BEST PRACTICES FOR BUILDING HARDWARE DESIGNS FOR LIVING COMPUTATIONAL SCIENCE APPLICATIONS

by

Robin Jacob Pottathuparambil

A dissertation submitted to the faculty of
The University of North Carolina at Charlotte
in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in
Electrical Engineering

Charlotte

2013

Approved by:

_____
Dr. Ronald R. Sass

_____
Dr. James M. Conrad

_____
Dr. Bharat S. Joshi

_____
Dr. Ryan Adams

_____
Dr. Taghi Mostafavi

ABSTRACT

ROBIN JACOB POTTATHUPARAMBIL. Best practices for building hardware designs for living computational science applications.
(Under the direction of DR. RONALD R. SASS)

Scientific computing or Computational science, is a field of study where engineers and scientists use computer simulations to solve equations that model the physical world. In some cases, these equations come from the first principles of physics. In the past, these simulations were run on a single processor machine. However, due to various technological reasons, the performance of these machines are not likely to improve at the same rate as in the past. In order to improve the performance per watt of these simulations, special-purpose hardware accelerators can be used. This work mainly focuses on using FPGA-based hardware accelerators. In order to run these simulations on an FPGA accelerator, the application code needs to be re-factored into software and hardware sections. These faster simulations have motivated scientists to capture more behavior of the physical world. As additional behavior is captured, the application code needs to be re-factored each time, and a significant effort is required to re-build the design. Unfortunately, these multiple cycles of re-design reduces the overall productivity of scientists and engineers.

This work proposes a set of hardware design guidelines for changing computational science codes or living computational science codes. These guidelines co-evolve the hardware with the software, reducing the overall effort of re-design and improving productivity. The design guidelines are evaluated for effectiveness, communicability, and broad applicability. Experimental results have shown that the overall re-design effort is reduced, and these guidelines are broadly applicable to a wide variety of scientific computing applications.

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to my advisor, Dr. Ronald R. Sass, his patience, effort, constant encouragement, constructive feedback, and for his dedicated support for my doctoral study and research.

I am also grateful to my dissertation committee members, Dr. James M. Conrad, Dr. Bharat S. Joshi, Dr. Ryan Adams, and Dr. Taghi Mostafavi, for their feedback and comments.

I also thank all the Reconfigurable Computing Systems (RCS) lab members who have directly and indirectly helped me in my research. I also thank all my lab members for the research discussions we had during my doctoral study.

Finally, I would like to thank my parents, P.K. Jacob and Sophy Jacob, and my brother, Justin Jacob, who have patiently supported me spiritually and financially to complete my doctoral study.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

FPGA	field programmable gate array
LFHD	literature followed hardware design
GFHD	guideline followed hardware design
HW	hardware design
P-V	photo voltaic
NN	neural networks
FDTD	finite-difference time domain
SpMV	sparse matrix vector multiplication
UPML	uniaxial perfectly matched layer
CFD	computational fluid dynamics
MD	molecular dynamics
HR	hessenberg reduction
GFLOPS	giga floating-point operations
DDR	double data rate
BLAS	basic linear algebra subprograms

CHAPTER 1: INTRODUCTION

Computational science, or scientific computing, is a field of study where scientists and engineers study the physical world by modeling it with computer simulations [15]. These simulations have led many to consider it a third branch of science[1], along with the experimental and theoretical branches. Scientists greatly depend on simulation experiments because frequently the equivalent experiments in the physical world are either not possible or are prohibitively expensive. For example, there is no way to recreate the "big bang" but computer simulations can provide insight. Likewise, many drugs can be synthesized in the lab. However, each requires the development of process which can take months of work. Computer simulations offer the ability to sort through numerous candidates, reducing the search space to the most encouraging ones.

1.1 Computer Simulations

Some examples of computer simulation of the physical world include molecular dynamics, computational fluid dynamics, and electrodynamics simulations. In some cases, such as electrodynamics simulations, the real world is modeled using a set of mathematical equations and are solved in a discrete time domain [17]. Sometimes the equations are simplified models of the physical world; in other cases they come from first principles of physics. This makes solving these mathematical equations analytically impractical because it is tedious and time-consuming. Computer simulations make the process simpler and faster by solving the equations numerically with computer programs [18].

[1] A Google search for "third branch of science" results in numerous blogs, articles [16], and web pages discussing the topic.

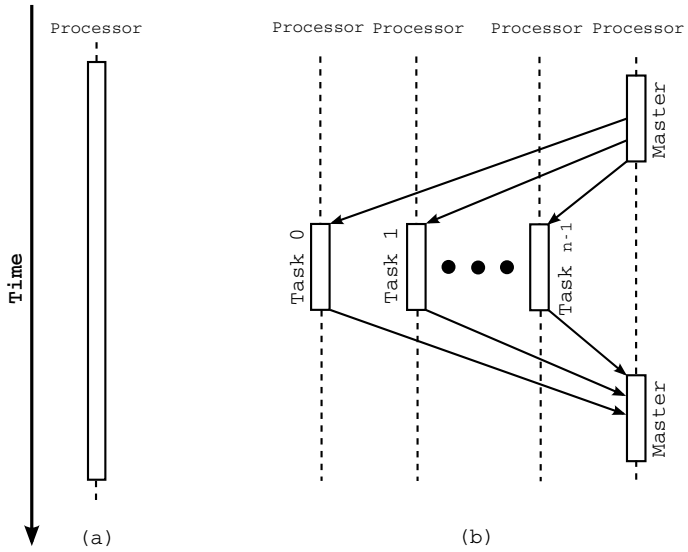

Figure 1.1: Sequential and parallel tasks

As the use of computers to study the behavior of the physical world has increased, scientists have become interested in capturing even more of the behavior of the physical world. This increases the complexity of the equations, resulting in longer simulation times. In order to reduce the time of these simulations, most computational scientists have relied upon computer engineers to produce ever-faster single processor machines. Unfortunately, for a number of technological reasons, single processor performance is not likely to continue to improve at the same rate as in the past [19]. This is forcing more computational scientists who want faster simulations to rely on parallel processing using parallel machines [20]. Instead of a single processor, parallel machines use multiple processors concurrently. This can be visualized in Figure 1.1 (time advances down the figure): (a) illustrates a single processor while (b) shows a collection of processors executing tasks concurrently. It also means more complicated parallel architectures because each processor has parallel cores.

In parallel processing, computational scientists divide their simulation experiments into smaller, independent tasks and execute their tasks concurrently (Figure 1.1b). These tasks are run on several processors simultaneously. In applications that scale

well, computational scientists can use more processors in parallel to improve the performance and reduce the computation time.

As more and more parallel processors are used to improve the performance of applications, the communication between tasks, components utilized, space, and power requirements are increasing [21]. The increase in the power requirements also increases the cooling requirements. To reduce these growing requirements due to computing using parallel processors, and to increase performance per watt, computational scientist are always looking for alternative ways.

1.2 Hardware Accelerators

One of the active research areas to improve the performance per watt of a computational science application is the use of special-purpose hardware accelerators [22, 23]. These hardware accelerators have shown to improve the performance per watt of ordinary applications [24, 25, 26]. For example, graphic processor units (GPUs) are used to accelerate the construction of images in a frame buffer. The improvement in performance per watt of ordinary applications using hardware accelerators have motivated researchers and computational scientists to explore using these accelerators for their applications [27, 28, 29, 30, 31]. Similarly, most supercomputers in (1 – 5) of Top 500 list are built using hardware accelerators [32, 33]. Thus, the use of hardware accelerators improves performance per watt and reduces the run time of the application. That is, given a power constraint, hardware accelerators improves the performance of the application.

Presently, hardware accelerators for scientific applications are built either using Application-Specific Integrated Circuits (ASICs) [29, 34], Graphic Processor Units (GPUs) [35, 36], Cell/B.E. [27, 30, 37], Intel's Many Integrated Core (MIC) [38, 39], or Field Programmable Gate Arrays (FPGAs) [40, 41, 11].The choice of ASIC, GPU, Cell, Intel's MIC, or FPGA-based hardware accelerator will depend on the application, the availability of hardware, and the developer's interests. However, as

a practical matter we only consider FPGA devices in this work. An FPGA is an integrated circuit that has programmable logic and can be configured by the end user to perform special-purpose operations. A complete description is provided in Chapter 2.

A single node FPGA accelerator consists of a single FPGA-based hardware accelerator connected to the main (host) system via a system bus. However, in most cases a tighter coupling is required with the host system. Presently, the Peripheral Component Interconnect (PCI) bus and CPU socket plug-in boards configurations are used to achieve the coupling with the host system [22].

Once the computational scientists can justify the use of an single node FPGA-based hardware accelerator and build hardware designs for their application, then they can improve the performance per watt of their application by employing multiple FPGA-based accelerator nodes running in parallel, as shown in Figure 1.2. These multiple FPGA nodes could be connected using an interconnect. Such nodes connected together to form a single entity is referred to as an FPGA cluster. An example of such an FPGA cluster is the Spirit reconfigurable computing cluster built at Reconfigurable Computing Systems (RCS) Lab, University of North Carolina at Charlotte [42]. This type of parallel computing with multiple FPGA nodes and host processors is called as hybrid or heterogeneous high-performance computing [43, 44]. For the purpose of evaluation, only a single node of the Spirit cluster will be used. The ideas and concepts presented in this dissertation can be extended to an FPGA cluster.

As the computational scientists build better systems with hardware accelerators to simulate the behavior of the physical world, the performance per watt improves, and thus the run time of the simulation experiment decreases. For example, case studies such as molecular dynamics [10], finite difference time domain (FDTD) [45], and acceleration of quantum Monte Carlo simulations [11] shows a performance per

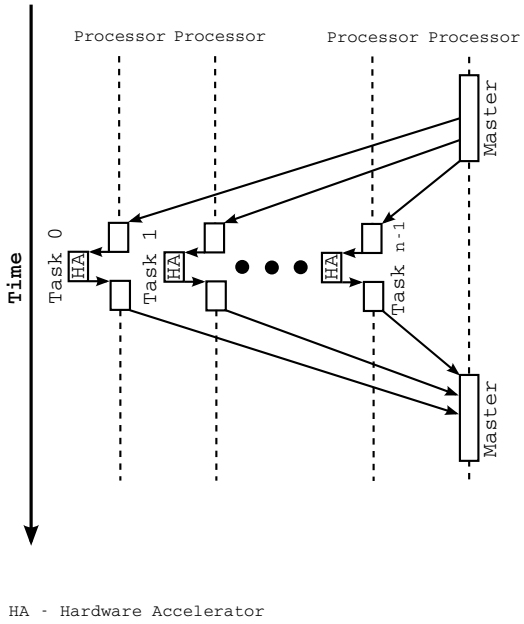

Figure 1.2: Computing using hardware accelerators

watt improvement using an FPGA-based accelerator. The performance per watt improvement of the applications aids computational scientists to increase the fidelity of their simulation or are interested in capturing more behavior of the physical world. These increases lead to an understanding of the changes needed in the next experiments. These additions in successive experiments evolves the application code. As the application code evolves, the hardware engineers refactor the application code into serial and accelerator code. This refactoring of the application code introduces a huge cost for the accelerator design.

1.3 Cost of Refactoring

We define a *living* computational science application as an application whose source code evolves over time, as computational scientists increase the fidelity of their simulation or reduce execution time or wish to explore new phenomena. However, once the changes or additions are incorporated in the application code, the next major step would be to run these application codes on an FPGA-based hardware accelerator. In order to run these new application codes, the hardware engineer

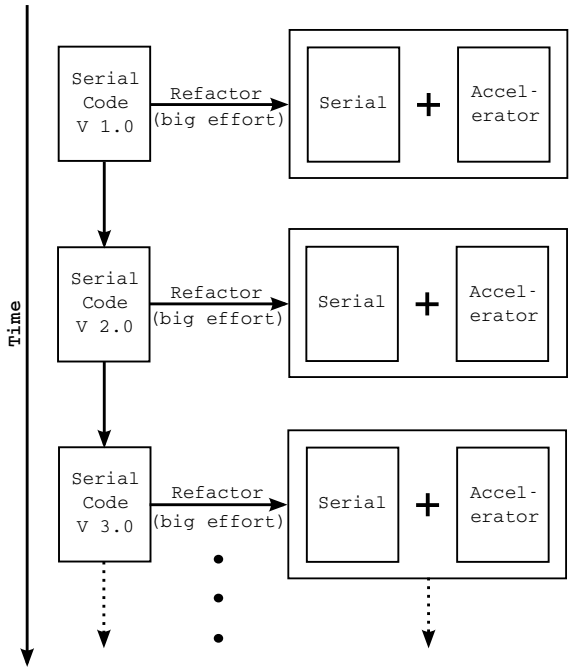

Figure 1.3: Human effort over time due code evolution

needs to refactor the application code into software and hardware sections. These hardware code sections are computed using an accelerator. As more phenomena are discovered, there are more revisions of the application code. Every revision of the application code introduces a new refactoring followed by a hardware redesign. Figure 1.3 illustrates this evolution of the source code over time. The critical point is that each time the code is revised (for example, going from version 1.0 to version 2.0), the serial application has to be refactored.

In a scenario where computational scientists are using FPGA-based hardware accelerator nodes in parallel to improve performance of their application, a small change in their application code introduces a refactoring of the application code. This refactoring forces the hardware engineer to re-design the hardware for every parallel node. As a result, a huge effort is required to re-build designs for an FPGA-based accelerator. This frequent design change considerably increase the wait time of the computational scientist to perform additional experiments, thereby reducing

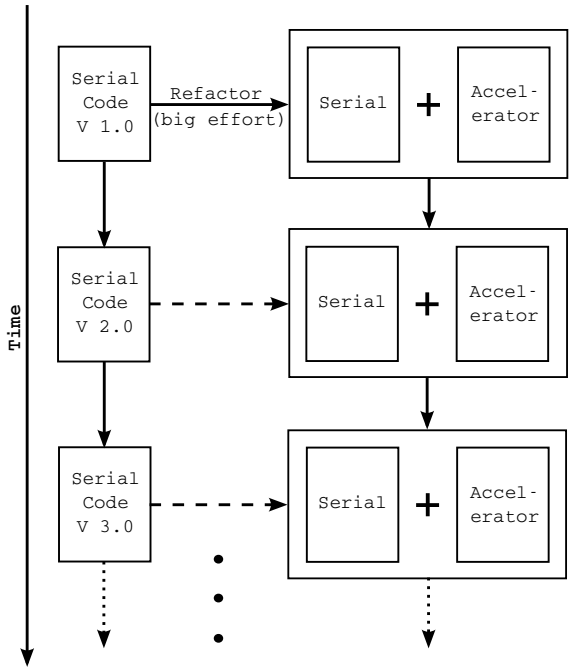

Figure 1.4: Key idea

their productivity.

Our hypothesis is that the frequent redesigns can be avoided by co-evolving the hardware design with the software. That is, as illustrated in Figure 1.4, only changes (or differences) from previous version is communicated, and the hardware is incrementally evolved or adapted. The overall effort is reduced by doing so, and productivity is improved. This key idea is not as simple as it sounds. However, we believe that if the hardware designers follow a set of design rules or guidelines for designing the initial hardware, the refactoring can be completely avoided, and performance and human effort can be preserved.

Thus, the thesis question we are trying to address is:—

> *Is there a set of hardware design guidelines for living computational science applications that can be easily communicated and is broadly applicable?*

If the answer is affirmative, then the guidelines help designers efficiently accommodate the evolving changes in the living computational science application and achieve

better performance and faster changes in the hardware design. If the answer is no, we have learned that these design guidelines cannot help designers efficiently accommodate the evolving changes. However, if these design guidelines are effective, then it could be used to formulate design guidelines for other hardware accelerator technologies.

As a computer engineer, we built hardware designs for several applications, such as exponential core design [46, 47], 3D-FDTD compute engine [48], N-body simulator [49], financial data feed handler [50], SpMV design [51], and Neural Network design [52]. In the process of building hardware designs for computational science applications, we realized that these applications could change. As we anticipated several design changes in these applications, we were motivated to formulate a set of design guidelines that could help the computer engineers or an FPGA specialist to co-evolve the hardware with the software. If this set of design guidelines is effective for living computational science codes for an FPGA accelerator, then these design guidelines can be used to experiment and formulate design guidelines for different accelerator technologies, such as GPUs, Cell/B.E, Intel's MIC, and ASICs.

1.4 Evaluation

To answer the thesis question, we propose a set of twelve design guidelines for hardware engineers to follow, with the expectation that it will help hardware designers to design hardware for living computational science applications. The set of design guidelines are then evaluated for its *effectiveness, communicability, and its broad applicability*. These design guidelines are simple and straight forward that fits in one page and are explained in 3.4. The guidelines can be easily understood and used by a hardware engineer to build FPGA hardware designs for living computational science applications. The set of twelve design guidelines are arranged in an order, so that, by following each guideline in the given order, they will help the hardware engineer to quickly understand the design guidelines and implement the hardware design. Thus,

we argue that "easily communicated" is self-evident.

In order to evaluate the guidelines for effectiveness and its broad applicability, we will conduct two major experiments. The first experiment evaluates the design guidelines for its effectiveness using two applications and an operation from the literature. The second experiment evaluates the guidelines' broad applicability.

1.4.1 Experiment 1: Effectiveness

To evaluate the set of design guidelines for its effectiveness, two computational science applications and a computational science kernel is chosen from the literature. These applications are photo-voltaic (PV) system modeling and electromagnetic wave analysis using finite difference time domain (FDTD). The computational science kernel is sparse matrix-vector (SpMV) multiplication operation. This kernel is widely used in many computational science applications. These computational science applications are created by the computational scientists, and the hardware for these applications and the kernel are created by hardware engineers (FPGA specialist). Hence, these applications and the kernel are designed by a co-design team involving computational scientists and hardware engineers. A software version (version 1.0) of these applications and kernel exists (or will be recreated). The software version will capture the behavioral findings of the application and the kernel from the literature. Real datasets will be used to test the functionality of these applications and kernel. However, if real datasets are not available, synthetic data will be used, provided the performance of the application does not change. The software version 1.0 code of each application is refactored into serial and accelerator code section, as shown in Figure 1.3. The accelerator code section is implemented on the accelerator referring to the literature, and this design is termed as *literature followed hardware design (LFHD)* or *control group*. These LFHD designs are derived from similar best designs that are existing in the literature. The accelerator code section of version 1.0 serial code is also built using the set of design guideline and is referred to as *guideline*

followed hardware design (GFHD) or *experimental group.*

Historical changes or additions from literature are introduced into version 1.0 software code to generate new versions (version 2.0, and 3.0) of software code. These historical changes or additions are chosen from the literature for respective applications and the kernel. These changes or additions are incorporated into the LFHD and into GFHD. Since these changes are historical, the hardware or the FPGA part used for their implementation could be old and outdated. As these FPGA parts are outdated, and cannot be used in the current computing platform, these LFHDs are to be designed and evaluated on a currently available FPGA (Virtex-4 FX60). In order to evaluate the designs effectively, and to maintain the fidelity of the designs, the architecture of the design, performance, and resource count has to be preserved when the LFHDs are built on a currently available Virtex-4 FX60 FPGA.

In order to preserve the architecture of the design, the design is reproduced to furthest extent possible on the Virtex-4 FX60 FPGA. The key point is that even though many of the older systems are not available, we are recreating the same environment. Similarly, to preserve the resource count, we constrain the design on Virtex-4 FX60 FPGA to recreate what was possible historically. In order to evaluate the designs effectively, the performance of the designs under evaluation (version 1.0, 2.0, and 3.0) needs to be preserved. Since the performance of an FPGA design mainly depends on the frequency of operation, the designs are operated at the reported frequency. Thus, the performance of the LFHDs are preserved, to maintain the fidelity of the design.

As the changes or additions from the literature is incorporated into the LFHD and into the GFHD, three key measurements are made. These measurements are (1) performance, (2) resource usage, and (3) lines of code changed and/or added (Δ LOC). Lines of code changed is an approximate measure of effort involved in changing a hardware design. These measurements are tabulated and plotted, and comparisons

are made, as shown in Figure 1.5. Plots are drawn for all the measurements comparing LFHD with GFHD (control group versus experimental group). The effectiveness of the design guidelines is measured by comparing the data points in the plots (performance, resource usage, and lines of code changed). If the data points for performance of GFHD and LFHD track each other with GFHD data points having higher or equal values, then the set of design guidelines is considered effective in terms of performance. That is, it is considered effective when the performance of guideline followed design (all versions) tracks the performance of literature design (all versions) as the scientific application changes (or new versions are created).

If the data points for resources of GFHD and LFHD track each other, then the set of design guidelines is considered effective in terms of resources. That is, as new versions of scientific code is created, if the resources used for guideline followed design is similar to the resources used for building literature followed design, then it is considered as effective in terms of resources. Similarly, if the data points for lines of code change for GFHD and LFHD diverge with GFHD data points having lower values, then the set of design guidelines is considered effective in terms of lines of code changed. When the data point diverge, there is large difference in the lines of code between literature design and guideline followed design to implement a change. That is, the effort involved in building guideline followed design is lesser than building literature followed design as the scientific application code changes.

If the performance and resource utilization data points for GFHD and LFHD track each other, and the lines of code diverge for GFHD and LFHD, then the degree of effectiveness is further used to affirm or deny the thesis question. On the contrary, if the performance and/or resource utilization data points do not track, and/or the lines of code data points do not diverge, then we can deny the thesis question.

If the plots show effectiveness, the degree of effectiveness is then calculated by classifying the effectiveness into bad, good, and excellent. If the performance and

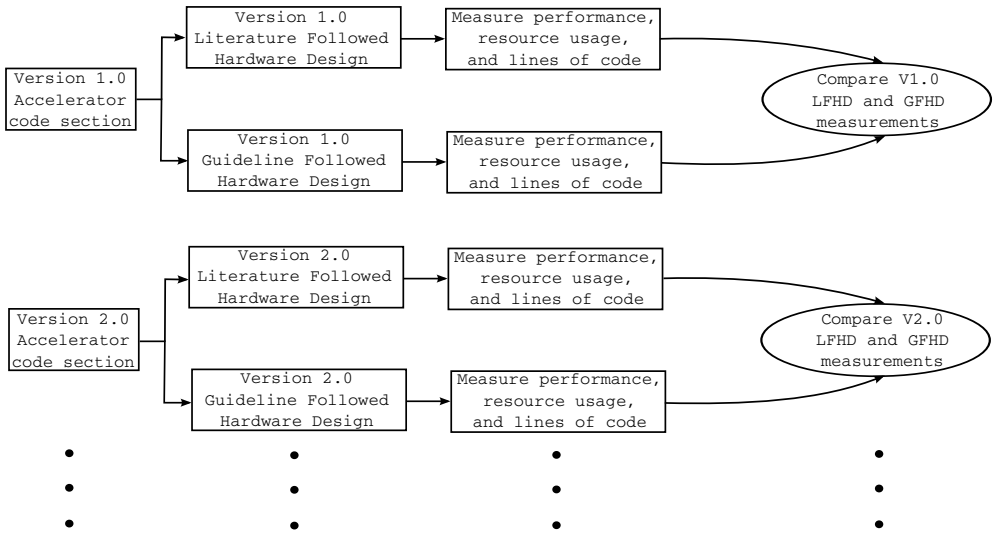

Figure 1.5: LFHD and GFHD evaluation

lines of code changed are classified as good or excellent, and the resources as bad or good or excellent for all the applications and the kernel, then we can ascertain that the set of guideline helps designers to effectively accommodate the evolving changes in the living computational science application. On the other hand, if the performance and lines of code changed are classified as bad, then the set of guidelines does not effectively accommodate the evolving changes in the living computational science application.

1.4.2 Experiment 2: Broad Applicability

To answer the question of broad applicability, besides the applications and kernel, a set of six applications are drawn from the literature and are evaluated using the set of guidelines. A guideline fitness plot is introduced to relate the number of guidelines followed to the performance of the application. The guideline fitness plot is plotted for each application, and each of these plots are combined to show the trend of application's performance with respect to the guidelines.

In order to answer the thesis question, the design guidelines needs to be evaluated, and these evaluations are carried out on a hardware platform. To get a better

understanding of the hardware platform, the readers can go through Chapter 2. This chapter also provides an insight of the various existing tools and practices for living computational science applications. Once the readers have a better understanding of the hardware platform and the existing practices, Chapter 3 gives the scope of the design guidelines, and the methodology to implement these guidelines. The evaluation and validation of these guidelines are carried out on two applications and a kernel, and is discussed in Chapter 4. The results are presented in Chapter 5 with the conclusion in Chapter 6.

CHAPTER 2: BACKGROUND

The designs and experiments in this dissertation are performed using an FPGA hardware platform. This section gives a brief description of FPGAs.

2.1 Field-Programmable Gate Arrays

An FPGA is an integrated circuit that mainly consists of logic blocks and interconnects. As the name suggests, an FPGA is configured by the end-users and not by the manufacturer. This feature gives end-users the ability to configure the device according to their needs. The term configuration is defined as the process of configuring the logical blocks and connecting them in a desired fashion to achieve a desired logical function. The work related to the dissertation was done using a Xilinx Virtex 4 FPGA device. This section will give a brief idea of the major components of a Virtex 4 FPGA that are listed below:

- Configurable logic blocks (CLBs)
- Digital clock managers (DCMs)
- Block RAMs (BRAM)
- PPC 405 processor
- XtremeDSP tile
- Ethernet MAC

2.1.1 Configurable Logic Blocks

Configurable logic blocks (CLBs) are the main building blocks of the FPGA that are used to build sequential and combinational circuits. A CLB consists of four

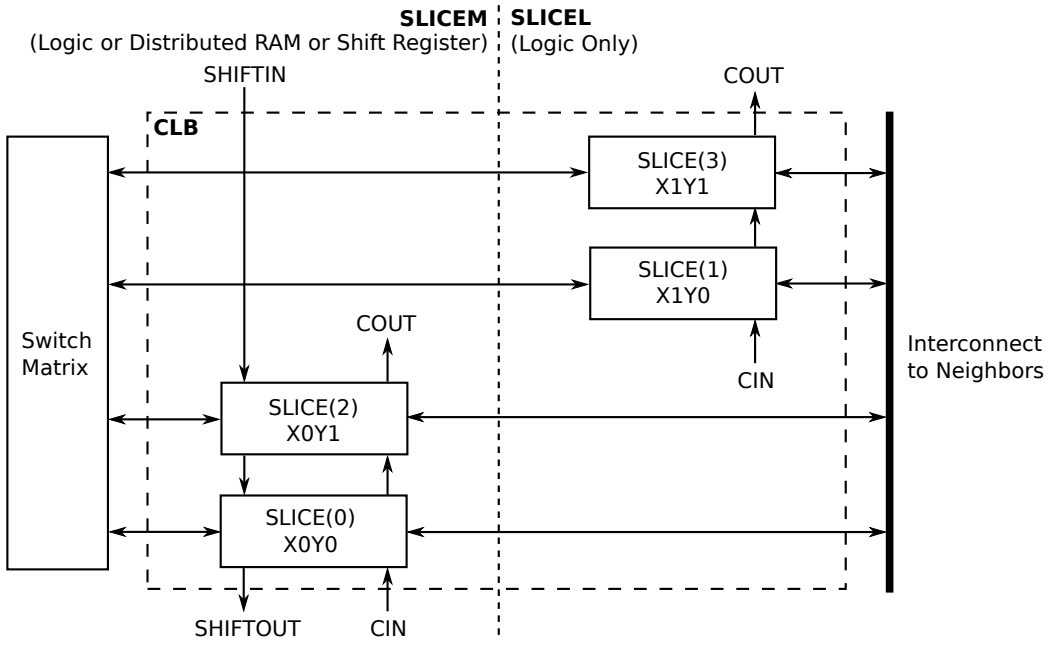

Figure 2.1: Configurable logic blocks and slices of a Virtex 4 FPGA

interconnected slices, shown in Figure 2.1. These slices are grouped in pairs and named as SLICEM and SLICEL. Both SLICEM and SLICEL have two look-up tables and two flip-flops. The SLICEM group has a distributed RAM and a 16-bit shift register. These SLICEL group can only be used for logic, however, SLICEM can be used to store a 64-bit word. The look-up table is a 4-input look-up table, and it is used to realize the digital logic functions.

2.1.2 Digital Clock Managers

The digital clock manager (DCM) generates clock signals for various modules in the FPGA. They provide a wide range of functions. It has a delay-locked loop (DLL) to eliminate clock delays, and it has features for doubling the clock and dividing the clock according to the requirements of the design. The DCM can also generate a phase-shifted clock that is required for designs and for interacting with the main memory.

2.1.3 Block RAMs

Block RAMs are used for on-chip storage. Each block RAM can store up to 18 Kbits of data. Writes and reads are synchronous to the clock. The data in the block RAM can be accessed using two ports that can be used for writing or reading. The block RAMs can be combined to form any wide or any deep memory blocks. For some combinations of block RAM, a small amount of fabric logic may be used.

2.1.4 PPC 405 Processor

The PowerPC (PPC) processor core is a hardware IP in the Virtex 4 FX series FPGA. The PPC405 can work at a maximum frequency of 400 MHz. The PPC 405 processor can interface the user-defined cores through the processor local bus (PLB). It is a 32-bit address and 64-bit data bus. The PPC 405 can also interface using the device control register (DCR) and the on-chip memory (OCM) controller interface. The DCR helps in interfacing on-chip registers for device control. The OCM helps in adding more main memory to the processor. A joint action test group (JTAG) port is also provided to facilitate debugging of the software code running on the processor. The PPC 405 has an auxiliary processor unit (APU) that helps the designer to extend the PPC 405 instruction set. An instruction that is issued, is decoded both by the processor unit and by the APU. If the processor unit is able to generate the control signals, then the instruction is executed. However, if the APU recognizes the instruction, then the operands are forwarded to the fabric co-processor module (FCM). The FCM then computes on the operands, and the results are written back to the processor's registers for a write back. The complete process is shown in Figure 2.2

2.1.5 XtremeDSP Tile

Each XtremeDSP tile in a Virtex 4 FPGA device has two DSP48 slices. A DSP48 slice can support many functions, such as multiplication, multiplication-accumulation (MACC), multiplication followed by addition, three input addition, barrel shift-

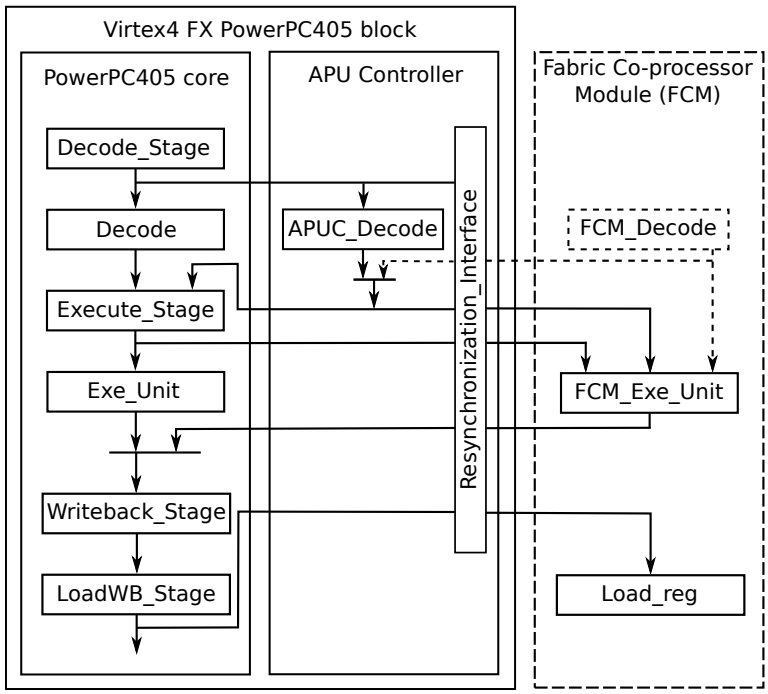

Figure 2.2: PPC, APU, and FCM interaction

ing, and magnitude comparison. Each DSP48 slice has a 18-bit×18-bit 2's complement multiplier followed by a 48-bit signed adder/subtractor/accumulator. Each XteremeDSP can be cascaded without the use of fabric logic. The XteremeDSP tile can be used for building floating-point operations, such as addition and multiplication. Since these XtremeDSP tiles are hardware IP, their operating speeds are high.

2.1.6 Ethernet MAC Block

The Virtex 4 FPGA Ethernet Block contains two Ethernet MACs. Each Ethernet MAC supports 10/100/1000 Mbps data rates. Each Ethernet MAC has an address filter to accept or reject packets. The Ethernet block has a clock management module that configures the output clock frequency according to the Ethernet MAC speed setting and the mode settings.

As the scientific application code grows or changes, it becomes difficult and time consuming to build hardware designs using hardware description languages (HDLs). These factors motivated researchers, scientists, and companies to build C-to-HDL

conversion tools and formulate design methodologies for FPGA designs.

2.2 Related

2.2.1 Hardware/Software Co-Design

For many years hardware engineers are building designs with hardware and software components. A huge time and effort is required to design a system with hardware and software components. Moreover, a large design problem makes it unlikely for human designers to optimize all the objectives of the design. In order to meet and optimize the system level objectives, hardware and software are designed concurrently through hardware/software co-design [53]. This co-design methodology is generally applied to embedded system designs, system on chip designs, and others with rapid prototyping requirements. Computational scientists and hardware engineers can take advantage of hardware/software co-design methodology to design hardware for computational science applications. However, a change or additions in the code due to improving the fidelity of experiments or reducing the execution time or exploring new phenomena could result in introducing a new cycle of hardware/software co-design.

2.2.2 Scientific Application Design Methodologies

The work by Herbordt et al. [41] discusses a set of design methodologies for high performance reconfigurable computing (HPRC) applications. This work lists a set of twelve methods to improve performance of non-trivial HPRC applications. The work uses computational biology and molecular dynamics application as a case study for developing the design methods. Some of the key methods discussed are application restructuring, design and implementation, arithmetic operations, and integration issues. The work discusses general methods involved in improving throughput and performance of HPRC applications; however, it does not discuss design methodologies for code changes in living computational science applications.

2.2.3 HDL Coding and Design Guidelines

There are many manuals and techniques to code a hardware description language (HDL). These are published in various books [54], articles, and websites. The set of design guidelines are complementary to the existing HDL coding guidelines. Similarly, there are many design techniques and guidelines for building FPGA-based hardware design, and these set of design guidelines are followed in addition to the existing HDL design techniques. These set of design guidelines augment specifically living computational science applications.

2.2.4 C-to-HDL Conversion Tools

C-to-HDL conversion tools converts C or C-like code to a HDL. The converted code can then be synthesized to configure an FPGA. The conversion tools help to overcome the issues related to application growth and changes in functionality of a scientific computing application. Over the past two decades, there has been a continuous effort made by different companies and researchers to build C-to-HDL conversion tools. A few of them are listed below, and their advantages and disadvantages are also listed.

SystemC [55] was an effort made by an open systemC initiative (OSCI) that is an open source extension of C++ for hardware/software co-design. The syntax of SystemC is a mix of C++ and VHSIC hardware description language (VHDL). One of the major advantage of SystemC is that the programmer can co-design and co-simulate a system. The drawback of SystemC is that it does not generate synthesizable HDL. However, a two step process is required to generate synthesizable HDL.SystemC is system-level modeling language. However, there has been lot of effort [56] to generate hardware synthesizable code and to generate transparency in the algorithm, which exposes the relationship between inputs and outputs.

Handel-C [57] is a high level programming language from Celoxica. Handel-C uses C-like constructs for inherent parallelism. The communication between parallel blocks is done using channels or first-in first-out (FIFO) hardware. Handel-C

does not support floating-point operations; however, library calls can be made for floating-point operations. An interface construct is used to communicate with external devices and external logic. Every interface construct has port definitions for communication. Handel-C supports pointers and pointer-to-functions and it code can also be simulated.

Dime-C was developed by Nallatech [58] and generates VHDL that can be port-mapped into other hardware designs. It supports American National Standards Institute (ANSI) C constructs; however, some special constructs help in optimizing the output VHDL code and support floating-point operations. The resources are shared between parallel processes, and it automatically optimizes code for parallel and pipeline implementation. Dime-C designs can be simulated, and, whenever possible, Dime-C uses DSP48 slices instead of fabric logic for IP cores. It can create two reads/writes or one read/write interface for the BRAM blocks. The major disadvantage of Dime-C is that it does not create optimized HDL[59].

ImpulseC was developed by Impluse Accelerated Technologies [60]. Impulse CoDeveloper is the integrated development environment (IDE) used for coding and simulation. This IDE includes ImpulseC, interactive parallel optimizer, and platform support packages (PSP) that can be configured for a wide range of FPGA-based computing systems. PSPs specify the type of FPGA on the board and how Impulse CoDeveloper can convert the code for that particular FPGA. A PSP is created to configure Impulse CoDeveloper for a specific FPGA board. Once Impulse CoDeveloper is configured through a PSP for a particular FPGA board, the end-user can write C code to design systems. The designs built using ImpulseC can be partitioned into software and hardware blocks. The software block runs on the processor of the FPGA, and the hardware block is converted into HDL to be synthesized. The user can transfer data between software and hardware blocks using *co_streams, co_registers, and co_signals*. Co_streams are FIFOs, co_registers are registered signals and co_

signals are used for handshaking and interrupts. The user can design systems completely that are written in ImpulseC or can design using VHDL and Impulse C. In cases where the designs are mixed (i.e., VHDL and Impulse C), the user can interface the ImpulseC system using co_streams and co_registers to the VHDL design.

Although C-to-HDL tools help hardware engineers to build deigns for FPGA, some of these tools don't support floating-point, some of them are used for simulation, and most of them do not generate resource optimized HDL. Some of C-to-HDL tools do not allow fine-grain control of the built hardware [61]. As computational scientist improve the fidelity of the code or reduce execution time or wish to explore new phenomena, the code changes or grows. As these scientific code grows, additional FPGA resources may be required. If C-to-HDL tools are used to build designs for growing computational science codes, these designs could use more resources when compared to designs built using VHDL.

The next section discusses scope and methodology for the thesis question we are trying to answer.

CHAPTER 3: SCOPE AND METHODOLOGY

As scientists try to simulate real world problems, their application code will continue to change overtime due to two facts: a new discovery lead to new questions and a desire to study new phenomena. If the scientists use a hardware accelerator to increase the rate of execution, then every new version would require substantial effort to accomplish the changes in the accelerator code. In this work, we assume that the accelerator is an FPGA device, and the following sections discuss the key idea, scope of the work, pre-design, and hardware design for living scientific applications.

3.1 Key Idea

Using a hardware accelerator core poses a real disadvantage to scientists whose scientific code is changing or evolving over time. Whenever, there is a change in the code, a refactoring is required. Every refactoring causes a computer engineer to inspect the source code and create accelerator-specific code plus some sequential code. This process is illustrated by the graphic originally shown in Chapter 1 and reproduced in Figure 3.1. In order to reduce the effort of refactoring, the hypothesis of this thesis is that designs can be built such that there is minimum effort for incorporating the changes. The changes from the new code, and the previous version of sequential code and accelerator code, is used to generate a new version of sequential and accelerator code. The key idea is illustrated by the graphic reproduced in Figure 3.2. The dotted lines represent that only the changes are extracted, and the arrow from the previous serial and accelerator block shows that the previous version of serial and accelerator code is used for creating the new version of serial and accelerator code. The overall effort is reduced by doing so, and the productivity is also improved. In order to accomplish the key idea, the designers need to design the hardware in a cer-

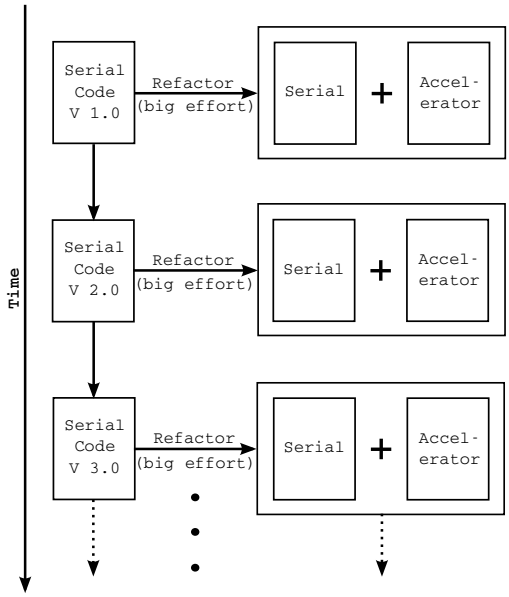

Figure 3.1: Effort due code evolution

tain style. This dissertation offers hardware design for the above discussed hardware. Before discussing the design, the next section discusses the scope of the design/work.

3.2 Scope of the Work

In order to answer the thesis question, the dissertation offers a set of design guidelines that, when followed, helps the hardware engineer to build designs that are effective when the design changes due to changes in the scientific application code. As discussed earlier, these design guidelines are applicable to FPGA designs that has shown consistent speed-up when compared to other hardware platforms used for scientific computing. This set of design guidelines is also specific for living computational science applications. In order to answer the thesis question, two applications and an operation are chosen from the literature, and the guidelines are evaluated for effectiveness, communicability, and broad applicability. These applications and the operation are built on a single-node FPGA, and the guidelines are evaluated using the single-node FPGA infrastructure. The next section gives a brief idea of analyzing the application code, and dividing the application code into serial and accelerator

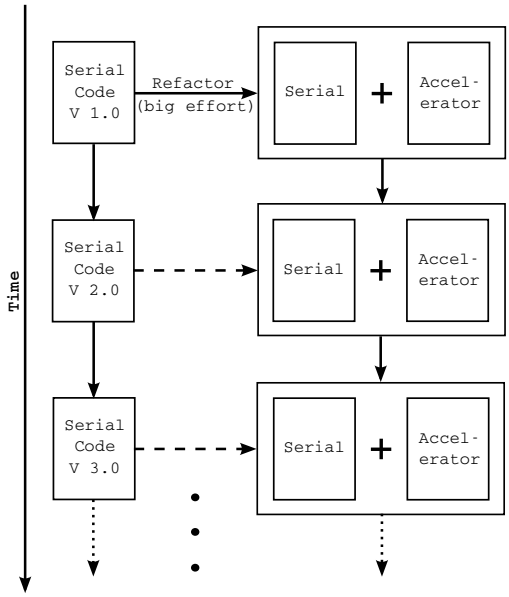

Figure 3.2: Key idea

code.

3.3 Analysis of Sequential Code

In order to speed-up a living scientific code, the code is profiled and analyzed, and the time consuming section/sections are identified, shown in Figure 3.3. One or more time consuming sections constitutes the accelerator code section. Once the accelerator section is identified, the number of parallel paths are identified, and the approximate time taken to marshal data between the software and hardware is calculated. With the number of parallel paths, marshaling time, and fraction of time spent (from profile information) in the accelerator code, an approximate speed-up is computed using Amdahl's law. The speed-up computed is just an approximate value as the actual value can be only calculated after the accelerator code is implemented using an FPGA. If the computed speed-up is not acceptable to the user, then the accelerator section coverage is increased so that the time spent in the section is increased. The increase in the coverage of the accelerator section is achieved by increasing the lines of code or by adding more time consuming sections. An approximate speed-up due

to the new accelerator section is again computed using Amdahl's law.

Assuming the speed-up is achieved using several time consuming sections, the speed-up of an individual time consuming section is defined by equation 3.2:

$$\gamma = \frac{hardware\ speed}{software\ speed} = \frac{\frac{1}{hardware\ time}}{\frac{1}{software\ time}} = \frac{software\ time}{hardware\ time} \tag{3.1}$$

$$\gamma(i) = \frac{s(i)}{h(i) + m(i)} \tag{3.2}$$

Where $h(i)$ and $s(i)$ are the time spent in hardware and software sections, respectively, for the time consuming section i. The time spent to marshal data between the processor and the FPGA is $m(i)$. Assuming that we use a single time consuming section to form the accelerator section, and knowing the fraction of time spent in a time consuming section ($k(i)$) from the profile information, we can calculate the overall speed-up using Amdahl's law. Amdahl's law states that if F is the fraction that can be enhanced and $(1 - F)$ is the fraction that cannot be enhanced, then the maximum speed-up that can be achieved can be computed using equations 3.3 and 3.4. Rewriting equation 3.4 with time consuming section $k(i)$ and its speed-up $\gamma(i)$ results in equation 3.5.

$$Speedup_{overall} = \frac{1}{(1 - Fraction_{enhanced}) + \frac{Fraction_{enhanced}}{Speedup_{enhanced}}} \tag{3.3}$$

$$Speedup_{overall} = \frac{1}{(1 - F_{enhanced}) + \frac{F_{enhanced}}{Speedup_{enhanced}}} \tag{3.4}$$

$$\Gamma = \left[(1 - k(i)) + \frac{k(i)}{\gamma(i)}\right]^{-1} \tag{3.5}$$

If the accelerator section consists of a set of time consuming sections $\mathbb{D}$, then the overall speed can calculated, as shown in equation 3.6.

$$\Gamma(\mathbb{D}) = \left[1 + \sum_{i \epsilon \mathbb{D}} \left(\frac{k(i)}{\gamma(i)} - k(i)\right)\right]^{-1} \tag{3.6}$$

If the new speed-up is not acceptable, then the accelerator section coverage is again increased. This process is continued until an acceptable speed-up is achieved. If such a speed-up cannot be achieved, then the acceleration of the application is not possible. If the application can be sped-up using an accelerator, an FPGA part is selected. The accelerator section's parallel paths are identified, and it is then divided into small hardware blocks that could fit the FPGA. These hardware blocks are chosen in such a way that the blocks can be reused. If possible, the hardware blocks are pipelined. The hardware blocks could be a simple arithmetic operation or a compound operation.

Once the hardware design is decided, the resource count is calculated. If the total resource count is less than the FPGA resources, a speed-up computation is again performed using equations 3.4, 3.5, and 3.6. If a desired speed-up can be achieved, then the set of guidelines is followed. If the resource requirement is more than the FPGA resources, then a better FPGA is chosen, and the hardware blocks are again designed for the new FPGA. The next section gives a detailed example of a living computational science application hardware design.

3.3.1 Example: Electrodynamics Application

For demonstration purposes, let us profile electrodynamics application for analyzing microstrip discontinuities (FDTD). The profile output is shown in Figure 3.4. The time consuming sections (kernel) of the application are $amp1$, $amp2$, $amp3$, $far1$, $far2$, and $far3$ subroutines. Let's compute the speed-up using equation 3.5. If the $amp1$ kernel is implemented on an FPGA accelerator, then the value of $k(i)$ is 16.97%. Assuming we achieved a speed-up of 10x compared to the software using an FPGA accelerator, the overall speed-up is computed as 1.094x. However, if $amp2$, $amp3$, $far1$, $far2$, and $far3$ are also implemented on an FPGA accelerator with an average

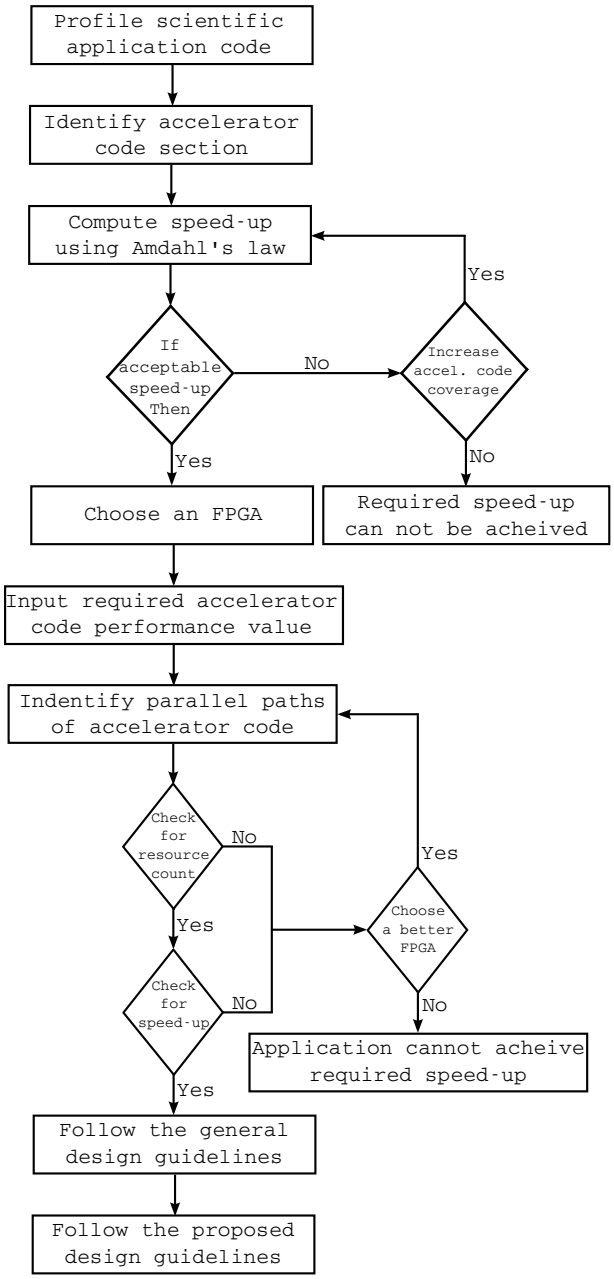

Figure 3.3: Analysis and pre-design

speed-up of 10x compared to the software, then the overall speed-up is computed as 5.29x, using equation 3.6. The $k(i)$ values for each kernel are shown in Figure 3.4.

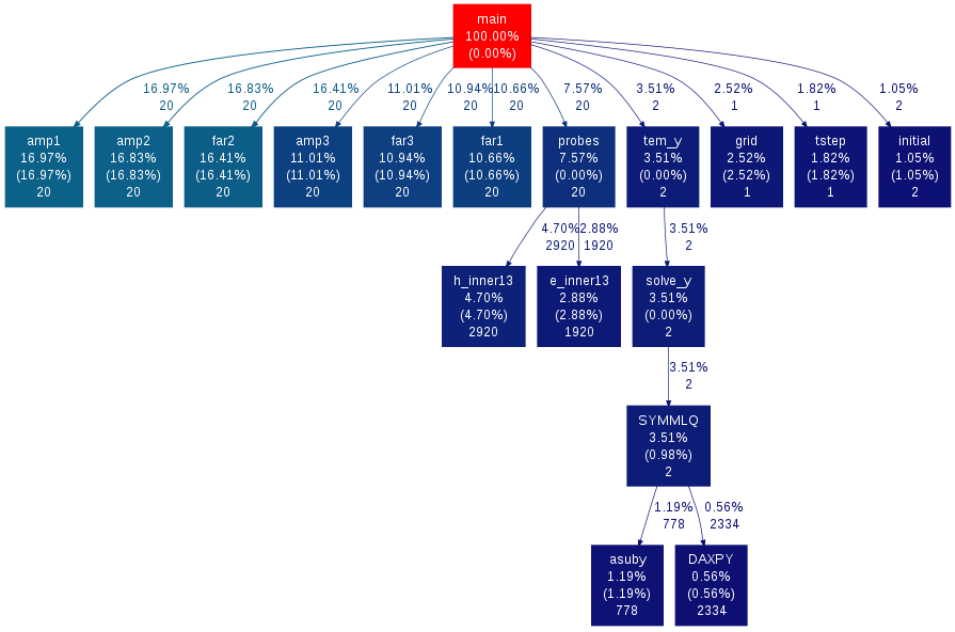

Figure 3.4: FDTD Profile Information

In this application, the material under experiment is spatially divided into small cubes (unit cells) along three dimensions. The problem is computing the magnetic and electric field in a leap frog fashion. In practice, the material is divided into a huge number (order of 100K) of tiny cubes (unit cells), and the computation is repeated 10,000 times over each unit cell. In order to perform the computation, the computational domain is chosen such that every computational sub-domain consists of thousands of similar unit cells. That is, the whole region under test is divided into smaller regions along one axis. Electromagnetic properties are then computed in parallel over these smaller regions to improve performance. As discussed above, the kernel is shown in equations 3.7 – 3.12, and it is broken down into smaller hardware blocks that can be reused. To build an efficient design for the FPGA, two important questions are to be addressed. The first question is the number of parallel cores and its dataset.

$$E^{n+1}_{x_{i,j,k}} = E^{n}_{x_{i,j,k}} + C_1\left(H^{n+1/2}_{z_{i,j+1,k}} - H^{n+1/2}_{z_{i,j,k}}\right) + C_2\left(H^{n+1/2}_{y_{i,j,k+1}} - H^{n+1/2}_{y_{i,j,k}}\right) \quad (3.7)$$

$$E^{n+1}_{y_{i,j,k}} = E^{n}_{y_{i,j,k}} + C_3\left(H^{n+1/2}_{x_{i,j,k+1}} - H^{n+1/2}_{x_{i,j,k}}\right) + C_4\left(H^{n+1/2}_{z_{i+1,j,k}} - H^{n+1/2}_{z_{i,j,k}}\right) \quad (3.8)$$

$$E^{n+1}_{z_{i,j,k}} = E^{n}_{z_{i,j,k}} + C_5\left(H^{n+1/2}_{y_{i+1,j,k}} - H^{n+1/2}_{y_{i,j,k}}\right) + C_6\left(H^{n+1/2}_{x_{i,j+1,k}} - H^{n+1/2}_{x_{i,j,k}}\right) \quad (3.9)$$

$$H^{n+1}_{x_{i,j,k}} = H^{n}_{x_{i,j,k}} + C_7\left(E^{n+1/2}_{z_{i,j+1,k}} - E^{n+1/2}_{z_{i,j,k}}\right) + C_8\left(E^{n+1/2}_{y_{i,j,k+1}} - E^{n+1/2}_{y_{i,j,k}}\right) \quad (3.10)$$

$$H^{n+1}_{y_{i,j,k}} = H^{n}_{y_{i,j,k}} + C_9\left(E^{n+1/2}_{x_{i,j,k+1}} - E^{n+1/2}_{x_{i,j,k}}\right) + C_{10}\left(E^{n+1/2}_{z_{i+1,j,k}} - E^{n+1/2}_{z_{i,j,k}}\right) \quad (3.11)$$

$$H^{n+1}_{z_{i,j,k}} = H^{n}_{z_{i,j,k}} + C_{11}\left(E^{n+1/2}_{y_{i+1,j,k}} - E^{n+1/2}_{y_{i,j,k}}\right) + C_{12}\left(E^{n+1/2}_{x_{i,j+1,k}} - E^{n+1/2}_{x_{i,j,k}}\right) \quad (3.12)$$

3.4 Hardware Design

As scientists discover new phenomena of their application code, they incorporate these new findings in the code. If scientists have an accelerator to accelerate their code, it becomes more difficult to incorporate those changes into the accelerator code. As discussed in the key idea section, the goal is to come up with a hardware design that can handle these changes. The design of such a hardware is not as simple as it sounds. We believe that if the initial hardware is built following a set of design guidelines or rules as tabulated in Table 3.1, then the successive hardware changes due to application code changes which can be easily incorporated.

This set of design guidelines form the basis of this work. These design guidelines are followed when the hardware for the scientific application is designed. A high-level design is shown in Figure 3.5. As shown in the figure, the serial code is divided into serial and accelerator code. The accelerator code and the design guidelines are then used to build the hardware design. The accelerator code is also used to build the software code that runs on the processor of the FPGA. The main function of software code is to configure or to connect the various resources in the FPGA, according to the requirements of the application. The resources are chosen according to the initial version (version 1.0) of the application. As the application code or the

Table 3.1: Design guidelines for living computational science applications

1	*Arrange and optimize input/output data for all compute blocks*
2	*Adhere to a widely accepted method of computation for arithmetic functions/operations*
3	*Build controller for every not likely to change compute blocks*
4	*Introduce a configurable dataflow path to connect resources of likely to change compute blocks*
5	*Introduce on-chip memory with configurable read/write logic, if necessary*
6	*Introduce dependency indicators to enhance parallel computations, if necessary*
7	*Use microcode to specify dataflow path, memory read/write, and parallel computations*
8	*Maximize resource utilization by improving runtime parallelism*
9	*Forward results between compute blocks/resources, if possible*
10	*Achieve functionality, and optimize design & resources to improve performance*
11	*Maximize computation until maximum memory bandwidth is utilized*
12	*Use a large and real dataset for test cases*

serial code changes, the accelerator code also changes. These changes (or differences) are communicated to the next version of serial and accelerator code. The changed accelerator code is then used to build the new software code for the processor in the FPGA. The next section discusses the evaluation of the set of design guidelines and its validation.

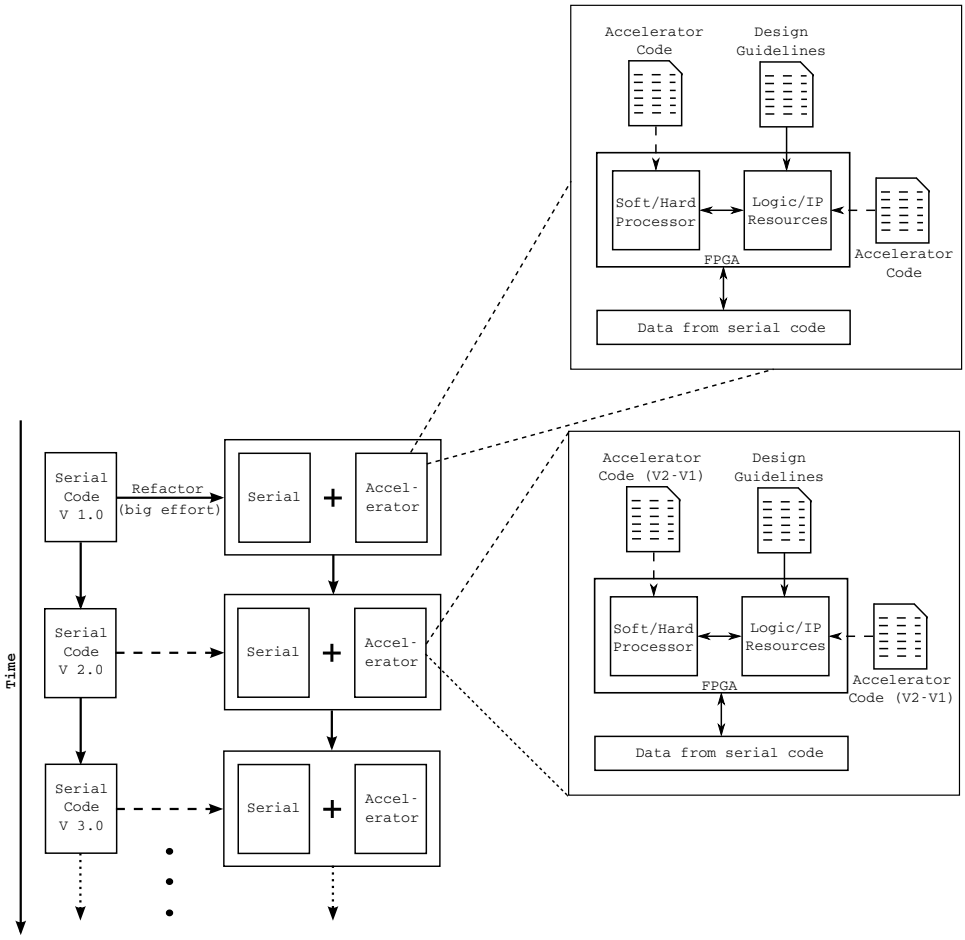

Figure 3.5: Hardware design

CHAPTER 4: EVALUATION AND VALIDATION

The set of design guidelines can be evaluated and validated by measuring the effort, performance changes, and resource usage as the hardware design evolves with the application. It is also required to evaluate the design guidelines for easy communicability and broad applicability. Hence, this section will discuss in detail about the various evaluations and their methods. To summarize, the key evaluations that will be performed in this section are:

- Effectiveness of design guideline
- Communicability of design guideline
- Broad applicability of design guideline

4.1 Effectiveness of design guideline

The set of design guidelines that is used to built hardware designs, can be evaluated by recording the lines of code added or changed, change in performance, if any, and resource utilization as the application evolves from one version to the next. These measurements are then compared to the recorded measurements of existing designs in the literature. For the sake of simplicity, the hardware design built using the set of design guidelines will be referred as *guideline followed hardware design (GFHD) or controlled group*. Similarly, the hardware design built from literature will be referred as *literature followed hardware design (LFHD) or experimental group*. The important part of the evaluation is to determine effectiveness of the set of design guidelines. To be more specific, the effectiveness of the design guideline is measured by finding the lines of code that is required to incorporate a change or addition in the hardware

design due to new findings. By implementing these set of design guidelines, it is also important to measure whether there is any performance changes between the LFHD and the GFHD. The performance changes, if any, due to additions or changes in the application will also be recorded for both the LFHD and GFHD to give the hardware designers an idea of performance changes.

The hardware engineer who is following the set of guidelines would be interested to know whether there is any additional resource usage introduced by the set of design guidelines. To be specific, this dissertation is focused on FPGA designs, and the resource usage will be indicated as slices, DSP blocks, BRAM blocks. The resources used due to additions or changes in the application will also be recorded for both the LFHD and GFHD to give the hardware designer an idea of overhead, if any. To summarize, the key measurements that will be recorded to evaluate the effectiveness of the solution are:

- Performance
- Resource usage
- Lines of code added or changed

4.1.1 Design Guideline Evaluation Metrics

This dissertation offers a set of twelve guidelines, as shown in Table 4.1 for hardware engineers. It also shows the order in which the set of guidelines needs to be followed. This order has been established by considering the design path an FPGA designer would normally follow. A change in the order of the design guideline will introduce multiple design cycles. In order to evaluate the set of design guidelines, two computational science applications and a computational science kernel is chosen from the literature. These scientific applications are designed by computational scientists, and the hardware for the application and the kernel is designed by hardware

engineers. Hence, these designs are team co-designs consisting of hardware engineer and computational scientists.

The applications are photo-voltaic (P-V) system modeling and electromagnetic wave analysis using finite difference time domain (FDTD). The computational science kernel is sparse matrix-vector multiplication (SpMV) operation. Photo-voltaic modeling and simulation application models the power generation of solar panels using neural networks (NNs). Software and hardware versions (1.0 and 2.0) of P-V generation model is created from the literature. The version 2.0 model adds battery model and regulator model for overall P-V system performance. Electromagnetic wave analysis application uses FDTD method to compute electric and magnetic field values of a material under test. Software and hardware versions (1.0, 2.0, and 3.0) of electromagnetic wave analysis created from the literature consists of 2D transverse magnetic (TM) model of FDTD with different types of boundary conditions. Sparse matrix-vector multiplication operation provides optimized routines to compute multiplication of a large sparse matrix with a vector of values. Software and hardware version (1.0, 2.0, and 3.0) are created from the literature. Each version provides a better multiplication computation to reduce the overall execution time.

Each of the above application consists of three versions of software code. The software version (version 1.0) of these applications exists (or will be recreated) and will be tested with real datasets. The software version 1.0 code of each application is refactored into serial and accelerator code. The accelerator code is implemented on the accelerator, and is termed as literature followed hardware design (LFHD) or control group. These LFHD designs are derived from the existing designs in the literature. A GFHD is also built for the version 1.0 accelerator code and is termed as experimental group.

In order to introduce changes or additions into version 1.0 software code, historical changes from the literature is used. These changes to the version 1.0 code generates

Table 4.1: Design guidelines for living computational science applications

1	*Arrange and optimize input/output data for all compute blocks*
2	*Adhere to a widely accepted method of computation for arithmetic functions/operations*
3	*Build controller for every not likely to change compute blocks*
4	*Introduce a configurable dataflow path to connect resources of likely to change compute blocks*
5	*Introduce on-chip memory with configurable read/write logic, if necessary*
6	*Introduce dependency indicators to enhance parallel computations, if necessary*
7	*Use microcode to specify dataflow path, memory read/write, and parallel computations*
8	*Maximize resource utilization by improving runtime parallelism*
9	*Forward results between compute blocks/resources, if possible*
10	*Achieve functionality, and optimize design & resources to improve performance*
11	*Maximize computation until maximum memory bandwidth is utilized*
12	*Use a large and real dataset for test cases*

new versions (version 2.0, and 3.0) of software code. These changes or additions are incorporated into LFHD and into GFHD. As the changes or additions are incorporated into LFHD and into GFHD, three metrics will be used to compare the performance, resource utilization, and lines of code between LFHD and GFHD. The method of comparison is clearly outlined in Figure 4.1.

As the software code changes from one version to the next version, the performance, resource used, and lines of code changed for the LFHD and GFHD for each version is tabulated and plots are drawn for all measurements comparing the LFHD with GFHD. The tabulated values of performance, resource used, and lines of code changed are further used to measure the degree of effectiveness. For every version, If the GFHD's measured performance is equal and less than 1.1x times the performance of LFHD, then the set of design guidelines is effective in terms of performance and is classified as good. If the GFHD's performance is more than 1.1x times the performance of LFHD, then the set of design guidelines is effective in terms of performance and is classified as excellent. However, If the performance of the GFHD is less than

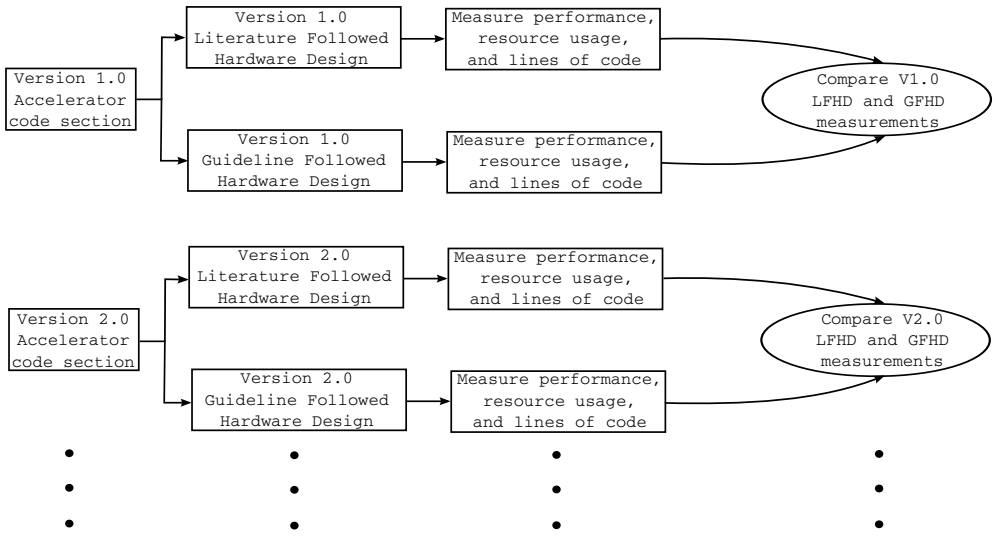

Figure 4.1: LFHD and GFHD evaluation

the LFHD, the set of design guidelines is non-effective in terms of performance and is classified as bad.

The resource utilization of an FPGA design has a vital role in understanding whether the changes in the application code increase the resource count, and if so, what percentage increase in the resources. For every version, if the GFHD requires less than 25% additional resources when compared with LFHD, then the set of design guidelines is effective in terms of resources, and is classified as excellent. If the GFHD requires between 25% and 50% of additional resources then the set of design guidelines is effective in terms of resources, and is classified as good. However, if the GFHD requires more than 50% of additional resources when compared to LFHD's resources, then the set of design guidelines is non-effective in terms of resources, and is classified as bad.

The effectiveness of the design guidelines is also determined by the lines of code added (or changed) to incorporate the changes or additions in the hardware of scientific application code. To be very specific the lines of code measurement will measure the lines of code added or changed in the new hardware due to application code

change. The lines of code is an approximate measure of effort required to implement a change in the hardware design. For every version, if the LFHD's lines of code changed is less than 25% additional code change when compared to GFHD, then the set of design guidelines is non-effective in terms of lines of code changed, and is classified as bad. If the LFHD's additional lines of code changed is between 25% and 50% when compared to GFHD, then the set of design guidelines is effective in terms of lines of code changed and is classified as good. If the LFHD's additional lines of code changed is more than 50% when compared to GFHD, then the set of design guidelines is effective in terms of lines of code changed and is classified as excellent.

To evaluate the design guideline with the help of metrics, this dissertation will use two applications and a computational science kernel as test cases. The results of the above metrics on each of the following applications and kernels will help to identify whether the design guidelines is a solution to tackle the problem of design changes in living computational science application code.

4.1.2 Applications and Kernel Under Test

The following section discusses about modeling and simulation of Photo-Voltaic system, electrodynamic analysis, and sparse matrix vector multiplication operation. This section presents the hardware design used for evaluation with its design parameters.

4.1.2.1 Modeling and Simulation of Photo-Voltaic (P-V) System

Computational scientists have conducted studies on modeling and simulating photo-voltaic (P-V) systems for several decades. They try to model P-V systems with the help of neural networks (NNs). NN is a computational model that is inspired by the structure of a brain neuron [62]. It consists of an interconnected group of neurons that process information and, in most cases, adapts itself to the changes based on the internal or external information by learning. NN is used to model complex input-output relationships that are then used to find similar patterns in any

given data. Likewise, NNs are used to capture the relationship between temperature, sunlight duration, relative humidity, and the power generated per day [63]. The information of temperature, sun light duration, and relative humidity is discretized over time and is fed into the NN to predict the power generated.

To evaluate the set of design guidelines, an P-V system modeled using NN will be used. The application's NN will be built using an FPGA and will have three layers: input, hidden, and output. Every node in the input, hidden, and output layer generates an output signal that is a function of a linear combination of the incoming signals. This function is called as the *activation function*. In this application, the activation function is a sigmoid function. The output of the nodes in the first layer are connected to the hidden layer. The training of the prediction model is done off-line using software, and the information is used to process the input data. Since the training is not required often, it would be appropriate to perform the training off-line [64]. The performance of the prediction model is measured as the rate at which the output is generated. In order to improve the prediction accuracy of the PV system, computational scientists find new ways to predict. As new ways are discovered, additions in the prediction model is required. This dissertation will introduce one addition to the basic version of P-V generation model. The basic version (version 1.0) and version 2.0 are taken from the literature. Figure 4.2 shows the versions of P-V generation model. These two models are then used to evaluate the set of design guidelines. The following sub-sections explain all the versions.

The version 1.0 P-V power generation model is modeled by Mellit et al. [65] and Mekki et al. [1]. The NN model takes temperature and total solar radiation as inputs. An overall configuration of the NN model is shown in Figure 4.3. The software version of this NN model will be built and tested using Matlab. The hardware block diagram of this NN model is shown in Figure 4.4. The number of clock cycles for the hardware based NN is 13 clock cycles. The computation is performed using 18-bit

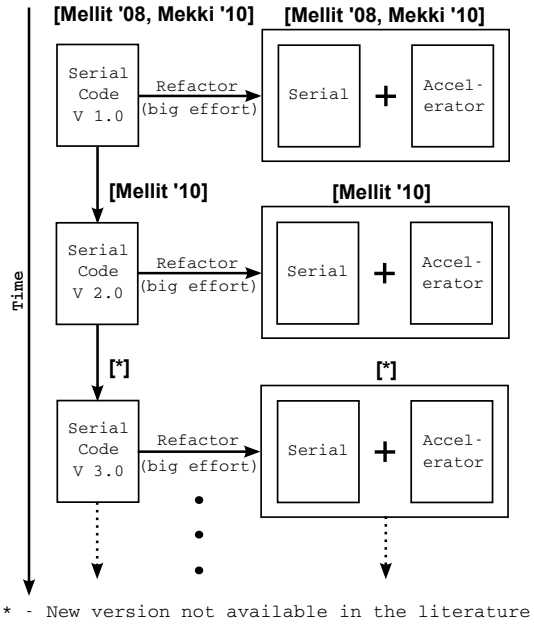

Figure 4.2: Versions of P-V system modeling using NN

Table 4.2: Version 1.0 P-V generation model HW design

Details	**Values**
FPGA	Virtex-II XC2v1000 FPGA
Frequency of operation	100 MHz
Clock cycles for computation	13 Cycles
Network Size	2×7×9×2
Computation Precision	18-bit fixed point
Activation function	Look-up table sigmoid

input values. The design is built on a Virtex-II FPGA. The hardware details are tabulated in Table 4.2. For the purpose of evaluation, the hardware designs (LFHD and GFHD) will be built on a Virtex-4 FX60 FPGA using 18-bit fixed point hardware cores.

The version 2.0 P-V power generation model is modeled by Mellit et al. [2] in 2011. The NN model takes temperature, and daily solar radiation as inputs. An overall configuration of the NN model is shown in Figure 4.5, 4.6, and 4.7. The software version of this NN model will be built and tested using Matlab. The hardware block diagram of this NN model is shown in Figure 4.8, 4.9, and 4.10. The number of clock

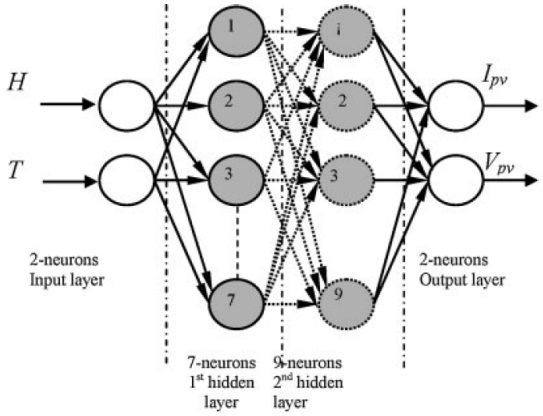

Figure 4.3: Version 1.0: P-V generation NN model [1]

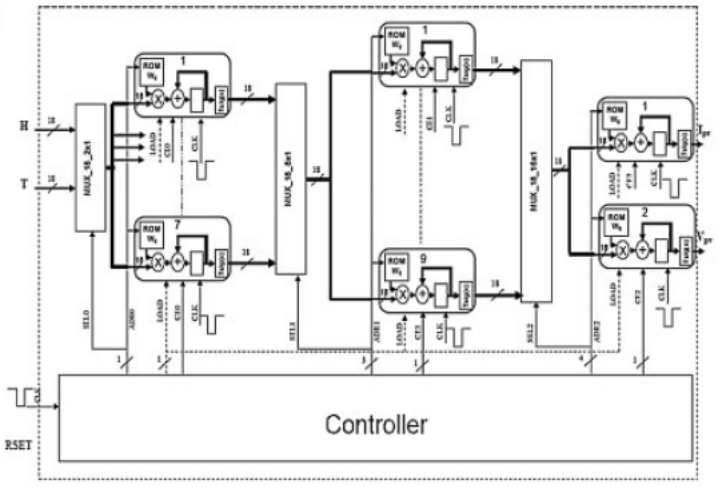

Figure 4.4: Version 1.0: P-V generation hardware design [1]

Table 4.3: Version 2.0 P-V generation, regulation, and battery model design

Details	**Values**
FPGA	Virtex-II XC2v1000 FPGA
Frequency of operation	100 MHz
Network 1 Size	2×7×9×2
Network 1 clock cycles for computation	13
Network 2 Size	2×14×10×1
Network 2 clock cycles for computation	19
Network 3 Size	3×7×12×1
Network 3 clock cycles for computation	15
Computation Precision	18-bit fixed point
Activation function	Look-up table sigmoid

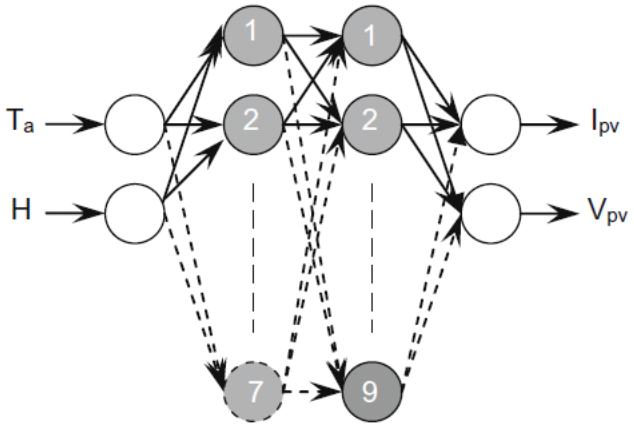

Figure 4.5: Version 2.0: P-V generator NN model [2]

cycles for the hardware based NN is 13 clock cycles for P-V NN model, 19 clock cycles for battery NN model and 15 clock cycles for regulator NN model. The computation is performed using 18-bit input values. The design is built on a Virtex-II FPGA (XC2v1000). The hardware details are tabulated in Table 4.3. For the purpose of evaluation, the hardware designs (LFHD and GFHD) will be built on a Virtex-4 FX60 FPGA using 18-bit fixed point hardware cores.

Design Parameters:

1. FPGA : As hardware designs cannot built using older FPGA parts, all the hardware designs (LFHD and GFHD) will be built on a Virtex-4 FX60 FPGA

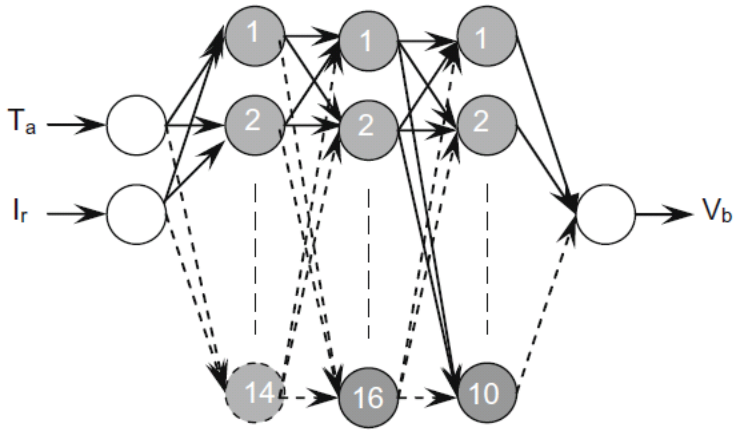

Figure 4.6: Version 2.0: P-V battery charging NN model [2]

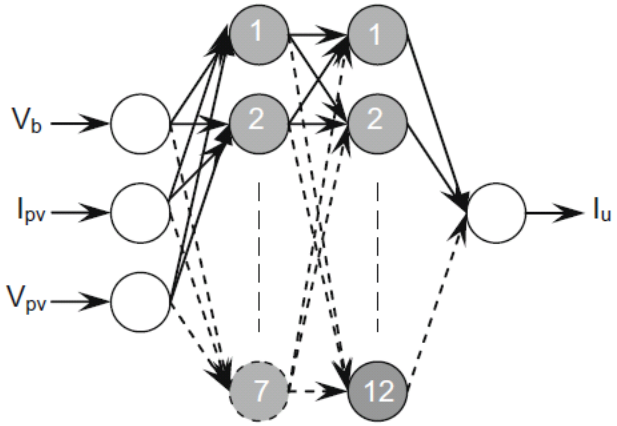

Figure 4.7: Version 2.0: P-V regulator NN model [2]

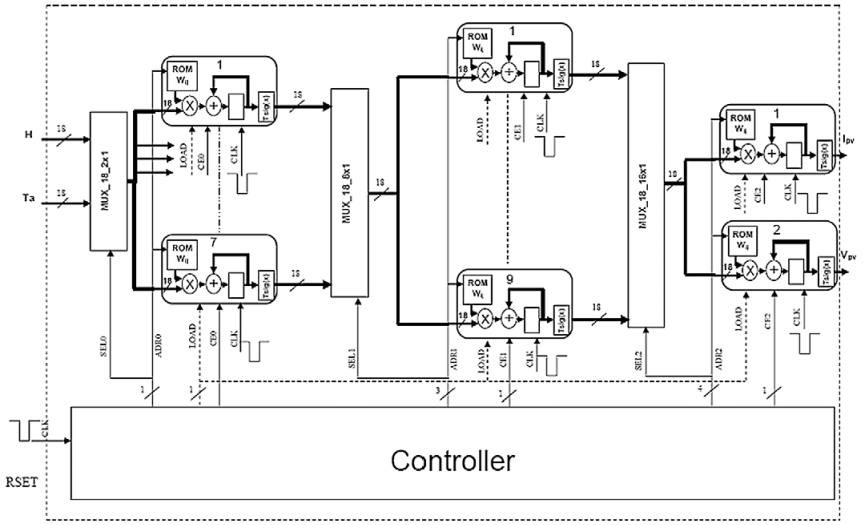

Figure 4.8: Version 2.0: P-V generator hardware design [2]

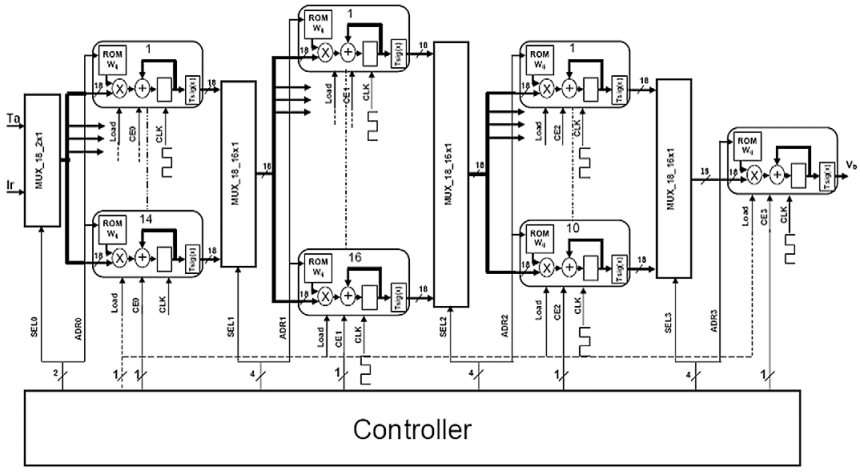

Figure 4.9: Version 2.0: P-V battery charging hardware design [2]

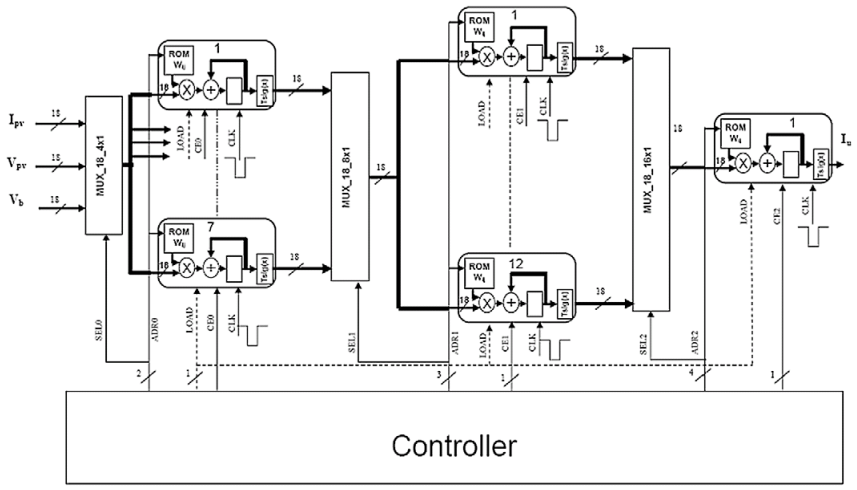

Figure 4.10: Version 2.0: P-V regulator hardware design [2]

for evaluation purposes.

2. Input Data : All the software, LFHD, and GFHD versions (version 1.0, 2.0, and 3.0) will be tested using the data observed by Florida Solar Energy Center (FSEC). The NN Matlab model will be trained using data from 2010 to 2011 (12 months). The NN hardware will be tested using 2010 to 2011 (6 months) data.

3. Activation Function : The activation function used in all the NN models discussed above are sigmoid activation function. The NN models described in the literature uses a table look-up method to compute the sigmoid function. The same sigmoid activation function will be used in the LFHD and GFHD.

4. Computation Precision : The NN models discussed above uses 18-bit fixed-point for computation. The LFHD and GFHD are also built using 18-bit fixed-point hardware cores

4.1.2.2 Electromagnetic Analysis using Finite-Difference Time Domain

Computational electrodynamics (electromagnetic wave analysis) is considered one of the important scientific application domains. It is used in many areas of research, such as radio frequency (RF) analysis in printed circuit board (PCB) analysis design, wave propagation in antennas, and microstrip discontinuities analysis. Electromagnetic wave analysis is performed by solving Maxwell's equations. These equations are partial differential equations that govern the propagation of electromagnetic waves. They are discretized over a finite volume, and the derivatives are approximated using central difference approximations. These finite-difference equations are then solved in a leap-frog manner to compute the electric and magnetic fields (E and H, respectively) in the finite-difference time domain (FDTD) method [17]. The performance of the FDTD application is measured as the rate at which the E and H are computed over a finite volume.

In the late 1960's, FDTD computation was first carried out on an unbounded boundary. In 1981, Mur [66] introduced highly absorbing boundary conditions (ABCs) to simulate the unbounded boundary. This boundary condition was achieved by truncating the mesh and using absorbing boundary conditions at its artificial boundaries to simulate the unbounded surroundings. In 1995-96, Sacks et al. [67] and Gedney et al. [68] introduced Uniaxial Perfectly Matched Layer (UPML) boundary conditions. The evaluation of the guidelines can be performed by building a basic version (version 1.0) of 2D-FDTD as discussed in [17]. In order to create version 2.0 and version 3.0, Mur's second order and UPML boundary conditions will be introduced to the version 1.0 model. Figure 4.11 shows the versions of 2D-FDTD hardware design. These three models are then used to evaluate the set of design guidelines. The following sub-sections explain all the versions.

The version 1.0 FDTD model is taken from the work conducted by Yee [17] in 1966. The discussed model is an 2D-FDTD model used to compute electromagnetic wave

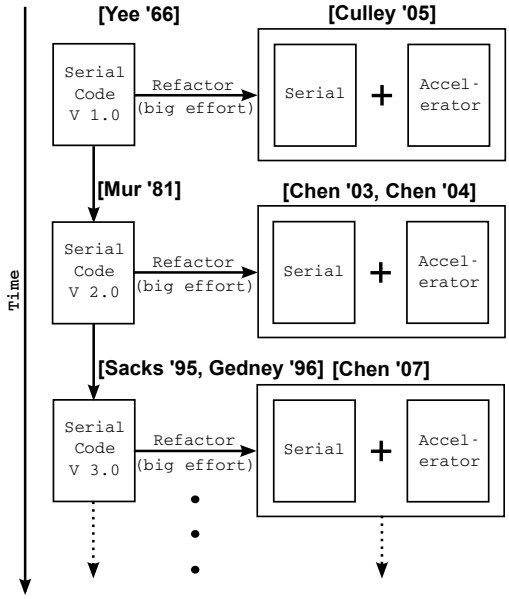

Figure 4.11: 2D-FDTD hardware design versions

properties of materials. The software version of this model will be built and tested using 'C'. The hardware version of this model will be built referring to the design in [3, 69]. The block diagram of design is shown in Figure 4.12. The design was built on a Virtex-II Pro FPGA (XC2VP50-7). A total of nine compute engines were built. The design was clocked at 100 MHz. The computation was performed using single precision floating-point cores. A single iteration for a grid size of 1003×1012 took 2.4 ms for 'E' computation and 2.38 ms for 'H' computation. The hardware details are tabulated in Table 4.4. For the purpose of evaluation, the hardware designs (LFHD and GFHD) will be built on a Virtex-4 FX60 FPGA using single precision floating-point hardware cores.

The version 2.0 FDTD model is a modification of version 1.0 FDTD model with Mur's second order boundary conditions as discussed in [66]. The software version of this model will be built and tested using 'C'. The hardware version of this model will be built referring to the design in [4, 70]. The block diagram of design is shown in Figure 4.13, 4.14, and 4.15. The design was built on a Xilinx Virtex-E XCV2000E

Table 4.4: Hardware design details for version 1.0 electromagnetic application

Details	**Values**
FPGA	Virtex-II Pro FPGA (XC2VP50-7)
Resources used	59% Slices, 7% Multiplier blocks, and 32% BRAM blocks
Frequency of operation	100 MHz
Performance	1003×1012 grid (2.4 ms for 'E' and 2.38 ms for 'H' computation)
Computation Precision	Single precision floating-point
Input Excitation	Delta pulse
Test Cases	Free Space
Boundary Conditions	Perfect Electrical Conductive (PEC) boundary conditions

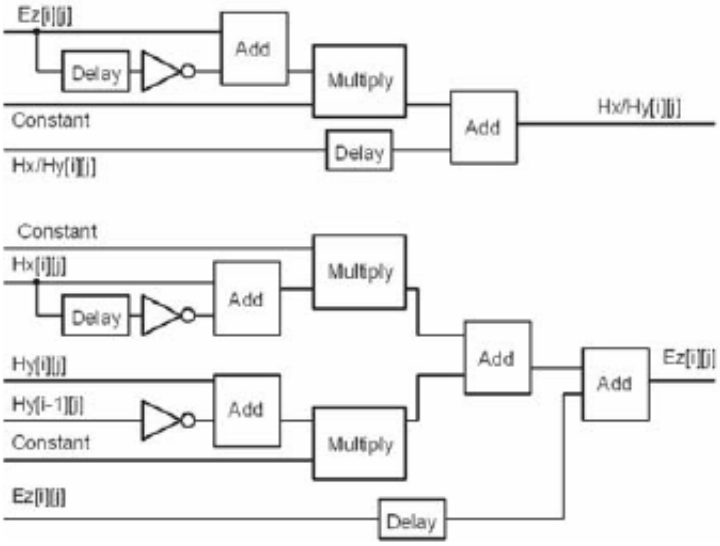

Figure 4.12: Version 1.0: 2D-FDTD hardware design [3]

Table 4.5: Hardware design details for version 2.0 electromagnetic application

Details	**Values**
FPGA	Virtex-E XCV2000E FPGA
Resources used	46% of slices and 54% of BRAM blocks
Frequency of operation	70 MHz
Performance	100×100 took 0.145 seconds for 200 iterations
Computation Precision	26-bits after binary point and 3-bits for integer part
Input Excitation	Electromagnetic wave from ground penetrating radar
Test Cases	Free Space
Boundary Conditions	Second-order Mur boundary conditions

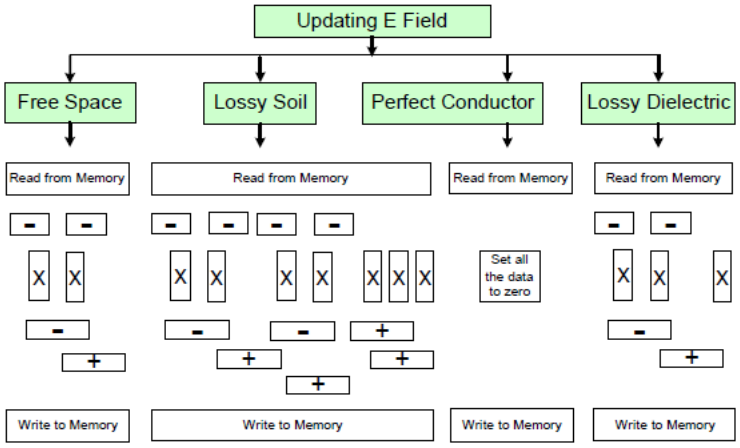

Figure 4.13: Version 2.0: 2D-FDTD 'E' field updating hardware design [4]

FPGA and consumed 46% of slices and 54% of BRAM blocks. A total of 227 compute engines were built. The design was clocked at 70 MHz. The computation was performed using fixed-point (26-bits after binary point and 3-bits for integer part) cores. A grid size of 100×100 took 0.145 seconds for 200 iterations. The hardware details are tabulated in Table 4.5. For the purpose of evaluation, the hardware designs (LFHD and GFHD) will be built on a Virtex-4 FX60 FPGA using single precision floating-point hardware cores.

The version 3.0 FDTD model is a modification of version 1.0 FDTD model with UPML boundary conditions as discussed by Sacks et al. [67] and Gedney et al. [68]. The software version of this model will be built and tested using 'C'. The hardware

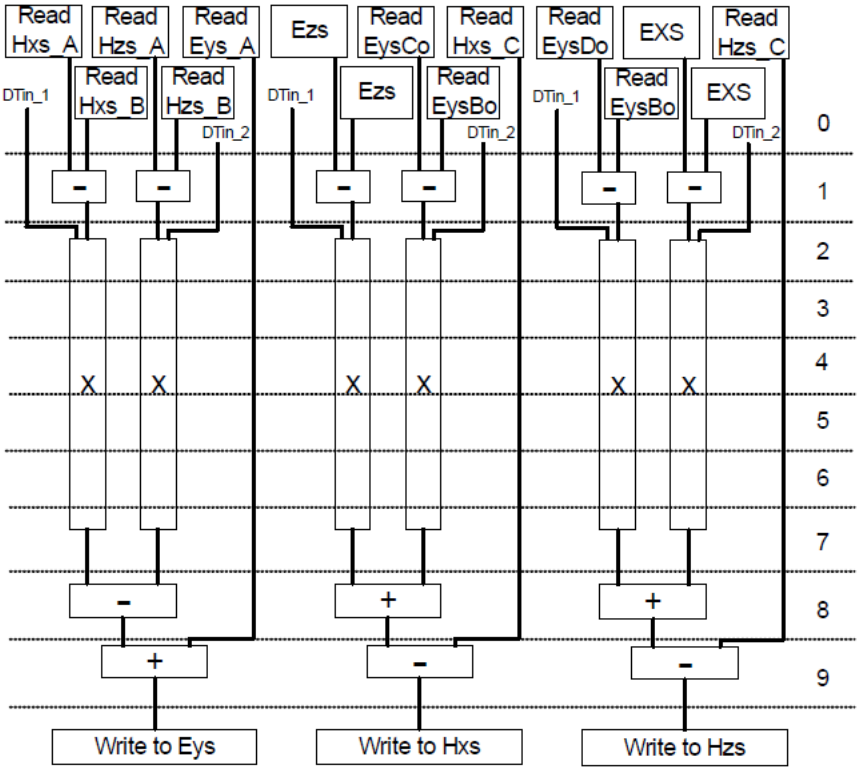

Figure 4.14: Version 2.0: 2D-FDTD 'H' field updating hardware design [4]

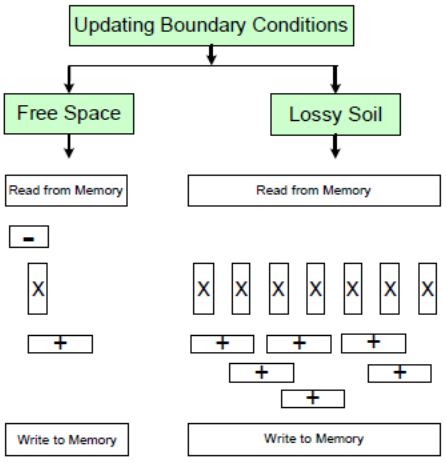

Figure 4.15: Version 2.0: 2D-FDTD boundary updating hardware design [4]

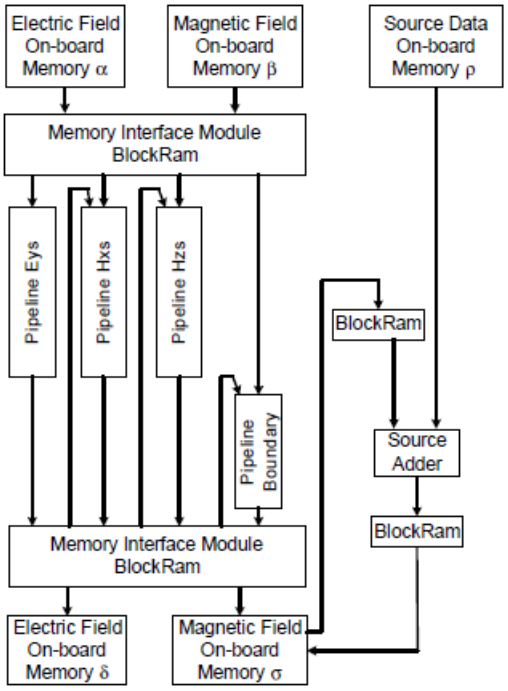

Figure 4.16: Version 2.0: 2D-FDTD overall hardware design [4]

version of this model will be built referring to the design in [5]. The block diagram of design is shown in Figure 4.17, 4.18, and 4.19. The design was built on a Xilinx Virtex II-Pro FPGA. The design is operated at 120 MHz. The computation was performed using fixed-point (33-bit field and 2-bit coefficient values) cores. A grid size of 481×481 took 13.1 seconds for 3,026 iterations. The hardware details are tabulated in Table 4.6. For the purpose of evaluation, the hardware designs (LFHD and GFHD) will be built on a Virtex-4 FX60 FPGA.

Design Constraints for Evaluation :

1. FPGA : As hardware designs cannot built using older FPGA parts, all the hardware designs (LFHD and GFHD) will be built on a Virtex-4 FX60 FPGA for evaluation purposes.

2. Test Material: All the software, LFHD, and GFHD versions (version 1.0, 2.0, 3.0, and 4.0) will be tested using a standard material of a fixed size, whose

Table 4.6: Hardware design details for version 3.0 electromagnetic application

Details	**Values**
FPGA	Xilinx Virtex II-Pro FPGA
Resources used	47% of Slices, 28% Multipliers, and 50% BRAM blocks
Frequency of operation	120 MHz
Performance	481×481 took 13.1 seconds for 3,026 iterations, 53.4 MNodes/second
Computation Precision	33-bits after binary point and 2-bits for integer part
Input Excitation	Electromagnetic wave from ground penetrating radar
Test Cases	Free Space
Boundary Conditions	UPML boundary conditions

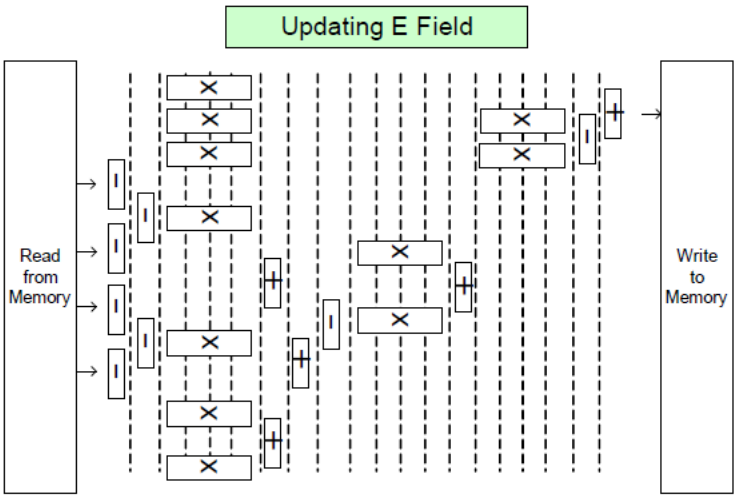

Figure 4.17: Version 3.0: 2D-FDTD UPML hardware design [5]

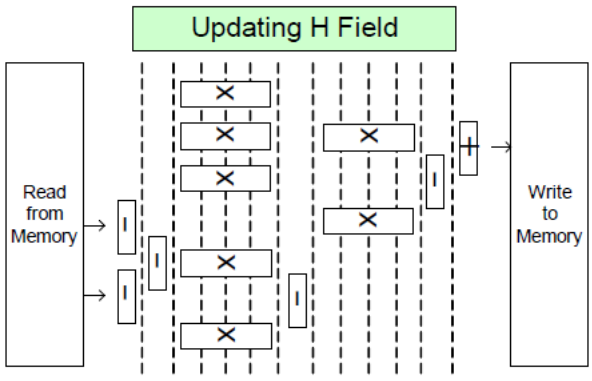

Figure 4.18: Version 3.0: 2D-FDTD UPML hardware design [5]

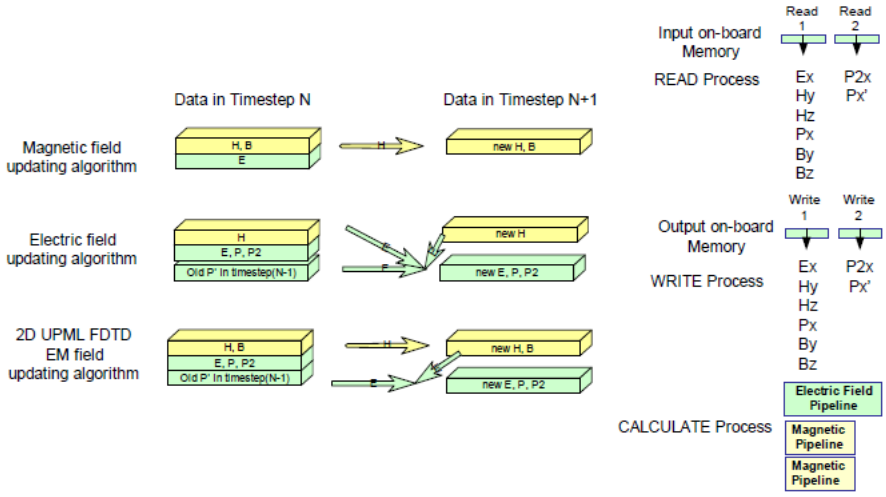

Figure 4.19: Version 3.0: 2D-FDTD UPML overall hardware design [5]

parameters will be stored in off-chip memory.

3. Input Excitation : The input excitation wave will be stored as look-up table values and will be loaded at each computation cycle.

4.1.2.3 Sparse Matrix-Vector Multiplication (SpMV)

Sparse matrix vector multiply (SpMV) routine does a matrix-vector multiplication of sparsely filled matrix, and the complexity of the routine is $O(n^2)$. SpMV routine is an important routine in many scientific applications that deals with matrix computations. As better computation methods are discovered, computational scientists implement those methods to reduce the execution time of their applications. In order to evaluate the design guidelines, three versions (version 1.0, 2.0, and 3.0) of SpMV is taken from the literature. Each version reduces the computation latency and improves throughput. Figure 4.20 shows the versions of SpMV hardware design. These three models are then used to evaluate the set of design guidelines. The following sub-sections explain all the versions.

The version 1.0 design computes sparse matrix vector product as described by Zhuo et al. [6]. The work discusses about a high throughput sparse matrix vector

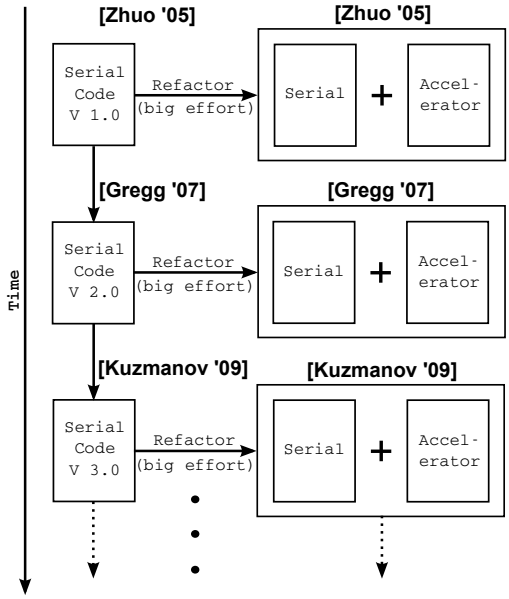

Figure 4.20: Sparse Matrix-Vector multiplication hardware design versions

Table 4.7: Hardware design details for version 1.0 SpMV multiply unit

Details	**Values**
FPGA	Xilinx Virtex-II Pro XC2VP70
Resources used	16,613 Slices
Frequency of operation	160 MHz
Performance	350 MFLOPs at 8.0 GB/s

multiply unit. The software version of the SpMV routine will be built and tested using 'C'. The hardware version of the SpMV will be built referring to the design in [6]. The block diagram of design is shown in Figure 4.21 and 4.22. The design was built on a Xilinx Virtex-II Pro XC2VP70. The design was clocked at 160 MHz. The design uses University of Florida sparse matrix collection for testing. A sustainable performance of 350 MFLOPs was observed. The hardware details are tabulated in Table 4.7. For the purpose of evaluation, the hardware designs (LFHD and GFHD) will be built on a Virtex-4 FX60 FPGA.

The version 2.0 design computes sparse matrix vector product as described by Sun et al. [7]. The work discusses about a high throughput sparse matrix vector multiply unit. The software version of the SpMV routine will be built and tested using 'C'.

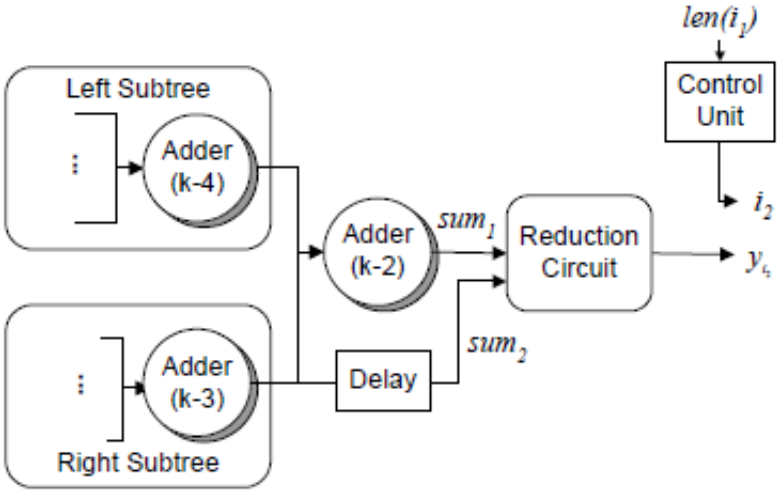

Figure 4.21: Version 1.0: SpMV multiply hardware [6]

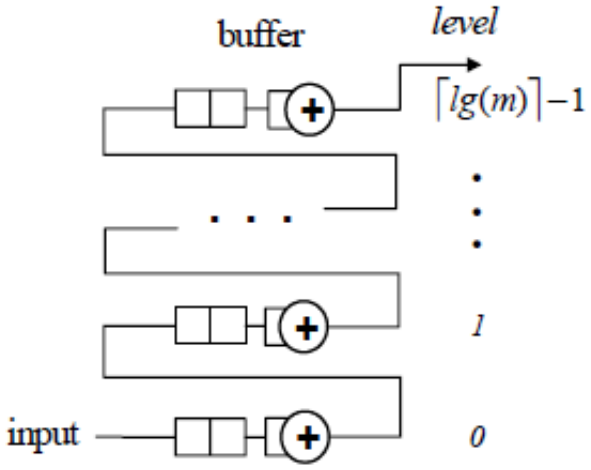

Figure 4.22: Version 1.0: SpMV reduction hardware [6]

Table 4.8: Hardware design details for version 2.0 SpMV multiply unit

Details	**Values**
FPGA	Xilinx Virtex-II X2CV6000-6
Resources used	10,050 Slices, 11 BRAM Blocks, and 12 multipliers
Frequency of operation	95 MHz
Performance	118.77 MFLOPs

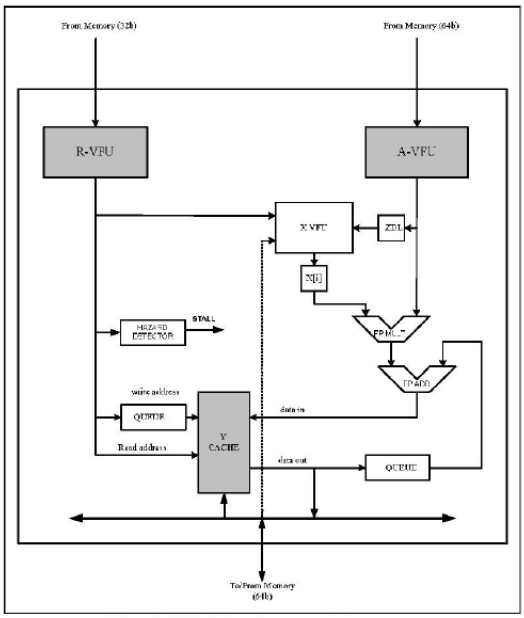

Figure 4.23: Version 2.0: SpMV multiply hardware [7]

The hardware version of the SpMV will be built referring to the design in [7]. The block diagram of design is shown in Figure 4.23, 4.24. The design was built on a Xilinx Virtex-II X2CV6000-6. The design was clocked at 95 MHz. The design uses University of Florida sparse matrix collection for testing. A sustainable performance of 118.77 MFLOPs was observed. The hardware details are tabulated in Table 4.8. For the purpose of evaluation, the hardware designs (LFHD and GFHD) will be built on a Virtex-4 FX60 FPGA.

The version 3.0 design computes sparse matrix vector product as described by Kuzmanov et al. [8]. The work discusses about a high throughput sparse matrix vector multiply unit. The software version of the SpMV routine will be built and tested

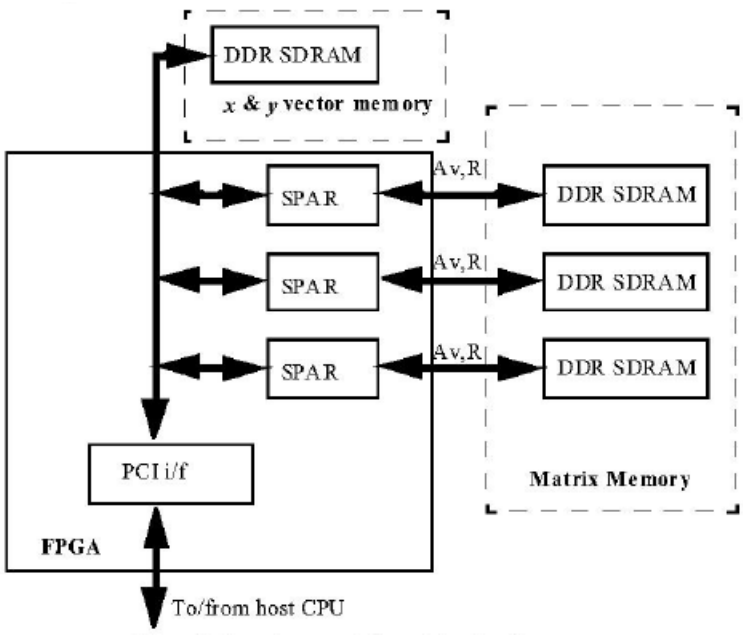

Figure 4.24: Version 2.0: SpMV reduction hardware [7]

Table 4.9: Hardware design details for version 3.0 SpMV multiply unit

Details	**Values**
FPGA	Xilinx Virtex-4 LX200
Resources used	22,700 Slices
Frequency of operation	100 MHz
Performance	1104 - 1571 MFLOPs at 8.0 GB/s

using 'C'. The hardware version of the SpMV will be built referring to the design in [8]. The block diagram of design is shown in Figure 4.25, 4.26. The design was built on a Xilinx Virtex-4 LX200. The design was clocked at 100 MHz. The design uses University of Florida sparse matrix collection for testing. A sustainable performance of 1104 - 1571 MFLOPs was observed. The hardware details are tabulated in Table 4.9. For the purpose of evaluation, the hardware designs (LFHD and GFHD) will be built on a Virtex-4 FX60 FPGA.

Design Constraints for Evaluation :

1. FPGA : As hardware designs cannot built using older FPGA parts, all the hardware designs (LFHD and GFHD) will be built on a Virtex-4 FX60 FPGA

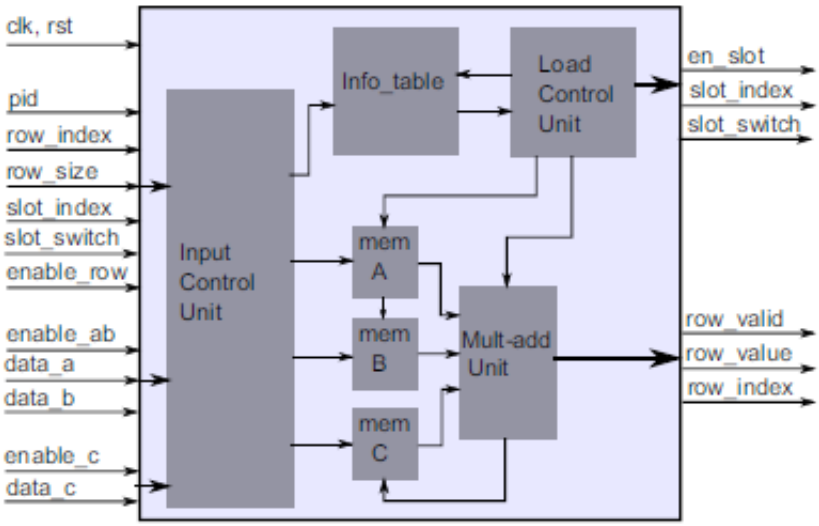

Figure 4.25: Version 3.0: SpMV multiply hardware [8]

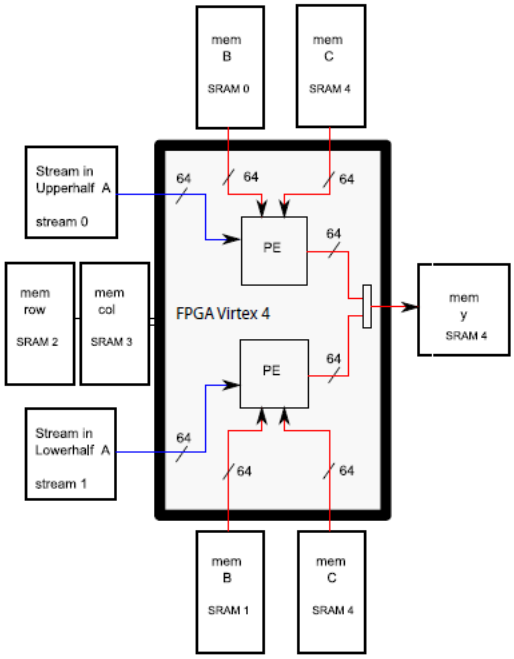

Figure 4.26: Version 3.0: SpMV overall hardware [8]

for evaluation purposes.

2. Test Data : University of Florida sparse matrix collection will be used for testing.

3. Computation Precision : All the computations will be done in double precision floating-point.

4.2 Communicability of the Design Guidelines

This section answers the question of whether these guidelines can be easily communicable. Table 4.1 describes the set of design guidelines. As seen, these design guidelines can be easily followed and understood by computer engineers. Each design guideline is self explanatory, and, by following the design guideline, one could achieve a hardware design for a living scientific application.

4.3 Broad Applicability of the Design Guidelines

In order to answer the broad applicability of the design guidelines, six applications are chosen from the literature, and the design guidelines are validated on the applications to understand the applicability of the design guidelines. A guideline fitness plot is plotted for each application to understand the relation between the performance and the number of design guidelines followed. The six applications that are chosen from the literature are:

- Computational Fluid Dynamics
- Computational Molecular Dynamics
- Quantum Monte Carlo Simulations
- Hessenberg Reduction
- Gaxpy - BLAS Routine
- N-Body Simulations

4.3.1 Guideline Fitness Plot

In order to find out the applicability of the design guidelines, a guideline fitness plot is introduced. This plot is shown in Figure 4.27. The x-axis represents the design guidelines followed, and the y-axis represents the performance of the application. A $+1$ value on the x-axis denotes that all the guidelines have been followed, and a -1 shows none of them have been followed. A $+1$ on the y-axis shows the anticipated performance is equal to the peak performance, and -1 on the y-axis shows no performance. A value of zero on the y-axis specifies 50% of the anticipated or the theoretical performance.

The guideline fitness plot has four quadrants. The first quadrant is the area between $+x$ axis and $+y$ axis, the second quadrant is between $-x$ axis and $+y$ axis, the third quadrant is between $-x$ axis and $-y$ axis, and the fourth quadrant is between $-y$ axis and $+x$ axis. If the fitness of an application falls in the first quadrant, then the application at least follows 50% of the design guidelines and has 50% of the anticipated or theoretical performance at a particular operational frequency or a better match to a good design. If the fitness of the application falls in the third quadrant, then less than 50% of the guidelines are followed, and the performance is less than 50% of the anticipated or theoretical performance. If the application fitness falls in the second quadrant, then less than 50% of the guidelines were followed, and still the application has poor performance. If the application fitness falls in the fourth quadrant, then more than 50% of the guidelines were followed, and the performance was poor or did not match the theoretical design. The next section will discuss six applications from the literature that will used to study the broad applicability of the design guidelines. An example application (computational fluid dynamics) is chosen from the six applications, and its evaluation is performed using the design guidelines for studying broad applicability. The resultant values are plotted on the guideline fitness plot.

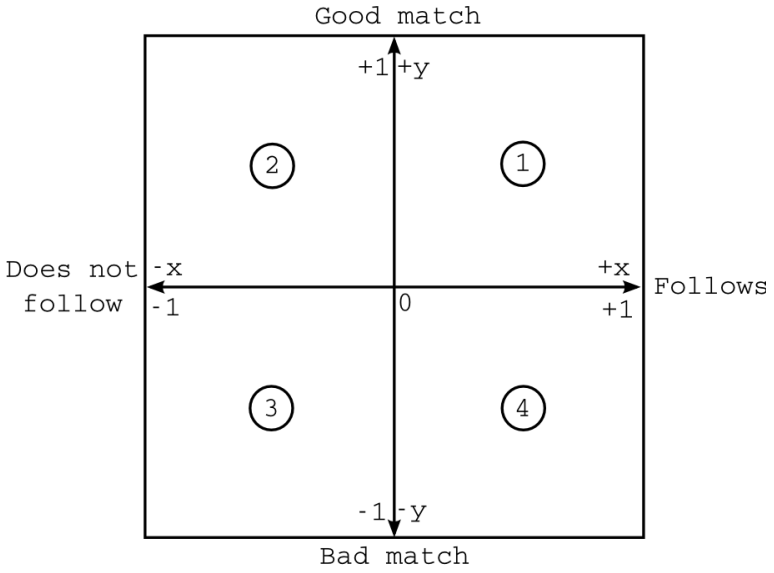

Figure 4.27: Guideline fitness plot

4.3.2 Computational Fluid Dynamics

Computational fluid dynamics (CFD) simulation is a numerical method to solve problems involving fluid flows on discrete space and time. The problem space is discretized into smaller regions to form a grid. The equations governing the fluid flow is solved on the grid for discrete time steps [71]. The work presented in [9] uses a flow-solver based on the fractional step method with finite difference schemes. In this method, a tentative velocity is computed, and then the kinematic viscosity of the fluid is computed using the governing equations. A systolic array approach is used for the hardware design, shown in Fig. 4.28. The governing equations are evaluated using the values from all the directions (north, east, west, and south). The design is tested for a 24 $\times$ 24 grid, which is a small dataset for testing the robustness of the design. The dataset is stored locally and is not streamed to use the maximum memory bandwidth.

The evaluation is carried out by a rubric. Every design guideline that is followed is given a score of +1, and those that are not followed is given a score of -1. The evaluated design guideline values are added to get the total score. These values are

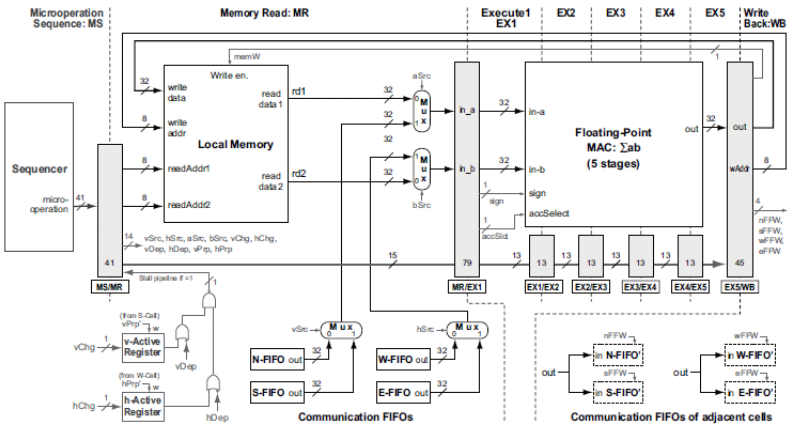

Figure 4.28: Computational fluid dynamics design [9]

used to plot the guideline fitness plot. The CFD design is evaluated to find out whether or not the design guidelines have been followed. Table 4.10 shows the design guidelines that have been followed. The guideline fitness plot is used to evaluate the application. After the evaluation, and from Table 4.10, the total score is 4. The scaled score is 4/12=0.33. The scaled score is plotted on the x-axis. The y-axis shows the performance. The work referred here uses a single Altera Stratix II FPGA of the DN7000k10PCI board. The size of the implemented systolic array is 12×8 cells. The implemented design consumes ≈50% logic cells and all of the embedded multipliers. Every cell has a multiplier accumulate unit (MACC), which can operate at 90 MHz but when put together can only operate at 60 MHz. The MACC has five stages, however, if these stages are increased further, a higher frequency can be achieved. Assuming the theoretical frequency to be 90 MHz, the performance is 0.67, and since the MACC unit is 98% utilized, the overall performance is 0.67×0.98=0.656. The scaled performance index (0.656*2)-1=0.31 is plotted on the y-axis of the guideline fitness plot, shown in Figure 4.29.

4.3.3 Computational Molecular Dynamics

Molecular dynamics (MD) simulation is a study of movements of atoms and molecules. This study is divided into two-body and three-body interactions [72].

Table 4.10: CFD design evaluated using the design guidelines

1	*Arrange and optimize input/output data for all compute blocks*	✔
2	*Adhere to a widely accepted method of computation for arithmetic functions/operations*	✔
3	*Build controller for every not likely to change compute blocks*	✔
4	*Introduce a configurable dataflow path to connect resources of likely to change compute blocks*	✘
5	*Introduce on-chip memory with configurable read/write logic, if necessary*	✔
6	*Introduce dependency indicators to enhance parallel computations, if necessary*	✔
7	*Use microcode to specify dataflow path, memory read/write, and parallel computations*	✘
8	*Maximize resource utilization by improving runtime parallelism*	✔
9	*Forward results between compute blocks/resources, if possible*	✔
10	*Achieve functionality, and optimize design & resources to improve performance*	✔
11	*Maximize computation until maximum memory bandwidth is utilized*	✘
12	*Use a large and real dataset for test cases*	✘

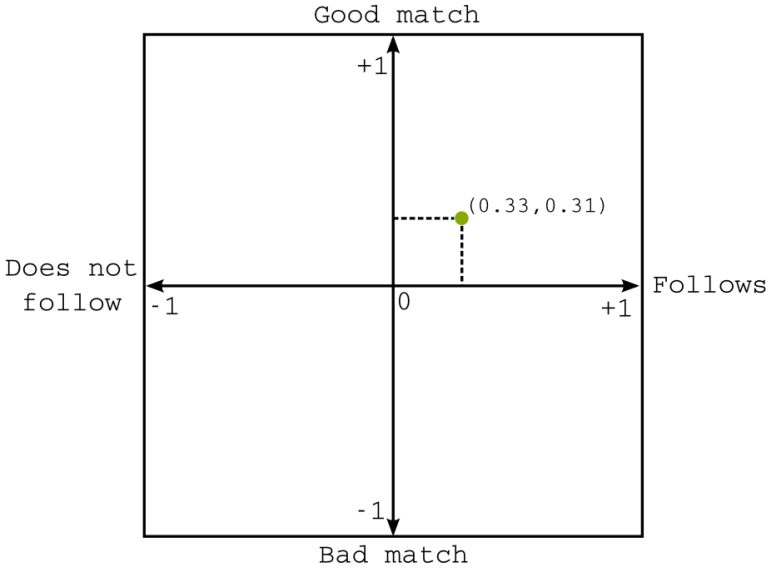

Figure 4.29: Guideline fitness plot for CFD application

In two-body MD, the simulation is based on the distance and force computation of two atoms. In three-body MD, the simulation is based on the distance and force computation of three atoms. A fully implemented two-body MD hardware is presented by Chiu et al. [10]. The hardware design is shown in Figure 4.30. The hardware design computes two-body force in reduced precision.

4.3.4 Quantum Monte Carlo Simulations

Quantum Monte Carlo (QMC) simulations studies the structural and energetic properties of a group of atoms or molecules. There are two types of QMC simulations: diffusion Monte Carlo (DMC) and variational Monte Carlo (VMC) simulations [11]. These methods are useful for studying the ground-state wave functions, local energies, and other ground-state properties of quantum many-body systems. DMC is a technique for numerically solving the many-body Schrodinger equation. The VMC method employs a set of adjustable parameters to yield a trial wave function that approximates the exact wave function. The VMC method is simpler and faster than the DMC method, but less accurate. The work described in [11] uses the VMC method to perform Monte Carlo simulations. In this method, a reference configuration is chosen, and a random displacement is added to the reference configuration. The energy and wave function of the new configuration is computed. The final step is to accept, or reject, the current configuration using the ratio of wave functions. The two most important kernels of the application is potential energy computation and wave function calculation. The pipeline of distance computation and wave function computation is shown in Figure 4.31. The design compromises the accuracy of the results by using fixed precision. Any changes in the computational algorithm would demand a complete redesign.

4.3.5 Hessenberg Reduction

Hessenberg reduction (HR) reduces a square matrix in to an upper or lower Hessenberg matrix. A upper Hessenberg matrix is a matrix with zero entries below the

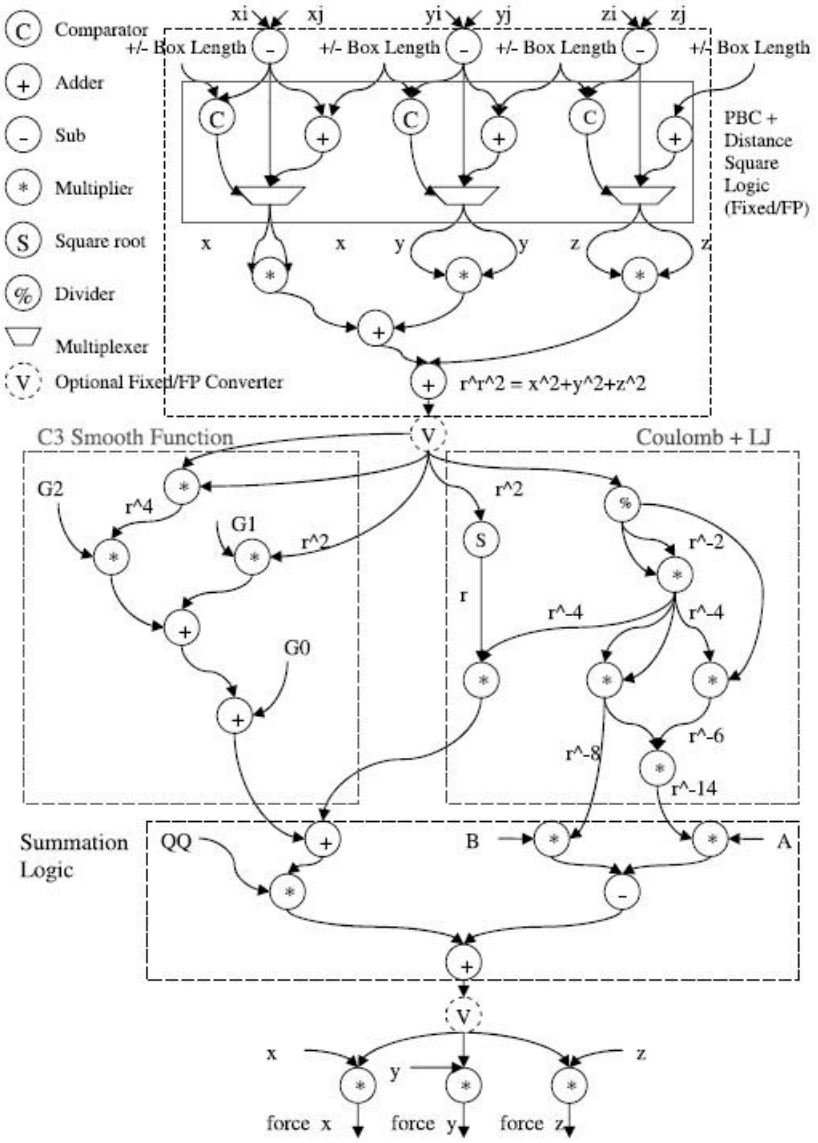

Figure 4.30: Molecular dynamics design [10]

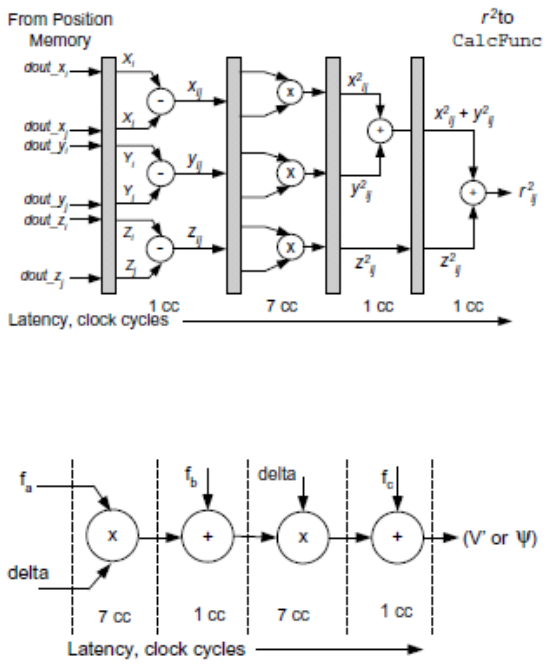

Figure 4.31: Quantum Monte Carlo simulation design [11]

sub-diagonal matrix, and a lower Hessenberg matrix is a matrix with zero entries above the sub-diagonal matrix. HR is a major step involved in finding the eigen values of a matrix [73]. This is an important reduction used in many of the high performance computing applications. A hardware for HR is discussed in [12]. The work is demonstrated on SGIs Altix RASC RC100 reconfigurable computer with Xilinx Virtex-4LX200 FPGA. The hardware is shown in Figure 4.32. One of the major design flaws of the hardware is that the dataset is stored on the local memory and not on the main memory (DDR). This would not enable the design to scale for larger matrices. The computation is done in a sequential fashion. Any improvements in the algorithm can not be accommodated unless the hardware is redesigned.

4.3.6 Gaxpy - BLAS Routine

Gaxpy routine is a BLAS level 2 routine. The routine does matrix-vector multiplication, and the complexity of the routine is $O(n^2)$ [15]. A Gaxpy hardware built

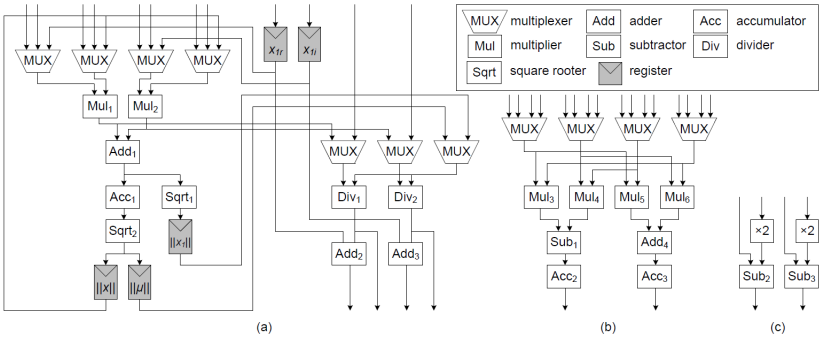

Figure 4.32: Hessenberg reduction design [12]

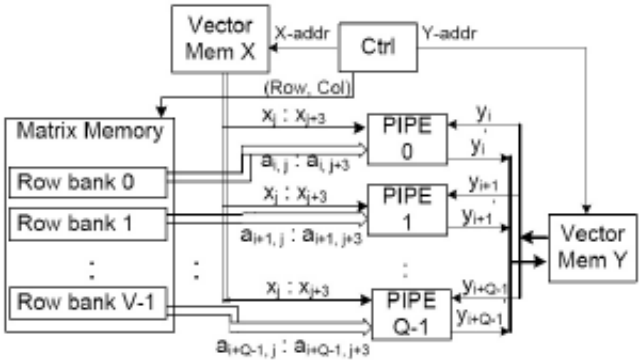

Figure 4.33: Gaxpy Routine Design [13]

on the FPGA is discussed in [13]. The hardware design is shown in Figure 4.33. The work was demonstrated on a BEE3 FPGA board that has four V5 LX155T FPGAs. The implementation has 16 processing elements, each computing a 4×4 matrix. The maximum size of the matrix that can be stored on the on-chip memory is 256×256. The peak theoretical performance at 100 MHz is 200 MFLOPs. There are 16 PEs per FPGA, and with four FPGAs, the peak theoretical performance is 12.8 GFLOPs, and the reported performance is 3.113 GFLOPs.

4.3.7 N-Body Simulations

N-Body simulation has been used by computer scientist to study the interaction of atoms and molecules for past several decades. In these studies, the atoms and

molecules are allowed to interact for a given period of time and the intermolecular forces are computed. The atomic interactions which are governed by the basic laws of physics are discretized and simulated using a computer program. These simulations help scientists understand the behavior of proteins, bio-molecules, and other materials that often cannot be observed directly. A N-body computation involves two major computations. The first computation is the inter-atomic distances between three atoms, followed by inter-atomic force computations for those atoms whose inter-atomic distances are less than the cut-off distance. Since the above computations (distance and force) involve three atoms, i.e, every atom is compared with every two other atoms, the distance and force computations are performed in a triply nested loop. For instance, a system with 'n' atoms has a computational complexity of $O(n^3)$. A N-Body hardware is built on the FPGA and is discussed in [14]. The hardware design is shown in Figure 4.34.

Once the evaluation for broad applicability is performed for the above six applications, a combined fitness plot will be plotted by combining the fitness plot for all applications. The next section discusses the validation of the results from the evaluation.

4.4 Validation

To validate this work, we must answer whether the designers can effectively accommodate the evolving changes in the living computational science application and achieve better performance and productivity by using the set of design guidelines. We must also answer the communicability and broadly applicability of the design guidelines. To answer the thesis question, we will use the experimental results from the evaluation metrics to further prove or disprove our solution. To be very specific, we will answer the following questions for each application:

- Does the performance data points of GFHD and LFHD track each other?
- Does the resource utilization data points of GFHD and LFHD track each other?

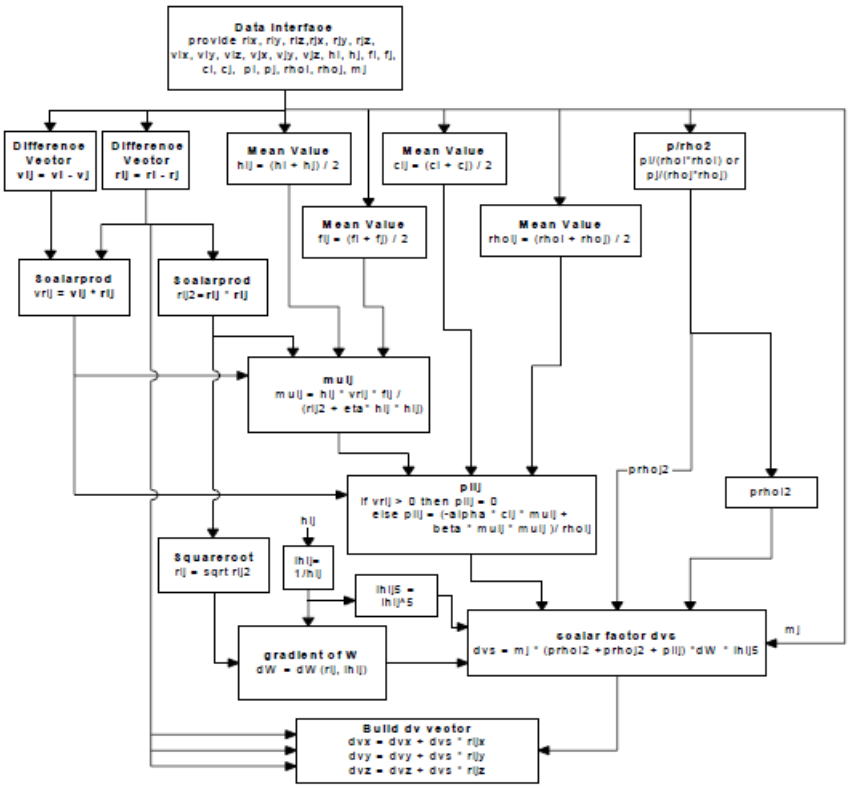

Figure 4.34: N-body hardware design [14]

- Does the lines of code changed data points of GFHD and LFHD diverge?
- Is the degree of effectiveness in terms of performance classified as good or excellent?
- Is the degree of effectiveness in terms of resources utilization classified as good or excellent?
- Is the degree of effectiveness in terms of lines of code changed classified as good or excellent?
- Is the design guideline easily communicable?
- Is the design guideline broadly applicable for many applications?

If the performance and resource utilization data points for GFHD and LFHD track each other, and the lines of code diverge for GFHD and LFHD, then the degree of effectiveness is further used to affirm or deny the thesis question. Example of such a scenario is shown in Figure 4.35, 4.36, and 4.37 using fictitious data. On the contrary, if the performance and/or resource utilization data points do not track, and/or the lines of code data points do not diverge, then we can deny the thesis question.

If results from the evaluation for all applications show an excellent or good response for performance, resource, and lines of code comparison, then we can strongly affirm the solution to thesis question. On the other hand, if the classifications for all applications generate a bad response for resources and an good, or excellent response for performance and lines of code changed for all applications, then we can conclude that the set of design guidelines helps hardware designers to design hardware for living scientific codes, but it requires more resources. If the degree of effectiveness for comparison of lines of code changed is bad for any application, then we can deny the thesis question.

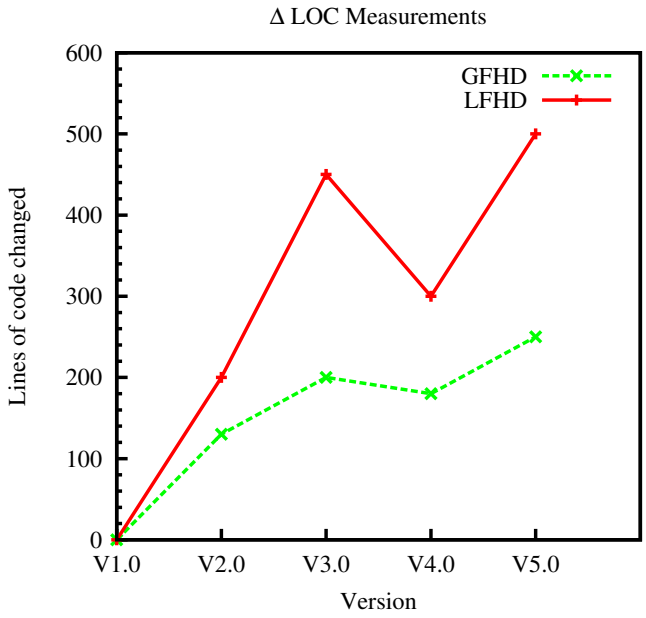

Figure 4.35: ΔLOC measurements for LFHD and GFHD (fictitious data)

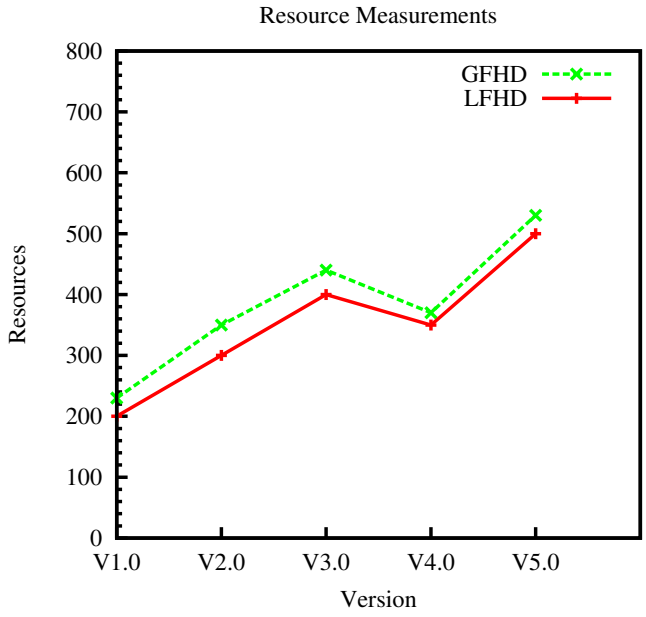

Figure 4.36: Resource utilization for LFHD and GFHD (fictitious data)

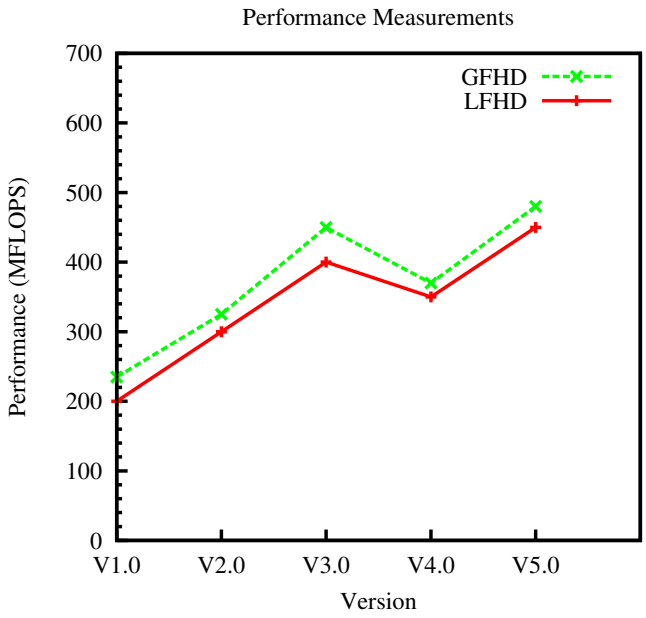

Figure 4.37: Performance measurements for LFHD and GFHD (fictitious data)

If the results from evaluation confirms easy communicability of the design guidelines, and broad applicability to various applications, then we can affirm that there exists a design guideline that is easily communicable and is broadly applicable.

CHAPTER 5: RESULTS

This section discusses the results obtained from effectiveness and broad applicability evaluation. The first section discusses the results from effectiveness evaluation followed by broad applicability evaluation.

5.1 Effectiveness of design guidelines

The evaluation of photo-voltaic (P-V) modeling using Neural Networks, electromagnetic wave analysis using finite difference time domain (FDTD), and the computational science kernel sparse matrix-vector (SpMV) multiplication operation is discussed in this section. Observations are made from the results and is presented in this section. The following sections present the results of evaluation for each of the application.

5.1.1 P-V System Modeling using Neural Networks (NN)

The P-V system modeling using neural networks helps in predicting the amount of electrical energy that could be made available from sunlight. This application takes solar irradiation and ambient temperature to predict voltage and current of the P-V system. In order to evaluate the design guidelines, two versions of this application are built using software, literature, and using the design guidelines. The following sections discusses about the designs.

5.1.1.1 Software Design

The software designs (version 1.0 and 2.0) were built using Matlab following the guidelines presented in [65, 1] (version 1.0), and [2] (version 2.0). The neural network based design was trained and evaluated using the data observed at Florida Solar Energy Center (FSEC). The center provides data from July 18, 2010 to the present day. The data is downloaded and divided into training set and evaluation set. The

training set consists of 12 months of data and the evaluation set consists of 6 months of data. The authors of the literature design have used five and ten years of data for training and one year for evaluation. As the data used does not affect the evaluation parameters (performance, resources, and lines of code changed) we decided to use the most accurate available data for training and evaluation of the literature based designs. The inputs, weights, and the results of the software based neural networks are used for testing and validation of the results with the literature and guideline followed hardware design.

5.1.1.2 Literature Followed Hardware Design

The literature followed hardware design is built using the guidelines presented in [65, 1] (version 1.0), and [2] (version 2.0). The main component of the design is the neuron. The VHDL code of the neuron is presented in the appendix of the paper. The VHDL code is used to design the neuron followed by the design of the multi-layered perceptron (MLP). The design is built as closely as possible. For example, the Table 5.1 shows the synthesis report taken from the synthesis log. The report has 20 ROMs, 40 multipliers, and 3 multiplexers. The literature design has 20 ROMs, 40 multipliers and 3 multiplexers in total. This confirms that the design is built as close as possible. The performance of the literature based design depends on the largest number of inputs of any layer and the latency of the neuron. As the number of inputs for each layer is known, and the neuron design is available, the performance can be computed. Thus, the performance of the design is also preserved. Once the designs (version 1.0 and 2.0) are built, the lines of code added or changed, performance, and the resources used are tabulated in the Table 5.2.

5.1.1.3 Guideline Followed Hardware Design

The guideline followed hardware design is built using the design guidelines and the input-output characteristics of neural networks presented in [65, 1] (version 1.0), and [2] (version 2.0). The neuron component is redesigned following the design guidelines,

```
HDL Synthesis Report

Macro Statistics
# ROMs                                          : 20
 16x18-bit ROM                                  : 1
 4x18-bit ROM                                   : 7
 8x18-bit ROM                                   : 12
# Multipliers                                   : 40
 18x18-bit multiplier                           : 40
# Adders/Subtractors                            : 80
 18-bit adder                                   : 40
 36-bit adder                                   : 40
# Registers                                     : 56
 1-bit register                                 : 10
 18-bit register                                : 23
 2-bit register                                 : 1
 3-bit register                                 : 1
 36-bit register                                : 20
 4-bit register                                 : 1
# Latches                                       : 20
 18-bit latch                                   : 20
# Comparators                                   : 340
 18-bit comparator greatequal                   : 20
 18-bit comparator less                         : 320
# Multiplexers                                  : 3
 18-bit 16-to-1 multiplexer                     : 1
 18-bit 4-to-1 multiplexer                      : 1
 18-bit 8-to-1 multiplexer                      : 1
```

Table 5.1: HDL synthesis report for P-V regulator model

Table 5.2: Results for P-V Modeling Application

Measurements	**Version 1.0**			**Version 2.0**			**Effectiveness**
	LFHD	GFHD	(G-L)/L	LFHD	GFHD	(G-L)/G	Bad/Good/Xlnt
Lines of code							
-Delta LOC	0	0	N.A	437	233	-87.55%	Xlnt
Resource Utilization							
-Slices	4,806	5,026	4.57%	18,471	20,298	9.89%	Xlnt
-BRAM blocks	40	36	-10%	118	108	-8.47%	Xlnt
-DSP48 slices	36	36	0%	158	158	0%	Xlnt
Measurements	**Version 1.0**			**Version 2.0**			**Effectiveness**
	LFHD	GFHD	G/L	LFHD	GFHD	G/L	Bad/Good/Xlnt
Performance							
-Million decisions/sec	8.27	9.85	1.19	6.47	7.39	1.14	Xlnt

Xlnt - Excellent

and the redesigned neuron is used to build the MLP. Once the guideline followed designs (version 1.0 and 2.0) are built, the lines of code added or changed, performance, and the resources used are tabulated in the Table 5.2.

5.1.1.4 Results

The results of the designed hardware from literature and using the guidelines are presented in Table 5.2. Plots 5.1, 5.2, 5.3, and 5.4 are drawn for lines of code changed, resources, and performance respectively. Guidelines 1, 4, and 10 were used for designing the GFHD. The (G-L)/L or (GFHD-LFHD)/LFHD column represents the percentage increase or decrease in the lines of code and resources used for every version (version 1.0 and 2.0). The degree of effectiveness column presents the classification for degree of effectiveness. Similarly, for the performance, decisions computed per second is reported in Table 5.2. The (G/L) or (GFHD/LFHD) column presents the final performance factor between the literature followed design and guideline followed design for every version.

5.1.1.5 Observations

Plots are drawn for the lines of code changed, resource usage, and performance measurements, as shown in Figures 5.1, 5.2, 5.3, and 5.4. The plot for lines of code changed (Figure 5.1) show that the data points for GFHD and LFHD diverge. Similarly, the plots for performance and resources used (Figures 5.4, 5.2, and 5.3) show that data points of GFHD and LFHD for performance and resource utilization track

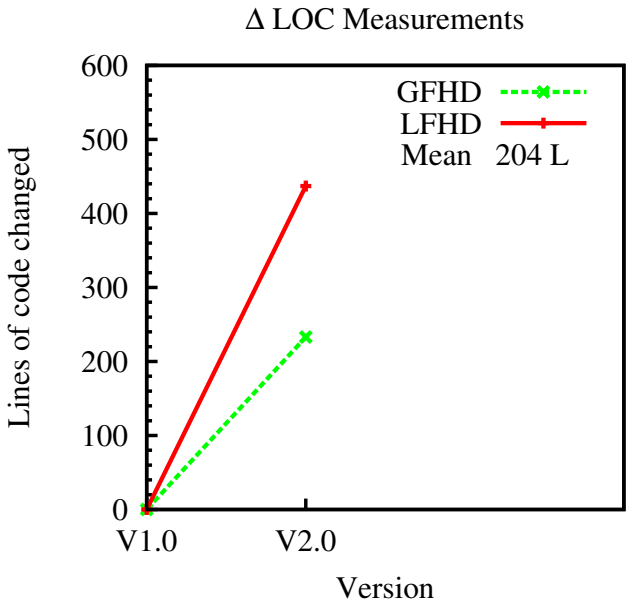

Figure 5.1: Plot comparing lines of code changed for P-V application

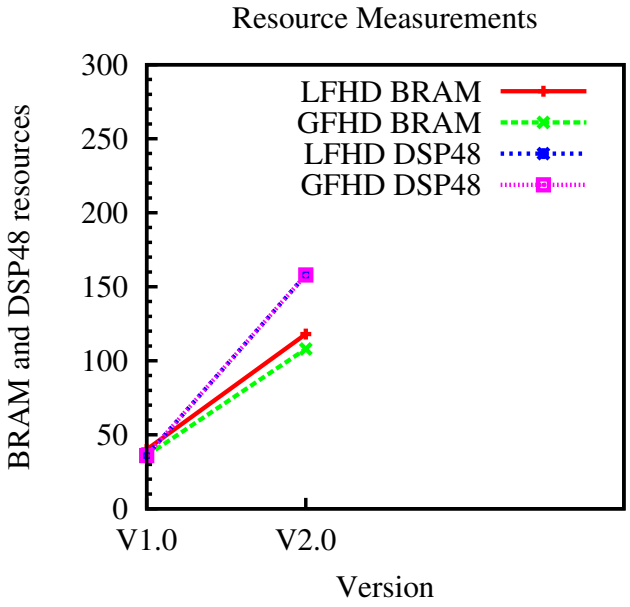

Figure 5.2: Plot comparing resource utilization for P-V application

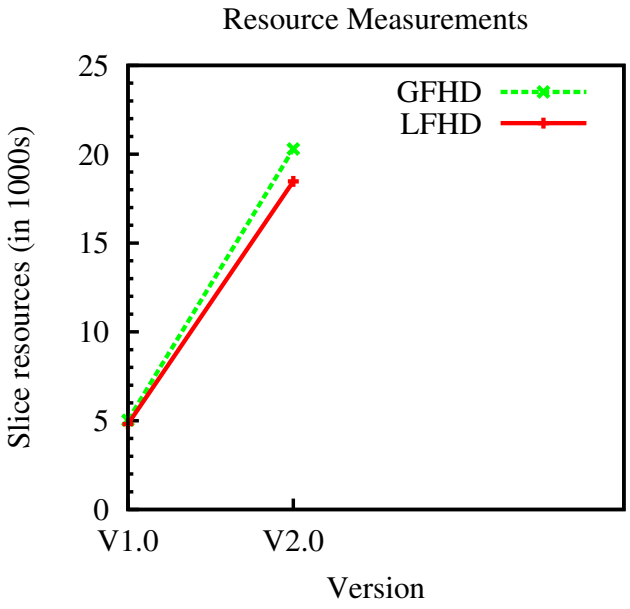

Figure 5.3: Plot comparing slice utilization for P-V application

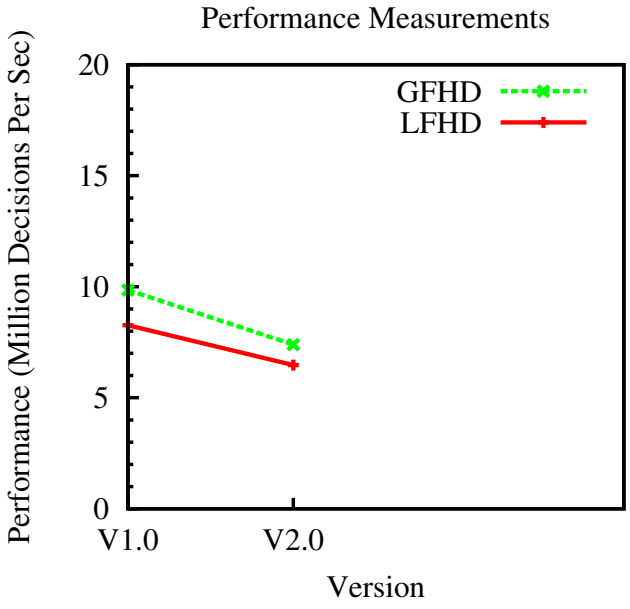

Figure 5.4: Plot comparing performance for P-V application

each other. The last column of the Table 5.2 indicates the degree of effectiveness, and shows good or excellent for the lines of code changed, resources, and performance. Hence, we can infer that guideline followed design is effective in terms of lines of code changed, resources, and performance for the P-V system modeling application. The performance of the guideline followed design is better than literature followed design, and the guideline followed design requires an additional 5% resources.

5.1.2 2D-Finite Difference Time Domain

Electromagnetic wave analysis is performed by solving Maxwell's equations. These equations are partial differential equations that govern the propagation of electromagnetic waves. They are discretized over a finite volume, and the derivatives are approximated using central difference approximations. These finite-difference equations are then solved in a leap-frog manner to compute the electric and magnetic fields (E and H, respectively) in the finite-difference time domain (FDTD) method [17]. The performance of the FDTD application is measured as the rate at which the E and H are computed over a finite volume.

5.1.2.1 Software Design

The software designs are created following the specifications described in [3, 69] (version 1.0), [4, 70] (version 2.0), and [5] (version 3.0). Each design has a source and a receiver port. A source signal is given at the source port and the response is measured at the receiver port. Results from the receiver port is then compared with the hardware designs. The software designs are built around perfect electric conductor (PEC) model, Mur model, and uniaxial perfectly matched layer (UMPL) model.

5.1.2.2 Literature Followed Hardware Design

The literature followed designs were built following the designs presented in [3, 69] (version 1.0), [4, 70] (version 2.0), and [5] (version 3.0). The designs are recreated as close as possible, with the resources and performance closely matched to the resources

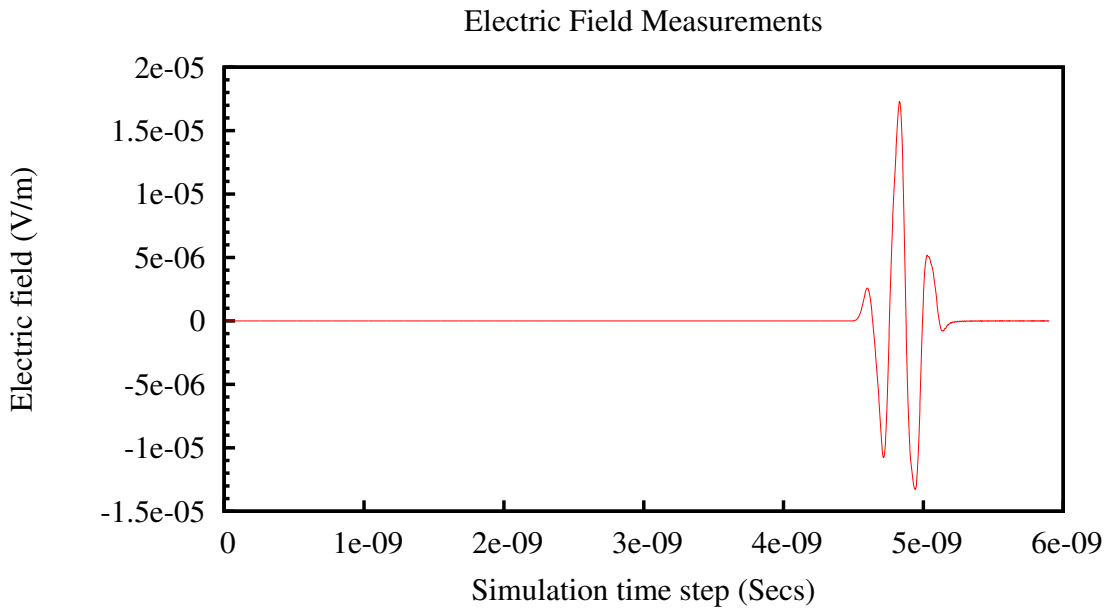

Figure 5.5: Electric field at receiver port for 2D-FDTD PEC Model

and the performance reported in the literature. The plots 5.5, and 5.6 shows the receiver port electric field values for PEC model. An error plot is shown in Figure 5.7 for 5000 iterations. For every iteration root mean square error (RMSE) is computed for all the Ez, Hx, and Hy field values. There were few error due to Ethernet packet drop and, these points were removed from the final plot to show the exact root mean square error of the fields for every iteration.

Plot 5.8 shows the electric field values, and the error between software and LFHD hardware electric field values. Similarly, 5.9, and 5.10 shows the electric field values for UMPL model. The resource used, performance and the lines of code for each version is presented in Tables 5.3 and 5.4.

5.1.2.3 Guideline Followed Hardware Design

The guideline followed design follows the design guidelines to build the designs presented in [3, 69] (version 1.0), [4, 70] (version 2.0), and [5] (version 3.0). Plots 5.11, 5.13, and 5.14 show the electric field values and the values are similar to the results from the software and the literature followed design results. An error plot is shown in

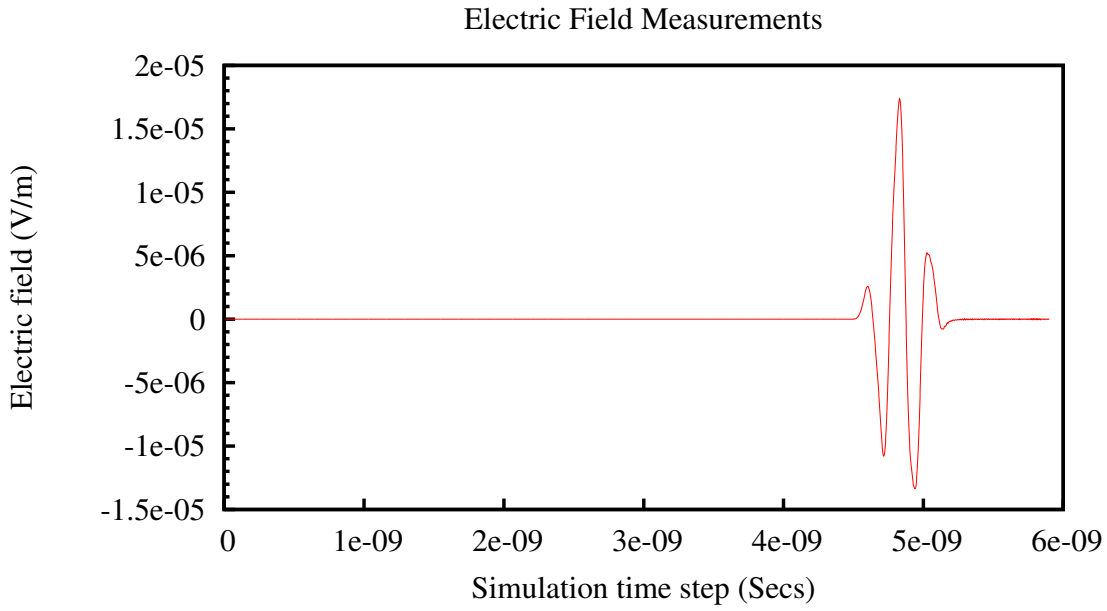

Figure 5.6: Electric field at receiver for 2D-FDTD LFHD PEC Model

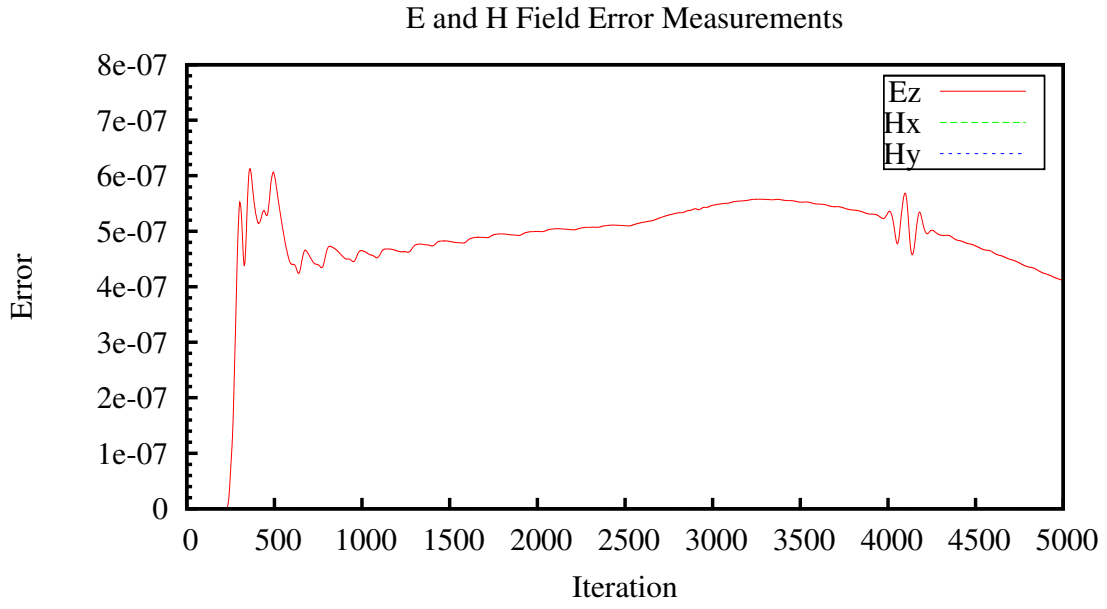

Figure 5.7: Root mean square error value for E and H Fields for PEC model

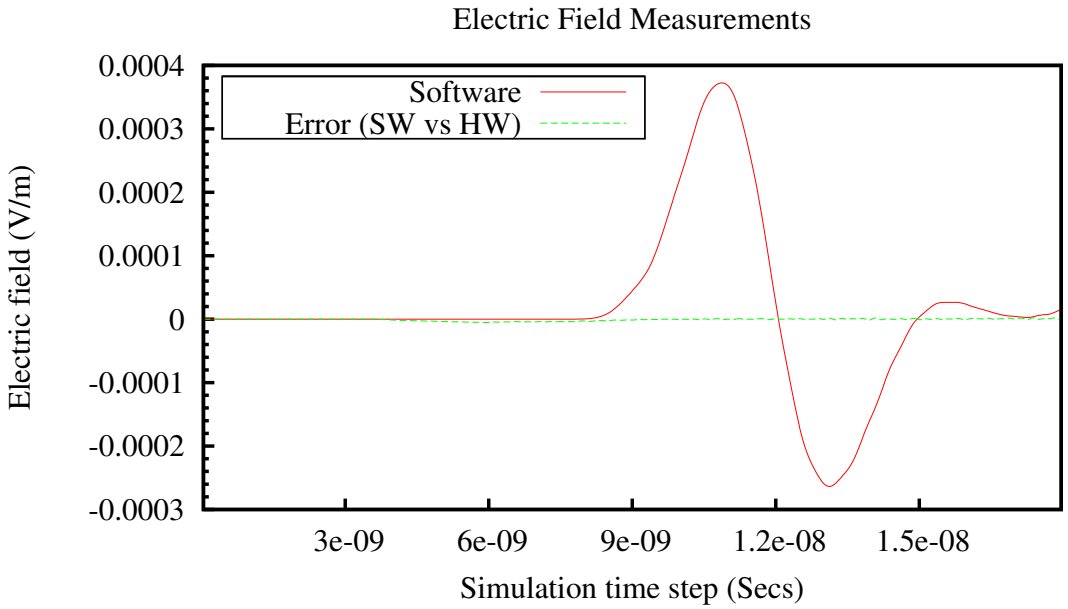

Figure 5.8: Electric field at receiver port for 2D-FDTD Mur Model

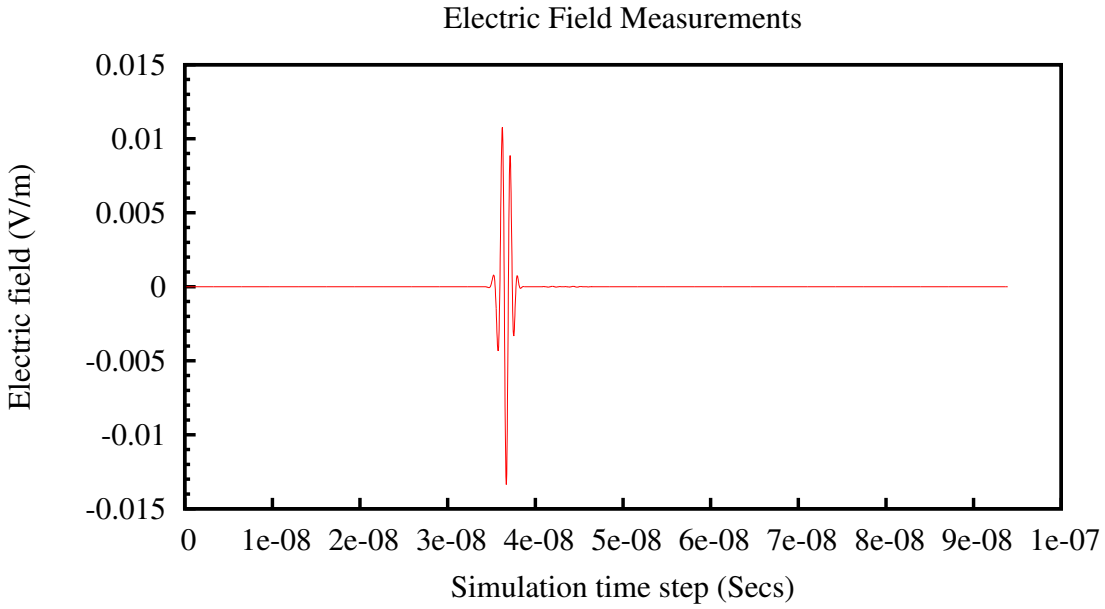

Figure 5.9: Electric field at receiver port for 2D-FDTD UMPL Model

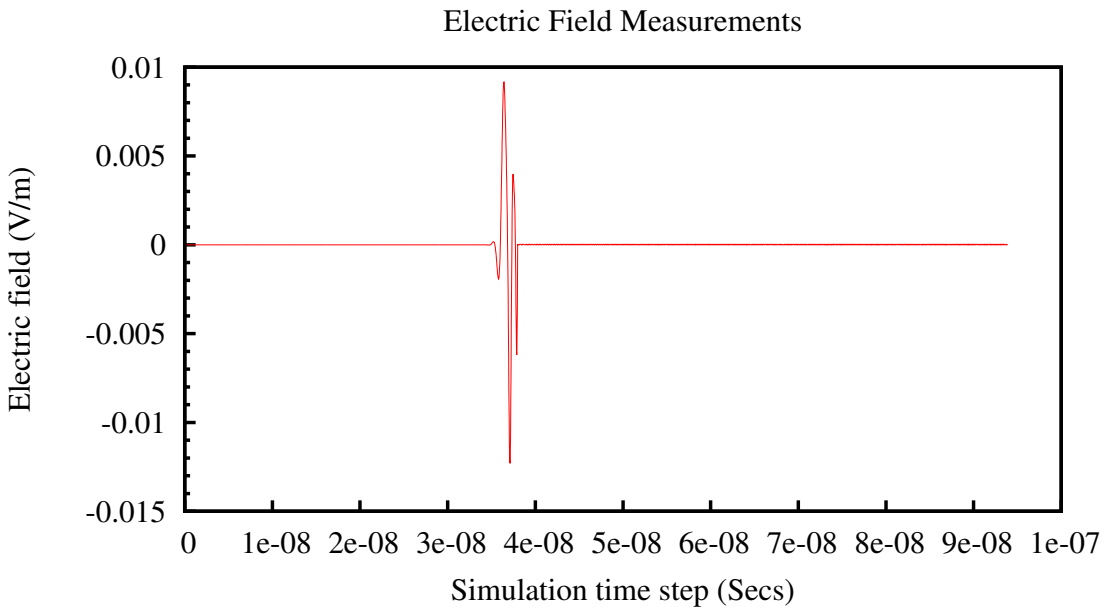

Figure 5.10: Electric field at receiver port for 2D-FDTD UMPL LFHD Model

Figure 5.12 for 5000 iterations. For every iteration root mean square error (RMSE) is computed for all the Ez, Hx, and Hy field values. The resource used, the performance, and the lines of code for each version after removing the resources for Ethernet core and its peripherals is presented in Tables 5.3 and 5.4.

5.1.2.4 Results

The results of the designed hardware from literature and using the guidelines are presented in Tables 5.3 and 5.4. Plots 5.15, 5.17, 5.16, and 5.18 are drawn for lines of

Table 5.3: Comparison of version 1 and 2 results for FDTD application

Measurements	**Version 1.0**			**Version 2.0**			**Effectiveness**
	LFHD	GFHD	(G-L)/G	LFHD	GFHD	(G-L)/G	Bad/Good/Xlnt
Lines of code							
-Delta LOC	0	0	N.A	2718	1570	-73.12%	Xlnt
Resource Utilization							
-Slices	12,582	12,111	-3.88%	11,616	10,471	-10.93%	Xlnt
-BRAM blocks	77	77	0%	93	93	0%	Xlnt
-DSP48 slices	16	16	0%	0	0	0%	Xlnt
Measurements	**Version 1.0**			**Version 2.0**			**Effectiveness**
	LFHD	GFHD	L/G	LFHD	GFHD	L/G	Bad/Good/Xlnt
Performance							
-Computation time (ms)	21	21	1.0	159	158	1.0	Good

Xlnt - Excellent

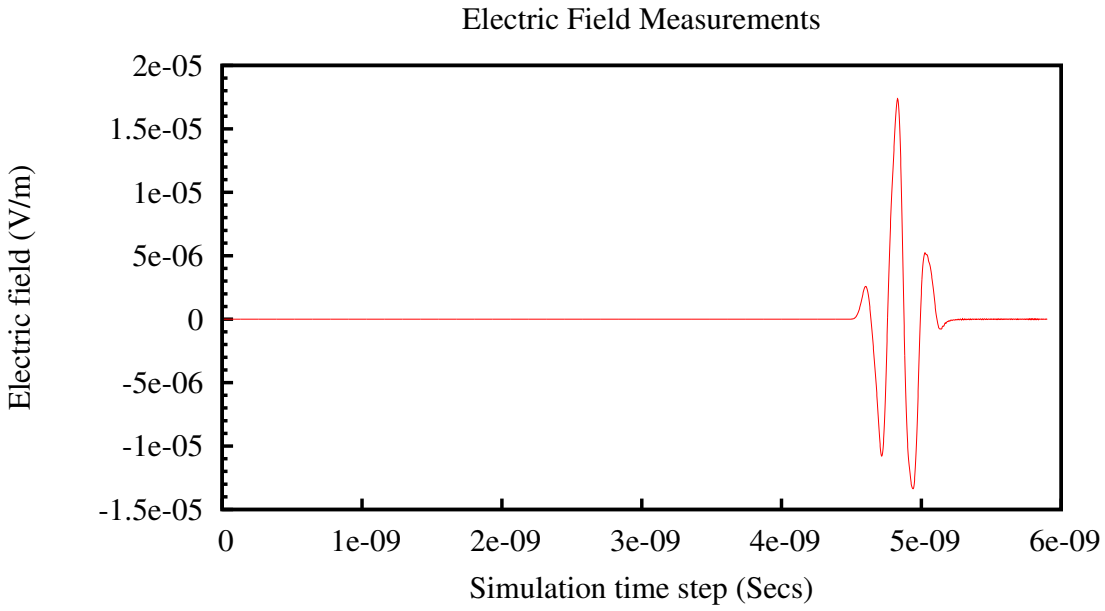

Figure 5.11: Electric field at receiver port for 2D-FDTD GFHD PEC Model

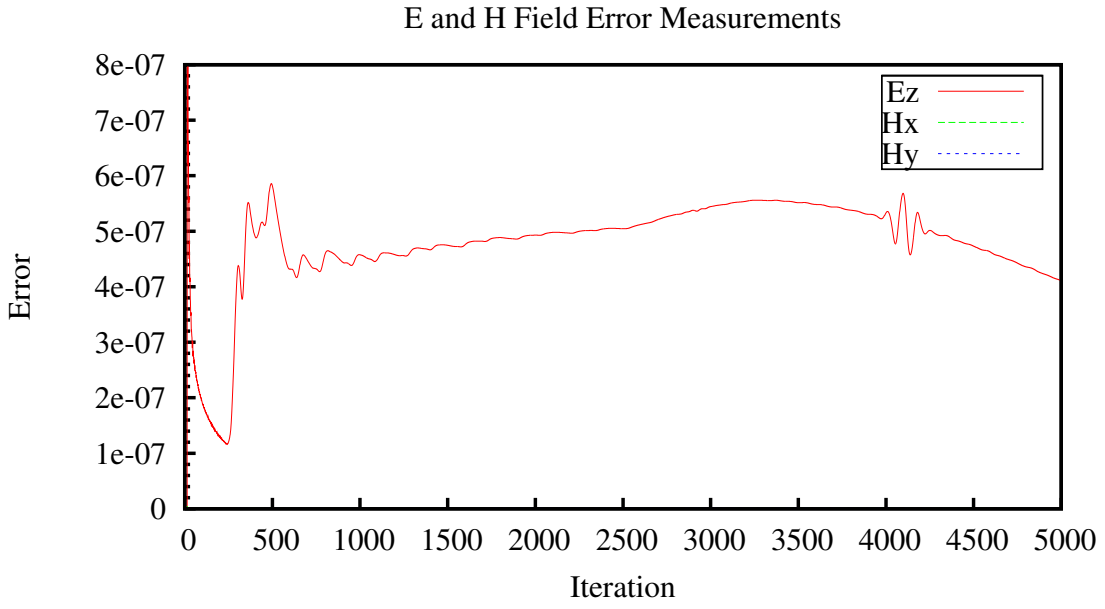

Figure 5.12: RMSE values for E and H Fields for GFHD PEC model

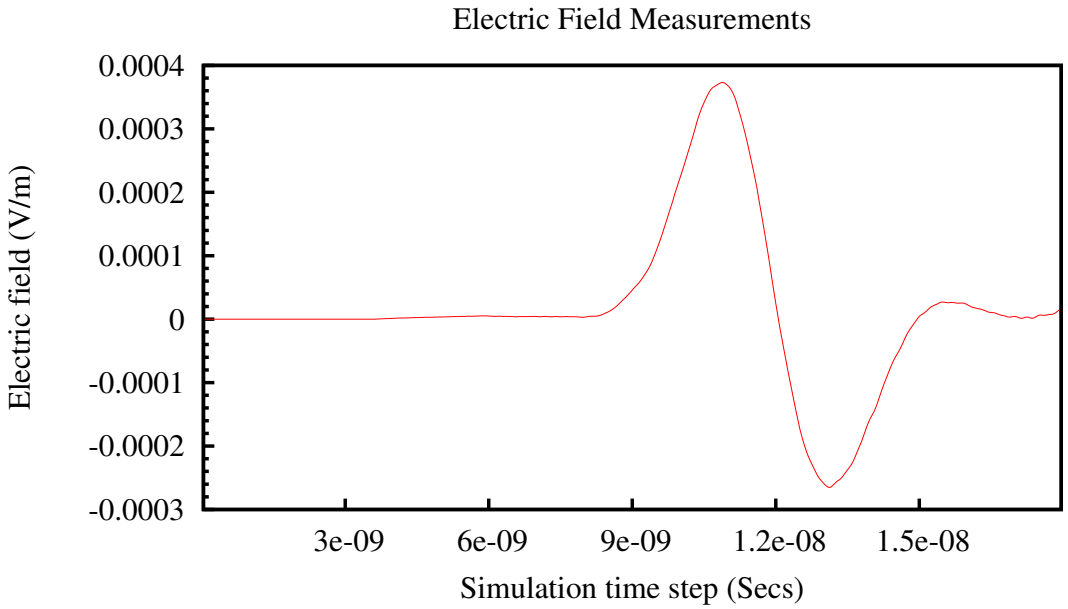

Figure 5.13: Electric field at receiver port for 2D-FDTD Mur GFHD Model

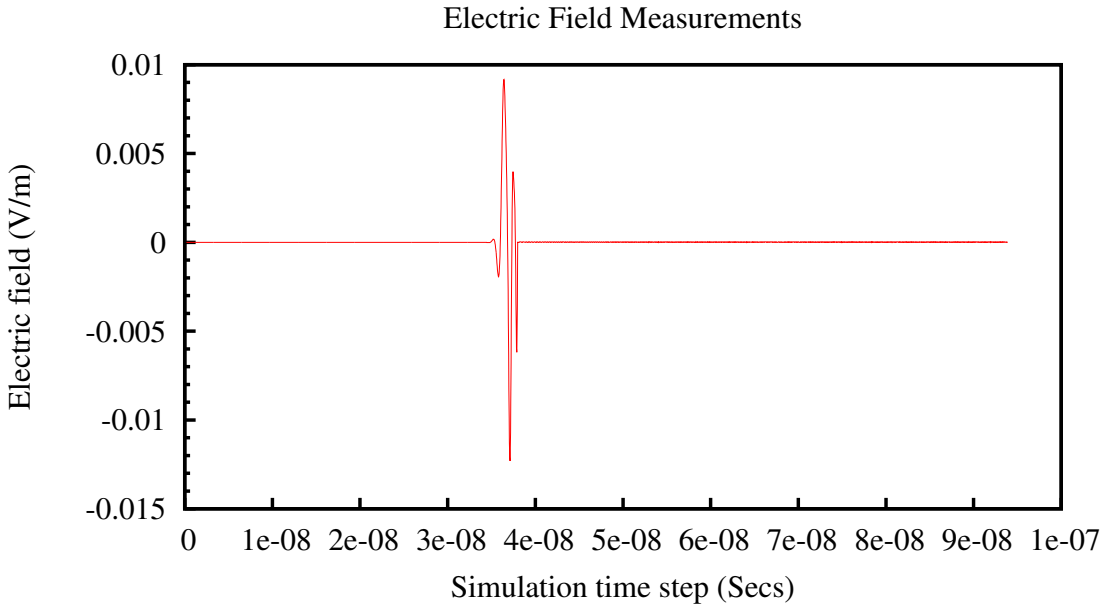

Figure 5.14: Electric field at receiver port for 2D-FDTD UMPL GFHD Model

Table 5.4: Comparison of version 2 and 3 results for FDTD application

Measurements	**Version 2.0**			**Version 3.0**			**Effectiveness**
	LFHD	GFHD	(G-L)/G	LFHD	GFHD	(G-L)/G	Bad/Good/Xlnt
Lines of code							
-Delta LOC	2718	1570	-73.12%	4842	3703	-30.75%	Good
Resource Utilization							
-Slices	11,616	10,471	-10.93%	15,750	14,452	-8.98%	Xlnt
-BRAM blocks	93	93	0%	197	161	-22.36%	Xlnt
-DSP48 slices	0	0	0%	84	64	-31.25%	Xlnt
Measurements	**Version 2.0**			**Version 3.0**			**Effectiveness**
	LFHD	GFHD	G/L	LFHD	GFHD	G/L	Bad/Good/Xlnt
Performance							
-Computation time (ms)	159	158	1.0	7350	7350	1.0	Good

Xlnt - Excellent

Table 5.5: Comparison of reported versus used resources for applications

Application	**Slices**			**BRAMs**			**DSP48 Slices**		
	Reported	LFHD	GFHD	Reported	LFHD	GFHD	Reported	LFHD	GFHD
FDTD Ver 1	7,640	15,712	15,241	6	12(77)	12(77)	16	16	16
FDTD Ver 2	8,832	14,746	13,601	20	40(93)	40(93)	0	0	0
FDTD Ver 3	15,787	18,880	17,582	166	197	161	92	84	64
SpMV Ver 1	16,613	19,125	19,163	ND	185	185	ND	64	64
SpMV Ver 2	10,050	12,933	13,429	ND	69	69	ND	16	16
SpMV Ver 3	2,140	11,354	11,203	ND	85	85	9	16	16

ND - No Data

code changed, resources, and performance respectively. Guidelines 4, 5, and 10 were used to design GFHD. The (G-L)/G or (GFHD-LFHD)/GFHD column represents the percentage increase or decrease in the lines of code and resources used for every version (version 1.0, 2.0, and 3.0), and the degree of effectiveness column presents the classification for effectiveness. Similarly, for the performance, execution time in milliseconds (milli secs) is reported in the Tables 5.3 and 5.4. The (G/L) or (GFHD/LFHD) column presents the final performance factor between the literature followed design and the guideline followed design for every version. Table 5.5 shows the reported resource usage from the literature, and the resources from the place and route report for LFHD and GFHD for FDTD and SpMV applications.

5.1.2.5 Observations

The observations from the plots 5.18 and 5.17, 5.16 shows that data points of GFHD and LFHD for performance and resource utilization track each other. Similarly, plot 5.15 for lines of code changed shows that the data points for GFHD and LFHD diverge. The observations from the plots shows that the design guidelines are

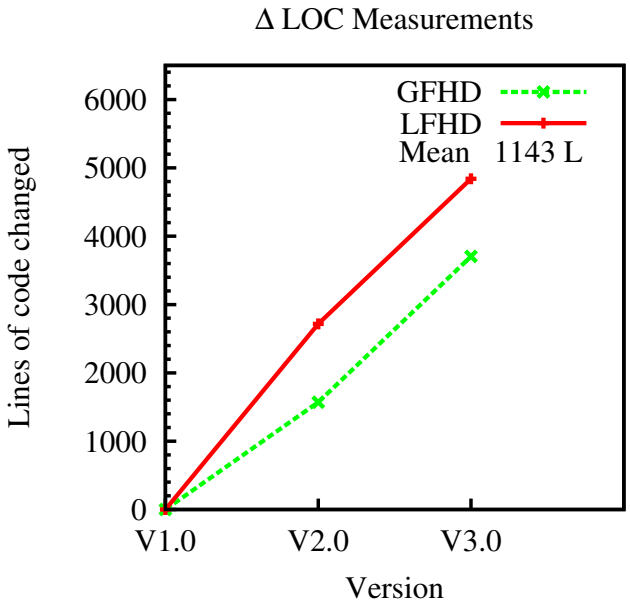

Figure 5.15: Plot comparing lines of code changed for FDTD

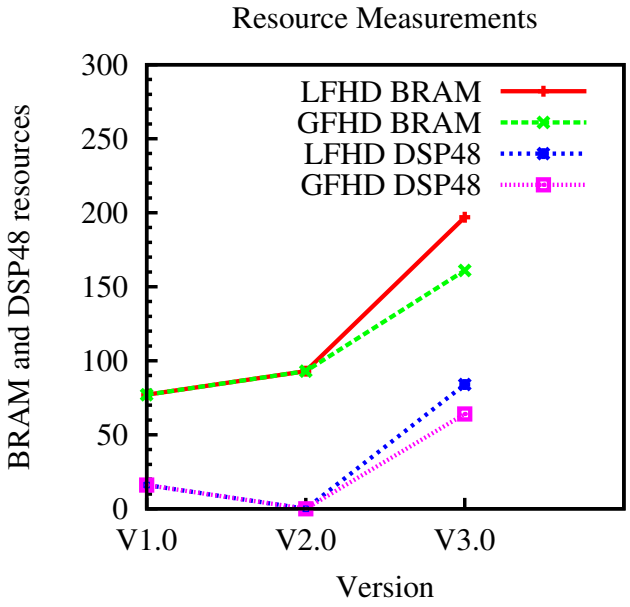

Figure 5.16: Plot comparing resource utilization for FDTD

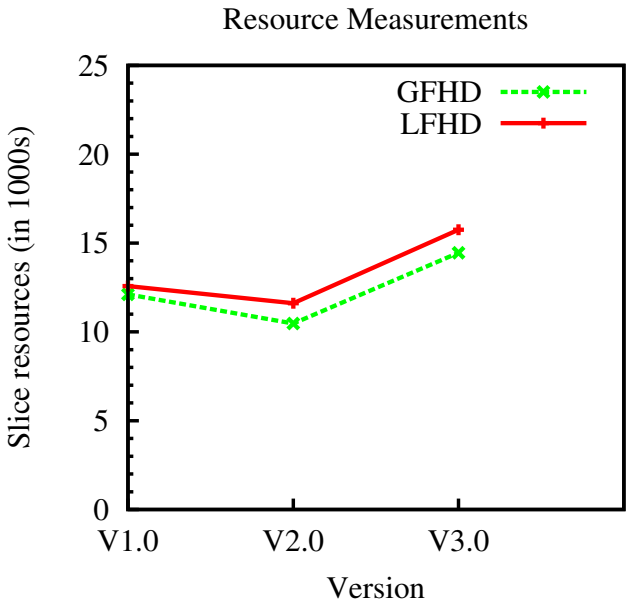

Figure 5.17: Plot comparing slice resource utilization for FDTD

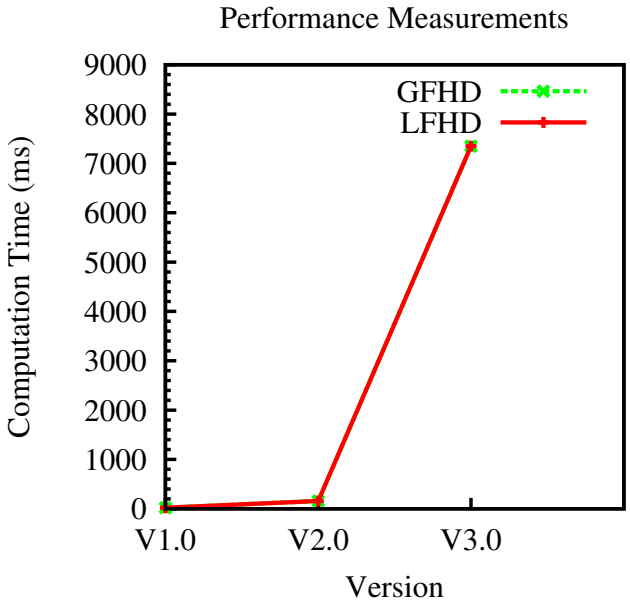

Figure 5.18: Plot comparing performance for FDTD

effective for FDTD design. The observations made from the last column of the Tables 5.3 and 5.4 shows the degree of effectiveness, and shows good or excellent for the lines of code changed, resources, and performance. Hence, we can infer that guideline followed design is effective in terms of lines of code changed, resources, and performance for the FDTD application. The performance of the guideline followed design is as close as literature followed design, and the guideline followed design requires an additional 10% resources.

5.1.3 Sparse Matrix Vector Multiplication

Sparse matrix vector multiplication (SpMV) is an operation that is very commonly used in many computational science application. Each version computes $Ax = y$, where 'A' is a sparse matrix and 'x' is a vector. The result of the computation is stored in 'y'. In order to evaluate the design guidelines, three versions of this application is built using software, literature, and using the design guidelines. The following sections discusses about the designs.

5.1.3.1 Software Design

Every version of sparse matrix design has a small percentage of software pre-processing. The software pre-processing consists of converting the raw matrix into compressed sparse row (CSR) format. Once the CSR format is obtained, the matrix is stored in the required format. Every version uses its own format for matrix computation. The matrix data is copied to FPGA's DDR2 SDRAM, and from where the data is then copied (DMA) for actual computation. The first design (version 1) is a row major design, the second design (version 2) is column major, and the third one (version 3) is multiple row format. Once the data is copied to the hardware, a matrix multiplication is performed in software. The software code is written in 'C', and the results are used to compare the results from the hardware.

Table 5.6: Performance for SpMV version 1 LFHD design

Matrix name	**RowsxCols**	**Non-Zeros**	**Sparsity**	**Clock Cycles**	**Performance**
raefsky3	21200×21200	1488768	0.331	438843	1085.59
bcsstk35	30237×30237	740200	0.081	331010	715.58
rdist1	4134×4134	94408	0.552	30767	981.91
memplus	17758×17758	126150	0.040	102014	395.71
gemat11	4929×4929	33185	0.137	16881	629.06
lns3937	3937×3937	25407	0.164	13488	602.78
sherman5	3312×3312	20793	0.190	11041	602.64
mcfe	765×765	24382	4.166	7472	1044.20
jpwh991	991×991	6027	0.614	3248	593.79
bp1600	822×822	4841	0.716	2655	583.47
str600	363×363	3279	2.488	1465	716.23

5.1.3.2 Literature Followed Hardware Design

The literature followed hardware design is built using the design guidelines presented in [6] (version 1.0), [7] (version 2.0), and [8] (version 3.0). The designs are recreated as close as possible. The designs are recreated such that the resources and performance are closely matched. The designs are tested for all the matrices presented in the literature. In cases where matrices are not specified, matrices of similar characteristics are taken from University of Florida sparse matrix collection. The results of the test are presented in Tables 5.6, 5.7, and 5.8. Tables 5.12 and 5.13 presents the resource used, performance, and the lines of code changed for each version.

5.1.3.3 Guideline Followed Hardware Design

The guideline followed hardware design is built using the design guidelines presented in Table 4.1, and were used to design the hardware presented in [6] (version 1.0), [7] (version 2.0), and [8] (version 3.0). The designs are tested for all the matrices presented in the literature. In cases where matrices are not specified, matrices of

Table 5.7: Performance for SpMV version 2 LFHD design

Matrix name	RowsxCols	Non-Zeros	Sparsity	Clock Cycles	Performance
ex40	7740×7740	456188	0.761	674381	128.53
ex19	12005×12005	259577	0.18	414105	119.10
raefsky	21200×21200	1488768	0.331	2198718	128.65
thread	29736×29736	2237308	0.253	3310498	128.41
mark3jac	45769×45769	268563	0.013	925710	55.12
TSOPF	56814×56814	4391071	0.136	6531825	127.73
Chebyshev	68121×68121	5377761	0.116	8142439	125.49
consph	83334×83334	3046907	0.044	4860309	119.11
s3dkq	90449×90449	2259087	0.028	3732869	114.99
m_t1	97578×97578	4925574	0.052	7525506	124.36
x104	108384×108384	4410993	0.038	6879856	121.82
torso	116158×116158	8516500	0.063	13004952	124.42
bone	127224×127224	2821913	0.017	5111085	104.90
bmwcr1	148770×148770	5395186	0.024	8429385	121.61
Si02	155331×155331	5719417	0.024	11467648	94.76
PR02R	161070×161070	8185136	0.032	14638567	106.24
Si41G	185639×185639	7598452	0.022	14504176	99.54
pwtk	217918×217918	5871175	0.012	9399439	118.68
bmw3	227362×227362	5757996	0.011	10193641	107.32
BenElachi	245874×245874	6698185	0.011	10816490	117.66

Table 5.8: Performance for SpMV version 3 LFHD design

Matrix name	RowsxCols	Non-Zeros	Sparsity	Clock Cycles	Performance
gemat12	4929×4929	33044	0.136	34774	190.05
k3plates	11107×11107	378927	0.307	379763	199.56
wang3	26064×26064	177168	0.026	183550	193.05
jnlbrng1	40000×40000	119600	0.007	123631	193.48
epb3	84617×84617	463625	0.006	482792	192.06
cont-300	180895×180895	539396	0.002	587866	183.51

Table 5.9: Performance for SpMV Version 1 GFHD design

Matrix name	RowsxCols	Non-Zeros	Sparsity	Clock Cycles	Performance
raefsky3	21200×21200	1488768	0.331	438839	1085.60
bcsstk35	30237×30237	740200	0.081	331020	715.56
rdist1	4134×4134	94408	0.552	30767	981.91
memplus	17758×17758	126150	0.040	102007	395.74
gemat11	4929×4929	33185	0.137	16881	629.06
lns3937	3937×3937	25407	0.164	13488	602.78
sherman5	3312×3312	20793	0.190	11041	602.64
mcfe	765×765	24382	4.166	7472	1044.20
jpwh991	991×991	6027	0.614	3248	593.79
bp1600	822×822	4841	0.716	2655	583.47
str600	363×363	3279	2.488	1465	716.23

similar characteristics are used from University of Florida sparse matrix collection. The results of the test are presented in Tables 5.9, 5.10, and 5.11. The resource used, performance, and the lines of code for each version after removing resources for additional peripherals is presented in Tables 5.12 and 5.13.

5.1.3.4 Results

The results of the literature followed hardware design and guideline followed hardware design are presented in Tables 5.12 and 5.13. Plots 5.19, 5.21, 5.20, and 5.22 are drawn for lines of code changed, resources, and performance respectively. Guidelines 4, 5, and 10 were used for designing GFHD. The (G-L)/G or (GFHD-LFHD)/GFHD column in Tables 5.12 and 5.13 represents the percentage increase or decrease in the lines of code and resources used for every version (version 1.0, 2.0, and 3.0), and the degree of effectiveness is presented in the last column. Similarly, for the performance, floating point operations per second (FLOPS) is reported in the Tables 5.12 and 5.13. The (G/L) or (GFHD/LFHD) column presents the final performance factor between the literature followed design and guideline followed design for every version.

Table 5.10: Performance for SpMV Version 1 GFHD design

Matrix name	RowsxCols	Non-Zeros	Sparsity	Clock Cycles	Performance
ex40	7740×7740	456188	0.761	675346	128.34
ex19	12005×12005	259577	0.18	414469	118.99
raefsky	21200×21200	1488768	0.331	2199836	128.59
thread	29736×29736	2237308	0.253	3315660	128.21
mark3jac	45769×45769	268563	0.013	972830	52.45
TSOPF	56814×56814	4391071	0.136	6532468	127.72
Chebyshev	68121×68121	5377761	0.116	8159338	125.23
consph	83334×83334	3046907	0.044	4898070	118.19
s3dkq	90449×90449	2259087	0.028	3880403	110.61
m_t1	97578×97578	4925574	0.052	7567432	123.67
x104	108384×108384	4410993	0.038	6994637	119.82
torso	116158×116158	8516500	0.063	14170591	114.19
bone	127224×127224	2821913	0.017	5375678	99.74
bmwcr1	148770×148770	5395186	0.024	8478997	120.90
Si02	155331×155331	5719417	0.024	11555164	94.04
PR02R	161070×161070	8185136	0.032	14986091	103.77
Si41G	185639×185639	7598452	0.022	14627052	98.70
pwtk	217918×217918	5871175	0.012	9612883	116.04
bmw3	227362×227362	5757996	0.011	10616153	103.05
BenElachi	245874×245874	6698185	0.011	11139398	114.25

Table 5.11: Performance for SpMV Version 1 GFHD design

Matrix name	RowsxCols	Non-Zeros	Sparsity	Clock Cycles	Performance
gemat12	4929×4929	33044	0.136	34774	190.05
k3plates	11107×11107	378927	0.307	379763	199.56
wang3	26064×26064	177168	0.026	183550	193.05
jnlbrng1	40000×40000	119600	0.007	123631	193.48
epb3	84617×84617	463625	0.006	482792	192.06
cont-300	180895×180895	539396	0.002	587866	183.51

Table 5.12: Comparison of version 1 and 2 results for SpMV application

Measurements	Version 1.0			Version 2.0			Effectiveness
	LFHD	GFHD	(G-L)/G	LFHD	GFHD	(G-L)/G	Bad/Good/Xlnt
Lines of code							
-Delta LOC	0	0	N.A	1838	1422	-29.25%	Good
Resource Utilization							
-Slices	18,321	18,359	0.21%	9,803	10,299	5.05%	Xlnt
-BRAM blocks	185	185	0%	69	69	0%	Xlnt
-DSP48 slices	64	64	0%	16	16	0%	Xlnt
Measurements	**Version 1.0**			**Version 2.0**			**Effectiveness**
	LFHD	GFHD	G/L	LFHD	GFHD	G/L	Bad/Good/Xlnt
Performance							
-Million FLOPS	395	395	1.0	114.42	112.33	0.98 (≈1.0)	Good

Xlnt - Excellent

Table 5.13: Comparison of version 2 and 3 results for SpMV application

Measurements	**Version 2.0**			**Version 3.0**			**Effectiveness**
	LFHD	GFHD	(G-L)/G	LFHD	GFHD	(G-L)/G	Bad/Good/Xlnt
Lines of code							
-Delta LOC	1838	1422	N.A	1804	1323	-36.36%	Good
Resource Utilization							
-Slices	9,803	10,299	5.05%	10,550	10,399	-1.45%	Xlnt
-BRAM blocks	69	69	0%	85	85	0%	Xlnt
-DSP48 slices	16	16	0%	16	16	0%	Xlnt
Measurements	**Version 2.0**			**Version 3.0**			**Effectiveness**
	LFHD	GFHD	G/L	LFHD	GFHD	G/L	Bad/Good/Xlnt
Performance							
-Million FLOPS	114.42	112.33	0.98 (≈1.0)	183.51	183.27	0.99 (≈1.0)	Good

Xlnt - Excellent

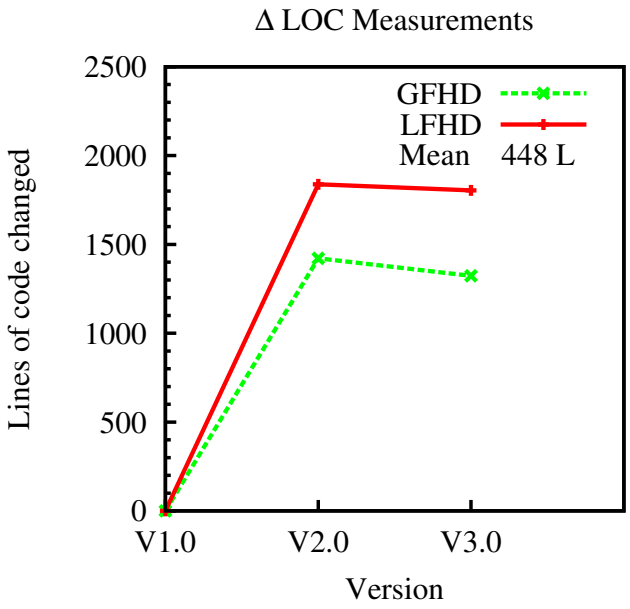

Figure 5.19: Plot comparing lines of code changed for SpMV operation

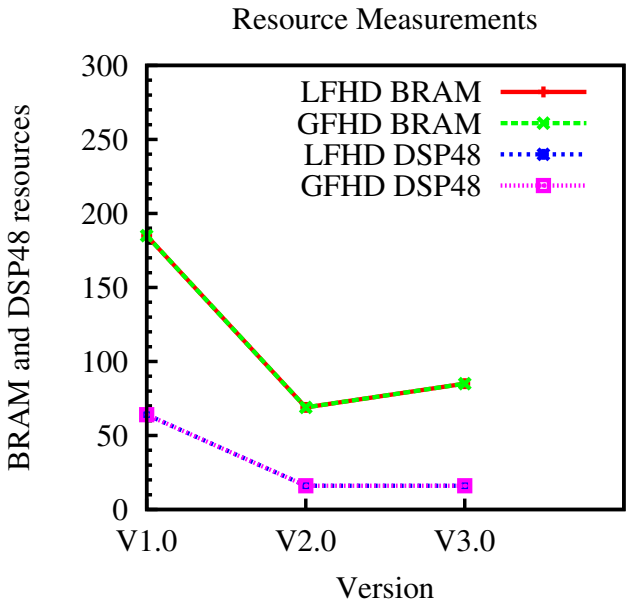

Figure 5.20: Plot comparing resource utilization for SpMV operation

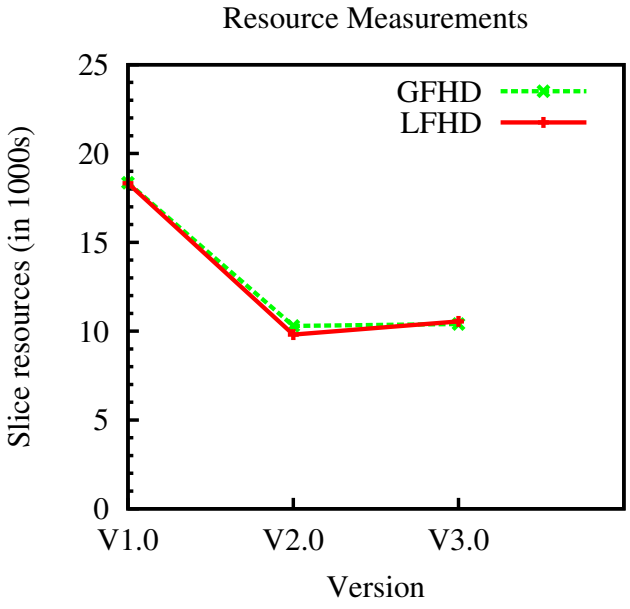

Figure 5.21: Plot comparing resource utilization for SpMV operation

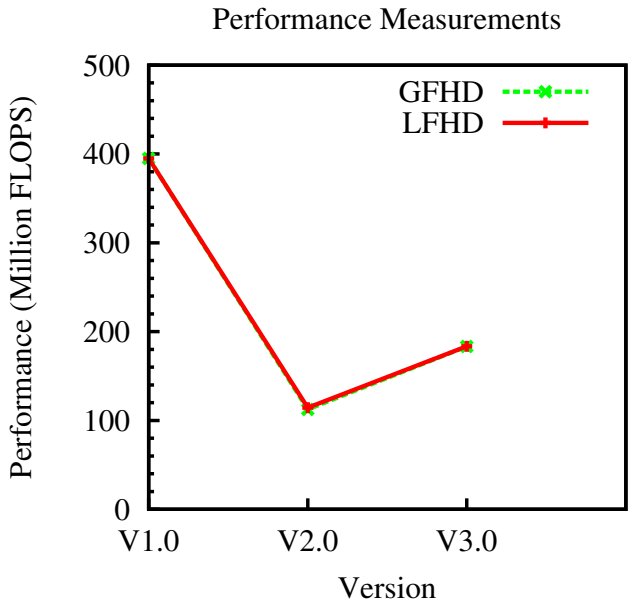

Figure 5.22: Plot comparing performance for SpMV operation

5.1.3.5 Observations

Observations from the plots 5.22, 5.21, and 5.20 show that the data points of GFHD and LFHD for the performance and the resource utilization track each other. Similarly, the data points for lines of code changed for GFHD and LFHD diverge, as shown in plot 5.19, and the observations from the plots suggests that the design guidelines are effective for sparse matrix vector multiplication operation. The observations made from the last column of the Tables 5.12 and 5.13 shows the degree of effectiveness, and the results are good or excellent for the lines of code changed, resources, and performance. Hence, we can infer that guideline followed design is effective in terms of lines of code changed, resources, and performance for the SpMV operation. The performance of the guideline followed design is as close as literature followed design, and the guideline followed design requires an additional 5% resources.

5.2 Broad Applicability of the Design Guidelines

In order to answer broad applicability of the design guidelines, a guideline fitness plot is plotted for six applications. The design guidelines are validated on the appli-

cations to understand the applicability of the design guidelines. A guideline fitness plot is plotted for each application to understand the relation between performance and the number of design guidelines followed. The six applications that are chosen from the literature are:

- Computational Fluid Dynamics
- Computational Molecular Dynamics
- Quantum Monte Carlo Simulations
- Hessenberg Reduction
- Gaxpy - BLAS Routine
- N-Body Simulations

5.2.1 Computational Fluid Dynamics

Computational fluid dynamics (CFD) simulation is a numerical method to solve problems involving fluid flows on discrete space and time. The set of design guidelines is evaluated on the design presented in [9]. The evaluation is carried out by a rubric discussed in section 4.3.1. Every design guideline that is followed is given a score of $+1$, and those that are not followed is given a score of -1. The evaluated design guideline values are added to get the total score. These values are used to plot the guideline fitness plot. The CFD design is evaluated to find out whether or not the design guidelines have been followed in Table 5.14, and the total score is 4. The scaled score is $4/12{=}0.33$, as plotted on the x-axis, and the y-axis shows the actual performance compared to the theoretical performance. The work referred here uses a single Altera Stratix II FPGA of the DN7000k10 PCI board. The implemented design [9] has a systolic array with $12{\times}8$ cells that consumes $\approx$50% logic cells with all of the embedded multipliers. Every cell has a multiplier accumulate unit (MACC),

Table 5.14: CFD design evaluated using the design guidelines

1	*Arrange and optimize input/output data for all compute blocks*	✔
2	*Adhere to a widely accepted method of computation for arithmetic functions/operations*	✔
3	*Build controller for every not likely to change compute blocks*	✔
4	*Introduce a configurable dataflow path to connect resources of likely to change compute blocks*	✘
5	*Introduce on-chip memory with configurable read/write logic, if necessary*	✔
6	*Introduce dependency indicators to enhance parallel computations, if necessary*	✔
7	*Use microcode to specify dataflow path, memory read/write, and parallel computations*	✘
8	*Maximize resource utilization by improving runtime parallelism*	✔
9	*Forward results between compute blocks/resources, if possible*	✔
10	*Achieve functionality, and optimize design & resources to improve performance*	✔
11	*Maximize computation until maximum memory bandwidth is utilized*	✘
12	*Use a large and real dataset for test cases*	✘

which can operate at 90 MHz, but, when put together, can only operate at 60 MHz. The MACC has five stages; however, if these stages are increased further, a higher frequency can be achieved. Assuming the theoretical frequency to be 90 MHz and the utilization to be 98%, the peak theoretical frequency is computed as $90/0.98 = 91.83$ MHz. The performance of the design depends on the frequency and its utilization, and the overall performance when compared to the theoretical peak is computed as $0.60/91.83 = 0.653$. The scaled performance index $(0.653\times2) - 1 = 0.31$ is plotted on the y-axis of the guideline fitness plot, as shown in Figure 5.23.

5.2.2 Computational Molecular Dynamics

Molecular dynamics (MD) simulation is a study of movements of atoms and molecules. A two-body simulation design is discussed in [10]. The design uses fixed precision for computation and does not use microcode to configure the resources. Due to these reasons, the design cannot be modified easily when there is change in the design. Table 5.15 shows the design guidelines that have been followed. Since parallel computations cannot be performed, only 95% performance can be achieved [10]. The

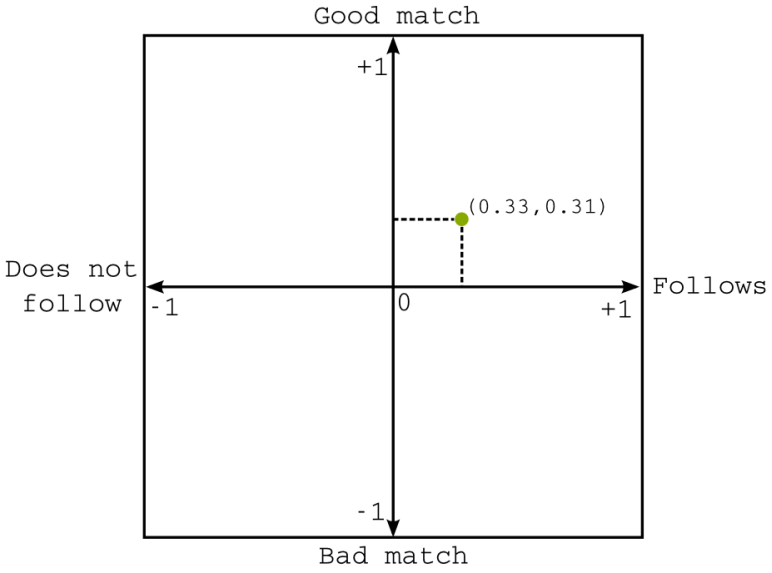

Figure 5.23: Guideline fitness plot for CFD application

scaled performance index $(0.95\times2) - 1 = 0.9$ is plotted on the y-axis of the guideline fitness plot, as shown in Figure 5.24.

5.2.3 Quantum Monte Carlo Simulations

Quantum Monte Carlo (QMC) simulations study the structural and energetic properties of a group of atoms or molecules. There are two types of QMC simulations: diffusion Monte Carlo (DMC) and variational Monte Carlo (VMC) [11]. The design uses fixed precision for computation and does not use microcode to configure the datapath. Table 5.16 shows the design guidelines that have been followed. A parallel design can be easily built using the resources and the available memory bandwidth. This could have increased the performance by 50%, and thus the theoretical peak performance is 1.5 times the actual performance. The actual performance compared to the peak theoretical performance is $1/1.5 = 0.667$. The scaled performance index $(0.667\times2) - 1 = 0.33$ is plotted on the y-axis of the guideline fitness plot, as shown in Figure 5.25.

Table 5.15: MD design evaluated using the design guidelines

1	*Arrange and optimize input/output data for all compute blocks*	✔
2	*Adhere to a widely accepted method of computation for arithmetic functions/operations*	✘
3	*Build controller for every not likely to change compute blocks*	✔
4	*Introduce a configurable dataflow path to connect resources of likely to change compute blocks*	✘
5	*Introduce on-chip memory with configurable read/write logic, if necessary*	✔
6	*Introduce dependency indicators to enhance parallel computations, if necessary*	✔
7	*Use microcode to specify dataflow path, memory read/write, and parallel computations*	✘
8	*Maximize resource utilization by improving runtime parallelism*	✔
9	*Forward results between compute blocks/resources, if possible*	✔
10	*Achieve functionality, and optimize design & resources to improve performance*	✔
11	*Maximize computation until maximum memory bandwidth is utilized*	✔
12	*Use a large and real dataset for test cases*	✔

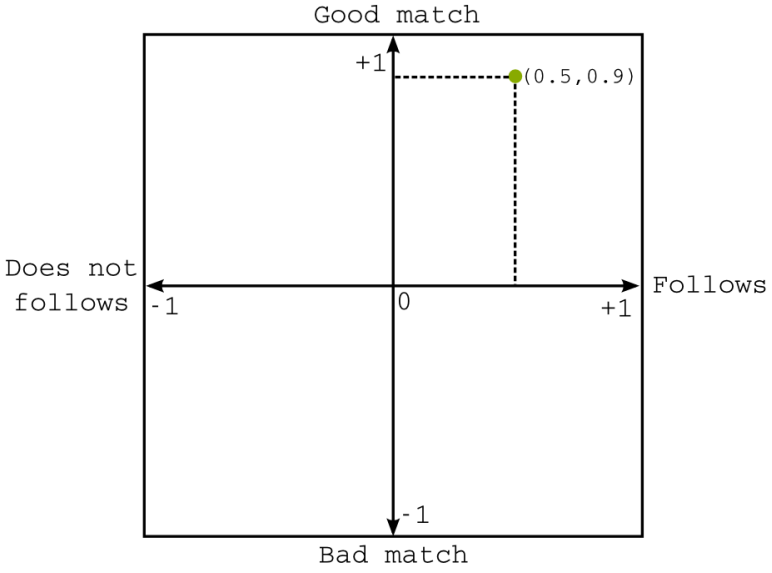

Figure 5.24: Guideline fitness plot for molecular dynamics application

Table 5.16: QMC design evaluated using the design guidelines

1	*Arrange and optimize input/output data for all compute blocks*	✔
2	*Adhere to a widely accepted method of computation for arithmetic functions/operations*	✘
3	*Build controller for every not likely to change compute blocks*	✔
4	*Introduce a configurable dataflow path to connect resources of likely to change compute blocks*	✘
5	*Introduce on-chip memory with configurable read/write logic, if necessary*	✔
6	*Introduce dependency indicators to enhance parallel computations, if necessary*	✔
7	*Use microcode to specify dataflow path, memory read/write, and parallel computations*	✘
8	*Maximize resource utilization by improving runtime parallelism*	✔
9	*Forward results between compute blocks/resources, if possible*	✔
10	*Achieve functionality, and optimize design & resources to improve performance*	✘
11	*Maximize computation until maximum memory bandwidth is utilized*	✔
12	*Use a large and real dataset for test cases*	✔

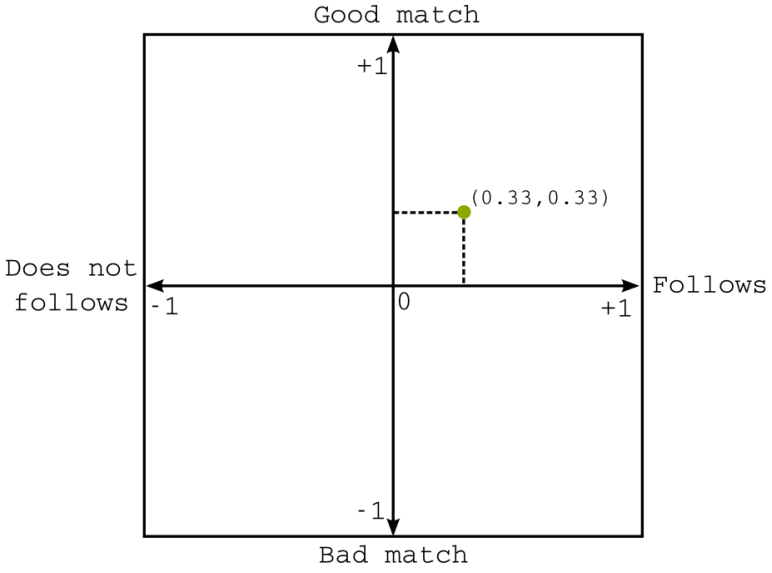

Figure 5.25: Guideline fitness plot for quantum Monte Carlo simulations

5.2.4 Hessenberg Reduction

Hessenberg reduction (HR) reduces a square matrix in to an upper or lower Hessenberg matrix. An upper Hessenberg matrix has zero entries below the sub-diagonal matrix, and a lower Hessenberg matrix has zero entries above the sub-diagonal matrix. HR is a major step involved in finding the eigen values of a matrix [73]. This is an important reduction used in many of the high performance computing applications. A hardware for HR is discussed in [12]. One of the major design flaws of the hardware is that the dataset is stored on the local memory and not on the main memory (DDR). This would not enable the design to scale for larger matrices. The computation is done in a sequential fashion. Any improvements in the algorithm can not be accommodated unless the hardware is redesigned. Table 5.17 shows the design guidelines that have been followed. The design consumes only 63% of the FPGA, and the local memory access could have been increased by 30%. The performance depends on the resource utilization and the memory access; and thus, the peak theoretical performance can be calculated as $1.3/0.63 = 2.06$, and the performance compared to the peak theoretical performance is computed as $1/2.06 = 0.49$. The scaled performance index $(0.49\times2) - 1 = -0.02$ is plotted on the y-axis of the guideline fitness plot, as shown in Figure 5.26.

5.2.5 Gaxpy - BLAS Routine

A Gaxpy routine is a BLAS level 2 routine, which computes matrix-vector multiplication, and the complexity of the routine is $O(n^2)$ [15]. A Gaxpy hardware built on the FPGA is discussed in [13]. The work was demonstrated on a BEE3 FPGA board that has four V5 LX155T FPGAs. The implementation has 16 processing elements, each computing a 4×4 matrix, and the maximum size of the matrix that can be stored on the on-chip memory is 256×256. Table 5.18 shows the design guidelines that have been followed. The peak theoretical performance at 100 MHz is 200 MFLOPs. There are 16 PEs per FPGA, and with four FPGAs, the peak theoretical performance is

Table 5.17: HR design evaluated using the design guidelines

1	*Arrange and optimize input/output data for all compute blocks*	✘
2	*Adhere to a widely accepted method of computation for arithmetic functions/operations*	✔
3	*Build controller for every not likely to change compute blocks*	✔
4	*Introduce a configurable dataflow path to connect resources of likely to change compute blocks*	✘
5	*Introduce on-chip memory with configurable read/write logic, if necessary*	✔
6	*Introduce dependency indicators to enhance parallel computations, if necessary*	✘
7	*Use microcode to specify dataflow path, memory read/write, and parallel computations*	✘
8	*Maximize resource utilization by improving runtime parallelism*	✔
9	*Forward results between compute blocks/resources, if possible*	✔
10	*Achieve functionality, and optimize design & resources to improve performance*	✘
11	*Maximize computation until maximum memory bandwidth is utilized*	✘
12	*Use a large and real dataset for test cases*	✘

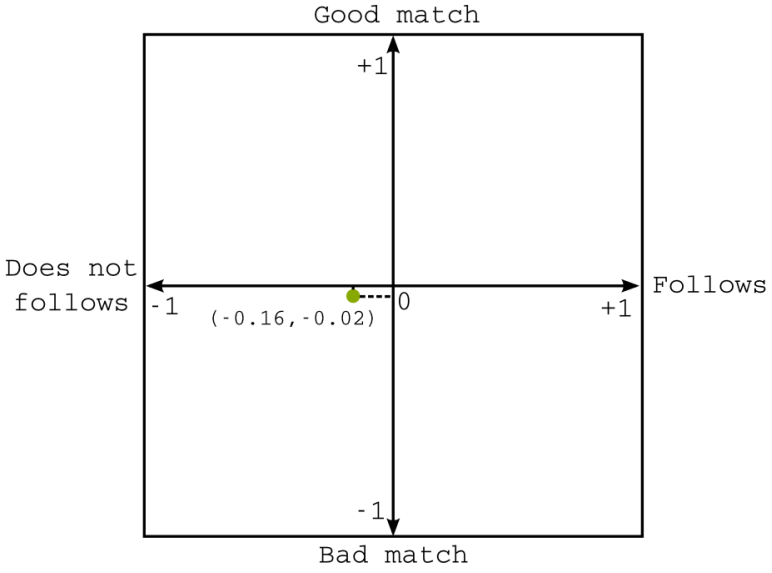

Figure 5.26: Guideline fitness plot for Hessenberg Reduction

Table 5.18: Gaxpy design evaluated using the design guidelines

1	*Arrange and optimize input/output data for all compute blocks*	✘
2	*Adhere to a widely accepted method of computation for arithmetic functions/operations*	✔
3	*Build controller for every not likely to change compute blocks*	✔
4	*Introduce a configurable dataflow path to connect resources of likely to change compute blocks*	✘
5	*Introduce on-chip memory with configurable read/write logic, if necessary*	✔
6	*Introduce dependency indicators to enhance parallel computations, if necessary*	✘
7	*Use microcode to specify dataflow path, memory read/write, and parallel computations*	✘
8	*Maximize resource utilization by improving runtime parallelism*	✔
9	*Forward results between compute blocks/resources, if possible*	✔
10	*Achieve functionality, and optimize design & resources to improve performance*	✘
11	*Maximize computation until maximum memory bandwidth is utilized*	✘
12	*Use a large and real dataset for test cases*	✘

12.8 GFLOPs, and the reported performance is 3.113 GFLOPs. Thus, the actual performance is $3.113/12.8 = 0.25$ of the theoretical performance. The scaled performance index $(0.25\times2) - 1 = -0.5$ is plotted on the y-axis of the guideline fitness plot, as shown in Figure 5.27.

5.2.6 N-Body Simulations

A N-body computation involves two major computations. The first is the inter-atomic distances between three atoms, followed by inter-atomic force computations for those atoms whose inter-atomic distances are less than the cut-off distance. An N-Body hardware is built on the FPGA and is discussed in [14]. The design uses a generic format for computation and utilizes only half the resources. Table 5.19 shows the design guidelines that have been followed. The performance can be improved by increasing the frequency to 75 MHz. The reported performance is 3.9 GFLOPs. Thus, the performance improvement due to frequency improvement is computed as $3.9\times(75/65) = 4.5$ GFLOPs. The design uses only 50% of the resources. The performance improvement by implementing a parallel design is calculated as $4.5\times2 = 9.0$

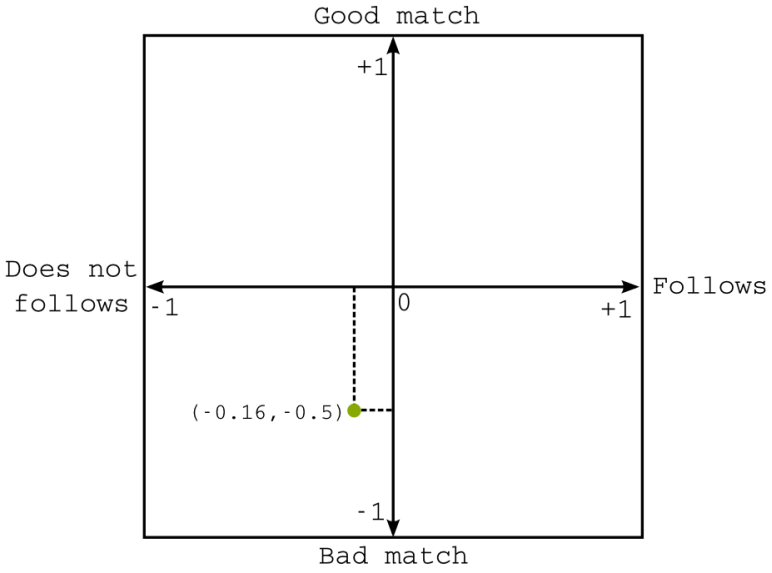

Figure 5.27: Guideline fitness plot for Gaxpy - BLAS Routine

GFLOPs. The performance compared to the theoretical peak performance is computed as 3.9/9.0 = 0.43. The scaled performance index (0.43×2) − 1 = −0.13 is plotted on the y-axis of the guideline fitness plot, as shown in Figure 5.28.

5.2.6.1 Observations

The combined plot is shown in plot 5.29, and all the data points fall either in the first or the third quadrant. When the data point of the applications fall in the first quadrant, 50% or more guidelines were followed and the performance was 50% or better. However, when less than 50% guidelines were followed, the performance was 50% or less. Hence, the guidelines have an impact on the performance of the application, and this is evident from the combined guideline fitness plot. This shows that the design guidelines have an impact on the application's performance, and also according to the evaluation criteria discussed in section 4.3.1, the set of design guidelines is applicable to wide variety of computational science applications.

Table 5.19: N-body design evaluated using the design guidelines

1	*Arrange and optimize input/output data for all compute blocks*	✔
2	*Adhere to a widely accepted method of computation for arithmetic functions/operations*	✘
3	*Build controller for every not likely to change compute blocks*	✔
4	*Introduce a configurable dataflow path to connect resources of likely to change compute blocks*	✘
5	*Introduce on-chip memory with configurable read/write logic, if necessary*	✔
6	*Introduce dependency indicators to enhance parallel computations, if necessary*	✘
7	*Use microcode to specify dataflow path, memory read/write, and parallel computations*	✘
8	*Maximize resource utilization by improving runtime parallelism*	✘
9	*Forward results between compute blocks/resources, if possible*	✔
10	*Achieve functionality, and optimize design & resources to improve performance*	✘
11	*Maximize computation until maximum memory bandwidth is utilized*	✘
12	*Use a large and real dataset for test cases*	✘

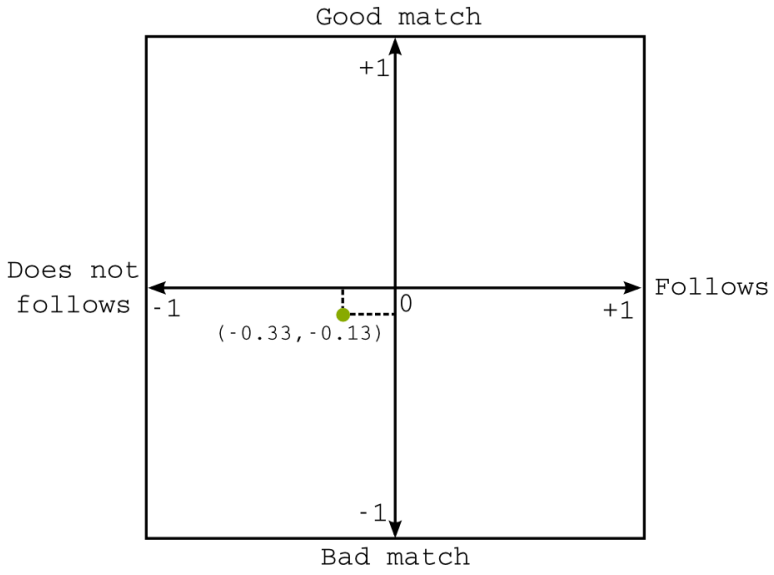

Figure 5.28: Guideline fitness plot for N-Body Simulations

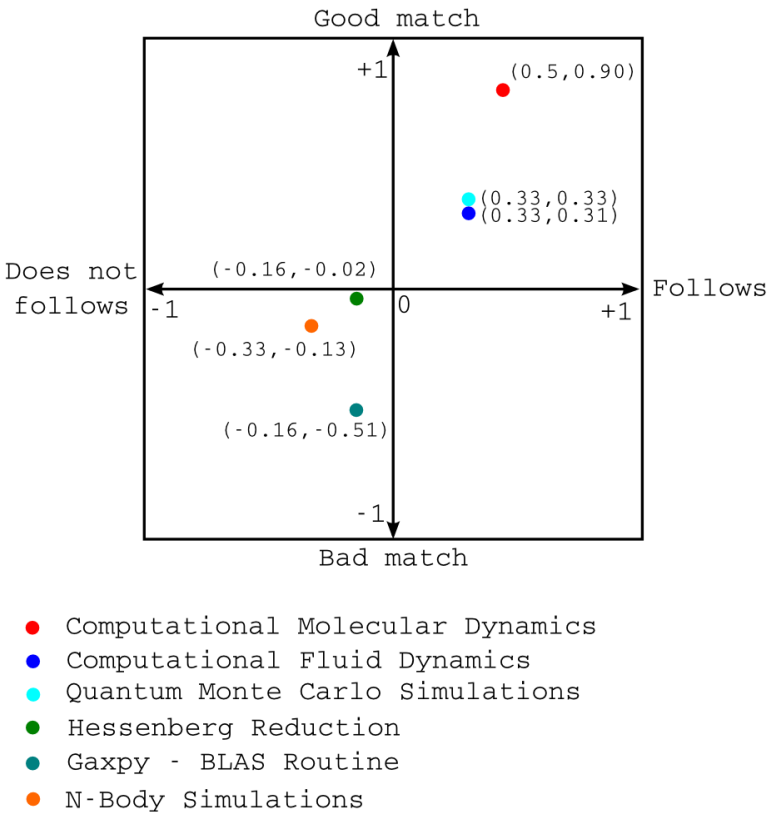

Figure 5.29: Combined fitness plots for above six applications

CHAPTER 6: CONCLUSION

To conclude, if the performance and resource utilization data points for GFHD and LFHD track each other, and the lines of code diverge(do not track) for GFHD and LFHD, then the degree of effectiveness is further used to affirm or deny the thesis question. On the contrary, if the performance and/or resource utilization data points do not track, and/or the lines of code data points track, then we deny the thesis question.

If the plots show effectiveness, the degree of effectiveness is then calculated by classifying the effectiveness into bad, good, or excellent. If the performance and lines of code changed are classified as good or excellent, and the resources as bad, good, or excellent for all the applications and the SpMV kernel, then we can ascertain that the proposed set of guideline helps designers to effectively accommodate the evolving changes in the living computational science application. On the other hand, if the performance and lines of code changed are classified as bad, then the proposed set of guidelines does not effectively accommodate the evolving changes in the living computational science application.

The results section clearly states the observations made for each application. The lines of code diverge, the resources and the performance track for all the application and the matrix multiplication operation. Further, the classification of effectiveness, resource used, and performance is either good or excellent. This shows that the set of design guidelines are effective for living computational science applications. Similarly, the combined guideline fitness plot show that, for all the applications, the score falls into either the first or third quadrant. This shows that the set of design guidelines are broadly applicable to computational science applications. Thus the thesis question is

affirmative, and the set of design guidelines help scientists to improve the productivity as code evolves. These set of design guidelines can be further used to formulate design guidelines for other hardware accelerator technologies.

REFERENCES

[1] H. Mekki, A. Mellit, S. A. Kalogirou, A. Messai, and G. Furlan, "FPGAbased implementation of a real time photovoltaic module simulator," *Progress in Photovoltaics: Research and Applications*, vol. 18, no. 2, pp. 115 – 127, Mar. 2010. [Online]. Available: http://onlinelibrary.wiley.com/doi/10.1002/pip.950/abstract

[2] A. Mellit, H. Mekki, A. Messai, and H. Salhi, "FPGA-based implementation of an intelligent simulator for stand-alone photovoltaic system," *Expert Systems with Applications*, vol. 37, no. 8, pp. 6036 – 6051, Aug. 2010. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0957417410001624

[3] R. Culley, A. Desai, S. Gandhi, S. Wu, and K. Tomko, "A prototype FPGA finite-difference time-domain engine for electromagnetics simulation," in *48th Midwest Symposium on Circuits and Systems, 2005.* IEEE, 2005, pp. 663 – 666 Vol. 1.

[4] W. Chen, "FPGA implementation of 2D FDTD algorithm," Master's thesis, Northeastern University, Boston, Massachusetts, Jul. 2003.

[5] ——, "Acceleration of the 3D FDTD algorithm in fixed-point arithmetic using reconfigurable hardware," Ph.D. dissertation, Northeastern University, Boston, Massachusetts, Aug. 2007.

[6] L. Zhuo and V. K. Prasanna, "Sparse Matrix-Vector multiplication on FPGAs," in *Proceedings of the 2005 ACM/SIGDA 13th international symposium on Field-programmable gate arrays*, ser. FPGA '05. New York, NY, USA: ACM, 2005, pp. 63 – 74. [Online]. Available: http://doi.acm.org/10.1145/1046192.1046202

[7] D. Gregg, C. Mc Sweeney, C. McElroy, F. Connor, S. McGettrick, D. Moloney, and D. Geraghty, "FPGA-Based sparse matrix vector multiplication using commodity dram memory," in *Field Programmable Logic and Applications, 2007. FPL 2007. International Conference on*, 2007, pp. 786 – 791.

[8] G. Kuzmanov and M. Taouil, "Reconfigurable sparse/dense matrix-vector multiplier," in *International Conference on Field-Programmable Technology, 2009. FPT 2009.* IEEE, Dec. 2009, pp. 483 – 488.

[9] K. Sano, T. Iizuka, and S. Yamamoto, "Systolic architecture for computational fluid dynamics on FPGAs," in *Field-Programmable Custom Computing Machines, 2007. FCCM 2007. 15th Annual IEEE Symposium on*, april 2007, pp. 107 – 116.

[10] M. Chiu and M. C. Herbordt, "Molecular dynamics simulations on high-performance reconfigurable computing systems," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 3, pp. 23:1 – 23:37, November 2010.

[11] A. Gothandaraman, G. D. Peterson, G. L. Warren, R. J. Hinde, and R. J. Harrison, "FPGA acceleration of a quantum Monte Carlo application," *Parallel Comput.*, vol. 34, pp. 278 – 291, May 2008.

[12] L. W. Miaoqing Huang and T. El-Ghazawi, "Accelerating double precision floating-point hessenberg reduction on FPGA and multicore architectures," in *Symposium on Application Accelerators in High Performance Computing, 2010, SAAHPC'10*, Knoxville, Tennessee, USA, July 2010, pp. 48 – 51.

[13] S. Kestur, J. Davis, and O. Williams, "BLAS comparison on FPGA, CPU and GPU," in *VLSI (ISVLSI), 2010 IEEE Computer Society Annual Symposium on*, july 2010, pp. 288 – 293.

[14] G. Lienhart, A. Kugel, and R. Manner, "Using floating-point arithmetic on FPGAs to accelerate scientific N-Body simulations," in *10th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, 2002. Proceedings.* IEEE, 2002, pp. 182 – 191.

[15] G. H. Golub and J. M. Ortega, *Scientific computing: an introduction with parallel computing*. USA: Academic Press, 1993.

[16] R. Pool, "The third branch of science debuts," *Science*, vol. 256, no. 5053, pp. 44 – 47, 1992. [Online]. Available: http://www.sciencemag.org/content/256/5053/44.short

[17] 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, May 1966.

[18] C. West and J. Deturk, "A digital computer for scientific applications," *Proceedings of the IRE*, vol. 36, no. 12, pp. 1452 – 1460, dec. 1948.

[19] D. A. Patterson and J. L. Hennessy, *Computer Organization and Design*. USA: Morgan Kaufmann, 2009.

[20] G. Hager and G. Wellein, *Introduction to High Performance Computing for Scientists and Engineers*, ser. Computational Science. Taylor & Francis, 2010.

[21] A. Grama, *Introduction to parallel computing*, ser. Pearson Education. Addison-Wesley, 2003.

[22] V. Kindratenko, "Novel computing architectures," *Computing in Science Engineering*, vol. 11, no. 3, pp. 54 – 57, may-june 2009.

[23] C. Edwards, "Computing - into the fastlane," *Engineering Technology*, vol. 2, no. 1, pp. 36 – 39, jan. 2007.

[24] Y. Maimaitijiang, M. Roula, K. Sobeihi, S. Watson, R. Williams, and H. Griffiths, "A parallel implementation of a magnetic induction tomography: Image reconstruction algorithm on the ClearSpeed advance accelerator board," in *Image Processing (ICIP), 2009 16th IEEE International Conference on*, nov. 2009, pp. 1757 – 1760.

[25] D. Scarpazza, O. Villa, and F. Petrini, "High-speed string searching against large dictionaries on the Cell/B.E. processor," in *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*, april 2008, pp. 1 – 12.

[26] D. Thomas and W. Luk, "Using FPGA resources for direct generation of multivariate gaussian random numbers," in *Field-Programmable Technology, 2009. FPT 2009. International Conference on*, dec. 2009, pp. 344 – 347.

[27] M. Xu and P. Thulasiraman, "Finite-difference time-domain on the Cell/B.E. processor," in *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*, april 2008, pp. 1 – 8.

[28] J. Gustafson, "The quest for linear equation solvers and the invention of electronic digital computing," in *Modern Computing, 2006. JVA '06. IEEE John Vincent Atanasoff 2006 International Symposium on*, oct. 2006, pp. 10 – 16.

[29] T. Ebisuzaki, T. Fukushige, M. Taiji, J. Makino, D. Sugimoto, T. Ito, S. Okumura, E. Hashimoto, K. Tomida, and N. Miyakawa, "GRAPE: special purpose computer for simulations of many-body systems," in *System Sciences, 1993, Proceeding of the Twenty-Sixth Hawaii International Conference on*, vol. i, jan 1993, pp. 134 – 143 vol.1.

[30] G. Shi and V. Kindratenko, "Implementation of NAMD molecular dynamics non-bonded force-field on the cell broadband engine processor," in *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*, april 2008, pp. 1 – 8.

[31] R. Weber, A. Gothandaraman, R. Hinde, and G. Peterson, "Comparing hardware accelerators in scientific applications: A case study," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 22, no. 1, pp. 58 – 68, jan. 2011.

[32] X. Yang, X. Liao, K. Lu, Q. Hu, J. Song, and J. Su, "The TianHe-1A supercomputer: Its hardware and software," *Journal of Computer Science and Technology*, vol. 26, no. 3, pp. 344 – 351, May 2011. [Online]. Available: http://www.springerlink.com/content/h70244371pr727g0/

[33] K. J. Barker, K. Davis, A. Hoisie, D. J. Kerbyson, M. Lang, S. Pakin, and J. C. Sancho, "Entering the petaflop era: The architecture and performance of roadrunner," in *High Performance Computing, Networking, Storage and Analysis, 2008. SC 2008. International Conference for.* IEEE, Nov. 2008, pp. 1 – 11.

[34] R. Larson, J. Salmon, R. Dror, M. Deneroff, C. Young, J. Grossman, Y. Shan, J. Klepeis, and D. Shaw, "High-throughput pairwise point interactions in Anton, a specialized machine for molecular dynamics simulation," in *High Performance Computer Architecture, 2008. HPCA 2008. IEEE 14th International Symposium on*, feb. 2008, pp. 331 – 342.

[35] C. I. Rodrigues, D. J. Hardy, J. E. Stone, K. Schulten, and W.-M. W. Hwu, "GPU acceleration of cutoff pair potentials for molecular modeling applications," in *Proceedings of the 5th conference on Computing frontiers*, ser. CF '08. New York, NY, USA: ACM, 2008, pp. 273 – 282.

[36] S. Tomov, R. Nath, H. Ltaief, and J. Dongarra, "Dense linear algebra solvers for multicore with GPU accelerators," in *Parallel Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on*, april 2010, pp. 1–8.

[37] G. Shi, K. Volodymyr, U. Ivan, M. Todd, P. James, and G. Steven, "Implementation of scientific computing applications on the cell broadband engine," *Scientific Programming*, vol. 17, no. 1, pp. 135 – 151, 2009.

[38] D. Kim, J. D. Trzasko, M. Smelyanskiy, C. R. Haider, A. Manduca, and P. Dubey, "High-performance 3D compressive sensing MRI reconstruction," in *2010 Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*. IEEE, Sep. 2010, pp. 3321 – 3324.

[39] I. Wald, "Fast construction of SAH BVHs on the intel many integrated core (MIC) architecture," *IEEE Transactions on Visualization and Computer Graphics*, vol. 18, no. 1, pp. 47 – 57, Jan. 2012.

[40] Y. Gu, T. VanCourt, and M. Herbordt, "Accelerating molecular dynamics simulations with configurable circuits," in *Field Programmable Logic and Applications, 2005. International Conference on*, aug. 2005, pp. 475 – 480.

[41] M. Herbordt, T. VanCourt, Y. Gu, B. Sukhwani, A. Conti, J. Model, and D. DiSabello, "Achieving high performance with FPGA-based computing," *Computer*, vol. 40, no. 3, pp. 50 – 57, march 2007.

[42] R. Sass, W. Kritikos, A. Schmidt, S. Beeravolu, and P. Beeraka, "Reconfigurable computing cluster (rcc) project: Investigating the feasibility of FPGA-Based petascale computing," in *Field-Programmable Custom Computing Machines, 2007. FCCM 2007. 15th Annual IEEE Symposium on*, april 2007, pp. 127 – 140.

[43] A. Patel, C. Madill, M. Saldana, C. Comis, R. Pomes, and P. Chow, "A scalable FPGA-based multiprocessor," in *Field-Programmable Custom Computing Machines (FCCM), 2006. 14th Annual IEEE Symposium on*, apr 2006, pp. 111–120.

[44] X. Meng and V. Chaudhary, "A high-performance heterogeneous computing platform for biological sequence analysis," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 21, no. 9, pp. 1267 – 1280, sept. 2010.

[45] S. Hauck and A. DeHon, *Reconfigurable computing: the theory and practice of FPGA-based computation*, ser. Systems on Silicon. Morgan Kaufmann, 2008.

[46] R. Pottathuparambil and R. Sass, "Implementation of a CORDIC-based double precision exponential core on an FPGA," in *Proceedings of the Fourth Annual Reconfigurable Systems Summer Institute*, Urbana, Illinois, USA, Jul. 2008, pp. 1 – 4.

[47] ——, "A parallel/vectorized double-precision exponential core to accelerate computational science applications," in *Proceedings of the ACM/SIGDA international symposium on Field programmable gate arrays*, ser. FPGA '09. New York, NY, USA: ACM, 2009, p. 285.

[48] R. Pottathuparambil, R. Adams, and R. Sass, "An FPGA implementation of 3-d FDTD field compute engines," in *Proceedings of the 16th International Symposium on Field-Programmable Custom Computing Machines*, Napa, California, USA, Apr. 2009.

[49] R. Pottathuparambil and R. Sass, "FPGA-based three-body molecular dynamics simulator," in *2010 International Conference on High Performance Computing and Simulation (HPCS)*. IEEE, Jun. 2010, pp. 599 – 605.

[50] R. Pottathuparambil, J. Coyne, J. Allred, W. Lynch, and V. Natoli, "Low-latency FPGA based financial data feed handler," in *2011 IEEE 19th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, May 2011, pp. 93 – 96.

[51] S. Jain, R. Pottathuparambil, and R. Sass, "Implications of memory-efficiency on sparse matrix-vector multiplication," in *2011 Symposium on Application Accelerators in High-Performance Computing (SAAHPC)*. IEEE, Jul. 2011, pp. 80 – 83.

[52] R. Pottathuparambil and R. Sass, "An FPGA-based neural network for computer vision applications," in *FPL 2011 Workshop on Computer Vision on Low-Power Reconfigurable Architectures*, Chania, Crete, Greece, 2011.

[53] A. Kalavade and E. Lee, "Hardware/Software codesign using ptolemy," in *Proceedings of IEEE International Workshop on Hardware/Software Codesign*, Estes Park, Colorado, Sep. 1992.

[54] P. J. Ashenden, *The Designer's Guide to VHDL, Third Edition*, 3rd ed. Morgan Kaufmann, May 2008.

[55] "Open systemc initiative," 2011. [Online]. Available: http://www.systemc.org

[56] C. Cote and Z. Zilic, "Automated systemc to vhdl translation in hardware/software codesign," in *Electronics, Circuits and Systems, 2002. 9th International Conference on*, vol. 2, June 2002, pp. 717 – 720 vol.2.

[57] "Handel-c," 2005. [Online]. Available: http://babbage.cs.qc.edu/courses/cs345/Manuals/HandelC.pdf

[58] "Dime-c," 2006. [Online]. Available: http://www.nallatech.com/Development-Tools/dime-c.html

[59] R. Bruce, M. Devlin, and S. Marshall, "An elementary transcendental function core library for reconfigurable computing," in *Proceedings of the Third Annual Reconfigurable Systems Summer Institute (RSSI'07)*, 2007, pp. 1 – 9.

[60] "Impulse c," 2003. [Online]. Available: www.impulseaccelerated.com

[61] J. Xu, N. Subramanian, A. Alessio, and S. Hauck, "Impulse c vs. VHDL for accelerating tomographic reconstruction," in *2010 18th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, May 2010, pp. 171 – 174.

[62] A. Jain, J. Mao, and K. Mohiuddin, "Artificial neural networks: a tutorial," *Computer*, vol. 29, no. 3, pp. 31 – 44, mar 1996.

[63] M. Gardner and S. Dorling, "Artificial neural networks (the multilayer perceptron)–a review of applications in the atmospheric sciences," *Atmospheric Environment*, vol. 32, no. 14-15, pp. 2627 – 2636, 1998.

[64] A. Ormondi and J. Rajapakse, *FPGA implementations of neural networks*. Springer, 2006.

[65] A. Mellit, H. Mekki, and S. Shaari, "FPGA-based neural network for simulation of photovoltaic array: application for estimating the output power generation," in *33rd IEEE Photovoltaic Specialists Conference, 2008. PVSC '08*. IEEE, May 2008, pp. 1 – 7.

[66] G. Mur, "Absorbing boundary conditions for the Finite-Difference approximation of the Time-Domain Electromagnetic-Field equations," *IEEE Transactions on Electromagnetic Compatibility*, vol. EMC-23, no. 4, pp. 377 – 382, Nov. 1981.

[67] Z. S. Sacks, D. M. Kingsland, R. Lee, and J. Lee, "A perfectly matched anisotropic absorber for use as an absorbing boundary condition," *IEEE Transactions on Antennas and Propagation*, vol. 43, no. 12, pp. 1460 – 1463, Dec. 1995.

[68] S. D. Gedney, "An anisotropic perfectly matched layer-absorbing medium for the truncation of FDTD lattices," *IEEE Transactions on Antennas and Propagation*, vol. 44, no. 12, pp. 1630 – 1639, Dec. 1996.

[69] R. J. CULLEY, "FDTD methods using parallel computations and hardware optimization," thesis, UNIVERSITY OF CINCINNATI, 2007.

[70] W. Chen, P. Kosmas, M. Leeser, and C. Rappaport, "An FPGA implementation of the two-dimensional finite-difference time-domain (FDTD) algorithm," in *12th international symposium on Field programmable gate arrays, 2004. FPGA 2004*. ACM Press, 2004, p. 213.

[71] J. H. Ferziger and M. Peric., *Computational Methods for Fluid Dynamics*. Berlin, Heidelberg: Springer-Verlag, 1996.

[72] D. C. Rapaport, *The Art of Molecular Dynamics Simulation*. New York, NY, USA: Cambridge University Press, 1995.

[73] A. S. Householder, "Unitary triangularization of a nonsymmetric matrix," *Journal of The ACM*, vol. 5, pp. 339 – 342, 1958.