12

FPGA Solutions for Big Data Applications

12.1 Introduction

We live in an increasingly digitized world where the amount of data being generated has grown exponentially – a world of Big Data (Manyika *et al.* 2011). The creation of large data sets has emerged as a hot topic in recent years. The availability of this valuable information presents the possibility of analyzing these large data sets to give a competitive and productivity advantage. The data come from a variety of sources, including the collection of productivity data from manufacturing shop floors, delivery times, detailed information on company sales or indeed the enormous amount of information currently being created by social media sites. It is argued that by analyzing social media trends, it should be possible to create potentially greater revenue generating products.

Big Data analytics (Zikopoulos *et al.* 2012) is the process by which value is created from these data and involves the loading, processing and analysis of large data sets. Whilst database analytics is a well-established area, the increase in data size has driven interest in using multiple distributed resources to undertake computations, commonly known as *scaling out*. This is achieved using a process known as MapReduce (Dean and Ghemawat 2004) which helps the user to distribute or *map* data across many distributed computing resources to allow the computation to be performed, and then bringing all of these computed outputs together, or *reducing*, to produce the result. Apache™ Hadoop® (Apache 2015) is an open source resource for achieving this.

Given this distributed nature of processing, the authors could therefore be accused of being opportunistic in including this hot topic in a book dedicated to implementation of DSP systems using FPGA technology, but there is sound reasoning for doing so. Data analytics comprises the implementation of computationally complex algorithms, and for classes of algorithms that cannot be *scaled out* there is a need to improve processing within the single computing unit. This is known as *scaling up* and requires the realization of efficient, scalable hardware to achieve this functionality. As will be demonstrated in this chapter, the process is similar to many of those applied to signal processing examples.

There is certainly a strong case, increasingly targeted at FPGA, for developing scaled-up solutions for a number of key data mining tasks, specifically classification, regression

FPGA-based Implementation of Signal Processing Systems,
Second Edition. Roger Woods, John McAllister, Gaye Lightbody and Ying Yi.

and clustering. Such algorithms specifically include decision tree classification (DTC), artificial neural networks (ANNs) and support vector machines (SVMs). There is a strong case for developing FPGA-based solutions to achieve scaling up as performance and especially power will become increasingly important. This chapter considers the implementation of such algorithms and, in particular, the *k*-means clustering algorithm.

Big data concepts are introduced in Section 12.2. Details are given on Big Data analytics and various forms of data mining are introduced in Section 12.3. In Section 12.4, the acceleration of Big Data analytics is discussed and the concepts of scaling up and scaling out are introduced. The case for using FPGAs to provide acceleration is made and a number of FPGA implementations reviewed, including the acceleration of the Heston model for determining share options. The computation of *k*-means clustering is then described in Section 12.5. The idea of using processors to make the heterogeneous FPGA-based computing platform more programmable is introduced in Section 12.6 and then applied to *k*-means clustering in Section 12.7. Some conclusions are given in Section 12.8.

12.2 Big Data

The number of digital information sources continues to grow as we look to digitize all sorts of information. These sources range from output from social media, storage of text information from mobile phones, personal digitized information (e.g. medical records), and information from the increasing number of security cameras. This growth of information as been labeled Big Data and is recorded in terms of exabytes (10^{18}) and zettabytes (10^{21}).

Of course, not only has the term "Big Data" emerged to define this new type of information, but marketing forces have pushed for the definition of its various forms. Big Data has been classified in terms of a number of characteristics namely, volume, velocity, variety, veracity and value, the so-called "five Vs." Some of the key features of these are outlined below.

- *Volume*, of course, refers to the amounts of data being generated. Whether this is social media data from emails, tweets etc. or data generated from sensor data (e.g. security cameras, telemetric sources), there is now a need to store these zettabytes or even brontobytes (10^{27}) of information. On Facebook alone, 10 billion messages are sent per day! Such data sets are too large to store and analyze using traditional *structured* database technology, and so there is a push to store them in *unstructured* form using distributed systems.
- *Velocity* refers to the rate at which these new data sets are generated and distributed. In our new era of instant financial transactions and smartphone connectivity, there is a need for immediate response (within seconds). This has major consequences for not only the computing infrastructure but also the communications technology to ensure fast low- latency connectivity. Lewis (2014) relates the major activity and cost involved in creating a fiber link between Chicago and New York just to shave several milliseconds off the latency for financial markets!
- *Variety* refers to the different flavors of data, whether it be social media data which may be incomplete, transitory data or even financial data which have to be secure.

Whilst structured data databases would have been used in the past for storing information, it is estimated that 80% of the world's data is now unstructured, and therefore cannot be put easily into conventional databases.

- *Veracity* refers to the trustworthiness of the data. Social media data are transitory and less reliable as they may be incorrect (and possibly deliberately so) and of poor quality, whereas security camera information is inaccurate but possibly of low quality or low information content. The challenge is then to develop algorithms to cope with the quality of data and possibly use volume as a means of improving the information content.
- *Value* is probably the most relevant data characteristic as it represents the inherent worth of the information. There is no doubt that it represents the most important aspect of Big Data as it allows us to make sense of it. However, a major challenge is to be able to extract the value from the data which is central aspect of Big Data analytics.

These are the challenges for Big Data analytics: it may be useful to have a high volume of valuable information that has strong veracity, but this is only useful if we can make sense of the information. Thus whilst Hadoop may provide an infrastructure for storing and passing information for processing, it is the implementation of very complex analytics that is critical. Note that McNulty (2014) talks about Big Data in terms of the "seven Vs," adding variability and visualization to our list above.

12.3 Big Data Analytics

The availability of such a rich form of data now presents the possibility of identifying important value. For example, insurance companies currently employ statistical models on the multiple sources of information available, including previous premium prices and even on-media and spending habits, to work out appropriate and acceptable insurance premiums! Of course, considerable potential also exists for analyzing Big Data for marketing reasons, and this represents a key driver for many of the data analytics algorithms. With 1.2 billion people using apps, blogs and forums to post, share and view content, there is a considerable body of available information.

Another very relevant area is *security*. With terrorism being a regular issue in our world today, new forms of technology are increasingly being used by terrorists to communicate with each other; these range from using mobile phones to entering information on social media sites. Extremist and terrorist groups use the internet for a wide variety of purposes, including dissemination of propaganda, recruitment, and development and execution of operational objectives.

As the volume and velocity of social media data rise exponentially, cyber threats are increasing in complexity, scale and diversity. Social media intelligence (SOCMINT) is an emerging science that aims to address this challenge through enhanced analytics that can present a step-change in a defense intelligence community's ability to instantaneously classify and interpret social media data, identify anomalies and threats, and prevent future attacks (Omand *et al.* 2012).

It is therefore clear that Big Data analytics is a growing field of study and seems to involve applying existing and new algorithms to make sense of data. It is argued that this is not simply statistics or the result of applying data mining algorithms and that the

challenges of Big Data require the development of new forms of algorithms. To this end, the term "Big Data scientist" has been used to describe a specialist who is able to develop new forms of the most suitable statistical and data mining techniques to data analysis. The key is to create value from Big Data, and this presents a new computing problem as it involves analysis of data sets that are at least of the order of terabytes.

With the increase in data size, the concept of using multiple distributed resources to undertake Big Data computations is now becoming commonplace. A key development was the creation of MapReduce, which is a programming model to allow algorithms and large data sets to be distributed on a parallel, distributed computing platform and then brought back together to produce the result. The open source manifestation of MapReduce has been through Apache™ Hadoop® (Apache 2015). This is one of the earliest open source software resources for reliable, scalable, distributed computing. A number of evolutions have occurred including a structured query language (SQL) engine, *Impala* (Cloudera 2016), a generic scheduler, *Yarn*, and API file formats, namely *Crunch*.

A lot of hype surrounds Big Data, but there are also a number of major technical challenges in terms of acquiring, storing, processing and visualizing the data. From an FPGA perspective, there is a clear need to implement complex data processing algorithms in a highly parallel and pipelined manner.

12.3.1 Inductive Learning

The key aspect in many Big Data applications is that the user has to learn from data in cases where no analytical solutions exist but the data are used to construct an empirical solution (Abu-Mostafa *et al.* 2012). This is typically known as *inductive learning* and is a core area of machine learning. It involves the user spotting patterns in the information and generalizing them, hoping that they are correct but providing no guarantee that the solution will be valid. It is the source of many data mining algorithms and probably is the area where machine learning and data mining intersect. A query can then be applied to deduce a possible answer from the data (see Figure 12.1).

The key focus is to create chunks of knowledge by applying specific algorithms about some domain of interest which is presented by the data to be analyzed; this will usually be capable of providing an answer by transcending the data in such a way that the answer cannot just be provided by extracting and aggregating value from the data, i.e. creating a model that represents the data. These predictive models are the core of data mining and usually involve the application of techniques to transform the data to create the model; the model is then easier to apply as only the attributes most useful for model creation need be used and links combined to create new models to provide a better prediction.

Usually the process involves looking at a domain of the data (e.g., financial transactions, hospital bed occupancy rates) which may in many cases be incomplete and described by a set of features (Marshall *et al.* 2014). A data set is typically a subset of the domain described by a set of features; the goal is to apply a set of data mining algorithms to create one or more models from the data.

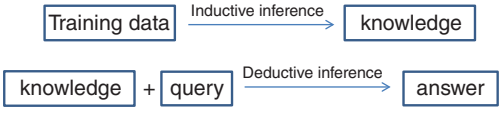


Figure 12.1 Inference of data. Source: Cichosz 2015. Reproduced with permission of John Wiley & Sons.

12.3.2 Data Mining Algorithms

The three most widely used data mining tasks, *classification*, *regression* and *clustering*, can be considered as *inductive learning* tasks (Cichosz 2015). These aim to make sense of the data:

- *Classification*. Prediction of a discrete target attribute by the assignment of the particular instance to a fixed possible set of classes, which could, for example, involve detecting odd behavior in credit card fraud.
- *Regression*. Prediction of a numerical target attributes based on some quantity of interest, for example, working out the length of stay distribution of geriatric patients admitted to one of six key acute hospitals (Marshall *et al.* 2014).
- *Clustering*. Prediction of the assignment of the instances to a set of similarly based clusters to determine the best cluster organization, i.e. similarity of data in a cluster.

12.3.3 Classification

Classification involves assigning instances X from a specific domain and is defined by a set of features which have been into a set of classes i.e. C. A simplified process is illustrated in Figure 12.2, and shows how classifier H_1 maximally separates the classes, while H_2 does not separate them. This classification process is known as a *concept* c and is defined by $X \rightarrow C$. A classification model or classifier $h : X \rightarrow C$ then produces class predictions for all instances $x \in X$ and is supposed to be a good approximation of the target concept c on the whole domain.

One way of looking at data for classification purposes is to create contingency tables or effectively histograms of the data. This is done by picking k attributes from the data set, namely $a_1, a_2, \ldots, a_k$ and then for every possible combinations of values, $a_1 = x_1, a_2 = x_2, \ldots, a_k = x_k$, recording how frequently that combination occurs. Table 12.1 shows how we can compare the school age of children against their choice of subject for a group aged between 14 and 18. Typically, on-line analytical processing (OLAP) tools can be used to view slices and aggregates of these contingency tables. Of course, these tables will have many more parameters and will comprise many more dimensions.

Classification algorithms comprise DTC, ANNs, Bayesian classifiers and SVM. DTC is carried out in two steps: a decision tree model is built up using records for which the category is known beforehand, then it is applied to other records to predict their class affiliation. They are attractive as they provide high accuracy even when the size of

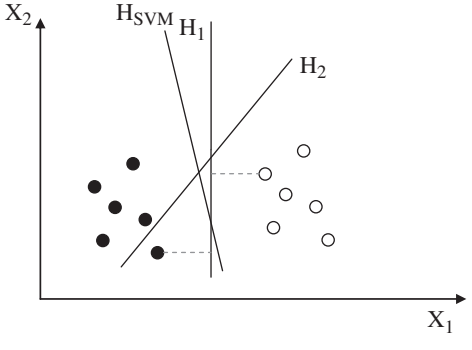


Figure 12.2 Simple example of classification

Table 12.1 Age of school children versus most popular subject

Age	Maths	English	Science	Language (α)
14	14	6	7	9
15	9	7	10	9
16	11	4	8	11
17	12	8	8	5
18	7	9	5	11

the data set increases (Narayanan *et al.* 2007). They are used for detecting spam e-mail messages and categorizing cells in MRI scans. They yield comparable or better accuracy when compared to ANNs.

ANNs comprise parallel and distributed networks of simple processing units interconnected in a layered arrangement, typically of three layers. They are based on the neural operation of the brain and are trained using some initial data. There was a lot of interest in the 1980s and 1990s in this approach, but it fell short of its potential. However, there has now been a resurgence of interest in a form of ANNs called convolutional neural networks (CNNs) (Krizevsky 2014). CNNs comprise layers of alternative local translation-invariant features, followed by switchable task-specific loss functions which are trained with stochastic gradient descent. The parameters are very large and represent a significant training challenge, but they show considerable potential. They are highly amenable to FPGA implementation due to the considerable routing resources that FPGAs offer.

Bayesian classifiers are a family of simple probabilistic classifiers based on applying Bayes' theorem which relates current probability to the prior probability. They tend to be useful in applications where the underlying probability has *independent* parameters. For example, clementine oranges may be characterized by being orange in color, squishy and 5 cm in diameter, but these features are considered to be independent of each other. So even if these parameters are interdependent, a naive Bayes classifier considers all of these properties to independently contribute to the probability that this fruit is a clementine orange!

An SVM formally constructs a hyperplane or set of hyperplanes in a high- or infinite-dimensional space, which can be used for classification, regression, or other tasks. Intuitively, a good separation is achieved by the hyperplane that has the largest distance to the nearest training data point of any class. This would be given as the hyperplane H_{SVM} in Figure 12.2 which would seem to best split the two groups by providing the best distance as well.

12.3.4 Regression

Regression is also an inductive learning task that can be thought of as *classification with continuous classes*, which means that the regression model predicts numerical values rather than discrete class labels (Cichosz 2015). It tends to be thought of in the classical statistical approach where we are trying to fit a regression model to a set of data, but if we view it as an algorithm applied in a data mining context, then we can think of regression as being able to provide numerical prediction. This leads to the development of a series of algorithms which can be used to predict the future demand for a product, volume of sales, or occupancy of beds in a hospital ward (Marshall *et al.* 2014).

In mathematical terms, for a target function f, $X \rightarrow \Re$ represents the true assignment of numerical values to all instances from the domain. This can be determined using a training set $T \subseteq D \subset X$ for regression which will contain some or all of the labeled instances for which the target set is available. The key is to find the relationship between the dependent and independent variables. The value of dependent variable is of most importance to researchers and depends on the value of other variables. Independent variables are used to explain the variation in the dependent variable.

Regression is classified into two types: *simple regression*, with one independent variable; and *multiple regression*, which has several independent variables. In simple regression, the aim is to create a regression equation involving several regression coefficients and then determine a best fit using the data; this involves determining the best linear relationship between the dependent and the independent variables. In multiple regression analysis, the coefficients indicate the change in dependent variables assuming the values of the other variables are constant. A number of tests of statistical significance are then applied, one of which is the F-test.

12.3.5 Clustering

A cluster is a group of objects that belong to the same class, so clustering is the process of making a group of abstract objects into classes of similar objects. It is applied to a broad range of applications such as market research, pattern recognition and image processing. Its main advantage over classification is that it is adaptable to changes and helps single out useful features that distinguish different groups.

Clustering methods can be classified as partitioning, hierarchical, density-based, grid-based, model-based and constraint-based. For a database of n objects, the partitioning method constructs $k \leq n$ partitions of the data, each of which will represent a cluster and where each group must contain an object and each object must belong to one group only. A well-known method is called k-means clustering (see Section 12.5).

12.3.6 The Right Approach

Whilst this section has highlighted a number of machine learning algorithms, it is important to identify the right estimator for the job. The key observation is that different estimators are better suited to different types of data and different problems. The scikit website (http://scikit-learn.org/stable/tutorial/machine_learning_map/index.html) provides a useful indication of how to progress from a data perspective with Python open source files.

The first stage is to determine if the user is looking to predict a category. If so, then classification or clustering will be applied; otherwise the user will look to apply regression and/or dimension reduction. If the categories and labels are known, then the user will apply classification; if the labels are not known, then the user will apply clustering to determine the organization.

If applying classification to a small data set (i.e. less than 100,000 samples), then a stochastic gradient descent learning routine is recommended; otherwise other classifiers such as SVMs are recommended (Bazi and Melgani 2006). If the data are labeled, then the choice of clustering will depend on the number samples. If the categories are known then various forms of k-means clustering are applied, otherwise it is best to apply a form of a Bayesian classifier.

Of course, this does not represent an expert view for choosing the right approach, and the main purpose of this section is just to provide an overview. We highlight some of the algorithmic characteristics which best match FPGAs; in particular, we are interested in the highly parallel computation but also the high level of interconnection of ANNs as these match well the numerous routing capabilities of modern FPGAs. This is a particularly attractive feature as this algorithmic interconnect would have to be mapped as multiple memory accesses in processors.

12.4 Acceleration

MapReduce involves scaling out the computation across multiple computers. Thus, it is possible to use resources temporarily for this purpose, and vendors such as Amazon now allow you to hire resources, with a higher premium being charged for more powerful computers in November 2016, Amazon announced a new resource called EC2 which allows access to FPGA resources (Amazon 2016). Figure 12.3 illustrates the process. For an original computation as illustrated in Figure 12.3(a), Figure 12.3(c) shows how it is possible to use Hadoop® to *scale out* the computation to improve performance. This works well for problems that can be easily parallelized and distributed, such as performing multiple searches on a distributed search engine.

12.4.1 Scaling Up or Scaling Out

It is also possible, however, to *scale up* computation as shown in Figure 12.3(b). For example, you may be performing multiple, highly complex operations on a single data set, in which case it makes more sense to scale up the resource as you then avoid the long communications delay involved in communicating between distributed computers. This is particularly relevant for highly computationally complex algorithms and also problems which cannot be easily parallelized invoking high volumes of communications which act to slow down computation. The issue of whether to scale out or scale up is a detailed decision (Appuswamy *et al.* 2013).

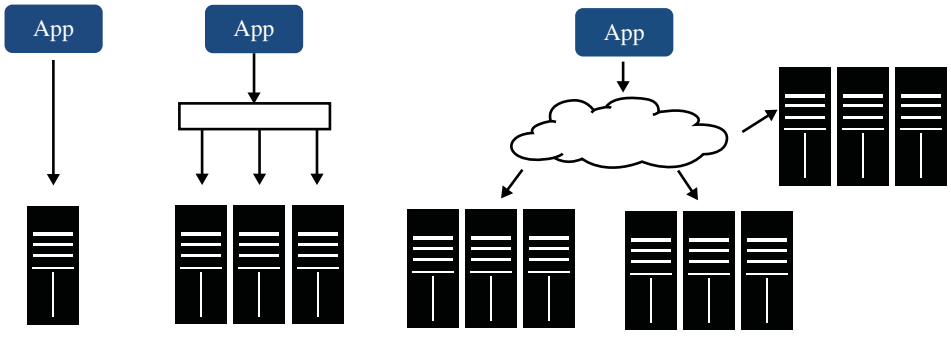


Figure 12.3 Scaling computing resources

12.4.2 FPGA-based System Developments

A number of major system developments have occurred which strongly indicate the potential of FPGA technology in new forms of computing architectures and therefore Big Data applications. These include the heterogeneous platform developments by Microsoft, Intel and IBM. To date, these have largely been driven by FPGA companies or academia, but there is interest among large computing companies driven mainly by the energy problems as indicated in Chapter 1 with the increased computing demand. Intel's purchase of Altera (Clark 2015) clearly indicates the emergence of FPGAs as a core component in future data centers.

Microsoft has developed flexible acceleration using FPGAs called the Catapult fabric which was used to implement a significant fraction of Bing's ranking engine. It showed an increased ranking throughput in a production search infrastructure by 95% at comparable latency to a software-only solution (Putnam *et al.* 2014). The Catapult fabric comprises a high-end Altera Stratix V D5 FPGA along with 8 GB of DRAM embedded into each server in a half-rack of 48 servers. The FPGAs are directly wired to each other in a 6×8 two-dimensional torus, allowing services to allocate scaling of the FPGA resources.

The authors implement a *query* and *document* request by getting the server to retrieve the document and its metadata and form several metastreams. A "hit vector" is generated which describes the locations of query words in each metastream; a tuple is also created for each word in the metastream that matches a query and describes the relative offset from the previous tuple, the matching query term, and a number of other properties. The frequency of use can then be computed, along with free-form expressions (FFEs) which are computed by arithmetically combining computed features. These are then used in a machine learning model which determines the document's position in the overall ranked list of documents returned to the user.

IBM and Xilinx have worked closely together to develop Memcache2, a general-purpose distributed memory caching system used to speed up dynamic database-driven searches (Blott and Vissers 2014). By developing a solution that allows tight integration between network, computer and memory and by implementing a completely separate TCP/IP stack, they are able to achieve an order-of-magnitude speed improvement over an Intel Xeon® solution. This improvement is even greater when considering power, which is a key issue in data centers.

12.4.3 FPGA Implementations

There have been a number of classification and regression implementations on FPGAs. Among these are a number of ANNs, including work implementing a general regression neural network (GRNN) used for iris plant and thyroid disease classification (Polat and Yıldırım 2009). They have developed an FPGA implementation using VHDL-based tools; it comprises summation, exponential, multiplication and division operations. However, the inability to realize the Taylor series efficiently in FPGA has meant that the implementation would not run much faster than the MATLAB model running in software on a P4 3 GHz, 256 MB RAM personal computer compared to a fixed-point Xilinx Spartan3 xc3s2000.

An FPGA-based coprocessor for SVMs (Cadambi *et al.* 2009) implemented on an off-the-shelf PCI-based FPGA card with a Xilinx Virtex-5 FPGA and 1 GB DDR2 memory gave an improvement of 20 times over a dual Opteron 2.2 GHz processor CPU with lower power dissipation. For training, it achieved end-to-end computation speeds of over 9 GMACs, rising to 14 GMACs for SVM classification using data packing. This was achieved by exploiting the low precision and highly parallel processing of FPGA technology by customizing the algorithm for low-precision arithmetic. This allowed the efficient reuse of the underlying hardware and reduction in off-chip memory accesses by packing multiple data words on the FPGA memory bus.

A number of implementations have been explored for implementing k-means clustering on FPGA. Lin *et al.* (2012) implemented an eight-cluster XC6VLX550T design with a clock frequency of 400 MHz; the design utilized 112 DSP blocks, 16 BRAMs, 2110 slices, 5337 LUTs and 8011 slice registers. A number of blocks were implemented for performing the distance calculations in parallel, one for each cluster. In Winterstein *et al.* (2013), k-means clustering was performed in FPGA without having to involve off-chip memory. A Xilinx XC4VFX12 with 5107/5549 slices, 10,216 LUTs and a maximum clock frequency of 63.07 MHz was achieved. It gave a speedup of 200 times over a MATLAB realization on a GPP Intel core 2 DUO E8400 and 3 GB RAM, and 18 times over GPU Nvidia GeForce 9600m GT graphics.

12.4.4 Heston Model Acceleration Using FPGA

The Heston model is a well-known model used in option determination in finance applications which involves calculating the risk for cases where volatility is stochastic (Heston 1993). The volatility of the underlying asset follows a Brownian motion, which in turn gives rise to a system of two stochastic differential equations:

$$dS_t = \mu S_t dt + \sqrt{V_t} S_t dW_t, \tag{12.1}$$

$$dV_t = \kappa(\theta - V_t)dt + \xi\sqrt{V_t} dW_t. \tag{12.2}$$

In these equations, S_t is the price variation, V_t is the volatility process, W_t is the correlated Brownian motion process and ξ is referred to as the volatility of the volatility. V_t is a square root mean-reverting process with a long-run mean of θ and a rate of mean reversion of κ. The mean reversion of the volatility means that the volatility is bound to revert to a certain value; so when $V_t < \theta$, the drift of the volatility encourages V_t to grow again, and conversely when $V_t > \theta$, the drift becomes negative and thus the volatility decreases.

The Heston model works well in many financial applications as the asset's log-return distribution is non-Gaussian and is characterized by big tails and peaks. It is argued that equity returns and the implied volatility are negatively correlated, which presents problems for models such as the Black–Scholes model (Black and Scholes 1973), which do not consider volatility, hence the interest in the Heston model.

A useful metric for measuring the performance of different implementations of the Heston model is the number of steps per second achieved by each technology. In this model, the cores implement a Monte Carlo simulator and the output of these is aggregated to form the final Heston output (Figure 12.4). The performance is dictated by the number of cores that can be implemented in the FPGA fabric; the achievable clock speed

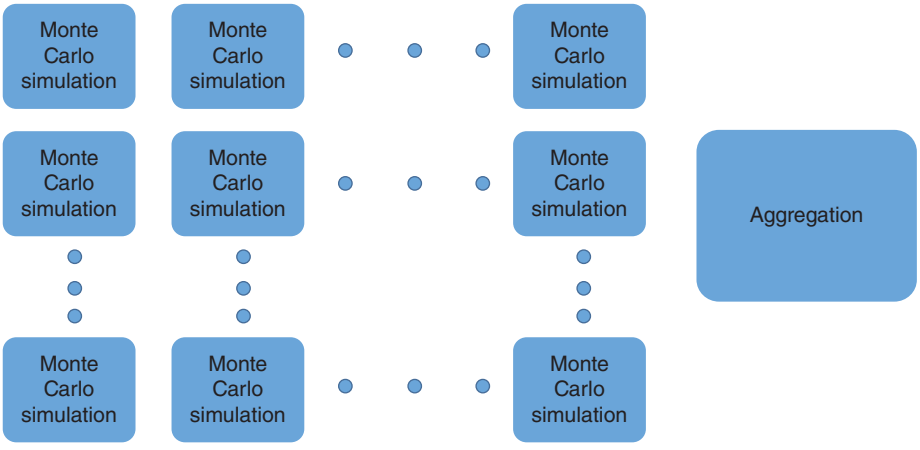


Figure 12.4 Core configuration where Monte Carlo simulations can be programmed

will vary depending on the number of cores that can be effectively placed and routed in the FPGA. A number of possible FPGA configurations have been implemented and are displayed in Table12.2. Performance is given as the throughput rate (TR) which is calculated in billions of steps per second (BSteps/s).

12.5 *k*-Means Clustering FPGA Implementation

Clustering is a process used in many machine learning and data mining applications (Jain 2010). It is an unsupervised partitioning technique which groups data sets into subsets by grouping each new data into groups with the have data points with similar features (e.g. same age groups, same image features). A flow diagram for the algorithm is shown in Figure 12.5. It is used in a range of image processing and target tracking applications (Clark and Bell 2007), when it is necessary to initially partition data before performing more detailed analytics.

The k-means algorithm requires the partitioning of a D-dimensional point set $X = \{x_j\}, j = 1, \ldots, N$, into clusters $S_i, i = 1, \ldots, k$, where k is provided as a parameter, usually

Table 12.2 32-bit Heston model implemented as both fixed- and floating-point

		Performance				
Data type	**MCs**	**LUTs**	**Flip-flops**	**DSP48E**	**Clock (MHz)**	**TR (BSteps/s)**
Fixed	64	57,757 (13%)	65,210 (8%)	320 (9%)	250	16.0
	128	64,865 (15%)	79,676 (9%)	640 (18%)	238	30.5
	256	78,061 (18%)	91,573 (11%)	640 (18%)	172	44
Floating	32	240,801 (56%)	366,642 (42%)	1280 (36%)	112	3.6

Note. MCs = Monte Carlo simulations. Percentages give the utilization of Xilinx Virtex-7 VC709 device.

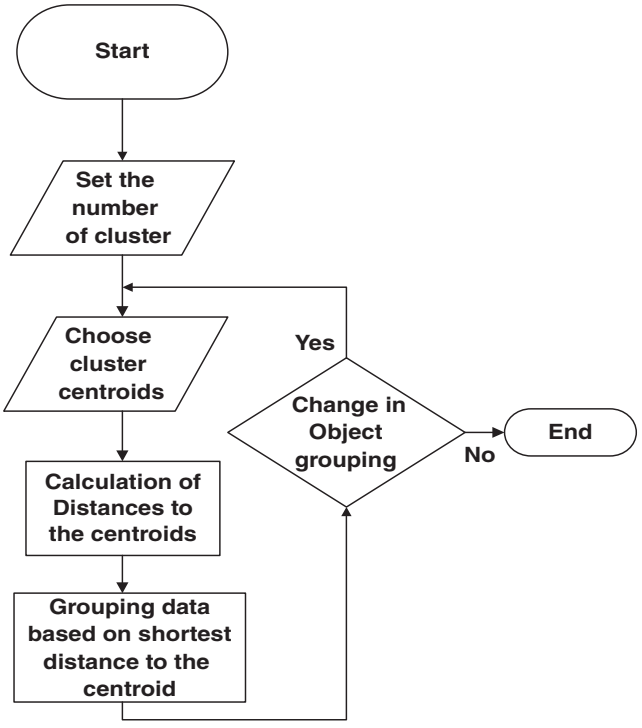


Figure 12.5 Flow chart for *k*-means algorithm

the number of required data sets, and is usually set by the user (Winterstein *et al.* 2013). The goal is to find the optimal partitioning which minimizes the objective function

$$J(\{S_i\}) = \sum_{i=1}^{k} \sum_{x_j \in S_i} \|x_j - \mu_i\|^2, \tag{12.3}$$

where μ_i is the geometric center (centroid) of S_i.

The ideal scenario for data organization is to group data of similar attributes closer together and farther away from data of dissimilar attributes. The *k*-means algorithm is one of the main unsupervised data mining techniques used to achieve this for large data sets (Hussain *et al.* 2011). In the *k*-means algorithm, a data set is classified into *k* centroids based on the measure of distances between each data set and the *k* centroid values (see Figure 12.6).

At the beginning, the number of centroids and their centers are chosen; each data item then belongs to the centroid with the minimum distance to it. There are many metrics for calculating distance values in the *k*-means algorithm, but the most commonly used ones are the Euclidean and Manhattan distance metrics. The Euclidean distance, D_E, is given by

$$D_E = \sqrt{\sum_{i=1}^{d} (X - C)^2} \tag{12.4}$$

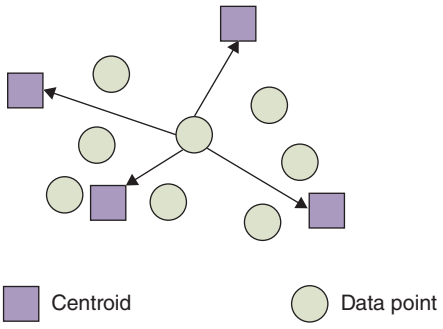


Figure 12.6 Distance calculation

where X is the data point, C is the cluster center and d is the number of dimensions of each data set. The Manhattan distance D_M is given by

$$D_M = \sum_{i=1}^{d} |X - C|. \tag{12.5}$$

Whilst the Euclidean distance metric is more accurate (Estlick *et al.* 2001), the Manhattan distance metric is preferred as it is twice as fast as the Euclidean distance calculation and consumes less resources (Leeser *et al.* 2002).

12.5.1 Computational Complexity Analysis of *k*-Means Algorithm

The stages involved in k-means algorithm are: distance calculation, comparison and averaging, as shown in Figure 12.7. The centroid values are chosen from among the existing data points/pixels or by generating random values and can be viewed as negligible from a computational analysis point of view. In the distance stage, the distances from each data point to the centroids are calculated. For each data point of an RGB image, the Manhattan distance metric is given by

$$D = |X_r - C_r| + |X_g - C_g| + |X_b - C_b|. \tag{12.6}$$

This involves 3 absolute values, 2 additions and 3 subtractions, giving rise to 8 operations. For n data points and k centroids, the number of operations involved in the distance calculation, k_D, is given by $k_D = 8nk$.

In the comparison module, the inputs are k distance values generated by each pixel. It takes $k - 1$ comparison steps to get the minimum distance. So for n data, the number of operations involved in the comparison block is given by $k_C = n(k - 1)$.

In the averaging block, data pixels in the dimension are added up and divided by the number in their dimensions in that cluster, giving an updated centroid value for the following frame. For a data pixel there are d additions, so for n data there are nd additions

Figure 12.7 Bits of data in and out of each block

and kd divisions. Hence the number of operations involved in the averaging block, K_A, is given by

$$K_A = nd + kd = d(n + k) = 3(n + k) \tag{12.7}$$

($d = 3$ for an RGB image).

The total number of operations, K_{total}, in the k-means algorithm is thus given by

$$K_{\text{total}} = 8nk + n(k - 1) + 3(n + k). \tag{12.8}$$

Figure 12.7 shows the number of operations and bits in and out of each block for 8 clusters. The number of operations in the distance calculation block is the highest and increases as the number of clusters increases. The operations in the distance calculation block are independent and as such can be put in parallel to speed up execution.

12.6 FPGA-Based Soft Processors

As some of the examples earlier in this book and even those reviewed in this chapter have indicated, FPGAs give a performance advantage when the user can develop an architecture to best match the computational requirements of the algorithm. The major problem with this approach is that generating the architecture takes a lot of design effort, as was illustrated in Chapter 8. Moreover, any simple change to the design can result in the creation of a new architecture which then incurs the full HDL-based design cycle which can be time-consuming. For this reason, there has been a lot of interest in FPGA-based software processors.

A number of FPGA-based image processors have been developed over the years, including the Xilinx MicroBlaze (Xilinx 2009) and the Altera Nios II processor (Altera 2015), both of which have used extensively. These processors can be customized to match the required applications by adding dedicated hardware for application-specific functions. The approach is supported by the FPGA company's software compilers. However, attempts to make the processor more programmable compromises the performance and have not taken advantage of recent technological FPGA developments.

A number of other processor realizations have been reported, including a vector processing approach (Russell 1978) which uses fixed, pipelined functional units (FUs) that can be interconnected; this takes advantage of the plethora of registers available in FPGAs. A soft vector processor VESPA architecture (Yiannacouras *et al.* 2012) employs vector chaining, control flow execution support and a banked register file to reduce execution time. Both approaches are limited to a clock rate of less than 200 MHz which is much less than the 500–700 MHz that is possible in implementing FPGA designs directly.

VENICE (Severance and Lemieux 2012) is a processor-based solution that provides support for operations on unaligned vectors, and FlexGrip (Andryc *et al.* 2013) is an FPGA-based multicore architecture that allows mapping of pre-compiled CUDA kernels which is scalable, programmable and flexible. However, both solutions only operate at 100 MHz. They offer flexibility, but the low frequency will result in relatively poorer implementations when compared to dedicated implementations.

Chu and McAllister (2010) have created a programmable, heterogeneous parallel soft-core processor architecture which is focused on telecommunications applications. Similarly, iDEA (Cheah *et al.* 2012) is a nine-stage, pipelined, soft-core processor which is based around the DSP48E1 and supports basic arithmetic and logical instructions by utilizing limited FPGA resources. The design runs at 407 MHz which is 1.93 times faster than Xilinx MicroBlaze and significantly faster than previous work. This improved performance provides a better proposition for achieving a soft-core-based approach for data applications. In this book, we concentrate on a processor that has been developed for image processing.

12.6.1 IPPro FPGA-Based Processor

A custom-designed DSP48-based RISC architecture, called IPPro (Siddiqui *et al.* 2014) has been developed; it uses the Xilinx DSP48E2 primitive as the ALU for faster processing and supports a wide range of instructions and various memory accesses. The following design decisions were made to optimize FPGA performance and image processing needs:

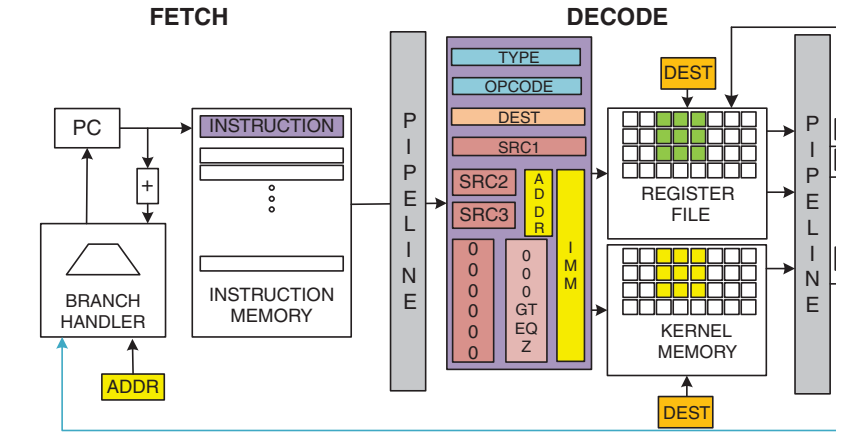
- High processing capability is required to handle the large amount of data (30–40 MB/s) needed for real-time video streaming. This is achieved by explicitly mapping the operations and logic to the underlying FPGA resource primitives and ensuring a good match. This allowed a 350–450 MIPS performance per processor to be achieved.
- Efficient memory utilization by distributing memory to hide data transfer overheads between main and local memory to keep IPPro busy in processing data. This matches the distributed nature of memory in FPGA resources. Dedicated kernel memory accelerates the linear filter operations and also reduces the code size by avoiding excessive load/store instructions and maximizing memory reusability.
- Optimized instructions/addressing modes and reduced branch penalty by decreasing the number of pipeline stages as unpredicted branches degrade performance. The creation of special instruction sets allows the acceleration of image processing operations; addressing modes to give flexibility to the programmer; and conditional execution in the form of a customizable and flexible branch controller to support mask-based conditional execution out-of-box without need of significant architectural changes.

Memory

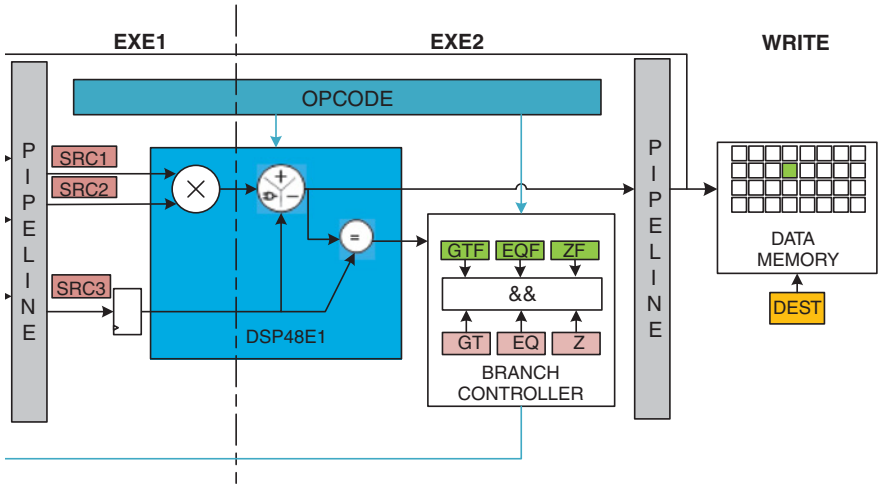
IPPro is capable of processing 16-bit operations, and uses distributed memory to build a memory hierarchy, with register file, data memory, and kernel memory. The IPPro architecture uses a five-stage balanced, pipelined architecture as shown in Figure 12.8.

IPPro is capable of running at 337 MHz on a Xilinx SoC, in particular XC7Z020-3, using one DSP48E, one BRAM and 330 slice registers per processor. The main idea of the processor was to keep it compact, reprogrammable and scalable as much as possible to achieve high throughput rates compared to custom-made HDL designs. It contains small, fast and efficient memory to locally store data and keep ALU busy in processing data. This helps to hide data transfer overheads between the main and local memories.

It supports various instructions and memory accesses and is capable of processing signed 16-bit operations. The IPPro processor architecture uses five-stage balanced



(a) First half



(b) Second half

Figure 12.8 IPPro architecture

pipelining and supports streaming mode operation where the input and output data are read and written back to FIFO structures, as shown in Figure 12.8.

IPPro strikes a balance between programmability and the need to maintain FPGA performance. Overall it has the following addressing modes: from local memory to local memory; from local memory to FIFO (LM–FIFO); from kernel memory to FIFO. The local memory is composed of general-purpose registers used mainly for storing operands of instructions or pixels. This memory currently contains 32 sixteen-bit registers. A FIFO is a single internal register of IPPro where the input and output streams from/to an external FIFO are stored. Kernel memory is a specialized location for coefficient storage in windowing and filtering operations with 32 sixteen-bit registers.

Data Path

The complete IPPro data path is shown in Figure 12.8. It has a loadstore five-stage balanced pipelined architecture giving a fixed latency of five clock cycles. It exploits the features of the Xilinx DSP48E1 to implement all of the supported instructions and provides a balance between hardware resource utilization, performance, throughput, latency and branch penalty. A balanced pipeline simplifies the compiler tool chain development compared to variable pipeline architecture. The deep pipeline comes at the cost of larger latency and branch penalty which adversely affects the overall performance. Various techniques predict branches, but none of them was deemed to give a shorter latency. The five pipeline stages are as follows:

1. Fetch (IF)
2. Decode (ID)
3. Execute 1 (EXE1)
4. Execute 2 (EXE2)
5. Write Back (WB).

Instruction Set

An example of the supported instructions can be seen in Table 12.3. This table shows the IPPro LM–FIFO addressing mode instructions and some miscellaneous others. The IPPro instruction set is capable of processing basic arithmetic and logical operations for different addressing modes. In addition to the unary and binary instructions, it has support for trinary expressions such as `MULADD, MULSUB, MULACC`.

Given the limited instruction support and requirements from the application domain, it is envisaged that coprocessor(s) could be added to provide better support for more complex processes such as division and square root. Ongoing research is being undertaken to design such a coprocessor (Kelly *et al.* 2016).

Flags and Branches

Flags are important status indicators in processors and used to handle exceptions encountered during data computation. IPPro currently supports the following data flags

Table 12.3 Instruction set

LM–FIFO		Misc	
ADD	LOR	JMP	GET
SUB	LNOR	BNEQ	PUSH
MUL	LNOT	BEQ	NOP
MULADD	LNAND	BZ	BYPASS
MULSUB	LAND	BNZ	DIV
MULACC	LSL	BS	
LXOR	LSR	BNS	
LXNR	MIN	BNGT	
	MAX	BGT	

but is flexible enough to allow new flags to be defined by modifying the branch controller shown in Figure 12.8:

1. Greater than (GTF)
2. Equal (EQF)
3. Zero (ZF)
4. Sign Flag (SF).

The flags are generated using the pattern detector function which is embedded inside the DSP48E1 block as dedicated functionality. It compares the two operands available at the input of DSP48E1 and sets the pattern detect (PD) bit in the very same clock cycle if both operands are equal. Therefore no additional clock cycle is needed to compute the flag bit which is important in the case of conditional/data dependent instructions being executed in the multicore architecture. The branch controller is flexible and scalable as it is created using combinational logic. A dataflow-based programming route has also been created (see Amiri *et al.* 2016)

12.7 System Hardware

An example of a typical system architecture is given in Figure 12.9. This gives details of the front-end processor architecture, prototyped on a Zedboard platform which comprises a Xilinx Zynq SoC which comprises on-chip dual-core ARM processors and programmable logic. The SIMD-IPPro is comprised of a number of IPPro cores connected together.

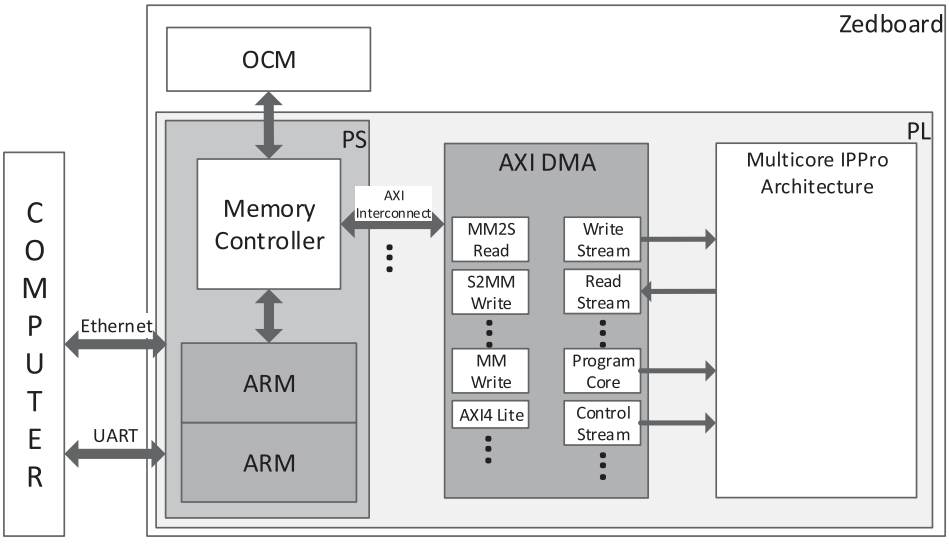


Figure 12.9 Proposed system architecture

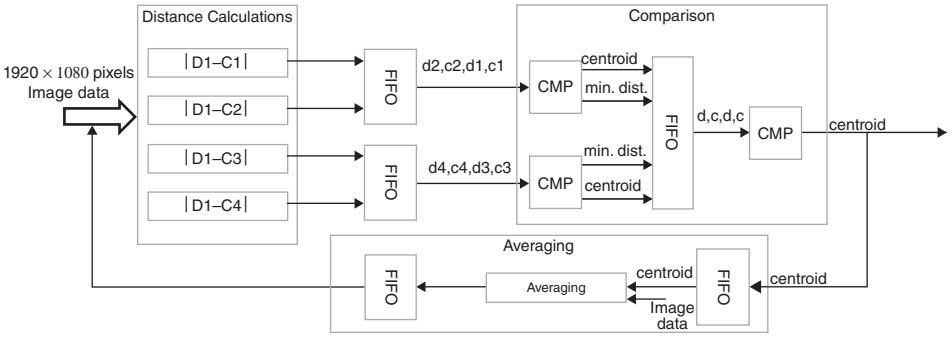


Figure 12.10 Proposed architecture for k-means image clustering algorithm on FPGA

The main aim from a design perspective for the k-means algorithm is to allocate the processing requirements (distance calculation, comparison and averaging) to the FPGA resources. The averaging is a feedback block responsible for recalculation of new centroids for new iteration. The image data is stored in off-chip memory, while the centroid values are stored in the kernel memory of the IPPro-based FPGA. Figure 12.10 shows our architecture for the k-means algorithm.

An example of IPPro code is shown in Table 12.4 for the computation of the equation

$$M10 = \sum_{i=0}^{3} (Mi * M(i+1)) \tag{12.9}$$

12.7.1 Distance Calculation Block

The distance calculation block is the most computationally intensive block in the k-means algorithm. Since the distances between each data point and the centroids are independent, the computation was organized across a number of clusters. One IPPro is dedicated to each cluster for the distance calculation, which calculates the distance

Table 12.4 Example of IPPro code representation

S/N	IPPro code	Description
1	LD, R1, M	Load from data memory M1 to register R1
2	LD, R2, M2	Load from data memory M2 to register R2
3	LD, R3, M3	Load from data memory M3 to register R3
4	LD, R4, M4	Load from data memory M4 to register R4
5	MUL R10, R1, R2	Multiply R1 and R2 and store in register
6	MULACC R10, R3, R4	Multiply R3 and R4 and accumulate in R10
7	ST R10, M10	Store from register to data memory

between the data pixels and each centroid based on their dimensions, finds their absolute values and sums them. The outputs of the distance block are the distances and the corresponding centroid values. The distance computation takes 58 cycles and the outputs are then stored in the FIFO for use by the next block of the *k*-means algorithm.

12.7.2 Comparison Block

The comparison block receives the distance and the associated centroid values from the distance calculation block serially. It compares the distance values and selects the minimum value, along with the centroid value that stands for the minimum value. These outputs are part of the inputs to the next comparison block. The last comparison block produces one centroid output which replaces the image data that it represents, in order to give a clustered image in the final iteration. The centroid values as an output of the comparison stage are fed into the averaging block. It takes 81 cycles to compare eight distance values representing eight clusters.

12.7.3 Averaging

This block receives the centroid values from the comparison block and uses them to group the image data according to their order. For instance, when it receives centroid 1, and then image pixel $D1$, it puts $D1$ under the cluster 1 group. In our architecture, the averaging block avoids performing the division operation at the end of the iteration, by averaging each group's value each time there is an addition; this is done by performing a binary shift to the right by one position. This is an approximation to doing division at the end of the pixel count. The outputs of the averaging block are the new centroid values for the next iteration. It takes 76 cycles to process a pixel for eight clusters.

12.7.4 Optimizations

Some of the ways that we used to achieve the optimization are: data parallelism, cluster parallelism and code reduction. The focus of the design is to reduce the IPPro assembly code to the best possible by ensuring that the most suitable and shortest codes are used to represent the algorithmic decomposition.

The computation was profiled for different multicore arrangements and the resulting realizations then compared in terms of execution time. After a number of mappings of cores per function, it was decided that one core would be used for the distance calculation, three cores for the comparison block and one core for the averaging block. Table 12.5 shows the results for 4, 8 and 9 centroids in terms of number of cycles,

Table 12.5 Results for 4, 8 and 9 centroids image clustering

	Dist.			Comp.			Aver.		
No. of clusters	4	8	9	4	8	9	4	8	9
No. of cycles	106	106	106	52	78	104	39	75	84
Execution time (s)	0.41	0.41	0.41	0.20	0.30	0.41	0.15	0.29	0.32
Latency (μs)	0.2	0.2	0.2	0.10	0.15	0.20	0.07	0.14	0.15
Throughput (MP/s)	5	5	5	10.0	6.7	5	14	7.2	6.5

execution time, latency and throughput profiled for a high-definition image of 1920 × 1080 in a single iteration using color images when the clock frequency is 530 MHz.

12.8 Conclusions

The purpose of the chapter was to acknowledge the major developments in the area of Big Data applications. This area and the need to develop data centers to meet the increasing needs of the Big Data analytics have done more than any recent developments to see FPGAs being used in modern computing systems. The interest by Intel, IBM and Microsoft has been substantial, ranging from major joint projects with the two main FPGA companies to the purchase of Altera by Intel.

In some cases, FPGAs offer computing gains over processor-based alternatives such as CPUs, DSP microprocessors and GPUs, particularly if power consumption is taken into consideration. Indeed, this is seen as a major advantage of FPGA technologies, so if they can offer a performance advantage for even a small range of functionality, then this would be seen as beneficial. Given that some of the data mining algorithms have characteristics similar to those seen in DSP algorithms, then it would appear that FPGAs have a major role to play in future computing systems. For this reason, the authors were motivated to include a chapter in this revised edition of the book.

Certainly the abolition of high-level programming languages described in Chapter 7, and processor architectures such as the IPPro system described in this chapter, will have a major impact on how these systems will be built and programmed. In any case, the length of time need to compile high-level languages onto FPGA hardware will need to be addressed.

Bibliography

Abu-Mostafa YS, Magdon-Ismail M, Lin H-T 2012 *Learning from Data*. AMLbook.com.

Altera Corp. 2015 *Nios II (Gen2) Processor Reference Handbook*, ver 2015.04.02. Available from http://www.altera.com (accessed February, 28 2016).

Amazon, 2016 Developer Preview – EC2 Instances (F1) with Programmable Hardware. Available from https://aws.amazon.com/blogs/aws/developer-preview-ec2-instances-f1-with-programmable-hardware/ (accessed December 28, 2016)

Amiri M, Siddiqui FM, Kelly C, Woods R, Rafferty K and Bardak B 2016 FPGA-based soft-core processors for image processing applications. J. of VLSI Signal Processing, DOI: 10.1007/s11265-016-1185-7.

Andryc K, Merchant M, Tessier R 2013 FlexGrip: A soft GPGPU for FPGAs. In *Proc. Int. Conf. on Field Programmable Technology*, pp. 230–237.

Apache 2015 *Welcome to Apache™ Hadoop®*. Available from http://hadoop.apache.org/ (accessed January 19, 2015).

Appuswamy R, Gkantsidis C, Narayanan D, Hodson O, Rowstron A 2013 Scale-up vs scale-out for Hadoop: Time to rethink? In *Proc. ACM 4th Annual Symp. on Cloud Computing*, article no. 20.

Bazi Y, Melgani F 2006 Toward an optimal SVM classification system for hyperspectral remote sensing images. *IEEE Trans. on Geoscience and Remote Sensing*, 44(11), 3374–3385.

Black F, Scholes M 1973 The pricing of options and corporate liabilities. *Journal of Political Economy*, 81 (3), pp. 637–654.

Blott M, Vissers K 2014 Dataflow architectures for 10Gbps line-rate key-value-stores. In *Proc. IEEE Hot Chips*, Palo Alto, CA.

Cadambi S, Durdanovic I, Jakkula V, Sankaradass M, Cosatto E, Chakradhar S, Graf HP 2009 A Massively Parallel FPGA-based coprocessor for support vector machines. In *Proc. IEEE Int. Symp. FPGA-based Custom Computing Machine*, pp. 115–122.

Cheah HY, Fahmy S, Maskell D 2012 iDEA: A DSP block based FPGA soft processor. In *Int. Conf. on Field Programmable Technology*, pp. 151–158.

Chu X, McAllister J 2010 FPGA based soft-core SIMD processing: A MIMO-OFDM fixed-complexity sphere decoder case study. In *Proc. IEEE Int. Conf. on Field Programmable Technology*, pp. 479–484.

Cichosz P 2015 *Data Mining Algorithms: Explained Using R*. John Wiley & Sons, Chichester.

Clark D 2015 Intel completes acquisition of Altera. *Wall Street J.*, December 28. Available from http://www. wsj.com/articles/ intel-completes-acquisition-of-altera-1451338307 (accessed March 4, 2016).

Clark D, Bell J 2007 Multi-target state estimation and track continuity for the particle PHD filter. *IEEE Trans. on Aerospace Electronics Systems*, 43(4), 1441–1453.

Cloudera 2016 *Impala SQL Language Reference*. Available from http://www.cloudera.com/documentation.html (accessed March 4, 2016).

Dean J and Ghemawat S 2004 MapReduce: Simplified data processing on large clusters. In *Proc. 6th Symp. on Operating Systems Design & Implementation*.

Estlick M, Leeser M, Theiler J, Szymanski JJ 2001 Algorithmic transformations in the implementation of *k*-means clustering on reconfigurable hardware. In *Proc. ACM/SIGDA 9th Int. Symp. on Field Programmable Gate Arrays*, pp. 103–110.

Heston S 1993 A closed-form solution for options with stochastic volatility. *Review of Financial Studies*, 6, 327–343.

Hussain HM, Benkrid K, Seker H, Erdogan AT 2011 FPGA implementation of *k*-means algorithm for bioinformatics application: An accelerated approach to clustering microarray data. In *Proc. NASA/ESA Conf. on Adaptive Hardware and Systems*, pp. 248–255.

Jain A K 2010 Data clustering: 50 years beyond *K*-means. *Pattern Recognition Letters*, 31(8), 651–666.

Kelly C, Siddiqui FM, Bardak B, Wu Y, Woods R 2016 FPGA based soft-core processors hardware and compiler optimizations. In *Proc. Int. Symp. of Applied Reconfigurable Computing*, pp. 78–90.

Krizevsky A 2014 One weird trick for parallelizing convolutional neural networks. arXiv:1404.5997 (accessed March 4, 2016).

Leeser ME, Belanovic P, Estlick M, Gokhale M, Szymanski JJ, Theiler JP 2002 Applying reconfigurable hardware to the analysis of multispectral and hyperspectral imagery. In *Proc. Int. Symp. on Optical Science and Technology*, pp. 100–107.

Lewis M 2014 *Flash Boys: A Wall Street Revolt*. Norton, New York.

Lin Z, Lo C, Chow P 2012 *K*-means implementation on FPGA for highdimensional data using triangle inequality. In *Proc. IEEE Int. Conf. on Field Programmable Logic*, pp. 437–442.

Manyika J, Chui M, Brown B, Bughin J, Dobbs R, Roxburgh C, Hung Byers A 2011 Big data: The next frontier for innovation, competition, and productivity. McKinsey Global Institute Report.

Marshall AH, Mitchell H, Zenga M 2014 Modelling the length of stay of geriatric patients in the Emilia Romagna hospitals using Coxian phase-type distributions with covariates. In Carpita M, Brentari E, Qannari EM (eds) *Advances in Latent Variables: Methods, Models and Applications*, pp. 1–13. Springer, Cham.

McNulty E 2014 Understanding Big Data: The seven V's. *Dataconomy*. Availavle at http://dataconomy.com/seven-vs-big-data/, (accessed March 4, 2016).

Narayanan R, Honbo D, Memik G, Choudhary A, Zambreno J 2007 An FPGA implementation of decision tree classification. In *Proc. Conf. on Design Automation and Test in Europe*, pp. 189–194.

Omand D, Bartlett J, Miller C 2012 #Intelligence. Available at http://www.demos.co.uk/publications/intelligence [Accessed 19 January 2015].

Polat Ö and Yıldırım T 2009 FPGA implementation of a general regression neural network: An embedded pattern classification system. *Digital Signal Processing*, 20(3), 881–886.

Putnam A, Caulfield AM, Chung ES, Chiou D, Constantinides K, Demme J, Esmaeilzadeh H, Fowers J, Gopal GP, Gray J, Haselman M, Hauck S, Heil S, Hormati A, Kim J-Y, Lanka S, Larus J, Peterson E, Pope S, Smith A, Thong J, Xiao PY, Burger D 2014 A reconfigurable fabric for accelerating large-scale datacenter services. In *Proc. IEEE Int. Symp. on Computer Architecture*, pp. 13–24.

Russell RM 1978 The CRAY-1 computer system. *Communications of the ACM*, 21, 63–72.

Severance A, Lemieux G 2012 VENICE: A compact vector processor for FPGA applications. In *Proc. Int. Conf. on Field-Programmable Technology*, pp. 261–268.

Siddiqui FM, Russell M, Bardak B, Woods R, Rafferty K 2014 IPPro: FPGA based image processing processor. In *Proc. IEEE Workshop on Signal Processing Systems*, pp. 1–6.

Winterstein F, Bayliss S, Constantinides G 2013 FPGA-based *K*-means clustering using tree-based data structures. In *Proc. Int. Conf. on Field Programmable Logic and Applications*, pp. 1–6.

Xilinx Inc. 2009 *MicroBlaze Processor Reference Guide Embedded Development Kit*. EDK 11.4 UG081 (v10.3). Available from http://www.xilinx.com (accessed February 28, 2016).

Yiannacouras P, Steffan JG, Rose J 2012 Portable, flexible, and scalable soft vector processors. *IEEE Trans. on VLSI Systems*, 20(8), 1429–1442.

Zikopoulos P, Eaton C, DeRoos R, Deutsch T, Lapos G 2012 *Understanding Big Data: Analytics for Enterprise Class Hadoop and Streaming Data*. McGraw-Hill, New York.