J Sign Process Syst (2011) 62:43–63
DOI 10.1007/s11265-008-0337-9

Accelerating Machine-Learning Algorithms on FPGAs using Pattern-Based Decomposition

Karthik Nagarajan · Brian Holland · Alan D. George · K. Clint Slatton · Herman Lam

Received: 28 April 2008 / Revised: 21 November 2008 / Accepted: 23 December 2008 / Published online: 16 January 2009

Abstract Machine-learning algorithms are employed in a wide variety of applications to extract useful information from data sets, and many are known to suffer from super-linear increases in computational time with increasing data size and number of signals being processed (data dimension). Certain principal machine-learning algorithms are commonly found embedded in larger detection, estimation, or classification operations. Three such principal algorithms are the Parzen window-based, non-parametric estimation of Probability Density Functions (PDFs), K-means clustering and correlation. Because they form an integral part of numerous machine-learning applications, fast and efficient execution of these algorithms is extremely desirable. FPGA-based reconfigurable computing (RC) has been successfully used to accelerate computationally intensive problems in a wide variety of scientific domains to achieve speedup over traditional software implementations. However, this potential benefit is quite often not fully realized because creating efficient FPGA designs is generally carried out in a laborious, case-specific manner requiring a great amount of redundant time and effort. In this paper, an approach using pattern-based decomposition for algorithm acceleration on FPGAs is proposed that offers significant increases in productivity via design reusability. Using this approach, we design, analyze, and implement a multi-dimensional PDF estimation algorithm using Gaussian kernels on FPGAs. First, the algorithm's amenability to a hardware paradigm and expected speedups are predicted. After implementation, actual speedup and performance metrics are compared to the predictions, showing speedup on the order of 20× over a 3.2 GHz processor. Multi-core architectures are developed to further improve performance by scaling the design. Portability of the hardware design across multiple FPGA platforms is also analyzed. After implementing the PDF algorithm, the value of pattern-based decomposition to support reuse is demonstrated by rapid development of the K-means and correlation algorithms.

Keywords FPGA · Design patterns · Machine learning · Pattern recognition · Hardware acceleration · Performance prediction

K. Nagarajan (✉) · B. Holland · A. D. George · K. C. Slatton · H. Lam
NSF Center for High-Performance Reconfigurable Computing (CHREC), Electrical and Computer Engineering Department, University of Florida,
Gainesville, FL 32611, USA
e-mail: nagarajan@chrec.org

B. Holland
e-mail: holland@chrec.org

A. D. George
e-mail: george@chrec.org

K. C. Slatton
e-mail: slatton@chrec.org

H. Lam
e-mail: lam@chrec.org

1 Introduction

Machine learning is a general term that has been used to refer to a large and diverse set of methods for extracting patterns from data. In general, it encompasses portions of statistical and adaptive signal processing, probabilistic decision theory, such as Bayes Rule, and meta-heuristic strategies, such as the genetic algorithm. As computational capacity has increased over the past few decades, machine-learning algorithms have become widely used for problems such as detection, estimation, and classification, involving diverse data types, including time series, image, and video

(spatio-temporal) data [1]. Raw audio, image, and video data do not explicitly contain high-level information and thus generally require an abstraction process to extract useful information from the data. This process is often accomplished via a sequence of data segmentation algorithms [1, 2]. Clustering algorithms are the most common type of algorithms used for segmenting data points into groups and employ various measures of similarity. Probabilistic algorithms can then be employed to represent the segmented data as Probability Density Functions (PDFs) rather than single data points (e.g., cluster centers). Such data representations are essential in applications involving decision-making based on probabilistic reasoning [3–5]. Alternatively, certain multimedia analysis tools use statistical methods to model relationships between segmented datasets as a function of the separation between them (dissimilarity). These methods are useful for applications such as video content retrieval and indexing [6], video segmentation, and automatic speech recognition systems [7, 8] that benefit from quantitative measures of the statistical relationships between features.

Many of the machine-learning algorithms mentioned above are primarily based on a data-driven approach wherein the computational burden is heavily impacted by the volume of data being processed. The Parzen-based non-parametric PDF estimation algorithm is one such algorithm that is computationally intensive, yet necessary for many applications in pattern recognition, such as fingerprint recognition, Bayesian classification, feature extraction [5], bioinformatics [3], networking [9], stock market prediction [10, 11], and image processing [4]. The high computational burden arises because most problems involve large volumes of data and require complex computations to be performed in a high-dimensional space. To mitigate this problem, application researchers have investigated ways to approximate the solution by either solving the problem under a reduced dimensional space or providing alternative methods like the Fast Gauss Transform (FGT) [12]. The FGT addresses the dimensionality issue to a certain extent by reducing the number of computations in each dimension. Irrespective of the approximation method used, the computation involved still grows exponentially with increasing dimensionality. Thus, fast execution of the PDF algorithm is important for solving high-dimensional problems without structural (algorithmic) approximations and within reasonable timeframes. Further, according to Amdahl's law [13], significant improvements to the overall execution time of the multimedia analysis tool are obtained only when significant portions of it are improved. In other words, overall efficiency gains are most likely to occur when the principal (or most often used) portions of the machine learning tasks are improved.

An emerging method for accelerating these algorithms involves the use of reconfigurable computing (RC) based on Field-Programmable Gate Array (FPGA) technology. FPGAs lie between general-purpose processors and ASICs (application-specific ICs) on the spectrum of processing elements in that they are highly flexible (like processors) and also have potential for high performance (hardware acceleration). Commonly used, deterministic benchmarking cores, such as Fast Fourier Transforms (FFTs) [14], convolution, Lower-Upper triangular matrix Decomposition (LUD) [15], and Basic Linear Algebra Subprograms (BLAS), have been effectively implemented on FPGAs by virtue of their inherent multi-level functional and data parallelism. In addition to these benchmarking cores, significant performance gains have been achieved with FPGAs by selecting, developing, and testing diverse applications from various fields in signal and image processing.

However, developing FPGA designs is often time-consuming and not every algorithm is amenable to hardware acceleration. Preliminary analyses should be performed to predict the algorithm's amenability to a hardware paradigm before undertaking a lengthy development process. In contrast to the general computing domain, there is a relatively modest variety of FPGA-based hardware platforms from which a designer can choose. These platforms primarily vary in the type of FPGA they house and the manner in which data is communicated between the FPGA and host processor; each platform can vary with respect to the number of constraints on the application design and performance. For example, decisions regarding bit precision, design architecture, memory utilization, and storage depend on the FPGA's resource availability and the size of the hardware design. The presence of dedicated arithmetic blocks also dictates the implementation options for certain mathematical instructions. Thus, it is best to determine the expected performance attainable at an early stage when migrating an algorithm to an FPGA platform by employing simple performance prediction tools.

Another barrier to successfully using FPGA-based RC technology for algorithm acceleration is the prohibitive time and effort that is required to develop the necessary FPGA core designs in a custom, case-specific manner. A more desirable approach would be to recognize and exploit common patterns in terms of computational structure and data flow across a variety of algorithms and prove that those design patterns are indeed suitable for use in other FPGA implementations. The primary goal is hence to develop a concise set of design patterns that can be formally used to represent the decomposition of an algorithm for parallel implementation. If the hardware design of an algorithm can be represented at a high level graphically via composite patterns (mixture of computation and communication patterns), then other algorithms having

a similar high-level representation can reuse the existing design with only minimal customization. Such a high-level description is also desirable to aid inexperienced FPGA designers (e.g., domain scientists) so they can easily comprehend and reuse existing hardware designs while developing new applications. Pattern-based descriptions of designs could be used to effectively disseminate and share hardware cores as well. Such an approach can hence lead to a significant increase in productivity via design reusability. Several important machine-learning algorithms, such as PDF estimation using Parzen windows, K-means clustering, correlation, and information-theoretic measures, have very similar underlying algorithmic and dataflow structures and thus could potentially benefit from such an approach. In general, the similarity in dataflow in many machine-learning algorithms is due to the fact that application data (input to the learning system) is compared with several representative values (parameters of the learning system) to perform inference. In addition to being structurally similar, most of these algorithms are often applied to very large volumes of data and impose difficult constraints on memory.

This paper describes the pattern-based decomposition, analysis, and implementation of a scalable and portable architecture for multi-dimensional, non-parametric PDF estimation using Gaussian kernels on FPGAs. The work also showcases the rapid development of two other algorithms, in particular K-means clustering and correlation, by reusing the structural and communication architecture developed for PDF estimation due to similarities in the composite patterns describing the respective algorithm decompositions.

The remainder of this paper is organized as follows. Section 2 discusses background on some related signal processing algorithms that are amenable for implementation on FPGAs, prior work on machine-learning for multimedia analysis, and discussions on relevant performance prediction methods and design patterns for Reconfigurable Computing (RC) used in this paper. The case-study algorithms and their underlying similarities are discussed in Section 3. An overview of design patterns and the motivations for pattern-based algorithm decomposition are given in Section 4. Preliminary analyses, the design methodology, and the basic architecture for PDF estimation are presented in Section 5 with discussions on results and performance analysis. Section 6 illustrates the concept of architecture reuse with K-means clustering and correlation used as case-study algorithms. We conclude the work in Section 7 by reviewing insights gained and potential for future work.

2 Background

Previous works have discussed the importance of non-parametric PDF estimation, K-means clustering, and correlation in multimedia analysis. The authors in [7] employ cross-modal correlations for clustering features in an image-audio dataset. Clustering and feature extraction methods are applied for video content retrieval and analysis in [16]. A novel video segmentation technique was presented in [2] based on correlating audio and video data. A simplified probability-based methodology using histograms was proposed in [6] for video analysis. Hence, a large number of applications can benefit from fast computations of PDF estimation, K-means clustering and correlation, especially in time-critical missions. Previous works have also discussed the feasibility of targeting some structurally similar algorithms for FPGAs. In [17], the authors use a Gaussian kernel estimator as one of their case studies in presenting a MATLAB-to-VHDL application mapper. It is a subset of the algorithm considered here, in that our work employs Gaussian kernels to estimate general PDFs and is hence more computationally intensive. In [18], the authors present simpler algorithmic variants of the K-means algorithm by considering Manhattan and Max distances instead of Euclidean distance for an efficient hardware implementation. Significant acceleration of the algorithm with acceptable accuracy was achieved in clustering a sample dataset. This work is different from [18] in that Euclidean distance is used for implementing K-means (which is the preferred method in most machine-learning applications) and, moreover, the primary motivation is to accelerate a set of machine-learning algorithms used in different stages of multimedia data processing by reusing hardware designs leading to shorter design times. Previous work in parallel processing for large data volumes is also relevant to this work. An object-recognition application was developed in [19] by traversing huge volumes of data in a parallel fashion. The paper showcased how a data-parallel programming model can be used to solve a general set of data-intensive algorithms. In many of these examples, the applications could achieve significant performance gains if there was a way to rapidly compute multi-dimensional PDFs. PDF estimation enables the computation of metrics such as noise, error rates, and uncertainties that are very important for researchers in the application domain [3–5, 20].

The authors in [21] and [22] had developed several signal-processing applications on FPGAs in order to obtain speedup and discussed tradeoffs in solution accuracy and resource utilization. However, these papers only predict performance factors in a relatively limited setting. Various performance prediction tools have been proposed [23, 24] that base their techniques on parameterizing the algorithm and target FPGA platform. Algorithms are decomposed and analyzed to determine their total size (in terms of resources) and computational density. Computational platforms are primarily characterized by their memory structure and capacity and interconnect bandwidth. In particular, the RC

Amenability Test (RAT) presented in [23] is a simple methodology that suggests a step-by-step procedure to predict the performance, in terms of speedup, of a specific design for an application on a specific platform before coding begins. This pre-implementation test also helps the designer to understand the strengths and weaknesses of a particular algorithm towards parallel implementation. Communication and computation parameters in RAT are quantified based upon algorithm and FPGA platform characteristics. An estimate of performance in terms of execution time in hardware is then derived from the analytical models embedded in RAT. Comparing the hardware prediction against a known execution time for a software baseline leads to the overall speedup estimation. In this work, we use RAT to predict performance of an algorithm on multiple FPGA platforms.

Building a good reconfigurable design requires skill and effort, and is often accompanied by a steep learning curve for designers without prior FPGA design experience. In [25], the authors emphasize the importance of identifying and cataloging an exhaustive list of design patterns to solve recurring challenges in reconfigurable computing and increase productivity. In this work, we validate the utility of design patterns by identifying and classifying primitive patterns relevant to this work. The patterns are then quantified and applied for algorithm decomposition and performance prediction using RAT.

3 Machine-learning Algorithms for Segmentation

Knowledge of the algorithm's data flow and computational complexity is essential in order to make strategic decisions during design development. In this section, we provide an overview of the algorithms used in the case studies in this work and investigate the commonalities in their structures.

3.1 PDF Estimation

The common parametric forms of PDFs (e.g., Gaussian, Binomial, Rayleigh distributions) represent mathematical idealizations and, as such, are often not well matched to densities encountered in practice. In particular, most of these classical parametric densities are unimodal (having a single local maximum), whereas many practical problems involve multimodal densities. Furthermore, because of correlations among the data features, it is unusual for high-dimensional density functions to be accurately represented as a simple product of one-dimensional densities, the latter of which are significantly easier to compute. Thus, an estimate of the joint (multi-dimensional), non-parametric PDF (i.e. a PDF that assumes no particular functional form) is often required. The computational complexity of the Parzen window-based PDF estimation algorithm is $O(Nn^d)$, where N is the total number of data points, n is the number of discrete points at which the PDF along a dimension is estimated (i.e. bins) and d is the number of dimensions. Table 1 illustrates the computational burden incurred in terms of time elapsed (where code is executed on a 3.2 GHz single-core Xeon processor) in computing PDFs of increasing dimension (number of signals considered) using the Parzen window technique and Gaussian kernels. The entries for the estimated elapsed time were made by scaling the number of computations exponentially corresponding to increasing dimensions. The total number of data samples processed was set at 204,800 (N) and the support size on each dimension, n, was set as 256. Mathematically, the probability that point i falls in a d-dimensional space is given by,

$$p(i) = \frac{1}{n_1 \ldots n_d} \sum_{j_1=1}^{n_1} \ldots \sum_{j_d=1}^{n_d} \varphi\left(x_i, x^{j_1}, h\right) \ldots \varphi\left(y_i, y^{j_d}, h\right) \tag{1}$$

where h is the bin size, which acts as a tuning parameter for resolution of the PDF estimate, the set $(x^j,...y^j)$ represents the d subsets of bin centers at which the PDF is estimated, and the set $(x_i,...,y_i)$ represents the i^{th} input data point in a d-dimensional space, where i ranges from 1 to N.

The kernel function φ could be as simple as a histogram function, a more general rectangular kernel, or the widely favored Gaussian kernel. The first two cases fall under the class of naïve estimators. In the first case (histogram function), the data range is divided into a set of successive and non-overlapping intervals (bins), and the frequencies of occurrence in the bins are plotted against the discretized data range. The rectangular kernel case is similar except that overlapping intervals are permitted. In either case, the bin size should be chosen such that a sufficient number of observations falls into each bin. The resulting PDF estimate depends on the bin size as well as the discretized dataset's range and is discontinuous at the bin boundaries.

Although naïve estimators yield discontinuous results, the construction can be easily generalized to achieve continuous PDF estimates by employing different kernel functions. The smooth reproduction of the underlying

Table 1 Time elapsed on a 3.2 GHz single-core Xeon processor in estimating PDFs using Parzen windows.

D	Time elapsed	Factor increase in elapsed time
1	0.578 s	1
2	158.75 s	~275
3	~12 h (est.)	$\sim 275^2$
4	~139 days (est.)	$\sim 275^3$

Gaussian process in Fig. 1a clearly shows the advantages of Gaussian kernels in this aspect and is thus the motivation for its use here. The mathematical formula for a Gaussian kernel is defined in Eq. (2)

$$\varphi\left(x_i, x^j, h\right) = \frac{1}{\sqrt{2\pi}h} e^{-\frac{\left(x_i - x^j\right)^2}{2h^2}}, \tag{2}$$

where h plays the role of the standard deviation. The resulting expression for the 1-D Parzen window PDF estimate is,

$$p\left(x^j\right) = \frac{1}{\sqrt{2\pi}hn} \sum_{j=1}^{n} e^{-\frac{\left(x_i - x^j\right)^2}{2h^2}} \tag{3}$$

Computing complex exponentials is challenging in hardware because it requires significant hardware resources. To make the algorithm more suitable for hardware, a truncated second-order Taylor series expansion replaces the exponential function for the development of the core in hardware, as defined in Eq. (4).

$$\phi\left(x_i, x^j\right) = \begin{cases} \frac{1}{\sqrt{2\pi}h}\left(1 - \frac{\left(x_i - x^j\right)^2}{2h^2}\right), & \text{for } \left(1 - \frac{\left(x_i - x^j\right)^2}{2h^2}\right) \geq 0 \\ 0, & \text{otherwise} \end{cases} \tag{4}$$

The degree to which this quadratic approximates the true Gaussian kernel in Eq. (2) decreases for data points that yield large values of $|x_i - x^j|$. But that condition is avoided by giving zero weight to points that cause the argument $1-(x_i-x^j)^2/2h^2$ to be less than zero, which occurs when $|x_i - x^j|$ is greater than $\sqrt{2}h$. The plot in Fig. 1a for the Gaussian kernel employs the approximation given by Eq. (4). Additional management of approximation error is achieved by pre-scaling the kernel to unity variance prior to computations on the FPGA. The datasets x_i and x^j are scaled by $\sqrt{2}h$ before being transferred from the processor to the FPGA, which reduces the factor $1-(x_i-x^j)^2/2h^2$ to $1-(x_i-x^j)^2$, thus making the computational error due to the use of the Taylor series representation on the FPGA independent of the particular choice of h. The dynamic range of the dataset can further be reduced by considering only the higher order bits during computation. This method is a simple task to implement since FPGAs are extremely efficient at performing bit-level manipulations.

3.2 K-Means Clustering

In any data analysis problem given a training set of points, unsupervised learning algorithms are often applied to extract structure from them. Typical examples of unsupervised learning tasks include the problem of image and text segmentation [1]. A simple way to represent data is to specify similarity between pair of objects. If two objects share the same structure, it should be possible to reproduce the data from the same prototype. This idea underlies clustering methods that form a rich subclass of unsupervised algorithms. The K-means algorithm is a popular unsupervised clustering algorithm that partitions N data points into m clusters by finding m mean vectors $\boldsymbol{\mu}_1$, $\boldsymbol{\mu}_2$,…, $\boldsymbol{\mu}_m$ representing the centroids of the clusters (see Fig. 1b for an illustration of data points represented by two features partitioned into four clusters). It is an iterative algorithm that tries to minimize the total intra-cluster variance, or, the squared error function

$$V = \sum_{j=1}^{m} \sum_{x_i \in S_j} \left(x_i - \mu_j\right)^2 \tag{5}$$

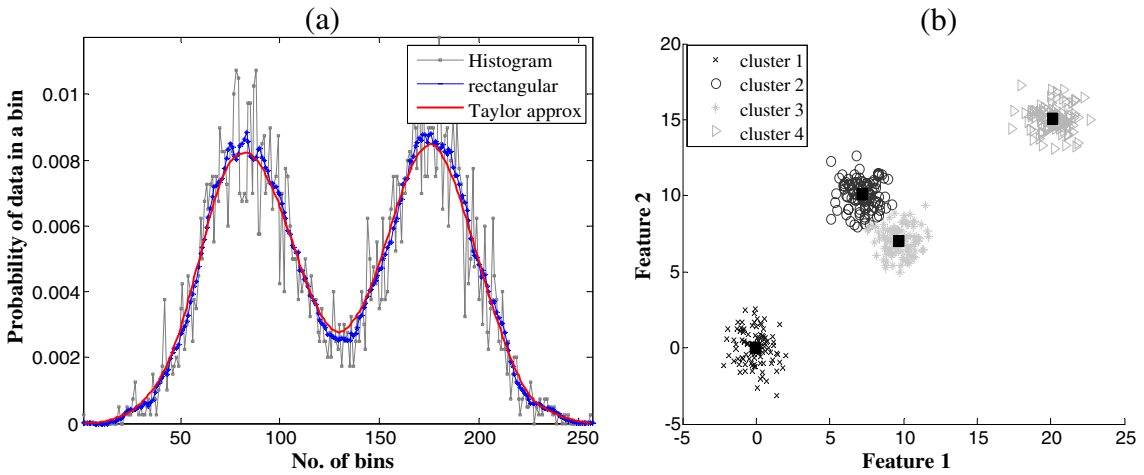


Figure 1 Illustration of **a** 1-D PDF estimation using three different kernels, and **b** K-means clustering. For the Gaussian kernel, the Taylor series approximation is used. In general, PDFs can be highly non-Gaussian (e.g., bimodal) as shown here. Parzen estimates using *rectangular*, and in particular Gaussian kernels, generally provide estimates of the true underlying PDF that are far superior to the histogram function. In the case of data clustering, it is desired to partition complex data sets in groups that share similar feature values.

where there are m clusters $\boldsymbol{S}_j$, $j = 1, 2, \ldots, m$ and $\boldsymbol{\mu}_i$ is the centroid of all the points $\boldsymbol{x}_i$ in $\boldsymbol{S}_j$.

To begin with, the data points are partitioned into m initial sets, either at random or using some heuristic approach. The centroid of each set is then computed and a new partition is constructed by associating each data point with the closest centroid. The centroids are then recalculated for the new clusters. This process is repeated until data points no longer switch clusters or alternatively centroids remain unchanged. The computational complexity of the algorithm is $O(NdmT)$ where d is the number of features representing the data point and T is the number of iterations.

3.3 Correlation

Correlation computes a measure of similarity between two datasets (e.g., audio signals) as they are shifted by one another across time. Correlation is often used to measure the delay between two signals having the same shape with one delayed relative to the other. Autocorrelation is the correlation of a signal with itself while cross-correlation is the correlation between two different signals. The correlation result reaches a maximum at the time when the two signals match best. For example, if the two signals are identical, this maximum is reached at $t = 0$ (i.e. no delay). Autocorrelation is useful for finding repeating patterns in a signal, such as determining the presence of a periodic signal that has been buried under noise, or identifying the fundamental frequency of a signal that does not actually contain that frequency component but implies it with many harmonic frequencies. Mathematically, the correlation between two discrete signals x_i and y_i with n samples each, where τ is the delay and m is the maximum delay, is given by,

$$R(\tau) = \sum_{i=1}^{n} x_i y_{i+\tau};\ 0 \le \tau \le m;\ m < n \tag{6}$$

3.4 Algorithm Similarity

Although the algorithms overviewed in the previous sections address different problems, involve different computations, and appear at different stages during data analysis, the underlying data flow within each algorithm has significant similarity. The data flow in these algorithms follows an exhaustive data permutation pattern wherein the interaction between every sample in two input vectors I_1 and I_2 affect the value of every sample in an output vector O.

In PDF estimation, it is between the data points (I_1) and the points at which the PDF is estimated (I_2). In K-means clustering it is the Euclidean distance between the data points (I_1) and the cluster centers (I_2), and in correlation it is the multiplication of two signals (I_1 and I_2) shifted over different time delays. Fig. 2 illustrates the dataflow and computations involved in each of the three algorithms in a common framework.

4 Pattern-Based Algorithm Decomposition

Designers who have achieved success in attaining orders of magnitude increase in performance (e.g., speed and power) through hardware acceleration of algorithms are in a select group and are typically highly skilled at exploiting FPGA-based systems and related tools (e.g., hardware description languages (HDLs), logic design, and performance prediction tools). To significantly increase the productivity of FPGA design and development in general, we must improve the productivity of designers who are not FPGA experts. High-level language tools, popularly known as application mappers (e.g., Impulse C, AccelDSP, Carte C, C2H), translate high-level language code to HDL versions, thereby improving productivity. Unfortunately, performance in such scenarios is highly dependent on the mapper's efficiency in extracting parallelism from the algorithm structure. The mapper also dictates algorithm decomposition for a parallel implementation, restricting the designer's freedom to explore different decomposition strategies.

Another way of improving design productivity is to reuse and leverage existing hardware designs. One of the primary motivations behind this work is to represent FPGA designs at a higher level of abstraction, thus making them more readable and therefore reusable. Design patterns [26] were introduced in the software engineering domain as common solutions to recurring problems and have successfully formed the basis for creating and exploring designs, reusing solutions for similar software applications. More recently, there have been efforts to apply pattern-based design to FPGA-based reconfigurable computing [27, 28]. In [25], the authors identified and cataloged an extensive list of design patterns that can be used to solve a broad range of problems spanning diverse fields. By decomposing algorithms in terms of simple design patterns, a developer is able to understand the dataflow and structure behind the parallel implementation without delving into the details of a hardware circuit. In our work, we have identified a list of primitive design patterns and categorized them in such a manner that is most appropriate for our use for the design of machine-learning algorithms and for efficiently exploring the designs using performance prediction tools like RAT [23]. Two of the most important categorizations of patterns and their examples are shown in Table 2: computation patterns and communication patterns. While computation patterns deal with extracting and implementing parallelism in the algorithm, communication patterns provide different means of regulating data flow to keep the computation

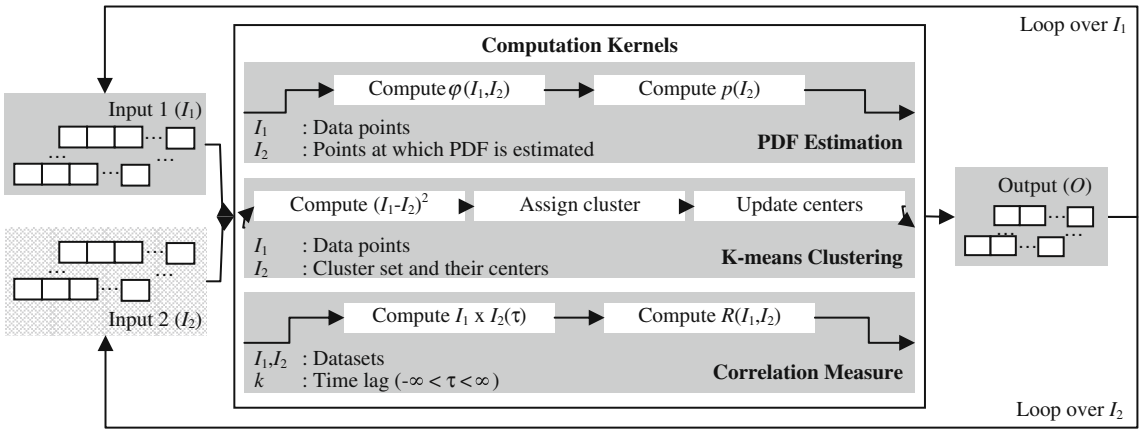


Figure 2 Dataflow and computational structure of PDF estimation, K-means clustering, and correlation.

patterns busy. Note that the patterns listed in Table 2 are primitive patterns. A variety of composite patterns can then be constructed from these primitive patterns to solve potentially any complex problem. For example, a wavefront pattern [25] can be constructed using a datapath replication pattern with each parallel datapath comprising of pipeline patterns of different lengths.

4.1 Pattern Description

The pipeline and datapath replication patterns are described in this section using a standard format suggested in [25]. These two primitive patterns are highly important in that any composite pattern for the purpose of expressing and implementing parallelism can be constructed from a combination of pipeline and datapath replication patterns.

4.1.1 Pipeline Pattern

- *Intent*: Used for implementing programming structures wherein successive/intermediate instructions overlap in execution. The instruction throughput is increased by executing a number of instructions per unit of time resulting in a lesser number of net clock cycles spent during processing.
- *Motivation*: The primary motivation is to achieve faster processing speeds. When a program runs on a processor, the basic assumption is that each instruction in the program is executed before the execution of a subsequent instruction is begun. However, in many cases the instruction throughput could be increased if the program allows the possibility to execute more instructions per unit of time. Each instruction is computed and linked into a 'chain' so each instruction's output is an input to a subsequent instruction.
- *Applicability*: This pattern is intended for program structures where different instructions in the program execution could operate over different data concurrently.
- *Participants*: A communication pattern regulates dataflow; partial/intermediate results may need to be accumulated or buffered for subsequent usage.

Table 2 List of primitive patterns and their description.

Pattern name	Description
Computation patterns	
1. Pipeline	Increases throughput by overlapping execution of independent computations
2. Datapath replication	Exploits parallelism by duplicating computational kernels
3. Loop fusion	Alleviates potential pipeline stalls in nested loops by fusing them
4. Tree [27]	Provides computational structure for implementing tree-based algorithms
5. Mesh [27]	Provides templates for implementing image-based or 2-D problems
6. Look-up tables [28]	Reduces run-time computation of complex operators by simpler lookup ops
Communication patterns	
1. Scatter	Distributes data among many computational kernels
2. Gather	Collects data (results) from many computational kernels
3. Broadcast	Replicates data to many computational kernels
4. Round robin	Scatters equally-sized blocks of data in sequential order among all kernels
5. Memory resolution	Resolves memory contention problems by inserting pipeline stages & buffers

- *Collaborations*: A communication pattern regulates dataflow and keeps data fed to the pipeline.
- *Consequences*

    1. When all instructions in the program execution are not independent, the pipeline control logic (i.e. the controller) must insert a stall or wasted clock cycle into the pipeline until the dependency is resolved.
    2. While pipelining can theoretically increase performance over a non-pipelined core by a factor of the number of stages (assuming the clock frequency also scales with the number of stages), in reality, design limitation will not always lead to ideal execution.
    3. The instruction latency in a non-pipelined design is slightly lower than in a pipelined equivalent. This is due to the fact that extra flip flops must be added to the data path of a pipelined design.
- *Known uses*: Any algorithm/program that has independent instructions in its structure (e.g., PDF estimation, Correlation).

4.1.2 Datapath Replication Pattern

- *Intent*: Used for exploiting computational parallelism observed in sequential programming structures such as computational loops.
- *Motivation*: The primary motivation is to achieve faster processing speeds. Parallelism observed in computations can be efficiently exploited using custom logic. Computational structures such as loops iterate over the same set of processing instructions multiple times. In case of zero data dependencies between iterations (or limited data dependency), the processing instructions can be replicated in hardware to perform multiple iterations of the loop in parallel.
- *Applicability*: This pattern is intended for exploiting parallelism in computational structures such as loops. An important consideration for implementing the pattern is data dependencies between loop iterations. Depending on the individual case, the implementation may need intermediate buffers or communication links between parallel implementations of computational kernels.
- *Participants*: Apart from the replicated kernels, a communication pattern regulates dataflow with possible requirements of additional buffers.
- *Collaborations*: Communication patterns regulating dataflow and the total number of iterations required on the kernel instances implemented in hardware.
- *Consequences*

    1. Since the pattern implements an area-time tradeoff, higher processing speeds achieved via parallel instances of the kernel come at the cost of increased implementation footprint in hardware.
    2. Additional overhead in terms of parallel data communication and control logic are present. Depending on the FPGA platform and implementation, the problem may be communication-bound, limiting the number of parallel kernels that can be fed with data in parallel.
- *Known uses*: PDF estimation, Molecular dynamics, K-Means clustering, Sorting algorithms.

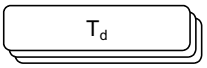
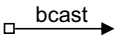
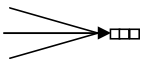
4.2 Quantifying Design Patterns

Since design patterns provide a formalized way to represent algorithm decomposition and the associated communication for parallel implementation, they can be quantified and used in analytical models like RAT to predict the algorithm's amenability to hardware acceleration (to be described in Section 5.1), before undertaking a lengthy (and possibly fruitless) development process. We can analyze the impact of each design pattern in terms of throughput and/or overhead contribution to design performance. In this work, *throughput* of a design pattern is defined as the maximum number of operations that it can perform per clock cycle. *Latency* is defined as the number of cycles spent in data transfers before any useful computation is performed. For example, a pipeline pattern has a *throughput* equal to the number of pipeline stages and contributes an equal cycle count on *latency* (i.e. clock cycles spent in filling up the pipeline stages). The *throughput*, *latency*, and graphical notation of patterns relevant to this work are given in Table 3. Having decomposed the algorithm as a composite of primitive patterns, the net *throughput* and *latency* can be inferred from the structure and behavior of the constructed composite pattern.

5 A Scalable and Portable Architecture for PDF Estimation

Having investigated and analyzed the commonalities shared by the three case-study algorithms in this work (see Section 3) we suggest that an efficient FPGA design developed for one of the algorithms, say PDF estimation, can be effectively reused for computing the others. For this reason, an in-depth explanation of the design and performance evaluation of a suitable architecture for only the PDF estimation algorithm will be given in the following sub-sections. Then, in Section 6, we will discuss how the architecture developed for PDF estimation can be effectively adapted by reusing most of its design for developing other similar algorithms.

Table 3 *Throughput*, *latency* and notations of primitive patterns.

Pattern	*Throughput* (ops/cycle)	*Latency* (cycles)	Notation
Pipeline	T_p = no. of pipeline stages	L_p = no. of pipeline stages	
Datapath replication	T_d = no. of replications	N/A	
Broadcast	N/A	N/A	
Scatter	N/A	L_s = buffer length	
Gather	N/A	L_g = buffer length	

The general architecture of the 1-D PDF estimation algorithm based on pattern-based decomposition is shown in Fig. 3. A computational kernel updates the PDF value at a particular point x (i.e. a bin) based on the value of a particular data sample x_i (where $x_i \in I_1$ and $x \in I_2$). The Parzen window technique is an embarrassingly parallel algorithm where the PDF values at multiple points can be evaluated at the same time by replicating the computational kernel (datapath replication pattern). The kernels in the parallel datapath are *seeded* with different values in I_2 via a scatter communication pattern. Thus, data samples can be broadcasted across kernels that can then process data independently in parallel (see Fig. 3). Each kernel can also benefit from a pipelined implementation while performing all the operations (see Eq. 4) required on every data sample (pipeline pattern). The PDF values are computed in $\varphi(I_1,I_2)$, accumulated in $p(I_2)$ and stored in output memory O. Load balancing and reduction of communication and synchronization requirements are necessary to ensure that hardware outperforms its sequential software counterpart. In estimating multi-dimensional PDFs, computation of the kernel functions φ in each dimension are independent of the others and hence performed in a parallel fashion within each pipeline. Internal registering for each bin keeps a running total of all processed data; these cumulative totals comprise the final estimation of the PDF function.

The development stages in the system design are highlighted in Fig. 4 and explained briefly in the following bullets for a 1-D PDF design. The same can be extended for higher-dimensional PDF estimation with suitable modifications made to the computational kernel.

- *Performance Prediction*—Based on a preliminary design of the algorithm (in the form of design patterns shown in Fig. 3a) and the basic resources available for a selected FPGA-based computing platform (in the form of parameter values to quantify the design patterns), the attainable speedups are predicted using RAT. Numerical

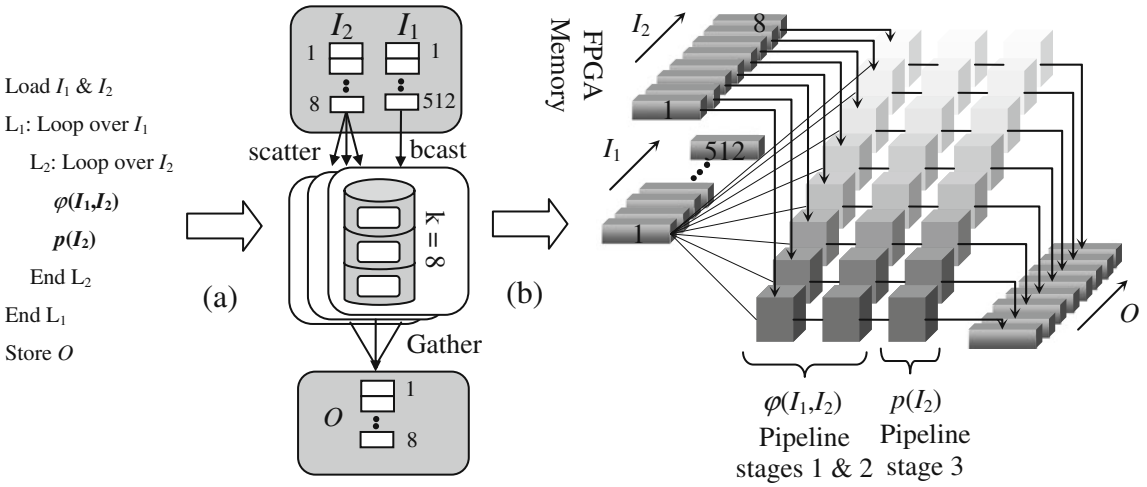


Figure 3 Pattern-based **a** decomposition and **b** design of the 1-D PDF estimation algorithm.

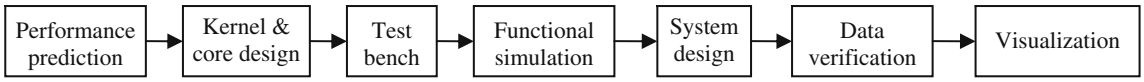


Figure 4 Development stages in system design.

analysis was conducted and a suitable fixed-point implementation is chosen because probability values lie between 0 and 1, negating the need for the dynamic range features of a floating-point format. Details of RAT's performance prediction are provided in Section 5.1.

- *Kernel and core design*—The basic task of a kernel in the 1-D case is the computation of the kernel function $\varphi = 1\text{-}(x_i\text{-}x)^2$. A core contains a number of kernels (k) with each kernel in the design performing the aforementioned computation for a different x (i.e. different *seed*). The computation increases in complexity while estimating higher-dimensional PDFs (refer Section 5.3). The parameter k is chosen based upon the preliminary resource analyses performed earlier. Details concerning the kernel and core designs are given in Section 5.2.
- *Test bench and simulation*—Memory instantiations for x and x_i are made, test bench files are generated, and functional simulations are performed to validate the design.
- *Overall system design, verification, and visualization*—Integration of the core with the host processor (middleware design) is developed followed by verification of the computed PDF.

5.1 Performance Prediction

Although FPGAs have much to offer in terms of flexibility and performance, they are not amenable for all algorithms. RAT [23] is a simple methodology developed for predicting the performance of a specific algorithm design on a specific platform. By parameterizing a particular design strategy and a platform selection into RAT, the developer can analyze and predict likely speedups attainable. For RAT, *speedup* is defined as the ratio of execution time on a relevant general-purpose processor (t_{GPP}) to the execution time on an FPGA (t_{RC}).

$$speedup = \frac{t_{GPP}}{t_{RC}} \tag{7}$$

For ease of predicting speedup, the RAT analytic models take the form of a worksheet, reproduced in Table 4, and feature two important steps. The first step deals with estimating the communication burden (t_{comm}) involved in transferring data in and out of the FPGA. The entries in the worksheet related to this step are the communication parameters ($throughput_{ideal}$, α_{write}, α_{read}) and the dataset parameters. The second step involves the estimation of time spent in performing computation (t_{comp}) over the data transferred ($N_{elements}$) on the FPGA. In the original formu-

Table 4 RAT input parameters for analysis of 1-D and 2-D PDF estimation algorithms.

Dataset parameters		1-D PDF	2-D PDF
$N_{elements}$, input	(elements)	512	1024
$N_{elements}$, output	(elements)	1	65536
$N_{bytes/elements}$	(bytes/elem)	4	4
Communication parameters		1-D PDF	2-D PDF
$throughput_{ideal}$	(MB/s)	1000	1000
α_{write}	$0<\alpha<1$	0.37	0.37
α_{read}	$0<\alpha<1$	0.16	0.16
t_{comm}	(sec)	6.0E-6	1.6E-3
Computation parameters		1-D PDF	2-D PDF
$N_{Ops/element}$	(ops/elem)	768	393216
T_d	(ops/cycle)	8	16
$Latency_{NET}$	(cycles)	515	131072
T_p	(ops/cycle)	3	3
f_{clock}	(MHz)	150	100
t_{comp}	(sec)	1.2E-4	4.3E-2
Software parameters		1-D PDF	2-D PDF
t_{GPP}	(sec)	0.578	158.8
N_{iter}	(iterations)	400	400
Predicted speedup		11.5	9.0

lation of RAT, parameters $N_{ops/element}$, f_{clock}, $throughput_{proc}$ and $N_{elements}$ determine computation time. The total time spent on the FPGA (t_{RC}) to execute the entire algorithm is calculated (see [23] for details) as,

$$\begin{aligned} t_{RC} &= N_{iter}\left(t_{comm} + t_{comp}\right) \\ t_{comm} &= t_{read} + t_{write} \\ t_{read/write} &= \frac{N_{elements} \times N_{bytes/element}}{\alpha_{read/write} \times throughput_{ideal}} \\ t_{comp} &= \frac{N_{elements} \times N_{ops/element}}{f_{clock} \times throughput_{proc}} \end{aligned} \tag{8}$$

The memory available on the FPGA or the board hosting the FPGA is often much smaller than what is required for storing all of the application data. N_{iter} denotes the number of iterations of communication and computation required to process all available data. The authors in [23] suggest that the parameter $throughput_{proc}$ be approximately chosen based upon the number of data samples that can be processed in parallel. This is not trivial to estimate in many cases and might lead to suboptimal predictions if carelessly chosen. A pattern-based design embeds the parallelism (*throughput*) and the accompanying overhead (*latency*) during algorithm decomposition and would help better parameterize RAT while estimating t_{comp}. Latency effects in pipelines and buffers are easily extracted from the patterns as against manual interpretation. For example, in the PDF design (see Fig. 3), t_{comp} is computed as,

$$\begin{aligned} t_{comp} &= \left(Latency_{net} + \frac{N_{elements} \times N_{ops/element}}{throughput}\right) \times \frac{1}{f_{clock}} \\ Latency_{net} &= L_s + L_p + L_g \\ throughput &= T_p \times T_d \end{aligned} \tag{9}$$

The worksheet in Table 4 shows the input parameters for the RAT analysis for the 1-D and 2-D PDF estimation algorithms. The communication parameters model a Nallatech H101-PCIXM card containing a Xilinx V4LX100 user FPGA connected to a Xeon host processor over a 133 MHz PCI-X bus. Although the entire application involves 204,800 data samples, each iteration of the 1-D PDF estimation on the FPGA will involve only a portion of that data (512 data samples, or 1/400 of the total set) because of memory constraints on the FPGA. A corresponding input buffer size of 512 is chosen for entry in the worksheet. Since the computation is performed in a two-dimensional space for a 2-D PDF, twice the number of data samples (in blocks of 512 words for each dimension) is sent to the FPGA. The number of output elements in the RAT table corresponds to the number of outputs for every call to the FPGA. In the 1-D case, the estimated PDF values at 256 points are returned from the FPGA after all calls to the FPGA are complete (i.e. after N_{iter} calls). There is sufficient block RAM on the FPGA to hold the results of the 1-D PDF algorithm. Since the PDF output elements need not be sent for each call, we account for the communication time by distributing the total number of output elements over N_{iter} calls. From Table 4, we understand that the FPGA is called 400 times resulting in an average value of <1 output element per call. We round this to the next largest integer (1 in this case). In the 2-D case, the PDF values are computed over 256×256 (i.e. 65536 in the worksheet) points and are sent back to the host for every call made to the FPGA due to insufficient block RAM to store results between calls.

The computational density of the algorithm is defined in terms of $N_{ops/element}$. Each data sample in the 1-D PDF case requires three operations at each of the 256 points at which the PDF is estimated, resulting in 768 operations. In the 2-D PDF algorithm, each data sample requires 6 operations at each of the 256×256 points leading to 393216 operations. The computational throughputs for the 1-D and 2-D designs are estimated using FPGA clock frequencies of 150 MHz and 100 MHz, respectively. The *throughput* of the design is determined based on the constructed composite pattern (see Fig. 3)—eight parallel datapaths, each comprising of a 3-stage pipeline, leads to a net *throughput* of 24 ops/cycle. L_s and L_g equal 256 (i.e. size of I_2 used to seed the parallel datapaths) and L_p equals three leading to a net *latency* of 515 cycles. For the 2-D PDF design, the latency L_s and L_g increase to 65536 (256×256 seeds) >while the number of replicated datapaths equals 16 (i.e. eight for each dimension). The software baseline (t_{GPP}) for computing speedup values was measured from an optimized C program (optimization flag was set to O3) executed on a 3.2 GHz single-core Xeon processor using single-precision floating point. The truncated Taylor series expansion was used in place of the exponential in the C program as well. RAT predictions are then estimated to determine an attainable execution time and later compared against experimentally measured software results to compute speedup. The predicted speedup for the 1-D and 2-D PDF algorithms was 11.5 and 9.0, respectively.

5.2 1-D PDF Design

A design of a 1-D PDF relies heavily on the availability of dedicated arithmetic units and memory blocks. Not only should the available resources be efficiently used but also the design should scale well with application complexity. Taking these points into account, a multi-core design with a key design parameter k (the number of kernels or parallel datapaths in a core) is proposed. The support size of the

PDF is 256 along one dimension (I_2) and the number of data samples processed for every call to the FPGA is 512 (I_1). The multi-core aspect of the design is developed with consideration toward future extensions to multi-FPGA systems. Larger FPGAs would be able to house more kernels and hence process more data in parallel leading to faster computations. In the following sections, single- and dual-core designs are described, along with a discussion of *scalability* with respect to the number of cores.

5.2.1 Single-Core Design

The FPGA receives data (I_1 and I_2) over an interconnect (e.g., PCI-X, PCI-Express, RapidIO) and the core accesses data from the FPGA memory. The basic blocks in the design are pictorially represented in Fig. 6a and the host-centric execution flow is illustrated in Fig. 7a. A set of data samples (I_1) is first loaded onto the FPGA that is then signaled to start the computation. As the FPGA processes data, the host processor polls for a "process complete" signal from the FPGA. The FPGA sends the "process complete" signal once the computation is finished and the host sends in the next set of data samples (I_1) until all data are processed. Data and control flows to the parallel kernels housed in the core are regulated by a finite-state machine explained as follows (see Fig. 5). The set of *seeds* x scattered to the k parallel kernels in the core are represented by I_2, the data samples x_i by I_1, and the PDF values by p in the state machine description.

- *Idle*—Remain in *idle* state until activation signal GO=0; transition to *scatter* state if GO=1.
- *Scatter*—Distribute k values of I_2 and $p(I_2)$ to the parallel kernels; transition to *compute* state.
- *Compute*—Broadcast data samples I_1 sequentially, compute $\varphi(I_1,I_2)$, and update $p(I_2)$; transition to *scatter* state to load subsequent k values of I_2, else move to *Gather* state.
- *Gather*—Update p in FPGA memory; transition to *done* state.
- *Done*—Send "process complete" signal done=1 to host; transition to *idle* state when GO=0.

5.2.2 Dual-Core Design

The architectural details and the execution flow of the dual-core design are illustrated in Figs. 6b and 7b, respectively. The state machine for a single-core design and the corresponding core are replicated with changes made to the data flow in the software control program at the host end. Since multiple cores share the same interconnect, arbitration for bus access is performed while loading data onto the on-chip memory of the FPGA and the subsequent polling for the "process complete" signal. The FPGA systems used in this work have a host processor for explicitly managing data movement in and out of the FPGA. Data is written to the first core and, while the first core processes the data, the next set of data samples is loaded to the second core to process. The host then polls the first core for the "process complete" signal as the second core processes its data. There could be a situation where the host cannot poll one of the cores while it is loading data to the other core. The core that is not being polled would sit idle during this contention period.

5.3 2-D PDF Design

In the 2-D case, the PDF is evaluated at a matrix of points or bins (n_1 rows of x values × n_2 columns of y values). The number of computation grows from $(N - n)^2 + c$ to $(N - n_1)^2 + (N - n_2)^2 + c$ and the basic kernel computation expands to $1-((x_i-x)^2 + (y_i-y)^2)$ where x_i, y_i represent one data sample. Despite the added complexity of the 2-D PDF algorithm, the increased quantity of parallelizable operations

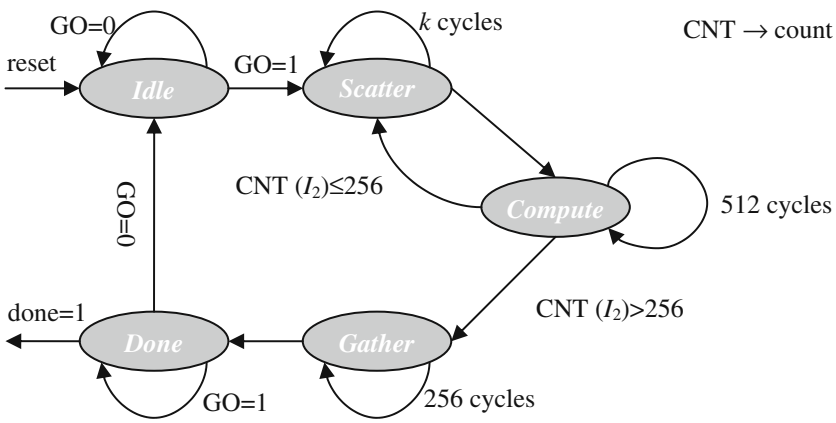


Figure 5 State transition diagram for a single-core 1-D PDF design.

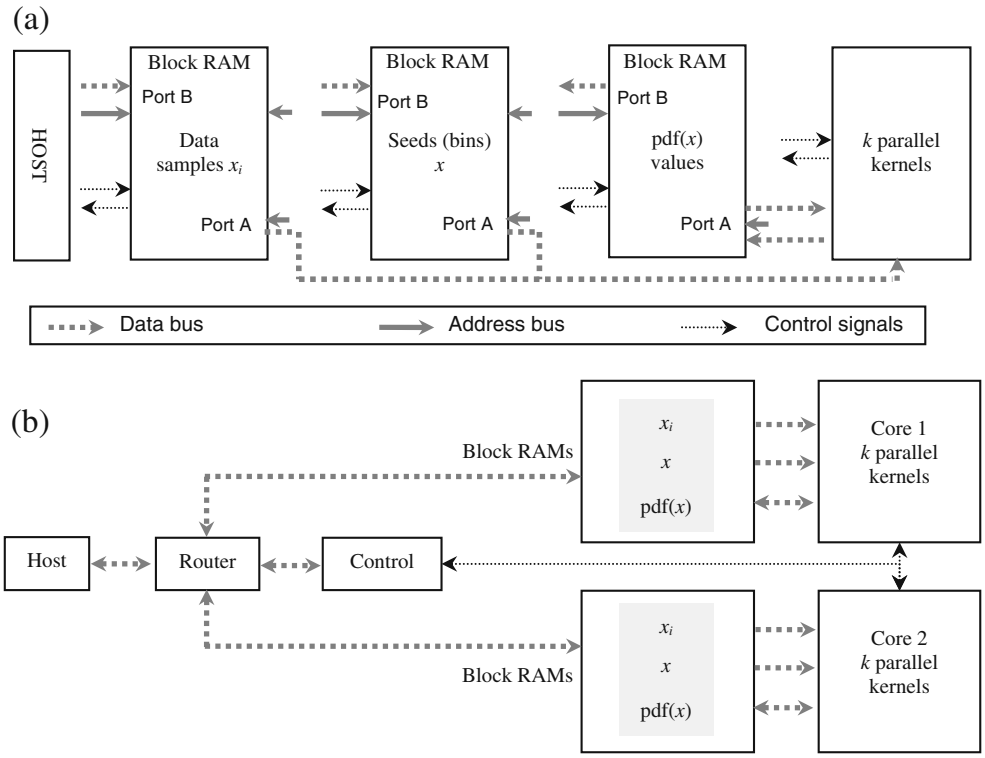


Figure 6 Design details of **a** single-core and **b** dual-core 1-D PDF.

makes this algorithm still amenable to the RC paradigm, assuming sufficient quantities of hardware resources are available. To perform the 2-D PDF estimation, the basic architecture designed for the 1-D case is extended with modifications made primarily to the kernel design (see Fig. 8) and the control flow. Due to the limited amount of FPGA memory, the PDF matrix is computed one column at a time (i.e. for all x and one y, represented by $p(I_{2x},y)$). This partial PDF matrix is returned to the host before a subsequent call to the FPGA is made. The kernels in the 2-D PDF design are initialized with two sets of seeds (x and y).

The finite-state machine regulating the data and control flow is illustrated in Fig. 9 and explained as follows. The set of seed values x and y scattered to the parallel kernels (k in number) in the core are represented by I_{2x} and I_{2y}, the data samples x_i and y_i by I_{1x} and I_{1y}, and the PDF column values by $p(I_{2x},y)$ in the state machine description.

- *Idle*—Remain in *idle* state until activation signal GO=0; transition to *scatter* state if GO=1.
- *Scatter*—Distribute y, k values of I_{2x}, and $p(I_{2x},y)$ to the parallel kernels; transition to *compute* state.
- *Compute*—Broadcast data samples I_{1x} and I_{1y} concurrently and update $p(I_{2x},y)$; transition to *scatter* state to load subsequent k values of I_{2x}, else move to *Gather* state.
- *Gather*—Update $p(I_{2x},y)$ (i.e. column of PDF matrix) in FPGA memory; transition to *done* state.
- *Done*—Set the "process complete" signal done=1; when GO=0 transition to *scatter* state if CNT(I_{2y})≤256 (point to next element in y), else move to *idle* state.

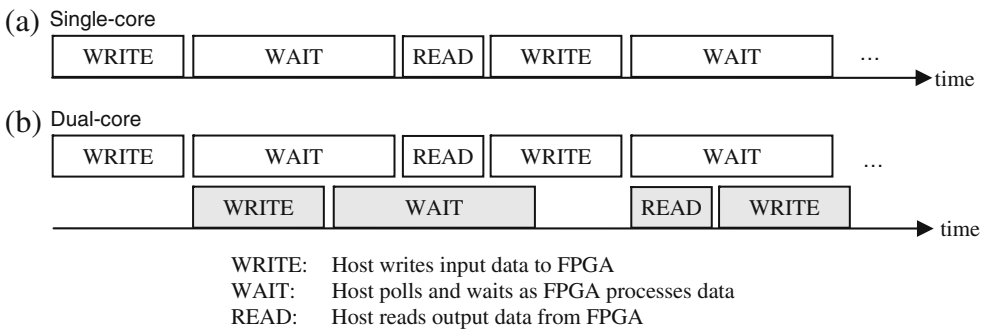


Figure 7 Host-centric execution flows for **a** single-core and **b** dual-core designs.

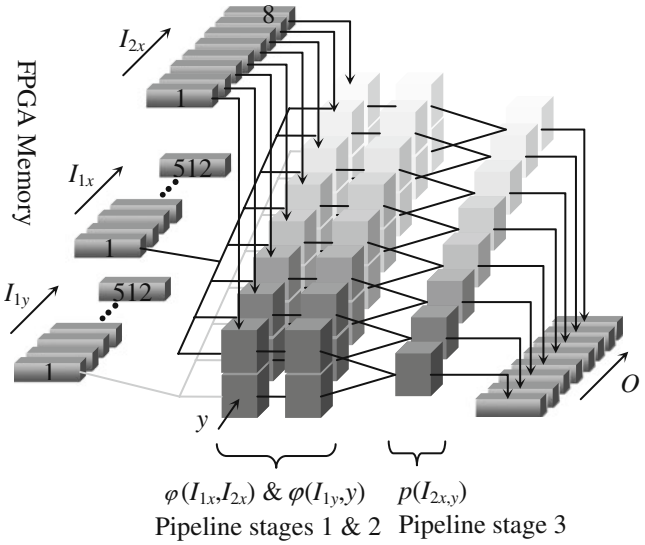


Figure 8 Architecture design for 2-D PDF estimation algorithm.

5.4 Experimental Results

In this section, experimental results with the PDF FPGA designs are presented and analyzed. The experimental platform used throughout these experiments contains the Nallatech H101 board with V4LX100 FPGA. This Xilinx chip has dedicated DSP48 arithmetic blocks capable of performing rapid 18-bit multiplies and multiply-accumulates compared to logic-based MAC cores. The board communicates with the host processor over a PCI-X interconnect. A 32-bit wide communication channel is used of which 9 bits were allocated for the fixed-point fractional segment. Function-level simulation was performed in ActiveHDL from Aldec, and H101 implementation was rendered using DIMEtalk from Nallatech.

5.4.1 Speedup and Resource Utilization

One of the primary objectives in this work is to obtain speedups in execution by exploiting parallelism at the hardware level when compared to the sequential version execution on a high-end CPU. The single-core 1-D and 2-D PDF designs operated at FPGA clock speeds of 150 MHz and 100 MHz, respectively. The number of kernels, k, in the core was set to 8. The basic criterion was to develop an efficient design where the resources (e.g., DSP48s, block RAMs, slices) are uniformly consumed. It should be noted that, since the algorithm is embarrassingly parallel, improved performance can be achieved by scaling the design, i.e. processing more data in parallel by increasing k until on-chip resources are exhausted.

Actual speedup and resource utilization factors for the 1-D and 2-D PDF algorithms are presented in Table 5. It can be seen from the comparisons made in Table 6 that the actual speedups obtained are reasonably close to the predicted speedups using RAT (originally shown in Table 4). The deviation between the values is primarily due to inaccuracies in the estimate of t_{comm} values. Although the channel efficiency values (α_{write}, α_{read}) chosen for our RAT analysis are a reasonable assumption for large data transfers, they tend to be lower for smaller data transfers as in this scenario. The t_{comp} predictions are relatively close to the experimental values validating the advantage of employing design patterns for algorithm decomposition and performance prediction. The number of DSP48s scales linearly with the number of kernels k in the core. In the 2-D PDF scenario, each kernel uses twice the number of DSP48s as was consumed in the 1-D case because the computation is conducted along two dimensions. Although the total amount of FPGA memory consumed by the PDF core is moderate, a significant percentage of it is consumed by DIMEtalk's interface to buffer and transfer data to and from the host and FPGA.

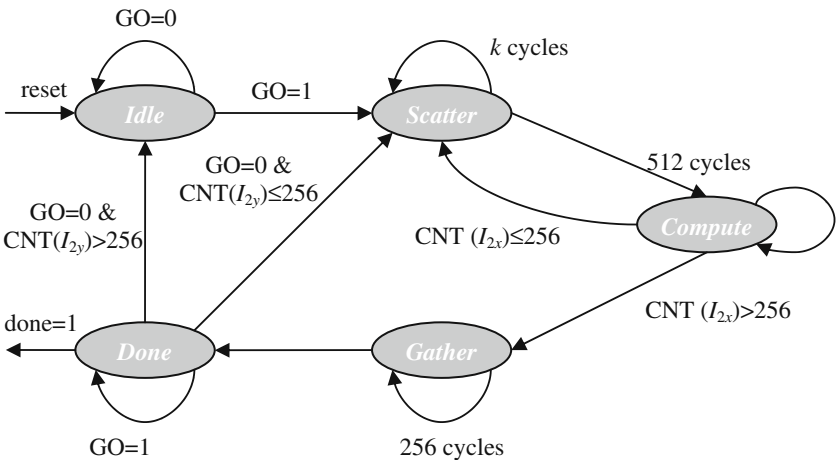


Figure 9 State transition diagram for a single-core 2-D PDF design.

Table 5 Speedup & utilization for single-core designs.

Description	1-D PDF	2-D PDF
DSP48s (%)	8	16
BRAM (%)	12	15
Slices (%)	11	13
Actual Speedup	7.8	7.1

5.4.2 Data Verification

Even though significant speedup was shown in the previous section, speedup is meaningless if the results are not accurate. In this section, we verify the accuracy of the data provided by our FPGA designs. Data samples from a bimodal mixture of Gaussian distribution and from a two-dimensional uni-modal Gaussian distribution were generated in MATLAB to verify the functionality of the 1-D and 2-D PDF designs. PDF values were computed by generating 800 KB (N=204,800) of data samples (n=256 for 1-D PDF; n_1=256 and n_2=256 for 2-D PDF). The computed PDF values were read and error in the solution was computed as the difference in FPGA and GPP estimates. Maximum error E of 3.8% and 0.57% were obtained for the 1-D and 2-D algorithms respectively (see Eq. 10), which are reasonable for most real-time applications that involve decision-making based on probabilistic reasoning. The errors in the estimates are due to the Taylor series truncation of the exponential function rather than the fixed-point effects (the exponential function was not truncated in the GPP estimates for this study). The resulting PDFs shown in Figs. 10 and 11 were plotted in MATLAB for verification.

$$E = \frac{\max\left(\left|p(x)_{RC} - p(x)_{GPP}\right|\right)}{p(x)_{GPP}} \tag{10}$$

5.4.3 Scaling to Multiple Cores

Since the PDF algorithm is embarrassingly parallel, significant speedup can be achieved by scaling the design to multiple cores. In this work, dual-core architectures were developed and evaluated to study inherent characteristics for scalability on single- or multiple-device systems. The dual-core results for the 1-D and 2-D cases are summarized and compared to the single-core implementation in Table 7. Design frequencies of 150 MHz and 70 MHz were obtained for the 1-D and 2-D algorithm designs, respectively. The reduction in frequency was primarily because of increased routing delays attributed to a larger design. Consumption of a greater percentage of DSP48 slices is also bound to increase routing delays as the DSP48 slices are located at specific areas across the IC. An important point to note here is that speedup does not double in a dual-core design (as compared to the single-core design) due to interconnect contention as discussed in Section 5.2.2 and a reduced frequency in the 2-D PDF case. These parameters would potentially differ from one platform to another depending upon the system interconnect bandwidth and its architecture. Also, the dual-core 1-D PDF design offered a higher increase factor in speedup when compared to the dual-core 2-D PDF design because time spent on data communication

Table 6 Performance factors for single-core designs.

Description	Predicted		Actual	
	1-D PDF	2-D PDF	1-D PDF	2-D PDF
t_{comm}(sec)	6.0E-6	1.6E-3	2.5E-5	1.1E-2
t_{comp}(sec)	1.2E-4	4.3E-2	1.4E-4	4.4E-2
t_{RC} (sec)	5.1E-2	1.7E+1	7.4E-2	2.2E+1
Actual Speedup	11.5	9.0	7.8	7.1

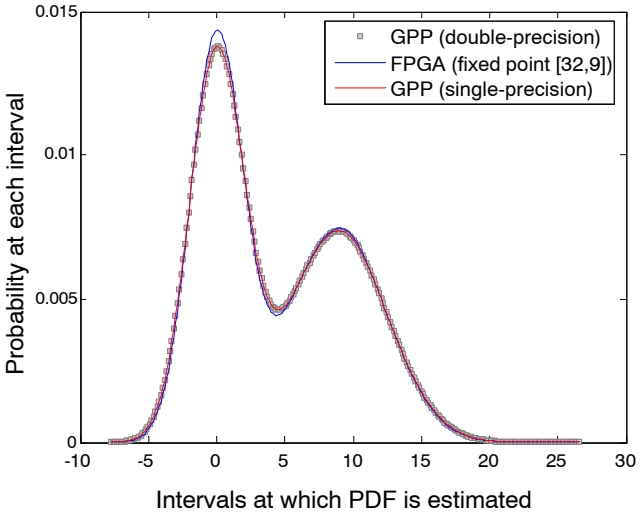


Figure 10 1-D PDF estimates: GPP vs. FPGA.

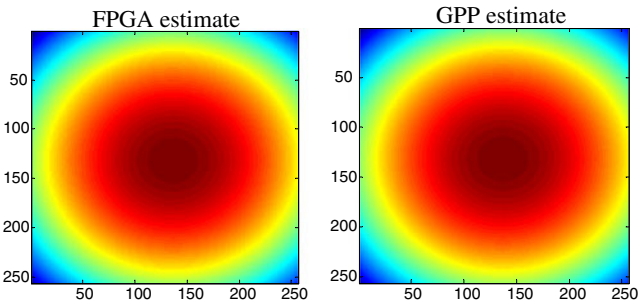


Figure 11 2-D PDF estimates: GPP vs. FPGA.

Table 7 Speedup & utilization for dual-core designs.

Description	Single-core		Dual-core	
	1-D PDF	2-D PDF	1-D PDF	2-D PDF
DSP48s (%)	8	16	16	33
BRAM (%)	12	15	15	21
Slices (%)	11	13	16	22
Actual Speedup	7.8	7.1	13.4	8.3

could be effectively hidden under time spent on computation in the 1-D PDF case. This outcome was not achievable in the 2-D PDF designs due to longer communication times (columns of the PDF matrix are sent back to the host after every call to the FPGA) and more importantly a reduced frequency of operation.

5.4.4 Portability Considerations

In addition to having scalability impacts, application characteristics also contribute greatly to platform selection and performance. In this work, optimum choice of the number of kernels per core, k, should be made based upon the interconnect bandwidth and FPGA resources available. An important byproduct of the RAT methodology, not mentioned earlier, is the computation and communication utilization factors given in Eq. 11.

$$util_{comm} = \frac{t_{comm}}{t_{comm} + t_{comp}}; \quad util_{comp} = \frac{t_{comp}}{t_{comm} + t_{comp}} \tag{11}$$

These metrics indicate the nature of a particular design in terms of whether it is communication-bound or computation-bound. If there are a variety of platforms to choose from, the designer could make use of these metrics to modify architecture selection for achieving optimum results. The primary resource-limiting factor in increasing the value of k in this algorithm design is the number of dedicated multiplier blocks in the FPGA. While platforms with FPGAs having more multipliers (e.g., the Cray XD1 with Virtex-2 Pro FPGAs) and faster theoretical interconnects (e.g., RapidArray in XD1) might intuitively suggest improvements in speedup by performing more computations in the PDF algorithm in parallel, these could be negated by poor read and write efficiencies (α_{read}, α_{write}) during data communication. This case was particularly true with the Cray XD1 system. Due to extremely low read speeds on CPU-initiated transfers from FPGA-host memory (~4 MB/s for small data transfers), RAT predicted a smaller speedup number (see Table 8) in migrating the 2-D PDF algorithm to the XD1 system even though it housed a theoretically faster interconnect. In comparison to the Nallatech platform, the utilization factors in Table 8 illustrate the fact that in the XD1 more time is spent in data communication as compared to algorithm computation for the 2-D PDF design. On the contrary, since the computed PDF values are sent to the host after all FPGA iterations are complete, fewer read operations are required in the 1-D PDF design resulting in the XD1 offering more than double the speedup achieved on the Nallatech platform.

6 Composite Patterns for Design Reuse

As discussed earlier, it is often the case that most of the multimedia analysis tools employ different algorithms at different stages of data analysis. To be highly productive in creating efficient hardware designs for each of those algorithms, it is essential that we reuse existing hardware designs. One of the goals of design patterns is to provide the framework to realize this concept. In this section, we exploit the commonalities between the algorithms described in Section 3.4 and the similarities in the composite patterns describing their decompositions by reusing the structural and communication architecture developed for PDF estimation. Also, state transition diagrams (see Figs. 5 and 9) primarily deal with managing data flow and activate the computations to execute at appropriate times. This task is particularly cumbersome to design and debug, and any means of reducing this burden would be helpful to designers. Due to similarities in algorithm decomposition, the state transition diagram developed for PDF estimation can be effectively reused for the other two algorithms.

6.1 K-means Clustering

K-means clustering can be decomposed in a similar fashion to that of PDF estimation. Computation of the Euclidean distance between every data sample and the k cluster centers can be done in parallel. The minimum distance indicating the cluster closest to the data sample can then be obtained hierarchically by comparing the Euclidean distances pair-wise in a parallel fashion (see Fig. 12). The basic update rule in K-means clustering for the cluster to which the new sample belongs can be

Table 8 Speedup & utilization for single-core designs across platforms.

Description	Nallatech		Cray XD1	
	1-D PDF	2-D PDF	1-D PDF	2-D PDF
$util_{comm}$	0.14	0.20	0.03	0.38
$util_{comp}$	0.86	0.80	0.97	0.62
RAT Predicted speedup	11.5	9.0	13.0	3.7
Actual Speedup	7.8	7.1	20.6	4.0

defined as Δcenter = η (sample$_{new}$ - center$_{old}$), where η is the learning rate [29]. In this work, the cluster centers are updated in the following manner, as and when a new data sample is assigned to a particular cluster:

$$center_{new} = \frac{center_{old} + sample_{new}}{2} \quad (12)$$

This approximate update rule (where η=1/2) is often used for on-line clustering and is applicable to real-time scenarios in which fast reaction to changing statistics is often desirable. The update method reduces the computational requirements to addition and right shift operations and eliminates the more resource-consuming division operation. While this offers an efficient hardware implementation, it can lead to slower convergence for broadly distributed clusters due to its relatively high sensitivity to each new sample. In such cases, the traditional approach proposed by MacQueen [29] can be used where η=1/n_j (here, n_j is the number of data points in class S_j). This choice for η achieves a more gradual updating of cluster centers as the number of data samples presented to algorithm grows. The design developed in this work can be extended to employ MacQueen's approach with minimal resource usage by the inclusion of a multiplier and look-up table (for storing η values) to implement the update rule. All the computations involved in the algorithm can further be implemented as a pipeline. Analyzing the composite pattern for decomposing K-means in Fig. 12, it can be inferred that the architecture developed for PDF estimation can be effectively reused for K-means clustering with minimal modifications as explained in the following section. The cluster centers (I_2) are used to *seed* the parallel datapaths in Fig. 3a (for the PDF estimation algorithm) and the data samples (I_1) are fed into the pipelines sequentially to determine their cluster association. The number of parallel datapaths and the length of I_2 are both increased from 8 (in the architecture of PDF estimation) to 64, corresponding to the number of clusters. Upon convergence, the updated cluster centers are read back by the host. Fig. 12 shows the architecture for performing K-means clustering on a 1-D dataset (N=102,400 and number of iterations for convergence of the algorithm T=100) over 64 clusters. In the first pipeline stage, the Euclidean distance between the cluster centers and a data sample is computed. In the second stage, the cluster closest to the sample is inferred and in the third stage, the inferred cluster center is updated according to Eq. 12. It can be inferred that only the pipeline computations are modified in Fig. 3b to implement the clustering algorithm while retaining most of the communication fabric. More importantly, the connection to the middleware design, which is platform-specific and often a time-consuming process, remains unchanged.

For the 1-D case (i.e. d=1) the architecture reduces the computational complexity of the algorithm from *O(NmT)* to *O(NT)*, where m is the number of clusters. This reduction in complexity is attainable as long as there are enough multiplier resources (m multipliers) on the FPGA. The experimental speedup and utilization for K-means clustering algorithm on the Nallatech platform is shown in Table 9. The speedup obtained is considerably low for the extent of parallelism extracted in the algorithm (i.e. all operations required on a data sample are performed in one cycle). The speedup is primarily affected by the overhead time spent in transferring data from the CPU to the FPGA over a relatively small-bandwidth low-efficient interconnect. While only 400 iterations (N_{iter} in the RAT worksheet) of FPGA communication and computation were required for PDF estimation, a total of 100×200 iterations are required for K-means clustering (200 for processing all available data and 100 for algorithm convergence) leading to a considerable increase in

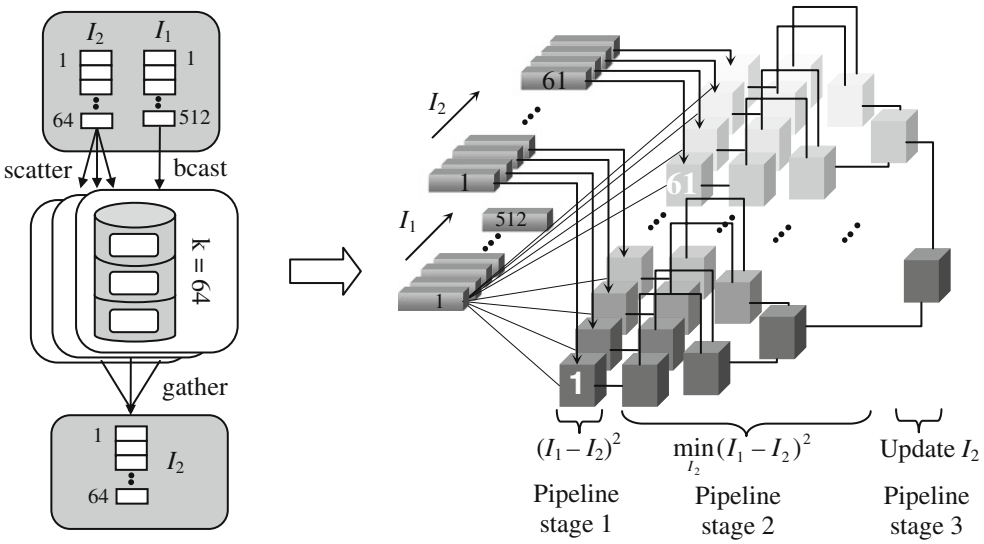


Figure 12 Architecture for K-means clustering where k corresponds to number of clusters, I_1 is the set of data points, and I_2 are the cluster centers.

Table 9 Speedup & utilization for PDF estimation, K-means clustering, and correlation.

Description	Single-core			
	1-D PDF	2-D PDF	K-Means	Correlation
DSP48s (%)	8	16	66	66
BRAM (%)	12	15	9	11
Slices (%)	11	13	9	11
$util_{comm.}$	0.14	0.20	0.18	0.21
RAT predicted speedup	11.5	9.0	4.3	9.6
Actual speedup	7.8	7.1	3.2	8.1

total RC time ($t_{RC}= N_{iter}\times(t_{comm}+t_{comp})$). The $util_{comm}$ factor for K-means clustering is slightly higher in comparison to that obtained for 1-D PDF estimation indicating a longer time spent in data communication.

6.2 Correlation

The architecture developed for PDF estimation is also a good fit for computing correlation on an FPGA. The computations involved in calculating the cross-correlation value at different lags τ can be decomposed and mapped in a similar way as that of PDF estimation for a parallel implementation (see Fig. 13b and 13c).

The similarity in dataflow between the two algorithms cannot easily be deciphered until we analyze a decomposition strategy for Correlation via design patterns. Consider two vectors $X=[x_1,x_2,x_3,\ldots,x_{512}]$ and $Y=[y_1,y_2,y_3,\ldots,y_{512}]$ whose cross-correlation needs to be computed. A number of product terms involved in the computation can be performed in parallel. In particular, the x_i values (I_2) *seed* the parallel datapaths in Fig. 3a while the y_i values (I_1) are fed sequentially in a pipeline flow. Due to resource

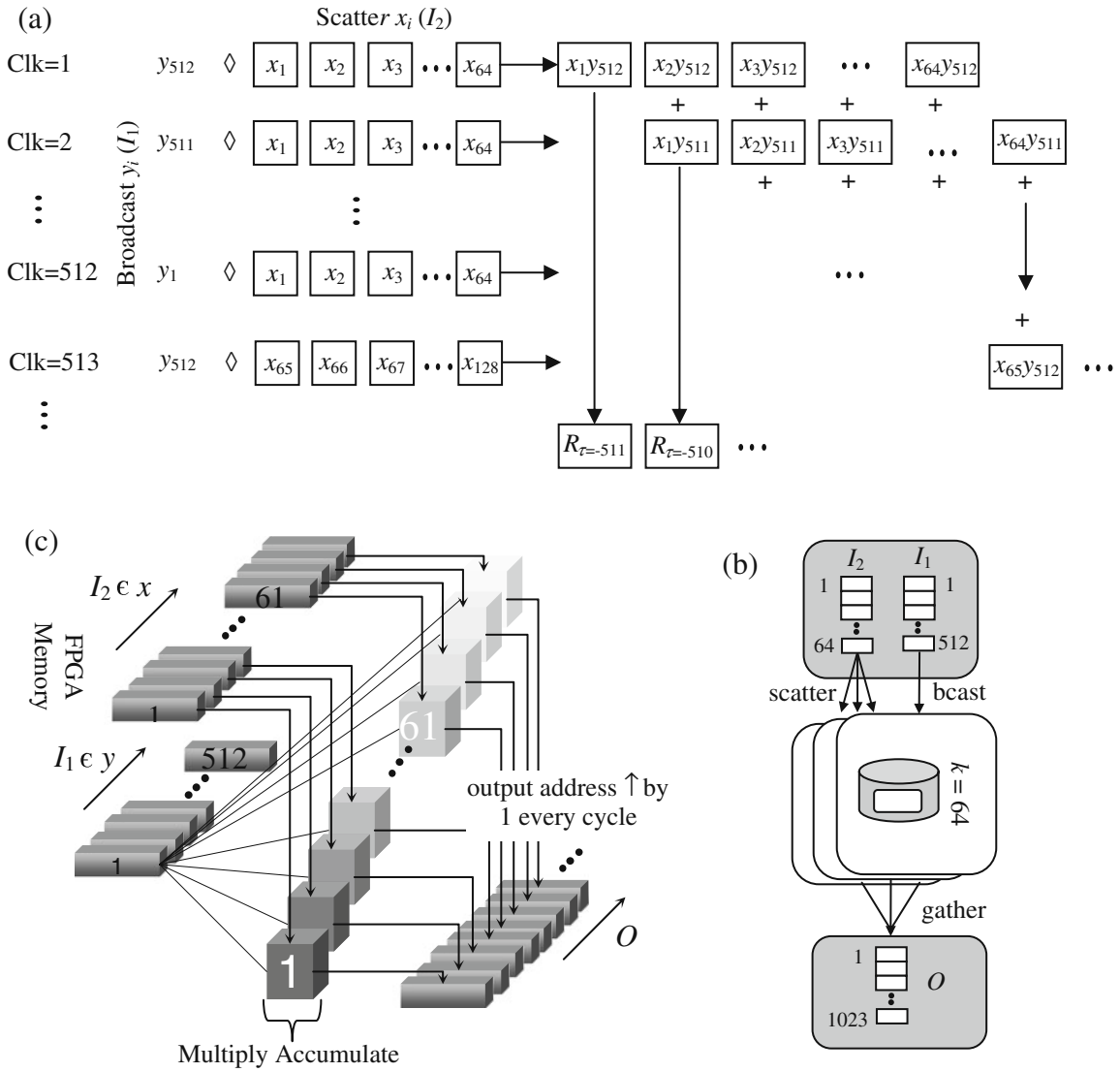


Figure 13 **a** Raster-like computation structure, **b** decomposition, and **c** design of cross-correlation.

limitations (multipliers) on the FPGA, correlation values are computed over 64 parallel datapaths (i.e. in blocks of 64 values of x_i) and accumulated over until all x_i values are accounted for. While accumulating values every cycle, the write address of output memory O is incremented by one to match the raster-like pattern observed in computing correlation at various lags (see Fig. 13a). Since the underlying architecture is scalable and portable (see Sections 5.4.3 and 5.4.4), the designs can be extended to solve larger scale problems (i.e. clustering data points into more clusters/computing correlation over longer datasets) and across various platforms having bigger FPGAs with minimal efforts. The experimental speedup and utilization for the correlation algorithm on the Nallatech platform is shown in Table 9.

7 Conclusions

In this paper we have identified the need for fast and efficient execution of certain principal machine-learning algorithms used often in multimedia analysis tools. In proposing FPGA-based reconfigurable computing as a suitable technology for hardware acceleration, we analyzed various challenges faced while designing and developing algorithms on FPGAs. Algorithm decomposition, performance prediction, and design reuse for parallel implementation all need to be performed in an efficient and structured manner for developing successful FPGA designs in a productive manner. To address these challenges, we proposed an approach for pattern-based decomposition of algorithms for FPGA design and development.

Significant performance improvements in terms of speedup were obtained by using FPGA-accelerated implementation of the Parzen window-based PDF estimation algorithm decomposed using primitive RC design patterns. Dual-core architectures were developed and evaluated on a single-device system by exploiting the scalability in the algorithm. Key design parameters were identified for tuning the architecture to suit different platforms as well. Precision effects were investigated and data verification along with error statistics suggested a sufficient fixed-point configuration for the algorithm. We also validated the benefit of composite patterns for design reuse. With the successful design of a scalable and portable architecture for the PDF algorithm, rapid hardware development for the K-means clustering and correlation algorithms was possible due to the similarity of their underlying design patterns. Further, the work also showcased the benefit of quantifying design patterns in efficiently exploiting a performance prediction tool, RAT, to predict an algorithm's amenability to a hardware platform before undertaking a lengthy development process.

The architecture developed in this work was designed with consideration toward future extensions to multi-FPGA systems. Further research is warranted in understanding the potential improvements that can be achieved and bottlenecks that might have to be addressed while migrating designs to multi-FPGA systems. Directions for future work also include investigating and developing design patterns for solving other problem sets that have underlying similarity in their algorithms. This would be a critical step in enabling FPGA-based RC for solving a general class of computationally intensive problems. This work has shown that one such general architecture exists for a class of machine-learning algorithms.

Acknowledgement This work was supported in part by the I/UCRC Program of the National Science Foundation under Grant No. EEC-0642422. The authors gratefully acknowledge vendor equipment and/or tools provided by Xilinx, Aldec, Cray, and Nallatech that helped make this work possible.

References

1. Camastra, F., & Vinciarelli, A. (2008). Machine Learning for Audio, Image and Video Analysis—Theory and Applications. Springer-Verlag London Limited. (16).
2. Dimitrova, N., Zhang, H., Shahraray, B., Sezan, I., Huang, T., & Zakhor, A. (2002). Applications of Video-content Analysis and Retrieval. *Journal of Multimedia*, *9*(3), 42–55. (21).
3. Lillo, F., Basile, S., & Mantegna, R. N. (2002). Comparative genomics study of inverted repeats in bacteria. *Bioinformatics*, *18* (7), 971–979. (4).
4. Pitie, F., Kokaram, A. C., & Dahyot, R. (2005). N-Dimensional Probability Density Function Transfer and its Application to Color Transfer. Proc 10th International Conference on Computer Vision, Beijing, 1434–1439. Oct. 2005. (8)
5. Kay, S. M., Nuttall, A. H., & Baggenstoss, P. M. (2003). Multidimensional probability density function approximations for detection, classification, and model order selection. *IEEE Transactions on Signal Processing*, *49*(10), 2240–2252. (14).
6. Chang, Y., Zeng, W., Kamel, I., & Alonso, R. (1996). Integrated image and speech analysis for content-based video indexing. Proc Multimedia Computing and Systems, Japan, 306–313. (17)
7. Zhang, H., Zhuang, Y., & Wu, F. (2007). Cross-modal correlation learning for clustering on image-audio dataset. Proc of the 15th International Conference on Multimedia, Augsburg, Germany, 273–276. (18)
8. Goecke, R., Millar, J. B., Zelinsky, A., & Robert-Ribes, J. (2001). Analysis of audio-video correlation in vowels in Australian English. Intl Conf. on Audio-Visual Speech Processing, Aalborg, Denmark, 115–120. (28)
9. Llyas, M. (1987). General probability density function of packet service times for computer networks. *Electronics Letters*, *23*(1), 31–32. (5).
10. Bliss, R. R., & Panigirtzoglou, N. (2002). Testing the stability of implied probability density functions. *Journal of Banking & Finance*, *26*(2), 381–422. (6).
11. Scheicher, M., & Glatzer, E. (2003). Modelling the implied probability of stock market movements. Working Paper Series 212 European Central Bank. (7)
12. Greengard, L., & Strain, J. (1991). The fast gauss transform. *SIAM Journal on Scientific and Statistical Computing*, *12*(1), 79–94. (15).

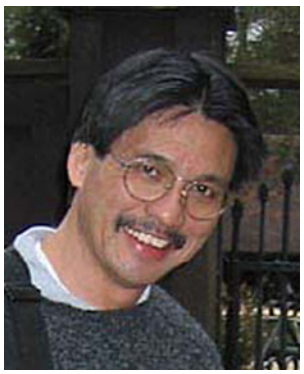
13. Culler, D. E. & Singh, J. P. (1999). Parallel computer architecture: a hardware/software approach. Morgan Kaufmann. (19).
14. Hemmert, K. S., & Underwood, K. D. (2005). An analysis of the double-precision floating-point FFT on FPGAs. IEEE Symp. on Field-Programmable Custom Computing Machines, Washinton, DC, 171–180. Apr. (1).
15. Govindu, G., Choi, S., Prasanna, V., Daga, V., Gangadharpalli, S., & Sridhar, V. (2004). A high-performance and energy-efficient architecture for floating-point based LU decomposition on FPGAs. IEEE Symposium on Parallel and Distributed Processing, Santa Fe, NM, 149. (2). Apr.
16. Jiang, H., Lin, T., & Zhang, H. Video Segmentation with the Assistance of Audio Content Analysis. International Conference on Multimedia and Expo, New York, NY, 1507–1510. (20).
17. Kim, J. S., Mangalagiri, P., Irick, K., Vijaykrishnan, N., Kandemir, M., Deng, L., et al. (2007). TANOR: A Tool for Accelerating N-body Simulations on Reconfigurable Platform. Proc of the 17th Int. Conf. on Field Programmable Logic and Applications, Amsterdam, 68–73. Aug. (9).
18. Leeser, M., Theiler, J., Estlick, M., & Szymanski, J. J. (2000). Design tradeoffs in a hardware implementation of the K-means clustering algorithm. Proc. Of Sensor Array and Multichannel Signal Processing Workshop, 520–524.
19. Frohlich, I., Gabriel, A., Kirschner, D., Lehert, J., Lins, E., Petri, M., et al. (2002). Pattern Recognition in the HADES—Spectrometer: An Application of FPGA Technology in Nuclear and Particle Physics. Proc International Conference on Field-Programmable Technology (FPT), Singapore, 443–444. Dec. (10).
20. Neo, S., Goh, H., Ng, W. Y., Ong, J., & Pang, W. (2007). Real-time Online Multimedia Content Processing: Mobile Video Optical Character Recognition and Speech Synthesizer for the Visual Impaired. Proc Intl. Convention on Rehabilitation Engineering and Assistive Technology, Singapore, 201–206. (22).
21. Schmit, H., & Thomas, D. (1995). Hidden Markov modeling and fuzzy controllers in FPGAs. Proc Symp. on FPGAs for Custom Computing Machines, Napa, CA, 214–221. Apr. (11).
22. VanCourt, T., &Herbordt, M. (2005). Three Dimensional Template Correlation: Object Recognition in 3D Voxel Data. Proc Computer Architecture for Machine Perception, Washinton, DC, 153–158. (12).
23. Holland, B., Nagarajan, K., Conger, C., Jacobs, A., & George, A. D. (2007). RAT: A Methodology for Predicting Performance in Application Design Migration to FPGAs. Proc. of High-Performance Reconfigurable Computing Technologies & Applications Workshop (HPRCTA 2007), SC'07, Reno, NV. Nov. 11. (3).
24. Steffen,C. P. (2007). Parametrization of Algorithms and FPGA Accelerators to Predict Performance. Proc. of Reconfigurable System Summer Institute (RSSI), Urbana, IL, 17–20. (23).
25. DeHon, A., Adams, J., DeLorimier, M., Kapre, N., Matsuda, Y., Naeimi, H., et al. (2004). Design Patterns for Reconfigurable Computing. IEEE Symp on Field Programmable Custom Computing Machines, Napa Valley, CA, 13–23. (24).
26. Gamma, E., Johnson, R., Helm, H., Vlissides, J.M., & Booch, G. (1994). Design Patterns: Elements of Reusable Object-Oriented Software. Addison–Wesley Professional, 416.
27. Anvik, J., MacDonald, S., Szafron, D., Schaeffer, J., Bromling, S., & Tan, K. (2002). Generating Parallel Programs from the Wavefront Design Pattern. Proc Intl Workshop on High-Level Parallel Programming Models and Supportive Environments, Fort Lauderdale, FL, 104–111.
28. Gribbon, K.T., Bailey, D. G., & Johnston, C. T. (2005). Design Patterns for Image Processing Algorithm Development on FPGAs. *IEEE TENCON*, 1–6.
29. Mashor, M. Y. (1998). Improving the Performance of K-means Clustering Algorithm to Position the Centres of RBF Networks. *International Journal of the Computer, the Internet and Management*, *6*(2).

Karthik Nagarajan received the B.E. degree in Electronics and Communication Engineering from the University of Madras, Chennai, India in 2003, and the M.S. degree in Electrical Engineering from the University of Florida, Gainesville in 2005. He is currently a Ph.D. Candidate in the Department of Electrical and Computer Engineering, University of Florida, and a Graduate Research Assistant in the Adaptive Signal Processing Laboratory (ASPL) and the NSF Center for High-performance Reconfigurable Computing (CHREC). His research interests include pattern recognition, information theory, graphical models and multiscale estimation concepts. He is also an active researcher in the field of high performance and reconfigurable computing on FPGAs with focus upon design and development, performance analysis and productivity improvements.

Brian Holland is a fourth year Ph.D. student at the University of Florida and a research assistant at the NSF Center for High-performance Reconfigurable Computing (CHREC). He received his B.S. degree in computer engineering from Clemson University in 2005 and his M.S. degree in computer engineering from the University of Florida in 2006. His research interests include reconfigurable computing, high-level languages for FPGA design, and strategic methods for parallel algorithm design and migration.

Alan D. George is Professor of Electrical and Computer Engineering at the University of Florida, where he serves as the Director of the NSF Center for High-performance Reconfigurable Computing (CHREC) and the High-performance Computing and Simulation (HCS) Research Laboratory. He received the B.S. degree in Computer Science and the M.S. in Electrical and Computer Engineering from the University of Central Florida, and the Ph.D. in Computer Science from the Florida State University. Dr. George's research interests focus upon high-performance architectures, networks, systems, and applications in reconfigurable, parallel, distributed, and fault-tolerant computing. He is a senior member of IEEE and SCS, a member of ACM and AIAA, and can be reached by e-mail at *ageorge@ufl.edu*.

K. Clint Slatton received B.S. and M.S. degrees in aerospace engineering and M.S. and Ph.D. degrees in electrical engineering, all from the University of Texas (UT) in Austin, TX in 1993, 1997, 1999, and 2001, respectively. From 2002 to 2003, he was a Postdoctoral Fellow with the Center for Space Research at UT, where he worked on multiscale data fusion techniques for interferometric synthetic aperture radar (InSAR) and airborne laser swath mapping (ALSM) measurements. He also worked at the NASA Jet Propulsion Laboratory in the Radar Sciences Section and was a recipient of the NASA Graduate Student Research Program Fellowship. He is currently an Assistant Professor with the Department of Electrical and Computer Engineering and the Department of Civil and Coastal Engineering at the University of Florida in Gainesville, FL. He is the Director of the Adaptive Signal Processing Laboratory and a coinvestigator for the National Center for Airborne Laser Mapping (NCALM), for which he develops information-theoretic segmentation methods for ALSM data in complex environments, such as forests. He was named a 2006 winner of the Presidential Early Career Award for Scientists and Engineers (PECASE) for work on predicting signal propagation in highly cluttered environments using remotely sensed geometry from ALSM. His research interests include remote sensing applications of ALSM and InSAR, high dimensional data segmentation, multiscale data fusion, graphical models, and photonics, with sponsored projects from NSF, NASA, the US Army, the US Navy, and NOAA. Dr. Slatton is a member of the AGU and also of Sigma Xi, Tau Beta Pi, and Sigma Gamma Tau.



Herman Lam is an Associate Professor of Electrical and Computer Engineering at the University of Florida. He has over 20 years of research and development experience in the area of database management, service-oriented architecture, and distributed computing. Currently, his work is focused on reconfigurable computing and is a research member of the NSF Center for High-Performance Reconfigurable Computing (CHREC). He has authored or co-authored over 85 refereed journal and conference articles and one textbook. In past several years, Dr. Lam has been on the program committees of the International Conference on Web Services (ICWS) and the International Conference on Web Information Systems Engineering and a reviewer for a number of workshops and conferences, including International Workshop on High-Performance Reconfigurable Computing Technology and Applications (HPRCTA'08) and 2008 International Conference on ReConFigurable Computing and FPGAs. He is a member of the editorial board of International Journal of Business Process Integration and Management.