Hardware Acceleration of Microwave Imaging Algorithms

Chapter 1
Hardware Acceleration of Microwave Imaging Algorithms

Mohammad Amir Mansoori and Mario R. Casu

Abstract Microwave Imaging (MWI) is a technique that allows to reconstruct an image of the internal structure of an object by irradiating the object with a known incident field and by acquiring and processing the scattered field. As such, MWI can be used as a diagnostic technique for various medical issues, such as brain stroke or breast cancer. However, the image reconstruction process entails a sequence of compute-intensive algorithms, which we call "kernels." The kernels execution time might become an issue when the time to diagnosis is a key factor. To speed up the kernels execution, we can use hardware acceleration techniques. In this chapter, we will identify the complex computational kernels that are recurrent in MWI. Then, we present the methodologies to design specific hardware accelerators in Field-Programmable Gate Arrays (FPGAs) for these kernels by means of an efficient design method called *High Level Synthesis (HLS)* and by several HLS-based hardware optimization techniques. In addition, we will present new methodologies for design automation in HLS-based hardware designs. The results show that the presented HLS optimizations and design automation techniques can significantly improve the efficiency of the hardware accelerator designs for MWI.

1.1 Introduction

Microwave Imaging (MWI) in medical applications can be used to obtain some information about the internal structure of the body by using electromagnetic fields at microwave frequencies. Although other imaging modalities such as CT-scan or MRI are more common in medical diagnosis, MWI has attracted attention in recent years because of its non-invasive, non-ionizing, and low-cost characteristics. There are mainly two methods to obtain the properties of an object by MWI: *Qualitative Imaging* and *Quantitative Imaging* methods.

Mohammad Amir Mansoori · Mario R. Casu e-mail: name@email.address
Name, Address of Institute e-mail: name@email.address

In ***Qualitative Imaging***, only the information such as shape or location of any anomalies in the object are determined. For example, in brain stroke imaging, the anomaly is a brain stroke that must be ultimately classified into ischemic or hemorrhagic stroke. By means of Qualitative Imaging it is possible to identify the position, and shape of the stroke, but not the type (ischemic or hemorrhagic), as this information is related to the value of the dielectric properties of the stroke.

On the contrary, in ***Quantitative Imaging*** a complete characterization of the targets in terms of morphology and values of the electromagnetic properties is achieved by processing the microwave measurements and by solving an *Inverse Scattering* problem. In spite of the non-linear nature of inverse scattering, one of the common approaches to tackle this problem is to linearize it by using the Born approximation. However, in highly heterogeneous objects, the linearized solutions are not effective and non-linear algorithms can provide better results. A common non-linear method is to iteratively use a Forward Solver to estimate the microwave measurements and a linear Inverse Solver to update the dielectric profile.

In addition, ***Machine Learning (ML)*** as a new category is recently introduced, which is used to learn both qualitative and quantitative information from the observations. ML algorithms are useful to obtain Qualitative or Quantitative information in MWI. The general processing steps of ML methods is as follows. After pre-processing the data (i.e., the microwave measurements), feature extraction techniques, such as for example Principal Component Analysis (PCA), can be used to obtain the most important information from data. The final stage in ML processing steps can be an image reconstruction, or the detection followed by classification of an anomaly. Different ML classifiers exist including Support Vector Machine (SVM), Random Forest (RF), and Neural Networks (NN), while Generative Adversarial Networks (GANs) can be used to reconstruct an image from microwave data.

1.1.1 Problem Statement

High execution time of MWI algorithms (especially for Quantitative, or ML methods) makes it difficult to use them in real-time biomedical systems or, in general, when the time-to-diagnosis is a critical factor. The complexity of each MWI algorithm is due to some compute-intensive parts that we call “kernels.” These kernels can be implemented in hardware accelerators to increase the speed of execution and so to improve the overall performance of an MWI-based biomedical system.

Field Programmable Gate Arrays (FPGAs) are powerful hardware accelerators used in different applications and that can be also used in MWI [1–3]. Compared to Graphical Processing Units (GPUs), FPGAs have the advantage of lower power consumption for a given performance level [4]. However, they are notoriously more difficult to program, as in general programming an FPGA requires an expertise in digital hardware design that GPU programmers do not need to possess. For the implementation of an algorithm in FPGA, however, the recent trend is to use High Level Synthesis (HLS) tools that are able to convert a software code written in C or

C++ into its corresponding hardware implementation [5]. The advantage is the faster design and development process compared to the traditional approach of manual hardware design.

In this chapter, we first identify a comprehensive set of computational kernels that are recurrent in MWI algorithms. After that, we present the methodologies to design specific hardware accelerators in FPGAs for these kernels with the HLS approach. To design efficient hardware accelerators, we introduce several HLS-based hardware optimization techniques. In addition, we present new methodologies for design automation in HLS-based hardware designs that can increase the design efficiency by allowing designers to find the optimum hardware configurations in a relatively short time. For the evaluation of hardware accelerators, we compare the performance of FPGAs with CPU and GPU-based designs in terms of processing time, power consumption, cost, form factor, and resource usage. In addition, the impact of different HLS optimization methods on the hardware performance is thoroughly explored. The results show that the presented HLS optimizations and design automation techniques can significantly improve the efficiency of the hardware accelerator designs for microwave imaging.

1.2 Microwave Imaging system

An MWI system uses a set of antennas emitting microwave radiations towards a given body part and collecting the reflected electromagnetic field. The discontinuity caused by the difference between the dielectric properties of different body tissues is the root cause of these reflections, a phenomenon termed *scattering*. By acquiring and processing by means of various methods the reflected signals, it is possible to either reconstruct an image that, in turn, can highlight the presence of an anomaly within the irradiated body part, or to directly detect and classify such anomaly.

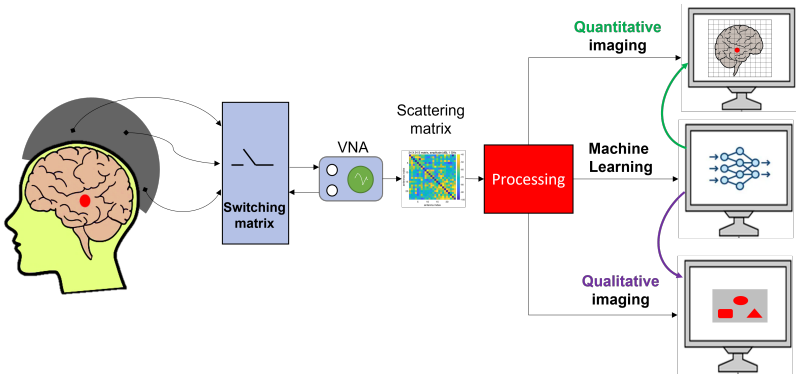

Fig. 1.1 General diagram of a Microwave Imaging system.

The components of an MWI system are illustrated in Fig. 1.1. The first component is a set of antennas arranged around the body part under consideration. A switching matrix is connected to these antennas, specifying which pairs of antennas must be active during the measurements. The switching matrix is connected to a Vector Network Analyzer (VNA) that measures the microwave radiations in the form of a scattering matrix. The last component of the device is a processing system that is used to convert the captured microwave radiations into the required information for medical diagnosis. The algorithms used in the processing system are usually compute-intensive and can benefit from hardware acceleration.

1.3 Compute-intensive kernels in Microwave Imaging

In this section, the compute-intensive kernels that are commonly used in biomedical MWI algorithms are introduced. In addition, a brief overview of the relevant previous works and their limitations will be presented. In the subsequent sections, the details of the proposed hardware accelerators for these kernels will be explained.

1.3.1 3D FDTD

Finite Difference Time Domain (FDTD) is a method to solve the Maxwell equations, which model the propagation of electromagnetic fields, by means of finite differences. As well known, the Maxwell equations state that any variation in the magnetic field creates an electric field and vice versa, as illustrated in Fig. 1.2.

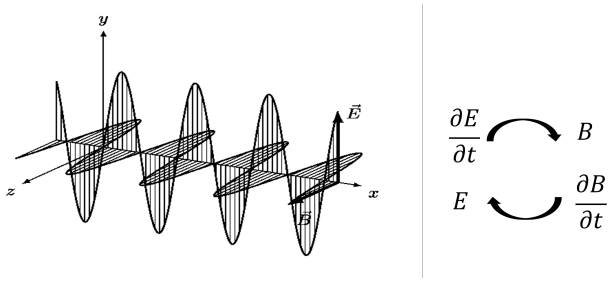

Fig. 1.2 Propagation of electromagnetic fields and the mutual dependency between electric and magnetic fields.

Several recent works propose GPU as FDTD hardware accelerator [6–9]. High power consumption of GPUs pushes hardware designers to use FPGA accelerators instead. More details on FPGA implementations of FDTD can be found in [10–15].

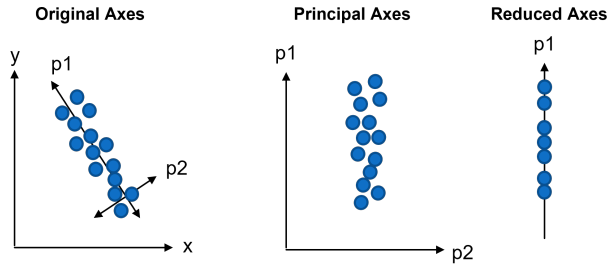

Fig. 1.3 PCA for dimensionality reduction to remove redundant information.

The above-mentioned FDTD accelerators have some limitations. Most of them considered simple boundary conditions for the Maxwell equations, such as Periodic Boundary Conditions (PBC), which cannot be used in medical MWI because they lead to a lower accuracy: more complex boundary conditions like Convolutional Perfectly Matched Layer (CPML) are required instead. In previous FPGA accelerators of FDTD designed in high level description language, CPML conditions were considered only in two directions (top and bottom) of the domain in which the equations are solved, while for the other directions PBC conditions were used. In addition, none of the previous works considered the impact of polarization currents in *dispersive* materials, which is important for a more complex and accurate modeling of propagation but complicates the design of hardware accelerators for FDTD.

1.3.2 PCA using SVD/EVD

PCA is used in ML for feature extraction and to eliminate redundant information from data. Its computation consists of different steps, which include Singular value or Eigenvalue Decomposition (SVD/EVD). When the dimensions of data is large, PCA is highly beneficial to reduce the data dimension. As shown in Fig. 1.3, by transforming the original data into its corresponding principal axes, it is possible to ignore the horizontal axis and reduce the dimensions of data. In MWI, PCA can remove redundancy in scattering matrices collected at different frequencies.

Different FPGA accelerators have been recently proposed for PCA, but most of them have been implemented starting from a manually obtained Register Transfer Level (RTL) design [16–19]. A few works concentrated on the acceleration of only some components of PCA such as EVD or SVD [20–26]. Other references used High Level Synthesis for the design of PCA accelerator [27–30].

Differently from our flexible PCA accelerator, most of the previous works did not propose to implement PCA in its entirety in a way to support different data dimensions. Instead, our FPGA accelerator can be used in different applications including MWI supporting large data dimensions.

1.3.3 SVM

One of the widely-used classification techniques in ML is Support Vector Machine (SVM). As illustrated in Fig. 1.4, SVM finds a decision boundary that can maximize the margin between two classes. Among all the possible decision boundaries (L1 to L3 in Fig. 1.4), only one line is optimum (L2) and can be obtained after the *training phase* of SVM. In the *inference phase,* new data points will be classified based on the previously obtained decision boundary. In addition to binary classification, SVM can be used in multi-class classification by using a *Kernel* function which transforms the data points into a higher dimension space in which the samples are linearly separable. In MWI, SVM can be used for the detection and classification of different anomalies such as breast cancer or brain stroke.

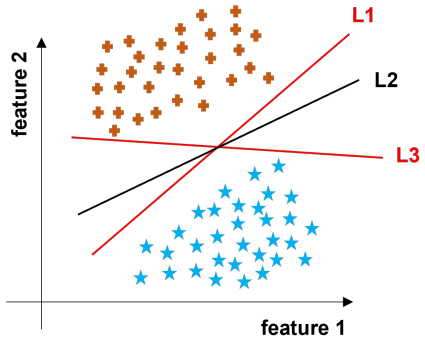

Fig. 1.4 SVM for binary classification finds the decision boundary with maximum margin between two classes (L2).

Several FPGA accelerators have been proposed for SVM [31–42]. Recent SVM accelerators in FPGA are reviewed in [43]. The main limitation of almost all the previous works is the lack of scalability, which would allow the same design to be used for larger data dimensions. These conventional accelerators assumed that all the SVM coefficients and input data are stored in an on-chip memory, which prevents these designs to be used for large data dimensions. Instead, we proposed a flexible and scalable SVM accelerator in FPGA that can be used for different data dimensions when we have a limited amount of on-chip memory.

1.3.4 Neural Networks

Artificial Neural Networks (ANNs) are progressively used in different application areas, including MWI. ANNs are inspired from the human brain structure and are organized in layers of artificial neurons: the input layer, several hidden layers, and

the output layer. Each layer receives the data from the previous layer and processes them before feeding the next layer. The operations performed by each neuron are relatively simple (multiplication, accumulation, and the application of a non-linear function), but the number of neurons in a layer, their connections, and the number of layers complicate in a substantial way the hardware implementation of ANNs.

Multi-Layer Perceptrons (MLPs) and Convolutional Neural Networks (CNNs) are two common structures of ANNs. In MLP, the layers are fully connected to each other. In CNNs, there are several filters in each layer that are convolved with the input data from the previous layers. Fig. 1.5 shows these two network architectures. One of the main challenges in designing an efficient FPGA accelerator for ANNs is

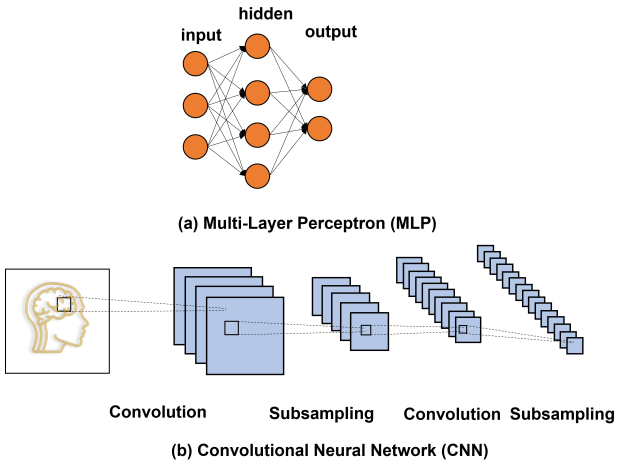

Fig. 1.5 Network structure for a) MLP and b) CNN.

how to optimize at the same time the training hyper-parameters and the hardware configuration to achieve the best performance in terms of training accuracy, speed, hardware resource usage and power consumption. Although there have been several recent works tackling this problem [44–49], there is still a great potential to further enhance and optimize the previous works. One of the areas that has not been fully explored in the context of co-optimization of network training parameters and hardware configurations is Bayesian Optimization which we consider in this work.

1.4 HLS-based Hardware Acceleration in FPGA

The traditional method to design a hardware accelerator in FPGA is manual RTL design using Hardware Description Languages (HDLs) like VHDL or Verilog. Although being still the dominant approach, RTL manual design using HDLs requires a significant amount design and development time. This precludes the exploration of

multiple alternatives in search of the optimum design, which is especially challenging for very large designs due to a large hardware design space. A recent approach that simplifies the design space exploration and can achieve a similar performance compared to the HDL languages with the advantage of faster development phase is High Level Synthesis (HLS) approach. HLS receives a software code written in C or C++ and transforms it into the corresponding RTL design by properly scheduling the tasks, binding the operations, and allocating the resources [50]. Since the RTL is automatically generated and the generation is fast, exploring multiple design options becomes viable even for very large designs.

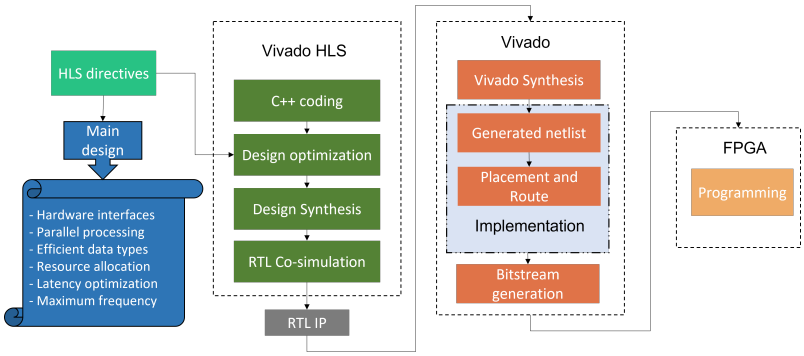

Fig. 1.6 Hardware design flow in Vivado [51].

Figure 1.6 shows the general HLS design flow in FPGAs for Xilinx devices. The input to the *Vivado HLS* tool is a software code in C or C++. In the next step, based on the design constraints such as timing and available resources, different hardware optimizations can be applied to the code to achieve the desired performance. This can be obtained by using so-called HLS directives [52]. The next step is the synthesis in which the netlist corresponding to the optimized code is generated. The RTL co-simulation step checks if the generated RTL is functionally identical to the software code. Finally, after generating an RTL Intellectual Property (IP) block, the implementation steps follow in the Vivado tool in the same way as if the RTL was manually generated [51].

1.5 FPGA Acceleration of 3D FDTD for Multi-Antenna MWI

Two main computational problems often found in MWI are the so-called *Forward Scattering* and *Inverse Scattering*. In the former, the dielectric profile is already known and the goal is to compute the scattered fields using this information. Instead, in the latter, the electromagnetic properties of the body tissues are unknown and the goal is to retrieve them from the known scattered fields. In iterative non-linear MWI

algorithms, usually the Forward and Inverse scattering are combined in such a way that the Inverse solver calls the Forward solver to iteratively update the solution.

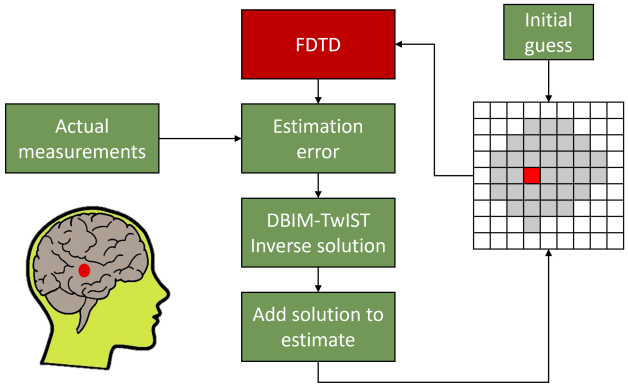

Fig. 1.7 General diagram of a non-linear image reconstruction iterative algorithm in MWI, with the compute-intensive FDTD step.

One of these cases is shown in Fig. 1.7. It is a non-linear iterative algorithm in which a forward electromagnetic solver, in this work an FDTD solver, is executed iteratively to update the initial guess of the dielectric profile based on the difference between the FDTD output and the actual microwave measurements received from a set of antennas. The error between the actual and simulated electromagnetic fields is minimized in multiple iterations by using a piece-wise linear inverse solution (in this work we used DBIM-TwIST algorithm developed in [53] for the inverse solution). As highlighted in Fig. 1.7 in red, FDTD is the bottleneck for the execution time of this algorithm, thus it is highly beneficial to have an efficient hardware accelerator.

Initially, a GPU implementation for the critical FDTD part was used. Although a speed-up can be achieved with GPU compared to the software design in CPU, the overall processing time is still high. This is why we focused on FPGA acceleration to further improve the performance.

The discretized domain in which FDTD solves the Maxwell equations, especially for 3D cases, contains so many unknowns that it is impossible to store them in the limited on-chip memory of the hardware accelerator, be it a GPU or an FPGA. Therefore, moving data between an external memory (i.e., outside the FPGA) and the local memory (inside) is necessary and can become a performance limiter. The FPGA accelerator proposed in this work uses a *spatial* blocking approach to reduce the data transfer time between the external memory (i.e., outside the FPGA) and the local memory (inside the FPGA). In this approach, a new plane from the 3D simulation space is read from the memory while the previous plane is being processed. The computations of FDTD for each plane consists of the famous Yee's update equations [54], in which we include also the effect of the boundaries.

In particular, the boundary regions are modeled with absorbing boundary conditions (CPML). Therefore, in addition to the variables representing the electric and

magnetic fields in the three directions ($E_x, E_y, E_z, H_x, H_y, H_z$), additional variables in these equations are used in the boundary regions. In addition, we model dispersive materials, which requires an extra variable termed J_P ((J_{Px}, J_{Py}, J_{Pz}) in FDTD equations that complicates the hardware design and has not been considered in previous works. The main features of our FDTD accelerator are:

- the impact of dispersive materials and complex CPML boundary conditions on FDTD equations are considered;
- to increase the flexibility we designed two hardware architectures, Large and Small, from which the designer can choose based on the specific characteristics of their design;
- we used High Level Synthesis (HLS) and several hardware optimization methods to implement the accelerator in an efficient way;
- for multiple antennas, we could design single- and multi-FPGA platforms that can process FDTD equations in parallel for different antennas;
- the high level design of our FDTD algorithm can be implemented in GPU as well with a performance comparable with a commercial GPU implementation.

1.5.1 FPGA Design of an FDTD Compute Unit

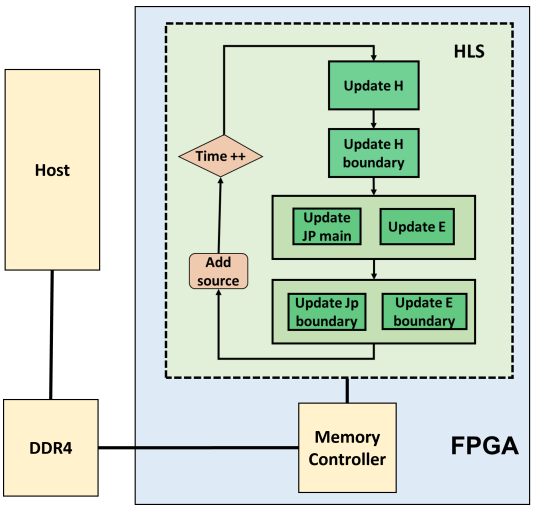

Fig. 1.8 FDTD accelerator in FPGA, JP is the polarization current.

Fig. 1.8 shows a block diagram of the HLS code for a single Compute Unit (CU) related to the FDTD design. Multiple CUs can be instantiated to work concurrently on different antennas. Each block in a CU is a function corresponding to an update equation (electric or magnetic update equations). Note that the J_P equations related

to the polarization currents in dispersive materials are merged with the Update E and E boundary equations in Fig. 1.8: this avoids multiple accesses to fetch the field values from the external memory. Table 1.1 shows the HLS-based hardware optimization methods used in our design. They will be explained in more detail in Section 1.5.3.

Table 1.1 HLS hardware optimization strategies for a FDTD CU.

Method	**functions**
Blocking	Update E and Update H
Merge JP	Update E and E boundary
Loop merge	Update E boundary and H boundary
Storage for Boundaries	Update H boundary and/or Update E boundary
Storage for constant coefficients	top-level function
Loop pipeline	all
function inline	all

1.5.2 Two Architectures: Large and Small

The difference between *Large* and *Small* designs is the number of AXI I/O ports toward the external memory and the amount of local resources in on-chip memory.

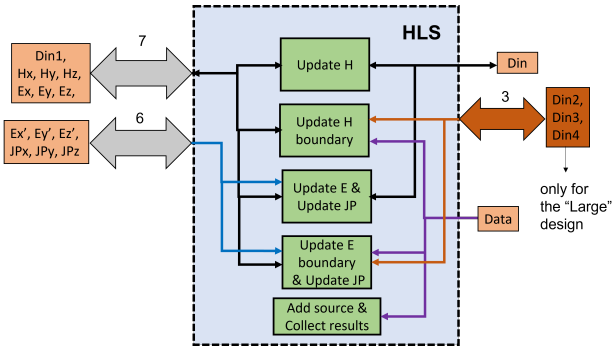

Fig. 1.9 Details of the interfaces of the CU for Small and Large designs. Numbers near the arrows indicate the number of variables read or written from and to the external memory ports. Din-n are multi-purpose ports, while specific memory ports are dedicated to E and H variables.

The Large design uses more AXI ports and compute resources resulting in higher speed for a single CU whereas the Small design consumes less resources and AXI ports. Nonetheless, since more Small CUs can fit in one FPGA, the Small design can lead to a higher throughput. Therefore, depending on the number of available FPGAs and the number of CUs to parallelize, the designer can select either Small

or Large design. Note that the larger number of AXI ports in the Large design (18 ports) is still compatible with the maximum memory bandwidth in the target FPGA, whereas in the Small design there are 15 AXI ports, as shown in Fig. 1.9.

1.5.3 HLS Optimizations

We briefly explain the HLS optimizations in Table 1.1 used in the proposed FDTD accelerator.

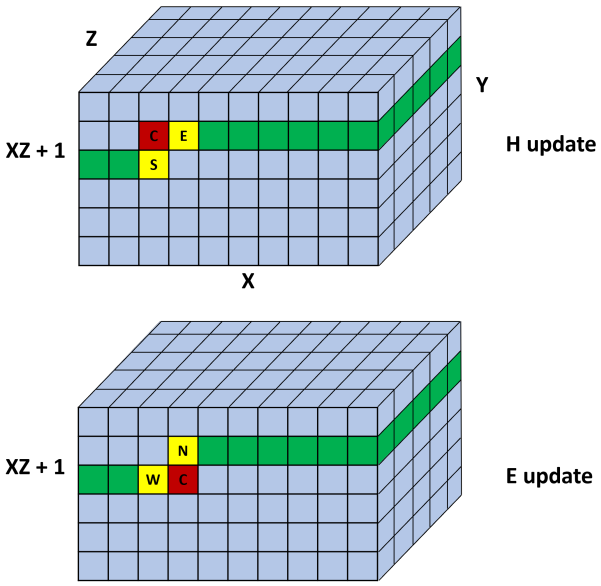

Fig. 1.10 Blocking method for 3D FDTD.

The main optimization is the spatial blocking approach used to reduce the memory access time in the main update equations. As shown in Fig. 1.10, processing of 2D planes in the simulation space can be pipelined by using a shift register in the local memory that can store one row (in 2D) or one plane (in 3D) of the simulation space plus an additional cell. After the shift register is full, the computation of the previous plane and the storage of the new plane in the shift register can occur in parallel. In this way, the shift register helps reduce the number of accesses to the external memory and the throughput can increase.

Loop merging was used for the loops in the boundary regions as well as the loops related to the polarization currents (J_P loops). In addition, we used local RAM memories available in Xilinx FPGAs, namely Block-RAMs (BRAMs) and Ultra-RAMs (URAMs), for the storage of Boundary fields and constant coefficients,

respectively. Finally we used “unrolling” and “pipelining” for the innermost loop in FDTD algorithm.

1.5.4 FDTD Result

The hardware platform used in our experiments is the Virtex UltraScale+ used in the Amazon EC2 F1 instance (vu9p-flgb2104-2-i). This FPGA consists of 3 Super Logic Regions (SLRs). There are 4 DDR4 memory interfaces. Each DDR interface can access a 16 GiB external memory.

The impact of the HLS optimization techniques described in Sec. 1.5.1 on the total latency is shown in Fig. 1.11. These optimizations are added incrementally starting from the original code without any HLS directives. The performance of the FDTD Compute Unit (CU) after each optimization is measured in terms of latency. The results show that each directive reduces the latency until a minimum latency of 8.6 s is obtained after applying all the directive in the Large design.

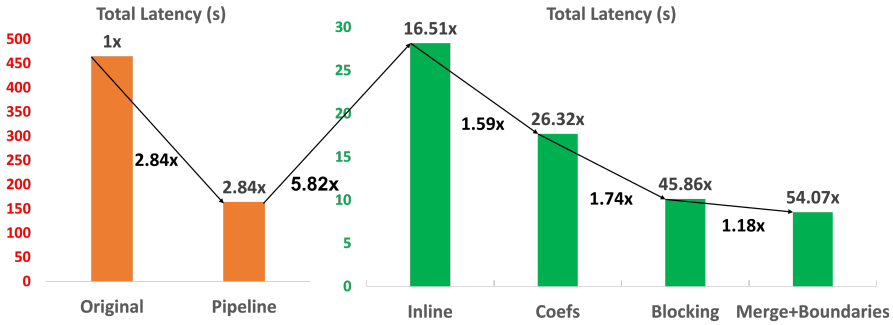

Fig. 1.11 Impact of different HLS optimization methods on the total latency. Numbers on top of the bars show the improvement compared to the original code, and numbers below the arrows show the improvement compared to the previous optimization method.

Table 1.2 presents a comparison between the performance of the FDTD accelerator on a CPU, three GPU designs, and FPGA. To have a fair comparison, the *time per antenna* is selected as the performance metric (total processing time divided by the number of antennas processed in parallel). The power consumption for FPGA in the table is the total consumed on-chip power that is calculated by the post-route report in the *Vivado Power Analysis* tool, while for the CPU and GPU, the maximum Thermal Design Power (TDP) is reported. The maximum TDP in our FPGA target is $128W$ which is still less than the TDP of CPU or GPU designs. By multiplying either the on-chip power or the maximum TDP to the Processing time per antenna, it is observed that the total energy consumption of our proposed FPGA design is the lowest.

Table 1.2 Performance comparison: CPU (Intel Xeon), three GPU designs (Tesla K20C and P40), and FPGA (UltraScale+).

Hardware	Ant. processed in parallel	Time per ant. (s)	Power (W)
Intel Xeon Gold 5120	1	24.97	105
GPU1 Tesla K20c (Matlab)	1	12.06	225
GPU2 Tesla P40 (Acceleware)	1	5.64	250
GPU3 Tesla K20c (This work)	1	4.88	225
UltraScale+ (This work, 15-ports)	3	3.38	16.24
UltraScale+ (This work, 18-ports)	2	4.3	14.5

A comparison between similar related works are presented in Table 1.3. It is observed that the previous works did not consider the impact of polarization currents which creates extra computations and complicates the data dependencies. It can be inferred that the performance in Mcells/s is not high in our work. However, due to the extra operations created by the polarization currents, the performance in GFLOP/s is superior to other implementations. In addition, our proposed accelerator operates in higher clock frequency than traditional works. Please refer to [55] for more details about the results of the proposed hardware accelerator for FDTD.

Table 1.3 Performance comparison between our single Small FPGA design and other FPGA implementations.

	Work [15]	**Work [11]**	**Work [14]**	**This work Small Design**
Boundary Conditions	Dirichlet (zero)	CPML (limited)	CPML (Full)	CPML (Full)
Polarization	No	No	No	Yes
Language	MaxCompiler (HLS)	MaxCompiler (HLS)	Verilog	Vivado HLS (C++)
Mcells/s	325	100	336	101
GFLOP/s	**11.7**	**5.7**	**29.2**	**34.2**
Freq (MHz)	**100**	**100**	**100**	**167**

1.6 High Level Design of PCA accelerator in FPGA

PCA can be broken down into a sequence of computations performed over the incoming data. These are *Mean*, *Covariance*, *EVD* (or *SVD*), and *Projection*. Each of these can be designed as an HLS function, and by using the Dataflow optimization approach all the functions can be executed concurrently in a pipelined manner. Array partitioning, loop unrolling, loop pipelining, and function inlining are among other HLS optimizations that are used in the hardware design to achieve a high performance in our FPGA accelerator.

The hardware design of our PCA accelerator is illustrated in Figure 1.12, which can be interpreted as the hardware implementation of the software code in C++. The software code consists of a top function including two sub-functions: *Dispatcher* and *PCA core*. In the Dispatcher, the required data are read and sent to the PCA core through First-in First-out (FIFO) channels. In the PCA core, the main processing steps of the PCA algorithm are executed. The final results are written back to the external DDR memory.

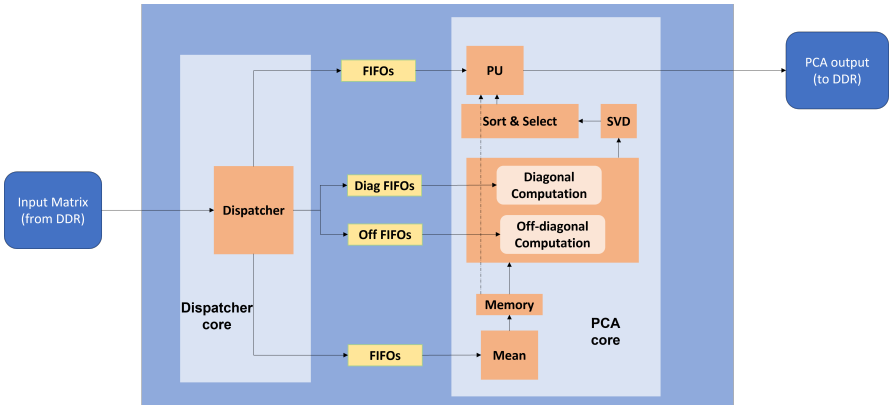

Fig. 1.12 Architecture of the proposed hardware accelerator for Principal Component Analysis (PCA) in Field-Programmable Gate Arrays (FPGA). The dispatcher distributed the input features to the PCA core units: mean, diagonal and off-diagonal computation, and projection unit. This last unit generates the PCA output and sends it to the external DDR memory.

The input to the PCA algorithm is a matrix with N samples and F features, and usually N is much greater than F. The first step in PCA algorithm is Mean computation which computes the mean values of all the samples. These mean values are stored in an internal on-chip memory to be used in Covariance (Cov) and Projection units. The first one computes the covariance matrix of the normalized input data and the SVD unit computes the Singular Values (SVs) of such covariance matrix: after sorting the SVs, the most important information from the data is preserved by keeping the highest values of the SVs as the Principal Components (PCs). In the Projection Unit, the normalized input data is multiplied with the PCs to obtain the output data with lower dimensions than the input data.

Since the covariance matrix has size $F \times F$, which is much smaller than the input matrix, computing the singular values is not as time-consuming as computing the covariance matrix itself. Therefore, hardware acceleration of Cov unit can significantly enhance the overall performance. For this reason, a new Block-streaming method is introduced for the Cov unit that can access the large input data in an efficient way to obtain the diagonal and off-diagonal elements of the covariance matrix. Optimization of the other units is done by applying the proper HLS *directives* such as loop unrolling, loop pipelining, and array partitioning.

1.6.1 Block-Streaming for Covariance Computation

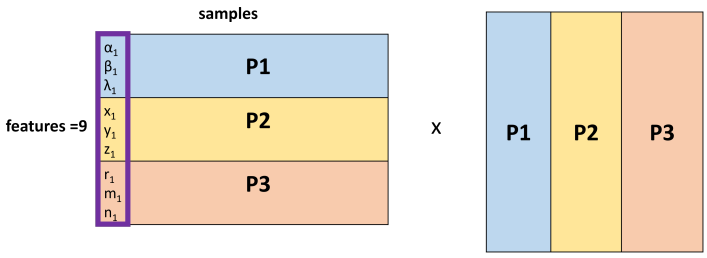

Fig. 1.13 Example of partitioning of input data into blocks. The total number of features is 9 and the block size is $B_{max} = 3$.

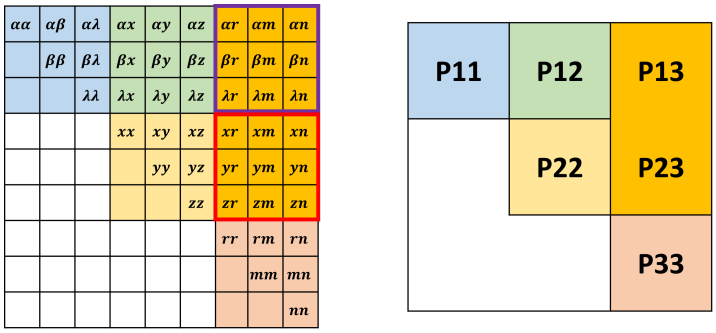

Fig. 1.14 Illustration of an example of covariance computation using the block-streaming method with 3 blocks ($B = 9$, $B_{max} = 3$).

The block-streaming method is useful for large data dimensions when there is limited amount of hardware resources available. The main idea is to partition the input data into multiple blocks, and then stream these blocks through the FIFO channels in a specific order to compute the diagonal and off-diagonal elements of the covariance matrix. This specific order is shown with an example in Figs. 1.13 and 1.14. The number of features in this example is set to 9 and the Block size is set to 3. Therefore, there are 3 data blocks shown in the figure as P1 to P3. In addition, there are two internal memories to store the streaming blocks of data; these memories are used in the Diagonal and Off-diagonal computations of the covariance matrix, so they are termed “Diag” and “Off-Diag” RAM respectively. The flowchart of the proposed blocking method is shown in Fig. 1.15.

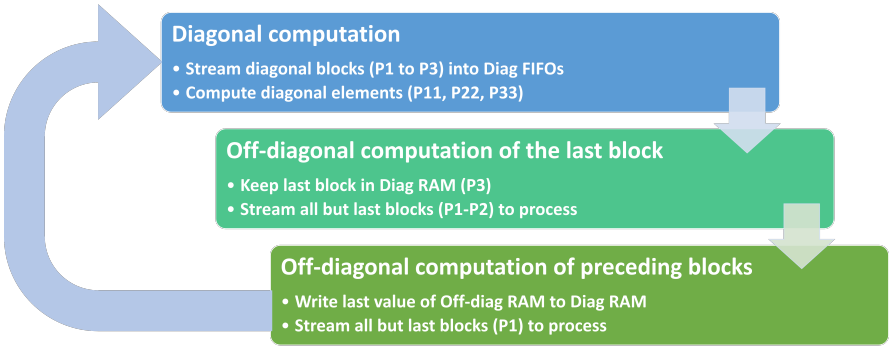

Fig. 1.15 Block-streaming algorithm in Covariance computation.

1.6.2 PCA results

In our experiments, the data dimensions are considered to be large enough to correspond to the Microwave Imaging scenarios. To evaluate our performance, we used the Virtex FPGA in the VC709U evaluation board as the hardware platform. The processing time (latency) for each compute unit of PCA is shown in Fig. 1.16. In this experiment, we fix the block size and number of features and change the number of samples. As shown in the figure, the latency is increased for larger number of samples. Note that the critical part of the design is Covariance computation due to its large input dimensions. However, by using the block-streaming strategy described in section 1.6.1, the Covariance computation can achieve its optimal performance.

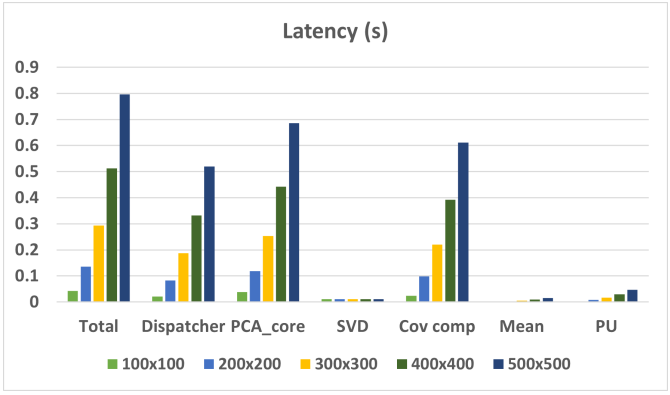

Fig. 1.16 Processing time for PCA compute units in FPGA.

In the next experiment, we quantify the impact on performance of varying the block size in the block-streaming approach. We fixed the number of features and samples, and increased the block size from 4 to 16. We can see from Figure 1.17

that increasing the block size can reduce the total latency in exchange of using more hardware resources. The critical compute units in terms of latency are the Covariance computation and Projection Unit. In addition, the Dispatcher's latency, which corresponds to the time to write the input data into the corresponding FIFO channels, is the limiting factor if we use the largest block size. This shows that we can speed up the PCA computation by using larger block size until the algorithm becomes memory-bound (the time to read data from external memory and write to the FIFOs exceeds the computation time). More details about the proposed hardware design for PCA algorithm can be found in [56] .

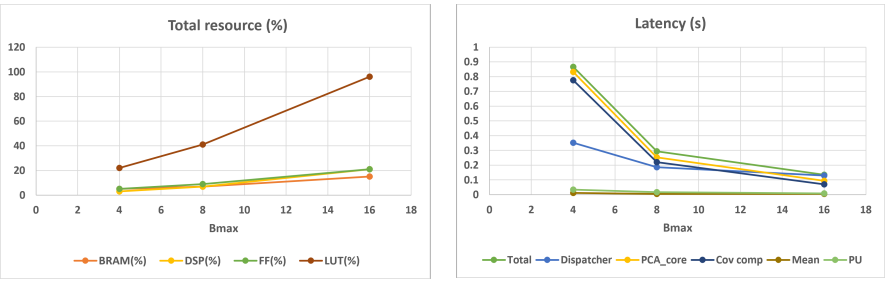

Fig. 1.17 Impact of block size (B_{max}) on the resource usage and latency for the Virtex7, features = 48, samples = 300 × 300, floating-point design.

1.7 Dataflow Hardware Architecture for SVM using HLS

Support Vector Machine (SVM) is a classification (or regression) technique that is useful in different application areas including microwave medical diagnosis. SVM inference can be briefly explained with the following equation:

$$\text{Decision} = \sum_{i=1}^{N_{SV}} \left(\alpha_i K(x, SV_i) + b_i\right), \tag{1.1}$$

in which N_{SV} is the number of Support Vectors, α_i and b_i are the SVM coefficients and bias values obtained after training, and $K(x, SV_i)$ is the Kernel function applied to the input data (x) and each Support Vector (SV_i).

Due to the heavy computations in SVM algorithm, hardware acceleration of SVM is of great importance in real-time embedded systems. For this purpose, several hardware accelerators have been recently proposed based on FPGAs. In [43], recent FPGA accelerators for SVM are reviewed. As opposed to previous SVM accelerators, we propose a scalable hardware architecture that uses the Dataflow HLS method to obtain concurrency between different computational parts of SVM algorithm, hence improving the throughput. In addition, thanks to the dataflow design, there is no need

to store all the SVM coefficients in the internal memory, hence reducing the on-chip memory requirements.

Fig. 1.18 shows the proposed hardware design consisting of three main building blocks: *Read*, *Kernel*, and *Decision*. The main idea is to partition our input features into multiple chunks of data, and while the current data chunk is being processed, we can read the next chunk by the Read function. In this way we can pipeline the execution of SVM computations and reading the inputs from the external memory. By using the Dataflow and Stream directives, this concurrent execution can be achieved in such a way that the *Read* function reads the data chunks and sends them through FIFO channels to the *Kernel* function. After Kernel computation, the *Decision* function receives the *Kernel* output and computes the final classification output (prediction or vote).

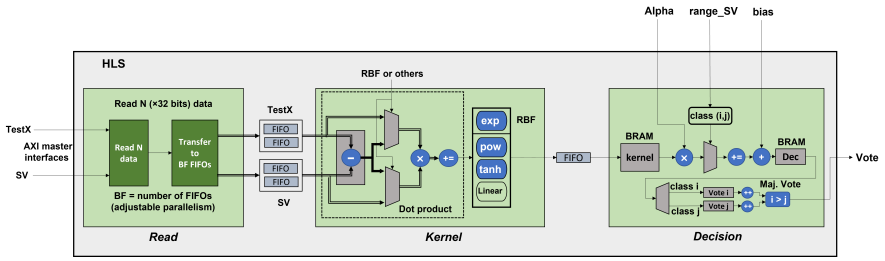

Fig. 1.18 Proposed SVM accelerator in HLS. The Support Vectors (SV) and the input values (TestX) are read and distributed to the middle block in order to compute the kernel function. The result is sent to the last block for the prediction (Vote), which needs to read from the memory three data: Alpha and bias are α_i and b_i in (1.1), while Range_SV is the range of the SV. The three blocks operate in pipeline fashion thanks to the FIFO queues.

1.7.1 Read SVM Inputs

In the *Read* function, we read N data features and send them through BF FIFO channels to the *Kernel* computation. These parameters (N and BF) can be adjusted based on the available resources and the desired performance. Ideally, the number of data to read (N) must be equal to the FIFO channels (BF) to have a balanced pipeline. However, as shown in Fig. 1.19, when the Initiation Interval (II[1]) of the *Kernel* function is higher than the *Read* function, increasing number of FIFOs (BF) to more than N can help reducing the overall latency. This is due to the fact that although we are slowing down the *Read* function, this degradation of the performance in the *Read* function is compensated by the higher parallelism (higher number of FIFOs) that is achieved by the *Kernel* function.

[1] The number of clock cycles between the start of subsequent iterations of a loop.

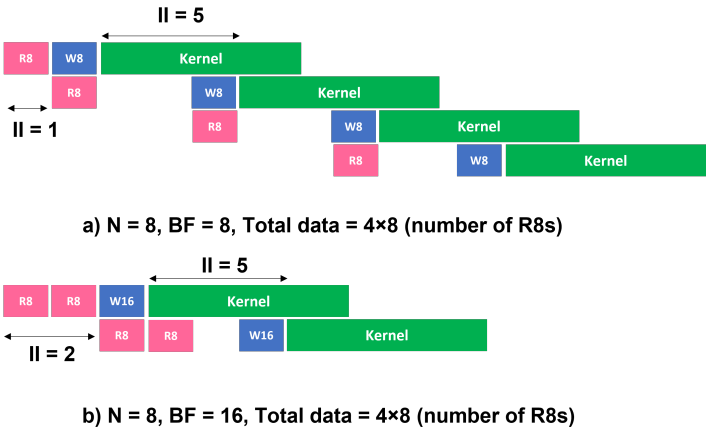

Fig. 1.19 Timing diagram showing the PCA execution. R8 stands for a read operation of 8 data, while W8 and W16 stand for a write operation (to the FIFOs) of a block of 8 or 16 data, respectively. The diagram shows the impact of the number of FIFO channels (BF) with a total of $4 \times 8 = 32$ data: (a) $BF = N$, II=5 (b) $BF = 2N$, II=2, hence the overall latency is reduced.

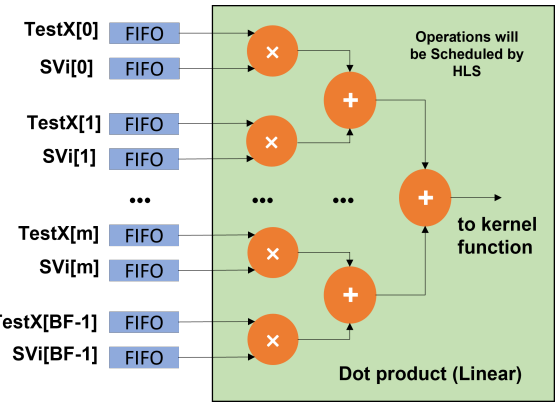

Fig. 1.20 Manual unrolling for kernel computation. Inputs (TestX) and support vectors (SV) are sent in parallel (depending on the level of parallelism BF) to the unit that computes their dot product.

1.7.2 Kernel Computation

The middle part of Fig. 1.18 shows the architecture of the hardware design for the *Kernel* computation. It consists of a dot product or square function between the input features and Support Vectors or their differences, respectively. Note that in HLS we can define the desired functionality by using proper C++ *macros*. Therefore, only the required kernel is implemented in hardware which is an advantage of HLS over manual RTL design.

To achieve a high performance for the Kernel computation, we increase the parallelism by using an adjustable parameter to specify the number of FIFOs (BF). As shown in Fig. 1.20, the computation of the dot product in the Kernel function can be partially unrolled with a factor of BF to match the parallelism of kernel computation to the number of FIFOs. To obtain this hardware architecture, we used manual unrolling with a factor of BF instead of the HLS *array partitioning* directive. This was necessary because accumulating the results of the kernel computation on a scalar variable (i.e., after the + = symbol in Fig. 1.18) resulted in data dependencies that the HLS tool was not able to solve in an efficient way, leading to keeping the accumulation not parallelized. Therefore, instead of a scalar variable, an array of size BF must be defined to hold the output of each multiplication, and this array must be fully partitioned to have concurrent access to all its elements. These elements are efficiently added together outside the loop to obtain the final scalar result.

1.7.3 Decision Function

The hardware architecture corresponding to the Decision function is shown in the right part of Fig. 1.18. The output of Kernel computation is streamed through a FIFO channel and is stored in an internal memory inside the Decision function. The Decision value is calculated for each pair of classes based on (1.1). Note that each class might have a different number of Support Vectors. The last step is to use the Majority Vote to obtain the final prediction.

There are two main loops in the Decision function corresponding to the computation of the Decision for one pair of classes. Similar to the Kernel computation, we manually unrolled these loops with a factor of DF, which is another adjustable parameter that can change the level of parallelism in our design.

1.7.4 SVM Results

To show the scalability of our hardware accelerator, we evaluated our design on a low-cost Zynq FPGA which has a low amount of hardware resources. We show that using two different data sets we can obtain optimal performances by adjusting the hardware configurations (N, BF, DF). The first data set is the MNIST and the second is a medical microwave data set of S-parameters provided by the biomedical microwave research team in Politecnico di Torino [57].

Table 1.4 shows the impact of HLS-based hardware parameters on the resource usage and latency for the MNIST data set. After training SVM with this data set, 16036 SVs with size 784 are obtained. The first three experiments use floating-point data precision while the last two are in fixed-point precision. As we can see, the resource usage increases by increasing the number of FIFOs (BF) while the latency is reduced. It should be noted that in the floating-point design, the main limitation is

Kernel computation and the design is compute-bound. However, in the fixed-point design the time-consuming part is the Read function, meaning that the design is memory-bound. That is why we could increase the number of data to read (N) in the last two experiments. Note the low accuracy loss in the fixed-point design compared to the floating-piint design. For the MNIST data set, the DF has negligible impact on the performance due to the low value of latency of the Decision function.

Table 1.4 MNIST dataset: performance and resource usage. BRAM is a block of Xilinx RAM; DSP48 is the Xilinx blocks for 48-bit multiplications and additions; FF is a flip-flop; LUT is a Look-Up Table for storing the truth table of logic functions.

Experiment	**1**	**2**	**3**	**fix1**	**fix2**
N	4	4	4	8	8
BF	4	8	16	8	16
BRAM (%)	15	16	19	12	15
DSP48 (%)	20	23	28	7	11
FF (%)	8	10	13	11	16
LUT (%)	24	28	35	40	62
Latency (ms)	166.93	91.57	58.69	18.12	15.72
Accuracy (%)	98.56 (float)			98.55 (fixed)	

The medical microwave data set contains 462 features originally, but they are reduced to 110 feature after applying PCA algorithm for feature extraction. The data set contains 9 classes that represent the type and positions of the brain stroke: stroke types (*ischemic, hemorrhagic, no stroke*) and locations (*top-right, top-left, bottom-right, bottom-left*). After the training step, 2009 Support Vectors are obtained.

Table 1.5 and Table 1.6 represent the results of the proposed hardware accelerator for the floating-point and fixed-point data types, respectively. The optimal latency of the design can be obtained by setting the HLS parameters to $N = 2, BF = 8, DF = 2$ which results in the optimal latency to be 3 ms. Note that with these configurations, the Kernel computation is still the dominant part of the design. Note that although increasing N and DF can reduce the latency of *Read* and *Decision* functions, the total latency will not change because of the higher latency of the *Kernel* function which is the dominant part.

By using fixed-point data precision, the optimal latency can be reduced to $0.55ms$ (a reduction of 5 times). Since in the fixed-point design the Decision latency is only limited by the number of memory ports that are used to store α_i parameter, Increasing DF will not reduce the total latency. Because of the random access pattern to the α_i memory, partial partitioning of this array is also not helpful. Therefore, changing DF does not have any impact on the total latency. In contrast, with floating-point data precision, due to a higher II_D (= 5), increasing DF can also reduce the total latency as well.

Table 1.5 Performance analysis of SVM accelerator for medical microwave data set using floating-point data precision (N_{SV} = 2009, $N_{features}$ = 110, $N_{samples}$ = 900). For the meaning of the acronyms see Tab. 1.4.

Experiment	float1	float2	float3	float4
N	2	2	8	8
BF	2	8	8	16
DF	2	2	8	8
BRAM (%)	4	3	3	3
DSP48 (%)	19	22	22	28
FF (%)	8	10	20	23
LUT (%)	22	28	48	55
Read Latency (ms)	1.1	1.1	0.3	0.3
Kernel Latency (ms)	6.5	3	3	3.1
Decision Latency (ms)	1.3	1.3	0.6	0.6
Total Latency (ms)	6.5	3	3	3.1

Table 1.6 Performance analysis of SVM accelerator for medical microwave data set using fixed-point data precision (N_{SV} = 2009, $N_{features}$ = 110, $N_{samples}$ = 900). For the meaning of the acronyms see Tab. 1.4.

Experiment	fix1	fix2	fix3	fix4
N	2	4	8	8
BF	2	4	8	16
BRAM (%)	3	3	3	3
DSP48 (%)	5	5	7	10
FF (%)	7	9	11	17
LUT (%)	23	29	40	62
Read Latency (ms)	1.1	0.56	0.28	0.28
Kernel Latency (ms)	1.36	0.84	0.58	0.44
Decision Latency (ms)	0.55	0.55	0.55	0.55
Total Latency (ms)	1.36	0.84	0.58	0.55

1.8 Hardware Design and Optimization of Neural Networks

Machine Learning (ML) models, including Deep Neural Networks (DNNs), are being progressively used in a wide range of applications, one of which is medical Microwave Imaging. Any ML model contains a set of training hyper-parameters that must be tuned to achieve the maximum accuracy. For example, the number of hidden layers and neurons, or the filter size in DNNs are among these parameters. When a hardware designer aims at the hardware implementation of a neural network, the optimal hardware performance must be obtained while keeping the highest training accuracy. However, the objectives in the hardware design such as low latency and low resource usage are in contrast with the objectives in the training step including the accuracy. Therefore, the designer must find a trade off between these objectives, and this is usually done by solving a multi-objective optimization problem.

In this work, we present a multi-objective framework, termed *joint optimization* to co-optimize the ML training parameters and hardware configurations in order

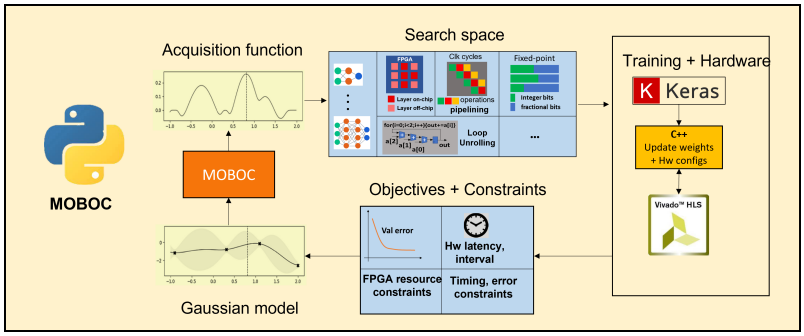

Fig. 1.21 Proposed methodology for training and hardware co-optimization in FPGA devices.

to achieve the optimal performance during both training and hardware design. The proposed framework is based on Multi-Objective Bayesian Optimization with Constraints (MOBOC) which is designed on top of HLS, and is evaluated on two network architectures: Multi-Layer Perceptron (MLP) and a Convolutional Neural Network (CNN) inspired from Lenet5. As opposed to conventional approaches that either use *Separate optimization*, or use Bayesian Optimization (BO) for the multi-objective optimization, our design supports multi-hardware configurations in which the search space consists of both network architectures and hardware configurations. In addition, our design supports truly multi-objective optimization without combining multiple objectives in a single objective function.

Fig. 1.21 shows the proposed joint optimization flow which can be divided into various parts as thoroughly described in the following.

1.8.1 Multi-Objective BO with Constraints (MOBOC)

Bayesian Optimization is a technique used to optimize functions which are difficult to evaluate. Each function in BO is assumed to be a black-box function that can be modeled with a Gaussian distribution. In MOBOC, after an initialization phase necessary to obtain a few samples from the function evaluations, a new function is created based on the Gaussian models that is termed *acquisition function*. The maximum value of the acquisition function determines the next sample from the search space that must be evaluated in the next iteration of the algorithm. By repeating this process, the uncertainties of the Gaussian models will be reduced until the convergence of the algorithm [58].

In our joint optimization, the MOBOC functions to evaluate and optimize and the constraints are hardware-related and ML-related: we used the prediction error on the validation set, throughput, and the hardware latency as the objectives, and we used the maximum available FPGA resources, clock period, and maximum error

threshold as the constraints. We used Keras for the training function evaluation and HLS C-simulation and Synthesis tool for the hardware function evaluation.

After extraction of the objective functions, the Gaussian models can be updated. In this work, we used Predictive Entropy Search for Multi-objective Optimization with Constraints (PESMOC) [59] for the acquisition function which updates the expected improvement of the estimated Pareto set. The maximum value of the acquisition function determines the next sample from the search space. A maximum number of iterations or a convergence criteria can be set to stop the simulation.

1.8.2 Search Space

We consider a large search space consisting of the network training parameters and HLS-based hardware configurations. Two network architectures are considered in this work which are MLP and a Convolutional Neural Network (CNN) inspired from Lenet-5. The search space for the MLP contains training hyper-parameters such as regularization parameter and learning rate as well as configurations for network architecture including number of hidden layers and neurons. For the CNN based on Lenet-5, the training parameters include the number of neurons, filters, and kernel size. In addition, the hardware configurations in FPGA are considered in the search space.

To design the neural network accelerator in FPGA, we used a high level description of the network in HLS. The HLS-based configurations include the choice between using off-chip or on-chip memory (the memory can be used to store the bias and weight values), loop pipelining and unrolling with adjustable unroll factor, Dataflow pragma[2], precision of the fixed-point data type, and the clock frequency. The optimal values of hardware configurations together with the training parameters can be obtained by the proposed framework based on MOBOC.

1.8.3 Evaluation on Neural Networks

We evaluated our framework on the two network architectures by using the MNIST data set as an example[3]. The target hardware is a Zynq7000 FPGA (XC7Z020-CSG484). For the implementation of MOBOC, the latest *Spearmint* package is used [59].

---

[2] A pragma is a HLS directive; a Dataflow pragma instructs the synthesizer to have multiple units work concurrently—i.e., in pipeline fashion—by means of FIFO buffers inserted between those units. Without the Dataflow pragma, the units operate sequentially [52].

[3] The MNIST database (Modified National Institute of Standards and Technology database) is a large database of handwritten digits that is commonly used for training various image processing systems [60]. Microwave imaging data sets can also be used for the evaluation.

1.8.3.1 Multi-Layer Perceptron

The training hyper-parameters and the ranges of the HLS configurations are shown in Tab.1.7.

Table 1.7 Ranges of parameters for the joint training/hardware optimization method.

Inputs	Clk (ns)	Hidden Layers	Neurons	Precision #total	Precision #Integer	Reuse factor	Array Partition	Learning rate	Regul-arization rate
Ranges	4 - 7	1 - 3	32 - 256 step = 32	12 - 16	4 - 6	1 - 4	2^x $x = [1-8]$	$1 \times 10^{(-x)}$ $x = [2-7]$	$1 \times 10^{(-x)}$ $x = [2-7]$

Figure 1.22 shows a comparison between the traditional separate method and the proposed Joint optimization. The prediction time in the Separate method is relatively higher than the proposed joint method. In addition, the Pareto-optimal points in the Joint optimization approach outperforms the other method. Please refer to [61] for more details about the proposed co-optimization framework for MLP network.

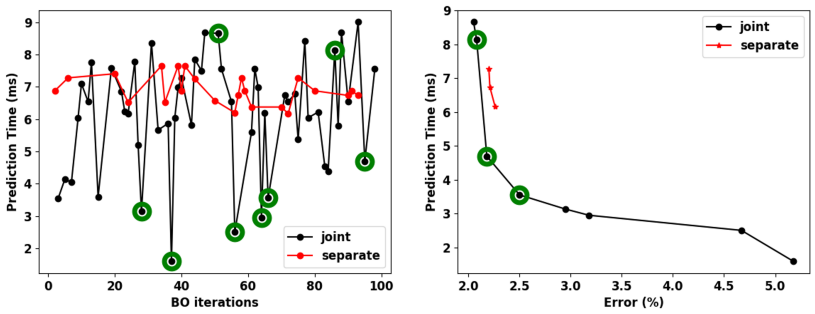

Fig. 1.22 Comparison of (a) prediction time and (b) Pareto fronts.

1.8.3.2 Convolutional Neural Network

The search space for the CNN accelerator inspired from Lenet-5 [60] is described in Tab. 1.8, in which *On Chip* refers to the option of selecting internal on-chip memory to store the weights and biases, so there are 5 options available for 5 layers. Total and Integer bits in Tab. 1.8 specify the fixed-point data precision. It should be noted that the search space is quite large and it would take years for an exhaustive search to achieve the optimum configurations (with total configurations around 10^{16}).

The Pareto-optimal points in terms of prediction error (after hardware implementation) and execution latency are reported for three different methods (Separate, Random Search, and proposed Joint methods) in Fig. 1.23. For all the methods, the

Table 1.8 Search space featuring network architecture and hardware configurations (UF: Unroll Factor).

Configs	Range	Configs	Range
#Filters Conv1	[2,20]	Dataflow	active/inactive
		On Chip	active/inactive
#Filters Conv2	[2,20]	Clk (ns)	[8,20]
		Total bits	[10,20]
Kernel size	2*([1,4]) + 1	Integer bits	[1,9]
#Neurons Dense1	[50,150]	Conv1 UF	2^([0,4])
#Neurons Dense2	[50,150]	Conv2 UF	2^([0,4])

search space is the same and the total number of iterations is set to 100. Note that in Random search, as opposed to other two methods, it is not possible to set a constraint for hardware error, so there is one point with error larger than the threshold (> 10%). With the exception of one single random point that is randomly detected by the Random search, all the other points in both the Random and Separate method are Pareto-dominated by the points found by the Joint method.

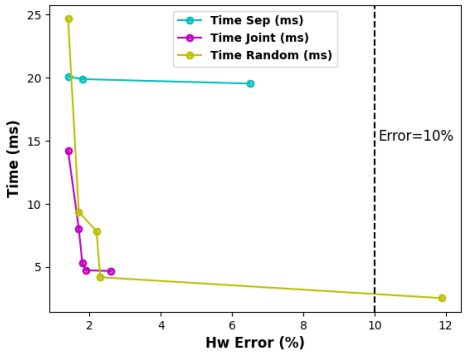

Fig. 1.23 Pareto-points found by the joint approach, random search, and conventional separate method in the space of prediction error (Hw error) and execution latency (Time). Total number of iterations is 100 for all methods.

In Fig. 1.24 all the suggested points in the three optimization methods are illustrated with three different colors. For the hardware error, a log scale is considered for better clarity. Due to the error threshold in MOBOC, the points are more concentrated below the threshold for the joint method which helps for better convergence of the algorithm.

1.8.3.3 Evaluation on microwave data set

For the anomalies detection and classification in MWI, MLP networks are useful [62]. The dataset that is used in this part is the same microwave data that we used in

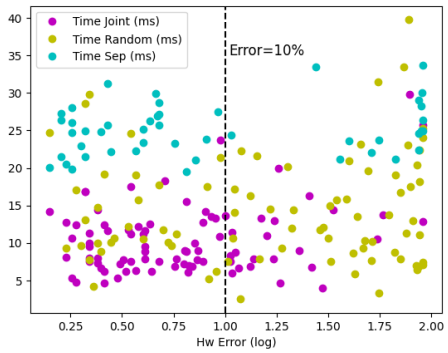

Fig. 1.24 Total points suggested by the joint, separate, and random search methods; note the concentration of the joint method on low errors ($< 10\%$).

Sec. 1.7. It contains 4500 samples of scattering matrix with 462 elements. There are 9 classes representing the presence, type, and location of the brain stroke. For feature extraction, PCA is used that results in the reduction of features to 110. In this part we focus on the training and hardware performance of an MLP network using the MWI dataset assuming that the optimal parameters of the network are obtained. We selected an MLP with 3 hidden layers and number of neurons per layer $220, 64, 64$, respectively. The target device is a Zynq SoC (ZedBoard), and we used fixed-point precision for the hardware implementation. The accuracy before and after hardware implementation, resource usage, and processing time are depicted in Table 1.9 and Figs. 1.25 and 1.26. Note the negligible accuracy loss in the hardware accelerator due to the reduced precision.

Table 1.9 Evaluation of MLP performance in microwave anomaly detection. Fixed-point precision is selected in hardware with total and integer widths of 16 and 10, respectively.

training accuracy (%)	test accuracy (%)	hardware accuracy (%)	Latency (ms)
99.6	98	97.5	1.12

1.9 Conclusions

In this work we presented efficient hardware design methodologies to accelerate the execution of compute-intensive algorithms that are recurrent in biomedical Microwave Imaging (MWI) techniques. Specifically, we proposed different hardware accelerators for 3D FDTD, PCA, SVM, as well as design optimization techniques

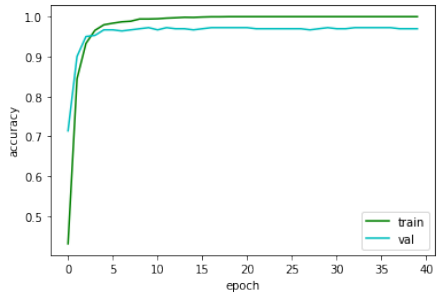

Fig. 1.25 Accuracy of MLP during training by MWI dataset.

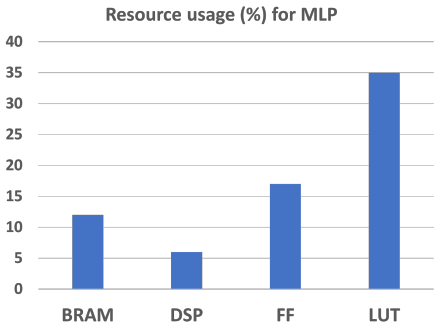

Fig. 1.26 Resource usage for MLP in Zynq FPGA.

for ML models, including Multi-Layer Perceptrons (MLPs) and Convolutional Neural Networks (CNNs). We used High Level Synthesis (HLS) as the most efficient approach to design these hardware accelerators and optimize their performance. The flexibility, completeness, efficiency, and high-level design techniques and hardware optimization methodologies are the distinguishing features of the proposed accelerators compared to previous related works. Specifically, these are the main contributions and achievements in our work:

- The accelerator for 3D FDTD considers the CPML boundary conditions for all directions, models the dispersive materials, and is designed as an internal step for a non-linear iterative MWI reconstruction algorithm. Two architectures, called Small and Large, were proposed based on the number of interfaces and the amount of consumed hardware resources. For the reference data set with the dimensions of $70 \times 70 \times 70$, each FPGA in the Small and Large design can be used for 3 and 2 antennas, respectively. The execution time per antenna for our FPGA accelerator is $3.38s$ for the Small design which is slightly better than the conventional GPU design with Acceleware library ($4.88s$). In addition, we presented a multi-FPGA design in which multiple antennas can be assigned to multiple FPGAs. The system level evaluation of our multi-FPGA design with 24 antennas and 8 UltraScale+

FPGAs shows a 13× speed-up compared to the single GPU design on Tesla P40 (Acceleware).

- The proposed PCA accelerator in FPGA consists of several computing elements, including the computations of *Mean*, *Covariance*, *SVD or EVD*, and *Projection of inputs*. Although separate acceleration techniques for each element have been already proposed, their integration in an efficient hardware accelerator has not been considered before. We proposed a flexible FPGA accelerator for PCA that can be used for different data dimensions, thanks to the HLS design methodology. In addition to the floating-point design, a fixed-point implementation was proposed to efficiently use the hardware resources in FPGA. For the computation of Covariance matrix, an efficient block streaming methodology was introduced to read blocks of input data from the external memory, store them in on-chip memory and process them inside the hardware accelerator, while reading the next block of data from external memory. Different data dimensions were analyzed in the evaluation, and compared to the GPU or multi-core CPU designs we could achieve either power efficiency or performance speed-up. In addition, we compared our HLS design an RTL implementation of PCA using VHDL and achieved a 2.3× speed-up as well as significant reduction of resource usage.
- FPGA accelerator for SVM uses a novel dataflow architecture in which the required input data are transferred to the accelerator while the SVM computations are performed. The proposed accelerator is scalable, meaning that it is not limited by the number of dimensions of Support Vectors and can be used for large data dimensions when there is limited on-chip memory. In addition, a fixed-point implementation was presented for the SVM accelerator which could improve the speed. The entire accelerator was designed using the HLS tool and we could apply different hardware optimization techniques. One of the main characteristics of our design is having *adjustable parallelism* which enables the designer to specify the amount of parallelism in hardware depending on the available resources. As opposed to most of the conventional SVM accelerators in FPGA, multi-class classification is also supported in our design. In addition, we explored the impact of different SVM kernels on the hardware performance. For the evaluation, we compared our HLS-based hardware accelerator with recent works which were designed using either HLS or the manual RTL approach. A 4.4× latency improvement could be achieved compared to the RTL design, and a minimum of 10× improvement in the latency was obtained compared to a similar HLS-based design.
- Finally, for the optimum design of Machine Learning models and specifically Neural Networks in FPGAs, a new methodology was proposed that could jointly optimize the training hyper-parameters and HLS-based hardware configurations based on multi-Objective Bayesian optimization. Although the proposed approach can be used for any ML model, we did a primary evaluation on two network architectures, which are Multi-Layer Perceptron (MLP) and Convolutional Neural Networks (CNNs). Compared to other solutions based on Bayesian Optimization, the search space in our method supports *multi-Hw* configurations and consists of a combination of network architectures and hardware configurations. In addition,

the available hardware resources in the FPGA can be set as *Constraints* in the optimization problem together with different contrasting *Objectives* such as hardware latency and accuracy loss. We considered a large search space and compared our joint optimization methodology with the separate optimization and the Random search approach. From the comparison, we could notice the improvement in the Pareto sets obtained by the proposed joint approach, with 1.7× and 1.4× improvement in the execution time for the minimum error compared to the random and separate methods, respectively, in a Lenet5-inspired CNN architecture, and 1.43× improvement in the prediction time without an increase in the prediction error compared to the separate design, in the MLP network architecture.

The above-mentioned hardware accelerators are highly beneficial for high-performance embedded computing systems. As stated previously, 3D FDTD is used in non-linear iterative medical microwave image reconstruction, and its acceleration helps in reducing the amount of time needed to reconstruct the final image. FPGA acceleration of SVD/EVD (included in PCA) is useful in linear microwave image reconstruction algorithms. Hardware acceleration of PCA, SVM, and Neural Networks are advantageous not only in MWI, but also in other ML applications which use these algorithms.

Acknowledgements This work was supported by the EMERALD project funded by the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No. 764479.

References

1. M. R. Casu, F. Colonna, M. Crepaldi, D. Demarchi, M. Graziano, and M. Zamboni, "Uwb microwave imaging for breast cancer detection: many-core, gpu, or fpga?," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 13, no. 3s, pp. 1–22, 2014.
2. I. Sarwar, G. Turvani, M. R. Casu, J. A. Tobon, F. Vipiana, R. Scapaticci, and L. Crocco, "Low-cost low-power acceleration of a microwave imaging algorithm for brain stroke monitoring," *Journal of Low Power Electronics and Applications*, vol. 8, no. 4, p. 43, 2018.
3. I. Saied, T. Arslan, R. Ullah, C. Liu, and F. Wang, "Hardware accelerator for wearable and portable radar-based microwave breast imaging systems," in *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1–5, 2021.
4. J. Cong, Z. Fang, M. Lo, H. Wang, J. Xu, and S. Zhang, "Understanding performance differences of fpgas and gpus," in *2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pp. 93–96, IEEE, 2018.
5. J. Cong, J. Lau, G. Liu, S. Neuendorffer, P. Pan, K. Vissers, and Z. Zhang, "Fpga hls today: Successes, challenges, and opportunities," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 15, aug 2022.
6. X. Wang, S. Liu, X. Li, and S. Zhong, "Gpu-accelerated finite-difference time-domain method for dielectric media based on cuda," *International Journal of RF and Microwave Computer-Aided Engineering*, vol. 26, no. 6, pp. 512–518, 2016.
7. Z. Bo, X. Zheng-hui, R. Wu, L. Wei-ming, and S. Xin-qing, "Accelerating fdtd algorithm using gpu computing," in *2011 IEEE International Conference on Microwave Technology & Computational Electromagnetics*, pp. 410–413, IEEE, 2011.

8. S. Liu, B. Zou, L. Zhang, and S. Ren, "Heterogeneous cpu+ gpu-accelerated fdtd for scattering problems with dynamic load balancing," *IEEE Transactions on Antennas and Propagation*, vol. 68, no. 9, pp. 6734–6742, 2020.
9. H. Zhang, Y. Lei, H. Ye, and Y. Gong, "Bistatic radar cross section prediction of 3-d target based on gpu-fdtd method," in *2018 12th International Symposium on Antennas, Propagation and EM Theory (ISAPE)*, pp. 1–4, IEEE, 2018.
10. T. Kenter, J. Förstner, and C. Plessl, "Flexible fpga design for fdtd using opencl," in *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*, pp. 1–7, IEEE, 2017.
11. H. Giefers, C. Plessl, and J. Förstner, "Accelerating finite difference time domain simulations with reconfigurable dataflow computers," *ACM SIGARCH Computer Architecture News*, vol. 41, no. 5, pp. 65–70, 2014.
12. Y. Takei, H. M. Waidyasooriya, M. Hariyama, and M. Kameyama, "Fpga-oriented design of an fdtd accelerator based on overlapped tiling," in *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA)*, p. 72, The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp), 2015.
13. H. M. Waidyasooriya, T. Endo, M. Hariyama, and Y. Ohtera, "Opencl-based fpga accelerator for 3d fdtd with periodic and absorbing boundary conditions," *International Journal of Reconfigurable Computing*, vol. 2017, 2017.
14. C. Kong and T. Su, "Parallel hardware architecture of the 3d fdtd algorithm with convolutional perfectly matched layer boundary condition," *Progress In Electromagnetics Research C*, vol. 105, pp. 161–174, 2020.
15. K. Okina, R. Soejima, K. Fukumoto, Y. Shibata, and K. Oguri, "Power performance profiling of 3-d stencil computation on an fpga accelerator for efficient pipeline optimization," *ACM SIGARCH Computer Architecture News*, vol. 43, no. 4, pp. 9–14, 2016.
16. D. G. Perera and K. F. Li, "Embedded hardware solution for principal component analysis," in *Proceedings of 2011 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing*, pp. 730–735, IEEE, 2011.
17. D. Fernandez, C. Gonzalez, D. Mozos, and S. Lopez, "Fpga implementation of the principal component analysis algorithm for dimensionality reduction of hyperspectral images," *Journal of Real-Time Image Processing*, vol. 16, no. 5, pp. 1395–1406, 2019.
18. A. Das, D. Nguyen, J. Zambreno, G. Memik, and A. Choudhary, "An fpga-based network intrusion detection architecture," *IEEE Transactions on Information Forensics and Security*, vol. 3, no. 1, pp. 118–132, 2008.
19. U. A. Korat and A. Alimohammad, "A reconfigurable hardware architecture for principal component analysis," *Circuits, Systems, and Signal Processing*, vol. 38, no. 5, pp. 2097–2113, 2019.
20. M. U. Torun, O. Yilmaz, and A. N. Akansu, "Fpga, gpu, and cpu implementations of jacobi algorithm for eigenanalysis," *Journal of Parallel and Distributed Computing*, vol. 96, pp. 172–180, 2016.
21. S. Kasap and S. Redif, "Novel field-programmable gate array architecture for computing the eigenvalue decomposition of para-hermitian polynomial matrices," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 22, no. 3, pp. 522–536, 2013.
22. S. Zhang, X. Tian, C. Xiong, J. Tian, and D. Ming, "Fast implementation for the singular value and eigenvalue decomposition based on fpga," *Chinese Journal of Electronics*, vol. 26, no. 1, pp. 132–136, 2017.
23. X. Wang and J. Zambreno, "An fpga implementation of the hestenes-jacobi algorithm for singular value decomposition," in *2014 IEEE International Parallel & Distributed Processing Symposium Workshops*, pp. 220–227, IEEE, 2014.
24. Y. Ma and D. Wang, "Accelerating svd computation on fpgas for dsp systems," in *2016 IEEE 13th International Conference on Signal Processing (ICSP)*, pp. 487–490, IEEE, 2016.
25. Y.-L. Chen, C.-Z. Zhan, T.-J. Jheng, and A.-Y. Wu, "Reconfigurable adaptive singular value decomposition engine design for high-throughput mimo-ofdm systems," *IEEE transactions on very large scale integration (VLSI) systems*, vol. 21, no. 4, pp. 747–760, 2012.

26. M. V. Athi, S. R. Zekavat, and A. A. Struthers, "Real-time signal processing of massive sensor arrays via a parallel fast converging svd algorithm: Latency, throughput, and resource analysis," *IEEE Sensors Journal*, vol. 16, no. 8, pp. 2519–2526, 2016.
27. A. A. S. Ali, A. Amira, F. Bensaali, and M. Benammar, "Hardware pca for gas identification systems using high level synthesis on the zynq soc," in *2013 IEEE 20th International Conference on Electronics, Circuits, and Systems (ICECS)*, pp. 707–710, IEEE, 2013.
28. M. Schellhorn and G. Notni, "Optimization of a principal component analysis implementation on field-programmable gate arrays (fpga) for analysis of spectral images," in *2018 Digital Image Computing: Techniques and Applications (DICTA)*, pp. 1–6, IEEE, 2018.
29. M. A. Mansoori and M. R. Casu, "Efficient fpga implementation of pca algorithm for large data using high level synthesis," in *2019 15th Conference on Ph. D Research in Microelectronics and Electronics (PRIME)*, pp. 65–68, IEEE, 2019.
30. M. A. Mansoori and M. R. Casu, "Hls-based flexible hardware accelerator for pca algorithm on a low-cost zynq soc," in *2019 IEEE Nordic Circuits and Systems Conference (NORCAS): NORCHIP and International Symposium of System-on-Chip (SoC)*, pp. 1–7, IEEE, 2019.
31. C. Kyrkou and T. Theocharides, "A parallel hardware architecture for real-time object detection with support vector machines," *IEEE Transactions on Computers*, vol. 61, no. 6, pp. 831–842, 2011.
32. C. Kyrkou, C.-S. Bouganis, T. Theocharides, and M. M. Polycarpou, "Embedded hardware-efficient real-time classification with cascade support vector machines," *IEEE transactions on neural networks and learning systems*, vol. 27, no. 1, pp. 99–112, 2015.
33. M. Qasaimeh, A. Sagahyroon, and T. Shanableh, "Fpga-based parallel hardware architecture for real-time image classification," *IEEE Transactions on Computational Imaging*, vol. 1, no. 1, pp. 56–70, 2015.
34. S. Saurav, R. Saini, and S. Singh, "Fpga based implementation of linear svm for facial expression classification," in *2018 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pp. 766–773, IEEE, 2018.
35. Y. Jiang, K. Virupakshappa, and E. Oruklu, "Fpga implementation of a support vector machine classifier for ultrasonic flaw detection," in *2017 IEEE 60th international midwest symposium on circuits and systems (MWSCAS)*, pp. 180–183, IEEE, 2017.
36. L. Han, Z. Yue, and X. Guo, "Image segmentation implementation based on fpga and svm," in *2017 3rd International Conference on Control, Automation and Robotics (ICCAR)*, pp. 405–409, IEEE, 2017.
37. M. Ruiz-Llata, G. Guarnizo, and M. Yébenes-Calvino, "Fpga implementation of a support vector machine for classification and regression," in *The 2010 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–5, 2010.
38. S. Afifi, H. GholamHosseini, and R. Sinha, "A low-cost fpga-based svm classifier for melanoma detection," in *2016 IEEE EMBS Conference on Biomedical Engineering and Sciences (IECBES)*, pp. 631–636, IEEE, 2016.
39. S. Afifi, H. GholamHosseini, and R. Sinha, "Svm classifier on chip for melanoma detection," in *2017 39th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, pp. 270–274, IEEE, 2017.
40. S. Afifi, H. GholamHosseini, and R. Sinha, "A system on chip for melanoma detection using fpga-based svm classifier," *Microprocessors and Microsystems*, vol. 65, pp. 57–68, 2019.
41. A. Baez, H. Fabelo, S. Ortega, G. Florimbi, E. Torti, A. Hernandez, F. Leporati, G. Danese, G. M Callico, and R. Sarmiento, "High-level synthesis of multiclass svm using code refactoring to classify brain cancer from hyperspectral images," *Electronics*, vol. 8, no. 12, p. 1494, 2019.
42. R. Campos and J. M. Cardoso, "On data parallelism code restructuring for hls targeting fpgas," in *2021 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pp. 144–151, IEEE, 2021.
43. S. Afifi, H. GholamHosseini, and R. Sinha, "Fpga implementations of svm classifiers: a review," *SN Computer Science*, vol. 1, no. 3, pp. 1–17, 2020.
44. A. Mehrabi, A. Manocha, B. C. Lee, and D. J. Sorin, "Bayesian optimization for efficient accelerator synthesis," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 18, no. 1, pp. 1–25, 2020.

45. Y. Xiong, R. Mehta, and V. Singh, "Resource constrained neural network architecture search: Will a submodularity assumption help?," in *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 1901–1910, 2019.
46. S.-C. Kao, G. Jeong, and T. Krishna, "Confuciux: Autonomous hardware resource assignment for dnn accelerators using reinforcement learning," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 622–636, 2020.
47. J. Ney, D. Loroch, V. Rybalkin, N. Weber, J. Kruger, and N. Wehn, "Half: Holistic auto machine learning for fpgas," in *2021 31st International Conference on Field-Programmable Logic and Applications (FPL)*, (Los Alamitos, CA, USA), pp. 363–368, IEEE Computer Society, sep 2021.
48. H. Lee, S. Lee, S. Chong, and S. J. Hwang, "Hardware-adaptive efficient latency prediction for NAS via meta-learning," in *Advances in Neural Information Processing Systems* (A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, eds.), 2021.
49. M. S. Iqbal, J. Su, L. Kotthoff, and P. Jamshidi, "Flexibo: Cost-aware multi-objective optimization of deep neural networks," *CoRR*, vol. abs/2001.06588, 2020.
50. Xilinx, *Introduction to FPGA Design with Vivado High-Level Synthesis*. Xilinx, 2019.
51. Xilinx, *Vivado Design Suite User Guide: Design Flows Overview*. Xilinx, 2022.
52. Xilinx, *Vivado Design Suite User Guide: High-Level Synthesis*. Xilinx, 2021.
53. Z. Miao and P. Kosmas, "Multiple-frequency dbim-twist algorithm for microwave breast imaging," *IEEE Transactions on Antennas and Propagation*, vol. 65, no. 5, pp. 2507–2516, 2017.
54. K. Yee, "Numerical solution of initial boundary value problems involving maxwell's equations in isotropic media," *IEEE Transactions on antennas and propagation*, vol. 14, no. 3, pp. 302–307, 1966.
55. M. A. Mansoori, P. Lu, and M. R. Casu, "Fpga acceleration of 3d fdtd for multi- antennas microwave imaging using hls," *IEEE Access*, vol. 9, pp. 122696–122711, 2021.
56. M. A. Mansoori and M. R. Casu, "High level design of a flexible pca hardware accelerator using a new block-streaming method," *Electronics*, vol. 9, no. 3, 2020.
57. V. Mariano, J. A. T. Vasquez, M. R. Casu, and F. Vipiana, "Model-based data generation for support vector machine stroke classification," in *2021 IEEE International Symposium on Antennas and Propagation and USNC-URSI Radio Science Meeting (APS/URSI)*, pp. 1685–1686, 2021.
58. S. Greenhill, S. Rana, S. Gupta, P. Vellanki, and S. Venkatesh, "Bayesian optimization for adaptive experimental design: A review," *IEEE Access*, vol. 8, pp. 13937–13948, 2020.
59. E. C. Garrido-Merchán and D. Hernández-Lobato, "Predictive entropy search for multi-objective bayesian optimization with constraints," *Neurocomputing*, vol. 361, pp. 50–68, 2019.
60. Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.
61. M. A. Mansoori and M. R. Casu, "Efficient training and hardware co-design of machine learning models," in *Applications in Electronics Pervading Industry, Environment and Society* (S. Saponara and A. De Gloria, eds.), (Cham), pp. 243–248, Springer International Publishing, 2022.
62. T. Reimer, J. Sacristan, and S. Pistorius, "Improving the diagnostic capability of microwave radar imaging systems using machine learning," in *2019 13th European Conference on Antennas and Propagation (EuCAP)*, pp. 1–5, IEEE, 2019.