an application. However, they lack prediction abilities: they cannot estimate performance on a different platform, or for a new application. Additionally, these tools can overwhelm the programmer with excessive information, making it difficult to filter out what is actually limiting the performance.

Our focus in this paper is on models of parallel applications running on heterogeneous platforms, that can be used to analyze (and predict) their combined performance. These models require input data (the characteristics of the target application and platform), and may provide different sets of output data, such as performance estimations or a list of bottlenecks.

In this section we describe methods to characterize devices and applications in order to feed performance models. We also describe the different classes of models that have been proposed and used in the literature: analytical, based on machine learning (ML), and based on simulation. Once a model is proposed, it must be validated and evaluated in order to demonstrate its usefulness beyond a single application/platform pair. A common way of doing this validation is through benchmark applications.

2.3.1 Characterizing Devices

Static information about the characteristics of a particular device can be collected in several ways, including manufacturer data sheets and manuals. Often, a manufacturer-specific API is available to directly query the device. To gather dynamic information about the way an application is using hardware resources, performance counters can be used. Latest chips also provide counters to obtain data about temperature and power usage.

A complementary source of information is that obtained through micro-benchmarks, programs designed to stress a particular component of the hardware and return performance metrics about it. This technique has been used extensively for CPUs, and has been extended to accelerators.

2.3.2 Characterizing Applications

Application characterization is the task of describing a program (or kernel) using metrics that can be used to feed a performance model. The accuracy of the model will depend on the quality of the captured metrics, and the metrics to extract from a program depend on the needs of the model they will feed. Usually, metrics include memory access patterns, register utilization, number of arithmetic operations, number of branches, etc. Program metrics can be obtained using a variety of mechanisms, including:

- Analysis of the source code, directly or after some transformations.
- Analysis of an intermediate representation (IR) of the code, an assembly-like code representation produced by the compiler before the generation of the binary files. A drawback of this approach is that, as it relies on the potential optimizations that the compiler might have performed, it may not exactly represent the original source code. CUDA's PTX format is a commonly used IR, that can be analyzed using the Ocelot [12] tool set.
- Analysis (disassembling) of the final binary files. These are the ones executed by the devices, so they provide maximum information about what the hardware will do.

2.3.3 Modeling Methodologies

A performance model is designed with the aim of describing the behavior of a system. A good model has prediction abilities: providing as input descriptions of an application and a platform, it generates estimators of the performance that would be achieved. After a review of the literature, we have identified three main approaches to modeling the performance of parallel systems: analytical modeling, machine learning and simulation.

An analytical model is an abstraction of a system in the form of a set of equations [13]. These equations try to represent and comprise all (or most) of the characteristics of the system. Machine Learning [14] is a branch of artificial intelligence related to the study of relationships between empirical data in order to extract valuable information. These techniques learn and train a model, for example a classifier, based on known data. Later, this model can be used with new (unseen) data to classify it. Finally, simulators are tools designed to imitate the operation of a system [13]. They are able to mimic system behavior step by step, providing information about system dynamics. The level of accuracy of the output information of the simulator depends, as in the other approaches, on the level of detail introduced in the model, taking into account that it is not always easy to exactly identify the actual way a system behaves.

Independently of the nature of the models, most have a set of parameters that must be tuned to suit the particular scenario to which they will be applied. Parameter values may be provided by experts, or measured in previous experiments but, in general, models are trained with data obtained from scenarios similar to the target one. For example, a ML model designed to predict execution times of a particular application on a particular hardware platform with a particular configuration (number of threads, thread grouping, input data, etc.) may take as training data set the execution times of applications running in the same or similar hardware for a variety of different configurations. If the data set is large and representative enough, one could expect promising performance estimations for the target application/hardware combination. However, the model would probably provide poor results for different applications or hardware. Benchmark suites can help to fine tune models, as they usually integrate a collection of diverse applications aimed to represent actual workloads that exploit different hardware features.

Additionally, a model will not be widely accepted unless it has been previously evaluated and tested for accuracy. Benchmark suites can also be used for this purpose, providing well defined yardsticks, easier to be accepted by the community than ad-hoc applications that could be considered unrealistic or barely representative of actual workloads.

3 A Review of Performance Models

Roughly speaking, a performance model can be seen as a system representation which provides output values based on a given set of input parameters. Depending on the characteristics of the model and the goal it has been designed for, the input and output data sets can be notably different.

Performance models can be classified according to different criteria. We already provided a classification in analytical models, machine-learning models and simulators. However, in this section we use a different classification criterion, based on the output generated by the model, that is, the information it provides about a particular hardware/software combination. We have identified models designed to:

1) predict the execution time of a target application on a target hardware platform
2) identify performance bottlenecks and propose code modifications to avoid them
3) provide estimations of power consumption
4) provide detailed, step-by-step information of resource usage, based on simulation

In the forthcoming sections we make a review of model proposals found in the literature, arranged using this classification. In Appendix F of the Supplementary File, available online, we have included a table for each model class, summarizing the main features of each model.

3.1 Execution Time Estimation

Estimating the execution time of parallel applications can be useful in several situations, such as making a decision about acquiring new hardware or testing an application for scalability. After a literature review of performance models aiming to predict the execution time of an accelerator-based parallel application, we have organized the different works taking into account whether they propose general models applicable to any program/kernel, or they are specific for a particular application or pattern. We have also taken into account the input information required by the model: actual kernel codes, or pseudocodes/patterns derived from the program structure.

3.1.1 General-Purpose Models

Models using kernel codes. As previously explained, GPUs hide latencies by handling thousands of in-flight active threads, rapidly switching stalled warps (due to memory transactions) with ready-to-run warps. A *warp* is the CUDA name of a group of 32 threads, the minimum scheduling unit in NVIDIA GPUs. Hong and Kim [15] model this behavior on NVIDIA GPUs using two different metrics: MWP and CWP. The first one, memory warp parallelism, measures the maximum number of warps that can overlap memory accesses. The second, computation warp parallelism, measures the number of warps that execute instructions during one memory access period. Expressions are provided to compute these values for a given device and application and, from them, the number of cycles required to run the kernel. The basic idea behind the model is that, when $MWP \leq CWP$, performance is dominated by the time spent accessing memory but, when $MWP > CWP$, memory accesses are mostly hidden and overall performance is dominated by the computation cycles. A variety of kernels and GPUs were used to validate the model. Authors report a geometric mean error of 13.3 percent between estimated and actual cycles per instruction (CPI). Due to its simplicity and accuracy reflecting the way GPUs work, this model has been used and extended by many other authors. For instance, Ganestam and Doggett use MWP-CWP in [16] as part of an auto-tuning tool for the optimized execution of a ray-tracing algorithm. We will see some additional examples throughout this paper.

Based on similar concepts, Kothapalli et al. [17] propose two kernel behaviors, also in a CUDA context: MAX model (maximum value between compute and memory cycles), and SUM model (sum of compute and memory cycles). Authors claim that for most of the tested cases, predictions provided by these SUM and MAX models are close to the real measurements but, unfortunately, they do not provide clues as to how to choose the right one for a given kernel. An extension of this work, presented by Gonzalez [18], includes more complex hardware features (memory copy overheads, branch-divergence latencies, etc.), and improves modeling accuracy, choosing in most of the cases the SUM model. As in the original work, no reason is given for this decision.

In addition to these analytical models, there are also others which rely on machine learning techniques identifying first a set of code and hardware features, and later using feature selection, clustering and regression techniques to estimate execution times. Kerr et al. [19] present four different models: application, GPU, CPU, and combined $\mathrm{CPU} + \mathrm{GPU}$. The paper states that the GPU model (estimation of application execution times on an unknown GPU) provides fairly accurate results, with a maximum deviation of 16 percent in the worst case. Unfortunately, the remaining models are not that accurate, producing poorer estimates and high variability. The proposal by Che and Skadron [20] has accuracy levels close to those of the MWP-CWP model [15] when used to predict the execution time of different applications on the GPUs used to build the model. However, accuracy is not that good, although still reasonable (between 15.8 and 27.3 percent), when estimating execution time on an unknown, new GPU. Finally, the paper presented by Sato et al. [21] discusses different machine learning models, reporting for the best one error rates around 1 percent. However, the process used to calibrate the model is not detailed sufficiently.

Models using program skeletons. Given the (serial) source code of an application, it would be highly desirable to know its potential performance when running it in an accelerator. There are some performance-prediction frameworks that aim to do precisely this, although they do not take the code directly as an input: the user is required to provide a skeleton (abstract view) of the application using a set of constructors that identify parallelization opportunities. Grophecy, presented by Meng et al. [22], takes an application skeleton and generates several proposals for implementing the accelerated version of the code, with different types of optimizations, such as tiling and loop unrolling. The framework makes use of the above-described MWP-CWP model to estimate the execution time of each proposal and recommends the most promising one.

Nugteren and Corporaal propose in [23] the boat hull model, inspired by the well-known roofline model [24]. They claim that the roofline model is not designed to predict performance, but to help a programmer detect bottlenecks and improve performance (see Section 3.2.1). In contrast, the boat hull model does provide performance estimations using a few architectural parameters and a description of the code. The target program must be split into several sections, and each section must be characterized as belonging to a certain class. Then, a modified roofline model (performance graph) specific to each class is applied to the different sections, using in the X axis a "complexity" metric (instead of operations per byte), and in the Y axis an "execution time" metric (instead of FLOP/s). All together,

the model can be used to predict the performance of an application on different target architectures, including GPUs and other accelerators. The model is applied to an image processing application on two GPUs, and the difference between predicted and measured execution times is 3 percent in one device and 8 percent in the other.

The two proposals discussed in this section can be used to predict application performance without carrying out accelerator-specific implementations. However, users are required to describe application behavior in an abstract way, learning a new syntax and carefully splitting the code into sections that must be well-defined and properly associated to a collection of skeletons or classes. These are not trivial tasks, especially when handling large, complex pieces of code.

3.1.2 Models Designed for Particular Applications

The works discussed above can be considered general-purpose, that is, suitable to any application. We can find in the literature other proposals which focus on modeling specific programs or program classes (patterns). For example, Meng and Skadron propose in [25] a framework that automatically selects the best parameters for an Iterative Stencil Loops (ISL) application, given some parameters of the domain and some features of the target GPU. ISL is a technique applied in many domains, including molecular dynamics simulation and image processing, that distribute computations into overlapping regions (tiles), with neighboring regions interacting via halo zones. Similarly, Choi et al. present in [26] an auto-tuned implementation of the Sparse Matrix-Vector (SpMV) multiplication. The difficulty in computing with sparse matrices is related to using compression algorithms (such as BCSR or ELLPACK [27]) that represent matrix data as lists. Authors designed their own implementation of the Blocked ELLPACK (BELLPACK) algorithm and built it with a model that, given architectural features, finds the application parameters that minimize execution time.

3.2 Bottleneck Highlighting and Code Optimization

While a program is being ported and fine-tuned, it is necessary to carefully analyze its behavior at run time on the target platform, looking for potential resource bottlenecks and, if possible, finding alternatives to eliminate or mitigate them. Toolkits for programming accelerators commonly include a profiler (e.g., NVIDIA Visual Profiler for the CUDA platform, CodeXL for the AMD OpenCL platform or VTune for the Intel Xeon Phi platform), which should be the first tools to use when optimizing a kernel. They analyze code execution, spotting bottlenecks, and can even make recommendations to the programmer about code changes or compiler flags. However, they cannot predict the performance benefits associated to these changes. To this end, several models have been proposed in the literature with the aim of helping developers in code optimization tasks.

3.2.1 The Roofline Model

The work by Williams et al. [24] has provided a well-accepted mechanism to visually describe those characteristics of a multi-core platform that may limit the performance of a particular application. Using a diagram with two axes, representing an "Operational Intensity (FLOP/Byte)" metric in the X-axis, and an "Attainable GFLOP/s" metric in the Y-axis, the compute platform is represented by a line or *roof* that first grows linearly with the operational intensity, until it reaches an inflection point and then flattens. An application (or kernel) fits somewhere in the X-axis (has a certain operational intensity) and, consequently, can reach a maximum performance (limited by the line representing the platform). The programmer can try code optimizations aimed at increasing operational intensity, moving the kernel to the right side of the graph, until it reaches the flat section where further improvements are not fruitful.

The limits imposed by the platform may vary depending on the use of different features. For example, by using SIMD instructions or transcendental functional units, if available, peak performance (the flat portion) can be improved. Using memory accesses that make better use of memory channels (such as FastPath in AMD GPUs or data coalescing in NVIDIA GPUs) raises the growing section. Therefore, the platform is not characterized by a single line, but by a collection of lines. The particular roof limiting a given application depends on the use the programmer makes of the device features. In [28] Jia et al. make a roofline characterization of two GPUs: AMD HD5850 and NVIDIA C2050, showing how the set of features affecting performance is different for each GPU and, therefore, the code modifications programmers may apply in each case are also different. The roofline model has also been used by Cramer et al. to characterize the Xeon Phi running OpenMP codes [29].

3.2.2 Program Dissection

Models by Lai and Seznec [30], Zhang and Owens [31] and Baghsorkhi et al. [32] provide a breakdown of a CUDA kernel execution into several stages, such as compute phases or memory stalls, although they differ in the way they retrieve application-related features. In all cases, platform information is gathered via micro-benchmarks, while the application is characterized by code analysis. Both [30] and [31] require running the target kernel within a simulator to gather performance counters. The former also requires an analysis of the disassembled kernel binary file. In contrast, Baghsorkhi et al. [32] work with the source code of the kernel, building from it a program dependence graph (PDG) [33] that represents the workflow of the application.

Once the information characterizing device and application has been gathered, the models are constructed and used to generate different sorts of output (or performance predictions): Lai and Seznec [30] present statistics of some hardware features such as the level of contention for the scalar cores; Zhang and Owens [31] present a detailed breakdown of the memory instructions, showing the number of cycles spent in global or shared memory instructions, along with the number of bank conflicts for the shared memory; finally, Baghsorkhi et al. [32] derive from the PDG a breakdown of the instructions into compute instructions and memory accesses, pointing out the number of cycles spent in stages such as synchronization or branch divergence, as well as in the different memory stages.

Regarding accuracy values, Lai and Seznec [30] report an average error of 4.64 percent when predicting the number of cycles used by the kernels chosen for the experiments. Zhang and Owens [31] report a relative prediction error within 5-15 percent. The model by Baghsorkhi et al. [32] could be expected to be the least accurate one, because program information is obtained from the source code instead of from lower-level measurements. Authors do not report numerical values, but their experiments show that the model is quite accurate, highlighting the most time-consuming stage in each execution.

3.2.3 Detection of Specific Bottlenecks

When a profiler or some of the previously discussed models point to a particular issue hindering performance when a kernel is running on a device, hints are required as to how to overcome it. Several models have been proposed to characterize specific bottlenecks and, thus, could be of use in this context.

The work by Ramos and Hoefler [34] is focused on creating a performance model for the cache coherence protocols used in many-cores, assessing the ways they impact communication and synchronization. Authors show how their analytical model helps in defining highly-optimized parallel data exchanges. Using as target platform the Xeon Phi, their methodology is tested against vendor-provided primitives (OpenMP and MPI libraries by Intel), achieving up to $4.3\times$ faster communications.

Baghsorkhi et al. [35] present a memory hierarchy model for GPUs through a framework that predicts the efficiency of the memory subsystem. They predict the latency of main memory accesses, and the hits and misses in the L1 and L2 caches. This memory model is based on the hardware memory hierarchy of the Tesla C2050 card, and validations with this particular hardware provide good results: authors compare the predictions of their model against the hardware counters, obtaining average absolute errors of 3.4 percent for L1 read hit ratios, 1.9 percent for L2 read hit ratios and 0.8 percent for L2 write hit ratios.

Cui et al. [36] present a performance model focused on reducing the effects of control-flow divergence in GPU kernels. They propose a framework that retrieves a representation of a kernel workflow, from which different groupings and reorderings of threads are derived, choosing those minimizing divergence. Authors report speedups up to $3.19\times$ in their experiments in GPUs, but the method is tested with kernels that do not have strong inter-thread dependences or synchronization.

3.2.4 Selecting Code Optimizations

As stated before, manufacturers provide best practices manuals recommending different optimizations that can be applied to accelerate the execution times of kernels. Some of these optimizations can even be implemented automatically by compilers. For example, in Chapter 10 of [37] Rahman describes a methodology and some heuristics to find and optimize parallel code in the Xeon Phi. He provides a taxonomy of potential optimizations, relates the metrics that indicate the presence of bottlenecks and describes some good practices to remove them. However, choosing the right combination of optimizations is not trivial, because of possible negative interactions among them.

Sim et al. present in [38] two related contributions: an elaborated analytical model for GPU architectures, and a framework that suggests the most profitable combination of optimizations for a CUDA kernel in terms of performance. The model is an extension of MWP-CWP that takes into account many hardware features of recent GPUs that were not present in the original model. These improvements include considering the effect of the cache hierarchy, the presence of special function units (which, in NVIDIA GPUs, implement transcendental functions) and a more detailed effect of the instruction level/thread level parallelism of the kernels. The performance projection framework is a tool that points out the most promising optimizations for a given kernel. Authors claim that their model closely estimates the speedup of potential optimizations, although no numerical values are given. For a particular kernel they tested 44 combinations of different optimizations. The best one runs three times faster, compared to the baseline implementation.

3.3 Power Consumption Estimation

A trending topic in the HPC field is improving the power efficiency of computers. To that extent, significant effort is being devoted to modeling the power characteristics of systems. Application power modeling aims to estimate the energy required to run a particular code on a particular device. It has to consider not only code and device properties, but also program input and some other runtime characteristics. In this section we review a collection of proposals aiming to model power efficiency.

3.3.1 Standalone Models

Wang and Ranganathan [39] and Hong and Kim [40] propose analytical power models. Both require the execution of a set of micro-benchmarks and external power consumption meters to characterize the target devices (in both cases, NVIDIA GPUs). Both use PTX intermediate representation ([40] processes it through Ocelot) to obtain information about the target kernel. The models estimate the total power consumed by the kernel execution, and are aimed to identify the right number of multiprocessors and blocks that would provide the best performance/power ratio. It is worth mentioning that the proposal by Hong and Kim is an extension of their MWP-CWP performance model (discussed in paragraph "Models using kernel codes" in Section 3.1.1), and that model validations carried out by them report a geometric mean error of 8.94 percent between estimated and actual power consumption.

In a similar fashion, the work by Shao and Brooks [41] presents an energy model of the Xeon Phi based on information gathered from performance counters and on measurements by an external meter. Through microbenchmarking they characterize the energy consumed by each instruction type (scalar, vector, prefetch) with operands in different locations (registers, L1 cache, global memory, etc.), compiling an energy-per-instruction table. The model is validated through a collection of benchmarks, reporting 1-5 percent discrepancies between predicted and actual power use. This work was carried out

with a pre-release of the Xeon Phi. The current marketed product implements power-related counters that can be used to avoid instrumenting the accelerator to perform power metering.

Other proposals gather performance and power measurements and use this data to build machine learning models, see for example [42], [43], [44], [45]. Samples of performance counters and energy measurements are taken at a given frequency while the kernel runs on the device, characterizing power use along time. This allows identifying power and performance bottlenecks, as well as efficient power/performance configurations. A main drawback of these approaches is that performance counters are collected per warp/GPU multiprocessor, and therefore the models assume that the applications use all the computing resources in a GPU. The most recent of these works [45] uses the NVIDIA Fermi architecture, which implements power-related hardware counters. This particular model is based on artificial neural networks (ANNs) and reports quite accurate estimations (2.1 percent error rate when predicting power consumption and 6.7 perecnt when predicting execution time).

3.3.2 Models Tied to Simulators

System simulators, such as those that will be discussed in the forthcoming section, can also be enhanced to provide not only performance-related metrics, but also power-related estimations. To that extent, a power model has to be integrated in the simulator, to allow it to compute cycle-by-cycle use of resources, together with the power implications of that use.

We have identified two analytical models that work with GPGPU-Sim (see Section 3.4): GPUWattch by Leng et al. [46], and GPUSimPow by Lucas et al. [47]. Both are adaptations to GPUs of the McPat power modeling framework for multi-core architectures [48], but differ in the way GPU architecture and power use is modeled. In terms of accuracy, comparing model predictions with power use in actual GPUs, both are similar (using the data provided by the authors): for GPUWattch the error is 9.9-13.4 percent, and for GPUSimPow it is 11.7-10.8 percent. An interesting feature of GPUWattch is a module that can be used by GPGPU-Sim to simulate power-related runtime optimizations, such as emulating dynamic voltage and frequency scaling (DVFS). Authors claim that using DVFS, GPU energy consumption could be reduced up to 14.4 percent, with less than 3 percent of performance loss.

3.4 Simulation

A simulator is a system representation (model) able to mimic, step-by-step, the behavior of the target (real) system. Simulators are widely used to carry out performance studies of existing hardware and software platforms, and also to analyze platforms that either do not exist, or are not available. The accuracy of the output information provided by a simulator depends on many factors, among them being the level of detail with which the system has been modeled and the quality and detail of the workloads provided to feed the model.

3.4.1 Simulators of GPU-Based Accelerators

Two popular simulators for GPU-based accelerators are GPGPU-Sim [49], [50] and Barra [51], [52]. Both are functional simulators of NVIDIA GPUs, capable of running CUDA codes (GPGPU-Sim works also with OpenCL). A collection of user-configurable parameters is required in order to fine-tune the way the target device is modeled: number and features of multiprocessors, interconnection network and its topology, memory size and organization, etc. As output, the simulators generate detailed step-by-step information about performance counters, memory accesses, resource utilization, etc.

Regarding simulation accuracy, authors state that GPGPU-Sim v3.1.0 reaches an accuracy of 98.3 percent for the instructions per cycle metric when simulating a NVIDIA GT-200 card and 97.3 percent for a Fermi card. For Barra, authors carry out some experiments comparing simulator-predicted values with hardware counters and, in most cases, discrepancies range between 0 and 23 percent. Only for one benchmark and a particular counter (measuring memory transactions), the discrepancy reaches 81.58 percent.

One of the weakest points of simulators is their execution speed: depending on the level of simulation detail, slowdown (comparison against execution on actual hardware) can be severe. GPGPU-Sim provides two ways of working: a fast functional way, which only executes the application and generates its output, and a detailed way (5 to 10 times slower), which collects performance statistics. However, the latter is the really useful mode: the fast one should be used only to verify that the code being analyzed runs in the GPGPU-Sim environment. Barra has been designed as a multi-core enabled program and, therefore, if its execution is slow, it can be accelerated by adding more cores: reported experiments show excellent scalability for up to four cores.

The flexibility of simulators, and the rich and detailed information they provide, have made them a tool of choice to feed other models. They can be used also as a complement to profilers, in order to drive performance optimizations (see for example [53], [54]).

3.4.2 Simulators of Hybrid Architectures

The simulators described in the previous section focus on kernels running on an accelerator (GPU) device. Other simulators go a step further, modeling hybrid CPU+GPU architectures that run heterogeneous applications, either as separate devices (CPU connected to GPU via PCIe, with separate RAMs) or as a fused chip (GPU+CPU, sharing the RAM). Three simulators in this class are MacSim [55], FusionSim [56] and Multi2Sim [57]. In order to narrow down the description, we will focus on the last one, which is under active development and whose features increase with each release. Currently, Multi2Sim supports simulations of systems integrating x86, MIPS-32 and ARM CPUs; AMD and NVIDIA GPUs; a memory hierarchy (per device and/or shared) and an interconnection network. It is possible to run a detailed, architectural simulation that mimics the use of hardware resources, but also a faster emulator that just replicates the behavior of program instructions. Tools are provided to pre-process the target applications in

order to run them in the simulated environment, with support for CUDA and OpenCL. A visual tool is provided to interactively follow (and pause/analyze/resume) a simulation. When assessing Muti2Sim accuracy, authors report errors between 7 and 30 percent estimating the execution cycles of OpenCL kernels on an AMD GPU. The large discrepancies are due to the lack of fidelity in the way the memory subsystem is simulated.

4 Conclusions

The appearance of new computing devices and the design of new algorithms in different fields of science and technology is forcing a fast evolution of HPC. Designing and developing programs that use currently available computing resources efficiently is not an easy task. As stated in [58], present-day parallel applications differ from traditional ones, as they have lower instruction-level parallelism, more challenging branches for the branch-predictors and more irregular data access patterns. Simultaneously, we are observing a processor evolution towards heterogeneous many-cores. The goal of these architectures is twofold: providing an unified address space that eliminates the need to interchange data with an external accelerator using a system interconnect and improving power efficiency reducing the total transistor count.

Tools to help programmers in this parallel and heterogeneous environment (debuggers, profilers, etc.) are slowly becoming available, but the programmer still needs to have an in-depth knowledge of the devices in which applications will run if performance and efficiency have to be taken into consideration. The performance models that have been proposed in the last years, and that we have tried to characterize in this review, aim to make the process of choosing the right options (device, program settings, optimizations, etc.) easier in order to efficiently run accelerator-based programs. From all the tools analyzed in this paper, a few of them stand out: MWP-CWP [15] as the model of choice to predict execution times on GPUs; the roofline model [24] to determine the factor limiting program performance in parallel systems (including accelerators); and the GPGPU-Sim simulator [49] of GPU-based accelerators, that can be enhanced with a power model. All currently available models have limitations, but they will be the foundation on which better tools will be constructed. In the following list we analyze some of these limitations, and different ways to overcome them.

- There is no accurate model valid for a wide set of architectures. Each model finds a different tradeoff between being more device-specific and therefore more accurate (e.g., [38]), or being more general purpose at the cost of losing accuracy (e.g., [23]). Related to this topic is the fast pace at which manufacturers market new products, with new or improved features, making models obsolete in a very short time. Performance models should be flexible enough to allow characterizing new devices.
- A majority of the models discussed in this review have been designed for CUDA, the most mature development environment for GPGPU. However the vendor neutrality of OpenCL and its availability for non-GPU accelerators is increasing its adoption by HPC programmers. In the past, OpenCL tools produced less efficient codes than their CUDA counterparts, but this is no longer true with the most current versions of OpenCL SDKs.
- Society is becoming aware of the great monetary and environmental costs derived from the high energy consumption of computing systems. The challenge is not only to squeeze the maximum performance out of a system, but also to do it with the minimum power. As often these two requirements cannot be optimized simultaneously, good tradeoffs have to be found. The power models reviewed in this paper can help to solve this bi-objective optimization problem.
- Reviewed models focus on outsourcing compute-intensive tasks to accelerator devices. This usually means leaving the CPU idle while the accelerator is busily crunching numbers. It is possible, however, to make both work simultaneously, significantly increasing system efficiency. There are proposals dealing with this workload distribution. Some of them divide the data to be processed into several chunks, obtaining performance measurements and, based on them, adjusting the optimal number of chunks to assign to each type of core [59]. Other authors run several benchmarks in the different compute resources using different balancing configurations, and use machine learning techniques to train a model to be able to predict the optimal chunk distribution for new applications [60]. Energy can also be included into the equation, making power-aware load balancing across heterogeneous systems [61].

As a final remark, we would like to point out an important difficulty we have found when crafting this survey: the lack of sufficient detail in different parts of the reviewed papers. For example, some authors neither indicate exactly which program features are needed by their models, nor the way used to obtain them. The methodologies used to test the accuracy of the models often lack detail too: proper testing methods require wide and representative data sets combined with state-of-the-art accuracy estimation techniques (such as cross-validation or bootstrapping) [62], taking care of not using test data in any step of the model creation process. Often, details about the exact accuracy estimation technique used with the models are missing: if this estimation is not performed in an sound way [63], models tend to over fit, providing (unrealistic) high levels of accuracy for the data used for training, but poor estimates for new, unseen data. Finally, it would be beneficial for the community to have access to models, tools and benchmarks, in order to cross-check results, to validate the models against applications not used by the developers, to test the viability of model variations, etc.

Acknowledgments

This work has been supported by the Saiotek and Research Groups (IT-609-13) programs (Basque Government), TIN2010-14931 (Ministry of Science and Technology) and COMBIOMED Network in Computational Biomedicine

(Carlos III Health Institute). U. Lopez is supported by a doctoral grant from the Basque Government. J. Miguel and A. Mendiburu are members of the HiPEAC European Network of Excellence.

REFERENCES

[1] J.D. Owens, D. Luebke, N. Govindaraju, M. Harris, J.Krüger, A.E. Lefohn, and T.J. Purcell, "A Survey of General-Purpose Computation on Graphics Hardware," *Computer Graphics Forum*, vol. 26, no. 1, pp. 80-113, 2007.

[2] GPGPU Community, http://www.gpgpu.org, Nov. 2013.

[3] J. Nickolls and W. Dally, "The GPU Computing Era," *IEEE Micro*, vol. 30, no. 2, pp. 56-69, Mar./Apr. 2010.

[4] C. Wittenbrink, E. Kilgariff, and A. Prabhu, "Fermi GF100 GPU Architecture," *IEEE Micro*, vol. 31, no. 2, pp. 50-59, Mar. 2011.

[5] J. Diaz, C. Munoz-Caro, and A. Nino, "A Survey of Parallel Programming Models and Tools in the Multi and Many-Core Era," *IEEE Trans. Parallel and Distributed Systems*, vol. 23, no. 8, pp. 1369-1386, Aug. 2012.

[6] R. Vuduc, A. Chandramowlishwaran, J. Choi, M. Guney, and A. Shringarpure, "On the Limits of GPU Acceleration," *Proc. Second USENIX Conf. Hot Topics in Parallelism (HotPar '10)*, pp. 13-13, 2010.

[7] S. Ryoo, C. Rodrigues, S. Baghsorkhi, S. Stone, D. Kirk, and W. Hwu, "Optimization Principles and Application Performance Evaluation of a Multithreaded GPU Using CUDA," *Proc. 13th ACM SIGPLAN Symp. Principles and Practice of Parallel Programming*, pp. 73-82, 2008.

[8] V. Lee et al., "Debunking the 100x GPU vs. CPU Myth: An Evaluation of Throughput Computing on CPU and GPU," *ACM SIGARCH Computer Architecture News*, vol. 38, no. 3, pp. 451-460, 2010.

[9] Khronos Group, *The OpenCL Specification*, 2008.

[10] NVIDIA, CUDA Home Page, http://developer.nvidia.com/cuda, Nov. 2013.

[11] J. Jeffers and J. Reinders, *Intel Xeon Phi Coprocessor High Performance Programming*. Morgan Kaufman, 2013.

[12] G.F. Diamos, A.R. Kerr, S. Yalamanchili, and N. Clark, "Ocelot: A Dynamic Optimization Framework for Bulk-Synchronous Applications in Heterogeneous Systems," *Proc. 19th Int'l Conf. Parallel Architectures and Compilation Techniques*, pp. 353-364, 2010.

[13] R. Jain, *The Art of Computer Systems Performance Analysis*. Wiley, 1991.

[14] T.M. Mitchell, *Machine Learning*. first ed., McGraw-Hill, Inc., 1997.

[15] S. Hong and H. Kim, "An Analytical Model for a GPU Architecture with Memory-Level and Thread-Level Parallelism Awareness," *ACM SIGARCH Computer Architecture News*, vol. 37, no. 3, pp. 152-163, 2009.

[16] P. Ganestam and M. Doggett, "Auto-Tuning Interactive Ray Tracing Using an Analytical GPU Architecture Model," *Proc. Fifth ACM Ann. Workshop General Purpose Processing with Graphics Processing Units*, pp. 94-100, 2012.

[17] K. Kothapalli, R. Mukherjee, M. Rehman, S. Patidar, P. Narayanan, and K. Srinathan, "A Performance Prediction Model for the CUDA GPGPU Platform," *Proc. IEEE Int'l Conf. High Performance Computing (HiPC)*, pp. 463-472, 2009.

[18] C.Y. Gonzalez, "Modelo de estimación de rendimiento para arquitecturas paralelas heterogéneas," master's thesis, Univ. Politécnica de Valéncia, 2013.

[19] A. Kerr, G. Diamos, and S. Yalamanchili, "Modeling GPU-CPU Workloads and Systems," *Proc. Third Workshop General-Purpose Computation on Graphics Processing Units*, pp. 31-42, 2010.

[20] S. Che and K. Skadron, "BenchFriend: Correlating the Performance of GPU Benchmarks," *Int. J. High Performance Computing Applications*, DOI: 10.1177/1094342013507960, http://hpc.sagepub.com/content/early/2013/10/11/1094342013507960, 2013.

[21] K. Sato, K. Komatsu, H. Takizawa, and H. Kobayashi, "A History-Based Performance Prediction Model with Profile Data Classification for Automatic Task Allocation in Heterogeneous Computing Systems," *Proc. IEEE Ninth Int'l Symp. Parallel and Distributed Processing with Applications (ISPA)*, pp. 135-142, 2011.

[22] J. Meng, V.A. Morozov, K. Kumaran, V. Vishwanath, and T.D. Uram, "GROPHECY: GPU Performance Projection from CPU Code Skeletons," *Proc. ACM Int'l Conf. for High Performance Computing, Networking, Storage and Analysis(SC '11)*, pp. 14:1-14:11, 2011.

[23] C. Nugteren and H. Corporaal, "The Boat Hull Model: Enabling Performance Prediction for Parallel Computing Prior to Code Development," *Proc. ACM Ninth Conf. Computing Frontiers*, pp. 203-212, 2012.

[24] S. Williams, A. Waterman, and D. Patterson, "Roofline: An Insightful Visual Performance Model for Multicore Architectures," *Comm. ACM*, vol. 52, no. 4, pp. 65-76, 2009.

[25] J. Meng and K. Skadron, "Performance Modeling and Automatic Ghost Zone Optimization for Iterative Stencil Loops on GPUs," *Proc. 23rd ACM Int'l Conf. Supercomputing*, pp. 256-265, 2009.

[26] J. Choi, A. Singh, and R. Vuduc, "Model-Driven Autotuning of Sparse Matrix-Vector Multiply on GPUs," *ACM Sigplan Notices*, vol. 45, no. 5, pp. 115-126, 2010.

[27] J. Rice and R. Boisvert, "*Solving Elliptic Problems Using Ellpack*," *Springer Series in Computational Math*, vol. 1, Springer, 1985.

[28] H. Jia, Y. Zhang, G. Long, J. Xu, S. Yan, and Y. Li, "GPURoofline: A Model for Guiding Performance Optimizations on GPUs," *Proc. 18th Int'l Conf. Parallel Processing (Euro-Par '12)*, vol. 7484, pp. 920-932, 2012.

[29] T. Cramer, D. Schmidl, M. Klemm, and D. an Mey, "OpenMP Programming on Intel Xeon Phi Coprocessors: An Early Performance Comparison," *Proc. Many-Core Applications Research Community Symp. at RWTH Aachen Univ.*, pp. 38-44, 2012.

[30] J. Lai and A. Seznec, "Break Down GPU Execution Time With an Analytical Method," *Proc. 2012 Workshop Rapid Simulation and Performance Evaluation: Methods & Tools*, pp. 33-39.

[31] Y. Zhang and J. Owens, "A Quantitative Performance Analysis Model for GPU Architectures," *Proc. IEEE 17th Int'l Symp. High Performance Computer Architecture (HPCA)*, pp. 382-393, 2011.

[32] S. Baghsorkhi, M. Delahaye, S. Patel, W. Gropp, and W. Hwu, "An Adaptive Performance Modeling Tool for GPU Architectures," *ACM SIGPLAN Notices*, vol. 45, no. 5, pp. 105-114, 2010.

[33] J. Ferrante, K. Ottenstein, and J. Warren, "The Program Dependence Graph and Its Use in Optimization," *ACM Trans. Programming Languages and Systems*, vol. 9, no. 3, pp. 319-349, 1987.

[34] S. Ramos and T. Hoefler, "Modeling Communication in Cache-Coherent SMP Systems: A Case-Study with Xeon Phi," *Proc. 22nd Int'l Symp. High-Performance Parallel and Distributed Computing*, pp. 97-108, 2013.

[35] S. Baghsorkhi, I. Gelado, M. Delahaye, and W. Hwu, "Efficient Performance Evaluation of Memory Hierarchy for Highly Multithreaded Graphics Processors," *Proc. 17th ACM SIGPLAN Symp. Principles and Practice of Parallel Programming*, pp. 23-34, 2012.

[36] Z. Cui, Y. Liang, K. Rupnow, and D. Chen, "An Accurate GPU Performance Model for Effective Control Flow Divergence Optimization," *Proc. IEEE 26th Int'l Parallel Distributed Processing Symp. (IPDPS)*, pp. 83-94, 2012.

[37] R. Rahman, *Intel Xeon Phi Coprocessor Architecture and Tools: The Guide for Application Developers*. Apress, http://books.google.es/books?id=pQV2nAEACAAJ, 2013.

[38] J. Sim, A. Dasgupta, H. Kim, and R. Vuduc, "A Performance Analysis Framework for Identifying Potential Benefits in GPGPU Applications," *Proc. 17th ACM SIGPLAN Symp. Principles and Practice of Parallel Programming*, pp. 11-22, 2012.

[39] Y. Wang and N. Ranganathan, "An Instruction-Level Energy Estimation and Optimization Methodology for GPU," *Proc. IEEE 11th Int'l Conf. Computer and Information Technology (CIT)*, pp. 621-628, 2011.

[40] S. Hong and H. Kim, "An Integrated GPU Power And Performance Model," *ACM SIGARCH Computer Architecture News*, vol. 38, no. 3, pp. 280-289, 2010.

[41] Y.S. Shao and D. Brooks, "Energy Characterization and Instruction-Level Energy Model of IntelÕs Xeon Phi Processor," *Proc. ACM/IEEE Int'l Symp. on Low Power Electronics and Design (ISLPED)*, 2013.

[42] X. Ma, M. Dong, L. Zhong, and Z. Deng, "Statistical Power Consumption Analysis and Modeling for GPU-Based Computing," *Proc. ACM SOSP Workshop on Power Aware Computing and Systems (HotPower)*, 2009.

[43] X. Ma, M. Rincon, and Z. Deng, "Improving Energy Efficiency of GPU Based General-Purpose Scientific Computing through Automated Selection of Near Optimal Configurations," University of Houston, Technical Report UH-CS-11-08 2011.

[44] H. Nagasaka, N. Maruyama, A. Nukada, T. Endo, and S. Matsuoka, "Statistical Power Modeling of GPU Kernels Using Performance Counters," *Proc. Green Computing Conf.*, pp. 115-122, 2010.

[45] S. Song, C. Su, B. Rountree, and K.W. Cameron, "A Simplified and Accurate Model of Power-Performance Efficiency on Emergent GPU Architectures," *Proc. IEEE 27th Int'l Symp. Parallel and Distributed Processing (IPDPS)*, pp. 673-686, 2013.
[46] J. Leng, T. Hetherington, A. ElTantawy, S. Gilani, N.S. Kim, T.M. Aamodt, and V.J. Reddi, "GPUWattch: Enabling Energy Optimizations in GPGPUs," *Proc. 40th Ann. Int'l Symp. Computer Architecture*, 2013.
[47] J. Lucas, S. Lal, M. Andersch, M. Alvarez Mesa, and B. Juurlink, "How a Single Chip Causes Massive Power Bills—Gpusimpow: A GPGPU Power Simulator," *Proc. IEEE Int'l Symp. Performance Analysis of Systems and Software (ISPASS)*, pp. 97-106, 2013.
[48] S. Li, J.H. Ahn, R.D. Strong, J.B. Brockman, D.M. Tullsen, and N.P. Jouppi, "McPAT: An Integrated Power, Area, and Timing Modeling Framework for Multicore and Manycore Architectures," *Proc. 42nd Ann. IEEE/ACM Int'l Symp. Microarchitecture (MICRO)*, pp. 469-480, 2009.
[49] GPGPU-Sim Online Manual, http://gpgpu-sim.org/manual/index.php5/GPGPU-Sim_3.x_Manual/, Apr. 2013.
[50] A. Bakhoda, G. Yuan, W. Fung, H. Wong, and T. Aamodt, "Analyzing CUDA Workloads Using a Detailed GPU Simulator," *Proc. IEEE Int'l Symp. Performance Analysis of Systems and Software (ISPASS '09)*, pp. 163-174, 2009.
[51] S. Collange, M. Daumas, D. Defour, and D. Parello, "Barra: A Parallel Functional Simulator for GPGPU," *Proc. IEEE Int'l Symp. Modeling, Analysis & Simulation of Computer and Telecomm. Systems (MASCOTS)*, pp. 351-360, 2010.
[52] Barra GPU Simulator, https://code.google.com/p/barra-sim/, Apr. 2013..
[53] N. Goswami, R. Shankar, M. Joshi, and T. Li, "Exploring GPGPU Workloads: Characterization Methodology, Analysis and Microarchitecture Evaluation Implications," *Proc. IEEE Int'l Symp. Workload Characterization (IISWC)*, pp. 1-10, 2010.
[54] N. Brunie, S. Collange, and G. Diamos, "Simultaneous Branch and Warp Interweaving For Sustained GPU Performance," *SIGARCH Computer Architecture News*, vol. 40, no. 3, pp. 49-60, 2012.
[55] Gatech, Macsim Simulator for Heterogeneous Architecture, http://code.google.com/p/macsim/, Jan. 2012.
[56] V. Zakharenko, "FusionSim: Characterizing the Performance Benefits of Fused CPU/GPU Systems," PhD dissertation, Univ. Toronto, 2012.
[57] R. Ubal, B. Jang, P. Mistry, D. Schaa, and D. Kaeli, "Multi2Sim: A Simulation Framework for CPU-GPU Computing," *Proc. 21st Int'l Conf. Parallel Architectures And Compilation Techniques*, pp. 335-344, 2012.
[58] M. Arora, S. Nath, S. Mazumdar, S. Baden, and D. Tullsen, "Redefining the Role of the CPU in the Era of CPU-GPU Integration," *IEEE Micro*, vol. 32, no. 6, pp. 4-16, Nov./Dec. 2012.
[59] M.E. Belviranli, L.N. Bhuyan, and R. Gupta, "A Dynamic Self-Scheduling Scheme for Heterogeneous Multiprocessor Architectures," *ACM Trans. Architecture and Code Optimization*, vol. 9, no. 4, p. 57, 2013.
[60] D. Grewe and M. O' Boyle, "A Static Task Partitioning Approach for Heterogeneous Systems Using OpenCL," *Proc. 20th Int'l Conf. Compiler Construction: Part of the Joint European Conf. Theory and Practice of Software*, pp. 286-305, 2011.
[61] M. Rofouei, T. Stathopoulos, S. Ryffel, W. Kaiser, and M. Sarrafzadeh, "Energy-Aware High Performance Computing with Graphic Processing Units," *Proc. ACM SOSP Workshop Power Aware Computing and Systems (HotPower)*, 2008.
[62] N. Japkowicz and M. Shah, *Evaluating Learning Algorithms: A Classification Perspective*. Cambridge Univ. Press, 2011.
[63] J. Reunanen, "Overfitting in Making Comparisons between Variable Selection Methods," *J. Machine Learning Research*, vol. 3, pp. 1371-1382, Mar. 2003.

Unai Lopez-Novoa received the MSc degree in computer science from the University of Deusto in 2010. He is currently working toward the PhD degree at the Department of Computer Architecture and Technology of the University of the Basque Country UPV/EHU. His main research interests include parallel and distributed computing and specially, massively parallel computing using accelerators.

Alexander Mendiburu received the BS degree in computer science and the PhD degree from the University of the Basque Country UPV/EHU, Gipuzkoa, Spain, in 1995 and 2006, respectively. He has been a lecturer since 1999, and an associate professor since 2008 in the Department of Computer Architecture and Technology, UPV/EHU. His main research interests include evolutionary computation, probabilistic graphical models, and parallel computing. He has published 15 papers in ISI peer-reviewed journals and more than 20 papers in international and national conferences.

Jose Miguel-Alonso received the MS and PhD degrees in computer science from the University of the Basque Country (UPV/EHU), Gipuzkoa, Spain, in 1989 and 1996, respectively. He is currently a full professor at the Department of Computer Architecture and Technology of the UPV/EHU. Prior to this, he was a visiting assistant professor at Purdue University for a year. He teaches different courses, at graduate and undergraduate levels, related to computer networking and high-performance and distributed systems, and has supervised (and currently supervises) several PhD students working on these topics. He is a member of the IEEE Computer Society.

Supplementary File of the TPDS Article:

A Survey of Performance Modeling and Simulation Techniques for Accelerator-based Computing

Unai Lopez-Novoa, Alexander Mendiburu, and Jose Miguel-Alonso, *Member, IEEE Computer Society*

Abstract—This Supplementary File contains the supporting materials of the TPDS paper: *A Survey of Performance Modeling and Simulation Techniques for Accelerator-based Computing*. It provides background information on accelerator hardware, developing tools and APIs, methods to characterize devices and applications, and a review of benchmark suites targeting accelerators. It also includes summary tables with the most relevant features of the models and simulators analyzed in the main body of the paper.

APPENDIX A
ACCELERATORS AND HETEROGENEOUS ARCHITECTURES

In this appendix we review the state-of-the-art devices used for accelerator-based computing. We have summarized in Figure 1 a time line of the main hardware devices developed by different manufacturers. A dashed line indicates the absence of a product family, while a shortened line indicates the discontinuation of a product. Brief descriptions of the devices mentioned in the figure are given throughout the different sections of this appendix.

A.1 Graphics Processing Units

GPUs are processors originally intended for the rendering of 2D and 3D images. Currently the main GPU manufacturers developing HPC-class products are NVIDIA and AMD (who bought ATI in 2006). Figure 1 shows the evolution of their GPU product families. Both AMD and NVIDIA provide desktop and server versions of their cards. Desktop cards are marketed as graphic accelerators that can also be used for GPGPU, running punctual workloads at full performance. Server cards are marketed as GPU-based accelerators, designed to run intensive workloads in an uninterrupted way.

There are other players in the GPU field. Not included in the figure is ARM [1], designer of a line of GPUs called Mali, targeting the embedded and mobile market: they are designed to accelerate graphics in low-power portable devices. Its design is radically different from those of AMD and NVIDIA, using a simpler memory hierarchy and a smaller number of processing units. In 2012, ARM announced the second generation of the Mali family, the T-600 series [2], which integrates support for GPGPU.

In essence, a GPU is an autonomous compute system composed of a set of processing cores and a memory hierarchy. A global memory space is accessible to all the cores, which can be exclusive (this is the common case if the GPU is a discrete accelerator plugged into a PCI-Express slot) or shared with other processing elements in the system, including the CPU (which is the common case when using a heterogeneous, multi-core chip). Discrete NVIDIA and AMD server-class accelerators (Tesla [3] and FireStream systems [4] respectively) include up to 8 GB of global memory.

In terms of processing elements, GPUs are arranged as a set of multiprocessors, each of them holding, among other components, several computing cores, a set of registers and some shared memory for inter-core communication. Common additional components include double-precision floating-point units, functional units for transcendental operations, and a cache hierarchy. The components and their interconnection vary depending on the manufacturer and card model. We refer the interested reader to [5], where authors make an in-depth comparison between the architecture of NVIDIA and AMD GPU families.

The strong point of GPUs is the way they handle thousands of in-flight active threads, and make context switching among them in a lightweight way. Running threads may stall when trying to access global memory, a relatively expensive operation. GPUs hide these latencies by rapidly context-switching stalled threads (actually, groups of threads) with active ones.

A.2 The Intel Xeon Phi coprocessor

Intel has recently developed a many-core architecture called MIC (Many Integrated Core) that has become a heavyweight player in the accelerator arena, as demonstrated by the prominent positions held by Phi-based supercomputers in the Top500 list [6].

A device with a preliminary version of the MIC architecture was Larrabee, a PCIe accelerator composed of

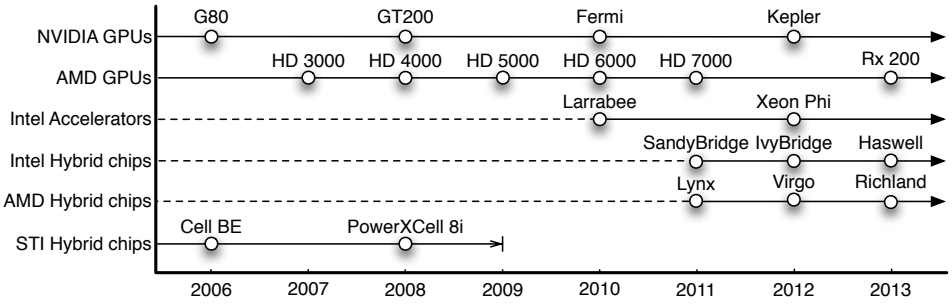


Fig. 1. Timeline of accelerator/hybrid devices

32 x86 cores. It was announced in 2008 [7], and some prototype cards were shipped in 2010. In 2012, Intel commercially launched the Xeon Phi Coprocessor [8] (codenamed Knights Corner) as an actual implementation of the MIC architecture, becoming an immediate success.

A Xeon Phi coprocessor of series 3100 houses 57 cores running at 1.1 GHz (1003 GFLOP/s of peak double precision performance), together with 6 GB of RAM, and can be attached to a host computer through PCIe. Series 7100 devices have better specifications: 61 cores at 1.238 GHz (1208 GFLOP/s) and 16 GB of RAM. The x86 cores used in these devices are less powerful than those in state-of-the-art Xeon processors, but they are still more general-purpose than those used in GPUs. They support 4-way Simultaneous Multithreading (SMT) and integrate Single Instruction, Multiple Data (SIMD) units. Interconnection among them and with the memory is via a ring bus.

A.3 Other accelerators

Field Programmable Gate Arrays (FPGAs) are programmable integrated circuits designed to be configured by the developer or user. Some of the most popular FPGA manufacturers are Altera and Xilinx. The main focus of FPGA manufacturers and developers is not HPC; however, they can be used to offload computationally intensive tasks after the hardware has been appropriately programmed, using tools based on languages such as VHDL, with which most programmers are not familiar [9]. The availability of new tools that automate or simplify the transformation of code from high-level, conventional programming languages (such as C) into VHDL code has made the HPC community pay renewed attention to FPGAs. Currently, both Altera and Xilinx are offering development environments based on the existing GPGPU frameworks, and (directly or through partners) PCIe-pluggable cards to be used as accelerators [10].

Digital Signal Processors (DSPs) can also be used as accelerators for general-purpose computing through specific or standardized APIs. For example, the SnuCL OpenCL framework [11] works with the C64x+ DSP by Texas Instruments.

A.4 Hybrid chips

In recent years, manufacturers have marketed hybrid chips combining different types of cores in the same die. A prototypical example is AMD's family of Accelerated Processing Units (APUs) [12], which combine several x86 cores with a GPU composed of several SIMD cores. Similarly, Intel has its own family of processors with integrated graphics [13]. The target market of these hybrid processors is low cost, low power computers, with the focus on mobility. A previous implementation of this class of hybrid chips was the Cell Broadband Engine [14].

GPUs integrated into hybrid chips are not very different from those used in discrete accelerators. They may provide reduced performance (fewer cores), due to cost and power limitations, thus being less attractive for HPC. However, they have an important advantage when compared to discrete accelerators: memory spaces for GPU and CPU are not separated; in fact, memory is shared. This drastically reduces the overheads associated with CPU–GPU communication and synchronization, because data movement through PCIe (required with discrete accelerators) is not necessary. Studies comparing the performance of the AMD Fusion platform against a discrete CPU+GPU system [15], [16] conclude that APUs are more compute efficient than a discrete GPU for applications involving massive data movements through the system bus.

ARM designs target low power systems on chip, such as those used in the mobile market. These chips are often hybrid, integrating blocks with different characteristics: combinations of low-power and high-performance CPUs (the big.LITTLE technology described in [17]), GPUs (we have already described the Mali family) and other specific-purpose units. Due to the increasing performance of ARM processors, while maintaining low power requirements, many proposals are appearing to use them in large-scale computing infrastructures, targeting greener cloud datacenters and supercomputers [18].

APPENDIX B
DEVELOPMENT TOOLS FOR ACCELERATORS

This appendix presents some APIs and tools to develop applications for heterogeneous computing systems. It is worth mentioning that a given API may be valid for a variety of devices (providing code portability), but this does not translate into performance portability: in most cases, programs must be tweaked for maximum performance on a target device, and the required reprogramming effort can be substantial.

B.1 CUDA

Compute Unified Device Architecture (CUDA) [19] is a framework released by NVIDIA for the development of general purpose applications on its GPUs. It includes a hardware (compute and memory) model, and a programming environment. CUDA has been widely adopted for the acceleration of massively data-parallel applications, due to its maturity and good performance. The interested reader can check [20] (on line) and [21] (printed) for additional information and learning materials targeting CUDA.

Regarding the hardware model, a CUDA GPU comprises several Streaming Multiprocessors (SMs) consisting of a collection of Streaming Processors (SPs), which are the processing cores. Additionally, each GPU has a dedicated, global memory space of several gigabytes. The host system (typically, the CPU) can copy data to/from this memory.

On the software side, each CUDA program is composed of a host side code and a collection of device *kernels*. The former runs on the host and manages data copies from/to host memory to/from device memory, and also the execution of the kernels on the GPU. The host part defines the size and geometry of the collection of *threads* that will run a kernel, and the organization of these threads into groups called *blocks*. At run time, each block will be assigned to a SM, where the kernel will be executed in a SIMD way by the SPs. Internally, each block is divided into sub-groups called *warps*, the minimum scheduling unit of the GPU. Up to version 5 of CUDA, warps have a fixed size of 32 threads.

B.2 OpenCL

Open Computing Language (OpenCL) [22] is a framework for the development of data/task parallel applications defined by the Khronos Group. It aims to be the vendor-neutral, standard API for programming accelerator devices. Like CUDA, it provides a hardware model and a programming environment. For additional information about OpenCL, the interested reader can check [23].

An OpenCL *platform* is a collection of compute *devices*. Each device comprises one or more *compute units* which, in turn, contain *processing elements*. A *host* device is in charge of managing the platform. A manufacturer releasing an OpenCL-capable device must deliver a companion OpenCL SDK, which makes the appropriate mapping of the actual hardware onto the OpenCL model.

An OpenCL application consists of some host-side and some device-side code. The host-side code manages the execution of the device-side code, and also the data movements between devices, including the host. OpenCL device applications are called *kernels*, and they are executed within a compute device by a number of developer-defined *work-items* (equivalent to CUDA threads). These work-items are aggregated into *work-groups* (equivalent to CUDA blocks), and its scheduling inside each compute unit is manufacturer-dependent. It is worth mentioning that although CUDA applications are compiled off-line, OpenCL kernels are commonly compiled at run time by a dynamic compiler included in the OpenCL SDKs.

B.3 Developing for the Xeon Phi coprocessor

Compared to GPUs, the Xeon Phi is a relative newcomer to the accelerator arena. However, it has inherited Intel's vast experience with multi-core processors, and the result is a rich and diverse collection of tools that makes the learning curve of the developer less steep.

Phi developers can use OpenMP (with Fortran, C and C++), Cilk Plus (an extension to C providing fine-grained task support), Intel Threading Building Blocks (a C++ template library to work with threads) and OpenCL. There is even an implementation of MPI that considers the coprocessor as a small-scale cluster. Two execution paradigms are provided: offload and native mode. In the former, applications are executed in the host processor and some compute-intensive portions are offloaded to the accelerator. In the latter, applications are executed completely in the device. See [24] for more details.

Compatibility with legacy codes and programming paradigms is one of the keys of the commercial success of the Xeon Phi. However, specific performance-oriented fine-tuning for the Phi is a must in order to fully exploit its capabilities [25].

B.4 Higher level tools

The frameworks described in the previous sections provide a low level of abstraction over the hardware. The programmer has a high degree of control of what the application does, so that he is finally responsible for fine-tuning the code to make efficient use of the accelerator. This is a difficult task, which becomes harder when we take into account the growing number of possible accelerators and families within them. In order to simplify programming, debugging and tuning accelerator-based applications, a diversity of higher-level tools and language constructions have appeared, which try to hide hardware complexities and to offer programmers a higher-level view of accelerators. Some of these approaches (whose descriptions fall outside the scope

of this paper) include the use of optimized libraries (for example, clMath [26]), programming with directives (such as OpenACC [27] or version 4 of OpenMP [28]), or relying on the parallelizing capabilities of the compiler (using, for example, the Par4All [29] open-source compiler workbench).

APPENDIX C
DEVICE CHARACTERIZATION

This appendix carries out a brief review of the literature about collecting those features that models can use to characterize a hardware device. The first source of information is the data sheets and detailed hardware manuals that the manufacturers provide (see for example device-specific details contained in [30], [31], [32]). Often, the development environment include mechanisms (an API) to query the device.

Relevant information about a device and the utilization that an application makes of it can be gathered using performance counters. Each device has its own collection of counters, and the manufacturer provides APIs and tools to obtain their values. For example, AMD offers GPUPerfAPI [33] for its line of GPUs, and NVIDIA does the same with CUPTI [34]. PAPI [35] is a common API to check performance counters, with bindings for many devices, including CUDA GPUs and the Intel Xeon Phi. Through this interface, it is possible to obtain not only resource-utilization, but also power-related data.

A complementary source of information is that obtained through micro-benchmarks, programs designed to stress a particular component of the hardware and return performance metrics about it. This technique has been used extensively for CPUs, and has been naturally extended to accelerators. Next we shortlist some of the works carried out in this field, grouped by device family.

- Wong et al. in [36] explore the behavior of branch sequences and memory hierarchies in NVIDIA GPUs. They are able to unveil some useful and undocumented features of the GT200 architecture.
- Taylor and Li identify and measure in [37] the architectural factors that cause performance bottlenecks in several AMD cards. Zhang et al. describe in [38] the use of micro-benchmarks to conduct a statistical analysis of performance and power consumption of AMD GPUs.
- In [39] and [40], Schmidl, Cramer et al. conduct several micro-benchmark based experiments on the Xeon Phi to measure latencies in the memory hierarchy, and also to assess the costs of OpenMP constructs.

APPENDIX D
APPLICATION CHARACTERIZATION

Techniques to characterize parallel applications have been widely studied in the context of multi-core systems [41], in clusters of supercomputers [42] and also in the field of accelerator based computing [43]. In this appendix we discuss some of the approaches to retrieve and characterize kernels, pieces of code offloaded to an accelerator.

One of the most intuitive ways to extract the features of a kernel is the analysis of its code in order to identify memory access patterns, arithmetic operations, branches, etc. This can be done directly on the source code or after some transformations. For example, in [44] authors use Clang [45] (the front-end of the LLVM [46] compiler infrastructure) to generate an Abstract Syntax Tree. Next, this tree is traversed to count the desired features.

It is also common to parse an Intermediate Representation (IR) of the program, an assembly-like code representation produced by the compiler before the generation of the binary files. Compared to source code, intermediate formats are simpler, easier to parse, and usually provide additional information, such as the number of registers per thread, size of buffers allocated in shared and constants memory, and so on. A drawback of this approach is that it relies on the potential optimizations that the compiler might have performed, and that may not exactly represent the original source code. In the case of CUDA applications for GPUs, the intermediate representation is in PTX, a virtual instruction set developed by NVIDIA. Regarding OpenCL, as compilation is performed at run-time, the IR of a kernel varies depending on the target accelerator and SDK. As a consequence, kernel metrics become device and compiler dependent. To avoid this undesirable side-effect, in 2012 the Khronos Group released the 1.0 version of the Standard Portable Intermediate Representation (SPIR) [47], a proposal of a common IR for all OpenCL frameworks, which is based on the one defined for the LLVM compiler toolchain.

A lower-level approach to code analysis consists of disassembling the final binary files. These are the files loaded onto and executed by the devices, so they provide maximum information about what the hardware will do. The drawback of this approach is that it is device or architecture specific, and that the obtained binary code might not be trivial to parse. Examples of this class of binary code analysis tools are decuda [48] and cuobjdump [49]; both are disassemblers for the CUDA binary format.

The Ocelot [50] modular compilation framework is a popular tool used to analyze CUDA kernels. Ocelot can transform a kernel from its PTX representation to LLVM's intermediate representation, and from this to other instruction sets (for example, x86 and ARM). Ocelot is also capable of emulating the execution of PTX codes and includes several source analysis tools. All together, it can be used to extract a variety of metrics from a PTX code (static information) and its execution (dynamic information).

APPENDIX E
BENCHMARK SUITES FOR ACCELERATORS

Benchmark suites of applications provide high-level measurements of the performance of a system when dealing with actual applications, and are considered an adequate tool to compare systems beyond theoretical FLOP/s. In fact, the Top500 list is elaborated using the High Performance LINPACK (HPL) benchmark [51]. In the context of performance modeling, benchmarks are widely used to train and adjust models, and to assess their accuracy. In this appendix we discuss some popular benchmark suites for HPC-oriented, accelerator-based computing. All of them contain implementations of commonly used algorithms, such as matrix operations, FFT or N-bodies problems.

Many of the applications included in accelerator-oriented benchmark suites have "accelerator-friendly" characteristics, that allow making good use of the capabilities of the devices: they have high compute-to-memory ratios, threads do not compete for access to shared data, or access patterns are regular. However, there are many real-world applications for which benefiting from the horsepower of accelerator devices is not trivial: applications that perform irregular memory access patterns, have complex control flows or make use of pointer-based structures such as graphs or lists. Porting these applications to an accelerator usually requires extensive algorithm rework, together with device-specific optimizations to achieve a good performance. A description of these classes of applications, represented in the form of patterns of "dwarfs" can be found in [52].

This is a list of popular benchmark suites targeting accelerators:

- The Rodinia application suite [53], [54], developed by the U. of Virginia, collects a large and diverse number of applications that, currently, are representative of dwarfs *structured grid*, *graph traversal*, *unstructured grid*, *dense linear algebra*, *dynamic programming* and *N-body*. However, this is a work in progress, with plans to introduce *spectral method*, *map reduce* and *sorting*. There is a multi-core (OpenMP) and a GPU-accelerated (CUDA or OpenCL) implementation for each application.
- The Parboil Benchmarks [55], maintained by the U. of Illinois, are a set of applications from different domains (image processing, biomolecular simulation, fluid dynamics and astronomy, among others). There are regular but also irregular applications (sparse matrix-dense vector multiplication is an example of the latter). For each application, typical performance bottlenecks derived from non-optimized implementations are identified. The suite provides a variety of multi-core (OpenMP) and accelerated (CUDA/OpenCL) implementations, including a simple, readable (and non-optimized) version, together with optimized versions.
- The Scalable HeterOgeneous Computing (SHOC) benchmark suite [56] from ORNL is a collection of benchmarks that aims to be diverse in the nature of its applications. Authors provide CUDA and OpenCL implementations of each benchmark, with an additional feature: the applications can be scaled to clusters using MPI. Initially tested on AMD and NVIDIA GPUs, the suite also runs on the Xeon Phi.
- LonestarGPU [57] is a collection of widely-used real-world applications that exhibit irregular behavior, maintained at the U. of Texas at Austin and the Texas State University. It includes seven applications, all written in CUDA.
- The Mantevo Project [58] maintains a collection of "miniapps" and "minidrivers" targeting a variety of application domains (explicit and implicit unstructured partial differential equations, explicit and implicit structured partial differential equations, molecular dynamics, hydrodynamics and circuit simulations) and hardware (including GPUs). Miniapps are small, self-contained programs that embody essential performance characteristics of key applications, while minidrivers are used to assess the performance of libraries.
- The NAS Parallel Benchmarks suite (NPB) [59] is a classic suite used to test the performance of multi-core computers and supercomputers. In 2011, Seoul National University adapted several of the NPB benchmarks to OpenCL and made them publicly available [60].
- Finally, test and demonstration programs included in the SDKs provided by vendors of accelerators have also been widely used for testing purposes.

APPENDIX F
SUMMARY OF PERFORMANCE MODELS

This appendix contains a collection of tables that summarize the main features of the models analyzed in Section 3 of the main body of the paper:

- Table 1: summary of the models studied in Section 3.1 Execution time estimation
- Table 2: summary of the models studied in Section 3.2 Bottleneck highlighting and code optimizationn
- Table 3: summary of the models studied in Section 3.3 Power consumption estimation
- Table 4: summary of the simulators studied in Section 3.4

REFERENCES

[1] "ARM home page," http://www.arm.com, Nov. 2013.

[2] "ARM Mali graphics plus GPU compute," http://www.arm.com/products/multimedia/mali-graphics-plus-gpu-compute, Jan. 2013.

[3] E. Lindholm, J. Nickolls, S. Oberman, and J. Montrym, "NVIDIA Tesla: A unified graphics and computing architecture," *Micro, IEEE*, vol. 28, no. 2, pp. 39–55, 2008.

[4] A. Bayoumi, M. Chu, Y. Hanafy, P. Harrell, and G. Refai-Ahmed, "Scientific and engineering computing using ATI Stream technology," *Computing in Science and Engineering*, vol. 11, no. 6, pp. 92–97, 2009.

Model	Method	Target platform	Input preprocessing	Limitations	Highlights
Hong & Kim [61]	AN	• NVIDIA GPUs (8800 GT, FX 5600, GTX 280)	• Execution of μBenchmarks • Analysis of PTX	• Cache misses and branch instructions not modeled • Memory model too simple	• Extendable to non NVIDIA GPUs
Kothapalli et al. [62]	AN	• NVIDIA GPUs (GTX 280)	• Analysis of kernel pseudo-code	• Two models are provided, with no hints as to which one choose • Overheads of intra-thread synchronization and atomic operations not modeled	• Extendable to non NVIDIA GPUs
Kerr et al. [63]	ML	• GPUs (4 NVIDIA models) • Multicore CPUs (3 different)	• Analysis of PTX using Ocelot	• Insufficient information about model validation procedures	• Model validated using a large benchmark
Che & Skadron [64]	ML	• NVIDIA GPUs (Tesla C2050)	• Execution of benchmarks	• Kernels that use texture memory are not modeled accurately	• Can provide performance predictions for hypothetical architectures when used with a simulator
Sato et al. [65]	ML	• OpenCL devices (NVIDIA Tesla C1060)	• Execution of kernels on target devices	• Lack of accuracy for kernels with loops with an unknown number of iterations	
Nugteren & Corporaal [66]	AN	• GPUs (NVIDIA GTX 470 & GTS 250) • Multicore CPUs (Intel Q8300 & i7-930)	• Execution of μBenchmarks • Code analysis and classification following the taxonomy in [67]	• Accuracy limited by the high level of modeling abstraction	• Extendable to new architectures • Graphical output • Provides proposals for optimization
Meng et al. [68]	AN	• NVIDIA GPUs (GTX 280)	• Execution of μBenchmarks • Analysis of ISL parameters	• Applicable only to ISL applications	• Support provided for the automatic generation of accelerated ISL codes
Choi et al. [69]	AN	• NVIDIA GPUs (Tesla C870, Tesla C1060)	• Analysis of input matrices	• Applicable only to SpMV multiply operation • Thread block scheduling latencies are not taken into account	• Proposes the GPU-friendly BELLPACK format for sparse matrices

TABLE 1
Summary of models for the estimation of execution time. In the "Method" column, AN means *analytical* and ML means *machine learning*. In the "Target platform" column, names inside parentheses refer to devices on which the model has been tested

[5] Y. Zhang, L. Peng, B. Li, J.-K. Peir, and J. Chen, "Architecture comparisons between NVIDIA and ATI GPUs: Computation parallelism and data communications," in *Workload Characterization (IISWC), 2011 IEEE Int. Symp. on*, pp. 205–215.

[6] "Top500 supercomputing sites," http://www.top500.org, Dec. 2013.

[7] L. Seiler, D. Carmean, E. Sprangle, T. Forsyth, M. Abrash, P. Dubey, S. Junkins, A. Lake, J. Sugerman, R. Cavin, R. Espasa, E. Grochowski, T. Juan, and P. Hanrahan, "Larrabee: a many-core x86 architecture for visual computing," *ACM Trans. Graph.*, vol. 27, no. 3, pp. 18:1–18:15, 2008.

[8] Intel, "Xeon Phi Coprocessor: Parallel Processing, Unparalleled Discovery," http://www.intel.com/content/www/us/en/high-performance-computing/high-performance-xeon-phi-coprocessor-brief.html, Nov. 2013.

[9] S. Che, J. Li, J. W. Sheaffer, K. Skadron, and J. Lach, "Accelerating compute-intensive applications with GPUs and FPGAs," in *IEEE Symp. on Application Specific Processors (SASP), 2008*, pp. 101–107.

[10] T. S. Czajkowski, U. Aydonat, D. Denisenko, J. Freeman, M. Kinsner, D. Neto, J. Wong, P. Yiannacouras, and D. P. Singh, "From OpenCL to high-performance hardware on FPGAs," in *IEEE 22nd Int. Conf. onField Programmable Logic and Applications (FPL)*, 2012, pp. 531–534.

[11] J. Lee, J. Kim, S. Seo, S. Kim, J. Park, H. Kim, T. T. Dao, Y. Cho, S. J. Seo, S. H. Lee, S. M. Cho, H. J. Song, S.-B. Suh, and J.-D. Choi, "An opencl framework for heterogeneous multicores with local memory," in *Proceedings of the 19th International Conference on Parallel Architectures and Compilation Techniques, PACT 2010*, pp. 193–204.

[12] A. Branover, D. Foley, and M. Steinman, "AMD Fusion APU: Llano," *Micro, IEEE*, vol. 32, no. 2, pp. 28–37, 2012.

[13] E. Rotem, A. Naveh, D. Rajwan, A. Ananthakrishnan, and E. Weissmann, "Power-management architecture of the intel microarchitecture code-named sandy bridge," *Micro, IEEE*, vol. 32, no. 2, pp. 20–27, 2012.

[14] J. Kahle, M. Day, H. Hofstee, C. Johns, T. Maeurer, and D. Shippy, "Introduction to the cell multiprocessor," *IBM journal of Research and Development*, vol. 49, no. 4.5, pp. 589–604, 2005.

[15] M. Daga, A. Aji, and W. Feng, "On the efficacy of a fused CPU + GPU processor (or APU) for parallel computing," in *IEEE Symp. on Application Accelerators in High-Performance Computing (SAAHPC), 2011*, pp. 141–149.

[16] K. L. Spafford, J. S. Meredith, S. Lee, D. Li, P. C. Roth, and J. S. Vetter, "The tradeoffs of fused memory hierarchies in heteroge-

Model	Method	Target platforms	Input preprocessing	Limitations	Highlights
Williams et al. [70]	AN	• Multi-core CPUs [70] • NVIDIA and AMD GPUs [71] • Intel Xeon Phi [40]	• Execution of μBenchmarks • Analysis of kernel		• Visual output to guide optimizations
Lai & Seznec [72]	AN	• NVIDIA GPUs (GT200)	• Execution of μBenchmarks • Analysis of kernel with simulator & cuobjdump	• Cache effects, instruction pipeline stages & memory transactions are not modeled in detail	
Zhang & Owens [73]	AN	• NVIDIA GPUs (GTX280)	• Execution of μBenchmarks • Analysis of kernel with the Barra simulator	• Assumes enough warps to hide latencies • Overheads of branch and synchronization instructions and effects of caches are not modeled	• Extendable to non NVIDIA GPUs
Baghsorkhi et al. [74]	AN	• NVIDIA GPUs (8800 GT)	• Execution of μBenchmarks • Analysis of kernel source code	• Compiler optimizations are not fully taken into account	• Provides a detailed breakdown of execution time into compute and memory transaction stages
Ramos & Hoefler [75]	AN	• Intel Xeon Phi (5110P)	• Execution of μBenchmarks	• Focused on inter-task communications	
Baghsorkhi et al. [76]	AN	• NVIDIA GPUs (Tesla C2050)	• Execution of μBenchmarks • Instrumentation of kernel	• Focused on the memory hierarchy	• Provides a detailed breakdown of memory operations
Cui et al. [77]	AN	• NVIDIA GPUs (GTS 250)	• Analysis of kernel using a simulator	• It is assumed that global memory is not a bottleneck • Multi-cycle operations or memory access latencies are not modeled	
Sim et al. [78]	AN	• NVIDIA GPUs (Tesla C2050)	• Execution of μBenchmarks • Analysis of kernel with profiler, Ocelot & cuobjdump	• Not easily portable to non NVIDIA GPUs	• Provides estimations of potential benefits of different optimizations

TABLE 2
Summary of models for bottleneck highlighting and code optimization

neous computing architectures," in *Proc. of the 9th conference on Computing Frontiers*, ser. CF '12, 2012, pp. 103–112.

[17] P. Greenhalgh, "Big. LITTLE processing with ARM Cortex.-A15 & Cortex-A7," *ARM Whitepaper*, 2011.

[18] N. Rajovic, N. Puzovic, L. Vilanova, C. Villavieja, and A. Ramirez, "The low-power architecture approach towards exascale computing," in *Proc. of the second workshop on Scalable algorithms for large-scale systems*. ACM, 2011, pp. 1–2.

[19] NVIDIA, "CUDA Home Page," http://developer.nvidia.com/cuda, Nov. 2013.

[20] ——, "Cuda education and training," http://developer.nvidia.com/cuda-education-training, Apr. 2013.

[21] D. B. Kirk and W.-m. W. Hwu, *Programming Massively Parallel Processors: A Hands-on Approach*, 1st ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2010.

[22] Khronos Group, *The OpenCL Specification*, 2008.

[23] B. Gaster, L. Howes, D. R. Kaeli, P. Mistry, and D. Schaa, *Heterogeneous Computing with OpenCL*. Burlington, MA: Elsevier, 2011.

[24] J. Jeffers and J. Reinders, *Intel Xeon Phi coprocessor high performance programming*. Morgan Kaufmann, 2013.

[25] J. Reinders, "An Overview of Programming for Intel Xeon processors and Intel Xeon Phi coprocessors," Intel Corporation, Tech. Rep., December 2012.

[26] AccelerEyes, "clMath: An Open Source BLAS and FFT Library for OpenCL," http://www.accelereyes.com/clmath, Nov. 2013.

[27] "OpenACC - Directives for accelerators," http://www.openacc-standard.org/, Nov. 2013.

[28] OpenMP Architecture Review Board, "OpenMP application program interface version 4.0," Jul. 2013.

[29] M. Amini, B. Creusillet, S. Even, R. Keryell, O. Goubier, S. Guelton, J. McMahon, F. Pasquier, G. Péan, and P. Villalon, "Par4all: From convex array regions to heterogeneous computing," in *2nd Int. Workshop on Polyhedral Compilation Techniques (Impact), 2012*.

[30] AMD, "Accelerated Parallel Programming OpenCL Best Practices Guide," http://developer.amd.com/tools/hc/AMDAPPSDK/assets/AMD_Accelerated_Parallel_Processing_OpenCL_Programming_Guide.pdf, Jan. 2012.

[31] NVIDIA, "OpenCL Best Practices Guide," www.nvidia.com/content/cudazone/CUDABrowser/downloads/papers/NVIDIA_OpenCL_BestPracticesGuide.pdf, Nov. 2013.

[32] R. Rahman, *Intel Xeon Phi Coprocessor Architecture and Tools: The Guide for Application Developers*. Apress, 2013. [Online]. Available: http://books.google.es/books?id=pQV2nAEACAAJ

[33] AMD, "GPUPerfAPI Home Page," http://developer.amd.com/tools-and-sdks/graphics-development/gpuperfapi, Nov. 2013.

[34] NVIDIA, "CUPTI User's guide," http://docs.nvidia.com/cuda/cupti, Nov. 2013.

[35] D. Terpstra, H. Jagode, H. You, and J. Dongarra, "Collecting Performance Data with PAPI-C," in *Tools for High Performance Computing 2009*, p. 157.

[36] H. Wong, M. Papadopoulou, M. Sadooghi-Alvandi, and

Model	Method	Target platforms	Input preprocessing	Limitations	Highlights
Wang & Ranganathan [79]	AN	• NVIDIA GPUs (GTX 460)	• Execution of μBenchmarks • External measurements • Analysis of PTX	• Overheads of branch instructions are not modeled	• Simple yet accurate
Hong & Kim [80]	AN	• NVIDIA GPUs (GTX 280)	• Execution of μBenchmarks • External measurements • Analysis of PTX using Ocelot	• Kernels with complex control flows are not modeled correctly	• Models thermal effects
Shao & Brooks [81]	AN	• Intel Xeon Phi (5110P)	• Execution of μBenchmarks • External measurements • Analysis of kernel using a profiler		• Integrated with Intel profiler for Xeon Phi
Ma et al. [82] [83]	ML	• NVIDIA GPUs (8800 GT)	• Execution of μBenchmarks • External measurements • Analysis of kernel using a profiler	• Power consumption peaks are not modeled accurately	
Nagasaka et al. [84]	ML	• NVIDIA GPUs (GTX 285)	• Execution of benchmarks • External measurements • Analysis of kernel using a profiler	• Kernels that use texture memory are not modeled accurately	
Song et al. [85]	ML	• NVIDIA GPUs (Tesla C2050, Tesla M2090)	• Performance counters (through NVIDIA API) • Power & Temperature measurements (through NVIDIA API)	• Kernels with few or asymmetric threads are not modeled accurately	• Extendable to other GPU architectures with minimal adaptations
Leng et al. [86]	AN	• NVIDIA GPUs (GTX 480, FX 5600)	• Execution of μBenchmarks • External measurements • Execution of kernels in GPGPU-Sim	• Short-running kernels are not modeled accurately	• Extendable to new architectures • Able to simulate feed-back driven runtime optimizations using GPGPU-Sim
Lucas et al. [87]	AN	• NVIDIA GPUs (GT 240, GTX 580)	• Execution of μBenchmarks • External measurements • Analysis of kernel using simulator		• Provides breakdown of power consumption over the GPU components

TABLE 3
Summary of models for power consumption estimation

A. Moshovos, "Demystifying GPU microarchitecture through microbenchmarking," in *IEEE Int. Symp. on Performance Analysis of Systems & Software (ISPASS), 2010*, pp. 235–246.

[37] R. Taylor and X. Li, "A micro-benchmark suite for AMD GPUs," in *IEEE 39th Int. Conf. on Parallel Processing Workshops (ICPPW), 2010*, pp. 387–396.

[38] Y. Zhang, Y. Hu, B. Li, and L. Peng, "Performance and Power Analysis of ATI GPU: A statistical approach," in *IEEE 6th Int. Conf. on Networking, Architecture and Storage (NAS), 2011*, pp. 149–158.

[39] D. Schmidl, T. Cramer, S. Wienke, C. Terboven, and M. Muller, "Assessing the Performance of OpenMP Programs on the Intel Xeon Phi," in *Euro-Par 2013 Parallel Processing*, ser. LNCS, vol. 8097, pp. 547–558.

[40] T. Cramer, D. Schmidl, M. Klemm, and D. an Mey, "OpenMP Programming on Intel Xeon Phi Coprocessors: An Early Performance Comparison," in *Proc. of the Many-core Applications Research Community Symp. at RWTH Aachen University*, 2012, pp. 38–44.

[41] L. Carrington, M. Laurenzano, A. Snavely, R. Campbell, and L. Davis, "How well can simple metrics represent the performance of HPC applications?" in *Proc. of the ACM/IEEE Conf. on Supercomputing (SC), 2005*, p. 48.

[42] J. Gonzalez, M. Casas, J. Gimenez, M. Moreto, A. Ramirez, J. Labarta, and M. Valero, "Simulating whole supercomputer applications," *Micro, IEEE*, vol. 31, no. 3, pp. 32–45, 2011.

[43] N. Goswami, R. Shankar, M. Joshi, and T. Li, "Exploring GPGPU workloads: Characterization methodology, analysis and microarchitecture evaluation implications," in *IEEE Int. Symp. on Workload Characterization (IISWC), 2010*, pp. 1–10.

[44] D. Grewe and M. O'Boyle, "A static task partitioning approach for heterogeneous systems using opencl," in *Compiler Construction*, ser. LNCS, vol. 6601, 2011, pp. 286–305.

[45] C. Lattner, "LLVM and Clang: Next generation compiler technology," in *The BSD Conf., Ottawa, Canada*, 2008.

[46] C. Lattner and V. Adve, "LLVM: A compilation framework for lifelong program analysis & transformation," in *IEEE Int. Symp. on Code Generation and Optimization (CGO), 2004*, pp. 75–86.

[47] Khronos Group, "OpenCL Standard Portable Intermediate Representation (SPIR) 1.0," http://www.khronos.org/registry/cl/specs/spir_spec-1.0-provisional.pdf, Sep. 2012.

[48] W. van der Laan, "Decuda and cudasm, the cubin utilities package," 2009.

[49] NVIDIA, "cuobjdump application note," Sep. 2012.

[50] G. F. Diamos, A. R. Kerr, S. Yalamanchili, and N. Clark, "Ocelot: a dynamic optimization framework for bulk-synchronous applications in heterogeneous systems," in *Proc. of the 19th international conference on Parallel architectures and compilation techniques*, 2010, pp. 353–364.

[51] J. J. Dongarra, P. Luszczek, and A. Petitet, "The linpack benchmark: Past, present, and future. concurrency and computation: Practice and experience," *Concurrency and Computation: Practice and Experience*, vol. 15, pp. 803–820, 2003.

Model	Target platforms	Target applications	Limitations	Highlights
GPGPU-Sim [88]	• NVIDIA GPUs (8600 GTS)	• CUDA • GPU OpenCL	• Requires actual GPU to compile OpenCL codes	• Power modeling features • Visual analysis tool
Barra [89]	• NVIDIA GPUs (9800 GX2)	• CUDA		• Scalable, multi-core implementation of the simulator
MacSim [90]	• NVIDIA GPUs (8800 GT, GTX 280, GTX 465) • Multi-core CPUs • Hybrid chips	• CPU sequential & parallel • CUDA	• GPU cores in hybrid systems modeled as CUDA SMs	• Power modeling features for CPU simulation
FusionSim [91]	• NVIDIA GPUs (FX 5800) • Multi-core CPUs • Hybrid chips	• CUDA	• GPU cores in hybrid systems modeled as CUDA SMs • Simulates only single-core CPUs	
Multi2Sim [92]	• NVIDIA and AMD GPUs (Radeon HD 5870) • Multi-core CPUs • Hybrid chips	• CPU sequential & parallel • GPU CUDA & OpenCL		• Visual analysis tool • Module for massive simulation

TABLE 4
Summary of simulators

[52] K. Asanovic, R. Bodik, B. C. Catanzaro, J. J. Gebis, P. Husbands, K. Keutzer, D. A. Patterson, W. L. Plishker, J. Shalf, S. W. Williams, and K. A. Yelick, "The landscape of parallel computing research: A view from berkeley," EECS Department, University of California, Berkeley, Tech. Rep., Dec 2006.

[53] S. Che, M. Boyer, J. Meng, D. Tarjan, J. Sheaffer, S. Lee, and K. Skadron, "Rodinia: A benchmark suite for heterogeneous computing," in *IEEE Int. Symp. on Workload Characterization (IISWC), 2009*, pp. 44–54.

[54] S. Che, J. Sheaffer, M. Boyer, L. Szafaryn, L. Wang, and K. Skadron, "A characterization of the rodinia benchmark suite with comparison to contemporary cmp workloads," in *Proc. of the IEEE Int. Symp. on Workload Characterization (IISWC)*, 2010.

[55] J. A. Stratton, C. Rodrigues, I.-J. Sung, N. Obeid, L.-W. Chang, N. Anssari, G. D. Liu, and W.-m. W. Hwu, "Parboil: A revised benchmark suite for scientific and commercial throughput computing," *Center for Reliable and High-Performance Computing*, 2012.

[56] A. Danalis, G. Marin, C. McCurdy, J. Meredith, P. Roth, K. Spafford, V. Tipparaju, and J. Vetter, "The scalable heterogeneous computing (shoc) benchmark suite," in *Proc. of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units*. ACM, 2010, pp. 63–74.

[57] M. Burtscher, R. Nasre, and K. Pingali, "A quantitative study of irregular programs on GPUs," in *Workload Characterization (IISWC), 2012 IEEE Int. Symp. on*, 2012, pp. 141–151.

[58] M. A. Heroux, D. W. Doerfler, P. S. Crozier, J. M. Willenbring, H. C. Edwards, A. Williams, M. Rajan, E. R. Keiter, H. K. Thornquist, and R. W. Numrich, "Improving performance via mini-applications," *Sandia National Laboratories, Tech. Rep*, 2009.

[59] D. Bailey, E. Barszcz, J. Barton, D. Browning, R. Carter, L. Dagum, R. Fatoohi, P. Frederickson, T. Lasinski, R. Schreiber *et al.*, "The NAS parallel benchmarks summary and preliminary results," in *Proc. of the ACM/IEEE Conf. on Supercomputing, 1991*, pp. 158–165.

[60] S. Seo, G. Jo, and J. Lee, "Performance characterization of the nas parallel benchmarks in opencl," in *Workload Characterization (IISWC), 2011 IEEE Int. Symp. on*. IEEE, 2011, pp. 137–148.

[61] S. Hong and H. Kim, "An analytical model for a GPU architecture with memory-level and thread-level parallelism awareness," *ACM SIGARCH Computer Architecture News*, vol. 37, no. 3, pp. 152–163, 2009.

[62] K. Kothapalli, R. Mukherjee, M. Rehman, S. Patidar, P. Narayanan, and K. Srinathan, "A performance prediction model for the CUDA GPGPU platform," in *High Performance Computing (HiPC), 2009 IEEE Int. Conf. on*, pp. 463–472.

[63] A. Kerr, G. Diamos, and S. Yalamanchili, "Modeling GPU-CPU workloads and systems," in *Proc. of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units*, 2010, pp. 31–42.

[64] S. Che and K. Skadron, "BenchFriend: Correlating the Performance of GPU Benchmarks," *To appear in Int. Journal of High Performance Computing Applications*, 2013.

[65] K. Sato, K. Komatsu, H. Takizawa, and H. Kobayashi, "A history-based performance prediction model with profile data classification for automatic task allocation in heterogeneous computing systems," in *Parallel and Distributed Processing with Applications (ISPA), 2011 IEEE 9th Int. Symp. on*, pp. 135–142.

[66] C. Nugteren and H. Corporaal, "The Boat Hull model: enabling performance prediction for parallel computing prior to code development," in *Proc. of the 9th conference on Computing Frontiers*. ACM, 2012, pp. 203–212.

[67] ——, "A modular and parameterisable classification of algorithms," No. ESR-2011-02, Eindhoven University of Technology, Tech. Rep., 2011.

[68] J. Meng and K. Skadron, "Performance modeling and automatic ghost zone optimization for iterative stencil loops on GPUs," in *Proc. of the 23rd ACM Int. Conf. on Supercomputing*, 2009, pp. 256–265.

[69] J. Choi, A. Singh, and R. Vuduc, "Model-driven autotuning of sparse matrix-vector multiply on GPUs," *ACM Sigplan Notices*, vol. 45, no. 5, pp. 115–126, 2010.

[70] S. Williams, A. Waterman, and D. Patterson, "Roofline: an insightful visual performance model for multicore architectures," *Communications of the ACM*, vol. 52, no. 4, pp. 65–76, 2009.

[71] H. Jia, Y. Zhang, G. Long, J. Xu, S. Yan, and Y. Li, "GPURoofline: A Model for Guiding Performance Optimizations on GPUs," in *Euro-Par 2012 Parallel Processing*, ser. LNCS, 2012, vol. 7484, pp. 920–932.

[72] J. Lai and A. Seznec, "Break down GPU execution time with an analytical method," in *Proc. of 2012 Workshop on Rapid Simulation and Performance Evaluation: Methods & Tools*, pp. 33–39.

[73] Y. Zhang and J. Owens, "A quantitative performance analysis model for GPU architectures," in *High Performance Computer Architecture (HPCA), 2011 IEEE 17th Int. Symp. on*, pp. 382–393.

[74] S. Baghsorkhi, M. Delahaye, S. Patel, W. Gropp, and W. Hwu, "An adaptive performance modeling tool for GPU architectures," in *ACM SIGPLAN Notices*, vol. 45, no. 5, 2010, pp. 105–114.

[75] S. Ramos and T. Hoefler, "Modeling communication in cache-coherent SMP systems: a case-study with Xeon Phi," in *Proc. of the 22nd Int. Symp. on High-performance parallel and distributed computing*, ser. HPDC, 2013, pp. 97–108.

[76] S. Baghsorkhi, I. Gelado, M. Delahaye, and W. Hwu, "Efficient performance evaluation of memory hierarchy for highly multi-

threaded graphics processors," in *Proc. of the 17th ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming*, 2012, pp. 23–34.

[77] Z. Cui, Y. Liang, K. Rupnow, and D. Chen, "An Accurate GPU Performance Model for Effective Control Flow Divergence Optimization," in *Parallel Distributed Processing Symp. (IPDPS), 2012 IEEE 26th Int.*, 2012, pp. 83–94.

[78] J. Sim, A. Dasgupta, H. Kim, and R. Vuduc, "A performance analysis framework for identifying potential benefits in GPGPU applications," in *Proc. of the 17th ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming*, 2012, pp. 11–22.

[79] Y. Wang and N. Ranganathan, "An Instruction-Level Energy Estimation and Optimization Methodology for GPU," in *Computer and Information Technology (CIT), 2011 IEEE 11th Int. Conf. on*, 2011, pp. 621–628.

[80] S. Hong and H. Kim, "An integrated GPU power and performance model," in *ACM SIGARCH Computer Architecture News*, vol. 38, no. 3, 2010, pp. 280–289.

[81] Y. S. Shao and D. Brooks, "Energy Characterization and Instruction-Level Energy Model of Intels Xeon Phi Processor," in *Proc. of the 2013 ACM/IEEE Int. Symp. on Low power electronics and design (ISLPED)*.

[82] X. Ma, M. Dong, L. Zhong, and Z. Deng, "Statistical power consumption analysis and modeling for GPU-based computing," in *Proceeding of ACM SOSP Workshop on Power Aware Computing and Systems (HotPower)*, 2009.

[83] X. Ma, M. Rincon, and Z. Deng, "Improving Energy Efficiency of GPU based General-Purpose Scientific Computing through Automated Selection of Near Optimal Configurations," UH-CS-11-08, Tech. Rep., 2011.

[84] H. Nagasaka, N. Maruyama, A. Nukada, T. Endo, and S. Matsuoka, "Statistical power modeling of GPU kernels using performance counters," in *Green Computing Conf., 2010*, pp. 115–122.

[85] S. Song, C. Su, B. Rountree, and K. W. Cameron, "A Simplified and Accurate Model of Power-Performance Efficiency on Emergent GPU Architectures," in *Proc. of the 2013 IEEE 27th Int. Symp. on Parallel and Distributed Processing (IPDPS)*, pp. 673–686.

[86] J. Leng, T. Hetherington, A. ElTantawy, S. Gilani, N. S. Kim, T. M. Aamodt, and V. J. Reddi, "GPUWattch: Enabling Energy Optimizations in GPGPUs," in *Proc. of the 40th Annual Int. Symp. on Computer architecture*, 2013.

[87] J. Lucas, S. Lal, M. Andersch, M. Alvarez-Mesa, and B. Juurlink, "How a single chip causes massive power bills - GPUSimPow: A GPGPU power simulator," in *Performance Analysis of Systems and Software (ISPASS), 2013 IEEE Int. Symp. on*, 2013, pp. 97–106.

[88] A. Bakhoda, G. Yuan, W. Fung, H. Wong, and T. Aamodt, "Analyzing CUDA workloads using a detailed GPU simulator," in *IEEE Int. Symp. on Performance Analysis of Systems and Software. ISPASS 2009.*, pp. 163–174.

[89] S. Collange, M. Daumas, D. Defour, and D. Parello, "Barra: a parallel functional simulator for GPGPU," in *IEEE Int. Symp. on Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS), 2010*, pp. 351–360.

[90] Gatech, "Macsim Simulator for heterogeneous architecture," http://code.google.com/p/macsim/, Jan. 2012.

[91] V. Zakharenko, "FusionSim: Characterizing the Performance Benefits of Fused CPU/GPU Systems," Ph.D. dissertation, University of Toronto, 2012.

[92] R. Ubal, B. Jang, P. Mistry, D. Schaa, and D. Kaeli, "Multi2Sim: a simulation framework for CPU-GPU computing," in *Proc. of the 21st international conference on Parallel architectures and compilation techniques*, 2012, pp. 335–344.

6

An Efficient Implementation of Kernel Density Estimation for Multi-core and Many-core Architectures

Original Article

International Journal of HIGH PERFORMANCE COMPUTING APPLICATIONS

The International Journal of High Performance Computing Applications
1–17

DOI: 10.1177/1094342015576813
hpc.sagepub.com
SAGE

An efficient implementation of kernel density estimation for multi-core and many-core architectures

Unai Lopez-Novoa[1], Jon Sáenz[2,3], Alexander Mendiburu[1] and Jose Miguel-Alonso[1]

Abstract

Kernel density estimation (KDE) is a statistical technique used to estimate the probability density function of a sample set with unknown density function. It is considered a fundamental data-smoothing problem for use with large datasets, and is widely applied in areas such as climatology and biometry. Due to the large volumes of data that these problems usually process, KDE is a computationally challenging problem. Current HPC platforms with built-in accelerators have an enormous computing power, but they have to be programmed efficiently in order to take advantage of that power. We have developed a novel strategy to compute KDE using bounded kernels, trying to minimize memory accesses, and implemented it as a parallel program targeting multi-core and many-core processors. The efficiency of our code has been tested with different datasets, obtaining impressive levels of acceleration when taking as reference alternative, state-of-the-art KDE implementations.

Keywords

Kernel density estimation, bounded kernel functions, parallel computing, many-core processors

1 Introduction

Kernel density estimation (KDE) is a statistical technique used to estimate the probability density function of a sample set with unknown density function. It was first introduced in the 1960s for univariate data and, due to its widespread adoption, multi-variate estimators appeared in subsequent years. It is considered a fundamental data-smoothing problem, and is widely used due to its properties, in contrast with other common density estimation techniques, such as histograms (Silverman, 1986; Ahamada and Flachaire, 2010).

KDE is a common tool in many research areas, used for a variety of purposes. For example, in Andrés Ferreyra et al. (2001) the authors use density estimates to forecast weather and other factors as part of a model for optimizing maize production. In the same field, it has been applied to evaluate the signature of climate change in the frequency of weather regimes (Corti et al., 1999). In Weissbach (2006), the effectiveness of a particular medical treatment is determined by means of KDE. In computer vision (Elgammal et al., 2003), it is applied for image segmentation and tracking. In the field of evolutionary computation, density estimation has been used to estimate a distribution of the problem variables in estimation of distribution algorithms (Bosman and Thierens, 2000; Luo and Qian, 2009). An extensive list of application fields of KDE can be found in Sheather (2004).

A common characteristic of the mentioned research areas is the need to deal with large volumes of data, and also the need to perform several KDE runs (varying parameters) in order to select the best estimators. For example, for some of the tests with real datasets carried out in Section 5.2, execution times with state-of-the-art implementations of KDE were longer than 30 hours. Thus, some form of fast KDE computation is required to solve in a short time these challenging problems. The

[1]Department of Computer Architecture and Technology, University of the Basque Country UPV/EHU, Donostia-San Sebastian, Spain
[2]Department of Applied Physics II, University of the Basque Country UPV/EHU, Leioa, Spain
[3]Plentzia Itsas Estazioa, University of the Basque Country UPV/EHU, Plentzia, Spain

Corresponding author:
Unai Lopez-Novoa, Department of Computer Architecture and Technology, University of the Basque Country UPV/EHU, Paseo Manuel de Lardizabal, 1 Donostia-San Sebastian, Gipuzkoa 20018, Spain.
Email: unai.lopez@ehu.es

way to go is the common one in high-performance computing: parallel processing.

Current high-performance systems are becoming hybrid: a combination of several multi-core CPUs and one or more accelerator devices, which can be graphics processing units (GPUs), or many-core coprocessors such as Intel's Xeon Phi (Jeffers and Reinders, 2013). GPUs have a huge raw performance (several TFLOPs), but require extensive fine-tuning in order to get close to their peak power. They are usually programmed using low-level APIs such as CUDA or OpenCL, but recently annotation-based frameworks such as OpenACC (OpenACC.org 2013) have eased the development of accelerator-based parallel applications. The Xeon Phi, although less powerful (in terms of peak TFLOPs: around one), is easier to program: a diversity of paradigms and tools is available for it, including those developed for state-of-the-art multi-core CPUs. In this paper, we test our KDE code in an Intel Core i7 multi-core processor, and in a many-core Intel Xeon Phi coprocessor.

This paper presents two main contributions. The first one is an analysis of the KDE problem, resulting in a novel algorithmic approach that offers important savings in terms of execution times, even when implemented as a sequential program. However, given the intrinsic data parallelism of KDE, a second contribution comes naturally: a multi-threaded parallel implementation of our proposal using OpenMP, suitable for both multi-core and many-core processors. The use of OpenMP has enabled us to develop a program that is not only portable, but also performance efficient in both platforms. We have compared the execution times of our code with those of two state-of-the-art implementations of KDE, for different datasets, in order to show the excellent performance achieved. Furthermore, we have analyzed our program to study its scalability, and to identify possible ways of further improving its efficiency.

The remainder of this paper is organized as follows. We provide a description of the KDE problem in Section 2, and present a novel approach to compute it in Section 3. We describe the development environment and our KDE implementation in Section 4. The experiments carried out to assess its performance are detailed in Section 5. Finally, Section 6 draws some final conclusions and describes our future lines of work on this topic.

2 Background on KDE

The *probability distribution* of a random variable X is described through its probability density function (PDF) f. This function f gives a natural description of the distribution of X, and allows us to determine the probabilities associated with X using the relationship

$$P(a<X<b) = \int_a^b f(x)\mathrm{dx} \tag{1}$$

Given several observed data points (samples) from a random variable X, with unknown density function f, *density estimation* is used to create an estimated density function $\hat{f}$ from the observed data. There are two approaches to density estimation, parametric and non-parametric. The former assumes that the data is drawn from a known parametric family of distributions, for example a normal distribution. The latter does not make this assumption. The main restriction of parametric methods is that they impose restrictions on the shapes that f can have. KDE is a non-parametric approach.

One of the most common techniques for density estimation of a continuous variable is the *histogram*, which is a representation of the frequencies of the data over discrete intervals (*bins*). It is widely used due to its simplicity, but it has several shortcomings, such as the lack of continuity. The KDE technique relies on assigning a kernel function K to each sample (observation in the dataset), and then summing all the kernels to obtain the estimate. In contrast to the histogram, KDE constructs a smooth probability density function, which may reflect more accurately the real distribution of the data. We now describe the KDE technique in more detail.

Given a multi-variate sample set $(\boldsymbol{x}_1,\boldsymbol{x}_2\ldots\boldsymbol{x}_n)$ where $\boldsymbol{x}_i$ is the i^{th} sample value from an unknown density f, KDE builds an estimation the following way

$$\hat{f}(\boldsymbol{x}) = \frac{1}{n}\sum_{i=1}^{n} K_H(\boldsymbol{x}-\boldsymbol{x}_i) \tag{2}$$

where:

- n denotes the cardinality of the sample set.
- $K_H(\boldsymbol{x}) = |H|^{-1/2}K(H^{-1/2}\boldsymbol{x})$.
- $K(\boldsymbol{x})$ refers to the kernel function used to define the weight or influence of each sample.
- H is a $d \times d$ dimensional diagonal matrix containing the *bandwidth* or *smoothing parameter* value for each dimension.

Intuitively, the kernel estimator is a sum of 'bumps' placed at the sample points. The kernel function K determines the shape of the bumps, while the smoothing factor h determines their width. As an example of KDE, Figure 1 depicts a simple estimator for a one-dimensional (1D) space with three sample points. Each of the sample points is surrounded by a kernel depicted as a bell and the estimator is depicted as the thick line over the kernels. Note that, even though a density estimation is a continuous function, KDE programs generate as output the values of the estimation in the discretized space requested by the user.

The bandwidth parameter h controls the influence area and smoothness of kernels. It is used to reduce the

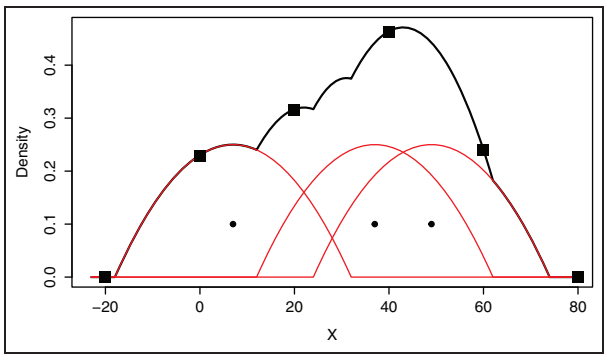


Figure 1. Example of KDE for 1D data. Small dots represent the samples, and the red lines are the kernels around them. The estimated density is computed at the indicated points (square dots) in a discretized grid with a step of 20.

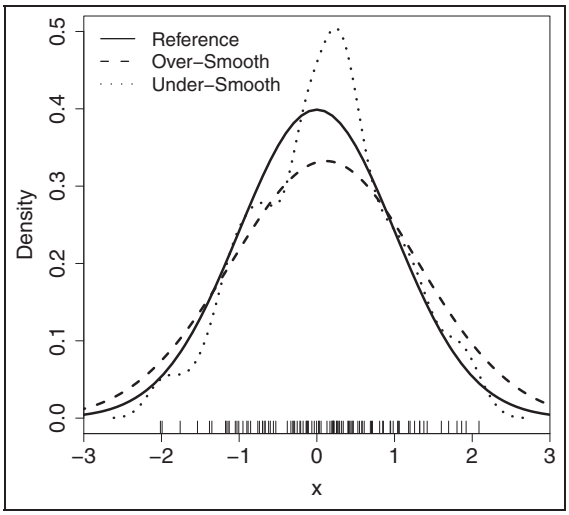


Figure 2. Effect of the smoothing parameter when estimating data sampled from a normal distribution.

noise in the density estimation, and it must be carefully selected. To describe its effect, in Figure 2 we depict a simulated random sample from the Gaussian distribution in 1D, and different estimations of the density derived from those samples. The solid curve depicts the real density of the data, while the other lines depict (kernel-based) density estimations using different values of h. The dashed line corresponds to a large h, and results in an over-smoothed estimator. In contrast, the dotted line corresponds to a small h and results in an exceedingly sharp estimator. None of the cases reflect accurately the actual density of the samples. Several techniques to aid in the selection of the optimum bandwidth are detailed elsewhere (Silverman, 1986; Scott, 2009).

A kernel function K is a symmetric but not necessarily positive function that integrates to one. Kernels can be classified into two groups: *bounded* and *unbounded*,

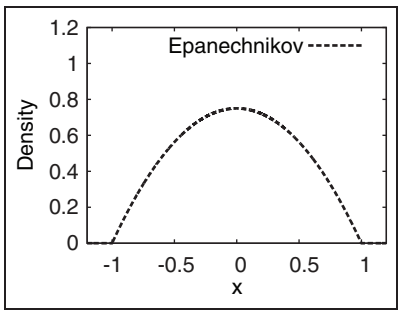


Figure 3. Shape of a Epanechnikov kernel in 1D.

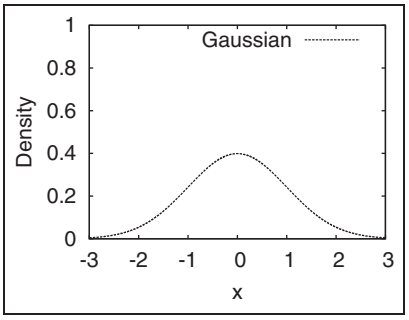


Figure 4. Shape of a Gaussian kernel in 1D.

depending on their area of influence. Two widely used kernel functions are *Epanechnikov* (which is bounded) and *Gaussian* (which is unbounded). They are defined in equations (3) and (4) respectively. In equation (3), c_d is the volume of the unit d-dimensional sphere: $c_1 = 2, c_2 = \pi, c_3 = 4\pi/3, \ldots$. For the sake of clarity, we have depicted in Figures 3 and 4 the Epanechnikov and Gaussian kernels for 1D spaces. Note how bounded kernels only affect those points in the space at a limited distance (1 in the figure), while the influence area of unbounded kernels spans infinitely in both directions.

$$K(\boldsymbol{x}) = \begin{cases} \frac{1}{2}c_d^{-1}(d+2)(1-\boldsymbol{x}^{\mathrm{T}}\boldsymbol{x}) & \text{if } \boldsymbol{x}^{\mathrm{T}}\boldsymbol{x} < 1 \\ 0 & \textit{otherwise} \end{cases} \quad (3)$$

$$K(\boldsymbol{x}) = \frac{1}{\sqrt{(2\pi)^d}} \exp\left(-\frac{1}{2}\boldsymbol{x}^{\mathrm{T}}\boldsymbol{x}\right) \quad (4)$$

The choice of the particular kernel to apply is up to the KDE user, taking into account the problem at hand. According to Silverman (1986), asymptotically, there are no differences between the different kernels at hand. Moreover, he states that it is desirable to base the choice of the kernel on other considerations, such as the degree of differentiability required or the computational effort involved. In particular, he describes how, when the right value of h is chosen, almost all the popular kernels produce a relatively small mean integrated square error in the estimation.

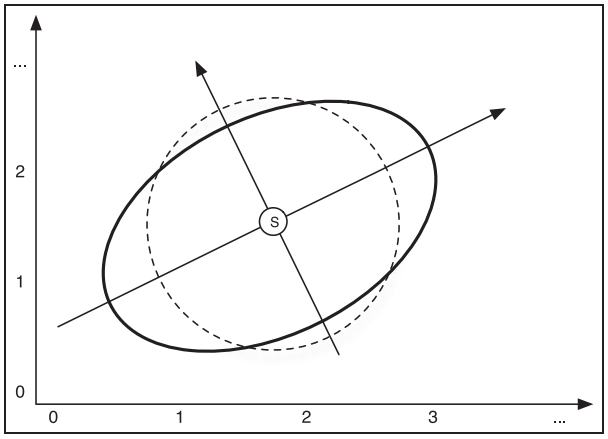


Figure 5. Example of an original and its corresponding whitened radial symmetric kernel in 2D.

When dealing with multi-variate scenarios, considering all dimensions as if they were on the same scale could lead to poor estimates, particularly when the data has very different variability in each dimension (Givens and Hoeting, 2005). To tackle this issue, Fukunaga (1990) suggested: (1) transforming the data to have a unit covariance matrix; (2) smoothing it using a radially symmetric kernel; and (3) transforming it back. The idea is to adapt the shape of the kernel to the distribution of the data, as depicted in Figure 5 for a two-dimensional (2D) space, where the dashed circle represents the original kernel and the ellipse is the transformed one. The distribution of the data is represented by the sample covariance matrix $\boldsymbol{\Sigma}$, which contains the variances and covariances between the d sample vectors of the dataset. If the original $\boldsymbol{\Sigma}$ matrix is unknown, an estimated $\hat{\boldsymbol{\Sigma}}$ must be created. Then, following Fukunaga (1990), equation (2) is adapted this way

$$\hat{f}(\boldsymbol{x}) = \frac{|\hat{\boldsymbol{\Sigma}}|^{-1/2}}{nh^d} \sum_{i=1}^{n} K\left(h^{-2}(\boldsymbol{x}-\boldsymbol{x}_i)^T \hat{\boldsymbol{\Sigma}}^{-1}(\boldsymbol{x}-\boldsymbol{x}_i)\right) \quad (5)$$

This transformation is called *whitening* the data and the effect is that the obtained estimator is more accurate, as the kernel is adapted to the distribution of the data. This approach also allows us to use a single, scalar h value for all the dimensions.

3 An efficient algorithm to compute kernel density estimation

KDE generates an estimation of the probability density function of a sample set. This estimation is computed in a user-defined *evaluation space* (or *evaluation grid*) that should include all the observation points in the sample set. In the example of Figure 1, the evaluation space spans from −20 to 80, covering the three samples plus some 'extra' space at the boundaries.

Algorithm 1 Evaluation-point-wise KDE (EP-KDE)

```
for each Evaluation Point e do
  value(e) = 0
  for each Sample sdo
    dist = computeDistance(e, s)
    value(e) + = computeDensity(dist)
  end for
end for
```

KDE implementations discretize this space, computing the density in equally separated *grid points*. This way, the evaluation space can be represented as a multi-dimensional matrix. The separation between grid points is defined by a user-provided *evaluation step* or *grid step*, that can be different in each dimension. In Figure 1, the density of the space will be computed for six evaluation points (−20, 0, 20, 40, 60 and 80), which are equally separated at distance 20.

The accuracy of the estimation generated by a KDE program, thus, depends on several parameters that must be provided by the user:

- The choice of kernel function.
- The bandwidth parameter.
- The evaluation space, determined by a vector of bounds.
- The grid step vector.

Most programs offer default values for these parameters, or heuristics to compute them. In the following sections we assume that these parameters are known, that the input of the KDE program is a dataset containing the samples, and that the output is a multi-dimensional matrix with the density estimation at all the evaluation points within the discretized evaluation space defined by the user.

3.1 Methods to compute KDE

KDE is commonly computed as described in Algorithm 1. For each evaluation point, the combined density that all the samples generate on it is computed, which depends on the kernel of choice. This computation requires measuring the distance between the evaluation point and each sample. The combined density at the evaluation point is the sum of all the partial densities. We call this approach *evaluation-point-wise KDE* or *EP-KDE* for short.

The computational complexity of EP-KDE is $O(k_d mn)$, where k_d is a constant related to the dimensionality of the dataset, m is the number of evaluation points, and n the number of samples. Note that m is proportional to the size of the evaluation space and the grid step. For large, multi-dimensional spaces or/and tight grid steps, m can be *huge*.

Algorithm 2 Sample-wise KDE (S-KDE)

```
for each Evaluation Point e do
  value(e) = 0
end for
b = computeBoundingBox
for each Sample sdo
  b_s = adjustBoundingBox(b, s)
  for each Evaluation Point e in b_s do
    dist = computeDistance(e, s)
    value(e) += computeDensity(dist)
   end for
end for
```

EP-KDE is directly parallelizable in a data-parallel fashion: the computations affecting a given evaluation point are independent of those affecting other points. If a thread is in charge of computing the density in an evaluation point, all the threads can progress concurrently without any sort of memory-write contention.

The EP-KDE approach is valid for both unbounded (e.g. Gaussian) and bounded (e.g. Epanechnikov) kernels. In the first case, all the samples affect all the evaluation points. In contrast, with bounded kernels, samples only contribute to the density in those evaluation points within their influence area. Therefore, using EP-KDE, in most cases the *computeDistance* function of Algorithm 1 will return a value outside the bounds of the kernel and, therefore, function *computeDensity* will return 0.

As EP-KDE with bounded kernels can lead to a huge number of worthless computations (that would depend on the size of the evaluation space and on the dispersion of the samples), we have developed a more efficient algorithm for this group of bounded kernels. It is described in Algorithm 2, and we call it *sample-wise KDE* or *S-KDE* for short. Instead of focusing one-by-one on each evaluation point and on the influence of all samples over it, S-KDE focuses one-by-one on each sample, computing its influence on the evaluation points surrounding it. As the kernel is bounded, the area of influence is confined within a *bounding box*, which is computed in advance. This bounding box is a hyperrectangle whose dimensions are determined by the maximum per-dimension distances of influence of the kernel. Note that this is kernel-dependent, but not sample-dependent. Thus, the size and shape of the bounding box $\boldsymbol{b}$ is computed just once. Then, for each sample the bounding box must be aligned to the evaluation grid, defining the per-sample bounding box $\boldsymbol{b_s}$.

With S-KDE, computations of distance and summations of influences are greatly reduced: for each sample, most evaluation points fall outside the influence area of the kernel, and are not considered in subsequent computations. The complexity of S-KDE is $O(k_d np)$, where k_d is a constant related to the dimensionality of the dataset, n is the number of samples and p is the size of the bounding box, which is expected to be much smaller than the total number of evaluation points m.

S-KDE is also naturally data-parallel, using a per-sample approach. However, two concurrent threads computing the partial densities of two different samples with overlapping influence areas can incur memory-write contention. Therefore, a mechanism to coordinate memory accesses must be put in place.

3.2 Computing the bounding box

An efficient implementation of S-KDE requires the computation of the bounding boxes, which surround the area of influence of each sample. In this section we describe how b can be computed, assuming that data is transformed using the Fukunaga approach described in Section 2. Our proposal is based on a method described in Fukunaga (1990). The area of influence of a sample using the (transformed) kernel has an elliptic shape in 2D spaces as depicted in Figure 5, and a d-dimensional ellipsoid shape for spaces of higher dimensions. The challenge is to find a box bounding the ellipsoid, and to align it to the discretized grid that defines the evaluation space. In Figure 6 we depict an example: a sample s in position *(6.75, 4.5)*, its area of influence (the ellipse), and the bounding, aligned rectangle (dashed line).

The starting point of the calculation is the covariance matrix of the sample set. It contains the information to determine the shape and size of the ellipsoid. In addition, the smoothing parameter h, which modifies the size of the kernel, must be taken into account. We first compute $\boldsymbol{\Sigma}' = \boldsymbol{\Sigma}h^2$, where $\boldsymbol{\Sigma}'$ is the modified covariance matrix with the applied bandwidth value. Then, the semi-axes of the ellipsoid are computed as the eigenvalues of $\boldsymbol{\Sigma}'$, and its orientations as the eigenvectors of that matrix. After this, the next step is to compute

$$\Theta = \sqrt{\lambda}\boldsymbol{I} \tag{6}$$

where Θ is a diagonal matrix containing the square-root values of the $\boldsymbol{\lambda}$ eigenvalues vector. I represents the identity matrix. The square root of each eigenvalue gives the length of each semi-axis from the center of the ellipsoid, but aligned to the coordinates axis. Thus, the matrix Θ of eigenvalues must be multiplied by the Φ eigenvectors matrix to transform the axes of the ellipsoid (to 'rotate' them)

$$\boldsymbol{Y} = \Phi\Theta \tag{7}$$

The result is a matrix $\boldsymbol{Y} = \left[\boldsymbol{y}_1^{\mathrm{T}}\boldsymbol{y}_2^{\mathrm{T}}\ldots\boldsymbol{y}_d^{\mathrm{T}}\right]$ where each $\boldsymbol{y}_i^{\mathrm{T}}$ column contains the coordinates of the semi-axes of the ellipsoid.

From this, a vector of distance $\boldsymbol{b}$ is obtained which determines, per direction, the absolute value of the

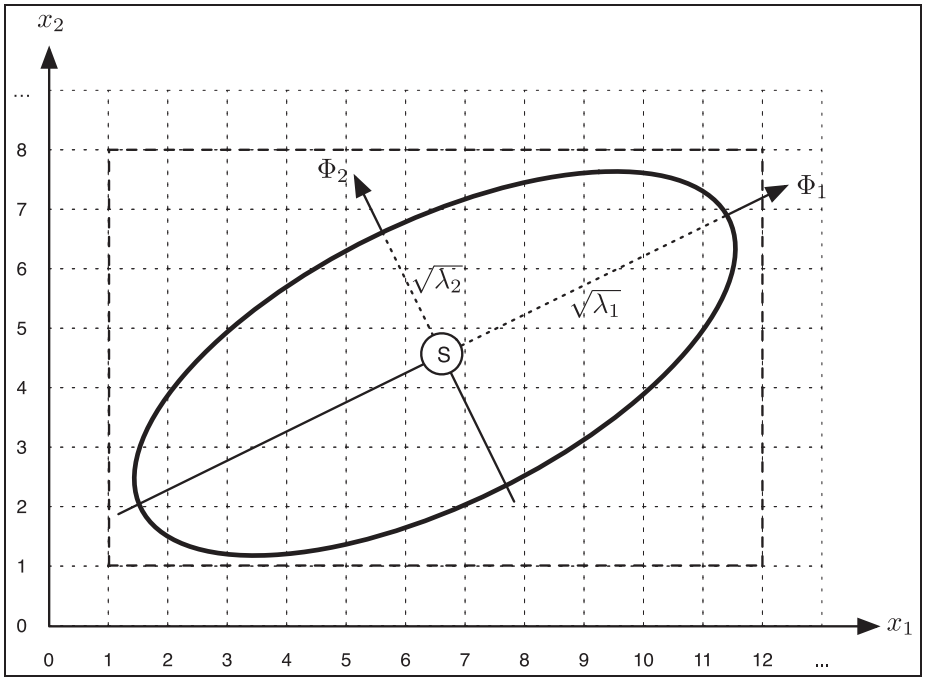


Figure 6. Area of influence of a sample and the corresponding bounding box.

maximum distance at which evaluation points are influenced by the kernel

$$b_i = ||\boldsymbol{y}_i|| \tag{8}$$

where $||\cdot||$ is the Euclidean norm operator. In other words, each element of vector $\boldsymbol{b}$ defines half the length of each edge of the hyperrectangle surrounding the kernel. Once $\boldsymbol{b}$ is known (note that this computation is done just once), the actual, aligned per-sample bounding box $\boldsymbol{b_s}$ can be computed, adding and subtracting $\boldsymbol{b}$ from/to the absolute coordinates of the sample, and rounding up the resulting values.

3.3 From d -dimensional bounding boxes to two-dimensional bounding rectangles

Using the process described in the previous section, we fit a d-dimensional hyperrectangle-shaped box around the ellipsoid-shaped influence area of a sample, as determined by the kernel function. The evaluation points influenced by the sample are *only* those within the ellipsoid, which means that the box contains evaluation points not belonging to the influence area of the sample. In fact, depending on the shape and angle of the ellipsoid, the evaluation points outside the ellipsoid could be more numerous than those inside it.

Motivated by this fact, we define a way to make a tighter delimitation of bounding boxes. The first idea is to reduce the dimensionality of the problem, splitting (chopping) the original d-dimensional box into several non-overlapping $(d-1)$-dimensional boxes. Each of them is again chopped into $(d-2)$-dimensional boxes, repeating the process until a collection of 2D rectangles (slices) is obtained. Then, each rectangle is cropped, reducing it to a smaller one to minimize the number of evaluation points not influenced by the sample. This two-step procedure is represented in Figures 7 and 8.

The number of slices per bounding box will be determined by the size of the evaluation space and the grid step. Without loss of generality, we will use a 3D space as an example. The chopping is performed along the third dimension of the space, which corresponds to the z-axis in the Cartesian coordinate space. For example, if a bounding box is located between coordinates 8 and 12 of the z-axis, and the step in the z-axis is 0.5, there would be 9 slices. Once 2D slices are obtained, the cropping is performed to reduce the 2D bounding box to make it surround the ellipse tightly. This process involves several steps that are described below and in Figure 9. A detailed description of this technique is carried out in the Appendix.

- Calculate the rotation angle of the ellipse in the slice.
- Calculate the length of the principal axes of the ellipse.
- Using the rotation and the length of the principal axes, calculate the coordinates of the edge vertices of the ellipse.
- Using the coordinates of the edge vertices, calculate the boundaries of the ellipse. Then, align the boundaries to the evaluation grid and calculate coordinates for the minimal bounding box.

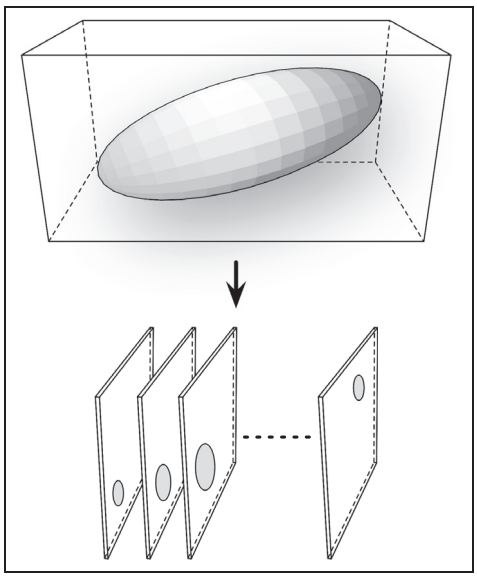


Figure 7. Chopping a 3D bounding box into 2D slices.

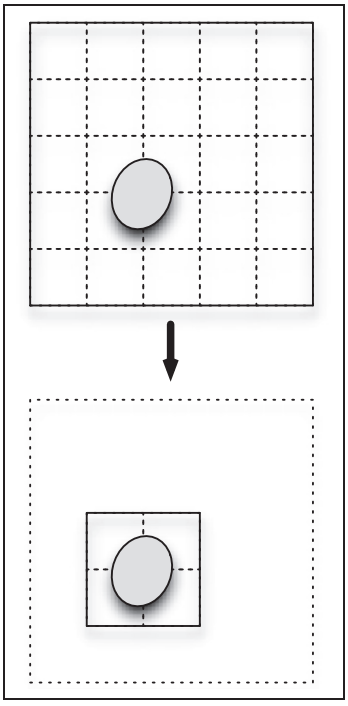


Figure 8. Cropping a 2D slice to obtain a minimum-size bounding rectangle.

4 Parallel implementation of S-KDE

The S-KDE algorithm described in Section 3 has a much lower computational complexity than the commonly used EP-KDE algorithm. However, it can be computationally challenging when dealing with large datasets and/or large, dense evaluation spaces. Thus, we have implemented it from the beginning as a data-parallel program, targeting multi-core and many-core processors.

4.1 Target hardware

The current landscape of HPC systems presents a wide variety of compute devices for the execution of massively parallel codes, from multi-core processors to accelerators with hundreds of processing units, such as GPUs. For our work we have chosen multi-core processors, due to their widespread presence, and many-core coprocessors, due to their growing popularity in the HPC community and their architectural similarity with multi-core processors (which simplifies code portability).

In particular, we have used an Intel Sandy Bridge-E multi-core processor and an Intel Xeon Phi coprocessor in the tests reported in this work. Both share the Intel 64-bit x86 architecture and Intel development tools (compiler suites, VTune Amplifier, …), but have strong architectural differences: e.g. Sandy Bridge-E processors hold at most 8 physical cores with 256-bit-wide vector units supporting SSE and AVX instruction sets. Xeon Phi coprocessors hold up to 61 computing cores with 512-bit-wide vector units and a different vector instruction set, AVX-512. Furthermore, Sandy Bridge-E processors have out-of-order execution capable cores and hold up to 20 MB of shared L3 cache, while Xeon Phi coprocessors are designed for in-order execution, and share up to 30 MB of L2 cache among all their cores. More detailed descriptions of the hardware architecture of these devices can be found in Jeffers and Reinders (2013), and a discussion about the programming difficulties of these devices in Lopez-Novoa et al. (2015).

4.2 Implementation details

We have developed our code in ANSI-C, and parallelization is achieved by making use of OpenMP and Intel compiler directives. In particular, we use OpenMP directives to define how tasks are mapped to threads, and the *#pragma simd* directive to vectorize parts of the code. In addition, we use the Meschach math library (Leyk and Stewart, 1994) for some matrix operations, such as the computation of eigenvalues and eigenvectors. All computations are performed in double precision.

The workflow of our S-KDE code is depicted in Figure 10. Each thread is in charge of computing the influence of a set of samples over the evaluation space. For each sample, the thread first adjusts its bounding box (as defined in Section 3.2). If the evaluation space is of dimensionality three or higher, the chop-and-crop procedure described in Section 3.3 is recursively applied

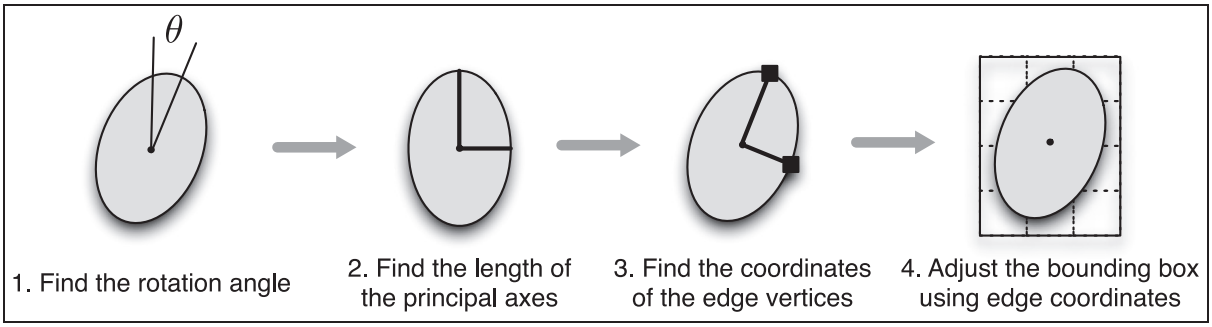


Figure 9. Steps to perform the cropping of a 2D bounding box.

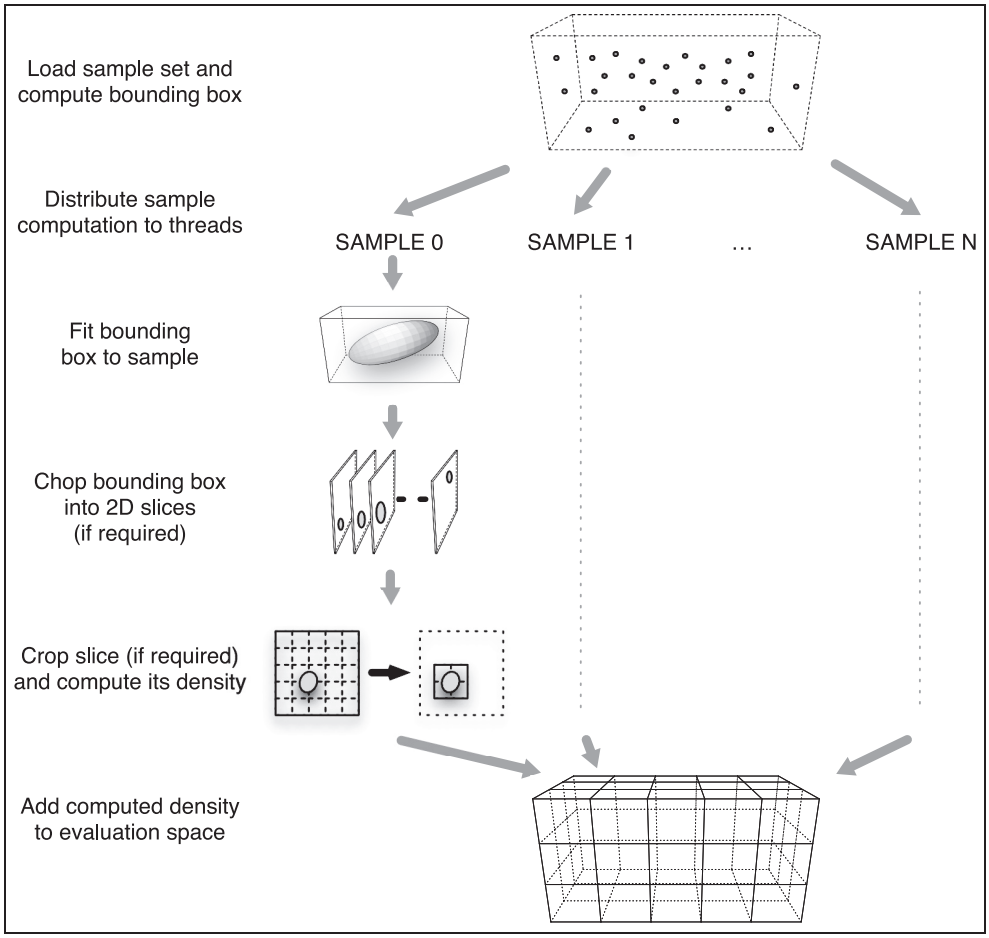


Figure 10. Workflow of our S-KDE implementation.

to reduce the computation to 2D slices. Then, for each 2D bounding rectangle the density that the sample creates in each evaluation point is computed.

One of the drawbacks of performing sample-wise computations is the memory contention that may appear when two or more different threads, managing samples whose influence area overlap, have to add partial density values into the memory positions that represent the same evaluation point. To reduce this harmful effect, each thread calculates every row of the slice in its private memory, and adds it into main memory using the *atomic* OpenMP pragma. This way, we ensure data write consistency. There is a cost to pay, though: atomic operations causes overheads due to the serialization of memory write operations. We analyze this overhead in detail in Section 5.

Regarding the Xeon Phi, one of our target platforms, it is worth mentioning that this device has two

working modes: offload and native. In the former, all the code runs on a host processor and just some parts are offloaded to the coprocessor. In the latter, the whole code runs in the Xeon Phi, which behaves as a separate computer. In our tests we have used the native mode, which requires cross-compilation, in the host computer, of the code and the companion libraries.

4.3 Other KDE implementations

We have found an extensive body of literature on KDE implementations, both general and those targeting specific computing platforms. We review this literature and describe it classified according to the algorithmic approach used to compute the KDE, with a focus on parallel implementations.

4.3.1 Evaluation-point-wise implementations. Initial parallel KDE implementations were based on MPI, distributing the workload among several computing nodes of a multi-computer. One of these implementations was presented by Racine (2002). A more complete version was introduced by Łukasik (2007), introducing approaches for load balancing and parallel techniques to compute some of the KDE parameters.

More recent GPU-based implementations of KDE are also available. We have identified two CUDA implementations of KDE, one by Srinivasan et al. (2010) and another by Michailidis and Margaritis (2013). As the code of the latter was not publicly available, we will focus on the former, called GPUML. It offers an interface for C/C++ and Matlab that provides CUDA implementations of several weighted kernel summation and matrix construction algorithms. It provides support for several kernel functions (including Gaussian and Epanechnikov). To speed-up the computations, the program implements data-reuse mechanisms on the shared memory of the GPU. In Section 5 we will compare the execution times of S-KDE and GPUML using in both cases Epanechnikov kernels.

We also tried to compare our code with some KDE implementations for the widely used Python environment, but they were not capable of running our experimental tests. The function *Gaussian_KDE* from *SciPy* (Millman and Aivazis, 2011) package seemed the most popular choice among Python practitioners, but it was not capable of loading our sample datasets, raising memory overflow errors. Function *KDEmulti-variate* from *StatsModel* (Seabold and Perktold, 2010) successfully loaded our datasets, but its performance was exceedingly poor. We linked our Python environment with the highly optimized Math Kernel Library[1] (MKL) by Intel. MKL provides vectorized and multithreaded versions of math routines, such as those in BLAS or LAPACK. Unfortunately, *StatsModel* seemed to take no benefit from these optimized math functions, and worked in a single-core fashion. In addition, due to its EP-KDE approach, its computational complexity is much higher than that of our S-KDE code. Thus, our experiments with *KDEmultivariate* were unfeasible in computation time, and we left them out of this work.

4.3.2 Sample-wise implementations. Within the popular R software environment, package *ks* (Duong, 2007) includes a *kde* function whose implementation holds similarities with our S-KDE: it performs computations in a sample-wise fashion, but using a Gaussian kernel. A kind-of bounding box around the Gaussian kernel is defined, using a parameter to limit its influence area. The default value for the threshold preserves at least 99% of the Gaussian kernel influence.

This approach makes it feasible to apply the S-KDE approach but, at the expense of losing precision in the influence area of the kernel. Our choice was to use the Epanechnikov kernel to make the computations, avoiding a potential loss of precision. In addition, our chop-and-crop technique allows us to work with large, multi-dimensional datasets, whose processing would be unfeasible otherwise.

As an additional feature, *kde* in *ks* includes a mode to compute the density estimation in a list of points provided by the user instead of a complete evaluation grid. This mode allows the user to perform KDE in spaces of dimensionality greater than three without excessive execution times, if the list not too long.

As we did with Python, we linked the R environment with Intel MKL. In this set-up, we verified that *ks-kde* effectively took advantage of multi-core CPUs for matrix operations. We include in Section 5 a comparison of S-KDE against R's *ks-kde*. Regarding the kernel influence threshold, we have used the default value.

5 Evaluation of S-KDE

In this section, we describe the experiments we undertook to analyze the performance of our S-KDE implementation under different configurations, comparing the execution times with those of an evaluation-point-wise version of KDE (EP-KDE), R's *ks-kde* and GPUML. We also identify possible ways of improving S-KDE performance even further.

5.1 Testing environment

As detailed in Section 4.1, we tested our S-KDE code in two different hardware platforms, a multi-core processor and a many-core coprocessor. In particular, we used an Intel Core i7 3820 CPU and an Intel Xeon Phi 3120A coprocessor. We also used an NVIDIA GTX 650 GPU for some comparison tests involving

Table 1. Hardware features of the processors used to run KDE. Note that the RAM in the i7 is the one installed in the computer powered by that processor, while the other reported values are those of the accelerator built-in memory.

	Core i7 3820	Xeon Phi 3120A	GTX 650
Architecture	Sandy Bridge-E	MIC	Kepler
Number of cores	4	57	384
Number of threads	8	228	384
Clock speed	3.6 GHz	1.1 GHz	1.05 GHz
Vector width	256 bit	512 bit	1024 bit
Main memory	8 GB DDR3	6 GB GDDR5	1 GB GDDR5

GPUML. The main features of these processors are summarized in Table 1.

As explained before, the complexity (or problem size, which determines the execution time) of S-KDE depends on two factors: the number of samples in the dataset and the number of evaluation points in each per-sample bounding box. The latter depends on three parameters: the dimensionality of the problem, the distance between the evaluation points (or per-dimension evaluation step) and the bandwidth.

In this work, we have performed several tests of KDE codes varying the size of the evaluation space for four different datasets (two 2D and two 3D). We have fixed the boundaries of the evaluation space, but the per-dimension evaluation step has been modified in order to increase or reduce the problem size.

For 2D tests we have used datasets from an actual, real-word problem: DNA sequencing. The datasets were generated by IonTorrent sequencing machines (Rothberg et al., 2011), which monitor millions of simultaneous sequencing reactions, and additionally produce reports with the collected information. The IonTorrent community makes publicly available some sample datasets corresponding to their AmpliSeq Cancer sequencing machines, along with their corresponding sample reports. In particular, a part of the generated reports is the density map of the chip, and in the tests reported in this paper we have reproduced that computation. More information about the Ion chips and the available datasets can be found at http://ion-community.lifetechnologies.com. We used the datasets reported for the Ion 316 and Ion 318 chips, which contain 3,473,932 and 5,885,326 samples each. The sizes of the evaluation spaces used in the tests (not in terms of actual, physical sizes but in terms of the number of points on which the density is estimated) are listed in Tables 2 and 3. Note that more points means tighter grids.

For the 3D tests we created two synthetic datasets using the *mvrnorm* function from *MASS* library within the R framework (Venables and Ripley, 2002). These datasets have 500,000 and 1,000,000 samples each. As we did with the 2D tests, we fixed the boundaries but modified the steps in order to compute problems of different sizes, working with lighter or denser evaluation spaces. The actual numbers of evaluation points per dimension for each test are listed in Table 4.

Table 2. Size of the evaluation spaces (number of grid points) used with the Ion 316 chip dataset.

Dim X	Dim Y	Total
560	540	302,400
560	1350	756,000
1400	1350	1,890,000
1400	2700	3,780,000
2800	2700	7,560,000

Table 3. Size of the evaluation spaces (number of grid points) used with the Ion 318 chip dataset.

Dim X	Dim Y	Total
760	680	516,800
760	1700	1,292,000
1900	1700	3,230,000
1900	3400	6,460,000
3800	3400	12,920,000

Table 4. Size of the evaluation spaces (number of grid points) used with the 3D datasets.

Dim X	Dim Y	Dim Z	Total
110	220	322	7,792,400
110	440	322	15,584,800
220	440	322	31,169,600
220	440	805	77,924,000
220	1100	805	194,810,000

In all these tests, the bandwidth value is a relevant parameter to be taken into account. It modifies the shape and spread of the kernel, that is its influence area, which impacts on the number of computations to be performed per sample. In this work we use a heuristic detailed in Silverman (1986) to compute an appropriate bandwidth for a given dataset:

$$h = A(K)n^{-1/(d+4)} \quad (9)$$

where n is the number of observed samples and d the number of dimensions. $A(K)$ is a constant calculated as

$$A(K) = \left\{8c_d^{-1}(d+4)(2\sqrt{\pi})^d\right\}^{1/(d+4)} \quad (10)$$

where c_d is the volume of the unit d-dimensional sphere (e.g. π for 2D or $4\pi/3$ for 3D).

Regarding compilation details, S-KDE was compiled for both the Core i7 CPU and the Xeon Phi coprocessor using the Intel C Compiler version 14.0.1 with $-$ *O2* optimization. The cross-compilation for the Xeon Phi required the $-$ *mmic* flag.

5.2 Measuring the performance of S-KDE

In this section we report the results of some tests carried out to evaluate the performance of our S-KDE parallel implementation. In order to provide a meaningful comparison, we developed a parallel implementation of an evaluation-point-wise approach (EP-KDE onwards), also using OpenMP. We will compare S-KDE with R's *ks-kde* (supported by MKL) and GPUML too. Our purpose is to confirm that our approach represents a significant improvement over state-of-the-art KDE implementations. Results confirm this fact: our code is, by far, the fastest one.

As we are testing different programs, that compute KDE using different approaches and with different kernel functions, we need to set a 'fair' comparison basis. We have tried to run the programs in such a way that they produce the same or very similar results. To measure this similarity we have used the score function described by Perkins et al. (2007), which returns the *similitude value* between two different density functions. Note that two different algorithm–kernel combinations may require different values of bandwidth in order to generate the same density estimation.

Our comparison procedure is as follows. We first compute the density estimation (DE) for a given dataset and evaluation space with our code, using the bandwidth value provided by the heuristic described above (Section 5.1), and measure the execution time. Then we run a competitor program (for example, GPUML) with the same dataset and evaluation space, but varying the bandwidth value until the resulting DE reaches a similitude score over 98%. The execution time of the run passing this similitude threshold is the one assigned to the competitor program. In the actual tests, the bandwidth values used with *ks-kde* resulted in a 98.2% similitude on average; for GPUML, similitude was 99.1% on average.

In the test, S-KDE and EP-KDE were executed in a multi-core i7 processor and in a Xeon Phi coprocessor. The *ks-kde* was executed in the multi-core processor (taking advantage of the parallel processing capabilities of MKL), and GPUML in a GTX 650 GPU. Each program–platform combination was used with the four datasets, varying the problem size (using different steps in the evaluation space). All the measured execution times are summarized in Figures 11(a) and 11(b) for the 2D IonTorrent datasets, and in Figures 11(c) and 11(d) for the 3D synthetic datasets. Note that GPUML data for the 3D experiments with large evaluation spaces are not reported because they could not be executed, due to limitations in the memory size of our GTX 650 GPU.

The first conclusion is that, in all the tested cases, S-KDE is significantly faster than the competitors. We show in Table 5 the speed-up of S-KDE running in both the Core i7 and the Xeon Phi compared to *ks-kde*, GPUML and EP-KDE.

The long execution times of *ks-kde* can be easily explained considering that: (1) R is an interpreted environment and, therefore, R programs have higher runtime overheads than compiled programs; (2) the bounding boxes used in *ks-kde* to prune Gaussian kernels are larger than those defined by Epanechnikov kernels and, thus, require higher computational effort; and (3) *ks-kde* does not implement the chop-and-crop technique to reduce computations in problems of dimensionality three and higher. The performance improvement derived from this feature of S-KDE will be analyzed in detail later, in Section 5.3.

Regarding GPUML and EP-KDE, the large execution times are a consequence of the evaluation-point-wise algorithm implemented: the whole evaluation space is swept, point-by-point. For each evaluation point, the influence that each and every sample has on it is computed—even when, in many cases, this influence is zero because the used kernel is bounded and the point falls outside the influence area of the sample. In addition, GPUML implementation suffers from overheads derived from the use of a GPU-based discrete accelerator, the cost of transferring input data and results from/to main memory to/from GPU memory (through a PCIe connection) being the main one. It is also worth mentioning that the size of the GPU memory imposes limits on the size of the problems: note that our GPU has only 1 GB of memory to store the d-dimensional matrix representing the evaluation space plus other data structures, while the i7 has 8 GB and the Xeon Phi has 6 GB.

5.3 Assessing the benefits of parallel computing and the chop-and-crop technique

The analysis in the previous section presents a high-level, black box evaluation of the performance of S-KDE. In this section we explore further into the code, in order to better understand where the performance gains are coming from. There are three basic elements contributing to a fast KDE program: (1) the sample-

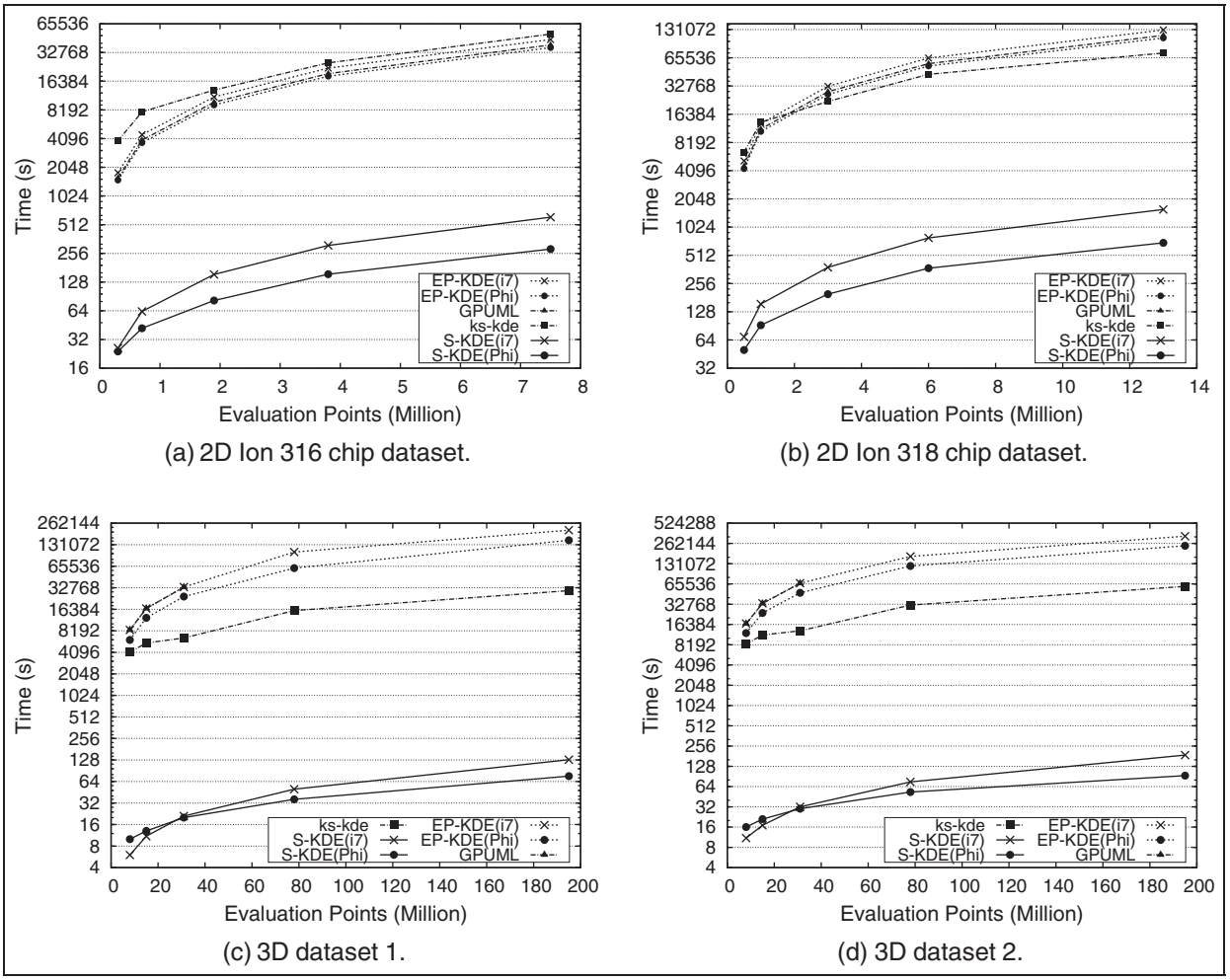


Figure 11. Execution times of the KDE implementations under test, for different datasets.

Table 5. Average speed-up of S-KDE (in i7 and Xeon Phi) compared to state-of-the-art KDE implementations.

	2D S-KDE (i7)	2D S-KDE (Phi)	3D S-KDE (i7)	3D S-KDE (Phi)
EP-KDE (i7)	76.4	138.5	1759.1	1797.9
EP-KDE (Phi)	62.9	114.0	1241.9	1257.5
GPUML	66.9	121.4	1628.3	1397.1
ks-kde	85.8	145.5	443.5	463.8

wise density estimation approach; (2) the chop-and-crop technique; and (3) the use of parallel processors. When needed, we compare our code with *ks-kde* which also uses bounding boxes and, through MKL, can take advantage of parallel hardware—but does not implement chop-and-crop. Also, we focus on a particular dataset: the 3D dataset 1. This is just to simplify tables and graphs. Results for other datasets show the same trends.

5.3.1 Benefits of parallel processing. S-KDE can run as sequential code, thus allowing us to evaluate the 'algorithmic' performance of our sample-wise approach combined with chop-and-crop. We did some tests running S-KDE with a single thread (that is, in a single core) in the four-core i7 platform, and found that the execution time is, on average 4.28 times longer. This is for 3D dataset 1, but the experiments for the remaining datasets report similar slow-down values. Combining these results with those in Table 5, we find that the sequential S-KDE is still between 20 and 100 times faster than *ks-kde* running in the i7. However, as both S-KDE and *ks-kde* have been measured in the same i7 and, through the MKL libraries, *ks-kde* can take advantage of the underlying parallel hardware, we consider

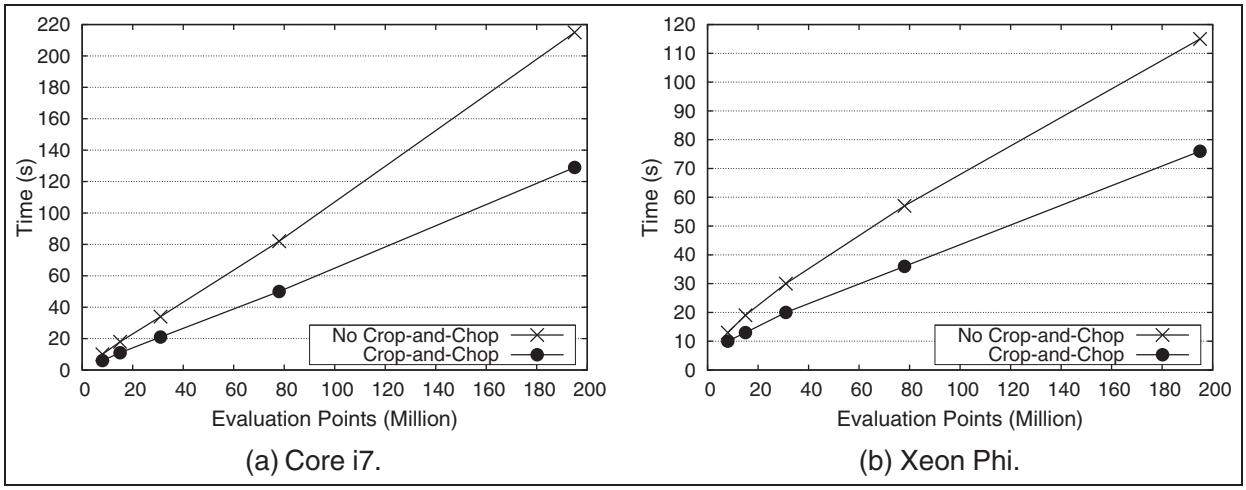


Figure 12. Execution times for 3D dataset 1 with crop-and-chop enabled/disabled.

that the reported comparisons yielding 85.8 and 443.5 speed-ups for S-KDE are fair. The Xeon Phi accelerator can add some extra performance when using all its cores simultaneously.

5.3.2 Benefits of chop-and-crop. As explained before, when dealing with datasets of dimensionality three or higher, our code uses a crop-and-chop process to avoid computations involving evaluation points contained in a bounding box but not affected by a sample. In order to assess the performance gains derived from using this technique, we ran tests with the 3D dataset 1, with crop-and-chop disabled (meaning that the per-sample bounding box is a rectangular prism) or enabled (meaning that, for each sample, a collection of bounding rectangles of different sizes is processed). The results are reported in Figure 12(a) (S-KDE in the i7) and 12(b) (S-KDE in the Xeon Phi). These tests clearly show the performance gains obtained with crop-and-chop: S-KDE runs 60.4% faster in the Core i7, and 49.2% faster in the Xeon Phi, due to the removal of useless computations (and the corresponding memory accesses). For example, in the tests with 194.81 million evaluation points, each 3D rectangular bounding box contains 102,461 points on average; crop-and-chop reduces the number of processed evaluation points by half to 53,511.

5.4 Identifying performance bottlenecks

Although the execution speed achieved by S-KDE in the two tested platforms is very competitive, we have performed additional tests and analyses in order to better understand the behavior of our code, and to identify possible bottlenecks and opportunities to improve its performance. To that extent we have used Intel's VTune Amplifier, which provides detailed information about the time spent by the program running different portions of the code. We have been able to dissect the execution times into three portions: (1) computation, i.e. the time devoted to calculations of KDE, without taking into account memory writes; (2) memory writes, i.e. the time spent in synchronized memory writes of partial results; and (3) overheads, i.e. initialization, ending and additional operations, including I/O.

As this dissection is very similar for all the tested scenarios, we focus again on the 3D dataset 1, depicting the results in Figure 13(a). It shows that, on average, the code spends half the time computing KDE, and the other half performing atomic memory writes. Overheads are almost negligible for the datasets and evaluation spaces that we tested.

Time spent in writing operations may appear excessive, but we have to consider that any algorithm computing KDE requires extensive traversals of large datasets: it is a memory-bounded problem. In our particular case, S-KDE needs to consolidate into main memory the per-sample results of the partial densities for each evaluation point inside the bounding boxes. Additionally, bounding boxes can overlap: the influence of several samples on the same evaluation point has to be aggregated. As different threads take care of different samples, writes must be synchronized using the atomic OpenMP pragma to avoid race conditions, and atomicity involves processing overheads. In order to understand the volumes of data we are talking about, we can provide these figures: working with the 3D dataset 1 in the largest evaluation space, each bounding box contains 53,511 evaluation points (with crop-and-chop enabled); the sum of all the bounding boxes would use a total of 199.34 GB but, as they overlap, they have to be consolidated into a matrix (representing the whole evaluation space) of 1.46 GB.

Considering these facts, there are two possible ways of accelerating S-KDE: reducing memory operations

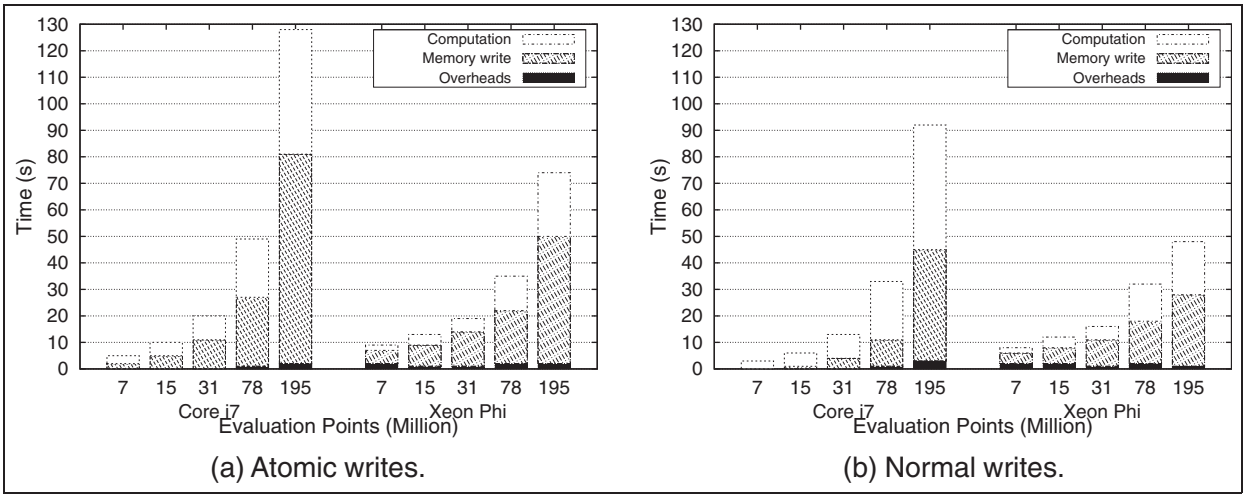


Figure 13. Dissection of execution times of S-KDE for 3D dataset 1.

and reducing inter-thread coordination when performing writes. We have already discussed the chop-and-crop technique for reducing the set of evaluation points computed per sample. Before dealing with the second, we were interested in measuring the *potential* benefits of avoiding write synchronization. To do so, we ran the same experiments removing the atomicity constraint, reporting the results in Figure 13(b). The resulting output values of those S-KDE runs are not valid, but the execution times allowed us to estimate the room for improvement. The removal of the *atomic* directive makes our code run, on average, 42.7% faster in the i7, and 14.5% faster in the Xeon Phi.

The first conclusion we can draw from these figures is that the effect of synchronized writing is more important in the i7 than in the Xeon Phi. This is due to the differences between these two architectures. The Core i7 CPU has four memory channels, with on-chip memory controllers, 256 kB of per-core L2 cache and 10 MB of shared L3 cache. In comparison, the 57 cores in the Xeon Phi are connected through a bidirectional ring, and share 28.5 MB of L2 cache. We presume that the cost of atomic writes is relatively higher in the i7 than in the Xeon Phi due to the smaller size of i7's caches, and also because memory operations are already partially serialized in the Xeon Phi by its interconnection network.

The second conclusion is that there is room for performance improvement, particularly in the multi-core CPU, if we guarantee the correct operation of memory writes while avoiding the overheads derived from the use of atomic directives. Our first idea is to coordinate the way threads process samples, in such a way that those samples being processed simultaneously have non-overlapping bounding boxes. This approach requires a previous ordering on samples in non-overlapping groups, with the risk of suffering from an ordering overhead that surpasses the achieved benefit. Another approach could be exploiting even further the chop-and-crop mechanism, making a different use of the available parallelism: samples could be processed sequentially, but per-sample cropped slices could be processed in parallel, because they never overlap. Currently, this approach is not applicable to 2D problems. Additionally, the number of slices should be large enough to efficiently use all the available processors, and distribution of slices to threads must be correctly load balanced, because they are of different sizes. We have left the implementation of these techniques as future work.

6 Conclusions and future work

Experiments with the current implementation of S-KDE have shown that, compared with available, state-of-the-art implementations of KDE, it provides, *by far*, the best performance, even when running in a modest i7 processor. This performance can be boosted if a many-core coprocessor is available. However, we do not consider S-KDE as a finished product: we have analyzed its behavior, and identified different mechanisms to accelerate it, especially for multi-core processors.

The speed achieved by S-KDE come from three sources. The main one is algorithmic: the sample-wise approach to estimate densities has lower complexity than the evaluation-point-wise approach, requiring fewer memory accesses. These accesses are further reduced by implementing the chop-and-crop technique for problems of dimensionality three and higher. Finally, S-KDE benefits from the exploitation of parallel architectures, in particular multi-core CPUs and many-core coprocessors.

Currently, S-KDE is only applicable to bounded kernels, and its scalability is limited by the available memory. Learning from R's *ks-kde*, we could include support for unbounded kernels if limits to their

influence area are imposed (effectively implementing bounding boxes); we could also work with extremely large evaluation spaces if the output of the program is not a complete matrix with all the evaluation points, but a list of user-defined evaluation points in which estimation of the density is required. Another approach to deal with large-scale problems in reduced-memory situations could be to adopt a divide-and-conquer approach, dividing the evaluation space in zones, consolidating them as a final step. These modifications, together with the exploration of mechanisms to reduce memory contention, are part of our lines of future work.

From the point of view of parallel processing, we are also working on making the multi-core and the attached coprocessor run concurrently on the same problem. Going further, several accelerators and CPUs on a single system could be used simultaneously to greatly accelerate KDE computations.

The S-KDE code is available upon request, for any researcher performing density estimation tasks.

Acknowledgements

Jose Miguel-Alonso and Alexander Mendiburu are members of the European Network of Excellence on High Performance and Embedded Architecture and Compilation (HIPEAC). The authors would like to thank Aritz Perez, Borja Calvo and Leticia Hernando for their aid in this work.

Funding

This work has been partially supported by the Saiotek and Research Groups 2013-2018 (IT-609-13) programs, funded by the Basque Government, the Ministry of Science and Technology (grant number TIN2013-41272P) and the COMBIOMED network in computational biomedicine (Carlos III Health Institute). The authors acknowledge financial funding from the MINECO, National R + D + i plan (grant number CGL2013-45198-C2-1-R). Additional funding from different calls from the University of the Basque Country (grant numbers GIU14/03 and UFI 11/55) allowed this paper to be finished. Unai Lopez-Novoa holds a grant from Basque Government (grant number BFI-2010-224).

Note

1 http://software.intel.com/intel-mkl

References

Ahamada I and Flachaire E (2010) *Non-Parametric Econometrics*. Oxford: Oxford University Press.

Andrés Ferreyra R, Podestá GP, Messina CD, Letson D, Dardanelli J, Guevara E, et al. (2001) A linked-modeling framework to estimate maize production risk associated with enso-related climate variability in Argentina. *Agricultural and Forest Meteorology* 107(3): 177–192.

Bosman PA and Thierens D (2000) Ideas based on the normal kernels probability density function. Technical Report 11, Department of Computer Science, Utrecht University, The Netherlands.

Corti S, Molteni F and Palmer T (1999) Signature of recent climate change in frequencies of natural atmospheric circulation regimes. *Nature* 398(6730): 799–802.

Duong T (2007) ks: Kernel density estimation and kernel discriminant analysis for multivariate data in R. *Journal of Statistical Software* 21(7): 1–16.

Elgammal A, Duraiswami R and Davis L (2003) Efficient kernel density estimation using the fast Gauss transform with applications to color modeling and tracking. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 25(11): 1499–1504.

Fukunaga K (1990) *Introduction to Statistical Pattern Recognition*. San Diego: Academic Press.

Givens G and Hoeting J (2005) *Computational Statistics*. New York: Wiley-Interscience.

Jeffers J and Reinders J (2013) *Intel Xeon Phi Coprocessor High Performance Programming*. San Francisco: Morgan Kaufmann.

Leyk Z and Stewart D (1994) Meschach: Matrix computation in C. *Proceedings of the CMA* 32: 1–240.

Lopez-Novoa U, Mendiburu A and Miguel-Alonso J (2015) A survey of performance modeling and simulation techniques for accelerator-based computing. *IEEE Transactions on Parallel and Distributed Systems* 26(1): 272–281.

Łukasik S (2007) Parallel computing of kernel density estimates with MPI. *7th international conference on computational science (ICCS 2007)*, Beijing, China, 27–30 May 2007, pp. 726–733. Berlin, Heidelberg: Springer.

Luo N and Qian F (2009) Evolutionary algorithm using kernel density estimation model in continuous domain. In: *7th asian control conference (ASCC 2009)*, Hong Kong, 27–29 August 2009, pp. 1526–1531. Piscataway: IEEE Press.

Michailidis PD and Margaritis KG (2013) Accelerating kernel density estimation on the GPU using the CUDA framework. *Applied Mathematical Sciences* 7(30): 1447–1476.

Millman KJ and Aivazis M (2011) Python for scientists and engineers. *Computing in Science & Engineering* 13(2): 9–12.

OpenACC (2013) The OpenACC Application Programming Interface. Technical Report 2.0a, OpenACC.org.

Perkins S, Pitman A, Holbrook N and McAneney J (2007) Evaluation of the AR4 climate models' simulated daily maximum temperature, minimum temperature, and precipitation over Australia using probability density functions. *Journal of Climate* 20(17): 4356–4376.

Racine J (2002) Parallel distributed kernel estimation. *Computational Statistics & Data Analysis* 40(2): 293–302.

Rothberg JM, Hinz W, Rearick TM, Schultz J, Mileski W, Davey M, et al. (2011) An integrated semiconductor device enabling non-optical genome sequencing. *Nature* 475(7356): 348–352.

Scott DW (2009) *Multivariate Density Estimation: Theory, Practice, and Visualization*. New York: Wiley.

Seabold S and Perktold J (2010) Statsmodels: Econometric and statistical modeling with Python. In: van der Walt S and Millman J (eds) *9th python in science conference (SciPy 2010)*, Austin, USA, 28 June–3 July 2010, pp. 57–61.

Sheather SJ (2004) Density estimation. *Statistical Science* 19(4): 588–597.

Silverman BW (1986) Density Estimation for Statistics and Data Analysis. xxx: Chapman & Hall.
Srinivasan BV, Hu Q and Duraiswami R (2010) GPUML: Graphical processors for speeding up kernel machines. In: *workshop on high performance analytics - algorithms, implementations, and applications*, Columbus, USA, 1 May 2010.
Venables WN and Ripley BD (2002) *Modern Applied Statistics with S*. New York: Springer.
Weissbach R (2006) A general kernel functional estimator with general bandwidth-strong consistency and applications. *Journal of Nonparametric Statistics* 18(1): 1–12.

Author biographies

Unai Lopez-Novoa received his MSc in Computer Science from the University of Deusto in 2010. He is currently working toward a PhD in the Department of Computer Architecture and Technology of the University of the Basque Country UPV/EHU. His main research interests include parallel and distributed computing, and especially, massively parallel computing using accelerators.

Jon Sáenz received his BSc in Physics of the Atmosphere from Complutense University, Madrid and a PhD in Physics from the University of the Basque Country UPV/EHU. His main research interests are inside the field of Atmospheric Physics, either using mesoscale atmospheric models, or analyzing climate data. Climate data derived from observations (surface, satellite..), global or regional models and reanalyses are his main source of research material. Considering the multi-dimensional characteristics of these data and the vast amounts of data involved, advanced computational methods are needed.

Alexander Mendiburu received his BSc in Computer Science and PhD from the University of the Basque Country UPV/EHU, in 1995 and 2006, respectively. He has been a Lecturer since 1999, and an Associate Professor since 2008 in the Department of Computer Architecture and Technology. His main research interests include evolutionary computation, probabilistic graphical models, and parallel computing.

Jose Miguel-Alonso received his MSc and PhD in Computer Science from the University of the Basque Country UPV/EHU, in 1989 and 1996, respectively. He is currently a full Professor in the Department of Computer Architecture and Technology. Prior to this, he was a visiting Assistant Professor at Purdue University for a year. He teaches different courses, at graduate and undergraduate levels, related to computer networking and high-performance and distributed systems, and has supervised (and currently supervises) several PhD students working on these topics.

Appendix

Detailed description of the cropping technique

This appendix complements Section 3.3, providing a detailed explanation of the cropping process. We describe here equations used to adjust an optimal bounding box to each ellipse in a slice.

From a d dimensional bounding box, a nested loop traverses every $(d-1)$ box until 3D boxes are obtained. The cropping calculations begin in 3D spaces. First, the equation that represents the shape of the kernel, i.e. the equation of a 3D ellipsoid, must be completed

$$Ax^2 + Bxy + Cy^2 + Dxz + Eyz + Fz^2 = 1 \tag{11}$$

where terms A to F are filled with values from the inverse of the covariance matrix of the dataset Σ^{-1}. At this point the bandwidth parameter must be applied as $\Sigma'^{-1} = \Sigma^{-1}h^{-2}$ to modify the kernel accordingly. Then, terms A, C and F of the equation are completed with $\Sigma'^{-1}_{1,1}$, $\Sigma'^{-1}_{2,2}$ and $\Sigma'^{-1}_{3,3}$ respectively, and terms B, D and E with $2\times\Sigma'^{-1}_{1,2}$, $2\times\Sigma'^{-1}_{1,3}$ and $2\times\Sigma'^{-1}_{2,3}$.

Once the equation of the kernel is completed, the 2D slices that form the 3D bounding box are traversed along the z -axis. Using equation (11), we give different values to z, according to the step of the evaluation grid. Each z value corresponds to a slice. Then, for a particular z value, we have an ellipse represented by this equation

$$Ax^2 + Bxy + Cy^2 + D'x + E'y + F' = 0 \tag{12}$$

where $D' = Dz$, $E' = Ez$ and $F' = Fz^2 - 1$.

From this equation the series of steps described in Section 3.3 are applied to crop the optimal bounding box to the 2D slice.

Step 1: Calculate the rotation angle of the ellipse in the slice. The rotation angle θ is calculated as

$$\theta = \frac{\arctan\left(\frac{B}{A-C}\right)}{2} \tag{13}$$

Step 2: Calculate the lengths of the principal axes of the ellipse. We first calculate the equation of the ellipse in an unrotated manner as

$$\begin{aligned} A_u &= A\cos^2\theta + B\cos\theta\sin\theta + C\sin^2\theta \\ B_u &= 0 \\ C_u &= A\sin^2\theta - B\cos\theta\sin\theta + C\cos^2\theta \\ D_u &= D'\cos\theta + E'\sin\theta \\ E_u &= -D'\sin\theta + E'\cos\theta \\ F_u &= F' \end{aligned} \tag{14}$$

where terms A, B, C, D', E' and F' are those of the original ellipse (equation (12)) and terms A_u to F_u represent

the terms of the equation of the same ellipse, but unrotated. From this new equation we can derive the lengths of the principal axes a and b of the ellipse as

$$a = \sqrt{\frac{-4F_u A_u C_u + C_u D_u^2 + A_u E_u^2}{4A_u C_u^2}}$$
$$b = \sqrt{\frac{-4F_u A_u C_u + C_u D_u^2 + A_u E_u^2}{4A_u^2 C_u}} \quad (15)$$

Step 3: Calculate the coordinates of the edge vertices. To do so, we first calculate the coordinates c_x, c_y of the center of the ellipse as

$$c_x = \frac{-D_u}{2A_u} \quad c_y = \frac{-E_u}{2C_u} \quad (16)$$

to then calculate the edge vertices of the unrotated ellipse vx_u and vy_u as

$$\begin{array}{llll} vx_{ux} & = c_x + a & vy_{ux} & = c_x \\ vx_{uy} & = c_y & vy_{uy} & = c_y + b \end{array} \quad (17)$$

Step 4: Crop the bounding box. First, we must apply the rotation to edge vertices vx_u and vy_u, to obtain the rotated edge vertices vx and vy

$$\begin{aligned} vx_x &= vx_{ux}\cos\theta + vx_{uy}\sin\theta \\ vx_y &= -vx_{ux}\sin\theta + vx_{uy}\cos\theta \, vy_x \\ &= vy_{ux}\cos\theta + vy_{uy}\sin\theta \\ vy_y &= -vy_{ux}\sin\theta + vy_{uy}\cos\theta \end{aligned} \quad (18)$$

to then find the boundaries of the ellipse, applying the Euclidean norm as

$$bound_x = \sqrt{vx_x^2 + vy_x^2} \quad bound_y = \sqrt{vx_y^2 + vy_y^2} \quad (19)$$

The last step is to calculate the coordinates of the bounding box aligned to the evaluation grid, rounding $bound_x$ and $bound_y$ to the evaluation step per dimension.

7

Kernel Density Estimation in Accelerators: Implementation and Performance Evaluation

Kernel Density Estimation in Accelerators: Implementation and Performance Evaluation

Unai Lopez-Novoa[a], Alexander Mendiburu[a], Jose Miguel-Alonso[a]

[a] *Intelligent Systems Group,*
Department of Computer Architecture and Technology,
University of the Basque Country UPV/EHU,
San Sebastian, 20018 Gipuzkoa, Spain

Abstract

Kernel Density Estimation (KDE) is a popular technique used to estimate the probability density function of a random variable. KDE is considered a fundamental data smoothing algorithm, and it is a common building block in many scientific applications. In a previous work we presented S-KDE, an efficient algorithmic approach to compute KDE that outperformed other state-of-the-art implementations, providing accurate results in much reduced execution times. Its parallel implementation targeted multi and many-core processors. In this work we present an OpenCL implementation of S-KDE, targeting modern accelerators in a portable way. We also analyze the performance of this implementation on three accelerators from different manufacturers, to find out to what extent our code exploits the performance offered by those devices.

Keywords:
Kernel Density Estimation, Performance Analysis, OpenCL, Many-core Processors, GPGPU

1. Introduction

Kernel Density Estimation (KDE) is a popular statistical technique to estimate the probability density function of a random variable with unknown characteristics [1]. It is considered a fundamental data smoothing problem in statistics and an alternative to other density estimation techniques such as the histogram, that relies on a simple binning. KDE is used in a wide variety of research areas, such as climatology for environmental model evaluation [2], computer vision for image segmentation and tracking [3] or biometry to estimate the effectiveness of a medical treatment[4].

Some of the problems that use KDE codes as building block usually require processing large datasets, which translates into long execution times. The literature shows different approaches to computing KDE. Given the complexity of the algorithm, a trade-off must be found between accuracy and execution time [5]. In a previous work, we introduced a novel algorithm to compute KDE whose complexity is lower than that of state-of-the-art KDE implementations, providing accurate results with shorter execution times. We called this algorithm S-KDE [6]. As modern

Email address: unai.lopez@ehu.es (Unai Lopez-Novoa)

scientific codes run in multi-core processors, we implemented and tested an OpenMP implementation of S-KDE that exploited the parallel capabilities of current CPUs, and many-core co-processors such as the Intel Xeon Phi [7]. The combined effect of the novel algorithmic approach and the exploitation of parallel processing resulted in impressive reductions in execution times.

We cannot ignore, though, that many of the computing platforms used nowadays, and those expected to be used in a near future, will integrate other classes of accelerator devices, not only the Xeon Phi [8]. The spectrum of devices is wide and includes platforms such as Graphics Processing Units (GPUs), FPGAs and other classes of many-cores. In order to make S-KDE available to a larger community, we decided to produce a new version of the code targeting the wider possible set of accelerators, being OpenCL the most logical choice of programming environment.

Two are the main contributions of this paper. First, the description of the porting of S-KDE to OpenCL. Redesigning a code for accelerators usually requires major changes due to the massive data parallel processing model they are aimed for [9], and not every application fits into it. Second, we evaluate the performance of our code when running it in three state-of-the-art accelerators: an AMD GPU, a NVIDIA GPU and an Intel Xeon Phi co-processor. We rely on some popular performance models and benchmark suites to characterize the devices, and provide some insights about how well our code exploits the performance achievable from each accelerator.

The remainder of this paper is organized as follows. We provide an overview of the state-of-the-art accelerator devices in Section 2, and describe briefly the fundamentals of our S-KDE approach in Section 3. We present the OpenCL implementation of KDE in Section 4, and conduct a performance analysis over it in Section 5. Finally, we summarize conclusions and future lines of work in Section 6.

2. Accelerator devices

Current supercomputers and data processing facilities are being built around hybrid compute nodes that include accelerator devices. The current landscape of devices includes reconfigurable circuits (e.g. FPGAs), discrete co-processors (e.g GPUs), hybrid chips (e.g. AMD HSA systems) or low power consumption systems (e.g. ARM or Intel Atom based systems)[10][11]. In this work we are going to focus on the most popular classes of accelerators, GPUs and Intel's Xeon Phi, due to their wide presence in HPC systems and for the extensive body of literature and ecosystem of tools around them.

GPUs are hardware devices designed to make efficient image processing. They are composed of hundreds of SIMD cores, capable of handling thousands of active threads, with lightweight context switching [9]. Since their adoption as general purpose coprocessors (coining the term GPGPU, from general-purpose processing on GPUs), they have been enhanced with features such as dedicated double-precision units or large cache hierarchies that make them ready to run efficiently a wide variety of HPC workloads. GPUs can be found as discrete coprocessors, connected to a host processor through PCI-Express, or integrated in the same die with other type of processing cores, as in the AMD APU architectures. In this work we will use two different discrete GPUs connected through PCI-Express.

The Xeon Phi is a many-core processor presented by Intel in 2012. It holds up to 61 x86 cores, 16 GB of dedicated memory and it is connected to a host system through PCI-Express. Compared

to the cores in Intel multi-core CPUs, Xeon Phi cores make in-order processing and hold several differences in the instruction set, such as using AVX-512 for vector computations. Current Xeon Phi devices present a theoretical peak performance of 1 TFLOP/s in double precision, and support a wide set of development frameworks, such as MPI, OpenMP, OpenCL or Cilk [7].

3. Kernel Density Estimation

KDE has been applied since the 80's as a density estimation technique in different environments [5]. It creates smooth density estimations, in contrast with other techniques, such as the histogram. Intuitively, given an evaluation landscape and a dataset of samples, KDE places in the landscape a "bump" around each sample, aggregating the effect of those bumps to create the estimated Probability Density Function (PDF). An example for a one-dimensional case is depicted in Figure 1, where a density estimate is created for a dataset with three samples. A kernel (a red "bump") is placed over each sample, and then the influence of all of them is summed creating the black thick line, which is the estimated PDF [1].

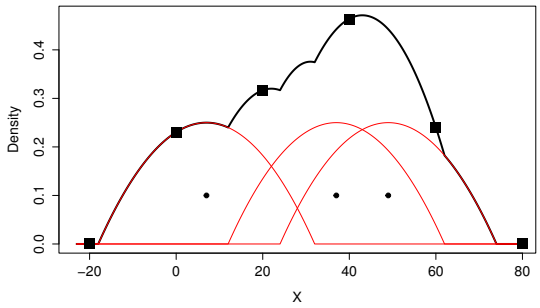


Figure 1: Example of KDE for 1D data

The resulting density estimation is continuous, but most KDE implementations provide it as a set of discrete values. The user defines the boundaries of the landscape and the separation between the points where the PDF will be evaluated (an array of per-dimension steps). Therefore, the output is actually a (discrete) evaluation grid, an array of evaluation points. In the example shown in Figure 1, the 1D evaluation space spans from -20 to 80, and the evaluation step is 20. Thus, the estimated function will be represented as a vector containing the densities in the evaluation points $-20, 0, 20, 40, 60$ and 80.

The most common way to compute KDE is to traverse every point in the evaluation grid, and compute and add, for each of them, the density influenced by each and every sample. This approach is completely parallelizable using a data parallel approach, but in many cases implies a vast number of useless computations. This is due to the fact that a sample affects only a portion of the evaluation space, a set of points around its position. The size of this influence area depends on the kernel of choice (the shape of the "bump") and other parameters. Thus, a more efficient approach is to define the influence area of a sample as a set of evaluation points, and then traverse just the evaluation points inside that area. The first approach has an $O(k_d mn)$ computational complexity, being k_d a dimensionality constant, m the number of evaluation points (the size of the evaluation grid), and n the number of samples. The second approach has an $O(k_d np)$ complexity, where k_d is a dimensionality constant, n the number of samples and p the number of evaluation points in the influence area of a sample. We must take into account that

usually p is *much smaller* than m. In this work we will implement the second approach. We call it S-KDE (from sample-wise KDE), and include some additional features to further avoid unneeded computations.

The KDE literature includes different proposals for kernel functions. Depending on the chosen one, the technique to confine the influence area of a sample will be different. In this work we use an Epanechnikov kernel and a technique based on the eigenvalues of the covariance matrix of the dataset to calculate a rectangular shaped box that delimits the influence area of a sample [12]. We will refer to this rectangle as the *bounding box* of a sample. In addition, we apply a technique called *Chop & Crop* that minimizes the size of the bounding box by removing evaluation points not belonging to the influence area of the kernel in spaces of dimensionality three or higher. It works by first reducing the d-dimensional bounding box to a set of 2D slices, and then cropping the slice to the minimum squared box. This two-step process is represented in Figures 2 and 3 respectively. The interested reader is referred to [6] for a detailed explanation of the S-KDE algorithm and its implementation for multi-core CPUs.

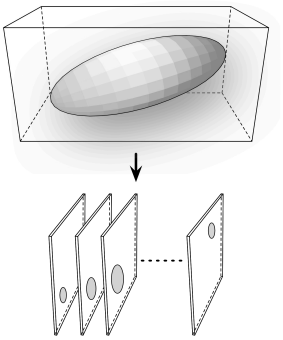


Figure 2: Chopping a 3D bounding box into 2D slices

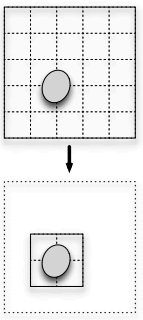


Figure 3: Cropping a 2D slice to obtain a minimum-size bounding rectangle

We can provide some example figures to illustrate the efficiency of S-KDE. We will assume a 3D dataset with 500k samples, and an evaluation space (grid) with 194.81 million evaluation points. Using the traditional KDE approach that traverses every evaluation point of the grid would lead to $9.74 * 10^{13}$ sample-evaluation point operations. In contrast, a rectangular 3D bounding box around each sample in the mentioned scenario contains on average 102461 points, and using the sample-wise KDE approach would require $5.12 * 10^{10}$ computations. On top of this, if we apply the *Chop & Crop* technique, the number of evaluation points per bounding box is reduced to 53511 on average, and the resulting total number of computations is $2.67 * 10^{10}$. Thus, S-KDE improves KDE efficiency by several orders of magnitude. We can go even further, exploiting massively parallel accelerators to run S-KDE.

4. An OpenCL KDE implementation for accelerators

In this section we describe the process followed to adapt S-KDE to accelerators. Our aim has been to create an algorithm fitting into the data-parallel computation model that most accelerators use, together with an implementation portable across the spectrum of existing hardware devices.

Currently, the only two frameworks that provide portability in accelerators are Microsoft DirectCompute and OpenCL, a standard proposed by the Khronos Group [13]. Given that the former is only available for Microsoft Windows platforms, we chose OpenCL.

4.1. OpenCL in a nutshell

An OpenCL application consists of a host part and a device part, which is called a *kernel* (not to be confused with the kernel functions used for density estimation). The host part orchestrates data movements from host to device and vice-versa, and manages the execution of kernels. The device is able to run simultaneously multiple threads or *work-items*, all of them running the same kernel code. Work-items are arranged in groups called *work-groups*, which may have a 1D, 2D or 3D structure; the developer chooses the shape and size of the work-groups.

An OpenCL *platform* is a collection of devices managed by a single host. OpenCL defines a device model in which the device is composed of a set of *processing elements*, arranged in *compute units*. At run time, the OpenCL framework assigns each work-group to a compute unit for its execution. Internally, work-items are mapped to processing elements.

Regarding memory, there is host memory and per-device global memory. The latter is accessible by all threads running in the device. Additionally, there is local memory shared by all treads in the work-group. Finally, each thread has its own, small, private memory and registers. The host code is in charge of moving data from host to device memory (and vice-versa), and threads can move data from the device's global memory to other memory zones.

All these abstractions enable the OpenCL framework to launch any data-parallel application (kernel) over any device, given a mapping between the device's characteristics and the OpenCL model, and also given that hardware requirements are fulfilled (for example, a certain amount of memory per thread is necessary, and while some devices can provide it, others could be more limited). Further information about OpenCL can be found in [14].

4.2. Implementing S-KDE with OpenCL

The starting point for the OpenCL code is the serial implementation of S-KDE. The steps that this code follows have been depicted in Figure 4. Note that steps 4 and 6 have been surrounded with parentheses, as they are operations required in the OpenCL code but not in the serial one.

Step 1 is the initialization, where the evaluation grid is set to zeros and the size of the bounding box is computed; this is a "generic" bounding box, that must be customized per sample, in order to deal with the discrete and bounded nature of the evaluation space. Then, the code traverses the samples of the dataset in an iterative way.

For each sample, the bounding box is first fitted to the grid, and then the chopping is applied (Step 2). This way, no matter the dimensionality of the problem, it is reduced to a computation of a series of bi-dimensional slices. Then, each slice of the bounding box is processed.

For each slice, cropping is applied to reduce it to its minimum size (Step 3). Once the size and coordinates of the slice are defined, its evaluation points are traversed. For each point within the slice, its distance to the sample and the density that the sample influences on the point are computed (Step 5). Finally, all partial densities affecting a point are accumulated in the corresponding position of the evaluation grid (Step 7).

The described serial algorithm presents clear opportunities for parallelization, but it also poses some challenges for its adaption to the accelerator model due to its limited data reuse and the very low compute to memory access ratio. That being said, a major rework has been done to adapt it to the OpenCL model, which has required some algorithm re-structuring as well as the inclusion of additional support operations.

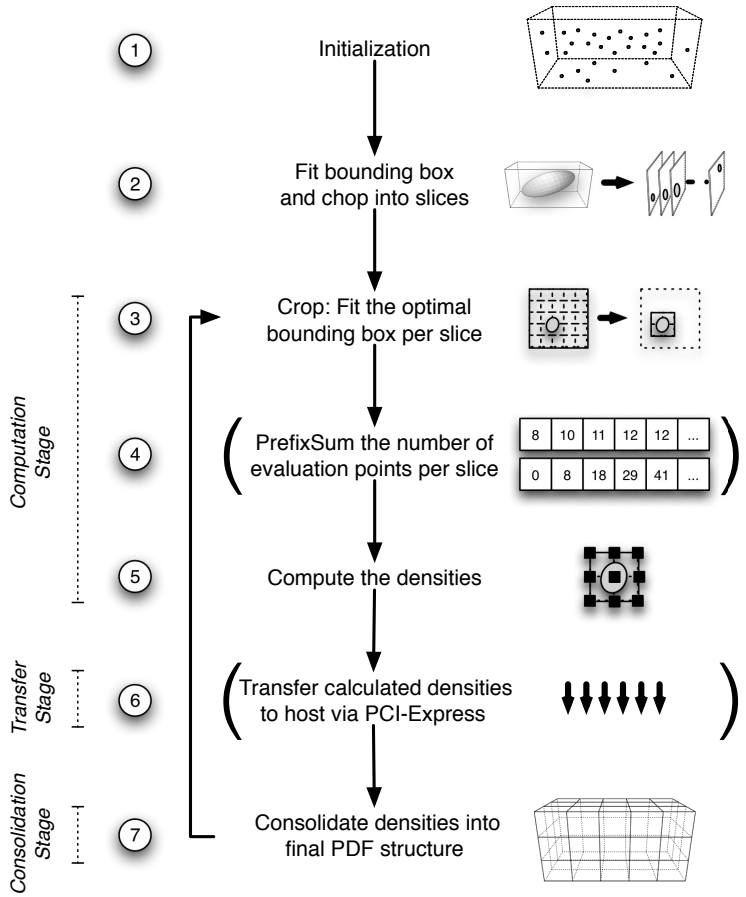


Figure 4: Workflow of KDE implementation

The main opportunity for adapting the algorithm to a data-parallel model comes from the fact that the bounding boxes around each of the samples can be processed simultaneously, without any kind of dependency. Therefore, we can have as many parallel threads as samples in the dataset. However, all these bounding boxes must be aggregated into a common landscape matrix in which the final PDF grid is computed, and as influence areas of samples (and, therefore, bounding boxes) may overlap, the accumulation step must be somehow synchronized to avoid memory write collisions. The way we have addressed this issue is explained later.

The OpenCL code has been structured as depicted in Figure 4. It includes new steps, as well as modifications to some of those described for the serial code.

1. *Initialization:* In a first step, the entire sample dataset is copied into the accelerator, along with the required support structures. We assume that all the dataset fits in the memory of the accelerator (note that this is not the evaluation grid). In this step we also compute the size of the generic bounding box. This is host code (executed in the CPU).
2. *Box fit and Chop:* For every sample, its bounding box is fitted to the grid and chopping is

applied. At this point, the problem has been reduced to a collection of 2D slices. This is implemented as an OpenCL kernel (executed in the accelerator in a data parallel way).

3. *Crop:* Cropping is applied to reduce the number of evaluation points in each slice. Additional information about each slice is computed, such as its coordinates in the evaluation space and the number of evaluation points it contains. This is an OpenCL kernel.
4. *PrefixSum:* A PrefixSum is applied to the vector that contains the number of evaluation points per slice. This is a support computation required by the next kernel for its threads to make ordered stores. We have used the OpenCL implementation of PrefixSum available in the SHOC benchmark suite [15].
5. *Density Computation:* Each thread calculates the influence created by a sample on an evaluation point of a given slice. The resulting densities are stored in an auxiliary vector and not consolidated into the global PDF structure. This is an OpenCL kernel.
6. *Densities Transfer:* The resulting vector of partial densities is transferred through PCI-Express from the accelerator to host memory. This is managed by the host.
7. *Consolidation:* The host reads the vector of partial densities and accumulates them into the evaluation space. This is host code.

As explained before, a critical step of S-KDE is the consolidation of partial densities into the global landscape (Step 7). If done in parallel without the proper synchronization mechanisms, results can be invalid, because the influence areas of samples overlap and, thus, threads may incur in memory write collisions. To avoid this issue, the current OpenCL implementation of S-KDE leaves this task to be performed by the host CPU in a serial way. This is pragmatic because the host has the whole output structure in main memory, and the serialized access guarantees the absence of memory write collisions.

We tried alternative approaches to run the consolidation phase in the accelerator, but they required either some sorting of partial results (an expensive operation that resulted in even longer execution times) or the use of atomic adds (an operation not supported in OpenCL for double precision floats [13], although some devices have specific extensions for it). Therefore, we have kept the consolidation part in the CPU.

As a side note, the presented workflow is intended for KDE problems of dimensionality three or higher. However, our implementation targets as well two dimensional spaces using the same workflow but without applying the Chop & Crop technique – because it is not needed. In this case, the workflow is exactly the same but without the second and third steps.

In terms of the data structures used in the OpenCL code, it should be clear that the threads running in the accelerator have access to the sample dataset and to auxiliary structures containing partial, non-consolidated densities. The final density matrix is managed exclusively by the CPU. The main program is iterative after the initialization step. Each iteration consists of processing a "chunk" of the problem, which is nothing more than a subset of the samples. This is done in a data-parallel fashion, using a chain of OpenCL kernels. The per-chunk intermediate results are stored in the accelerator and, later, transferred to the CPU for consolidation.

The iterative nature of the code has two main advantages. The first one is the ability to deal with accelerators with different memory sizes, that in many cases are not capable of holding the complete output matrix. This makes our code limited by the RAM managed by the CPU, but not by the device's memory, provided that the selected chunk size uses intermediate data structures that fit into the device. The second advantage is that it opens the possibility of working in a pipelined fashion: while the device computes a chunk, the CPU can be consolidating the results of the previous one. We will explore this possibility later in Section 5.3.

An important parameter of our program is the chunk size, or number of samples to process in each iteration of the algorithm. This size must be defined in such a way that the intermediate results from an iteration fit into a data object in the memory of the device, before being transfered to the CPU. Therefore, the chunk size has to consider characteristics of the device (maximum allocatable size) and size (number of points) of each bounding box around a sample. Note that the maximum allocatable size can be smaller than the device's global RAM, and that some space may be already allocated to other required data structures. The chunk size is rounded down to the closest power of two, to better match the work-group sizes managed by the devices.

For example, in a 3D dataset with 500k samples and an evaluation space with 194 million points, each per-sample bounding box could have up to 106609 points or 832.8 kB; this is problem-dependent, and computed at the initialization phase. In a device with 256 MB of maximum object size, our heuristic would assign a chunk size of 256 samples.

5. Performance analysis

This section presents a performance analysis of the KDE implementation described above on three different accelerators. The code was designed for portability, so that it can run, unmodified, in any modern co-processor supporting OpenCL. We will first present the characteristics of the platforms and datasets used in the experiments, and then carry out a performance analysis in top-down manner, i.e. getting first global performance measures and, afterwards, digging into details.

5.1. Settings used in the experiments

Our experiments have been conducted with three accelerator devices: an AMD Radeon HD 6950 GPU, a NVIDIA GTX 650 GPU and an Intel Xeon Phi 3120A Coprocessor, whose main features are summarized in Table 1. In addition, our code has been limited to the API features of OpenCL v1.1. Even though some of these devices support OpenCL 1.2 or 2.0, version 1.1 is the most supported one in currently available processors, including the Xeon Phi, GPUs, FPGAs and ARM-based systems.

	AMD Radeon HD 6950	**NVIDIA GTX 650**	**Intel Xeon Phi 3120A**
Architecture	Cayman	Kepler	MIC
Cores	1408	384	57
Core Clock	800 Mhz	1.05 Ghz	1.1 Ghz
Memory	2 GB GDDR5	1 GB GDDR5	6 GB GDDR5
DP Performance[1]	563 GFLOP/s	67 GFLOP/s	1 TFLOP/s
Max. Allocatable Size	445.5 MB	255.8 MB	1435.2 MB
Host CPU	Intel Core i5-2400S	Intel Core i7-3820	Intel Core i7-3820
OpenCL SDK	AMD APP v2.9.1	CUDA v6.0.37	Intel OpenCL v3.2.1

Table 1: Hardware features of the accelerators used in the experiments

As a reference, we also run in some experiments the serial implementation S-KDE algorithm (including Chop & Crop) compiled with gcc v4.7.2 and executed in a Intel Core i7-3820 (3.60 Ghz clock frequency).

[1] Theoretical peak performance in double precision, as declared by the manufacturer

As explained in Section 3, the complexity (or problem size) of a KDE execution depends mainly on the number of samples in the dataset and on the size of the evaluation space (determined by its boundaries and step size). The latter will also determine the size of the bounding box that limits the influence area of a kernel. In this work we have performed several tests varying the size of the evaluation space, for two different 3D datasets. We have fixed the boundaries of the evaluation space and modified the step size to increase/decrease the number of evaluation points, see Table 2. The datasets have been created synthetically, sampling a multivariate normal distribution. The first one contains 500k samples, and the second one contains 1M samples.

Finally, there is a parameter in every KDE computation that must be taken into account, and that has not been mentioned in Section 3: the bandwidth or smoothing parameter. This value modifies the smoothness and size of the kernel and, therefore, the number of evaluation points inside a bounding box. An exploration for the choice of the right bandwidth value is out of the scope of this work, and we have selected it using the heuristics detailed in [1].

Dim X	**Dim Y**	**Dim Z**	**Total**
110	220	322	7792400
110	440	322	15584800
220	440	322	31169600
220	440	805	77924000
220	1100	805	194810000

Table 2: Size of the different evaluation spaces (number of evaluation points in the grid) used

5.2. Initial assessment

To get an initial assessment of the performance of our OpenCL S-KDE, we compare its total execution time against that of the serial program, for the three target devices. Results are shown in Figures 5 and 6 for the dataset of 500k and 1M samples respectively.

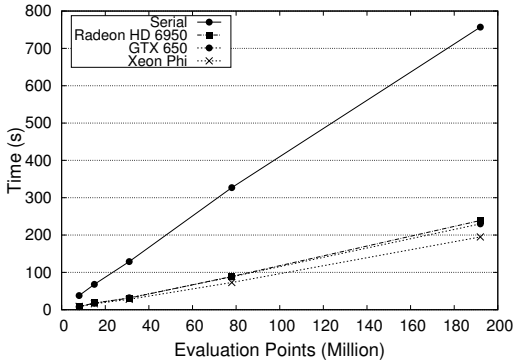


Figure 5: Comparison of execution times for dataset 500k

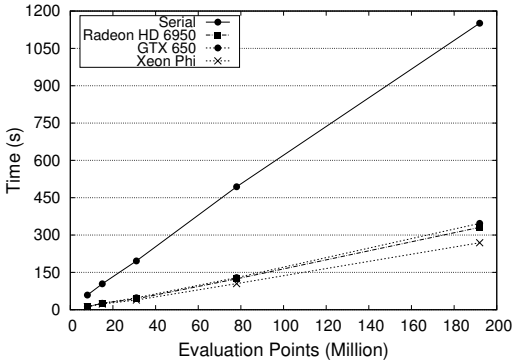


Figure 6: Comparison of execution times for dataset 1M

The first conclusion is that the OpenCL code runs significantly faster than the serial code. Speed-ups obtained with the largest problems are 3.47x, 3.31x and 4.27x for the Radeon, the GTX and the Phi respectively. These results, being good, are not as impressive as those reported for other HPC applications implemented in accelerators [16]. We can be (partially) satisfied

because we make use of the extra muscle provided by the accelerator, but we also want to explore if we could do better.

After this black-box assessment, we try to understand more in detail the limits and bottlenecks of the code when running in the accelerated platforms. In Figures 7 and 8 we show the accumulated time spent in each of the three stages depicted in Figure 4: Computation (steps 3, 4 and 5, executed in the accelerator), Transfer (step 6, moving data from the accelerator to the CPU) and Consolidation (step 7, executed in the CPU). It is to be highlighted that the most expensive stage in all cases is the non-accelerated part of the code: the Consolidation of partial results in the global density matrix, carried out by the CPU to avoid memory write collisions.

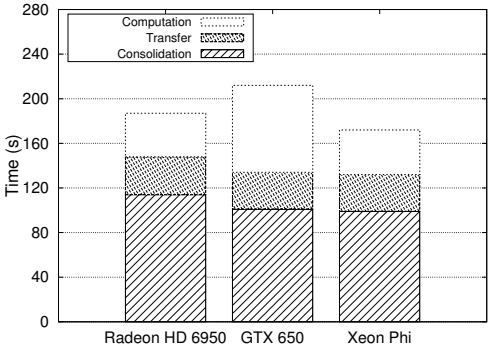


Figure 7: Dissected execution time of OpenCL S-KDE. 500k dataset and 194M evaluation points

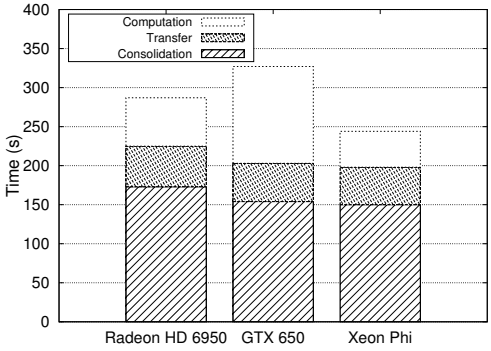


Figure 8: Dissected execution time of OpenCL S-KDE. 1M dataset and 194M evaluation points

If we focus on each stage separately, we can observe the effects in the execution time of the different elements participating in the computation:

- The time required for the Computation stage shows how the Intel Xeon Phi co-processor is the fastest of the tested accelerators, while the NVIDIA GTX is the slowest.
- The time used for accelerator-to-CPU transfers is approximately the same in the three tested platforms. This is to be expected, as all of them use the same PCI-Express interconnect.
- For the Consolidation stage, the i7 used in the Phi and GTX platforms is slightly faster than the i5 used in the Radeon platform.

Therefore, the good comparative results of the Phi platform comes from a combination of a fast CPU and a fast co-processor.

The main conclusions obtained from these experiments is that we have been partially successful with our OpenCL implementation of S-KDE: the program is faster than the serial version, but not as fast as we would like. We dig further inside the different parts of the code in order to understand what is limiting its performance.

5.2.1. Analyzing compute efficiency

In this subsection we focus on the Computation stage: we want to understand how the OpenCL kernels use the capabilities of the accelerators, and to discover if we are exploiting

them in an effective way. To do so, we have relied on the popular *roofline model*, presented by Williams et al. [17] in 2008. It provides a way to visually describe the features of a machine and a program. In particular, it is a diagram with two axes: the *Operational Intensity* of an application (FLOP/Byte) in the X-Axis and the *Attainable GFLOP/s* in the Y-Axis. An application will be positioned somewhere in the X-Axis depending on its operational intensity, and the maximum attainable performance will be given by the roof of the machine.

Even though the roofline model was originally presented for multi-core processors, several papers have extended it to characterize accelerators and massively parallel applications. In [18] GPURoofline is proposed, an adaption of the roofline model for NVIDIA and AMD GPUs that takes into consideration GPU-specific features. In [19] Kim et al. use the roofline model to explore the scalability of a code for electromagnetic field simulations in a NVIDIA GPU. In [20] Cramer et al. use the roofline to characterize the Intel Xeon Phi. In [21] Wang et al. use the roofline model to characterize a GPU and an Intel Xeon Phi, and the scalability of an OpenACC code in them. However, none of these works presents a generic way to build a roofline model for accelerator devices. This is what we do in this section using a method applicable to any OpenCL-capable device.

	DP Performance (GFLOP/s)	Off-chip Bandwidth (GB/s)
Radeon HD 6950	556.02	130.09
GTX 650	36.25	66.39
Xeon Phi 3120A	964.48	94.36

Table 3: OpenCL Benchmarking Results

To characterize the device we need to measure the maximum attainable performance in terms of GFLOP/s and the maximum bandwidth to the off-chip memory in GB/s. These values can be retrieved using a benchmark suite that stresses the devices and provides the effective peak values, which are usually lower than the theoretical ones advertised by the manufacturer. In particular, we use the tests from the ClPeak benchmark suite[2]. Relevant results for the three target accelerators have been listed in Table 3. The roofline plot is computed as *Min(Bandwidth * Operational Intensity, GFLOP/s)*.

The characterization of the application is done kernel by kernel. Each kernel is parsed using the LLVM compiler [22] to generate its intermediate representation, which we parse to count the number of floating point operations and memory accesses.

Figures 9, 10 and 11 show the roofline plots for the AMD Radeon, NVIDIA GTX and Xeon Phi respectively. Each figure shows the roofline of the machine as a line, the position of the Density Computation kernel (square tick), and the position of the whole Computation stage (without data transfer, cross tick). We can see how those ticks overlap, because the accelerators use most of their time to compute densities. Other kernels are not depicted in the graph because they are fast and, besides, some of them (i.e., PrefixSum) operate with integer type values and, thus, their operational intensity is zero.

We can see how the NVIDIA GTX accelerator is way below the others in terms of raw performance. We can also observe how our application has a very low operational intensity and, therefore, it is far from reaching the top performance in all cases. This is particularly harmful in powerful and expensive accelerators, such as the Xeon Phi. The low compute-to-memory ratio

[2]https://github.com/krrishnarraj/clpeak

and low data reuse makes S-KDE a memory bound algorithm – but it would be even worse using the classic, evaluation point-wise approach, or without implementing Chop & Crop.

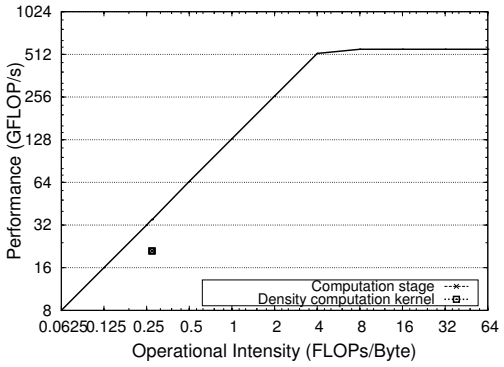


Figure 9: Roofline of Radeon HD 6950

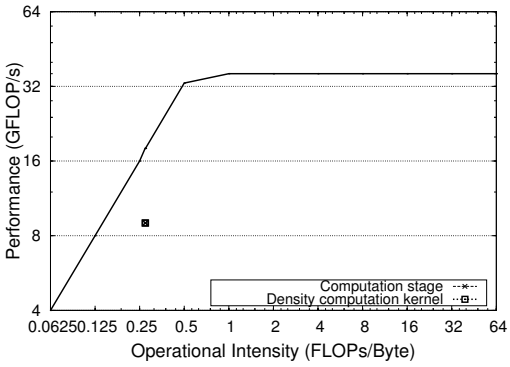


Figure 10: Roofline of GTX 650

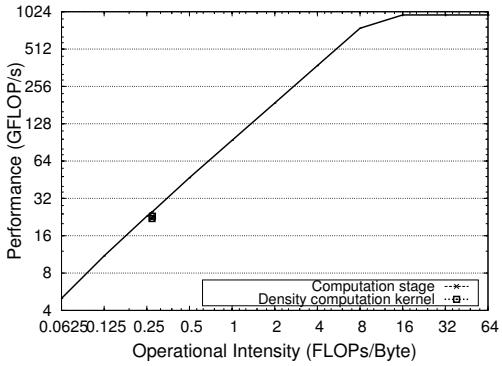


Figure 11: Roofline of Intel Xeon Phi 3120A

In order to better exploit the accelerators, modifications in the applications would be required to increase its operational intensity: running more (double precision) floating point operations per moved data item. Given the current S-KDE algorithm, we have not discovered any way of doing this.

In addition, we must highlight the choice of an OpenCL parameter that affects performance: the work-group size. As explained in Section 4.1, OpenCL assigns work-groups to Compute Units (CUs) for their execution. Assuming that hardware constraints are fulfilled, an excessively large work-group size might leave CUs unused in the device, and an excessively small one might cause stalls in some architectures such as GPUs. In the design of the OpenCL KDE code we made an exploration on the work-group size to find the one that minimized the execution times. We found 128 threads per work-group to be the best size for the three tested devices. This size has given a good CU occupancy / load balancing tradeoff in all tested cases. We refer the interested reader to [14][23][24] for more information on this topic.

5.2.2. Analyzing PCI-Express efficiency

Let us analyze now how the OpenCL S-KDE code makes use of PCI-Express. The data transfer stage is mandatory when using discrete accelerators and, depending on its use, it can turn into a bottleneck. As we did in the previous section, we first characterize the hardware and then the way our application exploits it. For the former, we used the *BusSpeedReadback* benchmark from the SHOC suite [15], which measures the attainable GB/s when reading from the accelerator, for different block sizes. Then, we measured the GB/s achieved by our application, using the block size (size of the intermediate data structures) determined by the chosen chunk size. Results are depicted in Figures 12, 13 and 14 for AMD Radeon, NVIDIA GTX and Intel Xeon Phi respectively. Note how the square tick is not in the same position in the three graphs, because of the different chunk sizes used.

Regarding the hardware side, we see how maximum efficiency of PCI-Express is only achieved when moving large data blocks. However, the curves for each device are different, due to differences in the OpenCL run-times and in the PCI-Express management routines in each platform. The chunk sizes used in our programs allow the transfers to be in the most efficient regions.

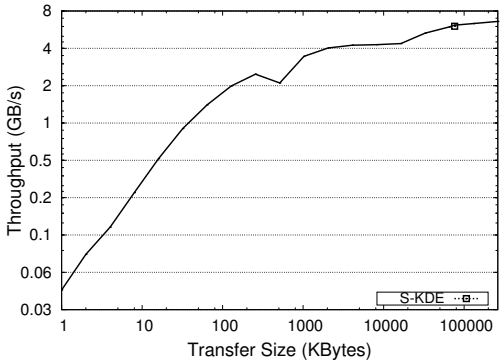


Figure 12: PCI-Express Throughput Test in Radeon 6950

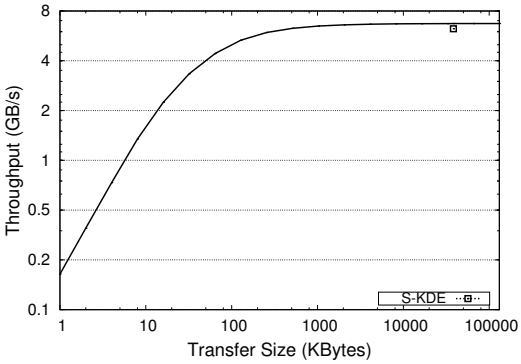


Figure 13: PCI-Express Throughput Test in GTX 650

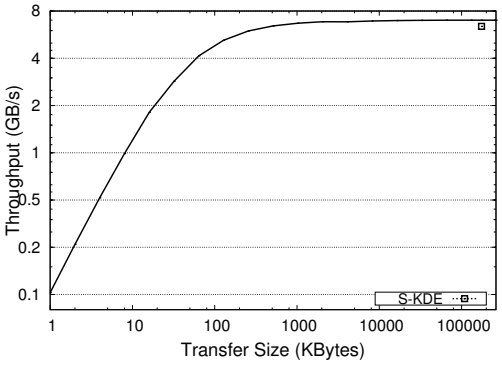


Figure 14: PCI-Express Throughput Test in Xeon Phi 3120A

5.3. Overlapping stages

As described in Section 4, the iterative behavior of our code makes it suitable for working in a pipelined mode, where operations over different chunks of data are overlapped. This mode of operation allows the simultaneous use of CPU and accelerator, thus further accelerating program execution.

We have implemented a two-stage pipeline as depicted in Figure 15: the computation of the partial densities corresponding to a chunk in the accelerator, followed by the transfer of the partial results through PCI-Express, is overlapped with the consolidation in CPU of the results from a previous chunk. This pipeline configuration has been motivated by the execution times of the different stages shown in Section 4. We have implemented it using the Pthreads API, launching a thread for the OpenCL-related operations, and a separate one for the Consolidation.

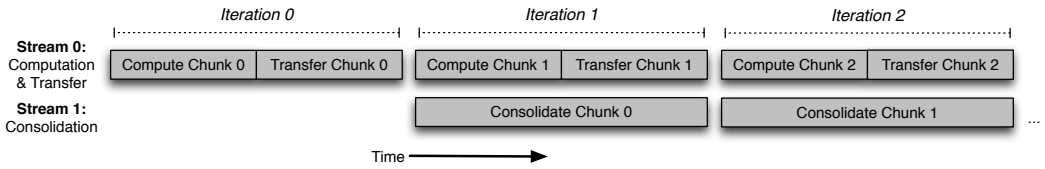


Figure 15: Pipelined execution of the OpenCL implementation of S-KDE

We illustrate the efficiency of the pipelined program with the execution times for the dataset with 1 million samples, for three different sizes of the evaluation space. Results are shown in Table 4. Each column shows the accumulated execution times in seconds for each of the stages, and the total accumulated execution time (excluding, for the sake of clarity, initialization and finalization). Improvements over the non-pipelined operations are important, for the three devices.

It is to be remarked that running two operations simultaneously cause, in some cases, extra delays. For example, in the three devices, the Consolidation costs are higher in the pipelined program than in the non-pipelined version. The same thing happens with the Transfer stage, but only in the Radeon GPU. The Computation kernels executed in the accelerators require the same time for both modes of operation. These overheads seem to be caused by the high pressure on the memory bus, as both the Transfer and the Consolidation are memory intensive operations. We tried leaving the Transfer stage out of the pipeline, overlapping in each iteration just Computation and Consolidation (doing the Transfer immediately afterwards) and then per-stage execution times where the same obtained without the pipeline. However, this last approach resulted in worse performance results, and we left the numbers out of the tables.

In a further step, we implemented a three-stage pipeline, where all the stages are overlapped. However, the global performance results where similar to the ones given by the two-stage pipeline. This was to be expected as, in the tested platforms, the Consolidation stage takes longer than the summed execution times of the Computation and the Transfer stages. This approach would be beneficial if the relative durations of the different stages was different, for example if Consolidation times were shorter.

The use of the presented two-stage pipeline improves the execution time differently in each device. Interestingly, the GTX GPU is the one offering a more efficient simultaneous operation of CPU and accelerator (1.81 performance gain), while the Xeon Phi and the Radeon are not that efficient (1.36 and 1.40 respectively). Compared to the serial version, total speed-up values (for the largest problem size) are now improved to 4.42x, 5.74x and 5.67x for the Radeon, the GTX

Ev. Space Size	AMD Radeon HD 6950				NVIDIA GTX 650				Intel Xeon Phi 3120A			
	Comp.	Transfer	Cons.	Total	Comp.	Transfer	Cons.	Total	Comp.	Transfer	Cons.	Total
No Pipeline												
7M	2.49	2.41	5.17	10.06	5.71	2.34	3.66	11.71	2.44	2.24	3.43	8.11
31M	9.13	8.48	22.24	39.86	19.38	8.19	17.61	45.19	8.94	7.96	17.00	33.90
195M	62.38	52.37	173.29	288.05	124.99	49.73	154.85	329.58	46.05	48.61	150.34	245.00
2 Stage Pipeline												
7M	2.47	3.85	6.46	6.78	5.68	2.33	3.46	8.05	2.40	2.27	4.59	4.80
31M	9.13	13.50	27.04	27.09	19.34	8.19	17.14	27.57	9.02	8.07	21.64	21.75
195M	63.72	75.04	204.69	204.77	125.88	50.90	181.62	182.05	47.10	48.81	178.71	178.86

Table 4: Runtimes (s) of different stages for the accelerators. for the problem with 1M samples. for different sizes of the evaluation spaces

and the Xeon Phi respectively. Note how an efficient pipelined operation results in the GTX being the best performed, when in theory this is the least powerful accelerator.

5.4. Discussion

We can summarize the analysis stating that we have reached acceptable levels of efficiency given the memory-bound nature of the S-KDE algorithm, that severely limits the attainable performance. The resulting OpenCL code can be considered useful (it offers 4.42x-5.74x speedup using the pipelined operation), but it does not make a good use of current accelerators: operational intensity is too low, and PCI-Express data transfer costs are high. An additional factor that prevents a really fast S-KDE code is the Consolidation stage: additional recoding efforts are necessary to improve this CPU-side code.

One of our main objectives when writing this program was portability in terms of both code and performance. We achieved it, without spending excessive time carrying out per-device optimizations. The performance analysis carried out has also been done with device-independent tools. More detailed information could have been obtained using tools (profilers) provided by the device manufacturers; for example CodeXL[3], Visual Profiler[4] or VTune[5] for AMD, NVIDIA and Intel platforms respectively. These tools are essential when coarse-grain optimizations are not applicable, or to fine-tune for a specific device.

From the previous analysis we can also draw some conclusions about the cost of accelerating S-KDE. We have not included in the previous tables the prices of the tested devices, because they change constantly, but they currently are around $100 for AMD and NVIDIA GPUs (these are consumer-grade devices, now discontinued) and $1700 for the Intel Xeon Phi (a server-grade device designed for a different, smaller market). In theory, the high price tag of the Xeon Phi should be balanced with the ease with which its peak performance can be reached. GPUs are more difficult to exploit, if the application does not show some specific characteristics (high data-parallelism, high operational intensity, low memory contention, low use of the PCI-Express, and so on). For S-KDE, which is not particularly well suited for accelerators using the OpenCL programming model, we can see that the performance of the cheapest GPU is as good as that of the most expensive accelerator.

6. Conclusions

In this work we have presented briefly S-KDE, an efficient algorithm for kernel density estimation, together with its OpenCL implementation targeting modern accelerators. This work complements [6], which discussed the implementation of S-KDE on multi and many-core devices. We have tested the code in three accelerators: AMD Radeon HD 6950 (GPU), NVIDIA GTX 650 (GPU) and Intel Xeon Phi 3120A (many-core).

The S-KDE code does not match particularly well with the OpenCL programming paradigm. It carries out simple operations over massive volumes of data, and it presents memory write contention issues that makes it difficult to delegate important parts of the code (the Consolidation stage) to the accelerator. Despite this, we have achieved significant acceleration (5x) in platforms

[3] http://developer.amd.com/tools-and-sdks/opencl-zone/codexl/

[4] http://developer.nvidia.com/nvidia-visual-profiler

[5] http://software.intel.com/en-us/intel-vtune-amplifier-xe

with very modest GPUs (around $100). We have been unable, though, to exploit efficiently the capabilities of more expensive accelerators, such as the Xeon Phi.

We have analyzed thoroughly the characteristics of our code when running on the target hardware, understanding its limits. The major issues with the current program are (1) the need of transferring data through PCI-Express, derived from the use of a discrete accelerator; (2) the low computational intensity of the kernels running in the accelerators, that do not exploit efficiently the floating point capabilities of those devices; and (3) the need to run at the CPU, to avoid memory write issues, a costly Consolidation step. Both the code and the tools used to understand its behavior are device and application independent.

Our future work will address two main issues. The most urgent one is to improve the Consolidation stage, which is currently the one determining the global execution time. This is not a trivial task, because we must deal with synchronization issues. Then, we would like to extend the work related in this paper to convert it into an simple but powerful analysis methodology, to be used with any OpenCL code, but device independent. The literature of code optimization for accelerators includes excellent device-specific resources (such as the good practices manuals [25][26] shipped by the manufacturers) and also application-specific works (e.g. [27] for cryptographic primitives or [28] for mathematical functions), but device and application-independent tools are still missing.

References

[1] B. W. Silverman, Density estimation for statistics and data analysis, Vol. 26, Chapman & Hall/CRC, 1986.

[2] U. Lopez-Novoa, J. Sáenz, A. Mendiburu, J. Miguel-Alonso, I. Errasti, G. Esnaola, A. Ezcurra, G. Ibarra-Berastegi, Multi-objective environmental model evaluation by means of multidimensional kernel density estimators: Efficient and multi-core implementations, Environmental Modelling & Software 63 (0) (2015) 123 – 136.

[3] A. Elgammal, R. Duraiswami, L. Davis, Efficient kernel density estimation using the fast gauss transform with applications to color modeling and tracking, IEEE Transactions on Pattern Analysis and Machine Intelligence 25 (11) (2003) 1499–1504.

[4] R. Weissbach, A general kernel functional estimator with general bandwidth-strong consistency and applications, Journal of Nonparametric Statistics 18 (1) (2006) 1–12.

[5] S. J. Sheather, Density estimation, Statistical Science (2004) 588–597.

[6] U. Lopez-Novoa, J. Sáenz, A. Mendiburu, J. Miguel-Alonso, An efficient implementation of kernel density estimation for multi-core and many-core architectures, International Journal of High Performance Computing Applications˙ doi:10.1177/1094342015576813.

[7] J. Jeffers, J. Reinders, Intel Xeon Phi Coprocessor High Performance Programming, 1st Edition, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2013.

[8] J. Nickolls, W. Dally, The gpu computing era, Micro, IEEE 30 (2) (2010) 56–69.

[9] D. B. Kirk, W.-m. W. Hwu, Programming Massively Parallel Processors: A Hands-on Approach, 1st Edition, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2010.

[10] Top500 supercomputing sites, `http://www.top500.org` (May 2015).

[11] U. Lopez-Novoa, A. Mendiburu, J. Miguel-Alonso, A survey of performance modeling and simulation techniques for accelerator-based computing, IEEE Transactions on Parallel and Distributed Systems 26 (1) (2015) 272–281.

[12] K. Fukunaga, Introduction to statistical pattern recognition (2nd ed.), Academic Press Professional, Inc., San Diego, CA, USA, 1990.

[13] K. O. W. Group, The opencl specification, A. Munshi, Ed.

[14] A. Munshi, B. Gaster, T. G. Mattson, J. Fung, D. Ginsburg, OpenCL Programming Guide, 1st Edition, Addison-Wesley Professional, 2011.

[15] A. Danalis, G. Marin, C. McCurdy, J. S. Meredith, P. C. Roth, K. Spafford, V. Tipparaju, J. S. Vetter, The scalable heterogeneous computing (shoc) benchmark suite, in: Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units, GPGPU '10, ACM, New York, NY, USA, 2010, pp. 63–74.

[16] V. W. Lee, C. Kim, J. Chhugani, M. Deisher, D. Kim, A. D. Nguyen, N. Satish, M. Smelyanskiy, S. Chennupaty, P. Hammarlund, R. Singhal, P. Dubey, Debunking the 100x gpu vs. cpu myth: An evaluation of throughput computing on cpu and gpu, SIGARCH Comput. Archit. News 38 (3) (2010) 451–460.

[17] S. Williams, A. Waterman, D. Patterson, Roofline: An insightful visual performance model for multicore architectures, Commun. ACM 52 (4) (2009) 65–76.
[18] H. Jia, Y. Zhang, G. Long, J. Xu, S. Yan, Y. Li, Gpuroofline: A model for guiding performance optimizations on gpus, in: C. Kaklamanis, T. Papatheodorou, P. Spirakis (Eds.), Euro-Par 2012 Parallel Processing, Vol. 7484 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2012, pp. 920–932.
[19] K.-H. Kim, K. Kim, Q.-H. Park, Performance analysis and optimization of three-dimensional FDTD on GPU using roofline model, Computer Physics Communications 182 (6) (2011) 1201 – 1207.
[20] T. Cramer, D. Schmidl, M. Klemm, D. an Mey, Openmp programming on intel xeon phi coprocessors: An early performance comparison, in: Many-core Applications Research Community Symposium, 2012.
[21] Y. Wang, Q. Qin, S. C. W. SEE, J. Lin, Performance portability evaluation for openacc on intel knights corner and nvidia kepler, in: HPC China 2013, 2013.
[22] C. Lattner, V. Adve, Llvm: a compilation framework for lifelong program analysis transformation, in: Code Generation and Optimization, 2004. CGO 2004. International Symposium on, 2004, pp. 75–86.
[23] Y. Torres, A. Gonzalez-Escribano, D. R. Llanos, ubench: exposing the impact of cuda block geometry in terms of performance, The Journal of Supercomputing (2013) 1–14.
[24] S. Seo, J. Lee, G. Jo, J. Lee, Automatic opencl work-group size selection for multicore cpus, in: Parallel Architectures and Compilation Techniques (PACT), 2013 22nd International Conference on, 2013, pp. 387–397.
[25] AMD, App opencl programming guide, `http://developer.amd.com/tools/hc/AMDAPPSDK/assets/AMD_Accelerated_Parallel_Processing_OpenCL_Programming_Guide.pdf` (Apr. 2013).
[26] NVIDIA, Opencl best practices guide, `www.nvidia.com/content/cudazone/CUDABrowser/downloads/papers/NVIDIA_OpenCL_BestPracticesGuide.pdf` (Jan. 2012).
[27] G. Agosta, A. Barenghi, A. Di Federico, G. Pelosi, Opencl performance portability for general-purpose computation on graphics processor units: an exploration on cryptographic primitives, Concurrency and Computation: Practice and Experience.
[28] S. Pennycook, S. Hammond, S. Wright, J. Herdman, I. Miller, S. Jarvis, An investigation of the performance portability of opencl, Journal of Parallel and Distributed Computing 73 (11) (2013) 1439 – 1450, novel architectures for high-performance computing.

8

Multi-objective Environmental Model Evaluation by Means of Multidimensional Kernel Density Estimators: Efficient and Multi-core Implementations

Environmental Modelling & Software 63 (2015) 123–136

Contents lists available at ScienceDirect

Environmental Modelling & Software

journal homepage: www.elsevier.com/locate/envsoft

Multi-objective environmental model evaluation by means of multidimensional kernel density estimators: Efficient and multi-core implementations

Unai Lopez-Novoa [a, *], Jon Sáenz [b, c], Alexander Mendiburu [a], Jose Miguel-Alonso [a], Iñigo Errasti [d], Ganix Esnaola [b], Agustín Ezcurra [b], Gabriel Ibarra-Berastegi [d]

[a] Intelligent Systems Group, Dept. of Computer Architecture and Technology, University of the Basque Country (UPV/EHU), P. Manuel Lardizabal 1, 20018, Donostia-San Sebastian, Spain
[b] Dept. Applied Physics II, University of the Basque Country (UPV/EHU), Sarriena Auzoa z/g, 48940, Leioa, Spain
[c] Plentzia Marine Station (PIE-UPV/EHU), Areatza Pasealekua, 48620, Plentzia, Spain
[d] Dept. Nuclear Engineering and Fluid Mechanics, University of the Basque Country (UPV/EHU), Alda. Urkijo, s/n, 48013, Bilbao, Spain

ARTICLE INFO

Article history:
Received 7 February 2014
Received in revised form
12 September 2014
Accepted 25 September 2014
Available online 25 October 2014

Keywords:
Multivariate kernel density estimation
Multidimensional kernel density estimation
Multi-core implementation
Environmental model evaluation

ABSTRACT

We propose an extension to multiple dimensions of the univariate index of agreement between Probability Density Functions (PDFs) used in climate studies. We also provide a set of high-performance programs targeted both to single and multi-core processors. They compute multivariate PDFs by means of kernels, the optimal bandwidth using smoothed bootstrap and the index of agreement between multidimensional PDFs. Their use is illustrated with two case-studies. The first one assesses the ability of seven global climate models to reproduce the seasonal cycle of zonally averaged temperature. The second case study analyzes the ability of an oceanic reanalysis to reproduce global Sea Surface Temperature and Sea Surface Height. Results show that the proposed methodology is robust to variations in the optimal bandwidth used. The technique is able to process multivariate datasets corresponding to different physical dimensions. The methodology is very sensitive to the existence of a bias in the model with respect to observations.

Software availability

Name or software or dataset: density-parallel
Developers: Unai Lopez-Novoa (1) Jon Sáenz (2) Alexander Mendiburu (1) Jose Miguel-Alonso (1). (1) Intelligent Systems Group, Dept. of Computer Architecture and Technology, University of the Basque Country (UPV/EHU), P. Manuel Lardizabal 1, 20018, Donostia-San Sebastian, Spain. unai.lopez@ehu.es, Phone: +34 943 018 012, alexander.mendiburu@ehu.es, Phone: +34 943 015 020 and j.miguel@ehu.es Phone: +34 943 018 019. (2) EOLO Group, Department of Applied Physics II, University of the Basque Country, (UPV/EHU), Barrio Sarriena s/n, 48940-Leioa, Spain. jon.saenz@ehu.es, Phone: +34 946 012 445.
Year first available: 2014
Hardware required: any single-core or multi-core processor.
Software required: (1) C Compiler (e.g. GCC). OpenMP compiling capabilities required for multi-threaded implementations. (2) MESCHACH math library. Available at http://homepage.math.uiowa.edu/~dstewart/meschach/. (3) netCDF data handling library. Available at http://www.unidata.ucar.edu/software/netcdf/
Program language: ANSI-C
Program size, including example data: 102 KB in a single .tar.gz file.
License: Open software, available under a "New BSD 3-clause license".
Availability: www.sc.ehu.es/ccwbayes/kde

1. Introduction

Climate models are the best tools that scientists currently have in order to assess the impact of increasing concentrations of greenhouse gases and other anthropogenic influences on the observed climate of the Earth. These tools allow scientists to understand climatic changes from a dynamical point of view and to give quantitative answers to questions about future climate. They

* Corresponding author. Tel.: +34 943 018 012.
E-mail address: unai.lopez@ehu.es (U. Lopez-Novoa).

http://dx.doi.org/10.1016/j.envsoft.2014.09.019
1364-8152/

make feasible the assessment of the characteristics (deterministic versus stochastic) of some climatic variations at different temporal or spatial scales. Finally, they are fundamental in the attribution phase of the study of the climate change problem, since they allow to confidently discard competing hypothesis such as whether the climate change is rooted in natural or anthropogenic causes (Bengtsson, 2013; Knutti, 2008).

Climate models must be evaluated against different observations (Otto et al., 2013) or paleoclimate data (Braconnot et al., 2012; Hind et al., 2012; Moberg, 2013; Sundberg et al., 2012) in order to get a quantitative indication on the confidence that we can put into their outputs depending on the efficacy of the models to reliably represent the climatic processes and feedbacks. Contrary to operational weather forecast models, there is no way to properly evaluate models against future climate, since future climate does not exist yet (Randall et al., 2007; Stocker et al., 2013). Additionally, since parameterizations of sub-grid scale processes are not fully independent from current climate, it is clear that evaluation against current climate is the only feasible, albeit not perfect, solution in terms of evaluating the projections for future climate (Errasti et al., 2011, 2013; Radić and Clarke, 2011; Reichler and Kim, 2008).

The rationale behind this hypothesis is that models that are able to better simulate current climate are the ones that we expect will also be the best ones in terms of the simulation of future climate. It is well known that this is not necessarily true due to the different behavior of models in terms of their internal feedback mechanisms (Andrews et al., 2012; Dessler, 2013). These feedbacks lead to differing values of the climate sensitivity of models and, hence, future warming is dependent on these different sensitivities, leading to the question whether all the models are equally valid (Knutti, 2010).

Particularly for downscaling applications and regional impact analysis, an evaluation of the adequacy of models is a common step (Brands et al., 2011; Radić and Clarke, 2011; Walsh et al., 2008). Considering that numerical downscaling is computationally expensive, it cannot be performed over all the models available from a big experiment such as CMIP3 or CMIP5, and it is usually only performed on a subset of the models (Hewitt and Griggs, 2004). Then, the best models are only used for downscaling in regional climate assessment, since they have been proven to be the most suitable over a particular region. This strategy of selecting the best subset of all the available models has been contested by some studies (Reifen and Toumi, 2009) and defended by others (Macadam et al., 2010), since this result depends on the removal of the seasonal cycle and the use of anomalies instead of raw output from the models.

There are several inter-comparison experiments that have been set-up in order to drive the models with common boundary conditions so that results between model runs can be compared. As an example of this kind of standard experimental setups, that in several cases have their origins in the nineties, we can cite the Atmospheric Model Intercomparison Project (Gates et al., 1999), the Project for Intercomparison of Land Surface Parameterization Schemes (PILPS) (Henderson-Sellers et al., 1995) or the Palaeoclimate Modelling Intercomparison Project (PMIP) (Kageyama et al., 1999), the Atmospheric Chemistry and Climate Model Intercomparison Project (ACCMIP), described by Lamarque et al. (2013), or the one that is most relevant for our study, since we use data from this experiment, the Coupled Model Intercomparison Project (CMIP) (Meehl et al., 2007; Taylor et al., 2012). These projects have covered several phases through the years and for the case of the CMIP data, CMIP5 can already be used. In general, models grouped under a similar experimental set-up such as CMIP3 or CMIP5 are considered as an ensemble of opportunity (Annan and Hargreaves, 2010). There are some limitations because even for coordinated experiments such as CMIP3, there is some freedom in the way the external boundary conditions are applied (they are not 100% equal for all the models), see Table 1 in Wang et al. (2007). The number of realizations from every model in the ensemble of opportunity is not the same, neither and, therefore, the influence of each model in the behavior of the ensemble is not the same. Additionally, models are less independent than they should be, since critical algorithms or components are shared by several models (Fernández et al., 2009; Knutti et al., 2013; Masson and Knutti, 2011; Pennell and Reichler, 2011).

In terms of evaluation of climate models, it is well known that climate simulations are run, most of the times, past the limit of deterministic predictability associated with predictability of the first kind, according to Lorenz's classification. Climate models simulate climate change under varying boundary conditions in terms of the Probability Density Functions (PDF) of climatic variables. The varying boundary conditions consist of external forcings such as the variability in solar irradiance, orbital parameters or anthropogenic greenhouse gas emissions, amongst other potential driving factors (Bengtsson, 2013). Some aspects of climate simulations are deterministic, such as the seasonal cycle at extratropical latitudes (Errasti et al., 2013). On the other side, some properties of the atmospheric circulation such as the blocking at extratropical latitudes can not be precisely forecast with lead times corresponding to several days without the use of ensemble forecast systems (Marshall et al., 2013) because of the sensitivity to initial conditions (Frederiksen et al., 2004) and the model formulation (Pelly and Hoskins, 2003) too. Therefore, climate model evaluations that are run past the limit of deterministic predictability should not a priori expect a close consistency between weather-linked variations of global or regional temperature or precipitation between models and observations, except at the longer time-scales responding to external forcings (Gleckler et al., 2008; Santer et al., 2011). These differences reflect the well-known difference of the sensitivity of the results to errors in the initial conditions (predictability of the first kind) or to errors in the evolving boundary conditions (predictability of the second kind) (Chu, 1999; Lorenz, 2006).

There is not a universally accepted strategy for climate model evaluation, since it is well known that climate model evaluation and corresponding skill scores fundamentally depend on the target area, variable or intended application of the model evaluation study (Knutti, 2008). Some studies make a focus on the deterministic parts of the model simulations (basically, the seasonal cycle) (Boer and Lambert, 2001; Taylor, 2001). Other studies (oriented to the study of droughts or floods) tend to focus on extreme percentiles, since they are much more meaningful indicators of climate change (DeAngelis et al., 2013).

This study by DeAngelis et al. (2013) is currently important for us, because it shows that models sometimes produce accurate values for the average of some climatic variable due to error compensation effects. They can, for instance, underestimate the frequency of high precipitation events and overestimate the

Table 1
Indices of distributional agreements for points derived from known gaussians using univariate scores for *X* and *Y* variables or two-dimensional scores.

		Univariate score		2D score
		G1-*X*	G1-*Y*	G1
G2	*X*	0.998		0.999
	Y		0.999	
G3	*X*	0.998		0.463
	Y		0.997	

frequency of low precipitation events. This points to the need to evaluate additional characteristics of climate model simulations beyond the mean value and standard deviation. Consequently, a few years ago, an index computed from the whole PDF of climatic variables was developed (Maxino et al., 2008; Perkins et al., 2007). It compares two PDFs and computes the minimum value of both PDFs at every abscissa. The area below this minimum represents the area below both PDFs. As such, for a perfect model, its value would be one, if both PDFs matched perfectly. This PDF-index or index of distributional agreement is the one that we will generalize to the multidimensional case in this contribution. The PDF-index analyses the correspondence of the whole PDF both from a model and observations (Maxino et al., 2008; Perkins et al., 2007). The one-dimensional PDF-index has very often been used in the literature through the last years (Brands et al., 2012; Errasti et al., 2011, 2013; Fu et al., 2013; Maxino et al., 2008; Perkins et al., 2007; Schwalm et al., 2013; Ylhäisi and Räisänen, 2013 to name a few). In this contribution we propose its extension to multiple dimensions, thus allowing to compare several features of climate or environmental models at a single step. The use of the PDF-index shows some advantages with respect to other approaches, in the sense that it samples the full PDF of the climatic variables. Therefore, the PDF-index is a very good index for the overall evaluation of the agreement between climate models and observed climate. However, there are other shortcomings, such as the fact that the analysis in terms of PDFs does not consider the time sequence of events, and the number of frost days or the number of continuous days without precipitation are important in terms of impacts (Brands et al., 2012). The PDF-index gives less weight to the tails of the distribution and, it is thus not adequate as the single index for the analysis of extremes (Brands et al., 2012).

In the case of the papers mentioned previously, the PDF-index is computed by means of unidimensional PDFs. However, in several cases, studies using the PDF-index or other scores (Dessai et al., 2005; Reichler and Kim, 2008) evaluate the skill of climate models according to several variables that may be of interest for the impact community. The most obvious instances might be precipitation and temperature, but if the scientists are interested in downscaling strategies, other variables such as geopotential height or sea level pressure appear very often (Brands et al., 2011; Errasti et al., 2011, 2013; Fu et al., 2013; Maxino et al., 2008; Radić and Clarke, 2011). In these previous references, the skill of the models is computed on a per-variable basis by means of univariate diagnostics. Their final skill score is computed by aggregating individual per-variable evaluations either by simple averaging or ranking of skill scores. However, there is currently a lack of universally accepted way of performing this combination of scores for different variables and the methodology that we propose in this contribution is aimed to fill this void, since a single index of distributional agreement is returned from the multidimensional PDF.

The main objective of this contribution is, therefore, to develop a methodology that can be applied to get a multidimensional score that allows to evaluate in a single step different variables from climate simulations against observations. In order to explain the advantages derived from using a multidimensional approach, we show a simple example derived from a synthetic dataset. We have created three synthetic datasets, G1, G2 and G3, derived from two-dimensional gaussian distributions. For each case, the gaussians are centered $\mu_i = 0$ but the corresponding covariance matrices used to create them are given by $S_1 = S_2 = \begin{pmatrix} 1 & 0.75 \\ 0.75 & 1 \end{pmatrix}$ for G1 and G2, whilst for G3, the third gaussian, the covariance matrix is given by $S_3 = \begin{pmatrix} 1 & -0.75 \\ -0.75 & 1 \end{pmatrix}$. It can be seen that, despite the difference in the structure of the distributions of points (Fig. 1, left), the univariate PDFs and corresponding indices show a good agreement (Fig. 1, middle and right and Table 1), even though the distributions are different. This is quite an artificial example, but it illustrates the point that some parts of the PDFs close to the diagonals can be projected onto similar areas over the axis when using unidimensional indexes of agreement, masking the differences between the PDFs of the model and the observations. Therefore, it is interesting to analyze the full structure of the multidimensional PDF, since it yields a realistic difference between the score corresponding to G1 versus G2 (good agreement) and G1 versus G3 (bad agreement).

To fill in the gap illustrated by the example, we generalize the PDF-index by Perkins et al. (2007) to n-dimensional phase spaces with the final aim of allowing an easy multi-criteria evaluation of models. That way, a single PDF-based index can group the performance of the models according to the multidimensional phase-space spanned by all the variables chosen for the evaluation of the model. In order to make it easier for other researchers to use this methodology, we present an implementation of this multidimensional extension by means of a set of tools that can properly address the computational problems that appear when making kernel-based estimations of PDFs with massive datasets. In the case studies that we show in this paper, we compute several realizations of the PDF-index using one to four dimensional phase spaces with up to 13,000 points for every particular model/realization. This is very intensive computationally, and for this reason we feel that an efficient implementation of the estimation of multidimensional PDFs could be of great help for researchers in this area. Thus, we have developed a general-purpose tool to compute kernel-based multidimensional PDF estimations that runs on state-of-the-art

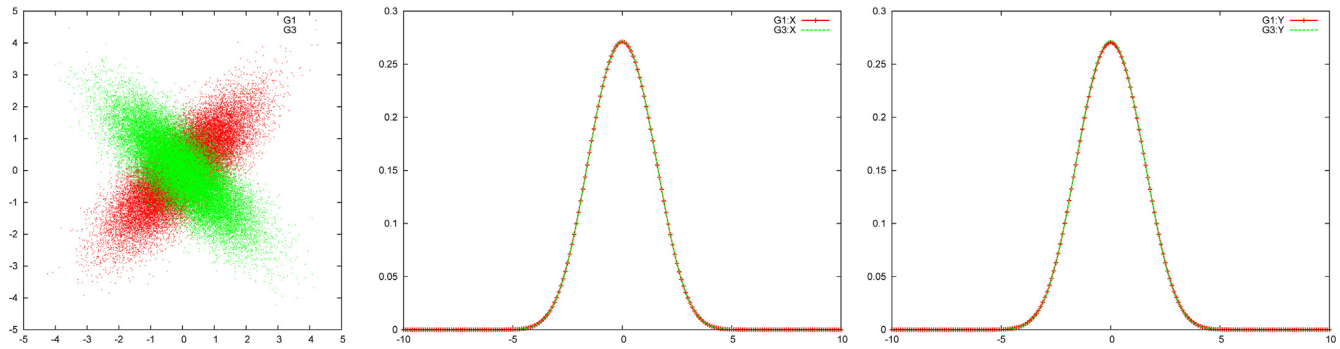


Fig. 1. Points created from G1 (red) and G3 (green) distributions (left), univariate probability distributions corresponding to the X variables (middle) from G1 (red) and G3 (green) and univariate probability distributions corresponding to the Y variable from G1 (red) and G3 (green). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

multi-core processors. Our proposal has two main characteristics: (1) a fine-tuned algorithm to calculate the PDF that minimizes the number of computations and (2) a parallel implementation of this algorithm that allows it to efficiently run in multi-core processors.

The method is applied to two different case-studies. The first case study corresponds to a realistic application of climate model evaluation (Errasti et al., 2013). Their results are re-analyzed using this new methodology. Additionally, a sensitivity of the results to the selection of the bandwidth parameter is carried out. Finally, the robustness of the results provided by the method to the existence of biases in the models is also studied. The second case study corresponds to the analysis of the performance of a coupled atmosphere-ocean reanalysis in reproducing the global scale Sea Surface Temperature and Sea Surface Height and it corresponds to a higher-dimensional problem. This second case study will be used to stress two of the merits attributed to the proposed methodology. On the one hand, this example in the context of the physical oceanography will demonstrate the wide range of the applicability of the methodology. On the other, the combination of variables with different physical dimensions will illustrate the ability of the method to process multivariate data and its ability to be applied to a large family of environmental models.

The remaining of this paper is structured as follows. Section 2 presents the materials and methods used in the paper. Results are shown in Section 3. The discussion is presented in Section 4, and the paper finishes with conclusions in Section 5.

2. Material and methods

2.1. Data representing the daily seasonal cycle of zonally averaged temperature from global climate models and reanalysis

For the first case study shown in this paper, we select a reduced dimensionality representation of the daily seasonal cycle of temperature that we already analyzed in Errasti et al. (2013). We analyze temperature of the air at the surface (TAS) daily data from seven models (20C3M simulations) of the CMIP3 experiment (Meehl et al., 2007) that were used by the Fourth Assessment Report of the IPCC (Randall et al., 2007). The models used are the BCCR-BCM2.0, GFDL-CM2.0, GFDL-CM2.1, MIROC3.2-HR, MIROC3.2-MR, MPI-ECHAM5 and MRI-CGCM2.3, and the basis for the selection of this subset of models and their characteristics can be found in Errasti et al. (2013). The same procedure is used for TAS data from ERA40 (Uppala et al., 2005) and NCEP/NCAR Reanalysis 1 (Kalnay et al., 1996), referred to as NCEP onwards. The TAS data from models and reanalyses were re-gridded to the same $2.5^\circ \times 2.5^\circ$ grid by means of bilinear interpolation, since this was the coarsest grid used by any of the reanalysis used (the one used by NCEP). To reduce the dimensionality of such a gridded dataset, the TAS data were zonally averaged and projected onto Legendre polynomials that constitute an adequate basis over the sphere. Those have very often been used as a basis in one-dimensional energy balance models (North et al., 1981) and are the basis used for the meridional components of spherical harmonics used in spectral decompositions of the equations of motion (Washington and Parkinson, 2005). The Legendre polynomials are orthonormal in the set of functions over the continuous interval $[-1, 1]$, but their corresponding discrete-grid counterparts are not orthogonal. For this reason, we applied the Gram-Schmidt orthogonalization procedure to obtain the leading discrete orthogonal $\tilde{P}_0(\mu)$, $\tilde{P}_1(\mu)$ and $\tilde{P}_2(\mu)$ Legendre polynomials. In the previous equations, $\mu = \sin(\theta)$ refers to the sine of latitude. The zonally averaged TAS profiles have been projected onto the orthogonal discrete $\tilde{P}_0(\mu)$, $\tilde{P}_1(\mu)$ and $\tilde{P}_2(\mu)$ Legendre polynomials and this has provided us with the corresponding time-varying coefficients $c_0(t)$, $c_1(t)$ and $c_2(t)$. Due to the meridional shape of the orthogonal discrete polynomials shown in Fig. 1 in Errasti et al. (2013), the physical meaning of the temporal expansion coefficients can be easily understood. The coefficient $c_0(t)$ describes the seasonal evolution of global-mean temperature linked to the different distances from the earth to the Sun corresponding to the apogee or the perigee positions, $c_1(t)$ describes the seasonal evolution of summer-winter from one Hemisphere to the other and $c_2(t)$ describes the TAS differences between the Equator and the poles. These $c_0(t)$, $c_1(t)$ and $c_2(t)$ coefficients represent the daily TAS seasonal cycle with a root-mean square error of the daily values ranging from 6.3 K (GFDL-CM2.0) to 8 K (MIROC3.2-HR) for the whole dataset. They represent 5%, 76% and 3.72% of the total variance of the zonally averaged TAS data for the case of ERA40 and similar values for the climate models and NCEP reanalysis, as extensively discussed by Errasti et al. (2013).

The use of two different reanalysis (ERA40 and NCEP) allows us to quantify the sensitivity of the results to the uncertainty of the temperature field. We consider that this uncertainty is given by the different values that we get from two reanalysis. Two different state-of-the-art evaluations of the TAS fields by two different reanalysis systems provide realistic representations of the surface temperature field, but the differences between them reflect the uncertainty in our knowledge of the detailed characteristics of the problem at hand. Both the 20C3M simulations and the reanalysis data cover the period 1961–1998 on a daily basis. The idea of using these data is to be able to compare the results of our multidimensional PDF index with already published results (Errasti et al., 2013) that used alternative techniques in terms of the skill score of the models. In Errasti et al. (2013) the PDF indices computed were one-dimensional and some diagnostics also involved the root mean square error. In this paper, we revisit the topic from a multidimensional point of view. For every i-th model, we have a three-dimensional vector of points $c_i(t) = (c_{0i}(t), c_{1i}(t), c_{2i}(t))$ that describes a trajectory in the phase space of the truncated daily zonally averaged air temperature at the surface. These data provide us with an interesting case study of the application of the new algorithm to a previous problem of evaluation of models where the multidimensional evaluation tool was not applied.

2.2. Multivariate oceanographic data

Two datasets containing joint values for the Sea Surface Temperature (SST) and Sea Surface Height (SSH, relative to the geoid) are also analyzed as a second case study. These datasets are used in the second part of the Results section to illustrate the potential application of the proposed methodology in a multivariate application that not only combines variables with different physical dimensions (e.g. SST and SSH), but also different variables resulting from different truncations of the same physical field with the same physical dimensions (e.g. different projections of the SST field). Additionally, the interest of this second case study is that it also shows the potential application of the methodology in the case of other environmental applications not restricted to climatology.

Gridded global coverage data for the SST and SSH variables over the 1993–2012 period are used. The two datasets include: a reprocessed Level 4 (gridded, gap-free) product based on the merging by means of optimal interpolation of satellite and in-situ data called ARMOR-3D (Guinehut et al., 2004, 2012) and the coupled atmosphere-ocean CFSR reanalysis (Saha et al., 2010). CFSR data covers the 1993–2010 part of the period considered here. For the 2011–2012 period CFSR is completed with data from the CFSv2 (Saha et al., 2014) analog product. In the following the CFSR and CFSv2 product will be referred simply as CFSR.

To allow inter-comparison of the datasets they are averaged to weekly data and re-gridded to a common $0.5^\circ \times 0.5^\circ$ resolution grid with an identical land-sea mask (following the minimal weekly time-frequency of the ARMOR-3D dataset and the minimal $0.5^\circ \times 0.5^\circ$ spatial resolution of the CFSR dataset). Like in the case of the TAS, it is necessary to reduce the dimensionality of the SST and SSH variables in the datasets because a global dataset at a $0.5^\circ \times 0.5^\circ$ horizontal resolution yields more than 2.0×10^5 grid-points, i.e. variables in the phase-space. This makes impossible to apply the proposed methodology to the raw data. In the case of the first case study (TAS) the dimensionality reduction is conducted by means of Legendre polynomials of zonally averaged values. In this case, the truncation is conducted by means of a Principal Component Analysis (PCA) (von Storch and Zwiers, 1999; Wilks, 2006) that extracts the Empirical Orthogonal Functions (EOFs) and the Principal Components (PCs) from the gridded anomalies (obtained after subtracting the time-mean). The gridded anomalies have been weighted to take into account the reduction of the grid-area with increasing latitude of a regular latitude–longitude grid. Therefore, the values at each grid point have been multiplied by the square root of the cosine of latitude (von Storch and Zwiers, 1999; Wilks, 2006). During this truncation process, orthonormal EOFs are used and, therefore, the units and the variances are retained by the new variables or PCs. For the sake of simplicity, the first two PCs will be retained for the SST (T1, T2) and SSH (H1, H2), making a total of 4 variables when all of them are combined. The percentage of total variances accounted for by those PCs are 83% (SST) and 19% (SSH) for the ARMOUR-3D dataset and 85% (SST) and 34% (SSH) in the case of CFSR.

2.3. Evaluation of the multidimensional PDFs, optimal bandwidth and evaluation of the multidimensional indices of PDF agreement

The methodology suggested in this paper follows three steps that we describe in this order:

1. Compute a PDF by means of kernel estimates for the multidimensional case.
2. Identify the optimal bandwidth to be used by the estimation of the multidimensional PDF.
3. Find the amount of space common to the multidimensional PDFs of every model and observations.

From the point of view of the application, the first step would be to compute the optimal bandwidth, next, compute the multidimensional PDFs and, finally, compute the scores of the models' and observations' multidimensional PDFs. However, from the point of view of the description of the algorithm it is better described using the previous order because, in order to fully understand the way the optimal bandwidth is computed, the way the PDF is computed must be considered at the beginning.

A different program has been implemented for each step, which will be described next. Finally, note that the main contribution of this paper is not the use of these steps, but in the extension to multiple dimensions of the PDF-score. This extension required completely new programs, or important modifications to previously available codes. To the best of our knowledge, no other tool set is available to provide equivalent functionality with similar computational performance.

2.3.1. Multidimensional kernel density estimation

The first program (*mpdfestimator*) performs an estimation of the multidimensional PDF in a multidimensional space. It takes as input a file containing all the observed points and optionally, a bandwidth value and the parameters defining the evaluation space (minimum, maximum and increments for every dimension). If the bandwidth value is not provided, the default corresponding to a multidimensional gaussian distribution with the same sample size is applied (Silverman, 1986).

```
a) Evaluation point-based
      estimation (EPB)
for each EvaluationPoint x {
      den_x = 0

      for each ObservedPoint x_i {
            den_x += density(x,x_i)
      }
}
```

```
b) Observed point-based
      estimation (OPB)
for each EvaluationPoint x {
      den_x = 0
}
compute Observation Influence Area A
for each ObservedPoint x_i {
      for each EvaluationPoint x in A {
            den_x += density(x,x_i)
      }
}
```

Listing 1. Approaches to computing PDF estimation expressed as pseudo-code.

Program *mpdfestimator* computes the PDF as: $\widehat{f}(\boldsymbol{x},h)=\frac{1}{nh^d}\sum K\left(\frac{\boldsymbol{x}-\boldsymbol{x_i}}{h}\right)$ where n is the number of observations, K is the kernel function, d is the number of dimensions of the dataset, h the bandwidth value, $\boldsymbol{x_i}$ is a vector containing each observed point, and $\boldsymbol{x}$ is a point of the space where the PDF is being evaluated (evaluation point onwards). The PDF is estimated in a user-defined space called *evaluation space*.

According to Silverman (1986), asymptotically, there are no differences between the different kernels at hand (Gaussian, triangular, Epanechnikov, etc.). Moreover, he states that it is desirable to base the choice of the kernel on other considerations, such as the degree of differentiability required or the computational effort involved. Other references also support the fact that the sensitivity of the results to the kernel chosen is small (Ahamada and Flachaire, 2010) and that the Epanechnikov kernel is very efficient (Scott, 1992). In our program, since the Epanechnikov kernel is bounded, we will take advantage of this boundedness to design a computationally efficient proposal.

The Epanechnikov kernel function is defined as: $K(\Delta\boldsymbol{x})=\frac{(d+2)}{2c_d}(1-\Delta\boldsymbol{x}^T\cdot\Delta\boldsymbol{x})$ where c_d represents the volume of the d-dimensional sphere of unit radius and $\Delta\boldsymbol{x}=\boldsymbol{x}-\boldsymbol{x_i}$ is the vectorial difference between an observed $\boldsymbol{x_i}$ and an evaluation $\boldsymbol{x}$ point. The use of a simple euclidean norm would be misleading in case the variables used to define the phase space were characterized by very different variances. In order to avoid this, the algorithm computes the spherically symmetric principal components $\boldsymbol{y_i}$ that can be derived from the $\boldsymbol{x_i}$ observed points. Note that the data are also initially centered to be able to compute principal components. There is no filtering of the main principal components in this step. If a reduction of dimensionality is required by the user, it must be performed by the user before calling this program. We, therefore, assume that the covariance matrix of the input data is of full rank. The fact that we use spherically symmetric principal components means that variables expressing more variance of the input data are given more importance in the computation of the kernel and, as such, the same bandwidth (h) can be used for all the dimensions. This is equivalent to using a Fukunaga-like estimator (Silverman, 1986). The same linear mapping that is used onto the observed $\boldsymbol{x_i}$ points to compute the corresponding spherically symmetric principal components $\boldsymbol{y_i}$ is applied to the centered evaluation points $\boldsymbol{x}$ yielding a set of scaled evaluation points $\boldsymbol{y}$. The difference vector and the kernel function are computed by using $\Delta\boldsymbol{y}=\boldsymbol{y}-\boldsymbol{y_i}$. Therefore, the PDF is actually defined in a set of coordinates $\boldsymbol{y}$ and, after the PDF has been evaluated, it is transformed back to the original evaluation grid $\boldsymbol{x}$. This means that it is divided by the Jacobian of the transformation (square root of the determinant of the covariance matrix) from the original evaluation space $\boldsymbol{x}$ to the new evaluation space $\boldsymbol{y}$ in order to recover the proper normalization of the PDF (Menke and Menke, 2012). If the data have been centered when the PDF was being computed, the axes defining the PDF are again translated at this point according to the original mean that has been computed before storing the results in the output netCDF files. At this point, it is convenient to stress that the resulting netCDF files are stored around the original average of the input dataset using the physical variables corresponding to the original phase space. As will be explained below, the internal use of principal components will also be a critical part in the development of the bootstrap for the multidimensional case. Additionally, it has to be mentioned that in multivariate cases, such as the comparison between SST (K) and SSH (m) that we present in the second case study, the metric that we use to evaluate the kernel using radially symmetric principal components allows us to use a non dimensional h, thus properly combining multivariate datasets. Thus, on the following pages, when h is mentioned, it refers to a non dimensional parameter.

A common approach to compute the estimate of the PDF, valid for any kind of kernel, is to make a loop that traverses all the evaluation points of the space, and calculates the kernel function for each < observation point, evaluation point > pair. This approach, evaluation point-based (EPB onwards), is shown in Listing 1.a. Its complexity is $O(k_d{\cdot}m{\cdot}n)$, being k_d a constant related to the dimensionality of the dataset, m the number of evaluation points and n the number of observed points.

In some cases, the kernel that defines the density of each sample is bounded and it affects only a small subset of the evaluation points in the evaluation space. In these cases, using the EPB approach is inefficient, as most of the evaluation points lie outside the area of influence of the kernel (but a calculation is required), being thus the corresponding density zero. Thus, as we have chosen a bounded kernel (the Epanechnikov kernel), we have defined a way to estimate the PDF that reduces the computational complexity of the previous approach, aiming to minimize the execution time of *mpdfestimator*.

Our approach, observation point-based (OPB onwards), performs a loop traversing the observation points, computing for each, the density over the evaluation points affected by the influence area of the kernel. This method requires to identify the set of evaluation points inside that influence area, which can be done by means of geometrical equations. As a visual example, we depict in Fig. 2 the influence area of a kernel as a grey ellipse. In our program, we calculate a square shaped *bounding box* around each observation point, which includes some evaluation points outside the area of influence, but it is easier to process by the program.

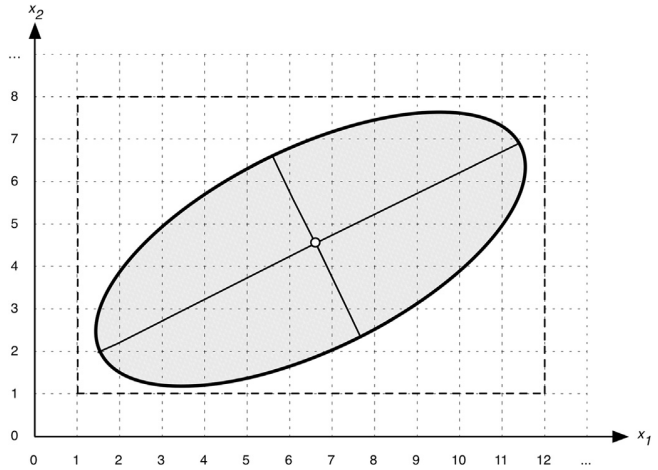


Fig. 2. Example of the influence area around an observed point in a 2D space, identifying all the grid points affected. The ellipse represents the actual influence area in the physical space, while the dashed rectangle shows the prismatic bounding box around it.

A simplified OPB is described in Listing 1.b and its computational complexity is $O(k_d \cdot m \cdot a)$, being k_d a constant related to the dimensionality of the dataset, m the number of observed points, and a the number of evaluation points inside a bounding box (generally much smaller than n).

mpdfestimator (as well as the remainder programs) has been implemented using the C programming language. All the linear algebra routines used are implemented through calls to the publicly available MESCHACH library (Stewart and Leyk, 1994). The PDF is written to a netCDF file in the physical evaluation space $\boldsymbol{x}$ requested by the user with the PDF centered and scaled according to the original data. This means that the user is free to evaluate PDFs from models and observations that are (are not) centered at the same n-dimensional average points (biased or unbiased datasets). We recommend, as shown below, that datasets are always bias-corrected before performing this analysis, but the system does not enforce the user to do so, althought it warns the user about this condition.

In the first case study used in this paper the phase-space is tridimensional, but the implemented program allows the user to go up to any dimension (starting from one), as shown by the second case study used in this paper. The one-dimensional case is also covered by our program even though there are other implementations that can cover the one dimensional case. The novelty of our contribution lays on the generalization of the one-dimensional score to higher dimensions.

In addition, standard OpenMP programming directives (Dagum and Menon, 1998) have been also included with the aim of exploiting the multi-core capability of present computers. The set of observed points will be equally distributed amongst the processors for their computation in parallel. This way, the workload is split among the available processors, reducing the execution time (almost) linearly to the number of cores used.

Following Faraway and Jhun (1990), we use a smoothed bootstrap procedure to estimate the squared error between two estimates of the PDF. The smoothed estimate starts from a reference evaluation of the PDF $\widehat{f}(\boldsymbol{x}, h_0)$ computed using a reference bandwidth h_0. Then, several estimations $\widehat{f_n}(\boldsymbol{x}, h)$ of the PDF are performed at varying values of the bandwidth parameter h and a number of $n = 1 \ldots N$ realizations for every h. The bootstrap program checks the error between the "reference" PDF used in the smoothed bootstrap procedure and the actual bootstrap samples by evaluating the squared error $\varepsilon_n(h) = \int \left(\widehat{f}(\boldsymbol{x}, h_0) - \widehat{f}_n(\boldsymbol{x}, h)\right)^2 d\boldsymbol{x}$. Then, the bootstrap-derived distribution of the squared errors is used to infer minimum, maximum, median, $P_{0.025}$ (2.5%) and $P_{0.975}$ (97.5%) percentiles of squared error for every value of h. This information is reported to the user at every h value. The h value producing the lowest values of the error estimates (we use the median of $\varepsilon_n(h)$ in the case studies in this paper) is the one selected as the optimum bandwidth.

This procedure has been implemented in the *mpdfestimator_bootstrap* program. It takes as input all the observed points and optionally (1) a reference bandwidth value, (2) a range of bandwidth values to be evaluated, (3) the boundaries of the evaluation space, and (4) the number of repetitions for the random sampling. If (1) is missing, the default corresponding to a multidimensional gaussian distribution with the same sample size is applied. If (2) is missing a range (±20% around (1), with a step such that the maximum bandwidth interval is divided in 10 subintervals) is defined. In case (3) is missing, *mpdfestimator_bootstrap* defines a range that ensures a space that surrounds all the observed points. Finally, if (4) is missing, 500 realizations are performed. The program generates as output squared errors for each of the provided bandwidth values. The pseudo-code is shown in Listing 2.

```
Compute the PDF for the reference bandwidth h0
for each h in the range [hmin,hmax]{
      for iter=1 to max_repetitions{
            generate a random sub-sample S
            compute PDF for S
            compute squared error
      }
      generate statistics of squared error
}
return statistics
```

Listing 2. Pseudo-code for the *mpdfestimator*_bootstrap program.

2.3.2. Selection of the optimal bandwidth

The second step that we describe (although it should be the first step in the application of the programs) is to find the optimal bandwidth value for the PDF computed in the first step. It is well known that the computation of the optimal bandwidth to be used in PDF estimations using kernels is a critical step in obtaining reliable PDFs. There are two major strategies for the determination of the optimal bandwidth (Scott, 1992; Silverman, 1986): cross-validation (Duong and Hazelton, 2005) and smoothed bootstrap (Faraway and Jhun, 1990).

The cross-validation approach leads to the convolution of the kernel with itself, a very tough mathematical problem for the Epanechnikov kernel with an open number of dimensions. It is usually solved by means of gaussian multiplicative kernels (Duong and Hazelton, 2005), but this wouldn't allow us to use the OPB approach explained in Section 2.3.1 above. In our case, we have selected the use of smoothed bootstrap estimates of the optimal bandwidth, since it simplifies the generalization of the solution to an open number of dimensions in the multidimensional case for the non-multiplicative Epanechnikov kernel we are using. In order to produce the new estimations in the multidimensional case we take advantage of the fact that the kernel is spherically symmetric in the space corresponding to the spherically symmetric principal components. Thus, the same strategy used by univariate kernel estimations is used for every direction in the space spanned by the spherically symmetric principal components (Silverman, 1986). Surrogate samples are created in this space, and this procedure guarantees that the structure of the covariance matrix is properly preserved.

2.3.3. Computation of the PDF score

The final step of the methodology is to compute the PDF score. Once the user has computed the PDFs (by means of the *mpdfestimator* program) corresponding both to the model and the observations using the optimal bandwidth value reported by *mpdfestimator_bootstrap*, program *mpdf_score* has to be executed to get the PDF score against the reference model.

The program *mpdf_score* takes as input two n-dimensional PDFs stored as netCDF files, generated for the same domain by the first program *mpdfestimator*, and provides as output a PDF-index S by means of the following equation (adapted in this case for a three-dimensional example): $S = \sum_{i,j,k} \min\left(Z^o_{ijk}, Z^m_{ijk}\right) dx_i dx_j dx_k$, where Z^o_{ijk} and Z^m_{ijk} refer to the evaluation of the PDF from observations and the model, respectively. Please note that for higher dimensions, the extension is straightforward.

The equation closely follows the one used by Perkins et al. (2007) or Maxino et al. (2008), but has been extended in this case for its use with a PDF defined in a multi-dimensional (n-dimensional) space. Additionally, when working in several dimensions, the volume of the n-dimensional interval where the PDF is being computed must be taken into account for normalization purposes, and so, the dx_i, dx_j and dx_k terms account for the fact that the range of the different variables can be very different (the average standard deviations of the coefficients in our first case study are 1.9 K, 9.7 K and 2.1 K). The program warns the user in case the bias for any of the dimensions is greater than 5% of the standard deviation of that variate.

2.4. Representation of marginalized PDFs for the interpretation of results

Finally, even though it is not part of the methodology we propose, in order to be able to identify the differences in the index corresponding to individual models and for illustration purposes of the results corresponding to the first case study, we have computed marginalized $\hat{f}_{2D}(c_i, c_j, h) = \int \hat{f}(\boldsymbol{x}, h) dc_k, \quad i \neq j \neq k$ two-dimensional PDFs and projected them onto the $i-j$ C0–C1, C0–C2 and C1–C2 planes, after marginalizing k axes C2, C1 and C0, respectively. This will allow us to show that using a single multidimensional score is better than using a set of unidimensional scores. We only present marginalized PDFs for the first case study in the paper and, for the second case study we just collect the aggregated values of the score in a table.

3. Results

3.1. Application to climate model simulation of the daily cycle of surface temperature

Fig. 3 shows the evolution of the median of the squared errors and the 95% confidence interval computed from the bootstrap analysis corresponding to the ERA40 data when the reference PDF is computed with two conservative estimates of bandwidth ($h_0 = 0.8$ and $h_0' = 0.67$) against the bandwidth that would correspond to the same sample size for a gaussian PDF, 0.637. It can be seen that the bootstrap estimate suggests a slightly lower value (0.55–0.60) of the bandwidth parameter than the one that would correspond to a gaussian multidimensional PDF. As will be identified in the marginalized PDFs later, this is to be expected, since the zonally averaged surface temperature is very non-normal and periodic, so that several fine scale features of the PDF must be resolved, and they can only be properly resolved if the bandwidth is not very high. Therefore, in the following steps an optimum bandwidth of $h = 0.6$ will be used unless otherwise explicitly stated.

In order to test the sensitivity of the classification to different values of the bandwidth used, Table 2 presents the results of the multidimensional PDF scores for different values of the bandwidth parameter (every model is centered and checked against ERA40). For the optimum bandwidth ($h = 0.6$), the best model available is the alternative reanalysis that is used in this study (the NCEP). This is something that we expected from the beginning, since both reanalyses are based on observations. This result supports the use of the method, since the method yields better results for alternative observation-based reanalyses. MIROC3.2-MR and HADGEM1 are the models that follow. Some of the model runs differ only on the initial conditions and most of them are grouped together, with the exception of HADGEM1. The interpretation of this result is that the variability of the index to the use of different initial conditions is very low, as should be expected. MIROC3.2-HR, GFDL, ECHAM5 and BCM2 follow the previous models. The ranking finishes (for the subset of models and diagnostic variable used in this study) by the five random runs corresponding to the MRI model. All the runs corresponding to MRI are grouped, with low values of the score that do not mix with values corresponding to the rest of the models. It seems, therefore, that the intra-ensemble variance is in general (without the exception of MIROC3.2-MR and HADGEM1) smaller than the inter-model variance of the score. In general, the main characteristics of these results are robust even with changes in the bandwidth that span a −33% to +33% interval from the optimum value found by means of bootstrap. The models that show the highest (lowest) performances with the optimum value of the bandwidth continue showing a similar performance for higher or lower values of the bandwidth. There are occasional excursions of a model to at most one alternative position up/down of the ranking, but, on the whole, models tend to maintain their relative ranks even when the bandwidth is changed by a ± 33% relative change around the optimum value.

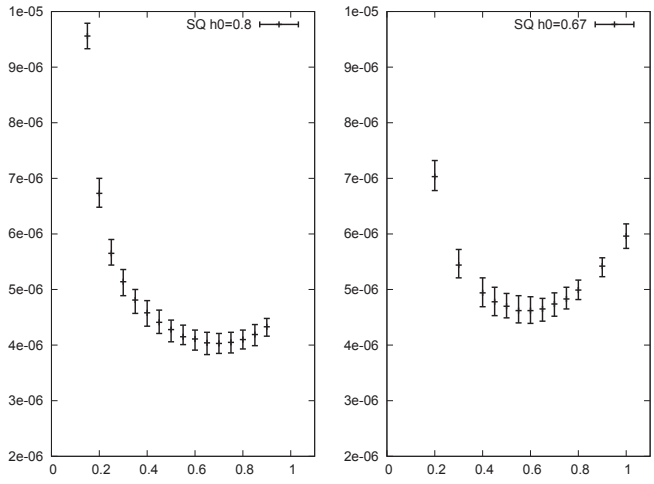


Fig. 3. Squared errors (median and 95% confidence interval as derived from the bootstrap estimates) between the randomly generated PDFs and the reference PDF (left, $h_0 = 0.8$ and right, $h_0 = 0.67$) used for the generation of the smoothed bootstrap.

Figs. 4–6 show the plots of the marginalized PDFs for the case of the NCEP (contours) versus ERA40 (shaded), a model showing a high value of the score, MIROC-3.2-MR (contours) versus ERA40 (shaded) and a model with a lower score, such as MRI (contours) versus ERA40 (shaded). In order to show simple numbers in the plots and scales, values of the marginalized PDFs are multiplied by one thousand before plotting. Before computing the PDFs, the biases between every model and ERA40 have been removed by centering all the series.

Fig. 4, left, shows that on the C0–C1 plane, the PDF is clearly bimodal, as should be expected from a periodic deterministic signal such as the seasonal cycle of temperature. C0 represents the global average of surface temperature and C1 represents the difference in temperature between the Northern and Southern Hemispheres. The main clusters of the C0–C1 PDF appear aggregated around each Hemisphere's summer. NCEP values show a slightly warmer global temperature (C0) during Southern Hemisphere summer than the values shown by ERA40. Fig. 4 (middle) shows that the amplitudes and phases of the mean global temperature (C0) and the equatorial bulge (C2) are similar in both reanalyses. On the C1–C2 plane, the marginalized PDF shows that the main difference between both reanalyses appears as a slightly higher difference of temperature between hemispheres (C1) in NCEP when the coefficient representing the equatorial bell (C2) is positive (summer in the Northern Hemisphere). However, the PDFs generated by both reanalysis are extremely similar, as reflected in the high value of the S index between NCEP and ERA40 (0.82). This is something that we expected from the beginning, since they correspond to observational datasets.

Fig. 5 corresponds to the marginal PDFs for MIROC3.2-MR model (second run), one of the best CMIP3 models according to the metric selected in this study. Over the C0–C1 plane (left), there is quite a good agreement between both PDFs, since both clearly represent the bimodal structure of the PDF. However, the differences between MIROC3.2-MR and ERA40 are higher than in the previous case, both in terms of the location of the Northern Hemisphere summer and also in transitions between seasons that appear in the areas between the maxima in the marginal PDF. In the case of the C0–C2 plane (middle), the highest disagreement appears at the precise location of the maxima of the marginal PDFs, particularly during Northern Hemisphere summer. A similar diagnostic can be derived from the marginal PDF over the C1–C2 plane. Despite both marginal PDFs are clearly bimodal, slight differences exist at the placing

Table 2
Values of the multidimensional *S* score corresponding to different values of the bandwidth parameter and associated rankings that would correspond to the models, when compared with ERA40 reanalysis data.

Model	$h = 0.4$		$h = 0.5$		$h = 0.6$		$h = 0.7$		$h = 0.8$	
	S score	Rank	*S* score	Rank	*S* score	Rank	*S* score	Rank	*S* score	Rank
BCM2.0	0.45	10	0.48	9	0.51	9	0.53	9	0.56	9
ECHAM5	0.45	9	0.47	10	0.48	10	0.50	10	0.51	10
GFDL-CM2.0	0.55	8	0.58	8	0.60	8	0.61	8	0.63	8
GFDL-CM2.1	0.58	7	0.60	7	0.62	7	0.64	7	0.65	7
HADGEM1	0.66	5	0.69	4	0.71	4	0.72	4	0.73	4
MIROCS3.2-HR	0.62	6	0.65	6	0.67	6	0.70	6	0.71	6
MIROCS3.2-MR-RUN01	0.66	4	0.68	5	0.70	5	0.72	5	0.73	5
MIROCS3.2-MR-RUN02	0.69	2	0.72	2	0.74	2	0.75	2	0.77	2
MIROCS3.2-MR-RUN03	0.69	3	0.71	3	0.73	3	0.74	3	0.75	3
MRI-RUN01	0.25	13	0.27	13	0.29	13	0.31	13	0.33	14
MRI-RUN02	0.25	14	0.27	14	0.29	14	0.31	14	0.33	13
MRI-RUN03	0.25	11	0.28	11	0.30	11	0.32	11	0.34	11
MRI-RUN04	0.25	12	0.27	12	0.29	12	0.32	12	0.34	12
MRI-RUN05	0.23	15	0.25	15	0.28	15	0.30	15	0.32	15
NCEP	0.80	1	0.81	1	0.82	1	0.83	1	0.84	1

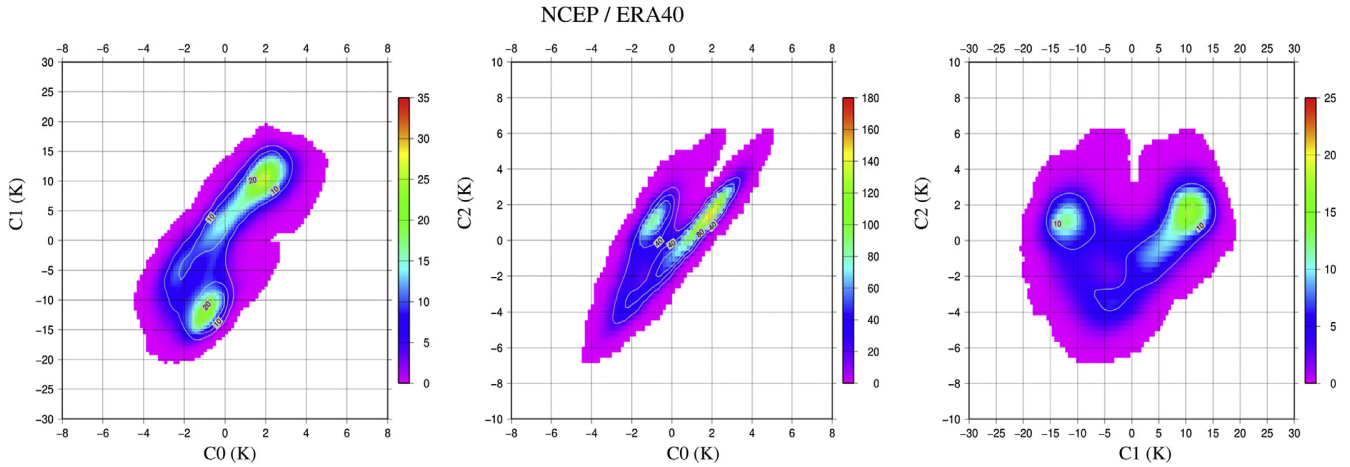


Fig. 4. Marginal PDFs of NCEP (contour) and ERA40 (shaded) projected onto the planes defined by the C0–C1 coefficients (left), C0–C2 coefficients (middle) and C1–C2 coefficients (right). Values of the PDF have been multiplied by 1000 in order to improve the representation of numbers.

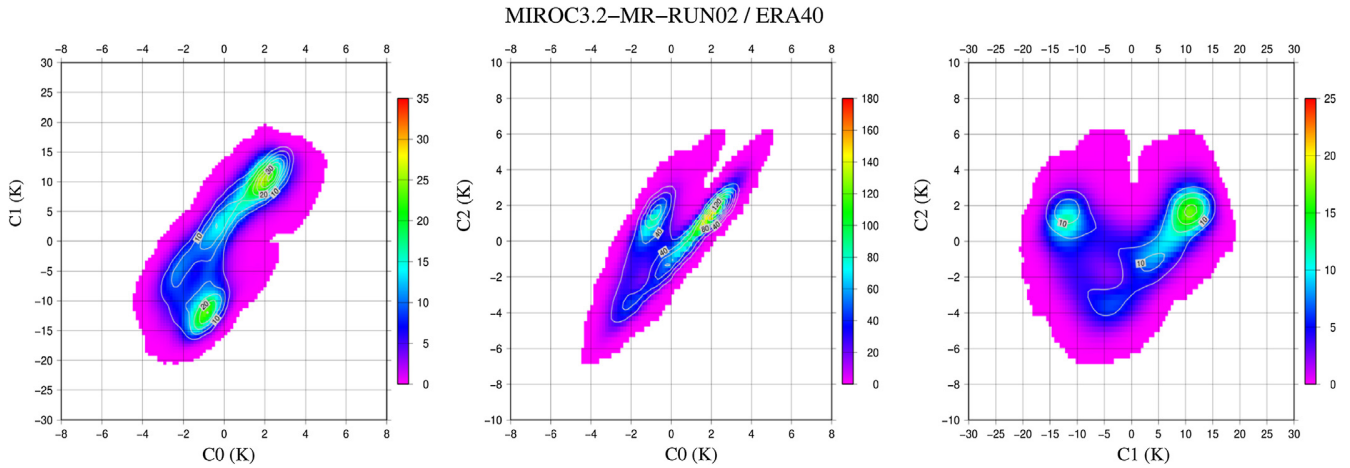


Fig. 5. Marginal PDFs of MIROC3.2-MR (run 2, contours) and ERA40 (shaded) projected onto the planes defined by the C0–C1 coefficients (left), C0–C2 coefficients (middle) and C1–C2 coefficients (right). Values of the PDF have been multiplied by 1000 in order to improve the representation of numbers.

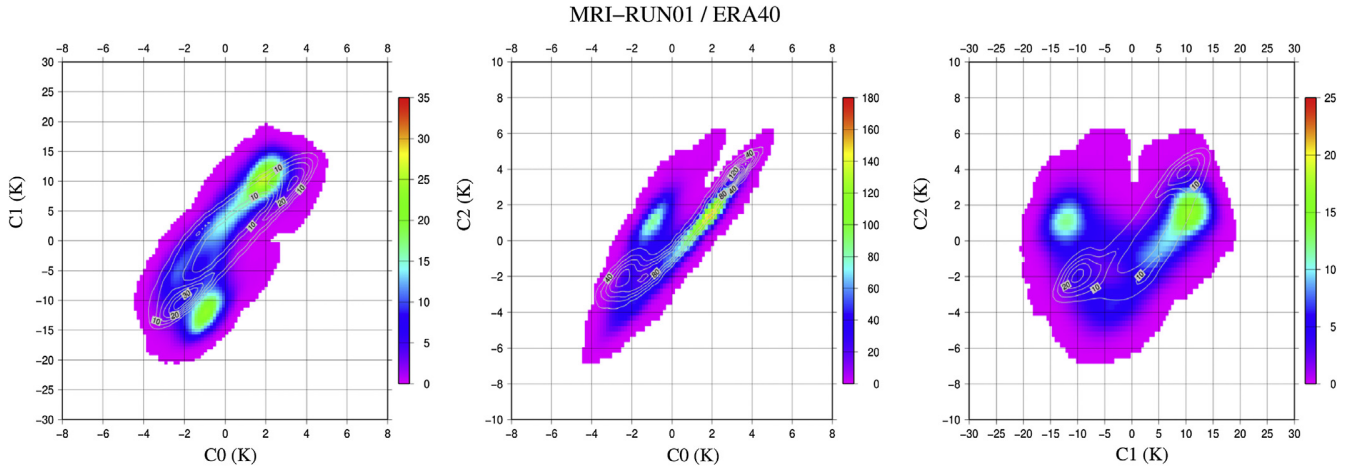


Fig. 6. Marginal PDFs of MRI (run 1, contours) and ERA40 (shaded) projected onto the planes defined by the C0–C1 coefficients (left), C0–C2 coefficients (middle) and C1–C2 coefficients (right). Values of the PDF have been multiplied by 1000 in order to improve the representation of numbers.

of the PDF maxima. The equatorial bell (C2) in MIROC3.2-MR is stronger than the one in ERA40 during negative phases (Northern Hemisphere winter) of inter-hemispheric temperature differences (C1).

Fig. 6 corresponds to MRI (run 1) model. The evolution of the daily seasonal cycle of temperature in terms of C0 (global T) and C1 (inter-hemispheric temperature contrast, left) does not present a bimodal structure with the PDF maxima placed at the same points shown by the reanalysis. The transitions between summer and winter regimes happen through routes that do not correspond to the ones in the ERA40 Reanalysis. The structure of the marginal PDF for the C0–C2 plane is markedly different between MRI and ERA40, with the cold maximum in the PDF during summer in the Southern Hemisphere quite misplaced in the case of MRI. This is also apparent in the marginal PDF corresponding to the C1–C2 plane, where maxima of the PDFs do not appear neither on the same places nor even with the same phases.

Finally, Fig. 7 shows that the index is very sensitive to the existence of a bias between the models and reference observations. In this case, the PDFs are computed without previously removing the bias between the surrogate model (NCEP data) and the observations (ERA40) and the S score index that we get between ERA40 and NCEP reanalyses is extremely low ($S = 0.075$). The marginal PDFs show that in general there is a very good agreement in the structure of the 3D PDFs, but the center of masses of both PDFs are not located at the same places. The biases for every coefficient are not very high, considering their variances. The bias of the C0 component is 0.7 K (0.2% relative error), the bias in C1 is 0.4 K (7.5% relative error) and the bias in C2 is −0.34 K (−1.33% relative error). However, even such low values of the bias lead to a score index that could be interpreted as poor performance of the surrogate model (NCEP reanalysis) versus ERA40 due to the complex structure of the 3D PDF. However, this is a false impression that can not be defended if the spatial patterns of the marginal PDFs are analyzed in detail. This means that the index should not be applied to model results that are biased against the reference observations. The existence of biases in the models leads to greater observational uncertainty when the model and observational datasets are not centered. The code does not force the centering of the datasets and, therefore, the user must take care of this when the dimensionality reduction stage of the data analysis is done. In particular, it is interesting to stress that, internally, when computing the n-dimensional PDFs, all the datasets are centered (each one using its n-dimensional average) before computing the corresponding spherically

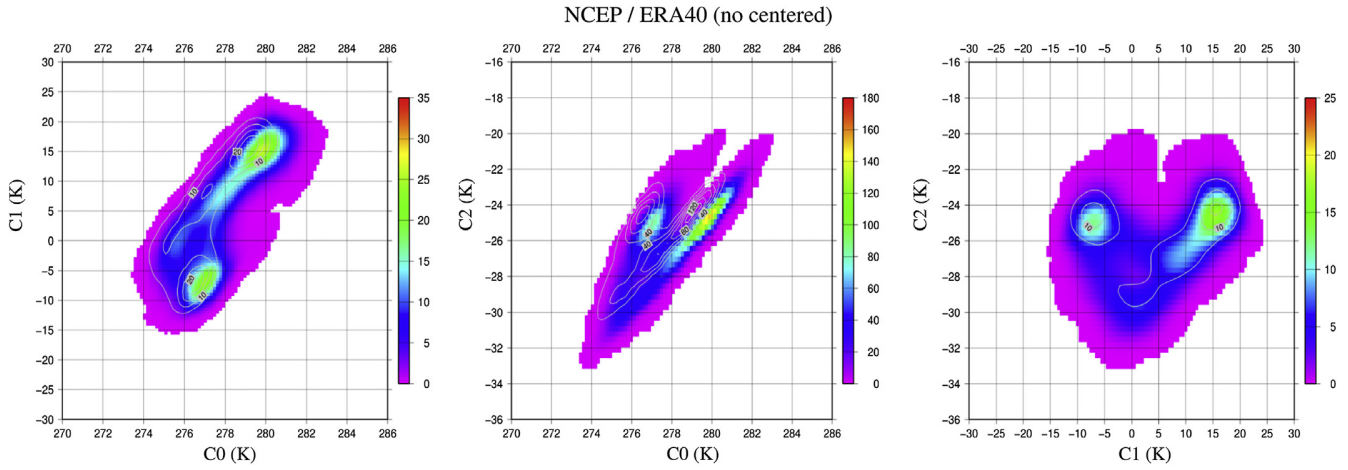


Fig. 7. Marginal PDFs of non-centered NCEP (contours) and ERA40 (shaded) projected onto the planes defined by the C0–C1 coefficients (left), C0–C2 coefficients (middle) and C1–C2 coefficients (right). Values of the PDF have been multiplied by 1000 in order to improve the representation of numbers. The bias between both reanalysis has been retained.

symmetric principal components. When the output netCDF files holding the PDFs are saved, the original units in the phase space of each dataset (model or observations) are recovered and the average is added to the anomalies derived from the PDF in the principal component space stored in the memory of the computer. Therefore, the key point here is that if there exists a constant bias between the model and the reference observations (first and second netCDF files passed to program mpdf_score), it could lead to very low values of the score despite the model representing properly the variability (anomalies). This means that the evaluation of the models in terms of a constant bias and the *n*-dimensional PDFs should be carried out as different steps.

From the point of view of performance, we have measured the execution time needed by each version of the program to complete the bootstrap procedure. On average, the serial OPB approach is 140 times faster than the serial GPB approach and, moreover, the parallel OPB program scales linearly with the number of cores, being 4.3 times faster than its serial counterpart when using 4 cores. This means that an evaluation of a single model that takes approximately 22 days with the serial GPB program, can be executed in less than 1 h using the most efficient and parallel implementation presented in this contribution. These experiments have been conducted in a desktop computer with an Intel i7 3820 processor (four cores, 3.6 GHz, Hyperthreading enabled) with 8 GB of RAM. Therefore, the use of this technique is not limited to the availability of specialized clusters or hardware that would limit its practical use.

3.2. Evaluation of Sea Surface Temperature and Sea Surface Height

The first two PCs of the global coverage weekly time-scale SST (T1, T2) and SSH (H1, H2) variables belonging to the ARMOR-3D (Guinehut et al., 2004, 2012) and CFSR (Saha et al., 2010, 2014) datasets will be used in the following to evaluate the second with respect to the former. This means that the ARMOUR-3D product (blended satellite and in-situ observation product) is the reference to evaluate the CFSR product (coupled atmosphere-ocean modeling product). Considering the first two PCs of each variable in the evaluation (T1, T2, H1, H2), the global-scale main variability modes of each variable are being take into account at a glance. As the seasonal cycle was not explicitly removed from the anomalies used to deduce the PCs, the four considered variables are almost completely related to the global-scale seasonal cycle (H2 contains some longer time-scale variability). Thus comparing combinations of different variables from CFSR with those of ARMOR-3D the capacity of the modeling product to jointly characterize different main global-scale variability modes (their seasonal cycles) is evaluated. For example, if T1 and H1 are considered at the same time (case T1H1) the capacity of CFSR to simulate the main global-scale components of the seasonal cycle of the SST and SSH variables is being evaluated in a single and multivariate score.

Table 3 shows the optimal *h* and the score obtained with the 6 analyzed cases going from the univariate T1 and H1 cases, the multi-dimensional univariate T1T2 and H1H2 (reserving the term multivariate to the cases with variables with different physical dimensions, i.e. Kelvins and meters) and the multivariate T1H1 and T1T2H1H2 cases. All variables have zero mean so no bias related issues will be observed in this case. Like in the previous case study on the TAS, the same three-step methodology was applied in this case: for a given row in Table 3, the optimal *h* using the bootstrap procedure is initially estimated. Next, the PDF using the optimal *h* is computed and, finally, the score (one dimensional, multidimensional or multivariate) is computed from the PDFs obtained from the CFSR and the ARMOR-3D variables.

The very high scores of T1 (0.98) and T1T2 (0.97) cases indicate that the model is reproducing well the global-scale SST seasonal cycle. The same stands for the SSH, as slightly lower but still high values are observed in the analog H1 (0.84) and H1H2 (0.79) cases. This difference between the SST and SSH is not strange as it is only showing the fact that the modeling of the SSH involves many more processes (dynamic anomalies, steric anomalies,...) compared to that of the SST (mixed layer heat-budget and currents). The multivariate cases T1H1 (0.67) and T1T2H1H2 (0.42) indicate a loss of performance with growing number of the dimensions of the multivariate PDF, or in other words, with increasing complexity. Any error in any of the tested dimensions leads to a smaller value of the score. It can be seen that, although univariate scores point to good modeling performances, the multivariate approaches combining the same variables yield worse results. In this application a single model is being evaluated, but results show that differences appear when considering multivariate scores that discriminate between errors that appear through all the phase space.

Table 3
Optimal *h* and resulting scores obtained from the evaluation of CFSR (model) with respect to ARMOR-3D (observations) based on the first PCs of the SST (T1, T2) and SSH (H1, H2) from each of the products. Univariate unidimensional (T1 and T2), univariate multidimensional (T1T2 and H1H2), multivariate unidimensional (T1H1) and multivariate multidimensional (T1T2H1H2) cases are shown to illustrate the number of potential combinations that can be considered in the framework of the proposed methodology.

Case name	*h* (optimal)	Score
T1	0.36	0.98
H1	0.54	0.84
T1T2	0.52	0.97
H1H2	0.69	0.79
T1H1	0.58	0.67
T1T2H1H2	1.04	0.42

4. Discussion

4.1. Comparison with the results of the evaluation of models using previous techniques

The results presented in this contribution extend the previous analysis by Errasti et al. (2013) with a new methodology. Therefore, the obvious first part of the discussion is a comparison of the rankings obtained by this methodology and the final rank obtained in the previous study. It can be seen that, for the most part, the aggregated rankings behave similarly in the current and previous studies (see Table 4). In general, the best models (NCEP, MIROC3.2-MR and MIROC3.2-MR) keep a good ranking also under the new metric proposed here. For the worst case (MRI-CGCM2.3), the same result holds, too. However, some changes appear in the relative rankings of models in the middle part of the classification. It must be kept in mind that the rankings in Errasti et al. (2013) also considered the root mean square errors between the seasonal

Table 4
Global rankings for the models according to their representation of the daily zonally averaged temperature in Errasti et al. (2013) and according to the methodology proposed in this contribution (3D-PDF).

	Errasti et al. (2013) rank	3D-PDF rank
BCM2.0	5	6
GFDL-CM2.0	7	5
GFDL-CM2.1	6	4
MIROCS3.2-HR	2	3
MIROCS3.2-MR	3	2
ECHAM5	4	7
MRI-CGCM2.3	8	8
NCEP	1	1

cycles, so that the rankings can not be completely the same. Anyway, as a result of the new technique, we find a similar classification of models albeit with a single objective index without the need of a posterior averaging of individual scores or ranks. Therefore, we think that this methodology makes it easier and more objective to classify the models according to their performance when simulating several variables.

The technique presented in this contribution is slightly different of those carried out in other model inter-comparison studies found in the climate literature. In some studies estimates of simple or derived climate variables simulated by the models and the observations are computed, but a final single performance rank is not presented (e.g., Maxino et al., 2008; Nieto and Rodríguez-Puebla, 2006; Russell et al., 2006; Vera et al., 2006; Ulbrich et al., 2008). Other studies propose a single final rank but based on the averaging of skill scores, ranks with or without different relative weight (e.g. Fu et al., 2013; Errasti et al., 2011, 2013). The use of this new single multidimensional index on PDFs would avoid the need to establish a final step in order to combine and weight the different climate variables analyzed, since the multidimensional distributional agreement index S already considers the characteristics of the PDF at the multidimensional space covered by the PDF.

An advantage of our technique is the interpretability of the rankings if the marginal PDFs are used the same way as in Figs. 4, fig 5, fig 6, fig 7. The marginal PDFs can be very easily computed from the 3D PDF (in general the n-dimensional PDF) that has been computed to produce the S score. The geometrical structure of the marginal PDFs throws light in terms of the differences between models even when temporal position of every point in the PDF space is lost, since geometrical relations between variables allow us to interpret them. This can not be achieved when using one-dimensional PDFs, since in that case, the relationship of the points between different variables can not be resolved.

4.2. Multivariate kernel density estimators in the context of model performance evaluation methods

In this subsection we consider the methodology for model evaluation based on multidimensional kernel density estimators proposed in this contribution in the general context of the model performance evaluation. Bearing in mind only the two case studies analyzed in the Results section, we could restrict this section to the framework of the modeling of geophysical fluids. There's no reason, however, to restrict the discussion in such a manner. On the contrary, and as already stated, the framework will be that of the evaluation of the performance of environmental models in general. To put the proposed technique in context, we will refer to a recent position paper by Bennett et al. (2013) in this journal on the characterization of the performance of environmental models. The paper by Bennett et al. (2013) describes a series of methods and procedures for both qualitative and quantitative model performance characterization. It also proposes a general purpose five-step recipe for model skill evaluation. Direct references will be made to the classification of evaluation methodologies and the five-step recipe proposed in Bennett et al. (2013), avoiding the inclusion of redundant information or descriptions here. Therefore, the reader is referred to that contribution for a complete understanding of the forthcoming discussion.

In the case of the classification categories of quantitative measures of model performances found in Bennett et al. (2013), the method based on multidimensional kernel density estimators would fall in more than a category at the same time. First, amongst the direct value comparison methods that compare all model and observation values as a whole (even for multidimensional and multivariate cases) to give a single value metric, the multidimensional score that is the result in our case. Note, however, that this classification is valid only for the final step of the technique when the multivariate (multidimensional) PDFs belonging to the observation and the model are compared. Note also that this PDF to PDF comparison could be understood as a high precision (number of categories) multidimensional contigency table, since PDFs are actually evaluated on a discrete grid. According to this, the last comparison step would fall amongst concurrent comparison methods. In addition, and paying attention to the step that is carried before the PDFs are compared, it is clear that the method must also be considered within the data transformation method class (transformation to PDF space). Summarizing and according to the classification of quantitative model performance evaluation tools described in Bennett et al. (2013) the proposed technique can be considered as a direct comparison method taken as a whole, but also as a data transformation method and a concurrent comparison method if two inner steps are considered separately.

Although they are not part of the proposed methodology, additional verification techniques considered in Bennett et al. (2013) have also been considered in this contribution, and it is worth mentioning those here. For instance, 2D marginalized PDFs (Figs. 4–7) were used in the first case study to understand by means of a visual inspection the reasons that drove the relative differences in the scores shown in Table 2. Once again not part of the proposed technique, but mentioned in Bennett et al. (2013) and used in our case studies, were the data transformation methods: the Lengendre polynomials or the zonally averaged values in the first case study, and the global-scale PCA analysis of the variables in the second one.

In the case of the five-step general recipe for model performance evaluation proposed by Bennett et al. (2013), our procedure should be taken into account in the fifth step of refinements due to its complexity. This does not mean that previous steps like checking the data before any additional step, performing some visual checks and deducing some basic metrics are to be left aside. In fact, they are more than convenient as part of any good practices procedure for model performance evaluation and were also applied in both case studies shown in Results section.

4.3. Additional capabilities and potential applications of the methodology

The case study described in Section 3.2 shows the evaluation of a single model with two multidimensional variables. The objective was not to make a thorough verification of the selected CFSR model, but to illustrate some concepts of broad applicability like possibility of using and also combining very different variables (TAS, SST and SSH in the presented case studies), the potential of the use of techniques to reduce the dimensionality of the dataset to whom the verification is to be applied (Legrendre Polinomial of zonally averaged values in the first case study, global-scale principal components in the second case), or the relative score changes of the univariate and multivariate cases, to mention some.

With regard to the results shown in Table 3, the most remarkable fact is the reduction of the scores in the multivariate case compared to the considerably better scores obtained in the univariate cases. This could be expected, however, as the growing complexity of the PDF with growing number of dimensions/variables will tend to enhance the differences in the model/observation PDFs. This example, although simple, has other potential uses in addition to the one discussed here. For instance, one may want to evaluate the evolution of the performance of a model to see the impacts of developing stages in the model. Then a comparison like the one in case study two will be useful, specially if the

improvement can be identified by a multivariate verification. Another application could include the joint evaluation of several model variables, like the one in the first case study. In addition, other variables could be added with selective criteria to the analysis (like the surface currents, the depth of the thermocline or the characterization of ENSO, to mention some) without increasing too much the number of variables, but considering many aspects of the physics accounted by the modeling.

4.4. Limitations of the methodology

As with any other model evaluation index, this index has limitations too. First, it is probably not adequate to detect the probability of very infrequent events, since multidimensional PDF estimations tend to be unreliable on those areas where the value of the PDF is already low. It has been shown in this contribution that its reliability is very low if there exists a bias between models and observations. Secondly, the sample size must be high enough for a modest dimensionality to be analyzed. It is well known that, when analyzing multidimensional datasets, the phase space is too empty when the dimensionality of the space grows (Bellman, 1961). This might also happen with environmental models in general if some kind of initial dimensionality reduction step was not applied to direct model output at the grid point level. Therefore, working with daily data for climatic applications is almost a must. For other kind of environmental models, fast time scales should in any case be used.

Anyway, it has to be considered that multivariate density estimation is always computationally expensive, so that the dimensionality of the dataset must be kept modest. When the dimensionality of the problem increases, the CPU needed increases as a result of the increase of grid points and the complexity of the linear algebra operations performed. Additionally, for high dimensionality, the size of the netCDF files and the memory needed to evaluate them scale too fast and the user is faced with limitations in the hardware of computers (memory and disk). As a result, we must conclude that it is always necessary to perform an initial step of data dimensionality reduction that allows to work in a fundamental set of low-dimensionality variables that reproduce the behavior of the system. In the two case studies presented in this contribution, the initial step of dimensionality reduction has been performed by projections onto Legendre polynomials or by means of principal component analysis.

Brands et al. (2012) showed that, in some circumstances, particularly when there exists a clustering of data near zero (such as it happens for specific humidity), the Kolmogorov–Smirnov test can be better applied than the univariate similarity index equivalent to the multidimensional one used in this paper. In addition, the existence of a well known distribution corresponding to the Kolmogorov–Smirnov test allows the researcher to make use of statistical hypothesis testing. This can not be done unless Monte Carlo techniques are used for the univariate similarity index. However, when working in multiple dimensions, the univariate Kolmogorov–Smirnov test can not be extended to several dimensions above two or even three (Fasano and Franceschini, 1987; Justel et al., 1997; Lopes et al., 2008; Peacock, 1983). Since the methodology that we propose in this study can work in multiple dimensions, we find that this methodology will, hopefully, allow to perform an easier evaluation of models according to several criteria, as was originally intended.

5. Conclusions

The index based on the common area between PDFs that is frequently used in the univariate evaluation of climate models, and that is computed as the common area under the PDFs corresponding to models and observations, has been extended to multidimensional problems. In addition, tools that allow its use have been developed and made freely available as open source software. The tools compute the kernel-based multidimensional PDF, identify the optimum value of the bandwidth by means of smoothed bootstrap and compute the common volume under two n-dimensional PDFs.

The use of multidimensional PDFs is very intensive in terms of CPU time, and it is particularly so for the case of the bootstrap estimation of the bandwidth, since several realizations of the PDF must be computed for every bandwidth value tested. The availability of a parallel version of the tool for higher dimensionality and for the bootstrap allows to carry out those computations in short times (less than 1 h in our case, using standard hardware).

The use of a multidimensional analysis produces a single index corresponding to every model even after analyzing several variables, and this result makes it easy to perform evaluation of the models under several target variables. In the contribution presented here, we have explored one case such as three Legendre coefficients that expand the daily cycle of zonally averaged temperature. However, the same approach could be applied to the joint analysis of temperature, outgoing longwave radiation or cloud cover (to name a few) such that the structure of the multidimensional PDFs (probably properly marginalized as in this contribution) could shed light over the behavior of models according to known physical mechanisms.

Even though a tool of the set described in this contribution allows to make an objective selection of the optimal bandwidth to be used in the generation of the PDFs by means of smoothed bootstrap, the case study in this paper shows that the ranking of the models is quite robust even under severe ($\pm$30% of the optimal bandwidth) changes in the bandwidth used for the generation of the multidimensional PDFs. Thus, the results obtained through the use of the tools presented in this contribution are reliable.

However, the index is extremely sensitive to the existence of a constant bias between models and it should not be used without previously centering the data, a finding in agreement with previous studies using univariate PDFs (Brands et al., 2011, 2012). The programs do not request that the datasets are centered, but a constant bias between the model and observations could lead to unphysical diagnostics in several dimensions. A potential solution is to perform the evaluation onto n-dimensional centered data, so that the bias is automatically removed. Alternatively, the analysis can be performed removing the bias from the model results with respect to observations. Therefore, the analysis of the bias must still be kept independent from the analysis of the shape of the PDF presented in this contribution. The current implementation of the mpdf_score program provides a warning if the bias at any of the dimensions is greater than 5% of the standard deviation of the corresponding variate.

The second case study demonstrated the applicability of the proposed methodology to multivariate and multidimensional data using data from oceanographic SST and SSH variables too. In addition, and although it is not part of the proposed technique, this case study also demonstrated the potential of the use of a preprocessing step for the reduction of the dimensionality of the data, based on a PCA analysis of the original dataset in this case. This shows that the method can potentially be applied to a large family of environmental problems.

The overall evaluation of environmental models is a complex task and different performance scores detect different weak or strong points of the available global models. We hope that the addition of a new methodology and tools that allow its easy application by other researchers make it easier the identification in future experiments of areas of models that can be improved.

Acknowledgments

Authors acknowledge constructive comments by three referees and the editor of this paper. These comments have lead to an improved version of the manuscript. We acknowledge the modeling groups, the Program for Climate Model Diagnosis and Inter-comparison (PCMDI) and the WCRP's Working Group on Coupled Modeling (WGCM) for their roles in making available the WCRP CMIP3 multi-model dataset. Support of this dataset is provided by the Office of Science, U.S. Department of Energy. ECMWF ERA-40 data used in this study have been provided by ECMWF. NCEP reanalysis data provided by the NOAA/OAR/ESRL PSD, Boulder, Colorado, USA, from their Web site at http://www.esrl.noaa.gov/psd/have been used. CFSR and CFSv2 data was provided by the Research Data Archive at the National Center for Atmospheric Research, Computational and Information Systems Laboratory, Boulder, Colorado. ARMOUR-3D data was obtained from MyOcean (http://www.myocean.eu/). Authors thank financial funding by project CGL2013-45198-C2-1-R (MINECO, National R + D + i plan), the SAIOTEK program from the Basque Government (project S-P11UN137). Additional funding from different calls from the University of the Basque Country (UFI 11/55, PPM12/01 and GIU 11/01) has allowed this paper to be finished. This work has also been partially supported by the Saiotek and Research Groups 2013–2018 (IT-609-13) programs (Basque Government),TIN2013-41272P (Ministry of Science and Technology), COMBIOMED-RD07/0067/0003 network in computational bio-medicine (Carlos III Health Institute). U. Lopez-Novoa holds a grant from the Basque Government. J. Miguel-Alonso and A. Mendiburu are members of the HiPEAC European Network of Excellence. Author contributions: JS, AM and JMA designed the research; ULN, JS, AM and JMA wrote the code distributed with this contribution; IE, AE, GIB and JS performed the computations that lead from data in the CMIP3 repository to the Legendre coefficients used in the first case study; GE performed the computations for the second example, ULN, JS, AM and JMA prepared the specific computations, graphics and results used in this contributions after the Legendre coefficients and ULN, JS, AM, IE, GE and JMA wrote the paper.

References

Ahamada, I., Flachaire, E., 2010. Non-parametric Econometrics. Oxford University Press, Oxford, 176 pages.

Andrews, T., Gregory, J.M., Webb, M.J., Taylor, K.E., 2012. Forcing, feedbacks and climate sensitivity in CMIP5 coupled atmosphere-ocean climate models. Geophys. Res. Lett. 39, L09712. http://dx.doi.org/10.1029/2012GL051607.

Annan, J.D., Hargreaves, J.C., 2010. Reliability of the CMIP3 ensemble. Geophys. Res. Lett. 37, L02703. http://dx.doi.org/10.1029/2009GL041994.

Bellman, R., 1961. Adaptive Control Processes: a Guided Tour. Princeton University Press, 255 pp.

Bengtsson, L., 2013. What is the climate system able to do “on its own”? Tellus B65, 20189. http://dx.doi.org/10.3402/tellusb.v65i0.20189.

Bennett, N.D., Croke, B.F.W., Guariso, G., Guillaume, J.H.A., Hamilton, S.A., Jakeman, A.J., Marsili-Libelli, S., Newham, L.T.H., Norton, J.P., Perrin, C., Pierce, S.A., Robson, B., Seppelt, R., Voinov, A.A., Fath, B.D., Andreassian, V., 2013. Characterising performance of environmental models. Environ. Model. Softw. 40, 1–20. http://dx.doi.org/10.1016/j.envsoft.2012.09.011.

Boer, G.J., Lambert, S.J., 2001. Second-order space-time climate difference statistics. Climate Dyn. 17, 213–218. http://dx.doi.org/10.1007/PL00013735.

Braconnot, P., Harrison, S.P., Kageyama, M., Bartlein, P.J., Masson-Delmotte, V., Abe-Ouchi, A., Otto-Bliesner, B., Zhao, Y., 2012. Evaluation of climate models using palaeoclimatic data. Nat. Climate Change 2, 417–424. http://dx.doi.org/10.1038/nclimate1456.

Brands, S., Herrera, S., San-Martín, D., Gutiérrez, J., 2011. Validation of the ENSEMBLES global climate models over southwestern Europe using probability density functions, from a downscaling perspective. Climate Res. 48, 145–161. http://dx.doi.org/10.3354/cr00995.

Brands, S., Gutiérrez, J.M., Herrera, S., Cofiño, A.S., 2012. On the use of reanalysis data for downscaling. J. Climate 25, 2517–2526. http://dx.doi.org/10.1175/JCLI-D-11-00251.1.

Chu, P.C., 1999. Two kinds of predictability in the lorenz system. J. Atmos. Sci. 56, 1427–1432 http://dx.doi.org/10.1175/1520-0469(1999)056<1427:TKOPIT>2.0.CO;2.

Dagum, L., Menon, R., 1998. OpenMP: an industry standard API for shared-memory programming. Comput. Sci. Eng. IEEE 5, 46–55. http://dx.doi.org/10.1109/99.660313.

DeAngelis, A.M., Broccoli, A.J., Decker, S.G., 2013. A comparison of CMIP3 simulations of precipitation over North America with observations: daily statistics and circulation features accompanying extreme events. J. Climate 26, 3209–3230. http://dx.doi.org/10.1175/JCLI-D-12-00374.1.

Dessai, S., Lu, X., Hulme, M., 2005. Limited sensitivity analysis of regional climate change probabilities for the 21st century. J. Geophys. Res. 110, D19108. http://dx.doi.org/10.1029/2005JD005919.

Dessler, A.E., 2013. Observations of climate feedbacks over 2000–10 and comparisons to climate models. J. Climate 26, 333–342. http://dx.doi.org/10.1175/JCLI-D-11-00640.1.

Duong, T., Hazelton, M.L., 2005. Cross-validation bandwidth matrices for multivariate kernel density estimation. Scand. J. Stat. 32, 485–506. http://dx.doi.org/10.1111/j.1467-9469.2005.00445.x.

Errasti, I., Ezcurra, A., Sáenz, J., Ibarra-Berastegi, G., 2011. Evaluation of IPCC AR4 models over the Iberian Peninsula. Theor. Appl. Climatol. 103, 61–79. http://dx.doi.org/10.1007/s00704-010-0282-y.

Errasti, I., Ezcurra, A., Sáenz, J., Ibarra-Berastegi, G., Zorita, E., 2013. Comparison of the main characteristics of the daily zonally averaged surface air temperature as represented by reanalysis and seven CMIP3 models. Theor. Appl. Climatol. 114, 417–436. http://dx.doi.org/10.1007/s00704-013-0842-z.

Faraway, J.J., Jhun, M., 1990. Bootstrap choice of bandwidth for density estimation. J. Am. Stat. Assoc. 85, 1119–1122.

Fasano, G., Franceschini, A., 1987. A multidimensional version of the Kolmogorov–Smirnov test. Mon. Not. R. Astron. Soc. 225, 155–170. http://dx.doi.org/10.1093/mnras/225.1.155.

Fernández, J., Primo, C., Cofiño, A.S., Gutiérrez, J.M., Rodríguez, M.A., 2009. MVL spatiotemporal analysis for model intercomparison in EPS: application to the DEMETER multi-model ensemble. Climate Dyn. 33, 233–243. http://dx.doi.org/10.1007/s00382-008-0456-9.

Frederiksen, J.S., Collier, M.A., Watkins, A.B., 2004. Ensemble prediction of blocking regime transitions. Tellus 56A, 485–500. http://www.tellusa.net/index.php/tellusa/article/view/14460.

Fu, G., Liu, Z., Charles, S.P., Xu, Z., Yao, Z., 2013. A score-based method for assessing the performance of GCMs: a case study of southeastern Australia. J. Geophys. Res. 118, 4145–4167. http://dx.doi.org/10.1002/jgrd.50269.

Gates, W.L., Boyle, J.S., Covey, C., Dease, C.G., Doutriaux, C.M., Drach, R.S., Fiorino, M., Gleckler, P.J., Hnilo, J.J., Marlais, S.M., Phillips, T.J., Potter, G.L., Santer, B.D., Sperber, K.R., Taylor, K.E., Williams, D.N., 1999. An overview of the results of the atmospheric model intercomparison project (AMIP I). Bull. Am. Meteorol. Soc. 80, 29–55 http://dx.doi.org/10.1175/1520-0477(1999)080<0029:AOOTRO>2.0.CO;2.

Gleckler, P.J., Taylor, K.E., Doutriaux, C., 2008. Performance metrics for climate models. J. Geophys. Res. 113, D22105. http://dx.doi.org/10.1029/2007JD008972.

Guinehut, S., Le Traon, P.-Y., Larnicol, G., Philipps, S., 2004. Combining argo and remote-sensing data to estimate the ocean three-dimensional temperature fields - a first approach based on simulated observations. J. Mar. Syst. 46 (1–4), 85–98. http://dx.doi.org/10.1016/j.jmarsys.2003.11.022.

Guinehut, S., Dhomps, A.-L., Larnicol, G., Le Traon, P.-Y., 2012. High resolution 3D temperature and salinity fields derived from in situ and satellite observations. Ocean Sci. 8 (5), 845–857. http://dx.doi.org/10.5194/os-8-845-2012.

Henderson-Sellers, A., Pitman, A.J., Love, P.K., Irannejad, P., Chen, T., 1995. The project for intercomparison of land surface parameterisaton schemes (PILPS) phases 2 and 3. Bull. Am. Meteorol. Soc. 76, 489–503 http://dx.doi.org/10.1175/1520-0477(1995)076<0489:TPFIOL>2.0.CO;2.

Hewitt, C.D., Griggs, D.J., 2004. Ensembles-based predictions of climate changes and their impacts. Eos Trans. AGU 85, 566. http://dx.doi.org/10.1029/2004EO520005.

Hind, A., Moberg, A., Sundberg, R., 2012. Statistical framework for evaluation of climate model simulations by u se of climate proxy data from the last millennium – Part 2: a pseudo-proxy study addressing the amplitude of solar forcing. Climate Past 8, 1355–1365. http://dx.doi.org/10.5194/cp-8-1355-2012.

Justel, A., Peña, D., Zamar, R., 1997. A multivariate Kolmogorov-Smirnov test of goodness of fit. Stat. Probab. Lett. 35, 251–259. http://dx.doi.org/10.1016/S0167-7152(97)00020-5.

Kageyama, M., Valdes, P.J., Ramstein, G., Hewitt, C., Wyputta, U., 1999. Northern hemisphere storm tracks in present day and last glacial maximum climate simulations: a comparison of the European PMIP models. J. Climate 12, 742–760 http://dx.doi.org/10.1175/1520-0442(1999)012<0742:NHSTIP>2.0.CO;2.

Kalnay, E., Kanamitsu, M., Kistler, R., Collins, W., Deaven, D., Gandin, L., Iredell, M., Saha, S., White, G., Woollen, J., Zhu, Y., Leetmaa, A., Reynolds, R., Chelliah, M., Ebisuzaki, W., Higgins, W., Janowiak, J., Mo, K.C., Ropelewski, C., Wang, J., Jenne, R., Joseph, D., 1996. The NCEP/NCAR 40-year reanalysis project,. Bull. Am. Meteorol. Soc. 77, 437–470 http://dx.doi.org/10.1175/1520-0477(1996)077<0437:TNYRP>2.0.CO;2.

Knutti, R., 2008. Should we believe model predictions of future climate change? Philos. Trans. R. Soc. A 366, 4647–4664. http://dx.doi.org/10.1098/rsta.2008.0169.

Knutti, R., 2010. The end of model democracy? an editorial comment. Clim. Change 102, 395–404. http://dx.doi.org/10.1007/s10584-010-9800-2.

Knutti, R., Masson, D., Gettelman, A., 2013. Climate model genealogy: generation CMIP5 and how we got there. Geophys. Res. Lett. 40, 1194–1199. http://dx.doi.org/10.1002/grl.50256.

Lamarque, J.-F., Shindell, D.T., Josse, B., Young, P.J., Cionni, I., Eyring, V., Bergmann, D., Cameron-Smith, P., Collins, W.J., Doherty, R., Dalsoren, S., Faluvegi, G., Folberth, G., Ghan, S.J., Horowitz, L.W., Lee, Y.H., MacKenzie, I.A., Nagashima, T., Naik, V., Plummer, D., Righi, M., Rumbold, S.T., Schulz, M., Skeie, R.B., Stevenson, D.S., Strode, S., Sudo, K., Szopa, S., Voulgarakis, A., Zeng, G., 2013. The atmospheric chemistry and climate model intercomparison project (ACCMIP): overview and description of models, simulations and climate diagnostics. Geosci. Model Dev. 6, 179–206. http://dx.doi.org/10.5194/gmd-6-179-2013.

Lopes, R.H.C., Hobson, P.R., Reid, I.D., 2008. Computationally efficient algorithms for the two-dimensional Kolmogorov–Smirnov test. J. Phys. Conf. Ser. 119, 042019. http://dx.doi.org/10.1088/1742-6596/119/4/0420.

Lorenz, E.N., 2006. Predictability, a problem partly solved (Chapter 3). In: Palmer, T., Hagedorn, R. (Eds.), Predictability of Weather and Climate. Cambridge University Press, Cambridge, 702 pp.

Macadam, I., Pitman, A.J., Whetton, P.H., Abramowitz, G., 2010. Ranking climate models by performance using actual values and anomalies: implications for climate change impact assessments. Geophys. Res. Lett. 37, L16704. http://dx.doi.org/10.1029/2010GL043877.

Marshall, A.G., Hudson, D., Hendon, H.H., Pook, M., Alves, O., Wheeler, M., 2013. Simulation and prediction of blocking in the Australian region and its influence on intra-seasonal rainfall in POAMA-2. Climate Dyn. 42 (11–12), 3271–3288. http://dx.doi.org/10.1007/s00382-013-1974-7.

Masson, D., Knutti, R., 2011. Climate model genealogy. Geophys. Res. Lett. 38, L08703. http://dx.doi.org/10.1029/2011GL046864.

Maxino, C.C., McAvaney, B.J., Pitman, A.J., Perkins, S.E., 2008. Ranking the AR4 climate models over the Murray-Darling Basin using simulated maximum temperature, minimum temperature and precipitation. Int. J. Climatol. 28, 1097–1112. http://dx.doi.org/10.1002/joc.1612.

Meehl, G.A., Covey, C., Taylor, K.E., Delworth, T., Stouffer, R.J., Latif, M., McAvaney, B., Mitchell, J.F.B., 2007. The WCRP CMIP3 multimodel dataset: a new era in climate change research. Bull. Am. Meteorol. Soc. 88, 1383–1394. http://dx.doi.org/10.1175/BAMS-88-9-1383.

Menke, W., Menke, J., 2012. Environmental Data Analysis with Matlab. Elsevier, Oxford, 263 pages.

Moberg, A., 2013. Comparisons of simulated and observed Northern Hemisphere temperature variations during the past millennium – selected lessons learned and problems encountered. Tellus B 65, 19921. http://dx.doi.org/10.3402/tellusb.v65i0.19921.

Nieto, S., Rodríguez-Puebla, C., 2006. Comparison of precipitation from observed data and general circulation models over the Iberian Peninsula. J. Climate 19, 4254–4275. http://dx.doi.org/10.1175/JCLI3859.1.

North, G.R., Cahalan, R.F., Coakley, J.A., 1981. Energy balance climate models. Rev. Geophys. Space Phys. 19, 91–121. http://dx.doi.org/10.1029/RG019i001p00091.

Otto, A., Otto, F., Boucher, O., Church, J., Hegerl, G., Forster, P.M., Gillett, N.P., Gregory, J., Johnson, G.C., Knutti, R., Lewis, N., Lohmann, U., Marotzke, J., Myhre, G., Shindell, D., Stevens, B., Allen, M.R., 2013. Energy budget constraints on climate response. Nat. Geosci. 6, 415–416. http://dx.doi.org/10.1038/ngeo1836.

Peacock, J.A., 1983. Two-dimensional goodness-of-fit testing in astronomy. Mon. Not. R. Astron. Soc. 202, 615–627. http://dx.doi.org/10.1093/mnras/202.3.615.

Pelly, J.L., Hoskins, B.J., 2003. How well does the ECMWF ensemble prediction system predict blocking? Q. J. R. Meteorol. Soc. 129, 1683–1702. http://dx.doi.org/10.1256/qj.01.173.

Pennell, C., Reichler, T., 2011. On the effective number of climate models. J. Climate 24, 2358–2367. http://dx.doi.org/10.1175/2010JCLI3814.1.

Perkins, S.E., Pitman, A.J., Holbrook, N.H., McAneney, J., 2007. Evaluation of the AR4 climate models' simulated maximum temperature, minimum temperature and precipitation over Australia using probability density functions. J. Climate 20, 4356–4376. http://dx.doi.org/10.1175/JCLI4253.1.

Radić, R., Clarke, G.K., 2011. Evaluation of IPCC Model's performance in simulating Late-Twentieth-Century climatologies and weather patterns over North America. J. Climate 24, 5257–5274. http://dx.doi.org/10.1175/JCLI-D-11-00011.1.

Randall, D.A., Wood, R.A., Bony, S., Colman, R., Fichefet, F., Fyfe, J., Kattsov, V., Pitman, A., Shukla, J., Srinivasan, J., Stouffer, R.J., Sumi, A., Taylor, K.E., 2007. Climate models and their evaluation. In: Solomon, S., Qin, D., Manning, M., Chen, Z., Marquis, M., Averyt, K., Tignor, M.M.B., Miller, H.L. (Eds.), Climate Change 2007, the Physical Science Basis, Contribution of Working Group I to the Fourth Assessment Report of the Intergovernmental Panel on Climate Change. Cambridge University Press, Cambridge, UK and New York, NY, USA, 996 pp.

Reichler, T., Kim, J.K., 2008. How well do coupled models simulate today's climate? Bull. Am. Meteorol. Soc. 89, 303–311. http://dx.doi.org/10.1175/BAMS-89-3-303.

Reifen, C., Toumi, R., 2009. Climate projections: past performance no guarantee of future skill? Geophys. Res. Lett. 36, L13704. http://dx.doi.org/10.1029/2009GL038082.

Russell, J., Stouffer, R.J., Dixon, K.W., 2006. Intercomparison of the Southern Ocean circulations in IPCC coupled model control simulations. J. Climate 19, 4560–4575. http://dx.doi.org/10.1175/JCLI3869.1.

Saha, S., Moorthi, S., Pan, H., Wu, X., Wang, J., Nadiga, S., Tripp, P., Kistler, R., Woollen, J., Behringer, D., Liu, H., Stokes, D., Grumbine, R., Gayno, G., Wang, J., Hou, Y., Chuang, H., Juang, H.H., Sela, J., Iredell, M., Treadon, R., Kleist, D., Van Delst, P., Keyser, D., Derber, J., Ek, M., Meng, J., Wei, H., Yang, R., Lord, S., Van Den Dool, H., Kumar, A., Wang, W., Long, C., Chelliah, M., Xue, Y., Huang, B., Schemm, J., Ebisuzaki, W., Lin, R., Xie, P., Chen, M., Zhou, S., Higgins, W., Zou, C., Liu, Q., Chen, Y., Han, Y., Cucurull, L., Reynolds, R.W., Rutledge, G., Goldberg, M., 2010. The NCEP climate forecast system reanalysis. Bull. Am. Meteorol. Soc. 91, 1015–1057. http://dx.doi.org/10.1175/2010BAMS3001.1.

Saha, S., Moorthi, S., Wu, X., Wang, J., Nadiga, S., Tripp, P., Behringer, D., Hou, Y., Chuang, H., Iredell, M., Ek, M., Meng, J., Yang, R., Mendez Malaquías, P., van den Dool, H., Zhang, Q., Wang, W., Chen, M., Becker, E., 2014. The NCEP climate forecast system version 2. J. Climate 27, 2185–2208. http://dx.doi.org/10.1175/JCLI-D-12-00823.1.

Santer, B.D., Mears, C., Doutriaux, C., Caldwell, P., Gleckler, P.J., Wigley, T.M.L., Solomon, S., Gillet, N.P., Ivanova, D., Karl, T.R., Lanzante, J.R., Meehl, G.A., Stott, P.A., Taylor, K.E., Thorne, P.W., Wehner, M.F., Wentz, F.J., 2011. Separating signal and noise in atmospheric temperature changes: the importance of timescale. J. Geophys. Res. 116, D22105. http://dx.doi.org/10.1029/2011JD016263.

Schwalm, C.R., Huntinzger, D.N., Michalak, A.M., Fisher, J.B., Kimball, J.S., Mueller, B., Zhang, K., Zhang, Y., 2013. Sensitivity of inferred climate model skill to evaluation decisions: a case study using CMIP5 evapotranspiration. Environ. Res. Lett. 8, 024028. http://dx.doi.org/10.1088/1748-9326/8/2/024028.

Scott, D.W., 1992. Multivariate Density Estimation: Theory, Practice and Visualization. John Wiley and Sons, New York. http://dx.doi.org/10.1002/9780470316849, 336 pp.

Silverman, B.W., 1986. Density Estimation for Statistics and Data Analysis. Chapman and Hall, London, 175 pp.

Stewart, D.E., Leyk, Z., 1994. Meschach: Matrix Computations in C. Centre for Mathematics and its Applications, the Australian National University, Canberra, Australia.

Stocker, T.F., Qin, D., Plattner, G.K., Tignor, M., Allen, S.K., Boschung, J., Nauels, A., Xia, Y., Bex, V., Midgley, P.M. (Eds.), 2013. IPCC, 2013: Climate Change 2013: the Physical Science Basis. Contribution of Working Group I to the Fifth Assessment Report of the Intergovernmental Panel on Climate Change. Cambridge University Press, Cambridge, United Kingdom and New York, NY, USA, 1535 pp.

Sundberg, R., Moberg, A., Hind, A., 2012. Statistical framework for evaluation of climate model simulations by use of climate proxy data from the last millennium – Part 1: Theory. Climate Past 8, 1339–1353. http://dx.doi.org/10.5194/cp-8-1339-2012.

Taylor, K.E., 2001. Summarizing multiple aspects of model performance in a single diagram. J. Geophys. Res. 106, 7183–7192. http://dx.doi.org/10.1029/2000JD900719.

Taylor, K.E., Stouffer, R.J., Meehl, G.A., 2012. An overview of CMIP5 and the experiment design. Bull. Am. Meteorol. Soc. 93, 485–498. http://dx.doi.org/10.1175/BAMS-D-11-00094.1.

Ulbrich, U., Pinto, J.G., Kupfer, H., Leckebusch, G., Spangehl, T., Reyers, M., 2008. Changing Northern Hemisphere storm tracks in an ensemble of IPCC climate change simulations. J. Climate 21, 1669–1679.

Uppala, S.M., Kållberg, P.W., Simmons, A.J., Andrae, U., da Costa Bechtold, V., Fiorino, M., Gibson, J.K., Haseler, J., Hernandez, A., Kelly, G.A., Li, X., Onogi, K., Saarinen, S., Sokka, N., Allan, R.P., Andersson, E., Arpe, K., Balmaseda, M.A., Beljaars, A.C.M., van de Berg, L., Bidlot, J., Bormann, N., Caires, S., Chevallier, F., Dethof, A., Dragosavac, M., Fisher, M., Fuentes, M., Hagemann, S., Hólm, E., Hoskins, B.J., Isaksen, L., Janssen, P.A.E.M., Jenne, R., McNally, A.P., Mahfouf, J.-F., Morcrette, J.-J., Rayner, N.A., Saunders, R.W., Simon, P., Sterl, A., Trenberth, K.E., Untch, A., Vasiljevic, D., Viterbo, P., Woollen, J., 2005. The ERA-40 re-analysis. Q. J. R. Meteorol. Soc. 131, 2961–3012. http://dx.doi.org/10.1256/qj.04.176.

Vera, C., Silvestri, G., Liebmann, B., Gonzalez, P., 2006. Precipitation variability in South America from IPCC-AR4 models. Part II: influence of circulation leading patterns. In: Proceedings of 8 ICSHMO, Foz Do Iguaçu, Brazil, INPE, April 24–28, pp. 477–485.

von Storch, H., Zwiers, F., 1999. Statistical Analysis in Climate Research. Cambridge University Press, Cambridge, 484 pp.

Walsh, J.E., Chapman, W.L., Romanovsky, V., Christensen, J.H., Stendel, M., 2008. Global climate model performance over Alaska and Greenland. J. Climate 21, 6156–6174. http://dx.doi.org/10.1175/2008JCLI2163.1.

Wang, M., Overland, J.E., Kattsov, V., Walsh, J.E., Zhang, X., Pavlova, T., 2007. Intrinsic versus forced variation in coupled climate model simulations over the Arctic during the twentieth Century. J. Climate 20, 1093–1107. http://dx.doi.org/10.1175/JCLI4043.1.

Washington, W.M., Parkinson, C.L., 2005. An Introduction to Three-dimensional Climate Modelling, second ed. University Science Books, Sausalito, CA. 353 pages.

Wilks, D.S., 2006. Statistical Methods in the Atmospheric Sciences, second ed. Academic Press, Burlington. 627 pp.

Ylhäisi, J.S., Räisänen, J., 2013. Twenty-first century changes in daily temperature variability in CMIP3 climate models. Int. J. Climatol. http://dx.doi.org/10.1002/joc.3773 (in press).

RESUMEN DE TESIS: Contribuciones al uso eficiente de coprocesadores de propósito general: La estimación de densidades basada en núcleos como caso de estudio

Unai Lopez Novoa
Directores: Alexander Mendiburu y Jose Miguel

9 de mayo de 2015

1. Introducción

La computación de alto rendimiento es la práctica de utilizar múltiples nodos de computo para el procesamiento de aplicaciones paralelas. Estas aplicaciones distribuyen su carga de trabajo entre los distintos procesadores de los nodos de cómputo, con el objetivo de reducir los tiempos de ejecución. Tradicionalmente los nodos de cómputo han sido sistemas homogéneos, donde todos los procesadores eran del mismo tipo, pero desde hace aproximadamente 10 años estos nodos están compuestos por procesadores de distintos tipos, conviertiéndose así en sistemas heterogéneos.

En la actualidad los sistemas heterogéneos están comúnmente formados por uno o varios procesadores multi-núcleo y uno o varios coprocesadores ó aceleradores de cómputo. Los procesadores multi-nucleo actuales cuentan con características para tratar de manera eficiente cualquier tipo de código serie (predictores de salto, módulos para ejecución fuera de orden,...) pero no proveen un rendimiento óptimo con aplicaciones que escalan a miles o millones de hilos. En cambio, los aceleradores son sistemas con arquitecturas pensadas para ejecutar tareas concretas, generalmente con alto grado de paralelismo, de manera más eficiente que los procesadores multi-nucleo. En los entornos de alto rendimiento actuales los aceleradores más comunes son las unidades de procesamiento gráfico (en inglés *Graphics Processing Units* o GPUs) y los sistemas *many-core* como el Intel Xeon Phi.

Las GPUs se empiezan a utilizar en entornos de alto rendimiento de manera extensiva alrededor de 2007, cuando el fabricante NVIDIA publica el entorno CUDA, que permite desarrollar aplicaciones paralelas para ser ejecutadas en unidades gráficas. Una GPU es un procesador con cientos de núcleos de capacidades reducidas que dan excelentes resultados en aplicaciones paralelas que lanzan miles de hilos ligeros. Las más recientes, diseñadas para altas prestaciones, pueden ofrecer una capacidad teórica de cómputo de hasta 1.8 TFLOP/s (1.8 billones de operaciones en coma flotante por segundo) en doble precisión. Por otra parte el Xeon Phi es un acelerador *many-core* comercializado por Intel. Los núcleos del Xeon Phi son una versión simplificada de los núcleos Intel multi-nucleo con consumo energético reducido para poder aunar hasta 61 en un sólo acelerador. El actual Xeon Phi proporciona hasta 1 TFLOP/s de capacidad de cómputo en doble precisión, y uno de sus puntos fuertes es la variedad de entornos de desarrollo con los que es compatible, tales como OpenMP, OpenCL, Cilk o MPI.

Aunque los actuales aceleradores proporcionan una gran capacidad teórica de cómputo (entre

4 y 8 veces más que un procesador multi-nucleo actual), uno de los grandes retos es extraer todo el rendimiento que prometen. Así como los multi-nucleo actuales están preparados con características que les permiten ejecutar código muy diverso con un rendimiento aceptable, los aceleradores carecen de ellas. A día de hoy, aplicaciones que requieren cálculos de álgebra lineal o que hacen uso extenso de FFTs, han conseguido extraer un rendimiento cercano al pico teórico de estos aceleradores.

En esta tesis se analizan algunos de los problemas que surgen al tratar de utilizar de manera eficiente coprocesadores de propósito general. Para ello, se utiliza como caso de estudio la técnica estadística *estimación de densidades basada en núcleos* (en inglés Kernel Density Estimation o KDE), la cual es usada comúnmente en aplicaciones científicas de altas prestaciones. La motivación principal de la tesis es proporcionar una pieza de literatura útil que ayude a los desarrolladores en el uso eficiente de coprocesadores modernos, incluyendo diferentes evaluaciones de rendimiento en distintos escenarios.

En primer lugar se presenta un análisis de la literatura sobre técnicas de modelado de rendimiento en coprocesadores de propósito general. En segundo lugar, se presenta el diseño de un nuevo algoritmo para calcular KDE llamado S-KDE. Después presentamos la implementación y evaluación del rendimiento de S-KDE, primero en procesadores y coprocesadores multi-núcleo, y luego en una gama más amplia de coprocesadores de propósito general. Por último, presentamos una aplicación de S-KDE en el dominio de la climatología.

2. Estudio de las técnicas de modelado de rendimiento para aceleradores

El modelado de rendimiento en el área de la computación de altas prestaciones se refiere a representar matemáticamente el comportamiento de un sistema o aplicación para su mejor comprensión y/u optimización. Este proceso requiere recoger información sobre el sistema y aplicación, generalmente en forma de contadores, para crear un modelo de rendimiento que genere una estimación, p.e., del tiempo de ejecución. Las técnicas para crear el modelo pueden ser variadas, como diseñar el modelo matemáticamente desde cero, basarse en uno existente, o crear uno mediante técnicas de aprendizaje automático.

Desde que aceleradores como las GPUs o el Xeon Phi se popularizaron en la computación de altas prestaciones, ha habido múltiples contribuciones que modelan algún aspecto del rendimiento en estos dispositivos, generando estimaciones de tiempos de ejecución o métricas sobre el uso de los dispositivos. Sin embargo, debido a la novedad del área, no hay ningún criterio establecido a la hora de evaluar las propuestas, ni está clara su forma de uso o aplicabilidad en algunos casos.

Estos modelos se pueden clasificar según diferentes criterios. Nuestra propuesta es una clasificación basada en la información generada por el modelo. Hemos identificado modelos diseñados para:

- Predecir el tiempo de ejecución de una aplicación en una plataforma determinada
- Identificar los cuellos de botella en la aplicación (en cuanto a rendimiento) y proponer modificaciones de código para evitarlos
- Proporcionar estimaciones de consumo de energía
- Proporcionar información detallada del uso de recursos, basado en simulación

En la revisión realizada, hemos recopilado 29 modelos de rendimiento, los hemos clasificado en base a esta taxonomía y para cada uno extrajimos las siguientes características:

- Requisitos de entrada, como ejecutar *micro-Benchmarks* o analizar códigos intermedios
- Limitaciones como, por ejemplo, falta de detalle en el modelado de memoria o inexactitudes al predecir un tipo particular de aplicación
- Puntos destacados y beneficios más allá de otros modelos, como facilidad de uso o la posibilidad de ampliación

Tras realizar el estudio pudimos identificar los modelos más relevantes para cada escenario, y llegar a la conclusión de que todos los modelos y herramientas disponibles en la actualidad tienen limitaciones, pero forman la base sobre la que serán construidas mejores herramientas. El estudio nos permitió de la misma forma, identificar algunas limitaciones generales de los modelos, como la inexistencia de un modelo preciso válido para un amplio conjunto de arquitecturas, o el sesgo general hacia la plataforma CUDA de NVIDIA.

3. S-KDE: Un algoritmo eficiente para la estimación de densidades basada en núcleos

La estimación de densidades basada en núcleos (o KDE del inglés *Kernel Density Estimation*) es una técnica aplicada desde los años 80 como técnica de estimación de densidad en diferentes problemas científicos. KDE crea estimaciones suaves, en comparación con otras técnicas como el histograma. Intuitivamente, dado un espacio de evaluación y un conjunto de observaciones, KDE coloca un "bulto" sobre cada observación y agrega el valor de esos bultos para crear la función de densidad de probabilidad.

La estimación resultante es continua pero la mayoría de implementaciones de KDE la proporcionan como un conjunto de valores discretos. El usuario define los límites del espacio de evaluación y la separación entre los puntos donde KDE evaluará la función de densidad de probabilidad. El resultado es una malla de evaluación discreta como una serie de puntos de evaluación.

La forma más común para computar KDE es recorrer cada punto de evaluación de la malla, y calcular para cada uno de ellos la densidad asociada a cada observación. Este enfoque es completamente paralelizable usando un paralelismo a nivel de datos, pero en muchos casos implica un gran numero de cálculos inútiles. Esto es debido a que una observación afecta solamente a una porción de la malla de evaluación, es decir, a un conjunto de puntos de evaluación a su alrededor.

El tamaño del área de influencia de una observación depende de la elección del núcleo (la forma del "bulto") y otros parámetros. Nuestra propuesta comienza por definir el área de influencia de una observación como un conjunto de puntos de evaluación, y después recorrer sólo los puntos de evaluación afectados por cada muestra. La complejidad computacional del enfoque tradicional es $O(k_d mn)$, siendo k_d una constante de dimensionalidad, m el número de puntos de evaluación (el tamaño de la malla de evaluación) y n el número de observaciones. La complejidad de nuestro enfoque es $O(k_d np)$, siendo k_d una constante de dimensionalidad, n el número de observaciones y p el número de puntos de evaluación en el área de influencia de una observación. La ventaja de nuestro enfoque se basa en que p es generalmente mucho más pequeño que m. Definimos nuestra propuesta como S-KDE.

La literatura sobre KDE incluye diferentes funciones núcleo a usar. Dependiendo de la elección, la técnica para definir el área de influencia de una observación será diferente. En este trabajo usamos un núcleo Epanechnikov (de forma elíptica) y una técnica basada en los valores propios de la matriz de covarianza del conjunto de observaciones. Esto nos permite calcular un cortono

de forma rectangular que delimita el área de influencia de una observación, que llamaremos *recuadro de delimitación* de una observación. Además, proponemos una técnica llamada Chop & Crop que ajusta el tamaño del recuadro de delimitación mediante la eliminación de puntos de evaluación no pertenecientes a la zona de influencia del núcleo. Esta técnica funciona reduciendo primero el recuadro d-dimensional a un conjunto de planos bidimensionales, y luego ajustando cada plano al mínimo rectángulo.

Podemos proporcionar algunos números para ilustrar la eficacia de S-KDE. Usaremos como ejemplo un conjunto de 1 millon de observaciones y un espacio de evaluación con 194 millones de puntos de evaluación. Utilizando el enfoque tradicional KDE que atraviesa todos los puntos de la malla de evaluación, se requerirían $1,94 * 10^{14}$ operaciones de par observación- punto de evaluación. En contraste, un recuadro de delimitación 3D alrededor de cada observación en el mencionado escenario contiene un promedio de 102.461 puntos, y utilizando el enfoque S-KDE se requerirían $1,02 * 10^{11}$ cálculos en total. Además, si aplicamos la técnica de Chop & Crop, el número de puntos de evaluación por cada recuadro de delimitación se reduce a 53.511 de promedio, y el resultante número total de cálculos descendería hasta $5{,}35 * 10^{10}$.

4. S-KDE en procesadores y coprocesadores multi-núcleo

El algoritmo S-KDE descrito en la sección anterior tiene una complejidad computacional mucho menor que el enfoque tradicional para KDE. Sin embargo, evaluar grandes conjuntos de datos y/o densos espacios de evaluación puede ser computacionalmente costoso. Por ello, nuestro siguiente paso consiste en el diseño e implementación de S-KDE como un programa paralelo, enfocado a procesadores y coprocesadores de múltiples núcleos. Hemos desarrollado nuestro código en ANSI-C, y la paralelización se realiza haciendo uso de OpenMP y las directivas que proporciona el compilador Intel.

En nuestra implementación, cada hilo calcula la influencia de un conjunto de observaciones sobre el espacio de evaluación. Para cada muestra, el hilo primero ajusta el recuadro de delimitación. Si el espacio de evaluación es de dimensionalidad tres o superior, el Chop & Crop es recursivamente aplicado para reducir el cómputo hasta una sucesión de planos bidimensionales. Finalmente, se computa la densidad que cada observación ejerce en cada plano 2D.

Uno de los inconvenientes de realizar los cálculos en base a las observaciones es la contención de memoria que podría aparecer cuando dos o más hilos diferentes, computando observaciones cuyas áreas de influencia se solapan, tienen que agregar valores parciales de densidad en la misma posición de memoria. Para reducir este efecto, cada hilo calcula cada fila del plano en su memoria privada, y lo añade a la memoria principal mediante una primitiva de escritura atómica de OpenMP. De esta manera, aseguramos la consistencia de los datos. Hay un precio a pagar, sin embargo: las operaciones atómicas provocan una degradación de rendimiento debido a la serialización de las escrituras de memoria.

Hemos ejecutado nuestro código S-KDE en dos plataformas de hardware diferentes: una CPU Intel Core i7 3820 (4 núcleos @ 3.60 Ghz) y un coprocesador Intel Xeon Phi 3120A (57 núcleos @ 1.1 Ghz), con conjuntos de datos de diversa dimensionalidad y diferentes tamaños de evaluación. La evaluación se lleva a cabo variando el espaciado entre los puntos de evaluación de la malla con el fin de aumentar o reducir el tamaño del problema.

En una comparación inicial, medimos los tiempos de ejecución de nuestra implementación S-KDE contra dos implementaciones públicas de KDE: *ks*, una librería del software estadístico R, enlazada con la biblioteca Intel MKL para realizar cómputo multi-núcleo y *GPUML*, una implementación basada en GPUs. Las pruebas de esta última se ejecutaron en una GPU NVIDIA GTX 650. Nuestra implementación de S-KDE en el Xeon Phi obtuvo valores de *speed-up* mayores

de 400x comparado con el resto de implementaciones en todos los escenarios.

Adicionalmente, realizamos distintas pruebas para evaluar aspectos más concretos de la implementación de S-KDE. En una de las pruebas ejecutamos S-KDE con Chop & Crop habilitado e inhabilitado para evaluar las mejoras de rendimiento derivadas de la utilización de esta técnica. Medimos que S-KDE funciona un 60,4 % más rápido en el Core i7 y un 49,2 % en el Xeon Phi usando Chop & Crop, debido a la eliminación de cálculos inútiles (y los accesos a la memoria correspondientes). A continuación, tratamos de encontrar los cuellos de botella del código. Se realizó una disección de los tiempos de ejecución para medir el peso de las diferentes etapas del código. Encontramos que la parte de escritura atómica en memoria es el mayor cuello de botella.

Con todos los experimentos concluimos lo siguiente: (1) la implementación actual de S-KDE, en comparación con implementaciones públicas recientes de KDE, proporciona el mejor rendimiento, incluso cuando se ejecuta en un procesador i7 modesto, y (2) aunque el rendimiento general del código es satisfactorio, hemos detectado varios cuellos de botella (siendo el principal la contención de memoria al escribir) que requieren una mayor exploración.

5. S-KDE en coprocesadores de propósito general

El algoritmo S-KDE presenta una significante mejora en el cómputo de KDE, y hemos validado cómo su implementación en procesadores y coprocesadores multi-núcleos es eficaz comparada con otras implementaciones del estado del arte. Sin embargo, esta implementación sólo es válida para un conjunto limitado de coprocesadores. El espectro de coprocesadores en el área de la computación de altas prestaciones es muy amplio en la actualidad. Por lo tanto, sentimos que nuestro enfoque S-KDE debería ser probado en un conjunto más amplio de aceleradores.

Debido al modelo de procesamientos masivo de datos para el que los aceleradores están pensados, portar un código requiere generalmente un importante rediseño partiendo desde el serie. En nuestro caso, hemos partido de la versión pensada para procesadores multi-núcleo y hemos divido el código en diferentes etapas. Con el fin de poder ejecutar nuestro código en el mayor número de dispositivos posible, usamos OpenCL como entorno de programación. Además, la versión para aceleradores ha requerido incluir ciertas operaciones auxiliares que no estaban presentes en la versión multi-núcleo.

El código OpenCL ha sido evaluado en tres aceleradores modernos: una GPU AMD, una GPU NVIDIA y un coprocesador Intel Xeon Phi. Además, nuestro objetivo ha sido realizar el análisis utilizando herramientas independientes de la plataforma, es decir, prescindir de herramientas especificas del fabricante. Para ello, utilizamos algunos de los modelos de rendimiento y herramientas presentadas en nuestro estudio inicial.

En esta evaluación hemos seguido un enfoque de análisis descendente, partiendo de métricas de rendimiento global para ir entrando poco a poco en detalles de menor nivel. Las pruebas iniciales fueron una comparación de tiempos globales de ejecución contra una implementación serie de S-KDE, donde el código OpenCL obtuvo niveles de aceleración en el rango de 3.6x a 5.1x para el más complejo de los problemas analizados. Después de las pruebas iniciales, nos centramos en aspectos más específicos. Utilizando modelos como *Roofline* y suites de benchmarks como SHOC, pudimos encontrar los cuellos de botella del código en los diferentes dispositivos.

Llegamos a la conclusión de que nuestro código OpenCL S-KDE alcanza una eficiencia aceptable en los aceleradores probados, dadas las restricciones del algoritmo: tiene una complejidad computacional (relativamente) baja, pero también hace un uso extensivo de la memoria. Por ello, aunque no encaja de manera simple en el modelo de procesamiento de los aceleradores, nuestra implementación llega cerca de los límites que el hardware permite.

6. Una nueva metodología para evaluar modelos ambientales basada en S-KDE

En esta parte final de la tesis, aplicamos nuestro algoritmo S-KDE a una operación costosa en el campo de la ciencias ambientales: la evaluación de modelos climáticos.

Los modelos climáticos son representaciones matemáticas de un sistema climático, basados en los principios biológicos, físicos y químicos. Por lo general, incluyen ecuaciones complejas que representan estas leyes, y se resuelven numéricamente. Los modelos climáticos proporcionan resultados discretos en el espacio y el tiempo, y su precisión dependerá de la resolución del modelo. Con el fin de medir la confianza de los resultados de un modelo climático, dichos resultados deben ser evaluados contra diferentes observaciones o mediciones reales. La razón tras esta hipótesis es que los modelos que mejor simulen el clima actual, serán los mejores en simular el clima futuro.

Usando S-KDE, proponemos una metodología novedosa para la evaluación de los modelos climáticos. Nuestra metodología se basa en construir funciones de probabilidad de densidad con los datos a escala diaria que producen los modelos climáticos, y evaluar en una sola vez varias variables producidas por un modelo climático, como la temperatura o la presión del aire. Parte de este trabajo se basa en la extensión a multiples dimensiones del índice de Perkins, presentado en su trabajo como unidimensional: sólo puede evaluar la calidad de una variable producida por un modelo climatico a la vez. Sin embargo, en algunos escenarios es útil poder evaluar varias variables en un solo paso y nuestro modelo soluciona este vacío en la literatura.

Nuestra propuesta require un uso iterativo de KDE, para crear multiples funciones de distribución de probabilidad de un conjunto de observaciones con diferentes parámetros. Este proceso toma la mayor parte del tiempo computacional y S-KDE es esencial para reducir los tiempos totales de ejecución. Sin S-KDE la evaluación de un modelo climatico contra diferentes conjuntos de observaciones no sería posible en un tiempo razonable.

La metodología propuesta se ha aplicado a dos casos de estudios diferentes. El primero de ellos corresponde a una aplicación real de evaluación de modelos climáticos. El segundo consiste en evaluar la precisión de un nuevo análisis de modelos acoplados para la reproducción a escala global de la temperatura superficial del mar y la altura de la superficie del mar.