Shahrouzi and Perera *EURASIP Journal on Embedded Systems* (2017) 2017:25
DOI 10.1186/s13639-017-0074-x

EURASIP Journal on
Embedded Systems

RESEARCH Open Access

Dynamic partial reconfigurable hardware architecture for principal component analysis on mobile and embedded devices

S. Navid Shahrouzi and Darshika G. Perera*

Abstract

With the advancement of mobile and embedded devices, many applications such as data mining have found their way into these devices. These devices consist of various design constraints including stringent area and power limitations, high speed-performance, reduced cost, and time-to-market requirements. Also, applications running on mobile devices are becoming more complex requiring significant processing power. Our previous analysis illustrated that FPGA-based dynamic reconfigurable systems are currently the best avenue to overcome these challenges. In this research work, we introduce efficient reconfigurable hardware architecture for principal component analysis (PCA), a widely used dimensionality reduction technique in data mining. For mobile applications such as signature verification and handwritten analysis, PCA is applied initially to reduce the dimensionality of the data, followed by similarity measure. Experiments are performed, using a handwritten analysis application together with a benchmark dataset, to evaluate and illustrate the feasibility, efficiency, and flexibility of reconfigurable hardware for data mining applications. Our hardware designs are generic, parameterized, and scalable. Furthermore, our partial and dynamic reconfigurable hardware design achieved 79 times speedup compared to its software counterpart, and 71% space saving compared to its static reconfigurable hardware design.

Keywords: Data mining, Embedded systems, FPGAs, Mobile devices, Partial and dynamic reconfiguration, Principal component analysis, Reconfigurable hardware

1 Introduction

With the proliferation of mobile and embedded computing, a wide variety of applications are becoming common on these devices. This has opened up research and investigation into lean code and small footprint hardware and software architectures. However, these devices have stringent area and power limitations, lower cost and time-to-market requirements. These design constraints pose serious challenges to the embedded system designers.

Data mining is one of the many applications that are becoming common on mobile and embedded devices. Originally limited to a few applications such as scientific research and medical diagnosis, data mining has become vital to a variety of fields including finance, marketing, security, biotechnology, and multimedia. Many of today's data mining tasks are compute and data intensive, requiring significant processing power. Furthermore, in many cases, the data need to be processed in real time to reap the actual benefits. These constraints have a large impact on the speed-performance of the applications running on mobile devices.

To satisfy the requirements and constraints of the mobile and embedded devices, and also to enhance the speed-performance of the applications running on these devices, it is imperative to incorporate some special-purpose hardware into embedded system designs. These customized hardware algorithms should be executed in single-chip systems, since multi-chip solutions might not be suitable due to the limited footprint on mobile and embedded devices. The customized hardware provides superior speed-performance, lower power consumption, and area efficiency [12, 40], compared to the equivalent software running on general-purpose microprocessor, advantages that are crucial for mobile and embedded devices.

* Correspondence: darshika.perera@uccs.edu
Department of Electrical and Computer Engineering, University of Colorado, 1420 Austin Bluffs Parkway, Colorado Springs, CO 80918, USA

For more complex operations, it might not be possible to populate all the computation circuitry into a single chip. An alternative is to take the advantage of reconfigurable computing systems. Reconfigurable hardware has similar advantages as special-purpose hardware, leading to low power and high performance. Furthermore, reconfigurable computing systems have added advantages: a single chip to perform the required operation, flexible computing platform, and reduced time-to-market. This reconfigurable computing system could address the constraints associated with mobile and embedded devices, as well as the flexibility and performance issues in processing a large data set.

In [30], an analysis of single-chip hardware support for mobile and embedded applications was carried out. These analyses illustrated that FPGA-based reconfigurable hardware provides numerous advantages, including flexibility, upgradeability, compact circuits and area efficiency, shorter time-to-market, and relatively low cost, which are important for mobile and embedded devices. Multiple applications can be executed on a single chip, by dynamically reconfiguring the hardware on chip from one application to another as needed.

Our main objective is to provide efficient dynamic reconfigurable hardware architectures for data mining applications on mobile and embedded devices. In this research work, we focus on reconfigurable hardware support for dimensionality reduction techniques in data mining, specifically principal component analysis (PCA). For mobile applications such as signature verification and handwritten analysis, PCA is applied initially to reduce the dimensionality of the data, followed by similarity measure.

This paper is organized as follows: In Section 2, we discuss and present the main tasks in data mining, issues in mining high-dimensional data, and elaborate on principal component analysis (PCA), one of the most commonly used dimensionality reduction techniques in data mining. Our design approach and development platform are presented in Section 3. In Section 4, the partial and dynamic reconfigurable hardware architecture for the four stages of the PCA algorithm is introduced. Experiments are carried out to evaluate the speed-performance and area efficiency of the reconfigurable hardware designs. These experimental results and analysis are reported and discussed in Section 5. In Section 6, we summarize our work and conclude.

2 Data mining techniques

Data mining is an important research area as many applications in various domains can make use of it to sieve through large volume of data to discover useful patterns and valuable knowledge. It is a process of finding correlations or patterns among various fields in large data sets; this is done by analyzing the data from many different perspectives, categorizing it, and summarizing the identified relationships [6].

Data mining commonly involves any of the four main high-level tasks [15, 28]: classification, clustering, regression, and association rule mining. From these, we are focusing on the mostly widely used clustering and classification, which typically involves the following steps [15, 28]: pattern representation, pattern proximity measure, grouping (for clustering) and labeling (for classifying), and data abstraction (optional).

Pattern representation is the first step toward clustering or classification. Patterns (or records) are represented as multidimensional vectors, where each dimension (or attribute) represents a single feature [34]. Pattern representation is often used to extract the most descriptive and discriminatory features in the original data set; then these features can be used exclusively in subsequent analyses [22].

2.1 Mining high-dimensional data

An important issue often arises, while clustering or classifying, is the problem of having too many attributes (or dimensions). There are four major issues associated with clustering or classifying high-dimensional data [4, 15, 24]:

- Multiple dimensions are impossible to visualize. Also, since the amount of data often increases exponentially with dimensionality, multiple dimensions are becoming increasingly difficult to enumerate [24]. This is known as curse of dimensionality [24].
- As the number of dimensions increase, the concept of proximity or distance becomes less precise; this is especially true for spatial data [13].
- Clustering typically group objects that are related based on the attribute's value. When there is a large number of attributes, it is highly likely that some of the attributes or features might be irrelevant, thus negatively affects the proximity measures and the creation of clusters [4, 24].
- Correlations among subsets of features: When there is a large number of attributes, it is highly likely that some of the attributes are correlated [24].

To overcome the above issues, pattern representation techniques such as feature extraction and feature selection are often used to reduce the dimensionality before performing any other data mining tasks.

Some of the feature selection methods used for dimensionality reduction include mutual information [28], chi-square [28], and sensitivity analysis [1, 56]. Some of the feature extraction methods used for dimensionality reduction include singular value decomposition [14, 37], principal component analysis [21, 23], independent component analysis [20], and factor analysis [7].

2.2 PCA: a dimensionality reduction technique

Among the feature extraction/selection methods, principal component analysis (PCA) is the most commonly [1, 23, 37] used dimensionality reduction technique in clustering and classification problems. In addition, due to the necessity of having a small memory footprint of data, PCA is applied to many data mining applications that are appropriate for mobile and embedded devices such as: handwritten analysis or signature verification, palm-print or finger-print verification, iris verification, and facial recognition.

PCA is a classical technique [42]: The main idea is to extract the prominent features of the data set and to perform data reduction (compression). PCA finds a linear transformation, known as Karhunen-Loeve Transform (KLT), which reduces the number of the dimensions of the feature vectors from m to d (where $d < < m$) in such a way that the "information is maximally preserved in minimum mean squared error sense" [11, 36]. PCA reduces the dimensionality of the data by transforming the original data set to a new set of variables called principal components (PCs) to extract the prominent features of the data [23, 42]. According to Yeung and Ruzzo [57], "PCs are uncorrelated and ordered, such that the k^{th} PC has k^{th} largest variance among all PCs; and the k^{th} PC can be interpreted as the direction that maximizes the variation of the projection of the data points such that it is orthogonal to the first (k-1) PCs." Traditionally, the first few PCs are used in data analysis, since they retain most of the variants among the data features (in the original data set), and eliminate (by the projection) those features that are highly correlated among themselves; whereas the last few PCs are often assumed to retain only the residual noise in the data [23, 57].

Since PCA effectively reduces the dimensionality of the data, the main advantage of applying PCA on original data is to reduce the size of the computational problem [42]. Normally, when the number of attributes of a data set is large, it takes more time to process the data, since the number of attributes is directly proportional to processing time; thus, by reducing the number of attributes (dimensions), running time of the system can be minimized [42]. In addition, for clustering, it helps to identify the characteristics of the clusters [22], and for classification, it improves classification accuracy [1, 56]. The main disadvantage of applying PCA is the loss of information, since there is no guarantee that the sacrificed information is not relevant to the aims of further studies, and also there is no guarantee that the largest PCs obtained will contain good features for further analysis [21, 57].

2.2.1 The process of PCA

PCA computation consists of four stages [21, 37, 38]: mean computation, covariance matrix computation, eigenvalue matrix, thus eigenvector computation, and PCs matrix computation. Consider the original input data set $\{X\}_{m\text{X}n}$ as an $m\text{X}n$ matrix, where m is the number of dimensions and n is the number of vectors. Firstly, the mean is computed along the dimensions of the vectors of the data set. Secondly, the covariance matrix is computed after determining the deviation from the mean. Covariance is always measured between two dimensions [21, 38]. With covariance, one can find out how much the dimensions vary from the mean with respect to each other [21, 38]. Covariance between one dimension and itself gives the variance.

Thirdly, eigenanalysis is performed on the covariance matrix to extract independent orthonormal eigenvalues and eigenvectors [2, 21, 37, 38]. As stated in [2, 38], eigenvectors are considered as the "preferential directions" or the main patterns in the data, and eigenvalues are considered as the quantitative assessment of how much a PC represents the data. Eigenvectors with the highest eigenvalues correspond to the variables (dimensions) with the highest correlation in the data set. Lastly, the set of PCs is computed and sorted by their eigenvalues in descending order of significance [21].

Various techniques can be used to perform PCA computation. These techniques typically depend on the application and the data set used. The most common algorithm for PCA involves the computation of the eigenvalue decomposition of a covariance matrix [21, 37, 38]. There are also various ways of performing eigenanalysis or eigenvalue decomposition (EVD). One well known EVD method is cyclic Jacobi method [14, 33]. However, this is only suitable for small matrices, where number of dimensions are less than or equal to 10 ($m = <10$) [14, 37]. For larger matrices [29], where the number of dimensions are more than 10 ($m > 10$), other algorithms such as QR [1, 56], Householder [29], or Hessenberg [29] methods should be employed. Among these methods, QR algorithm, first introduced in 1961, is one of the most efficient and accurate methods to compute eigenvalues and eigenvectors during PCA analysis [29, 41]. It can simultaneously approximate all the eigenvalues of a matrix. For our work, we are using QR algorithm for EVD.

In summary, clustering and classifying high-dimensional data presents many challenging problems in this big data era. The computational cost of processing massive amount of data in real time is immense. PCA can reduce a complex high-dimensional data set to a lower dimension, in order to unveil the simplified structures that are otherwise hidden, while reducing the size of the computational cost of analyzing the data [21, 37, 38]. Hardware support could further reduce the computational cost of processing data and improve the speed-performance of the PCA analysis. In this research work, we introduce partial and dynamic reconfigurable hardware to enhance the PCA computation for mobile and embedded devices.

3 Design approach and development platform

For all our experiments, both software and hardware versions of the various computations are implemented using a hierarchical platform-based design approach to facilitate component reuse at different levels of abstraction. The hardware versions include static reconfigurable hardware (SRH) and dynamic reconfigurable hardware (DRH). As shown in Fig. 1, our design consists of different abstraction levels, where higher-level functions utilizes lower-level sub-functions and operators: the fundamental operators including add, multiply, subtract, compare, square-root, and divide at the lowest level; mean, covariance matrix, eigenvalue matrix, and PC matrix computations at the next level; and the PCA at the highest level.

All our hardware and software experiments are carried out on the ML605 FPGA development board [51], which is built on a 40-nm CMOS process technology. The ML605 board utilizes a Xilinx Virtex 6 XC6VLX240T-FF1156 device. The development platform includes large on-chip logic resources (37,680 slices), MicroBlaze soft processors, and onboard configuration circuitry for development purpose. It also includes 2-MB on-chip BRAM (block random access memory) and 512-MB DDR3-SDRAM external memory to hold large volume of data. To hold the configuration bitstreams, ML605 board has several external non-volatile memories including 128 MB of Platform Flash XL, 32-MB BPI Linear Flash, and 2-GB Compact Flash. Additional user desired features could be added through daughter cards attached to the two onboard FMC (FPGA Mezzanine Connectors) expansion connectors.

Both the static and dynamic reconfigurable hardware modules are designed in mixed VHDL and Verilog. They are executed on the FPGA (running at 100 MHz) to verify their correctness and performance. Xilinx ISE 14.7 and XPS 14.7 are used for the SRH designs. Xilinx ISE 14.7, XPS 14.7, and PlanAhead 14.7 (with partial reconfiguration features) are used for the DRH designs. ModelSim SE and Xilinx ChipscopePro 14.7 are used to verify the results and functionalities of the designs. Software modules are written in C and executed on the MicroBlaze processor (running at 100 MHz) on the same FPGA with level-II optimization. Xilinx XPS 14.7 and SDK 14.7 are used to verify the software modules.

As a proof-of-concept work [31, 32], we initially proposed reconfigurable hardware support for the first two stages of the PCA computation, where both the SRH [31] and the DRH [32] are designed using integer operators. Unlike our proof-of-concept designs, in this research work, both the software and hardware modules are designed using floating-point operators, instead of integer operators. The hardware modules for the fundamental operators are designed using single precision floating-point units [50] from the Xilinx IP core library. The MicroBlaze is also configured to use single precision floating-point unit for the software modules.

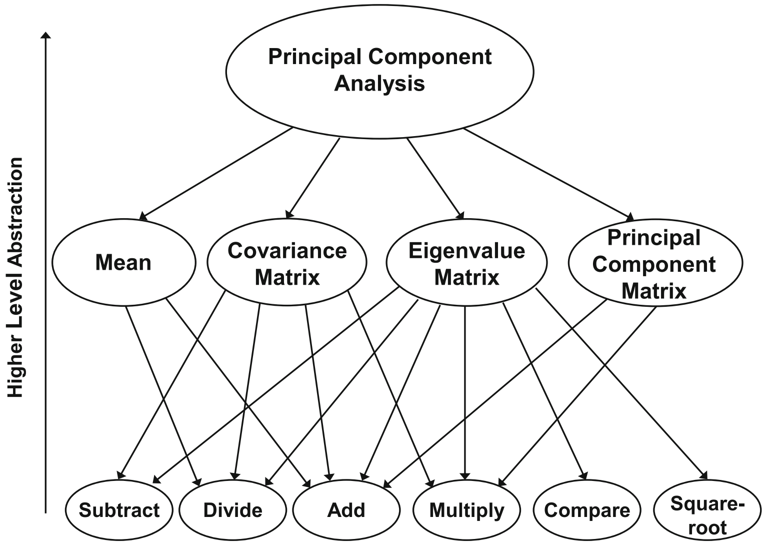

Fig. 1 Hierarchical platform-based design approach. Our design consists of different abstraction levels, where higher-level functions utilize lower-level sub-functions

The performance gain or speedup resulting from the use of hardware over software is computed using the following formula:

$$\text{Speedup} = \frac{\text{BaselineExecutionTime(Software)}}{\text{ImprovedExecutionTime(Hardware)}} \tag{1}$$

Since our intention is to provide reconfigurable hardware architectures for data mining applications on mobile and embedded devices, we decided to utilize a data set that is appropriate for applications on these devices. After exploring several databases, we decided on a real benchmark data set, the "Optdigit" [3], for recognizing handwritten characters. The database consists of 200 handwritten characters from 43 people. The data set has 3823 records (vectors), where each record has 64 attributes (elements). We investigated several papers that used this data set for PCA computations and obtained source codes written in MatLab for PCA analysis from one of the authors [39]. Results from the MatLab code on the optdigit data set are used to verify our results using reconfigurable hardware designs as well as software designs. In addition, a software program written in C for the PCA computation is executed on a personal computer. These results are also used to verify our results from the embedded software and hardware designs.

3.1 System-level design

ML605 consists of large banks of external memory which can be accessed by the FPGA hardware modules and the MicroBlaze embedded processor using the memory controllers. The FPGA itself contains 2 MBs of on-chip memory [51], which is not sufficient to store the large volume of data commonly found in many data mining applications. Therefore, we integrated a 512-MB external memory, the DDR3-SDRAM [54] (double-data-rate synchronous dynamic random access memory) into the system. DDR3-SDRAM and AXI memory controller run at 200 MHz, while the rest of the system is running at 100 MHz. As depicted in Fig. 2, the AXI (Advanced Extensible Interface) interconnect acts as the glue logic for the system.

Figure 2 illustrates how our user-designed hardware interfaces with the rest of the system. Our user-designed hardware consists of the user-designed hardware module, the user-designed BRAM, and the user-defined bus. As shown in Fig. 2, in order for our user-designed hardware module (i.e., both the SRH and DRH) to communicate with the MicroBlaze and the DDR3-SDRAM, it is connected to the AXI4 bus [46] through the AXI Intellectual Property Interface (IPIF) module, using a set of ports called the Intellectual Property Interconnect (IPIC). Through the IPIF module, our user-designed hardware module is enhanced with stream-in (or burst) data from the DDR3-SDRAM. The AXI Master Burst [47] provides an interface between the user-designed module and and AXI bus and performs AXI4 Burst transactions of 1–16, 1–32, 1–64, 1–128, and 1–256 data beats per AXI4 read or write request. For our design, we used the maximum data beats of 256 and burst width of 20 bits. As stated in [47], the bit width allows a maximum of 2^n-1 bytes to be specified for transaction per command submitted by the user on the IPIC command interface, thus 20 bits provides 1,048,575 bytes per command.

With this system-level interface, our user-designed hardware module (both SRH and DRH) can receive a signal from the MicroBlaze processor via the AXI bus

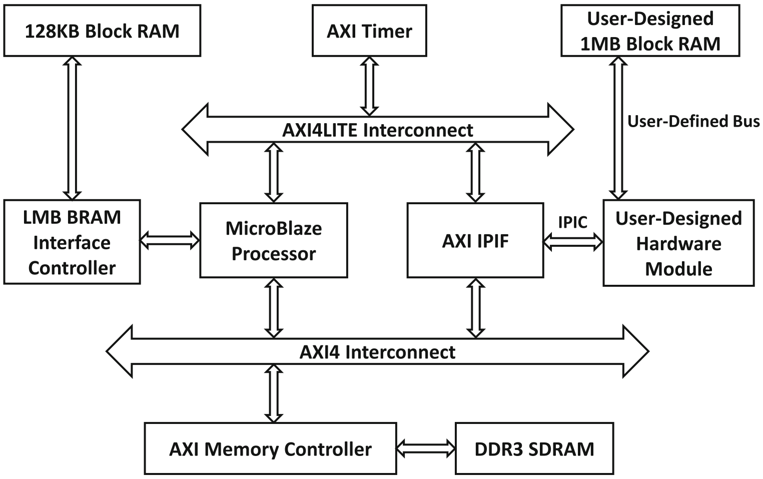

Fig. 2 System-level interfacing block diagram. This figure illustrates how our user-designed hardware interfaces with the rest of the system. In this case, the AXI (Advanced Extensible Interface) interconnect acts as the glue logic to the system

and start processing, read/write data/results from/to the DDR3-SDRAM, and send a signal to the MicroBlaze when execution is completed. When MicroBlaze sends a signal to the hardware module, it can then continue to execute other tasks until the hardware module writes back the results to the DDR3-SDRAM and sends a signal to notify the processor. The execution times for the hardware as well as MicroBlaze are obtained using the hardware AXI Timer [49] running at 100 MHz.

3.1.1 Pre-fetching technique

From our proof-of-concept work [32], it was observed that a significant amount of time was spent on accessing DDR3-SDRAM external memory, which was a major performance bottleneck. For the current system-level design, in addition to the AXI Master Burst, we designed and incorporated a pre-fetching technique to our user-designed hardware (in Fig. 2) in order to overcome this memory access latency issue.

The top-level block diagram of our user-designed hardware is demonstrated in Fig. 3, which consists of two separate user-designed IPs. User IP1 consists of the Step X Module (i.e., hardware module designed for each stage of the PCA computation), the slave registers, and the Read/Write module; whereas User IP2 consists of the BRAM.

User IP1 can communicate with the MicroBlaze processor using the software accessible registers known as the slave registers. Each stage of the PCA computation (Step X module) consists of a data path and a control path. Both the data and control paths have direct connections to the on-chip BRAM via user-defined interfaces. Within the User IP1, we designed a separate Read/Write (R/W) module to support the pre-fetching technique. The R/W module translates the IPIC signals to the control path and vice versa, thus reducing the complexity of the control path.

User IP2 is also designed to support the pre-fetching technique. User IP2 consists of 1 MB BRAM [45] from the Xilinx IP Core library. This dual-port BRAM supports simultaneous read/write capabilities.

3.1.1.1 During the read operation (pre-fetching): The essential data for a specific computation is pre-fetched from the DDR3-SDRAM to the on-chip BRAM. In this case, firstly, the control path sends the read request, the start address, and the burst length to the R/W module. Secondly, the R/W module asserts the necessary IPIC signals in order to read the data from SDRAM via IPIF. The R/W module waits for the ready-read acknowledgment signal from the DDR3-SDRAM. Thirdly, the data is fetched (in burst read transaction mode) from the

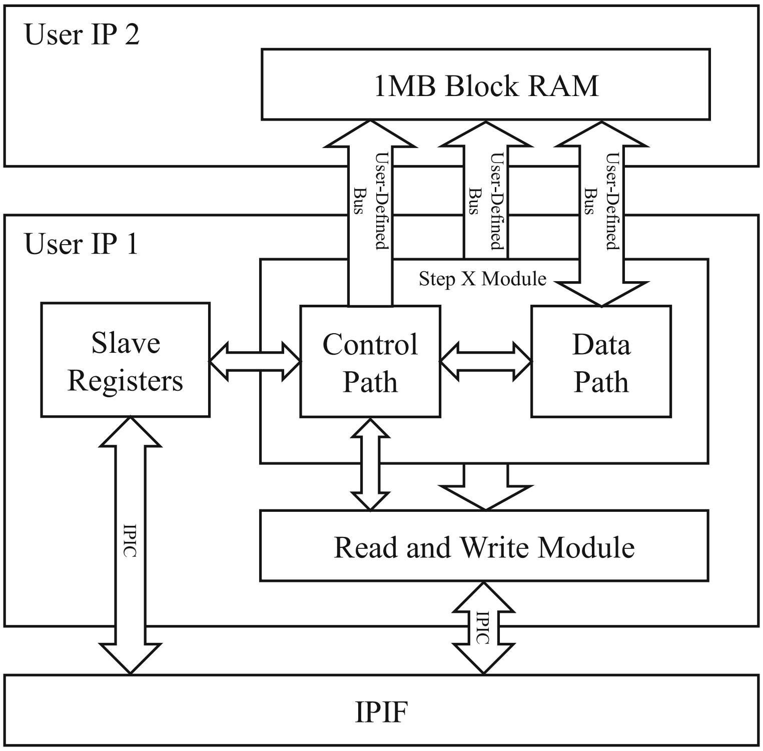

Fig. 3 Top-level user-designed hardware block diagram. The top-level module consists of our two user-designed IPs

SDRAM via R/W module and buffered to the BRAM. During this step, the control path sends the write request and the necessary addresses to the BRAM.

3.1.1.2 During the computations: Once the required data is available in the BRAM, the data is loaded to the data path in every clock cycle, and the necessary computations are performed. The control path monitors the data path and enables appropriate signals to perform the computations. The data paths are designed in pipelined fashion; hence most of the final and intermediate results are also produced in every clock cycle and written to the BRAM. Only the final results are written to the SDRAM.

3.1.1.3 During the write operation: In this case also, initially, the control path sends the write request, the start address, and the burst length to the R/W module. Secondly, the R/W module asserts the necessary IPIC signals in order to write the results to the DDR3-SDRAM via IPIF. The R/W module waits for the ready-write acknowledgment signal from the SDRAM. Thirdly, the data is buffered from the BRAM and forwarded (in burst write transaction mode) to the SDRAM via R/W module. During this step, the control path sends the read request and the necessary addresses to the BRAM.

The read/write operations from/to the BRAM are designed to overlap with the computations by buffering the data through the user-defined bus. Our current hardware designs are fully pipelined, further enhancing the throughput. All these design techniques led to higher speed-performance compared to our proof-of-concept designs. These performance analyses are presented in Section 5.2.3.

3.2 Reconfiguration process

Reconfigurable hardware designs, such as FPGA-based designs, are typically written in a hardware description language (HDL) including Verilog or VHDL [5, 17]. This abstract design has to undergo the following consecutive steps to fit into FPGA's available logic [17]: The first step is logic synthesis, which converts high-level logic constructs and behavioral code into logic gates; the second step is technology mapping, which separates the gates into groupings that match the FPGA's logic resources (generates net list); the next two consecutive steps are placement and routing, where placement allocates the logic groupings to the specific logic blocks and routing determines the interconnect resources that will carry the signals [17]. The final step is bitstream generation, which creates a "configuration bitstream" for programming the FPGA.

We can distinguish reconfigurable hardware into two types: static and dynamic. With static reconfiguration, a full configuration bitstream of an application is downloaded to the FPGA at system start-up, and the chip is configured only once and seldom changed throughout the run-time-life of the application. In order to execute a different application, a full configuration bitstream of that application has to be downloaded again and the entire chip has to be reconfigured. The system has to be interrupted for every download and reconfiguration process. With dynamic reconfiguration, a full configuration bitstream of an application is downloaded to the FPGA at system start-up, and the on-chip hardware is configured, but is often changed during the run-time-life of the application. This kind of reconfiguration allows changing either parts of the chip or the whole chip as needed on-the-fly, to perform several different computations without human intervention and in certain scenarios without interrupting the system operations.

In summary, dynamic reconfiguration has the ability to perform hardware optimization based upon present results or external stimuli determined at run-time. In addition, with dynamic reconfiguration, we can run a large application on a smaller chip by partitioning the application into sub-circuits and executing the sub-circuits on chip at different times.

3.2.1 Partial reconfiguration on Virtex 6

There are two different reconfiguration methods that can be used with Virtex-6 FPGAs: MultiBoot and Partial Reconfiguration. MultiBoot [19] is a reconfiguration method that allows full bitstream reconfiguration, whereas partial reconfiguration [52] allows partial bitstream reconfiguration. We used partial reconfiguration method for our dynamic reconfigurable hardware design.

Dynamic partial reconfiguration allows reconfiguring parts of the chip that requires modification, while interfacing with the other parts that remain operational [9, 52]. First, the FPGA is configured by loading an initial full configuration bitstream for the entire chip upon power-up. After the FPGA is fully configured and operational, multiple partial bitstreams can be downloaded simultaneously to the chip, and specific regions of the chip can be reprogrammed with new functionality "without compromising the integrity of the applications" running in the remainder of the chip [9, 52]. Partial bitstreams are used to reconfigure only selective parts of the chip. Figure 4, which is modified from [52], illustrates the basic premise of partial reconfiguration. During the design and implementation process, the logic in the reconfigurable hardware design is divided into two different parts: reconfigurable and static. As shown in Fig. 4, the functions implemented in reconfigurable modules (RM), i.e., reconfigurable parts, are replaced by the contents of the partial bitstreams (.bit files), while the current static parts remain operational, completely unaffected by the reconfiguration [52].

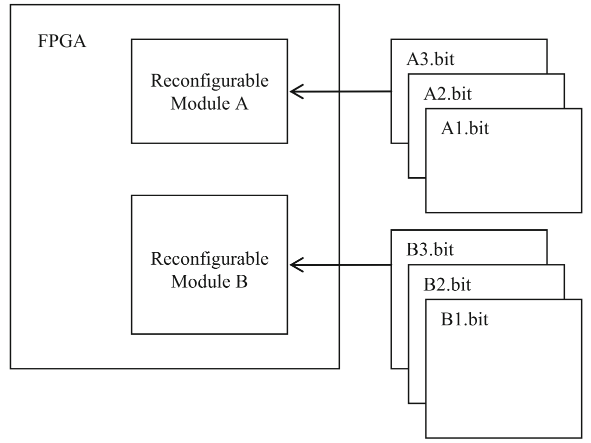

Fig. 4 Basic premise of partial reconfiguration. This figure is modified from [52]. The functions implemented in RMs are replaced by the contents of the bit files

In the late 2010s, partial reconfiguration tools used Bus Macros [26, 35] which ensures fixed routing resources for signals used as communication paths for reconfigurable parts, and when the parts are reconfigured [26]. With the PlanAhead [53] tools for partial reconfiguration, Bus Macros become obsolete. Current FPGAs (such as Virtex-6 and Virtex-7) have an important feature: a “non-glitching” (or “glitchless”) technology [9, 55]. Due to this feature, some static parts of the design could be in the reconfigurable regions without being affected by the act of reconfiguration itself, while the functionality of reconfigurable parts of the design is reconfigured [9]. For instance, when we partition a specific region and consider it as a reconfigurable part, some static interfacing might go through the reconfigurable part or some static logic (e.g., control logic) might exist in the partitioned region. These are overwritten with the exact program information, without affecting their functionalities [9, 48].

Internal Configuration Access Port (ICAP) is the fundamental module used to perform in-circuit reconfiguration [10, 55]. As indicated by its name, ICAP is an internally accessed resource and not intended for full chip configuration. As stated in [19], this module “provides the user logic access to the FPGA configuration interface, allowing the user to access configuration registers, readback configuration data, and partially reconfigure the FPGA” after initial configuration is done. The protocol used to communicate with ICAP is a subset of the SelectMAP protocol [9].

Virtex-6 FPGAs support reconfiguration via internal and external configuration ports [9, 55]. Full and partial bitstreams are typically stored in external non-volatile memory, and the configuration controller manages the loading of the bitstreams to the chip and reconfigures the chip when necessary. Configuration controller can be either a microprocessor or routines (small state machine) programmed into the FPGA. The reconfiguration can be done using a wide variety of techniques, one of which is shown in Fig. 5 (modified from [9, 52]). In our design, the full and partial bitstreams are stored in the Compact Flash (CF), and ICAP is used to load the partial bitstreams. In this design, the ICAP module is instantiated and controlled through software running on the MicroBlaze processor. During run-time, the MicroBlaze processor transmits the partial bitstreams from the non-volatile memory to the ICAP to accomplish the reconfiguration processes.

4 Embedded reconfigurable hardware design

In this section, reconfigurable hardware architecture for the PCA is introduced using partial reconfiguration. This hardware design can be dynamically reconfigured to accommodate all four stages of the PCA computations. For mobile applications such as signature verification and handwritten analysis, PCA is applied initially to reduce the dimensionality of the data, followed by similarity measure.

We investigated different stages of PCA [8, 21, 37], considered each stage as individual operations, and provided hardware support for each stage separately. We then focused on reconfigurable hardware architecture for all four stages of the PCA computation: mean, covariance matrix, eigenvalue matrix, and PC matrix computations. Our

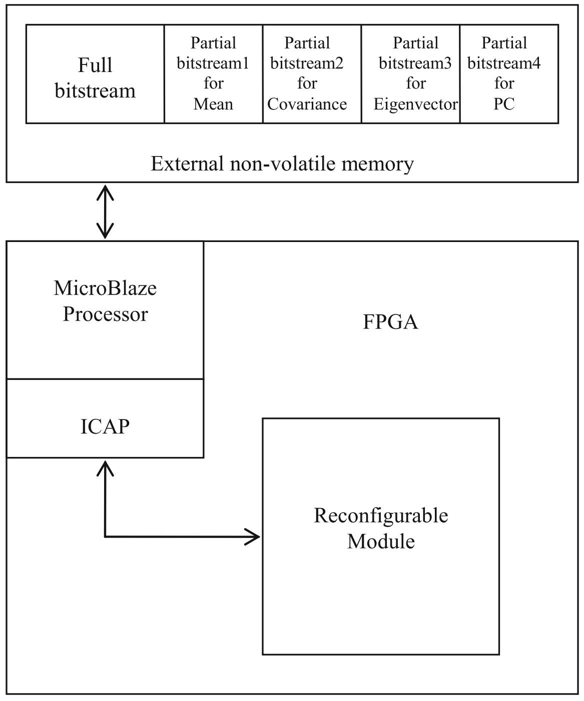
Full bitstream
Partial bitstream1 for Mean
Partial bitstream2 for Covariance
Partial bitstream3 for Eigenvector
Partial bitstream4 for PC
External non-volatile memory
MicroBlaze Processor
FPGA
ICAP
Reconfigurable Module

Fig. 5 Partial reconfiguration using MicroBlaze and ICAP. This figure is modified from [9, 52]. The full and partial bitstreams are stored in the external non-volatile memory

hardware design can be reconfigured partially and dynamically from one stage to another, in order to perform these four operations on the same area of the chip.

The equations [21, 37] for mean and covariance matrix for the PCA computation are as follows:

Equation for mean:

$$\overline{X_j} = \frac{\sum_{i=1}^{n} X_{ij}}{n} \tag{2}$$

Equation for covariance matrix:

$$Cov_{ij} = \frac{\sum_{k=1}^{n} \left(X_{ki} - \overline{X}_i\right)\left(X_{kj} - \overline{X_j}\right)}{(n-1)} \tag{3}$$

For our proof-of-concept work [32], we modified the above two equations slightly in order to use integer operations for the mean and covariance matrix computations. It should be noted that we only designed the first two stages of the PCA computation in our previous work [32]. For this research work, we are using single precision floating-point operations for all four stages of the PCA computations. The reconfigurable hardware designs for each stage consist of a data path and a control path. Each data path is designed in pipelined fashion; thus, in every clock cycle, the data is processed by one module, and the results are forwarded to the next module, and so on. Furthermore, the computations are designed to overlap with the memory access to harness the maximum benefit of the pipelined design.

The data path of the mean, as depicted in Fig. 6, consists of an accumulator and a divider. The accumulator is designed as a sequence of an adder and an accumulator register with a feedback loop to the adder. Mean is measured along the dimensions; hence the total number of mean results is equal to the number of dimensions (m) in the data set. In our design, the numerator of the mean is computed for an associated element (dimension) of each vector and only the final mean result goes through the divider.

As shown in Fig. 7, the data path of the covariance matrix design consists of a subtractor, a multiplier, an accumulator, and a divider. The covariance matrix is a square symmetric matrix, hence only the diagonal elements and the elements of the upper triangle have to be computed. Thus, the total number of covariance results is equal to $m^*(m + 1)/2$, where m is the number of dimensions. The upper triangle elements of the covariance matrix are measured between two dimensions, and the diagonal elements are measured between one dimension and itself.

In our design, the deviation from the mean (i.e., the difference matrix) is performed as the first step of the covariance matrix computation. Apart from using the difference matrix in subsequent covariance matrix computations, these results are stored in the DDR3-SDRAM via BRAM, to be reused for the PC matrix computation in stage 4. Similar to the mean design, the numerator of the covariance is computed for an element of the covariance matrix and only the final covariance result goes through the divider.

The eigenvalue matrix computation is the most complex operation from the four stages of the PCA computation. After investigating various techniques to perform EVD (presented in Section 2.2.1), we selected the QR algorithm [29] for our eigenvalue matrix computation. The data path for the eigenvalue matrix, as shown in Fig. 8, consists of several registers, two multiplexers, a multiplier, a divider, a subtractor, an accumulator, a square-root, and two comparators. The input data to this module is the *mXm* covariance matrix, and the output results from this module is a square *mXm* eigenvalue matrix.

Eigenvalue matrix computation can be illustrated using the two Eqs. (4) and (5) [29] below.

As shown in Eq. (4), the QR algorithm consists of several steps [29]. The first step is to factor the initial A matrix (i.e., the covariance matrix) into a product of orthogonal matrix $Q1$, and a positive upper triangular matrix $R1$. Second step is to multiply the two factors in the reverse order, which results in a new A matrix. Then these two steps are repeated. This is an iterative process that converges when the bottom triangle of the A matrix becomes zero. This part of the algorithm can be written as:

Equation for the QR algorithm:

$$A_1 = Q_1R_1, \quad R_kQ_k = A_{k+1} = Q_{k+1}R_{k+1} \tag{4}$$

where $k = 1,2,3,...$ and Q_k and R_k are from the previous steps, and the subsequent matrix Q_{k+1} and positive upper

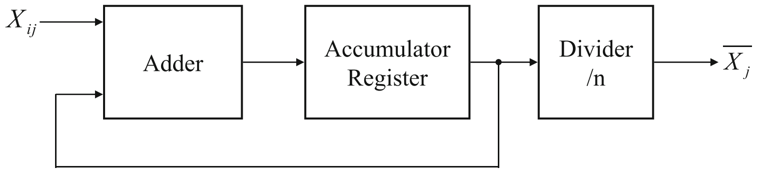

Fig. 6 Data path of the mean module. Data path consists of an accumulator and a divider

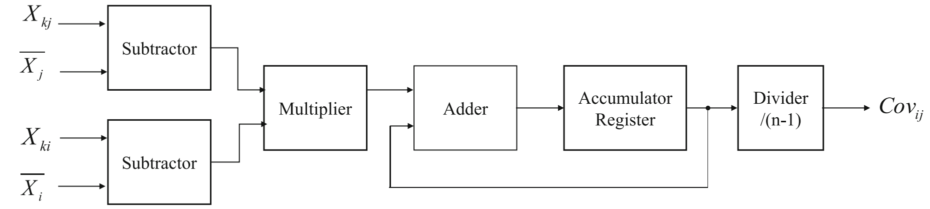

Fig. 7 Data path of the covariance matrix module. Data path consists of a subtractor, a multiplier, an accumulator, and a divider

triangular matrix R_{k+1} are computed using the numerically stable form of the Gram-Schmidt algorithm [29].

In this case, since the original A matrix (i.e., the covariance matrix) is symmetric, positive definite, and with distinct eigenvalues, then the iterations converge to a diagonal matrix containing the eigenvalues of A in decreasing order [29]. Hence, we can recursively define:

Equation for eigenvalue matrix followed by the QR algorithm:

$$S_1 = Q_1, \quad S_k = S_{k-1}Q_k = Q_1Q_2 \ldots Q_{k-1}Q_k \tag{5}$$

where $k > 1$.

During the eigenvalue matrix computation, the data is processed by four major operations before being written to the BRAM. These operations are illustrated using Eqs. (6), (7), (8), and (9), which correspond to the modules 1, 2, 3, and 4, respectively, in Fig. 8.

For operation 1 (corresponding to Eq. (6) and the module 1 in Fig. 8), the multiplication operation is performed on the input data, followed by the accumulation operation, and the intermediate result of the multiply-and-accumulate is written to the BRAM. These results are also forwarded to the temporary register for subsequent operations or to the comparator to check for the convergence of EVD.

$$A_{jk} = \sum_{i=1}^{m} B_{ji} \times C_{ik} \tag{6}$$

For operation 2 (corresponding to Eq. (7) and the module 2 in Fig. 8), the square-root operation is performed on the intermediate result of the multiply-and-accumulate, and the final result is forwarded to the BRAM. These results are also forwarded to the temporary register for subsequent operations and to the comparator to check for zero results.

$$A_{jj} = \sqrt{\sum_{i=1}^{m} {B_{ij}}^2} \tag{7}$$

For operation 3 (corresponding to Eq. (8) and the module 3 in Fig. 8), the multiplication operation is performed on the data, followed by the subtraction operation, and the result is forwarded to the BRAM.

$$A_{ik} = A_{ik} - A_{ij} \times B_{jk} \tag{8}$$

For operation 4 (corresponding to Eq. (9) and the module 4 in Fig. 8), the division operation is performed on the data, and the result is forwarded to the BRAM.

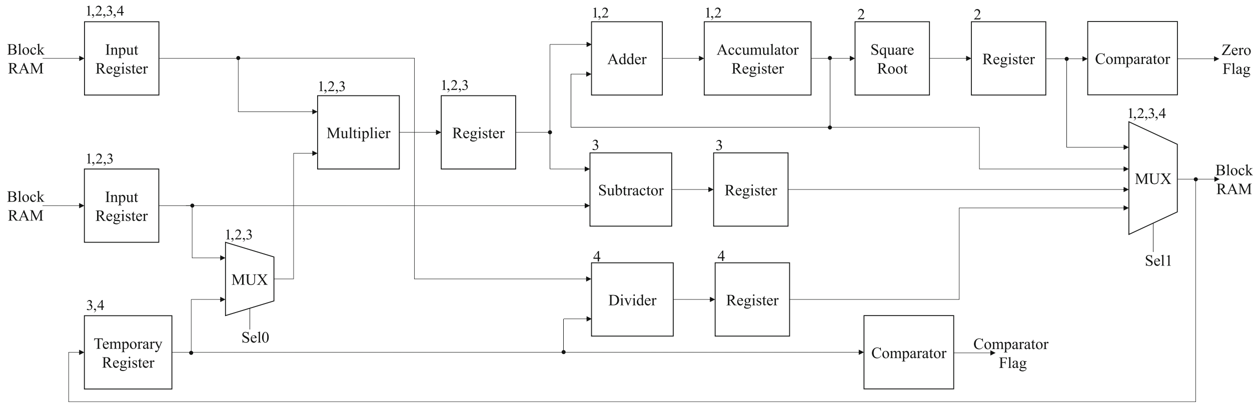

Fig. 8 Data path of the eigenvalue matrix module. Data path consists of several registers, two multiplexers, a multiplier, a divider, a subtractor, an accumulator, a square-root, and two comparators

$$A_{ij} = \frac{A_{ij}}{B_{jj}} \tag{9}$$

More details about the eigenvalue matrix computation including the QR algorithm can be found in [29].

The data path of the PC matrix computation, as depicted in Fig. 9, consists of a subtractor, a multiplier, and an accumulator. As illustrated in the PC matrix computation Eq. (10), the S elements of the eigenvalue matrix are multiplied by the difference matrix, which is already computed and stored in SDRAM in Stage 2. In this case, i^{th} raw of difference matrix is multiplied by the j^{th} column of eigenvalue matrix. Total number of PC results are equal to *m*n*, where *m* and *n* are number of dimensions and number of vectors, respectively.

Equation for PC matrix:

$$Z_{ij} = \sum_{k=1}^{m} S_{kj} \times \left(X_{ik} - \overline{X_k}\right) \tag{10}$$

The partial and dynamic reconfiguration process of the above four stages are as follows: Firstly, the full bitstream that includes the reconfigurable module (RM) of the mean design is downloaded to the FPGA and the mean computation is performed. After execution of the mean, the RM for mean sends a signal to the processor. Secondly, the processor downloads the partial bitstream for the RM for covariance matrix and the covariance computation is performed. Loading of the partial bitstreams and modifying the functionalities of the RM are done without interrupting the operations of the remaining parts of the chip. After the execution of the covariance matrix, the RM for covariance sends a signal to the processor. Thirdly, the processor downloads the partial bitstream for the RM for eigenvalue matrix computation and the eigenvalue analysis is performed. Finally, after the processor receives the completion signal from the RM of the eigenvalue matrix computation, it downloads the partial bitstream of the PC matrix computation, and the PC computation is performed.

As shown in Fig. 5, the partial bitstreams for mean, covariance matrix, eigenvalue matrix, and PC matrix modules are stored in an external non-volatile memory and downloaded to the region, of the RM, when necessary.

After processing one set of data for all four stages: mean, covariance matrix, eigenvalue matrix, and PC matrix; the processor can dynamically and partially reconfigure the chip again to the mean, without downloading the full bitstream. Thus, any number of PCA computations can be performed for any number of data sets, without interrupting the operation of the system.

5 Experimental results and analysis

5.1 Space and time analysis

In order to investigate the feasibility of our partial and dynamic reconfigurable hardware design, cost analysis on space and time is carried out for static reconfigurable hardware (SRH) and dynamic reconfigurable hardware (DRH).

5.1.1 Space saving

As illustrated in Table 1, the total number of occupied slices and the total number of DSP slices required for SRH (with mean (hw_v1a), covariance matrix (hw_v1b), eigenvalue matrix (hw_v1c), and PC matrix (hw_v1d) computations as separate entities) are 21,237 and 32, respectively. Conversely, the total number of occupied slices and the total number of DSP slices required for the DRH (hw_v2) are 6173 and 11, respectively, which is from the reconfigurable hardware design with the eigenvalue matrix computation module (i.e., the largest RM of the four).

From these analyses, it is observed that space saving using partial reconfiguration is about 71% since the same area of the chip is being reused (by reconfiguring the hardware on chip from one computation to another) for all four stages of the PCA computation in DRH design; thus saving a significant space on chip, which is crucial for mobile and embedded devices with their limited hardware footprint.

5.1.2 Space overhead for reconfiguration

As detailed in [43], AXI hardware ICAP (Internal Configuration Access Port) is used to perform in-circuit reconfiguration. It enables an embedded processor, such as the MicroBlaze, to read and write the FPGA's configuration memory through the ICAP. In our design, we used the MicroBlaze and the ICAP to fetch the full and partial configuration bitstreams from the SystemACE

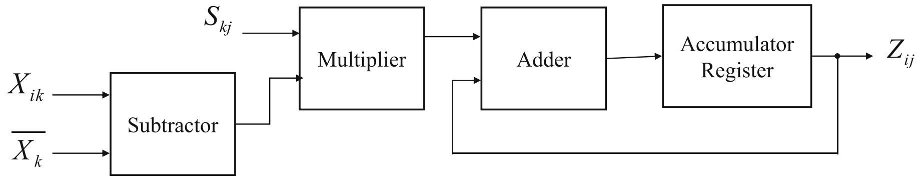

Fig. 9 Data path of the PC matrix module. Data path consists of a subtractor, a multiplier, and an accumulator

Table 1 Space statistics for various configurations: SRH vs. DRH

Configuration	Occupied area on chip	
	Number of occupied slices	Number of DSP48E1s
hw_v1a—SRH (mean as a separate entity)	4991	5
hw_v1b—SRH (covariance matrix as a separate entity)	5683	8
hw_v1c—SRH (eigenvalue matrix as a separate entity)	5352	11
hw_v1d—SRH (PC matrix as a separate entity)	5211	8
hw_v2—DRH with largest RM (eigenvalue matrix)	6173	11

compact flash (CF), and then to download and reconfigure the chip at run-time. The on-chip AXI SystemACE Interface Controller [44] (also known as the AXI SYSACE) acts as the interface between the AXI4-Lite bus and the SystemACE CF peripheral.

As mentioned in Section 3.2.1, the bus macros are obsolete with the current PlanAhead tools for partial reconfiguration [53]. Also, in our design, we are storing the full and partial bitstreams in the external CF. As a result, the only extra hardware required on chip for reconfiguration is the ICAP and the SystemAce Interface Controller. On Virtex 6, the resource utilizations for AXI ICAP [43] and the AXI SystemAce Interface Controller [44] (required for the CF) are about 436 and 46 slices, respectively, resulting in a total of 482 slices. These resource utilization numbers should be regarded as estimates, since there might be slight variations when these peripherals are combined with other designs in the system.

In summary, for our design, the reconfiguration space overhead, which is the extra hardware required on chip for reconfiguration, is constant and is about 1.28% of the chip.

5.1.3 Time overhead for reconfiguration

The reconfiguration time overhead is the time required to load and change the configuration from one computation to another. In our design, the reconfiguration time overhead is around 681 ms (from Table 3) with the MicroBlaze running at 100 MHz.

During the design and implementation with PlanAhead, the partial bitstream for the reconfigurable module is 351,216 bytes, or 2,809,728 bits. As indicated in [52], using ICAP at 100 MHz and 3.2 Gbps, a partial bit file can be loaded in about 2,809,728 bits/3.2 Gbps = 878 μs. This is significantly less than the measured 681 ms.

After further investigations, it is found that this big difference is quite normal due to the partial bitstreams being stored in the CF and also the sequential access nature of the MicroBlaze processor.

The above calculation is correct, provided that ICAP is continuously enabled. That is, the ICAP should meet the following requirements at the input of ICAP: Clk is 100 MHz and is applied continuously; Chip Enable of ICAP is asserted continuously; and write ICAP is asserted continuously and input data are given in every input Clk.

In the above scenario, the configuration uses the full bandwidth of 3.2 Gbps (100 MHz × 32 bits), and the reconfiguration can be completed within 878 μs. However, MicroBlaze executes instructions sequentially, and the partial reconfiguration sequence is as follows:

- MicorBlaze requests SystemACE controller to retrieve data from the CF.
- SystemACE controller reads data from the CF (since CF is external to the chip, there is access delay).
- MicroBlaze requests this data from SystemACE controller and stores it in an internal register.
- MicroBlaze writes the data to ICAP.

Because of this sequential execution, partial reconfiguration takes about 681 ms. Partial reconfiguration time is usually in the range of milliseconds for the bit files of size similar to this case (around 2,809,728 bits).

There is several existing research work on enhancing the ICAP architecture in order to accelerate the reconfiguration flow [16, 18, 25, 27]. We are currently investigating these architectures and design techniques, and planning to explore ways to design and incorporate similar techniques, which could potentially reduce the reconfiguration time overhead of our current reconfigurable hardware designs.

5.2 Results and analysis for SRH and DRH

We performed the experiments on Optdigit [3] benchmark dataset to evaluate both the SRH and the DRH designs. For our previous proof-of-concept experiments [32], the data were read directly from the DDR3-SDRAM, processed, and the intermediate/final results were written back to the SDRAM. This external memory access latency incurred a significant performance bottleneck. For our current experiments, the data are prefetched from the off-chip DDR3-SDRAM to the on-chip BRAM [45], processed, and some of the intermediate results are also stored in the on-chip BRAM, and the final results are written back to the SDRAM.

Our reconfigurable hardware designs (both SRH and DRH) are parameterized: i.e., the data size (*nXm*), the number of vectors (*n*), and the number of elements (*m*) of the vectors are variables, which can be changed externally, without changing the hardware architectures. The experiments are performed using various data sizes in order to examine the scalability. The number of

Table 2 Separate execution times for four stages for SRH

Data size	No. of vectors	Execution time in AXI_clk_cycles				
		Stage 1	Stage 2	Stage 3/iterations	Stage 4	Total
24,448	382	50,866	887,481	351,467,386/361	1,718,363	354,124,096
48,960	765	101,761	1,760,340	235,622,553/242	3,436,911	240,921,565
73,408	1147	152,487	2,630,976	755,125,927/775	5,150,948	763,060,338
97,856	1529	203,239	3,501,560	180,065,909/185	6,865,011	190,635,719
122,368	1912	254,095	4,374,471	259,014,857/266	8,583,585	272,227,008
146,816	2294	304,821	5,245,042	343,858,083/353	10,297,648	359,705,594
171,264	2676	355,586	6,115,652	409,170,278/420	12,011,737	427,653,253
195,712	3058	406,299	6,986,249	215,183,290/221	13,725,774	236,301,612
220,224	3441	457,194	7,859,173	254,209,810/261	5,444,335	277,970,512
244,672	3823	507,920	8,729,744	789,451,894/810	17,158,385	815,847,943

elements is kept the same, and only the number of vectors is varied to obtain various data sizes. The number of covariance results depends on the number of elements.

5.2.1 Execution times for SRH

To evaluate our dynamic reconfigurable hardware (DRH) design for the four stages of the PCA computation, we designed and implemented static reconfigurable hardware (SRH) for the mean (hw_v1a), covariance matrix (hw_v1b), eigenvalue matrix (hw_v1c), and PC matrix (hw_v1d) computations as separate entities.

Our intention is to provide hardware support for applications running on mobile and embedded devices. Considering the stringent area requirements of these devices, large and complex algorithms such as PCA might not fit into a single chip. In this case, the algorithm has to be decomposed into several stages; thus each stage will fit into the chip at a time. To illustrate this concept, for SRH, each stage is designed and implemented as separate entities with full bitstream per stage.

With the SRH design, a full bitstream consisting of the mean is downloaded and the chip is reconfigured only once. After the execution of the mean, in order to execute the covariance matrix, a full bitstream consisting of the covariance has to be downloaded and the entire chip has to be reconfigured. This process continues until all the stages are downloaded, reconfigured, and executed in the following order: mean (Stage 1) → covariance matrix (Stage 2) → eigenvalue matrix (Stage 3) → PC matrix (Stage 4). The system's operation has to be interrupted for every download and reconfiguration process.

The experiments are performed separately on SRH designs for mean, covariance matrix, eigenvalue matrix, and PC matrix as individual entities, with varying data sizes, and the execution times are obtained and

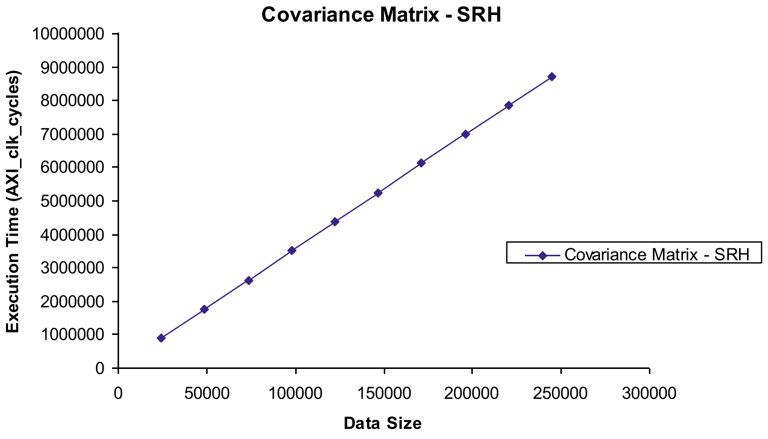

Fig. 10 Graph for covariance matrix for SRH. Execution time vs. data size for static reconfigurable hardware (SRH)

Table 3 Separate execution times for four stages for DRH

Data size	No. of vectors	Execution time in AXI_clk_cycles							
		Stage 1	S1→S2 reconfig.	Stage 2	S2→S3 reconfig.	Stage 3/iterations	S3→S4 reconfig	Stage 4	Total
24,448	382	50,879	68,103,418	887,481	68,121,324	351,467,386/361	68,112,480	1,718,389	558,461,357
48,960	765	101,761	68,097,812	1,760,340	68,108,008	235,622,553/242	68,109,251	3,436,924	445,236,649
73,408	1147	152,487	68,097,604	2,630,976	68,114,545	734,657,700/775	68,109,985	5,150,974	946,914,271
97,856	1529	203,239	68,098,307	3,501,560	68,120,760	180,065,909/185	68,108,534	6,865,011	394,963,320
122,368	1912	254,121	68,093,087	4,374,471	68,118,532	259,014,857/266	68,108,228	8,583,637	476,546,933
146,816	2294	304,821	68,097,653	5,245,029	68,117,134	343,858,083/353	68,112,569	10,297,674	564,032,963
171,264	2676	355,586	68,102,156	6,115,678	68,113,687	409,170,278/420	68,108,523	12,011,737	631,977,645
195,712	3058	406,299	68,102,063	6,986,275	68,119,069	215,183,290/221	68,110,364	13,725,761	440,633,121
220,224	3441	457,194	68,090,545	7,859,173	68,118,434	254,209,823/261	68,110,646	15,444,322	482,290,137
244,672	3823	508,843	68,074,050	8,730,693	68,095,324	789,451,907/810	68,088,858	17,159,321	1,020,108,996

presented in Table 2. The execution time for each stage is measured 10 times, and the average is presented. In this case, the total execution times (last column of Table 2) do not include the download and reconfiguration time between entities.

As illustrated in Fig. 10, the execution time for the covariance matrix (for SRH) increases linearly with the size of the data. The mean and the PC matrix showed similar linear behaviors. Similar to Fig. 12 for DRH design, the execution time for eigenvalue matrix does not increase linearly with the size of the original data set. It should be noted that the input data size to the eigenvalue matrix is an *mXm* matrix, where *m* is the number of elements of the vectors in the original data set. This *m* value is typically a constant for a certain data set; hence the input data size to the eigenvalue matrix computation is the same regardless of the size of the original data set. However, this execution time depends on and increases linearly with the number of iterations (similar to Fig. 13 for DRH).

5.2.2 Execution times for DRH

With the DRH design, a full bitstream, which consists of the reconfigurable module (RM) of mean, is downloaded, and the mean operation is performed. After the execution of the mean, the partial bitstream for the RM of the covariance matrix is downloaded to the specific region of the chip consisting of the mean module, and that region is reconfigured to the covariance matrix operation. Then the covariance matrix operation is performed. This partial and dynamic reconfiguration process continues until all the stages of the PCA computation are downloaded, reconfigured, and executed in the following order: mean (Stage 1) → covariance matrix

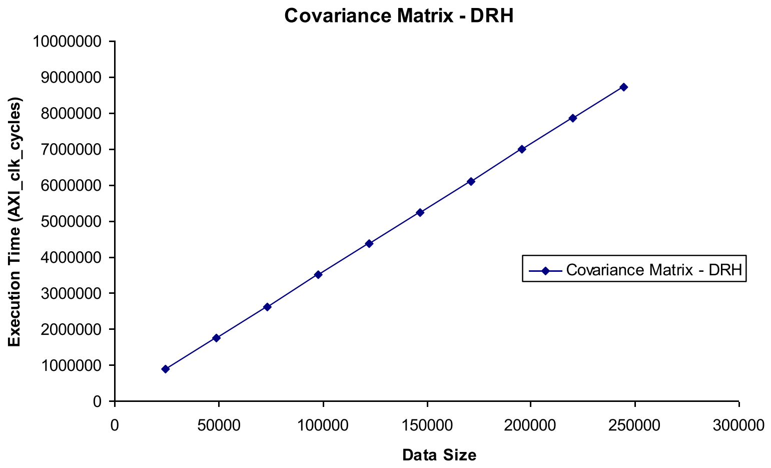

Fig. 11 Graph for covariance matrix for DRH. Execution time vs. data size for dynamic reconfigurable hardware (DRH)

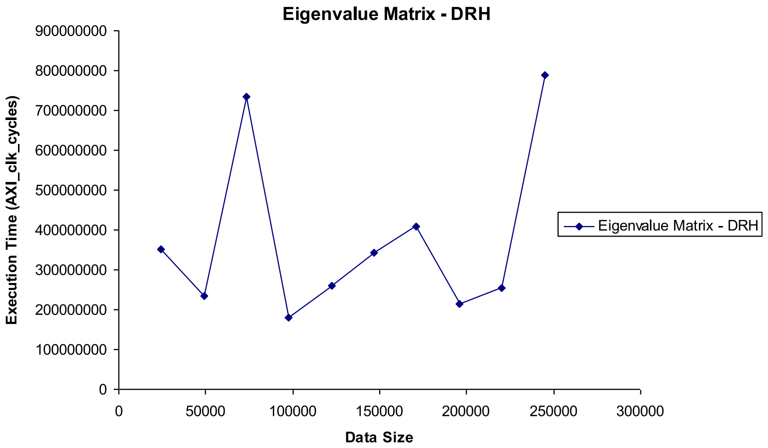

Fig. 12 Graph for eigenvalue matrix for DRH (execution time vs. data size). Execution time vs. data size for dynamic reconfigurable hardware (DRH)

(Stage 2) → eigenvalue matrix (Stage 3) → PC matrix (Stage 4). In order to process varying data sizes or different data sets, the hardware is again reconfigured to the first PCA computation, i.e., mean operation, without downloading the full bitstream or without interrupting the system's operation.

Unlike SRH designs, for DRH designs, the execution times are measured sequentially for consecutive stages of the PCA computations. The execution times for varying data sizes are measured 10 times, and the average is presented in Table 3. The reconfiguration time overhead from one stage to another are presented in columns 4, 6, and 8. The values in these three columns demonstrate that the reconfiguration time does not vary with the size of the data set. Although there is a slight variation, it is less than 1 ms. This is expected, since the reconfiguration time does not depend on the number of data being processed but on the size of the partial bitstream (i.e., the area of the RM).

Similar to the SRH designs, as depicted in Fig. 11, the execution time for the covariance matrix for DRH increases linearly with the size of the data set. The mean and the PC matrix showed similar linear behaviors. Furthermore, as shown in Fig. 12, the execution time for eigenvalue matrix for DRH does not increase linearly with the size of the original data set, since the input data size to the eigenvalue matrix computation is an *mXm* matrix, and is the same regardless of the size of the original data set (i.e., the number of vectors). However, as demonstrated in Fig. 13, this execution time increases linearly with the number of iterations.

From Table 3, it is evident that the eigenvalue matrix, which is the largest and most complex design, takes the

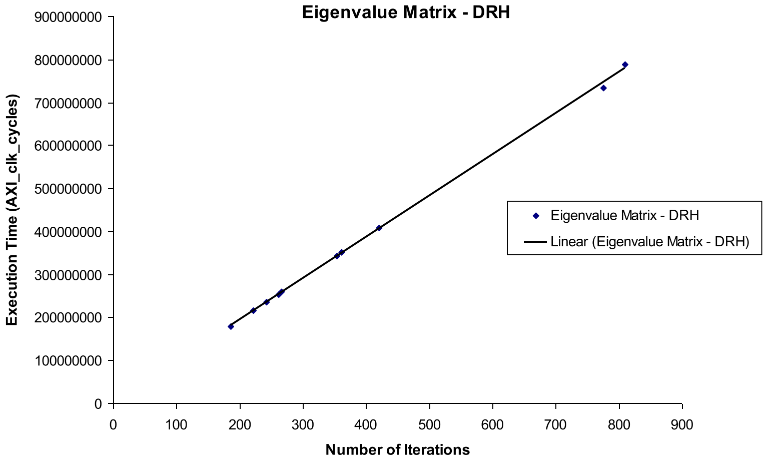

Fig. 13 Graph for eigenvalue matrix for DRH (execution time vs. number of iterations). Execution time vs. number of iterations for dynamic reconfigurable hardware (DRH)

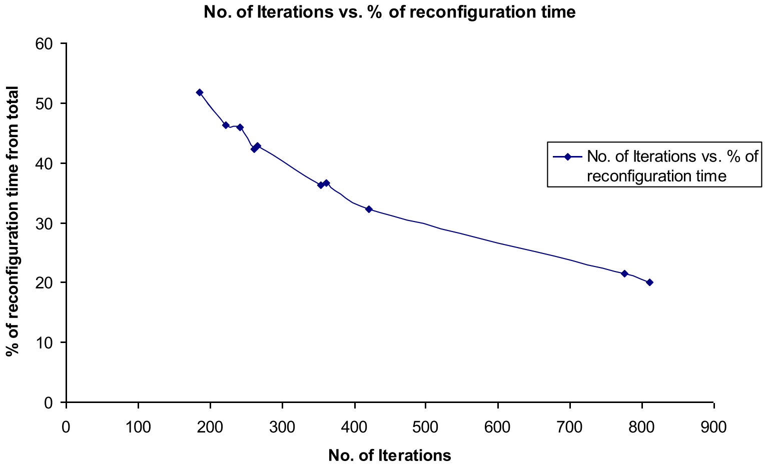

Fig. 14 Graph for number of iterations vs. percentage of reconfiguration time from total execution time. The percentage of reconfiguration time is amortized and decreases as the number of iterations increases

longest time to process compared to other three stages, thus impacting the total execution time. As a result, the total execution time for the whole process increases linearly with the number of iterations.

Figure 14 illustrates the percentage of reconfiguration time (from the total execution time) vs. the number of iterations for the DRH. For lower number of iterations, a significant percentage of total time is spent on the reconfiguration. However, the percentage of reconfiguration time is amortized and decreases as the number of iterations increases. This illustrates that the more compute-intensive the operations are, the smaller the impact of the reconfiguration time overhead is.

5.2.3 Speed-performance comparison: SRH and DRH vs. software on MicroBlaze

Additional software experiments are performed using the MicroBlaze soft processor on the same platform, in order to evaluate the DRH as well as the SRH. Similar to the DRH designs, the execution times for the software designs are also measured in a sequence: mean (Stage 1) → covariance matrix (Stage 2) → eigenvalue (Stage 3) → PC (Stage 4). Execution times are obtained using varying data sizes. However, only the largest data set of size 244,672 is used (presented in Table 4) for the performance comparison purposes for the two reconfigurable hardware designs and the software designs.

From Table 4, considering the total execution times, the DRH is 53 times faster, while the SRH is 66 times faster than the equivalent software (Sw) running on the MicroBlaze. This difference is due to the time overhead incurred for reconfiguration for DRH. Although our SRH is faster than our DRH, the space saving (as demonstrated in Table 1) using the dynamic and partial reconfiguration is significant. It is important to consider these speed-space tradeoffs, especially in mobile and embedded devices with their limited hardware footprint.

Considering the execution times for individual modules, SRH and DRH achieved similar speedups, and the speedups vary from 60 to 79. It is evident that our current reconfigurable hardware designs achieved superior speedups (79 times faster than software on MicroBlaze), compared to our previous proof-of-concept designs (6 times faster than software on MicroBlaze) [32]. This significant improvement of speedups is due to several hardware optimization techniques we incorporated in our current designs including:

Table 4 Performance Comparison: SRH and DRH vs. Software on MicroBlaze

	Execution time in AXI_clk_cycles			Speedup	
	SRH	DRH	Sw on MicroBlaze	SRH vs. Sw	DRH vs. Sw
Stage 1	507,920	508,843	30,564,884	60.18	60.07
Stage 2	8,729,744	8,730,693	686,064,039	78.59	78.58
Stage 3	789,451,894	789,451,907	51,909,640,837	65.75	65.75
Stage 4	17,158,385	17,159,321	1,246,451,248	72.64	72.64
Total	815,847,943	1,020,108,996	53,872,721,008	66.03	52.81

fully pipelined designs, designing computations to overlap with memory access, and burst transfer and pre-fetching techniques to reduce the memory access latency.

6 Conclusions

In this paper, we introduced reconfigurable hardware architecture for PCA using partial reconfiguration method, which can be partially and dynamically reconfigured from mean → covariance matrix → eigenvalue matrix → PC matrix computations. This design showed a significant space saving (about 71%), since the same area of the chip is being reused (by reconfiguring the hardware on chip from one computation to another) for all the four stages of PCA computation, which is crucial for mobile and embedded devices with their limited hardware footprint.

The extra hardware required for reconfiguration is relatively low compared to the whole chip (about 1.28%) and remains constant regardless of the size of the reconfiguration module. Considering the reconfiguration time overhead, there is a difference between the theoretical estimate and the experimental value. This is mainly because we used a MicroBlaze processor as a configuration controller, which executes instruction sequentially. We could potentially get similar values as the theoretical ones, by using a FSM as the configuration controller and downloading the configuration bitstream using "bit-parallel" mode. Furthermore, we are investigating the existing ICAP architectures and design techniques used in [16, 18, 25, 27] and planning to explore ways to design and incorporate similar techniques, to enhance the reconfiguration process.

Our current reconfigurable hardware designs executed up to 79 times faster than the equivalent software running on the embedded microprocessor. This is a significant improvement from our proof-of-concept designs [32], which executed up to 6 times faster than their software counter parts. From our proof-of-concept work [32], it was observed that a large amount of time (93–95%) was spent on data transfer to/from the external memory, which used to be a major performance bottleneck. This substantial improvement of speedups is mainly due to several hardware optimization techniques we incorporated in our current designs: burst transfer and pre-fetching techniques to reduce the memory access latency, fully pipelined designs, designing computations to overlap with memory access.

Our proposed hardware architectures are generic, parameterized, and scalable. Hence, without changing the internal hardware architecture, our hardware designs can be used to process different data sets with varying number of vectors and with varying number of dimensions; used for any embedded applications that employ the PCA computation; executed on different development platforms, including platforms with recent FPGAs such as Virtex-7 chips.

Power consumption is another major issue in mobile and embedded devices. As demonstrated in [30], although reconfigurable hardware typically consumes less power than microprocessor-based software-only designs, we are planning and designing experiments to evaluate the power consumption in reconfigurable hardware designs for data mining applications.

The results shown in our experiments are encouraging and demonstrate great potential in implementing data mining applications such as PCA computation using reconfigurable platform. Complex applications can indeed be implemented in reconfigurable hardware for mobile and embedded applications.

Authors' contributions

DGP has been conducting this research and performed the proof-of-concept work. SNS is DGP's student. DGP and SNS have designed the SRH and DRH as well as the software for the PCA computation. Under the guidance of DGP, SNS has implemented the reconfigurable hardware for the PCA computations and performed the experiments. DGP wrote the paper. Both authors read and approved the final manuscript.

Authors' information

S. Navid Shahrouzi received his M.Sc. and B.Sc. degrees in Electronics and Electrical Engineering from University of Guilan (Iran) in 2007 and K.N.Toosi University of Technology (Iran) in 2004, respectively. Navid is pursuing his Ph.D. and working as a research assistant in the Department of Electrical and Computer Engineering, University of Colorado under the guidance of Dr. Darshika G. Perera. His research interests are digital systems and hardware optimization.

Darshika G. Perera is an Assistant Professor in the Department of Electrical and Computer Engineering, University of Colorado, USA, and also an Adjunct Assistant Professor in the Department of Electrical and Computer Engineering, University of Victoria, Canada. She received her Ph.D. degree in Electrical and Computer Engineering from University of Victoria (Canada), and M.Sc. and B.Sc. degrees in Electrical Engineering from Royal Institute of Technology (Sweden) and University of Peradeniya (Sri Lanka), respectively. Prior to joining University of Colorado, Darshika worked as the Senior Engineer and Group Leader of Embedded Systems at CMC Microsystems, Canada. Her research interests are reconfigurable computing, mobile and embedded systems, data mining, and digital systems. Darshika received a best paper award at the IEEE 3PGCIC conference in 2011. She serves on organizing and program committees for several IEEE/ACM conferences and workshops and as a reviewer for several IEEE, Springer, and Elsevier journals. She is a member of the IEEE, the IEEE Computer Society, and the IEEE Women in Engineering.

Competing interests

The authors declare that they have no competing interests.

Received: 14 July 2016 Accepted: 8 February 2017
Published online: 21 February 2017

References

1. JFD Addison, S Wermter, GZ Arevian, A comparison of feature extraction and selection techniques, in *Proc. of Int. Conf. on Artificial Neural Networks (ICANN)*, 2003, pp. 212–215
2. Agilent Technologies, Inc, *Principal component analysis*, 2005, Santa Clara, CA, USA http://sorana.academicdirect.ro/pages/collagen/amino_acids/materials/PCA_1.pdf. Accessed in June 2016
3. E Alpaydin, C Kaynak, Optical recognition of handwritten digits data set. Available in UCI Machine Learning Repository, July 1998
4. P Berkhin, Survey of clustering data mining techniques. Technical Report, Accrue Software, 2002
5. K Compton, S Hauck, Reconfigurable computing: a survey of systems and software. ACM Computing Surveys (CSUR) **34**(2), 171–210 (2002)
6. Data mining: what is data mining? http://www.anderson.ucla.edu/faculty/jason.frand/teacher/technologies/palace/datamining.htm. Accessed in June 2016

7. J DeCoster, *Overview of factor analysis*, 1998. http://www.stat-help.com/notes.html. Accessed in June 2016
8. CHQ Ding, X He, Principal component analysis and effective k-means clustering, in *Proc. of SDM*, 2004
9. D Dye, *Partial reconfiguration of Xilinx FPGAs using ISE design suite, WP374 (v1.1)*, 2011
10. V Eck, P Kalra, R LeBlanc, J McManus, *In-circuit partial reconfiguration of rocketIO attributes XAPP662 (v2.4)*, 2004
11. K Fukunaga, *Introduction to statistical pattern recognition*, 2nd edn. (Academic, New York, 1990)
12. P Garcia P, K Compton, M Schulte , E Blem, and W Fu. An overview of reconfigurable hardware in embedded systems. EURASIP Journal on Embedded Systems, (2006), 1–19
13. A Gnanabadkaran, K Duraiswamy, An efficient approach to cluster high dimensional spatial data using K-mediods algorithm. European Journal of Scientific Research **49**(4), 617–624 (2011)
14. GH Golub, CF van Loan, *Matrix computations*, 3rd edn. (John Hopkins University Press, Baltimore, 1996)
15. DJ Hand, H Mannila, P Smyth, *Principles of data mining* (The MIT Press, Cambridge, 2001)
16. SG Hansen, D Koch, J Torresen, High speed partial run-time reconfiguration using enhanced ICAP hard macro, in *Proc. of IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW)*, 2011, pp. 174–180
17. S Hauck, and A Dehon Reconfigurable computing: the theory and practice of FPGA-based computing (Morgan Kaufmann Publishers Inc. San Francisco, CA, USA., 2008)
18. M Hübner, D Göhringer, J Noguera, J Becker, Fast dynamic and partial reconfiguration data path with low hardware overhead on Xilinx FPGAs, in *Proc. of Reconfigurable Architectures Workshop (RAW'10)*, 2010
19. J Hussein, R Patel, *MultiBoot with Virtex-5 FPGAs and Platform Flash XL, XAPP1100 (v1.0)*, 2008
20. A Hyvarinen, A survey on independent component analysis. Neural Computing Survey **2**, 94–128 (1999)
21. JE Jackson, A user's guide to principal components (John Wiley & Sons, Inc. Publications, Wiley-Interscience, Hoboken, New Jersey, USA, 2003)
22. AK Jain, MN Murty, PJ Flynn, Data clustering: a review. ACM Computing Surveys **31**(3), 264–323 (1999)
23. IT Jolliffe, *Principal component analysis* (Springer, New York, 2002)
24. HP Kriegel, P Kröger, A Zimek, Clustering high-dimensional data: a survey on subspace clustering, pattern-based clustering, and correlation clustering. ACM Transactions on Knowledge Discovery from Data, (TKDD), New York, NY, USA, **3**(1), 1–58 (2009)
25. V Lai, O Diessel, ICAP-I: a reusable interface for the internal reconfiguration of Xilinx FPGA, in *Proc. of International Conference on Field-Programmable Technology (FPT)*, 2009, pp. 357–360
26. D Lim, M Peattie, *Two flows for partial reconfiguration: module based and small bit manipulation, XAPP290*, 2002
27. M Liu, W Kuehn, Z Lu, A Jantsch, Run-time partial reconfiguration speed investigation and architectural design space exploration, in *Proc. of IEEE International Workshop on Field Programmable Logic and Applications (FPL)*, 2009, pp. 498–502
28. DC Manning, P Raghvan, and H Schutze, Introduction to information retrieval. Cambridge University Press (2008)
29. PJ Olver, Orthogonal bases and the QR algorithm. University of Minnesota, (2008)
30. DG Perera, KF Li, Analysis of single-chip hardware support for mobile and embedded applications, in *Proc. of IEEE Pacific Rim Int. Conf. on Communication, Computers, and Signal Processing*, 2013, pp. 369–376
31. DG Perera, KF Li, Embedded hardware solution for principal component analysis, in *Proc. of IEEE Pacific Rim Int. Conf. on Communication, Computers and Signal Processing (PacRim'11)*, 2011, pp. 730–735
32. DG Perera, and KF Li, FPGA-based reconfigurable hardware for compute intensive data mining applications, In Proc. of 6th IEEE Int. Conf. on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC'11), 2011, pp.100-108. (Best Paper Award)
33. K Reddy, T Herron, Computing the eigen decomposition of a symmetric matrix in fixed-point arithmetic, in *Proc. of 10th Annual Symp. on Multimedia Communication and Signal Processing*, 2001
34. G Salton, MJ McGill, *Introduction to modern information retrieval* (McGraw-Hill, New York, 1983)
35. P Sedcole, B Blodget, T Becker, J Anderson, P Lysaght, Modular dynamic reconfiguration in Virtex FPGAs. IEE Computers and Digital Techniques **153**(3), 157–164 (2006)
36. A Sharma, KK Paliwal, Fast principal component analysis using fixed-point algorithm. Pattern Recognition Letters **28**(10), 1151–1155 (2007)
37. J Shlens, A tutorial on principal component analysis. Institute on Nonlinear Science, UCSD, Salk Insitute for Biological Studies, La Jolla, CA, USA. (2005). http://www.cs.cmu.edu/~elaw/papers/pca.pdf. Accessed in June 2016
38. LI Smith, A tutorial on principal component analysis. Cornell University, 2002
39. M Thangavelu, R Raich, On linear dimension reduction for multiclass classification of Gaussian mixtures, in *Proc. of IEEE Int. Conf. on Machine Learning and Signal Processing*, 2009, pp. 1–6
40. TJ Todman, GA Constantinides, SJE Wilton, O Mencer, W Luk, PYK Cheung, Reconfigurable computing: architectures and design methods. IEE Computer and Digital Techniques **152**(2), 193–207 (2005)
41. LN Trefethen, and D Bau, Numerical linear algebra. (SIAM Bookstore, Philadelphia, PA, USA, 1997)
42. P Valarmathie, MV Srinath, K Dinakaran, An increased performance of clustering high dimensional data through dimensionality reduction technique. Theoretical and Applied Information Technology **5**(6), 731–733 (2005)
43. Xilinx, Inc., LogiCORE IP AXI HWICAP, DS817 (v2.03.a) (2012). http://www.xilinx.com/support/documentation/ip_documentation/axi_hwicap/v2_03_a/ds817_axi_hwicap.pdf. Accessed in June 2016
44. Xilinx, Inc., LogiCORE IP AXI System ACE Interface Controller, DS789 (v1.01.a) (2012). http://www.xilinx.com/support/documentation/ip_documentation/ds789_axi_sysace.pdf. Accessed in June 2016
45. Xilinx, Inc., LogiCORE IP Block Memory Generator, PG058 (v7.3) (2012). http://www.xilinx.com/support/documentation/ip_documentation/blk_mem_gen/v7_3/pg058-blk-mem-gen.pdf. Accessed in June 2016
46. Xilinx, Inc., LogiCORE IP AXI Interconnect, DS768 (v1.06.a) (2012). http://www.xilinx.com/support/documentation/ip_documentation/axi_interconnect/v1_06_a/ds768_axi_interconnect.pdf. Accessed in June 2016
47. Xilinx, Inc., LogiCORE IP AXI Master Burst (axi_master_burst), DS844 (v1.00.a) (2011). http://www.xilinx.com/support/documentation/ip_documentation/axi_master_burst/v1_00_a/ds844_axi_master_burst.pdf. Accessed in June 2016
48. Xilinx, Inc., LogiCORE IP AXI SystemACE Interface Controller, DS789 (v1.01.a) (2011). http://www.xilinx.com/support/documentation/ip_documentation/ds789_axi_sysace.pdf. Accessed in June 2016
49. Xilinx, Inc., LogiCORE IP AXI Timer, DS764 (v1.03.a) (2012). http://www.xilinx.com/support/documentation/ip_documentation/axi_timer/v1_03_a/axi_timer_ds764.pdf. Accessed in June 2016
50. Xilinx, Inc., LogiCORE IP Floating-Point Operator, DS335 (v5.0) (2011). http://www.xilinx.com/support/documentation/ip_documentation/floating_point_ds335.pdf. Accessed in June 2016
51. Xilinx, Inc., ML605 Hardware User Guide, UG534 (v1.5) (2011). www.xilinx.com/support/documentation/boards_and_kits/ug534.pdf, Accessed in June 2016
52. Xilinx, Inc., Partial Reconfiguration User Guide UG702 (v12.3) (2010). http://www.xilinx.com/support/documentation/sw_manuals/xilinx12_3/ug702.pdf. Accessed in June 2016
53. Xilinx, Inc., PlanAhead User Guide, UG632 (v 11.4) (2009). http://www.xilinx.com/support/documentation/sw_manuals/xilinx11/PlanAhead_UserGuide.pdf. Accessed in June 2016
54. Xilinx, Inc., Virtex 6 FPGA Memory Interface Solutions, DS186 (v1.03.a) (2012). http://www.xilinx.com/support/documentation/ip_documentation/mig/v3_92/ds186.pdf. Accessed in June 2016
55. Xilinx, Inc., Virtex-6 FPGA Configuration User Guide UG360 (v3.2) (2010). http://www.xilinx.com/support/documentation/user_guides/ug360.pdf. Accessed in June 2016
56. JT Yao, Sensitivity analysis for data mining, in *Proc. of 22nd Int. Conf. of Fuzzy Information Processing Society*, 2003, pp. 272–277
57. KY Yeung, and WL Ruzzo, Principal component analysis for clustering gene expression data. Bioinformatics. **9**, 763-774, (2001)