REPORT DOCUMENTATION PAGE

Form Approved OMB NO. 0704-0188

1. REPORT DATE (DD-MM-YYYY)	2. REPORT TYPE	3. DATES COVERED (From - To)
27-12-2017	Final Report	30-May-2016 - 29-Aug-2017

4. TITLE AND SUBTITLE
Final Report: An FPGA Testbed for Characterizing and Mapping DOD Applications

5a. CONTRACT NUMBER
W911NF-16-1-0326

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER
611103

6. AUTHORS

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAMES AND ADDRESSES
North Carolina A&T State University
1601 East Market Street

Greensboro, NC 27411 -0001

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS (ES)
U.S. Army Research Office
P.O. Box 12211
Research Triangle Park, NC 27709-2211

10. SPONSOR/MONITOR'S ACRONYM(S)
ARO

11. SPONSOR/MONITOR'S REPORT NUMBER(S)
68589-CS-RIP.3

12. DISTRIBUTION AVAILIBILITY STATEMENT
Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES
The views, opinions and/or findings contained in this report are those of the author(s) and should not contrued as an official Department of the Army position, policy or decision, unless so designated by other documentation.

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	15. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Clay Gloster, Jr.
UU	UU	UU	UU		19b. TELEPHONE NUMBER 336-285-3134

RPPR Final Report
as of 29-Dec-2017

Agency Code:

Proposal Number: 68589CSRIP **Agreement Number: W911NF-16-1-0326**
INVESTIGATOR(S):

Name: Christopher Doss
Email: cdoss@ncat.edu
Phone Number: 3363347760
Principal: N

Name: Clay Gloster, Jr.
Email: cgloster@ncat.edu
Phone Number: 3362853134
Principal: Y

Organization: **North Carolina A&T State University**
Address: 1601 East Market Street, Greensboro, NC 274110001
Country: USA

DUNS Number: 071576482 EIN: 566000007
Report Date: 29-Nov-2017 Date Received: 27-Dec-2017
Final Report for Period Beginning 30-May-2016 and Ending 29-Aug-2017
Title: An FPGA Testbed for Characterizing and Mapping DOD Applications
Begin Performance Period: 30-May-2016 **End Performance Period:** 29-Aug-2017
Report Term: 0-Other

Submitted By: Clay Gloster, Jr. Email: cgloster@ncat.edu
Phone: (336) 285-3134

Distribution Statement: 1-Approved for public release; distribution is unlimited.

STEM Degrees: 3 **STEM Participants:** 12

Major Goals: The widespread use of unmanned autonomous surveillance systems in military operations offers significant advanced new capability while simultaneously introducing complex challenges in the rapid development/deployment and testing of these systems. Current payloads for these systems have stringent constraints with respect to their maximum weight, power consumption, processing speed, sensor types, interface requirements, and overall cost. The traditional approach to the design of these payloads involves repeated system level engineering design steps identifying onboard computers and application specific boards, each supplied by a specific manufacturer. This approach results in systems with closed or highly proprietary architectures that have a very limited potential for reuse. While each board typically enables a particular function for the payload, the problem of meeting payload design specifications is exacerbated by the need to identify manufacturers with interfaces that match the sensors. Hence, for each new system, there is typically at least one complete redesign to meet the dynamic nature of advances in sensor technology.

The solution to many of these issues is to use Field Programmable Gate Arrays (FPGAs) in conjunction with onboard processors as a platform to develop a Context Neutral Payload (CNP) that supports a myriad of sensor types, meets size and weight constraints, provides superior processing power, can serve a dual role as a ground station, and can reduce the data requirements for communication of sensor data to satellites/ground stations. Reconfigurable Computing (RC) systems combine onboard processors with FPGAs to offer programmable hardware systems that can be reloaded for disparate applications while offering 1-2 orders of magnitude speedup.

This proposal involves the basic research required for the development and implementation of a context neutral payload (CNP). The CNP can be reused in various unmanned vehicles (air, ground, or sea) with a wide range of sensors. More specifically, this proposal attempts to leverage the previous work to develop a Remote And Reconfigurable Computing Environment (RARE) while addressing the problem of porting High Performance Computing (HPC) applications directly to FPGA-based architectures. The major goals of the proposed research are to: develop a comprehensive floating point library of essential functions for military applications; demonstrate order of magnitude speedup of reconfigurable computing applications; and train students in the use of RARE tools for rapid military application development.

RPPR Final Report
as of 29-Dec-2017

As an outcome of this research, the institution will be able to demonstrate 10-100X speedup of various HPC algorithms using its novel patented architecture. The institution will develop a RARE resource that can be accessed from a remote site. Experiments conducted by RARE researchers will continue to verify that REMOTE HARDWARE versions of specific computation-intensive algorithms can complete execution significantly faster than LOCAL SOFTWARE versions of the same algorithm. In addition, students will be introduced to research conducted as a part of this grant and encouraged to pursue careers in engineering positions throughout the DOD.

Accomplishments: Recent research shows that a Reconfigurable Computing (RC) System can be used to achieve an order-of-magnitude speedup of High-Performance Computing (HPC) applications. An RC System, within the context of our research, is a host processor connected to one or more Field Programmable Gate Arrays (FPGAs). Typically RC systems provide better performance than conventional processors and consume less energy than Graphics Processing Units (GPUs). RC systems can also be integrated into cloud-computing environments and data centers to enhance the accessibility of these hardware accelerators. RC-based systems are a viable choice for accelerating applications in various areas including bio-informatics, neuroinformatics, physics, computer vision, etc. where significant processing capability is required.

While extensive research is underway to accelerate computationally intensive tasks in HPC applications on RC systems, few studies report frameworks and multi-memory architectures developed to provide access to them from a remote site. The remote environment introduced in this thesis, provided to access RC system accelerators, proves beneficial to users with limited hardware/software knowledge. The remote environment allows users to access RC system accelerators remotely from any part of the world. The remote environment also helps to reduce overall system cost since the users are not required to invest in an expensive RC system design and infrastructure development.

Memory bandwidth is a major factor in developing RC systems with execution times faster than traditional processors. For achieving high performance, RC system processing elements must be able to consume/produce data at the highest possible data rates. If there are multiple memory banks, the number of memory accesses required to bring all data into the RC system processing unit can be significantly reduced. Similarly, outputs can be transmitted in few number of memory write operations. In turn, increasing memory bandwidth will positively impact the overall performance of the RC system accelerator. However, adding multiple memories adds extra hardware to the FCCM, increasing size/complexity, as an interface is needed for each additional memory.

In this project, we present a Java-based environment to access RC system accelerators from a remote site. We have developed a hardware/software interface which supports multiple FPGAs connected to a server. This remote environment allows users with limited knowledge of the underlying hardware/software to take full advantage of RC system accelerators by accessing RC systems in a remote server from a client machine. We also provide a multi-memory architecture accepting multiple inputs and producing multiple outputs per clock cycle. The architecture includes processor cores with pipelined functional units specifically tailored for each application. Additionally, we present an approach to achieve an order-of-magnitude speedup over a traditional software implementation executing on a conventional multi-core processor. Even though the clock frequency of the RC system is an order of magnitude slower than a conventional multi-core processor, the RC system is significantly faster.

This report presents a case study using the Taylor Series to demonstrate the merits of the local FCCM. The Taylor Series is a computationally intensive application used in many HPC applications, i.e., computer vision, cellular networks and deep neural network. In our experiments, we executed the Taylor Series in software and compared execution times with an RC system interfaced to the same machine. Our experiments show that the results obtained using our multi-memory architecture is approximately 481X faster than software executing the Taylor Series on a typical server. We also implemented other computational primitives, i.e., Power, Natural Logarithm and Exponential on a local RC system and compared their execution time with their respective software implementation. The experiments show that, with our multi-memory architecture, we achieved 1-2 orders-of-magnitude speedup over software implementation.

This report also presents results that demonstrate the merits of a remote RC system with the multi-memory architecture. In our experiments, we executed the RC system implementation of the same applications on the remote server machine and transmitted data to be processed from a client machine. Our experiments show that the execution time for the remote RC system is not constrained by the specialized processor, but rather by the network bandwidth. On a network with 300 Mbps upload speed, and 900 Mbps download speed of the Internet, execution of an application on the remote RC system with our remote framework, with two clients and two servers, can achieve

10X speedup over execution of a software implementation of the same application on the local machine.

Training Opportunities: Nothing to Report

Results Dissemination: Dissemination activities of this project involved the participation of two students from underrepresented minority groups in the Undergraduate Research Apprenticeship program. One student, Alexandria Fuller, was from North Carolina A&T State University and the other student, Leslie Hernandez, was from Dartmouth University. A brief description of their project follows.
The Remote And Reconfigurable Environment (RARE) Project Undergraduate Apprenticeship Program

Student Project Description. Two students participated in the RARE undergraduate apprenticeship program. Two undergraduate students worked together to develop applications capable of reverse engineering a software executable program for the RARE project. They implemented a system that processes a given unknown software executable and determines which program, from a database of known programs, is the best match to this executable. For example, the students are given an unknown executable and use their applications to discover that that the unknown program is actually the quicksort program. The system developed by the students also outputs a confidence level (a value between 0 and 100 that tells the confidence at which the two programs match. In this way, students are be able to apply forensics to an unknown executable program and tell whether it is malicious software, a virus, or determine exactly its functionality.

The students attempted to identify an unknown executable program with a system composed of three individual programs. A disassembler, i.e. IDA Pro, along with a script developed by the students will convert the executable program into a text file containing hexadecimal values. This is the actual machine language instructions of the original executable program. The students then used a program that they developed to compute the Fast Fourier Transform (FFT) of the hex code generated. Finally, the students wrote a program that compares the FFT of the unknown algorithm to each entry of a database containing the FFTs of hundreds of known algorithms. The program outputs the closest match from the database along with the confidence that the match is correct.

The activities of this project performed by the students included: planning, creating, testing, and deploying two versions of the system: a software based implementation of the database matching program and an FPGA implementation of the same program. The students compared execution times of software and hardware implementations. These applications have been integrated into the RARE project environment so that they can be executed on a local or remote computer server.

Research Outcomes. As a result of this project, the students were engaged to develop applications or software that can be quickly integrated into the RARE Computing Environment. These students conducted state-of-the-art research that will become an integral part of a DOD-sponsored research project. They gained new programming, web application development, and database management skills. They also gave presentations at weekly research group meetings of faculty and graduate students who are a part of the RARE project team. Deliverables produced by the students include: source code for tools they develop; a manual on how to use these tools; and a programming development document to facilitate future software upgrades.

Honors and Awards: The PI was nominated by the Dean of the School of Technology for the University Intellectual Property Award at the university's research awards dinner sponsored by the Division of Research and Economic Development. Dr. Gloster was the representative for the School of Technology that competed for the award at the university level.

Protocol Activity Status:

RPPR Final Report
as of 29-Dec-2017

Technology Transfer: US Patent #9,111,068 B2
Multiple Memory Application-Specific Digital Signal Processor
August 8, 2015

An integrated circuit device is provided comprising a circuit board and one or more digital signal processors implemented thereon. The digital signal processor comprises a data unit comprising a function core configured to perform a specific mathematical expression in order to perform at least a portion of a specific application and an instruction memory storing one or more instructions configured to send commands to the control unit and the data unit to perform the specific application, and a control unit configured to control the flow of data between a plurality of memory banks and the function core for performing the specific application, and the plurality of memory banks coupled to each of the one or more digital signal processors and comprising at least two or more local memory banks integrated onto the circuit board.

PARTICIPANTS:

Participant Type: Co PD/PI
Participant: Christopher Doss
Person Months Worked: 2.00 **Funding Support:**
Project Contribution:
International Collaboration:
International Travel:
National Academy Member: N
Other Collaborators:

ARTICLES:

Publication Type: Journal Article Peer Reviewed: Y **Publication Status:** 1-Published
Journal: Journal of Circuits, Systems and Computers
Publication Identifier Type: DOI Publication Identifier: 10.1142/S0218126617500153
Volume: 26 Issue: 01 First Page #: 1750015
Date Submitted: 12/27/17 12:00AM Date Published: 1/1/17 10:00AM
Publication Location:
Article Title: A Neuron Library for Rapid Realization of Artificial Neural Networks on FPGA: A Case Study of Rössler Chaotic System
Authors: Ismail Koyuncu, Ibrahim Sahin, Clay Gloster, Namik Kemal Sartekin
Keywords: FPGA; VHDL; artificial neural networks; neuron library;
Abstract: Artificial neural networks (ANNs) are implemented in hardware when software implementations are inadequate in terms of performance. Implementing an ANN as hardware without using design automation tools is a time consuming process. On the other hand, this process can be automated using pre-designed neurons. Thus, in this work, several artificial neural cells were designed and implemented to form a library of neurons for rapid realization of ANNs on FPGA-based embedded systems. The library contains a total of 60 different neurons, two-, four- and six-input biased and non-biased, with each having 10 different activation functions. The neurons are highly pipelined and were designed to be connected to each other like Lego pieces. Chip statistics of the neurons showed that depending on the type of the neuron, about 25 selected neurons can be fit in to the smallest Virtex-6 chip and an ANN formed using the neurons can be clocked up to 576.89MHz. ANN based Rössler system was constructed to ...
Distribution Statement: 1-Approved for public release; distribution is unlimited.
Acknowledged Federal Support: **N**

CONFERENCE PAPERS:

RPPR Final Report
as of 29-Dec-2017

Publication Type: Conference Paper or Presentation **Publication Status:** 1-Published
Conference Name: SoutheastCon 2017
Date Received: 08-Dec-2017 Conference Date: 30-Mar-2017 Date Published:
Conference Location: Concord, NC, USA
Paper Title: An automated Reconfigurable-Computing Environment for accelerating software applications
Authors: Shikant Jadhav, Clay Gloster, Vance Alford, Christopher Doss, Youngsoo Kim
Acknowledged Federal Support: **Y**

A High Performance Computing Testbed for Characterizing and Mapping Military Applications

1 Abstract

Recent research shows that an Field-Programmable Custom Computing Machine (FCCM) can be used to achieve an order-of-magnitude speedup of High Performance Computing (HPC) applications. An FCCM, within the context of our research, is a host processor connected to one or more Field Programmable Gate Arrays (FPGAs). Typically FCCMs provide better performance than conventional processors and consume less energy than Graphics Processing Units (GPUs). FCCMs can also be integrated into cloud-computing environments and data centers to enhance accessibility of these hardware accelerators. FCCMs are a viable choice for accelerating applications in various areas including bio-informatics, neuro-informatics, physics, computer vision, etc., where significant processing capability is required.

While extensive research is underway to accelerate computationally intensive tasks in HPC applications on FCCMs, few studies report frameworks developed to provide access to FCCMs from a remote site. The remote environment introduced in this report, provided to access FCCM accelerators proves beneficial to users with limited hardware/software knowledge. The remote environment allows users to access FCCM accelerators remotely from any part of the world. The remote environment also helps to reduce overall system cost since the users are not required to invest in an expensive FCCM infrastructure and design.

The memory bandwidth is the major factor that impacts performance of the FCCMs. If there are multiple memory banks, the number of memory accesses can be significantly reduced. In turn, increasing memory bandwidth will positively impact the overall performance of the FCCM accelerator. However, adding multiple memories adds extra hardware to the FCCM because an interface is included for each memory. This increases the size and complexity of the FCCM.

In this project, we present a Java-based environment to access FCCM accelerators from a remote site. We have developed a hardware/software interface which supports multiple FPGAs connected to a host processor. This remote environment allows users with limited knowledge of the underlying hardware/software to take full advantage of FCCM accelerators. We also provide a multi-memory framework accepting multiple inputs and producing multiple outputs per clock cycle. The framework also includes processor cores with pipelined functional units. Additionally we present an approach to achieve an order-of-magnitude speedup over a traditional software implementation executing on a conventional multi-core processor. Even though the clock frequency of the FCCM is an order of magnitude slower than a conventional multi-core processor the FCCM is significantly faster.

This report presents a case study using the Taylor series to demonstrate the merits of the remote FCCM with multiple memory banks environment. The Taylor series is a computationally intensive application used in many HPC applications, i.e. computer vision [1], cellular networks [2]and deep neural network [3]. In our experiments we executed the Taylor series in software and using the FCCM interfaced to the same machine. Our experiment shows that the results obtained using our framework is approximately 200X faster than software executing the Taylor series.

2 Introduction to HPC System Implementation

High Performance Computing (HPC) has evolved over the past decade to meet the increasing demand for processing speed. From conceptual design to engineering, from testing design to manufacturing, from weather prediction to medical discoveries, our daily life is becoming more and more dependent on HPC. HPC allows research scientists and engineers to solve complex science, business and engineering problems using applications that require high bandwidth, enhanced net-

working and very high computing power. A highly efficient HPC system requires a high-bandwidth, low-latency network to connect multiple nodes and clusters.

HPC technology is implemented in multidisciplinary areas including: bio-science, geographical data, oil and gas industry modeling, electronic design automation, climate modeling, media and entertainment. HPC systems are also used by researchers in social media, semantics, geology, archeology, materials, urban planning, graphics, genomic, brain imaging, economics, game designing and even music. This list continues to expand as people from various fields get introduced to HPC and integrate their knowledge of HPC in their field. Some of these HPC applications are memory-bound and some of these are compute-bound.

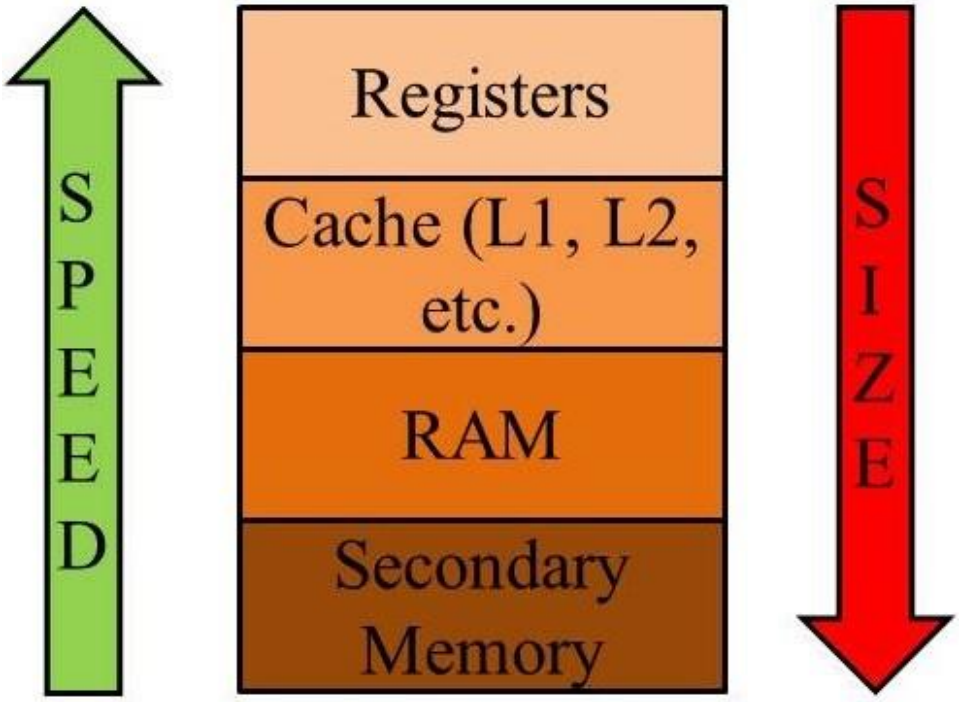


Figure 1: Memory Hierarchy in Computer

The world of technology has witnessed the evolution of the Central Processing Unit (CPU) from the single-core to the multi-core processor. CPUs with multi-core processors are typically used in servers which execute HPC applications. Multi-core processors can execute a different task in each core and hence they are significantly faster than single-core processors.

The memory hierarchy of traditional CPUs is shown in Figure 1. It includes registers, cache, Random Access Memory (RAM) and secondary memory. The registers are on the CPU which

are used to hold variables and temporary results. The size of the register file is very small but since it is located on the CPU it provides the fastest access to the data. The cache is smaller and faster than the RAM. The data that is more frequently accessed is placed in cache so that it can be accessed more quickly than accessing it from main memory. Within the cache there are three levels L1, L2 and L3. L1 typically has fastest access time and L3 typically has slowest access time. The RAM is a volatile memory which is larger and slower than cache memory. The information in RAM is lost when power is turned off. The processor can access the data only when required data is available in the RAM or else required data has to be loaded from secondary memory into the RAM. The secondary memory is slowest, largest and non-volatile memory. It is used to store large volume of data and it can retain the data even if power is turned off.

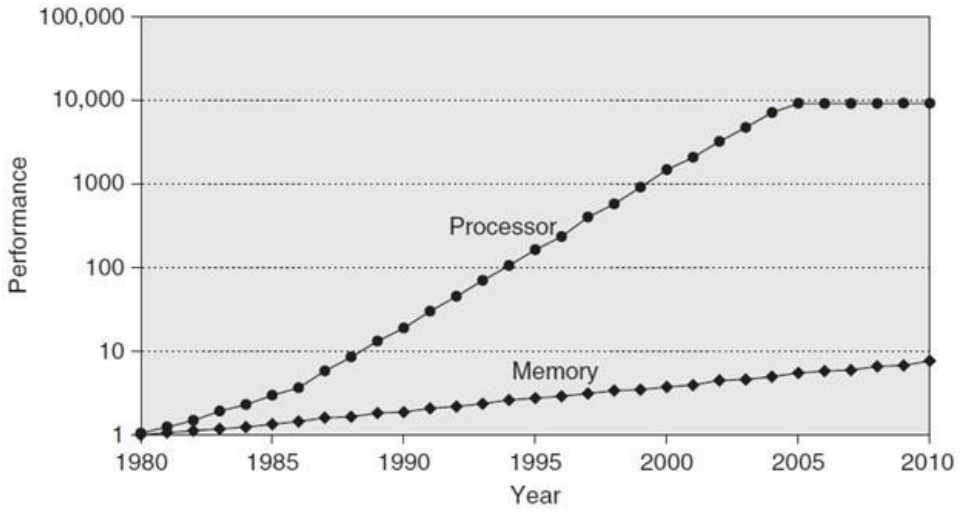


Figure 2: Performance gap between processor and memory

The importance of the memory hierarchy has increased with advances in performance of processors. It must be optimized to enhance the overall performance. Figure 2 shows processor performance from the year 1980 to 2010 2000 [4]. Similarly the figure shows the memory performance for the same period. The gap between the processor performance graph and memory performance graph increases each year. The figure shows that the performance gap between processor and memory was less during 1980-1985 but there is tremendous increase in gap after year.

The increase in demand for computing power has led to the development of heterogeneous systems which are typically composed of conventional processors coupled with a high speed co-

processor, i.e. the Graphics Processing Unit (GPU). The GPU is gaining popularity as a parallel computing platform for accelerating compute intensive applications. Basically the GPU is designed to accelerate applications with many graphics operations. For applications requiring many graphics operations, the GPU operates at a much faster rate than conventional processors. GPUs play a vital role in accelerating applications in engineering, artificial intelligence, robotics, etc. As GPUs have a massively parallel architecture with thousands of cores, applications accelerated with GPUs can achieve speedup over conventional processors.

A GPU is typically included as part of a CPU environment whose primary purpose is to accelerate image processing applications. With the advent of a parallel computing platform and programming model called the Compute Unified Device Architecture (CUDA) the GPUs are now easier to use. Also the development of Open Computing Language (OpenCL), a framework for writing programs that execute across heterogeneous platforms and Open Graphics Library (OpenGL) a standard Application Program Interface (API) has made communication with GPUs far easier.

The literature shows that many research publications on accelerating computationally intensive operations have been published. Currently, research on providing accelerators through cloud environment is also a popular topic. However, few research publications report on techniques used to make these accelerators accessible from remote site. While there are many tools/frameworks that allow users to accelerate compute-intensive applications using CPUs, GPUs and/or FCCMs, few provide frameworks for these systems from a remote site.

In this project we provide a framework to access FCCMs from a remote site. Hence users need only to send their data to the FCCM via the Internet. Experiments shows that for some applications the execution of compute-intensive algorithms on remote hardware can be faster than execution in software on a local machine.

Even though many HPC applications are developed and implemented, there are many problems associated with accelerating HPC applications. There are many HPC applications which still requires weeks and months to execute on a conventional processors. Such applications can be implemented on FCCMs which can reduce execution time to days, hours, minutes and seconds.

Figure 3 presents an FCCM. In our research context we define an FCCM as a host processor

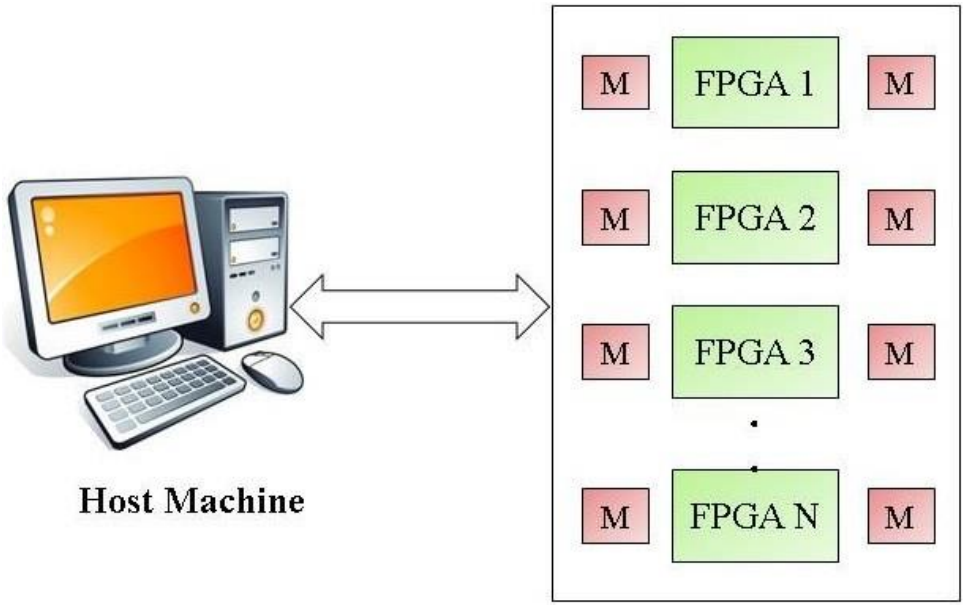


Figure 3: Field-Programmable Custom Computing Machine

attached to one or more FPGAs. Each FPGA is connected to multiple memories on the board. These FPGA boards are connected to the host computer via the high speed PCIe bus. Our approach for accelerating HPC applications is to develop a custom FCCM for each application. We have designed and implemented FCCMs which achieve order-of-magnitude speedup over a software implementation of the algorithm. We have also provided the framework using the custom FCCMs which can be accessed from the remote site.

There are several advantages of using an FCCM as a hardware accelerator. The implementation of computationally intensive operations on an FCCM can yield order-of-magnitude speedup over a software implementation executing on a conventional processor. Depending on the size of an FCCM, multiple pipelined functional cores can be mapped onto the FCCM. Thus, the parallel computing capability of the FCCM achieved with multiple cores helps to enhance the performance even though the clock frequency of an FCCM is an order-of-magnitude slower than a conventional processor. The same FCCM hardware can be reused for diverse applications as it can be reprogrammed an infinite number of times. For example at time *t1* FCCM hardware can be used for an image processing application, at time *t2* the same FCCM hardware can be used for a video processing application, at time *t3* the same FCCM hardware can be used as a remote sensing application, etc.

Even though there are many advantages of using an FCCM, there are few disadvantage too. In order to develop a FCCM, a person needs to be well versed in hardware design as well as software development. The hardware design knowledge is required to implement hardware for computationally intensive algorithms and the software development knowledge is required to access the underlying hardware from a local or remote host. Also development time required to design and implement an FCCM can vary from months to years. This is significantly longer than implementing an application in software on a conventional processor. Since debugging hardware signals is hard than debugging software program, adds up extra time in hardware implementation. Thus FCCM developer need both hardware designing and software development knowledge and can take from months to years for development. Memory bandwidth is also one of the major factors affecting the performance of the FCCM. On-chip memory is faster but is very small and off-chip is larger but has latency involved.

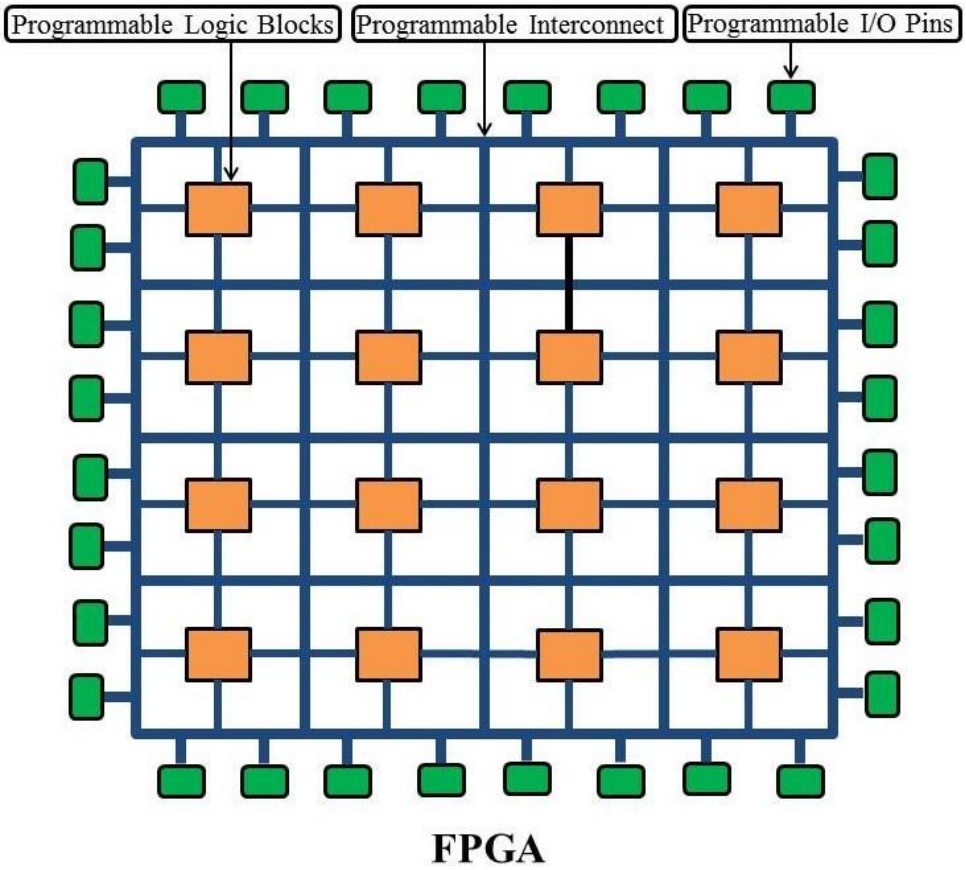


Figure 4: Field Programmable Gate Array

Figure 4 illustrates the architecture of an FPGA. An FPGA architecture includes programmable

logic blocks, programmable interconnect and programmable Input/Output (I/O) blocks. Programmable logic blocks, also called as Configurable Logic Blocks (CLBs), are arranged in a two dimensional grid and are interconnected by programmable routing resources. The logic functions are implemented on the programmable logic blocks. These logic functions are connected together by programmable interconnect. The programmable I/O blocks are arranged at the boundary of the grid. The I/O blocks are connected to the programmable logic blocks through programmable interconnect and makes the off-chip connections.

The FPGAs are programmable/reconfigurable for their ability to implement a different function on the same chip for different applications. FPGAs are used to accelerate many HPC applications like remote sensing, weather modeling, bio-informatics, molecular dynamics, physics, computer vision, etc. FPGAs have become popular because of its potential to include many parallel arithmetic and/or logical units and its memory bandwidth. The multi-core processor is coupled with the FPGA devices for the primary purpose of accelerating computationally intensive applications.

This project attempts to solve the following problem:
Given a computationally intensive application that requires excessive execution time (> several hours) on a conventional processor i.e. bio-informatics, molecular dynamics, network security, weather modeling, remote sensing, etc., find an FCCM for the application that significantly improves the performance and can be accessed remotely.

This project addresses the problems i.e. accessing FCCMs from remote site, increasing memory bandwidth, accelerating HPC applications. The focus of this research is to provide a remote environment which will make FCCM accelerators available to users from remote site. By presenting our flexible and generic Java-based remote environment framework, we are attempting to minimize design effort needed for interfacing between custom hardware accelerators and the host computer. We also providing the multi-memory framework which helps to increase the bandwidth and hence the performance of the FCCM. The remote environment is composed of both a hardware layer and a software layer. The hardware layer consists of a custom hardware accelerator connected to multiple memories on the FCCM. The software layer consists of a framework executing the user request on the FCCM.

The proposed research objectives are: to implement a computationally intensive application on an FCCM to achieve an order of magnitude speedup over a conventional processor, to develop a toolset using which the time required for FCCM development can be reduced from years to months to weeks, to provide toolset so that users who are not knowledgeable in hardware/software design can take advantage of the potential speedup offered by an FCCM, and to provide a remote environment which will allow such users to take advantage of the potential increase in system performance and FCCM hardware re-programmability.

The contributions of this project are: a) remote toolset to access the FCCM from remote site b) interface connecting multiple memories on the board c) 200x speedup using optimized hardware Taylor core over software implementation of Taylor series

3 Related Work

Oliver Knodel, Rainer G. Spallek presents cloud environment consisting virtualized FPGA based hardware accelerator [5]. The authors have proposed Reconfigurable Common Cloud Computing Environment (RC3E) hw and sw framework with virtual FPGAs which is used to execute multiple virtual user designs on one physical FPGA. They have provided cloud environment with three service models: RSaaS, RAaaS and BAaaS. The RC3E hypervisor manages resources and allows the user design to execute on virtual FPGAs. As authors has not provided GUI, users can access cloud service from command line. The RC3E framework incurs overhead in local and remote FPGA status calls. The paper is concluded by providing results of executing matrix multiplication using RC3E framework. The throughput with signal core on physical FPGA is higher than two cores. Thus multiple cores results in communication bottleneck as they share the bandwidth resulting in low throughput.

In [6], an open source stack is proposed which can be used for development of custom FPGA boards. Considering current linux requirements this stack possesses a PCI driver, an IP core for DMA engine, a hardware abstraction library for IO operations and a buffer management library for data transfers between the FPGA board and its application. This stack has been tested for various types of software and hardware. The various building blocks provided by this stack speeds

up the performance for instance the DMA engine IP provides high performance data transfers in PCIe 4-lane boards with Xilinx PCIe cores with the speed of 700 MB/s for write and 380 MB/s for read which is maximum measured performance. The buffer management library allows use of 80-95% of bandwidth with minimum effort and resource consumption. This work has made available the framework consisting of a collection of IP cores and software libraries which makes use of the custom FPGA board simpler because the software cannot use the available bandwidth completely as complexity of the hardware interfaces increases.

The RIFFA 2.0 is a reusable integration framework for FPGA accelerators [7]. The FPGA accelerated applications uses this framework for the communication and synchronization resulting to simple interfaces of hardware and software. In this work the FPGA can be used as an accelerated platform for an open source framework which easily integrates software running on CPUs with FPGA cores. This framework uses PCIe to connect FPGAs to a CPUs system bus. RIFFA 2.0 can be considered as a higher version of RIFFA and which can support more classes of Xilinx FPGAs which are Spartan 6, Virtex 6, ans 7 series. It also supports multiple FPGAs, more PCIe link configurations upto x8 for PCIe Gen 1 and 2, higher bandwidths and also support Linux and Windows operating systems. It also supports C/C++, Java and Python. The highest bandwidth can be achieved. The test results in this work shows that the software can saturate the PCIe link to achieve the highest bandwidth possible.

The design related to interfacing of custom accelerators in the FPGA devices and the host computer through the high speed PCI express interconnections is difficult [8]. To attenuate the design effort a framework has been proposed in this work. This framework encompasses both hardware and software. The hardware layer in the FPGA device consists a generic accelerator architecture and a communication and the software layer at the host consists of a device driver and an API to speed up the application development. This work provides an integrated stack comprising of an FPGA base system, a generic and scalable many core accelerator, a device driver and a low level API. The FPGA system has HIB (Host Interface Bridge) that helps in implementing all necessary hardware modules to connect to a custom accelerator to the PCIe interconnection. The generic accelerator consists of a communication and management controller and the basic

building blocks for creating custom accelerators. The device driver interfaces the CPU and the FPGA through a PCIe interconnection and lastly the low level API provides a convenient routine for controlling the accelerators PEs and data transfers between host and accelerator memory. In this work the achieved throughput is 2.3 GB/s between the host CPU and the accelerators global memory.

Hayden Kwok-Hay So et al presents Berkeley Operating system for ReProgrammable Hardware (BORPH) FPGA based hw/sw co-design methodology [9]. The FPGAs are treated as computational resources and the process running on it is termed as hardware process. UNIX inter-process communication mechanism is used to communicate between hardware and software. BORPH object file is used to configure the FPGA and other information about the process running on the FPGA. They have used five Xilinx Virtex II pro xc2vp70 FPGAs and communications between peripherals is done via OPB and PLB. They have measured the performance of their system by computing the time required for reading and writing from the on-chip memory on a FPGA. The author has also presented a case study on real time wireless signal processing using their hw/sw co-design system.

Robert Brzoza-Woch, Piotr Nawrocki propose a FPGA-based web services for performing operations[10]. The authors presented a system architecture that increase the utilization of unused resources of FPGA which are used for short period of time. The utilization of unused resources can be used for performing other tasks and can be offered as web services. The FPGA-based web services can be connected to using LAN or can be accessed using VPN. Proxy Service in service management subsystem is used to provide access to the local FPGAs. Several Proxy Services and control logic is connected to the Enterprise Service Bus (ESB) which provide simultaneous access to several clients. The FPGA-based web service is implemented using SOA concepts. After careful analysis between WS* (SOAP) and RESTful web service, authors opted for WS* (SOAP) to implement FPGA-based web service. WS* (SOAP) web service based on HTTP defines the service in Web Service Description Language (WSDL). The drawback of the architecture is that they lack the algorithm which will allow the continuous web service operation by automatically moving computational tasks between FPGA-based web services and service management subsystem.

Stuart Byma et al. provide a framework based on OpenStack cloud computing[11]. The authors have used same system and commands that allows Virtual Machines (VM) boot into cloud computing system. But instead of VM they have used Virtualized FPGA Resources (VFR) to boot into virtualized FPGA-based hardware accelerators cloud system. According to authors boot time of VM is 71 seconds and that of VFR is just 2.6 seconds. OpenStack resource management concepts are used to provide virtualized FPGA as resources to users through OpenStack. Partial Reconfiguration (PR) is used to virtualize the FPGA. Instead of providing whole FPGA as resource they divided FPGA into several reconfigurable regions using PR. Each reconfigurable region is managed by each VFR. As there are several OS images in regular cloud computing, in FPGA-based cloud computing there are several PR bitstream images. The image of bitstream corresponding to user hardware is forwarded to Agent which helps to reconfigure the bitstream to available reconfigurable region. In this system there is possibility that error may occur during bitstream configuration. This may happen because of the packet being transferred during reconfiguration. This may lead to data loss and cause error in user hardware.

4 Research Approach

The primary research objective of the Remote And Reconfigurable Environment (RARE) group is executing high performance computing applications using FCCM integrated remote servers. Within this scope, we also conduct minor research projects that are of similar value. To assist with these tasks, we have designed and actively utilized our own custom floating point library. The RARE floating-point library is composed of hardware designs specified for floating-point arithmetic cores which include: adders, subtractors, dividers, multipliers, square root units, exponential units, logarithm units, and a power (y^x) unit. Mapping and implementing these hardware components on FCCMs serve as the underlying foundation for executing HPC applications. Our arithmetic cores are also used for HPC applications including high dynamic range imaging, weather prediction, synthetic aperture radar, and many other tasks.

Figure 5 shows the configuration of a local FCCM. In a local FCCM, the FPGA board is connected to the motherboard via a high speed PCIe bus. The APIs are used in host software to

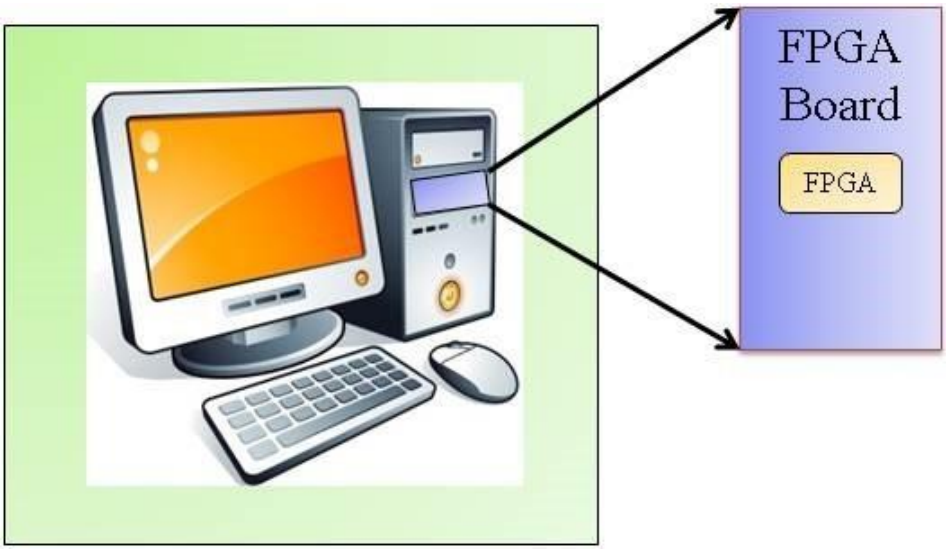


Figure 5: Local FCCM configuration

communicate with the FPGA board.

Figure 6 represents the design for a multi-memory Application-Specific Digital Signal Processor (ASDSP) [12]. The core components of this ASDSP framework include a control unit, data unit, internal instructional memory as well as external memory and a user specified functional core. The ASDSP block serves as a customized high performance processor that executes a single application. Scalable memory allows for reading and writing data synchronously without waiting for any process to finish. These components provide the infrastructure of a standard reconfigurable system capable of performing custom tasks while maintaining the ability to change functions at any given time.

The ASDSP multi-memory architecture interfaces to 64-bit memories. Because of the multi-memory architecture, the number of memory accesses is greatly reduced. A single memory access is required to fetch two 32-bit single precision floating point operands from the 64-bit memory. In addition, operands stored in different memories can be fetched simultaneously. For instance, an ASDSP with four, 64-bit, memories can access eight, 32-bit operands in a single memory access.

Figure 7 presents the example of a typical floating point function core. The Enable and Done pins are used to control function cores. The Enable pin indicates when there is valid data on the inputs and the Done pin indicates when there is valid data on the output. The n-input function core requires n registers, as the outputs of register files are connected to the inputs of function core. The outputs from registers are shared among all function cores. The function cores concatenated

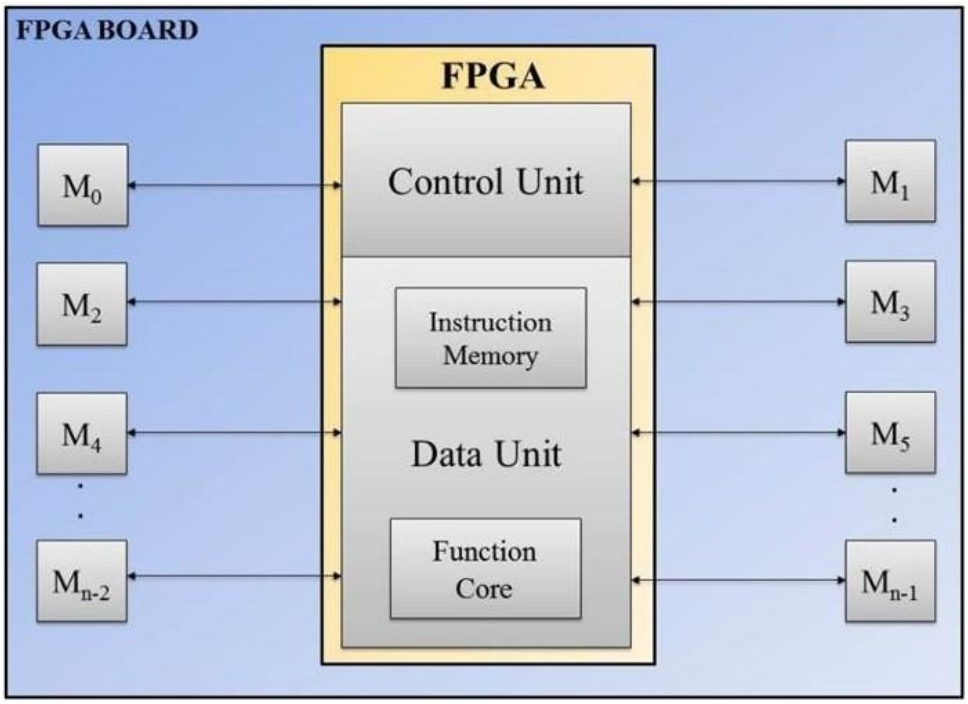


Figure 6: Multi-Memory Architecture

together in order to implement complex functions. For instance as shown in Figure 8 to implement the function Y = (R0*R1) + (R2*R3), three function cores with two input can be concatenated. One of the three function core will compute product (R0*R1), the second will compute the product (R2*R3) and the remaining third will add the products of the first two function cores.

The function cores are used to execute floating point vector operations such as addition, subtraction, multiplication, division and accumulation. The program FUNCOREGEN generates simple netlist describing the interconnection of standard arithmetic units. Their structure includes n-inputs and a single output as shown in Figure 7. All inputs and outputs are 32-bits each.

ASDSP framework contains multiple function cores, which significantly reduced number of cycles required for computation. For instance, ASDSP framework used to compute matrix multiplication contains 8 pipelined multiply-accumulate units whereas traditional processor contains a single pipelined multiply-accumulate unit. Thus number of cycles required to compute matrix multiplication using ASDSP framework will be reduces by factor of eight over using traditional processor to compute matrix multiplication.

Figure 9 shows the block diagram of one input memory and one output memory framework

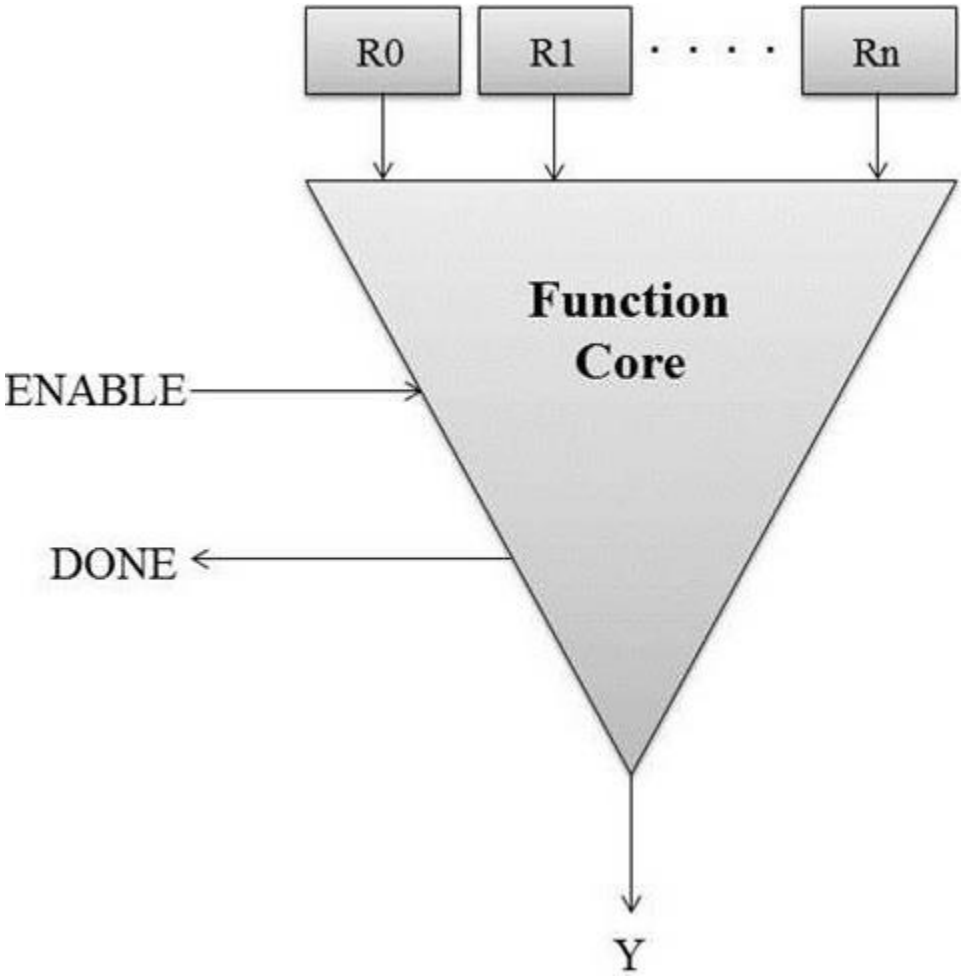


Figure 7: Floating point function core

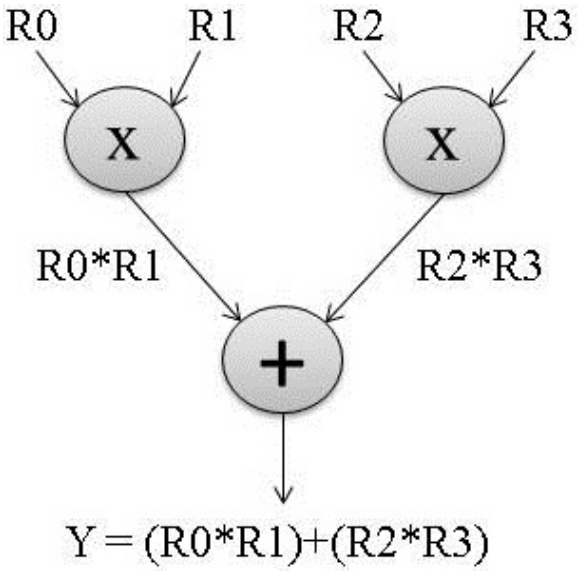


Figure 8: Example of floating point function core

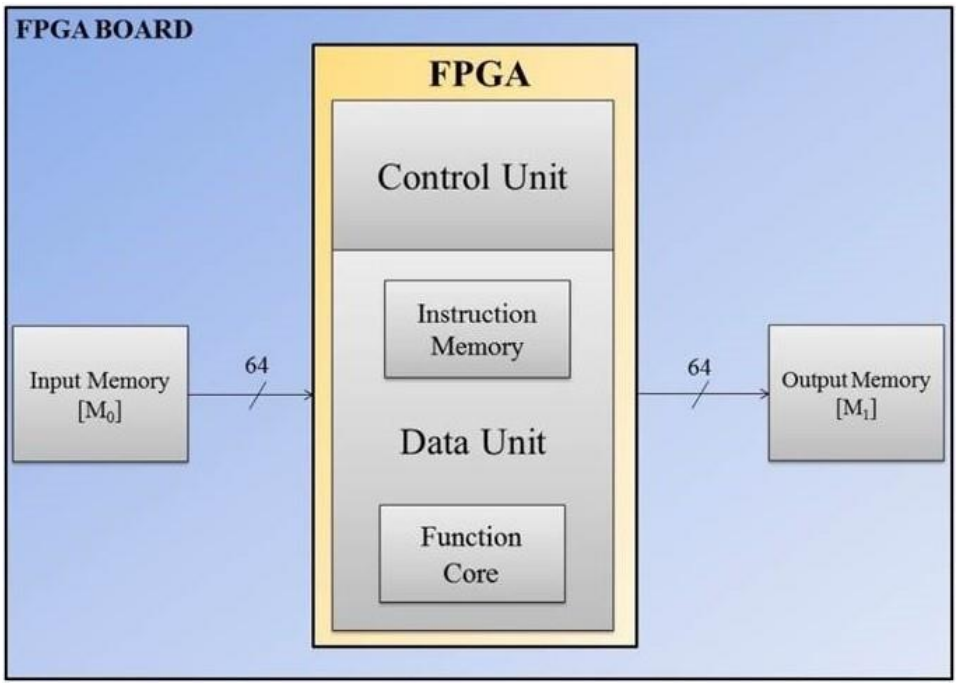


Figure 9: Single Input Memory-Single Output Memory Design

which constitutes control unit and data unit implemented on the FPGA which also contains the digital signal processor. Data unit handle computation part in coordination with control unit. The control unit is finite state machine which controls memory reads and writes, classifies instructions, checks if instruction is complete, send instruction completion signal to host and return control back to host processor. The data unit constitutes standard processor components such as registers, a program counter, an instruction register, memory address registers, counters and function cores. The data unit typically contains multiple function cores as discussed above.

This ASDSP contains two function cores. The ASDSP is able to read and write 64-bit data to the memory. Hence, using two function cores simultaneously with two memories reduces the number of memory accesses and achieves significant speedup over software running on a typical multi-core processor. The ASDSP framework is capable of handling four function cores when there are two input memories and two output memories. The architecture is scalable to handle a large number of input and output memories. However, with each memory comes additional hardware to manage memory accesses.

Case Study: Taylor Series

To prove the efficiency of our framework we have selected Taylor Series as our case study. The reason behind choosing Taylor series is a) computing Taylor series involves intensive calculations. b) the Taylor series is used in some of the HPC applications.

Taylor series have widespread applications in engineering, physics, mathematics, etc. Taylor series are widely used to perform numerical analysis, where it is hard to find analytical solution for given problem. The use of Taylor series is mainly done in numerical techniques for solving differential equations and in problems involving optimization. Taylor series is used in number of real-life applications. It is used as imaging function in fisheye camera which is widely used in robot vision field to capture wide view of the area at one time [13]. Taylor expansion has a role to play in Phasor Measurement Units used for wide area monitoring systems [14]. Taylor series expansion helps to linearize pitch control design for wind turbine using linear quadratic regulator (LQR) [15]. In multi-static radar system Taylor series algorithm is used to evaluate the localization performance of different multi-static configurations of the experimental data [16].

A Taylor series is a representation of a function as an infinite sum of terms about a single point. Taylor Series is represented as follows:

$$\sum_{n=0}^{\infty} \frac{f^{(n)}a}{n!}(x - a)^n \tag{1}$$

where n is the degree.

Trigonometric functions like sin, cos, tan, sec, arcsin, arccos and arctan are represented by using Taylor series. Hyperbolic functions like sinh, cosh, tanh, arsinh and artanh are also represented by using Taylor series. It is also used to compute exponential and natural logarithm of a number. Taylor Series for sin is represented as follows:

$$\sin x = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n+1)!}x^{(2n+1)} = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} \tag{2}$$

$$\sin x = C_0 + C_1X + C_2X^2 + C_3X^3 + C_4X^4 + C_5X^5 + C_6X^6 + C_7X^7 \tag{3}$$

Coefficients of sin function for degree 7 are $C_0 = 0.0$, $C_1 = 1.0$, $C_2 = 0.0$, $C_3 = -\frac{1}{3!} = -0.167$,

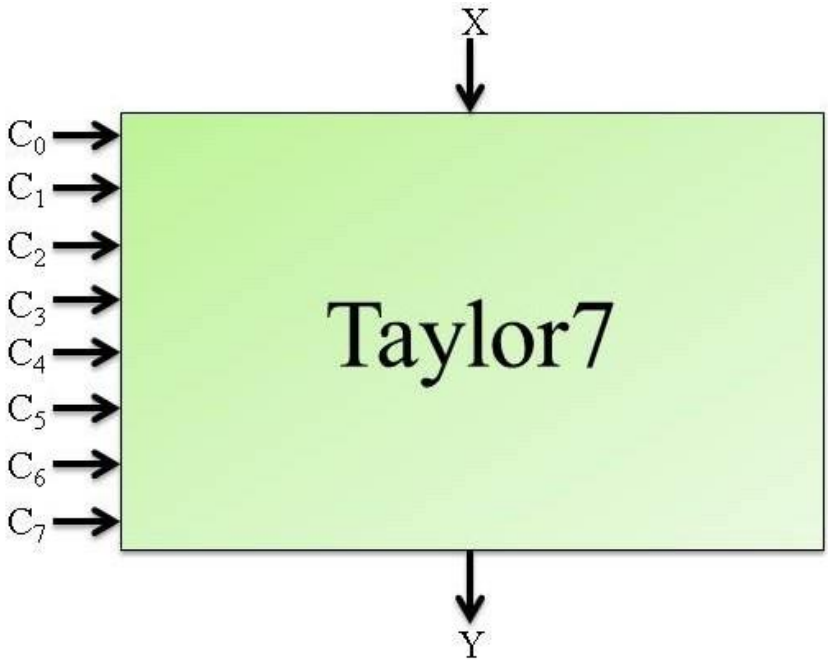
$C_4 = 0.0,\ C_5 = -\frac{1}{5!} = -0.0083,\ C_6 = 0.0,\ C_7 = -\frac{1}{7!} = -1.98 \times 10^{-4}$

Taylor Series for cos is represented as follows:

$$\cos x = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n)!} x^{(2n)} = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^8}{8!} \qquad (4)$$

$$\cos x = C_0 + C_1X + C_2X^2 + C_3X^3 + C_4X^4 + C_5X^5 + C_6X^6 + C_7X^7 + C_8X^8 \qquad (5)$$

Coefficients of cos function for degree 8 are $C_0 = 1.0,\ C_1 = 0.0,\ C_2 = -\frac{1}{2!} = -0.5,\ C_3 = 0.0,$ $C_4 = \frac{1}{4!} = 0.0417,\ C_5 = 0.0,\ C_6 = -\frac{1}{6!} = -0.00139,\ C_7 = 0.0,\ C_8 = \frac{1}{8!} = 2.48 \times 10^{-5}$



$$Y = C_0 + C_1X + C_2X^2 + C_3X^3 + C_4X^4 + C_5X^5 + C_6X^6 + C_7X^7$$

Figure 10: Hardware Design: Taylor Series (n=7)

Figure 10 shows the top level block diagram of our Taylor Series function core. The generated hardware description of a basic Taylor core contains eight coefficients, each 32-bits wide, the input value to be evaluated, *X* and the resulting output, *Y*. Simple control signals allow the function core to be easily interfaced with other basic and complex function cores.

Figure 11 shows the second level block diagram of the custom Taylor Series unit. It consists

of two components: Power unit and the Sum of Product unit. The Power unit computes powers of X at each stage of the pipeline. The Sum of Product unit multiplies the coefficients with the

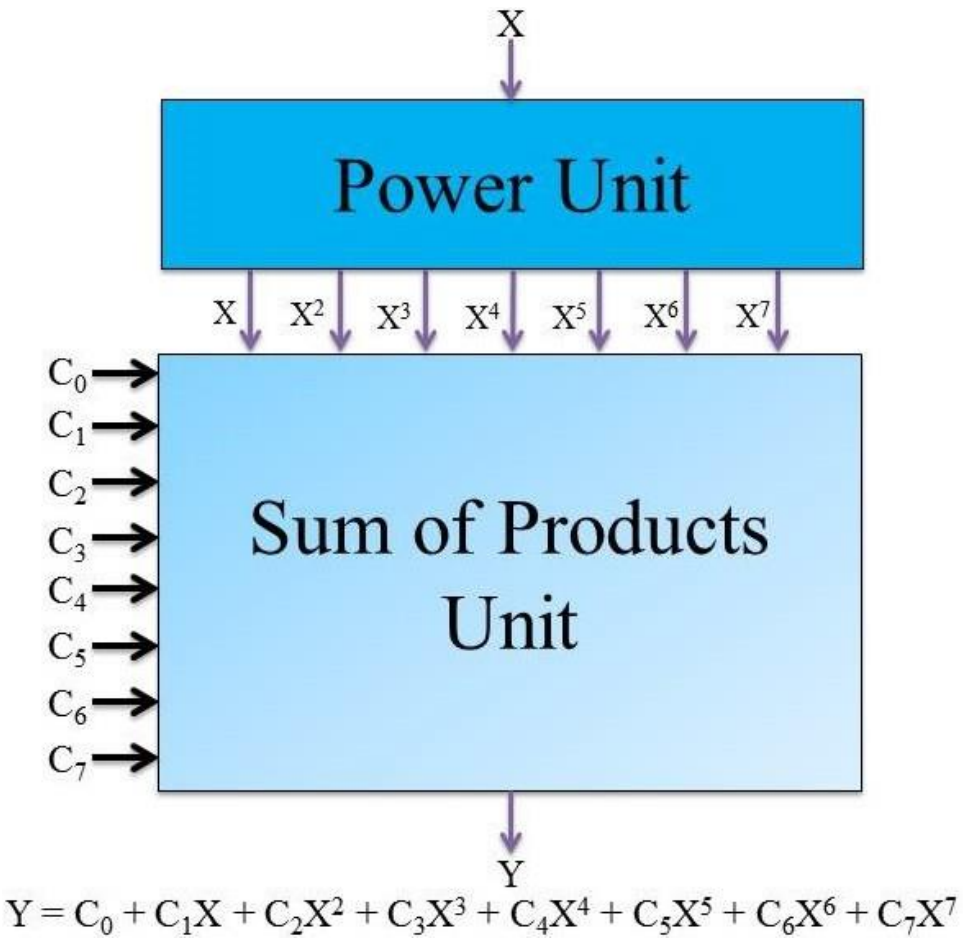


Figure 11: Taylor Series Unit: Second Level Block Diagram

powers of *X* and adds these values together to produce the final output.

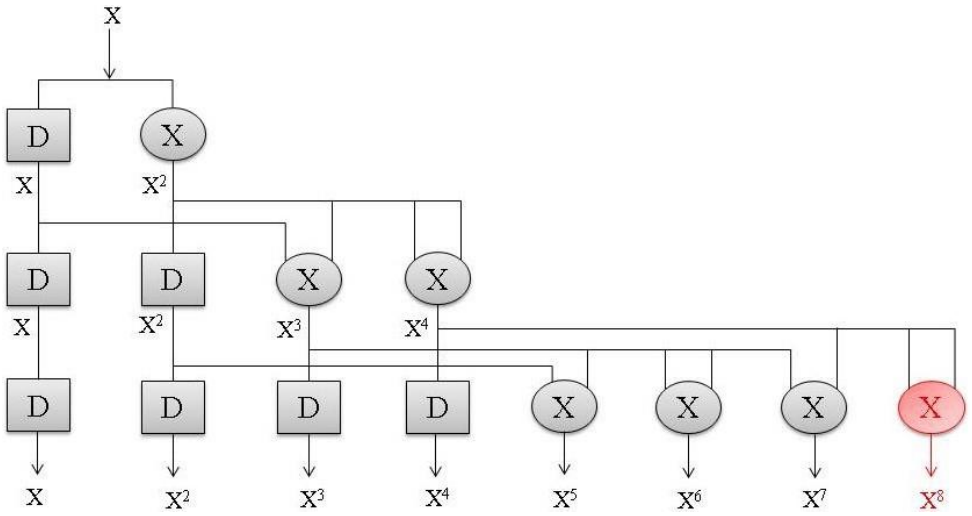


Figure 12: Power Unit (n=7)

Figure 12 shows the components used to make up the power unit. It is made up of floating point multipliers and delay units. At each stage of the pipeline, k powers of *X* are multiplied with *X* to produce 2k powers of *X*. The pipeline generated is fully balanced, thus, all paths from any input to any output goes through the same number of levels. After all of the powers of *X* have been successfully produced/computed they are sent to the Sum unit.

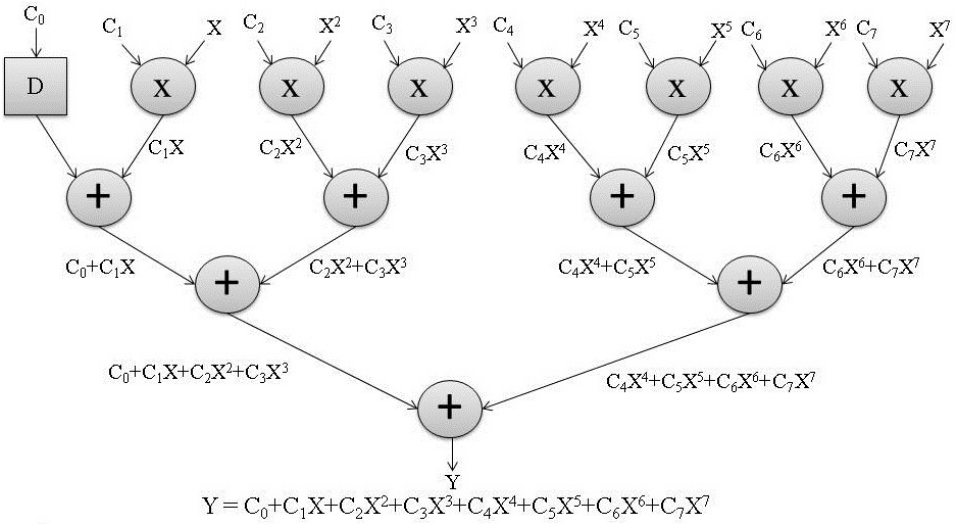


Figure 13: Sum Unit (n=7)

The Sum of Product unit consists of only multiplication and addition computations, with one initial delay module that is always connected to the first coefficient, C_0. As shown in Figure 13,

the first level of the pipeline multiplies all coefficients, except C_0, with the expanded values from the power unit.

The final process of the summation unit is to send the resulting values through an adder tree until the final value is produced. Additional delays are incorporated in the later stages of the adder tree when odd inputs are being passed through on the same level. Serving the same purpose as the delay in the power unit, this temporarily stores the value and balances the pipeline. Similar to the power unit all signals in the summation unit are 32-bits

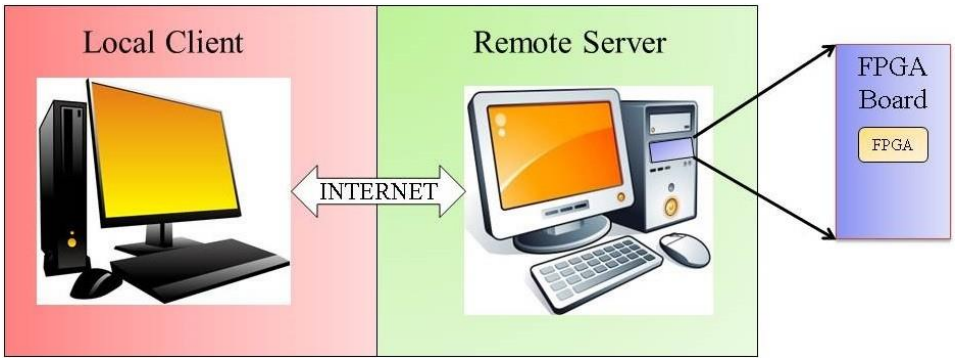


Figure 14: Remote FCCM Configuration

Figure 14 shows the configuration for Remote FCCM. The local client machine connects to remote FCCM server via Internet. The remote server consists of the FPGA board which are used for computation.

Figure 15 shows the configuration specifics of remote FCCM. Using Java Remote Method Invocation (RMI) technology we have implemented client-server wrapper around our FCCM server. Using this system user will be able to send the data file to be processed on the server. Java RMI technology is used to establish connection between client machine and the server machine. The java program on the client invokes the java program on the server through RMI. The user can send data file from remote client to server using Java RMI interface.

Java Native Interface (JNI) programming interface is used, since the host program which communicate with the FPGAs on the RARE server is implemented in C language. The host program communicates with FPGA through API. The JNI interface is required to connect the java program on the server with the native host C code. Thus using JNI programming interface we have implemented the java program on server through which we can call host program implemented

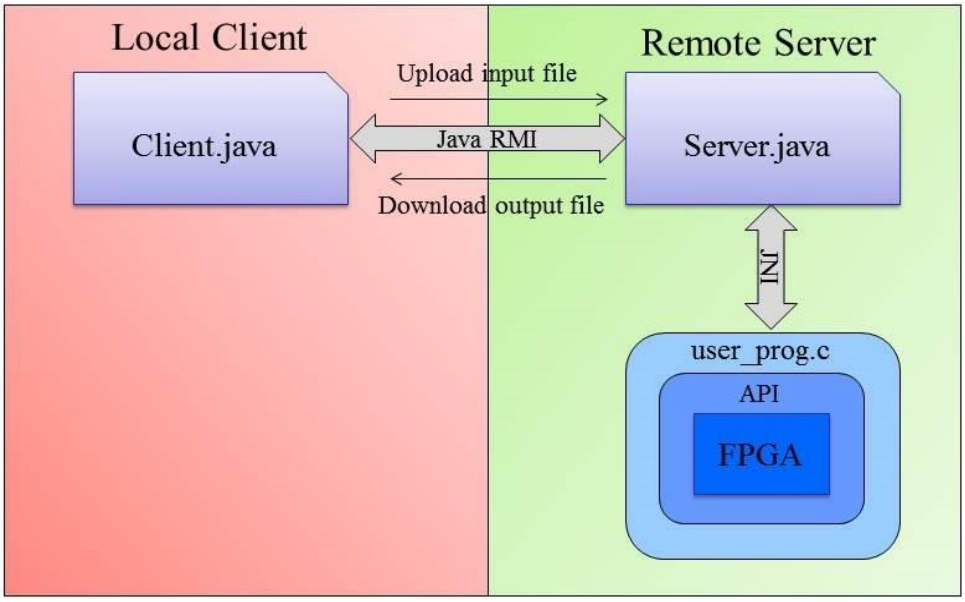


Figure 15: Remote FCCM Configuration Specifics

in C language. The host C program is declared as native function and called from Java JNI source code.

Thus using client java program the user can send input data file to RARE server through RMI. On RARE, the server java program captures the data file. The data file on server is send to host C code through java JNI interface. The host C code sends the data file to FPGA on the RARE server. The data sent by the user is processed on the FPGA and the output is written to the other file. The processed data is sent back to host C code. The host C code sends the data to server through JNI interface. The server sends the processed output file back to remote client machine via RMI.

5 Experimental Results

5.1 Experiment 1: Acceleration framework using local FCCM to achieve speedup over soft-core MicroBlaze processor.

In this experiment we have implemented a local FCCM to compute the natural logarithm value. All operations use floating point data. The hardware implementing natural logarithm unit is

connected as a co-processor to a soft-core processor called a MicroBlaze. Fast Simplex Link (FSL) is used to connect the MicroBlaze processor and co-processor. Our design consumes 28% of the available resources on the FPGA. We compared the time in milliseconds required to compute the natural logarithm on the MicroBlaze soft-core processor verses the combined MicroBlaze - natural logarithm co-processor design. Our FCCM has achieved approximately 527X speedup over MicroBlaze soft-core design.

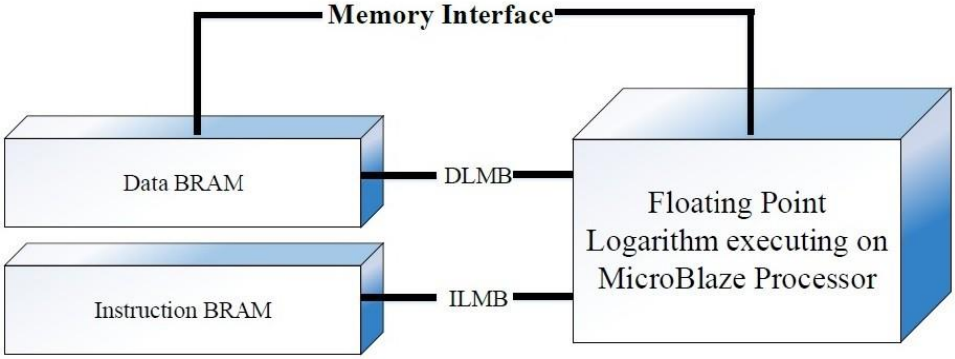


Figure 16: Executing floating point logarithm on the MicroBlaze soft-core processor

Figure 16 shows the MicroBlaze soft-core processor executing the natural logarithm. For the experiment we executed floating point logarithm on the MicroBlaze soft-core processor. We used the results to compare and show the acceleration achieved using co-processor

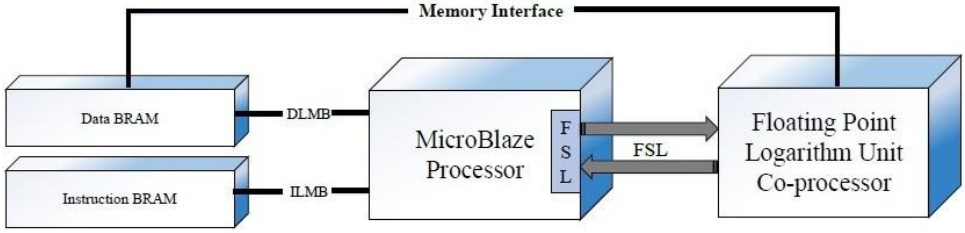


Figure 17: Combined MicroBlaze - natural logarithm co-processor

Figure 17 shows the natural logarithm unit core is connected as a co-processor to the MicroBlaze processor via a dedicated link, FSL. FSL is a 32-bit point-to-point FIFO-based communication protocol. Using a separate co-processor to compute the natural logarithm reduces the number of clock cycles required as compared to computing the natural logarithm on the MicroBlaze soft-core processor. Thus using a separate co-processor reduces the burden on MicroBlaze processor and increases the efficiency and productivity of the MicroBlaze processor.

Table 1: Speedup of the custom HW floating point logarithm on softcore logarithm

# of Samples	Time in MS (Custom HW + MB Softcore)	Time in MS (MB Softcore)	Speedup Achieved
8	2.45×10^3	1.255	512.45
64	1.91×10^2	10.043	525.01
512	1.52×10^1	80.348	527.18
4096	1.21	642.785	527.45
32768	9.74	5142.282	527.49
131072	38.99	20569.129	527.494

Table 1 shows the acceleration achieved by natural logarithm core on the FPGA. The execution time for various number of sample size is reported in milliseconds. The last column of the table shows the acceleration achieved by natural logarithm co-processor over the soft-core MicroBlaze processor. We have achieved the acceleration of approximately 527X.

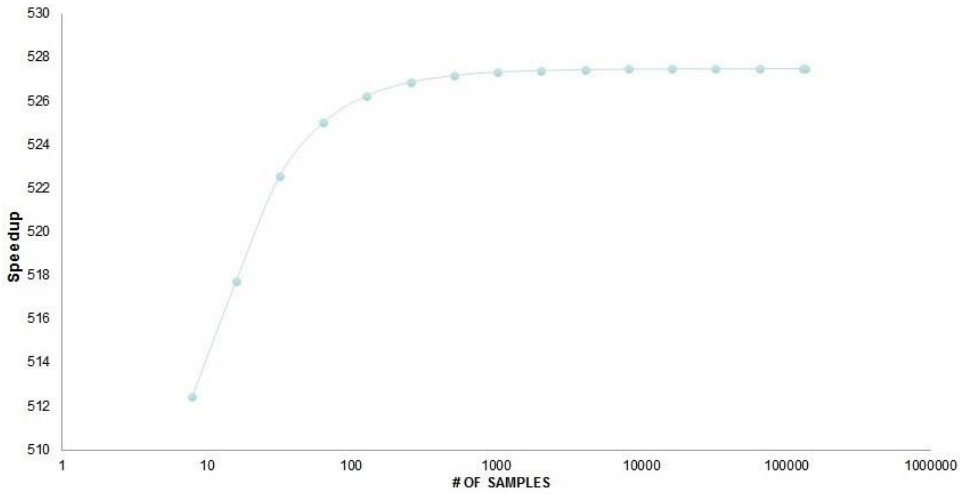


Figure 18: Speedup corresponding to # of samples for the natural logarithm on FPGA

From Figure 18 we can see the graph goes on increasing and at certain point the graph becomes linear. The number of samples are plotted on X-axis of the graph and corresponding speedup is plotted on Y-axis of the graph. The graph shows that after certain amount of samples the acceleration achieved gets stabilized to approximately 527. In our case for the samples 4096 and above the speedup achieved is approximately 527.

5.2 Experiment 2: Design and implementation of local FCCM using multi-memory interface.

In this experiment, we have designed the framework which allows to communicate with multiple memories on the FCCM. For this experiment, we have used the Nallatech PCIe-180 board consisting of five SRAMs of 8 MB each and 64 bits wide and one SDRAM of 512 MB and 256 bits wide. We implemented the framework with two SRAM's connected together. In our framework, one SRAM is used as the input memory and one SRAM is used as output memory as shown in Figure 9. The framework is capable of reading two single precision floating point values from input memory and writing the processed two single precision floating point values to output memory. The Taylor core used as an example core in our design can process one single precision floating point value. Therefore in our design we are using two Taylor cores that can process two single precision floating point values at a time.

Table 2: Taylor Series : Speedup/Execution Time of unoptimized hardware versus software

# of Samples	SW Time in msec	HW Time in msec	Speedup Achieved
1024	1.2269	0.3597	3.4109
2048	2.6034	0.5992	4.3448
4096	3.5436	0.9965	3.5560
8192	4.8801	1.8118	2.6935
16384	8.9916	3.9001	2.3055
32768	17.963	7.6785	2.3394
65536	34.9173	16.6976	2.0912
131072	69.5273	33.8048	2.0567
262144	143.2414	67.3017	2.1283
524288	282.0264	135.303	2.0844
1048576	566.1639	269.6091	2.0999
2097152	1118.6764	539.5125	2.0734

Table 2 shows the results that are obtained using unoptimized framework. In the unoptimized

framework, one value is read from input memory, processed on the Taylor core and written to the output memory at a time. It processes each sample sequentially. We are also not making use of our fully pipelined Taylor core. Since the size of the SRAM is 8 MB, a total of 2^{21} = 2097152 words fits in the SRAM. The software implementation of the Taylor series is executed on the same server which is used to execute the FCCM hardware Taylor series on an FPGA board plugged-in through PCIe bus. The server used for experimental purpose has an Intel(R) Xeon(R) CPU E5-2623 v3 running at 3.00 GHz processor. The column 2 and 3 in Table 2 show the time required in milliseconds for software and hardware Taylor series to execute for varying blocks of sample. The experimental results shows that using unoptimized framework we have achieved approximately 2.5X speedup over software implementation of Taylor series.

Table 3: Taylor Series : Estimated and Actual Speedup/Execution Time of optimized hardware versus software

# of Samples	SW Time in msec	Estimated HW Time in msec	Actual HW Time in msec	Estimated Speedup	Actual Speedup
1024	1.2269	0.002835	0.0097	126.4845	432.7689594
2048	2.6034	0.005395	0.0767	33.9426	482.557924
4096	3.5436	0.010515	0.0113	313.5929	337.0042796
8192	4.8801	0.020755	0.0901	54.1632	235.1288846
16384	8.9916	0.041235	0.0555	162.0108	218.0574754
32768	17.963	0.082195	0.205	87.6244	218.5412738
65536	34.9173	0.164115	0.3442	101.4448	212.7611736
131072	69.5273	0.327955	0.5908	117.6833	212.0025613
262144	143.2414	0.655635	0.777	184.3519	218.477354
524288	282.0264	1.310995	1.4856	189.8401	215.1239326
1048576	566.1639	2.621715	2.8229	200.5611	215.9517339
2097152	1118.6764	5.243155	5.5249	202.4790	213.3593991

Table 3 shows the estimated and actual speedup obtained using optimized framework. The estimated hardware time is computed using core latency. The optimized framework can read, compute and write operations simultaneously. The optimized framework exploits two fully pipelined

Taylor cores. It can be seen from the table that estimated speedup is approximately same as actual speedup. It can be also seen from the tables that results obtained using the optimized framework are significantly faster than unoptimized framework. The experimental results shows that using the optimized framework, we achieved approximately 200X speedup over software implementation of Taylor series.

Table 4: Estimated speedup using additional memories

# of Input Memory	# of Output Memory	Projected Speedup
2	2	400
3	3	600
4	4	800

The Table 4 shows the estimated speedup using additional memories. As we know that latency decreases and bandwidth increases with the maximum number of memories used. From the Table 4, we can see that the projected speedup increases with the increase in number of memories. With the current Nallatech PCIe-180 board we can obtain approximately 800x speedup. This speedup is for one Nallatech PCIe-180 board. The speedup will continue increasing with the number of Nallatech PCIe-180 board used. Our current configuration contains one Nallatech PCIe-180 board, hence we could, in principle achieve a maximum of 800X speedup of a single server.

5.3 Experiment 3: Performance comparison of the ASDSP for polynomial evaluation to conventional multi-core processor

The major emphasis of this study is to demonstrate the effectiveness of the RARE toolset for generating a polynomial evaluation processor. The RARE toolset generates algorithm specific DSPs with execution times that outperform conventional microprocessors while operating at clock frequencies that are an order magnitude slower.

Figure 19 shows the top level block diagram of our polynomial evaluation function core. The

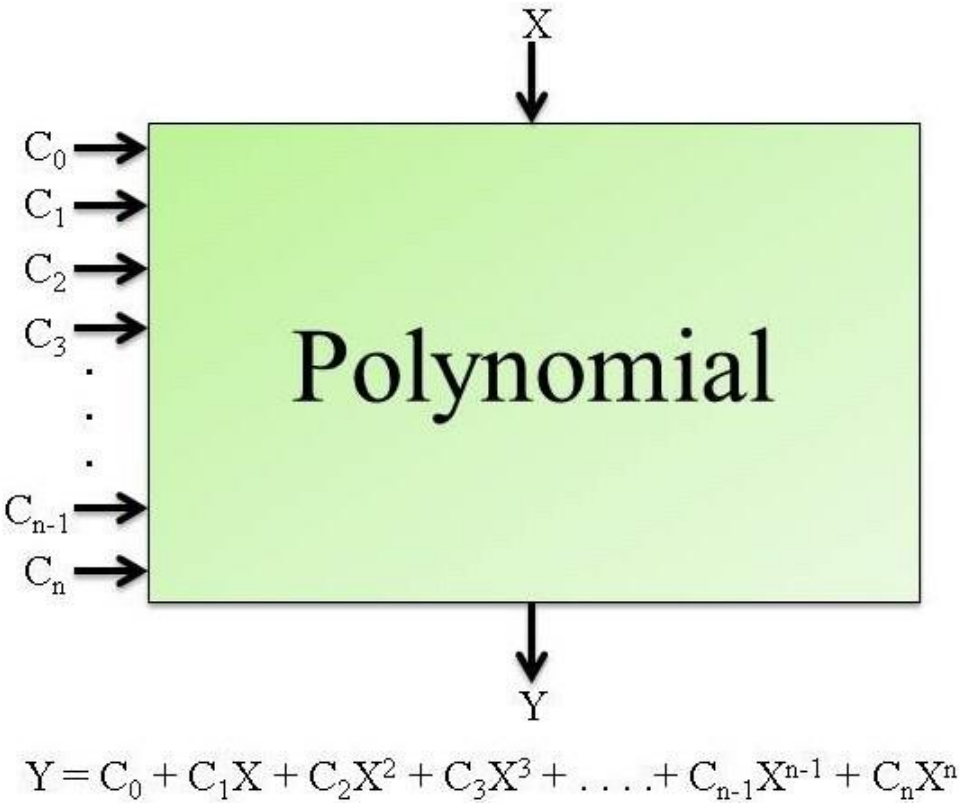


Figure 19: Hardware Design: Polynomial (degree = n)

generated hardware description of a basic polynomial evaluation function core contains n+1 coefficients, each 32-bits wide, the input value to be evaluated, *X* and the resulting output, *Y* . Simple control signals allow the function core to be easily interfaced with other basic and complex function cores.

Figure 20 shows the second level block diagram of the custom polynomial evaluation unit. It consists of two components: Power unit and the Sum of Product unit. The Power unit computes powers of *X* at each stage of the pipeline. The Sum of Product unit multiplies the coefficients with the powers of *X* and adds these values together to produce the final output.

We conducted an experiment to determine the maximum degree (n) for a two polynomial function cores that could fit on our FPGA board. Results for this experiment are shown in Table 5. The table shows the estimated slices versus the actual slices required for two cores stopping at n=20. Since the design for degree n=20 consumes 97.86% of the FPGA resources, this is the maximum degree that can be used for an ASDSP with two cores. Similarly, an ASDSP with four cores has maximum degree of n=10, but would be able to execute twice as many polynomial

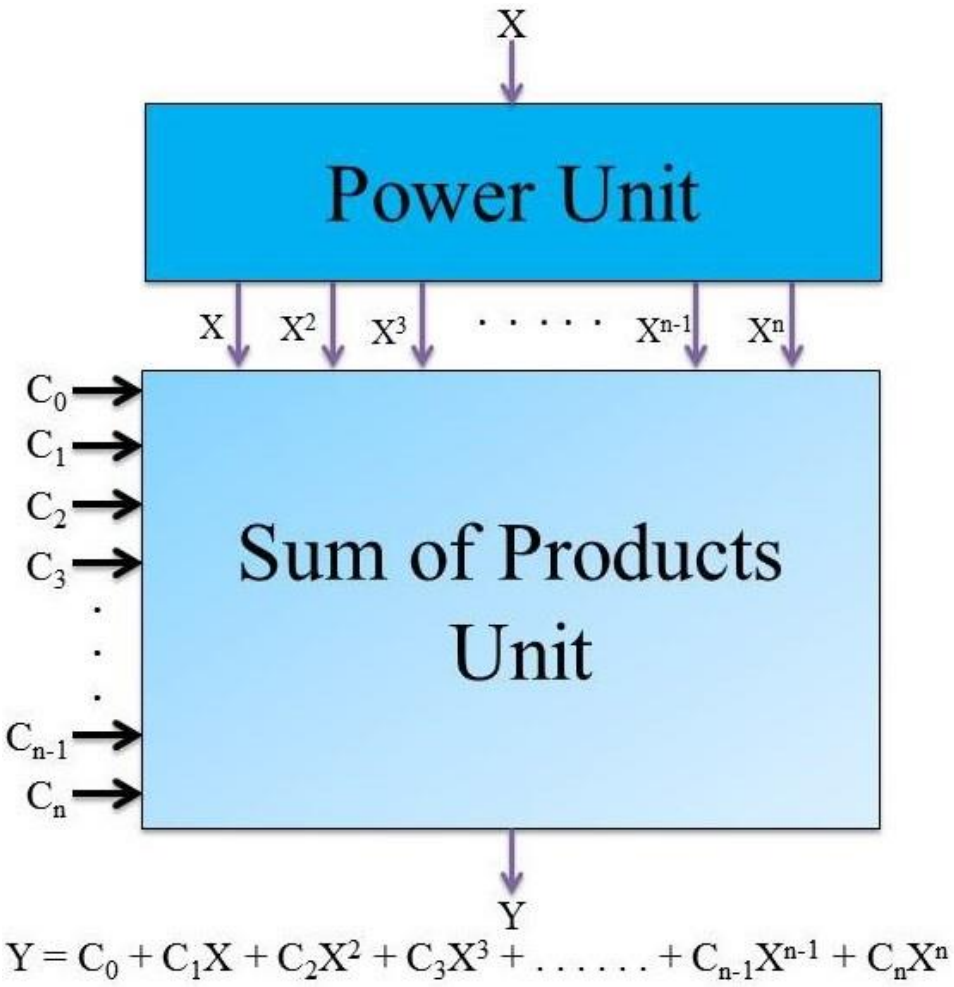


Figure 20: Polynomial Unit: Second Level Block Diagram

Table 5: PolyGen Resource Utilization (2 Cores) for Virtex-5 board

Degree n	Estimated Slices	Actual Slices
8	40826	37166 (38.20%)
9	46568	42766 (43.96%)
10	52574	47736 (49.07%)
11	58580	52630 (54.10%)
12	64982	56214 (57.78%)
13	71252	61006 (62.71%)
14	77786	65534 (67.36%)
15	84320	70194 (72.15%)
16	91646	74848 (76.94%)
17	98444	80818 (83.07%)
18	105506	85618 (88.01%)
19	112568	90410 (92.93%)
20	120026	95202 (97.86%)

evaluations.

We synthesized the entire ASDSP after using the RARE toolset with four polynomial function cores for various degrees to determine the FPGA resources needed for the complete ASDSP. Our results showed that the total number of resources needed for the entire ASDSP was approximately equal to the resources needed for the function cores alone. For example, a four core ASDSP for degree n=7 requires 63,786 slices. In contrast, since a single core requires 15,935 slices, four cores require 63,470 slices. Hence there is a difference of 316 slices.

Table 6: Maximum Resource Utilization: PolyGen (2 and 4 Cores) on Virtex 5

Degree n	# of Cores	Slices	% Slices
20	2	95202	97.86%
10	4	95472	98.14%

Table 6 shows the results of our experiment to determine the maximum degree polynomial fitting on the Virtex 5. The FPGA board in our laboratory can be used to effectively hold a two core design with maximum degree n=20 and a four core design with maximum degree n=10.

Since the goal of the RARE project is to develop FCCM-based units that execute an order of magnitude faster than executing the same algorithm on a desktop computer, we measure the speedup for the units that would fit on the Virtex 5 part. We measure speedup of the FPGA over a server with a multi-core processor. While the server has a clock frequency of 3.00 GHz, the FPGA board runs at a clock frequency of 200 MHz. A software version of the polynomial evaluation using PolyGen method was executed on the server to determine the execution time for polynomial evaluation.

Table 7 and 8 presents the estimated result using ASDSP framework with two and four cores. The table shows a comparison of execution time on the multi-core processor versus the hardware implementation for PolyGen methods when performing 2^{21}=2,097,152 polynomials evaluations. We used fixed coefficients for the experiment. The focus was on the maximum degree polynomial that fits on the Virtex 5 part from the previous table. From the Table 7 it can be seen that 2

Table 7: Polynomial Evaluation (2 Core) : Estimated Speedup/Execution Time of hardware versus software

Degree n	Core Latency	# of Cycles	Estimated HW Time in msec	SW Time in msec	Estimated Speedup
7	56	1048631	5.243155	818.0000	156.0129350
8	56	1048631	5.243155	981.6520	187.2254396
9	72	1048647	5.243235	1147.936	218.9365916
10	72	1048647	5.243235	1310.000	249.8457536
11	72	1048647	5.243235	1488.000	283.7942606
12	72	1048647	5.243235	1658.000	316.2169920
13	72	1048647	5.243235	1825.000	348.0675575
14	72	1048647	5.243235	2011.032	383.5479432
15	72	1048647	5.243235	2186.000	416.9181812
16	72	1048647	5.243235	2345.343	447.3083888
17	88	1048663	5.243315	2517.000	480.0398221
18	88	1048663	5.243315	2700.735	515.0815848
19	88	1048663	5.243315	2848.000	543.1678242
20	88	1048663	5.243315	3021.000	576.1622180

Table 8: Polynomial Evaluation (4 Core) : Estimated Speedup/Execution Time of hardware versus software

Degree n	Core Latency	# of Cycles	Estimated HW Time in msec	SW Time in msec	Estimated Speedup
7	56	524343	2.621715	818.0000	312.0095052
8	56	524343	2.621715	981.6520	374.4312406
9	72	524359	2.621715	1147.936	437.8568990
10	72	524359	2.621715	1310.000	499.6729240

core design with maximum degree n=20 can achieve estimated speedup of approximately 576X over a server with multi-core processor. From the Table 8 it can be seen that 4 core design with maximum degree n=10 can achieve estimated speedup of approximately 499X over a server with multi-core processor.

6 Conclusions and Future Work

This report describes an FCCM which can be helpful in accelerating compute-intensive HPC application. The multi-memory framework was presented showing the enhanced performance by achieving speedup by hardware Taylor core over software implementation of Taylor series. To accelerate the execution of the Taylor core the framework was optimized for concurrent execution of memory reads/writes. The optimized framework significantly reduced the number of clock cycles required for executing Taylor core. The results demonstrate that although the clock speed of the FPGA board that we used operates at maximum clock frequency of 200 MHz is orders-of-magnitude slower than the processor clock frequency 3.00 GHz used in this study, the FCCM implementation was still significantly faster.

In the first experiment, we presented the acceleration framework which achieved speedup by offloading the complex computational algorithm from the MicroBlaze processor. The result shows that approximately 527X speedup was achieved by hardware natural logarithm over implementation of natural logarithm on MicroBlaze. In second experiment, we demonstrated the speedup of approximately 200X using an optimized framework and the speedup of approximately 2.5X using an unoptimized framework of hardware Taylor core over software implementation of Taylor series. In third experiment the result table shows that the estimated FPGA resource utilization is approximately equal to actual resource utilization for 2 and 4 core polynomial evaluation unit generated by automated tool PolyGen. The 2 core design with maximum degree n=20 and 4 core design with maximum degree n=10 can fit on Virtex 5. The result tables also shows that estimated speedup of 576X with 2 core design and 499X with 4 core design core design can be achieved over software implementation of polynomial evaluation.

The future work will focus on implementing Remote And Reconfigurable-computing Environ-

ment (RARE) framework. For this we will use Java RMI and JNI technology to establish remote connection between client and FCCM server. The RARE framework will help the users not knowledgeable in hardware/software design to access FCCM accelerators from a remote site. The future work will also include the developing the toolset that will automate the process of core insertion into the ASDSP.

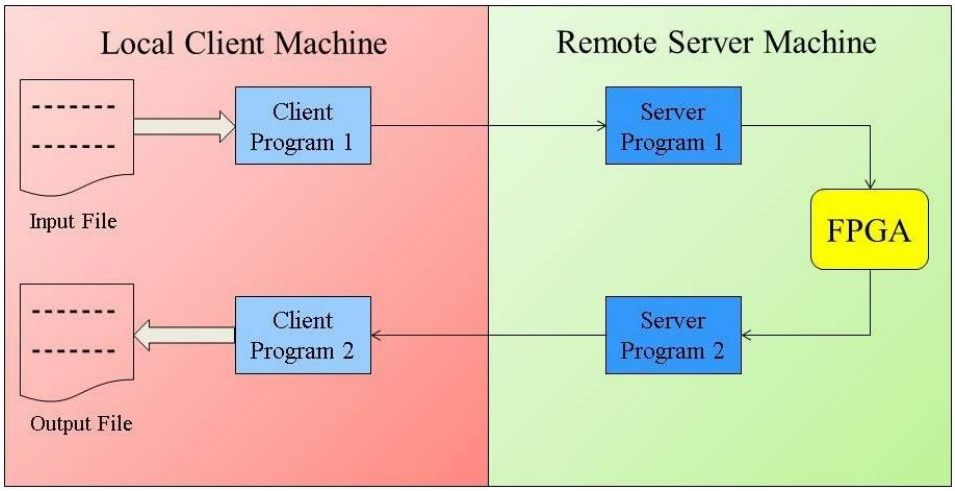


Figure 21: Client/Server Streaming Application

Additionally, future work will involve conducting experiments with Taylor series to measure speedup from the remote site. We are also going to conduct the experiments with Power (y^x) unit to measure speedup from remote site. Figure 21 shows the proposed prototype of the streaming environment. The optimized multi-memory framework can be used for streaming run-time data. Two controllers will be required for streaming data. One controller will handle the data coming from client and feed it to the FPGA for processing, while other controller will be responsible to read the processed data from the FPGA and send the data back to client.

References

[1] M. Ali, J. Gao, and M. Antolovich, "Classification on stiefel and grassmann manifolds via maximum likelihood estimation of matrix distributions," in *2016 International Joint Conference on Neural Networks (IJCNN)*, pp. 3751–3757, July 2016.

[2] S. Lv, C. Xing, Z. Zhang, and K. Long, "Guard zone based interference management for d2d-aided underlaying cellular networks.," *IEEE Transactions on Vehicular Technology*, vol. PP, no. 99, pp. 1–1, 2016.

[3] H. Chung, S. J. Lee, and J. G. Park, "Deep neural network using trainable activation functions," in *2016 International Joint Conference on Neural Networks (IJCNN)*, pp. 348–352, July 2016.

[4] J. L. Hennessy and D. A. Patterson, *Computer Architecture, Fifth Edition: A Quantitative Approach*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 5th ed., 2011.

[5] O. Knodel and R. G. Spallek, "RC3E: provision and management of reconfigurable hardware accelerators in a cloud environment," *CoRR*, vol. abs/1508.06843, 2015.

[6] G. Marcus, W. Gao, A. Kugel, and R. Mnner, "The mprace framework: An open source stack for communication with custom fpga-based accelerators," in *Programmable Logic (SPL), 2011 VII Southern Conference on*, pp. 155–160, April 2011.

[7] M. Jacobsen, D. Richmond, M. Hogains, and R. Kastner, "Riffa 2.1: A reusable integration framework for fpga accelerators," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 8, pp. 22:1–22:23, Sept. 2015.

[8] N. Neves, P. Tomás, and N. Roma, "Host to accelerator interfacing framework for high-throughput co-processing systems," 2015.

[9] R. Brodersen, A. Tkachenko, and H. K. H. So, "A unified hardware/software runtime environment for fpga-based reconfigurable computers using borph," in *Proceedings of the 4th Inter-*

national Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS '06), pp. 259–264, Oct 2006.

[10] R. Brzoza-Woch and P. Nawrocki, "Reconfigurable fpga-based embedded web services as distributed computational nodes," *Annals of Computer Science and Information Systems*, vol. 6, pp. 159–164, 2015.

[11] S. Byma, J. G. Steffan, H. Bannazadeh, A. L. Garcia, and P. Chow, "Fpgas in the cloud: Booting virtualized hardware accelerators with openstack," in *Field-Programmable Custom Computing Machines (FCCM), 2014 IEEE 22nd Annual International Symposium on*, pp. 109–116, May 2014.

[12] C. Gloster, W. Gay, and M. Amoo, "Multiple-memory application-specific digital signal processor," Aug. 18 2015. US Patent 9,111,068.

[13] S. Chan, X. Zhou, C. Huang, S. Chen, and Y. F. Li, "An improved method for fisheye camera calibration and distortion correction," in *2016 International Conference on Advanced Robotics and Mechatronics (ICARM)*, pp. 579–584, Aug 2016.

[14] R. Ferrero, P. A. Pegoraro, and S. Toscani, "Dynamic fundamental and harmonic synchrophasor estimation by extended kalman filter," in *2016 IEEE International Workshop on Applied Measurements for Power Systems (AMPS)*, pp. 1–6, Sept 2016.

[15] K. S. Islam, W. Shen, A. Mahmud, M. A. Chowdhury, and J. Zhang, "Stability enhancement of dfig wind turbine using lqr pitch control over rated wind speed," in *2016 IEEE 11th Conference on Industrial Electronics and Applications (ICIEA)*, pp. 1714–1719, June 2016.

[16] J. Zhang, T. Jin, L. Qiu, W. Liu, and Z. Zhou, "Performance analysis for t-rn multistatic radar system," in *2016 IEEE International Geoscience and Remote Sensing Symposium (IGARSS)*, pp. 6106–6109, July 2016.

North Carolina Agricultural and Technical State University

DOD URAP SITE VISIT: A REMOTE AND RECONFIGURABLE ENVIRONMENT

Clay Gloster, Jr

cgloster@ncat.edu

July 13, 2017

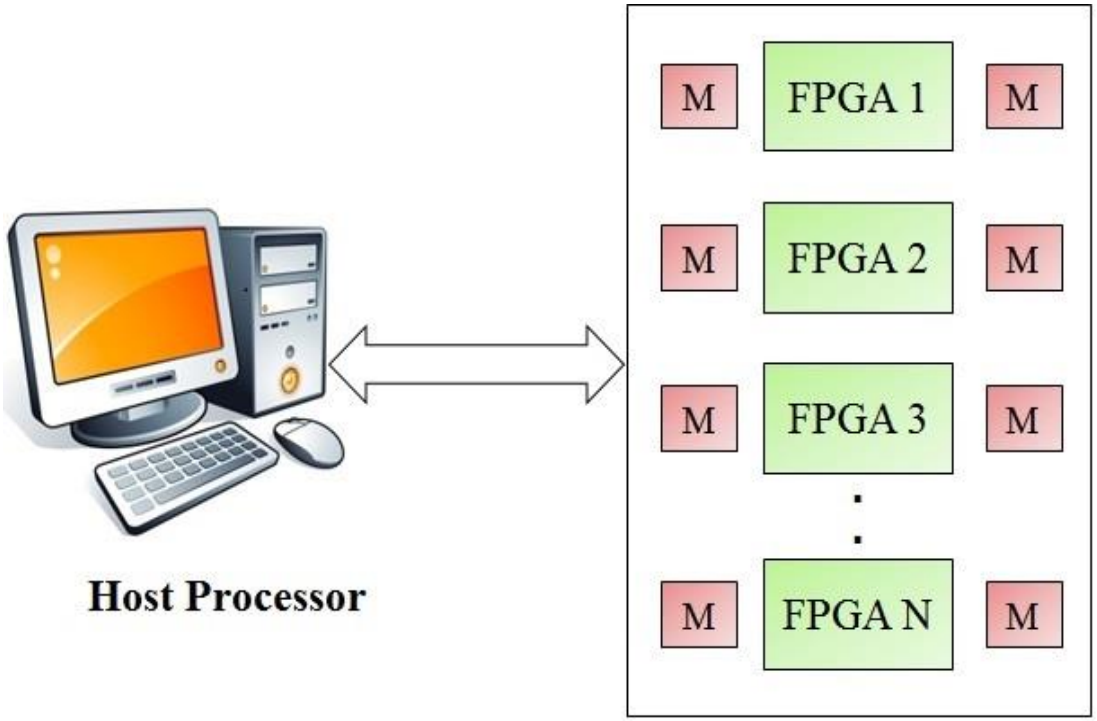
AGGIES DO

Problem Statement

- Given: A computationally intensive application that requires excessive execution time (> several hours) on a conventional processor.
  - Ex. bioinformatics, molecular dynamics, network security, weather modeling, remote sensing, etc.

- Find: An RC implementation of the application that significantly reduces the execution time.

Reconfigurable Computer



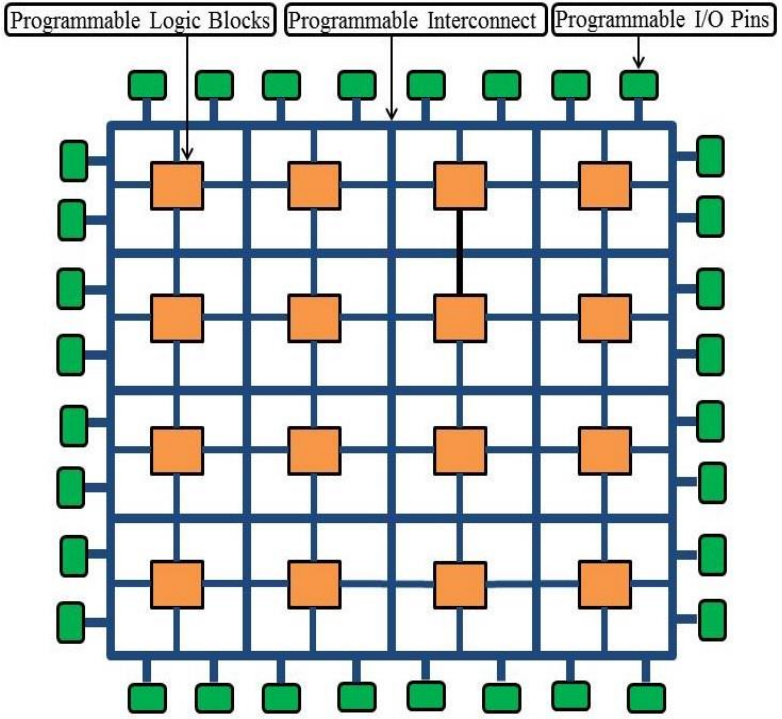
We define a Reconfigurable Computer (RC) as a host processor attached to one or more Field Programmable Gate Arrays (FPGAs) connected to multiple memories.

What is an FPGA?

A programmable integrated circuit.

At time t_1, it can be programmed as X_1 (image edge detection).

At time t_2, it can be programmed as X_2 (virus detection/bioinformatics).



FPGA

RC Systems

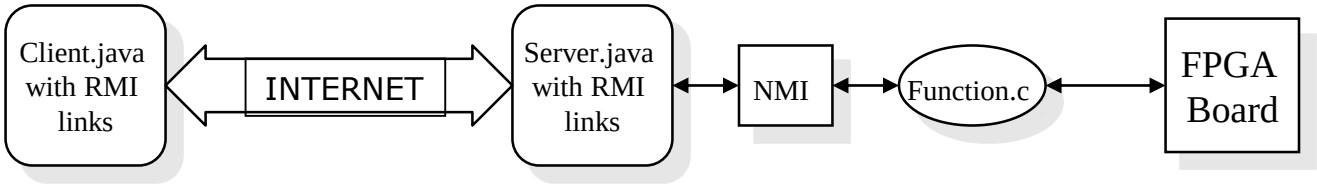
- Advantages
  - Applications implemented on an RC can achieve an order of magnitude speedup over an implementation on a conventional processor.
  - The same RC hardware can be reused for diverse applications.

- Disadvantages
  - To develop an RC, knowledge of both hardware design and software development is necessary
  - Development time to design and implement an RC can be from longer than traditional software development.

The RARE Project Infrastructure

The RARE software is developed using Java. The Java language is selected because it offers a number of advantages over other programming languages.

Java supports native methods, remote method invocation and network security. The native method feature allows the use of software routines written in other programming languages such as C/C++ to be called from Java applications. Remote method invocation and network security features make it possible to execute Java programs from a remote site.



PNN Execution Times

Implementation Type	Local (ms)	Remote (ms)
Software (Java)	628.71	2887.74
Software (Cpp)	861.04	3116.17
Hardware	104.07	371.01

Remote hardware can be faster than local software!!!!

Research Objectives

- To implement a computationally intensive application on an RC and achieve an order of magnitude speedup over a conventional processor.
- To develop tools that reduce overall RC development time.
- To allow users who are not knowledgeable in hardware/software design to take advantage of the potential increase in system performance.

Case Study: Polynomial Evaluation using Taylor Series

- Taylor Series is a representation of a function as an infinite sum of terms that are calculated from the values of the function's derivatives at a single point.
  - It is represented as $\sum_{n=0}^{\infty} \frac{f^{(n)}(a)}{n!}(x-a)^n$
  - For example, sin(x) = $\sum_{n=0}^{\infty} \frac{(-1)^n}{(2n+1)!}x^{2n+1} = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots$
- Polynomial evaluation involves computation of a sum of products, where each product is a coefficient multiplied by a power of a variable evaluated at a particular point.
  - Y = $\sum_{n=0}^{\infty} C_n X^n = \mathrm{C}_0 + \mathrm{C}_1\mathrm{X} + \mathrm{C}_2\mathrm{X}^2 + \cdots$

Polynomial Evaluation (n=7): Speedup/Execution Time of optimized hardware versus software

# of Samples	SW Time in msec	Estimated HW Time in msec	Actual HW Time in msec	Estimated Speedup	Actual Speedup
65536	34.91	0.164	0.344	212.76	101.44
131072	69.52	0.328	0.590	212.00	117.68
262144	143.24	0.656	0.777	218.47	184.35
524288	282.02	1.311	1.485	215.12	189.84
1048576	566.16	2.622	2.823	215.95	200.56
2097152	1118.67	5.243	5.524	213.35	202.47

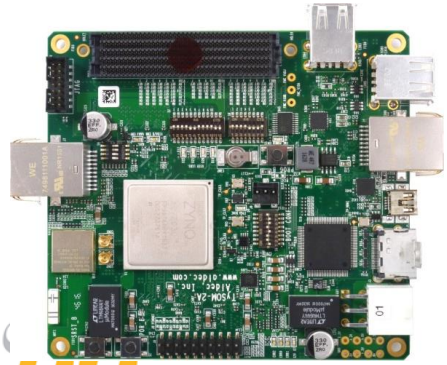
We have achieved 200X speedup over the software implementation.

- **Servers:**
  - IBM Power System S824L
    - X2 64-bit POWER8 processor cards
    - Cloud Computing and Superior Cloud Economics
    - High capacity of PCIe Gen3 x16 and x8 lanes to maximize cloud computing

  - **x2** Dell PowerEdge R730
    - Extensive I/O options
    - Large Memory Footprint
    - High-Capacity Storage

- **Graphics Processing Units (GPU)**
  - Nvidia P100
    - ***Most advanced data-center GPU ever built***
    - NVLink performance boost – 5.3 TFLOPS double-precision with 732 GB/s throughput
  - Nvidia K80 and K40
    - K80 – 2.91 TFLOPS with 480 GB/s throughput
    - K40 – 1.4 TFLOPS with 288 GB/s throughput

- **Field Programmable Gate Arrays (FPGA)**
  - PCIe-Based FPGA Cards with OpenCL High Level Synthesis (HLS) capability
    - Nallatech 510T
    - Nallatech 385 CAPI FPGA
    - Bittware XUP3SR
    - DE5-Net FPGA Development Kit
  - Independent FPGA Systems
    - ALDEC TySOM-2A
    - ZYBO
    - PYNQ



THE RARE Team (Current and Previous Members)

- Dr. Youngsoo Kim (Assistant Professor at San Jose State University)
- Kitty Huynh (PhD degree at NC A&T State University)
- Winston Hill (PhD degree at NC A&T State University)
- Shrikant Jadhav (PhD at NC A&T State University)
- Jannatun Naher (PhD degree at NC A&T State University)
- Emmanuel Olalere (MS degree at NC A&T State University)
- Vance Alford (MS degree student at NCAT, Now with First Citizen)
- Jacob Anderson (BS student at UNC Chapel Hill)
- Andrew Harvey (BS Student at UNC Chapel Hill)
- Dylan Jordan (BS degree student at NC State University)
- Lessley Hernandez (BA degree at Dartmouth College)
- Vernon Kornegay (BS degree at NC A&T State University)
- Alexandria Fuller (BS degree at NC A&T State University)

Server Information

Server Name	OS Installed	Card Installed	Processor	RAM	HD Space
rarediamond	Cent OS 5	Nallatech PCIe – 180 PCIe – 385 Nvidia Quadro K600	Intel(R) Xeon(R) E5-2630 v2 @ 2.60 GHz	15 GB	854 GB
rareamber	Windows 7 Enterprise	RARE Documents	Intel(R) Xeon(R) E5 – 2620 @ 2 GHz	32 GB	1 TB
raretopaz	Cent OS 5	No Card	Intel(R) Xeon(R) E5 – 2620 @ 2 GHz	32 GB	1 TB
rarecoral	Cent OS 5	Nvidia Quadro K600	Intel(R) Xeon(R) E5 – 2620 @ 2 GHz	32 GB	1 TB
rareemerald	Ubuntu 14	-	Intel(R) Xeon(R) E5 – 2620 @ 2 GHz	32 GB	1 TB
rareopal	Cent OS 5	Nallatech PCIe – 180 Nvidia Quadro K600	Intel(R) Xeon(R) E5-2630 v2 @ 2.60 GHz	15 GB	854 GB
rareruby	Cent OS	Nallatech 510T			
rarepearl	Cent OS	Nallatech PCIe – 385A			

Server Information (continued)

Server Name	OS Installed	Card Installed	Processor	RAM	HD Space
raresapphire	Cent OS	-			
rarequartz		-			96 TB
rareonyx	Cent OS	BittWare StratixV GXEAB Stratix V GSED8 Arria 10 GX Arria 10 SX	Intel Xeon E5-2650 @ 2.6 GHz	8 GB	256 GB
rareobsidian		BittWare 3 Xilinx UltraScale+ PCIe Virtex VU9P	Intel Xeon E5-2650 @ 2.2 GHz	8 GB	256 GB