



8-2003

Approaches for MATLAB Applications Acceleration Using High Performance Reconfigurable Computers

Saumil Girish Merchant
University of Tennessee - Knoxville

Recommended Citation

Merchant, Saumil Girish, "Approaches for MATLAB Applications Acceleration Using High Performance Reconfigurable Computers. "
Master's Thesis, University of Tennessee, 2003.
https://trace.tennessee.edu/utk_gradthes/2127

This Thesis is brought to you for free and open access by the Graduate School at Trace: Tennessee Research and Creative Exchange. It has been accepted for inclusion in Masters Theses by an authorized administrator of Trace: Tennessee Research and Creative Exchange. For more information, please contact trace@utk.edu.

To the Graduate Council:

I am submitting herewith a thesis written by Saumil Girish Merchant entitled "Approaches for MATLAB Applications Acceleration Using High Performance Reconfigurable Computers." I have examined the final electronic copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Electrical Engineering.

Dr. Gregory D. Peterson, Major Professor

We have read this thesis and recommend its acceptance:

Dr. Donald W. Bouldin, Dr. Michael A. Langston, Dr. Seong G. Kong

Accepted for the Council:

Dixie L. Thompson

Vice Provost and Dean of the Graduate School

(Original signatures are on file with official student records.)

To the Graduate Council:

I am submitting herewith a thesis written by Saumil Girish Merchant entitled "Approaches for MATLAB Applications Acceleration Using High Performance Reconfigurable Computers." I have examined the final electronic copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Electrical Engineering.

Dr. Gregory D. Peterson

Major Professor

We have read this thesis and
recommend its acceptance:

Dr. Donald W. Bouldin

Dr. Michael A. Langston

Dr. Seong G. Kong

Accepted for the Council:

Anne Mayhew

Vice Provost and
Dean of Graduate Studies

(Original signatures are on file with official student records.)

**Approaches for MATLAB Applications
Acceleration Using High
Performance Reconfigurable Computers**

A Thesis

Presented for the

Master of Science Degree

The University of Tennessee, Knoxville

Saumil Girish Merchant

August, 2003

Dedicated to my parents Girish Merchant and Kokila Merchant,
my uncle and aunt Sanjay Merchant and Jayshree Merchant and
my sister Snehal Merchant

Acknowledgements

First and foremost, I extend sincere thanks to my advisor Dr. Gregory D. Peterson for his continued guidance and support, constant encouragement with ideas and criticisms that has made this thesis possible. He showed faith in my work and it has been a great pleasure to have him as my thesis mentor. I would also like to express my gratitude to Dr. Donald W. Bouldin for all the teachings and guidance. I would like to acknowledge with appreciation participation of Dr. Michael Langston and Dr Seong Kong in my thesis defense.

Special thanks to Dr. Chandra Tan for the help extended to me whenever I approached him and the valuable discussions that I had with him during the course of my research. He made things look possible when I had almost lost hope of getting around them.

I am especially grateful to the team at OIT and my supervisor, Robin McNeil, with whom I have worked throughout my stay at Knoxville. They have been very understanding and cooperative all the time and a finest team.

Many thanks are due to my friends and colleagues – Mahesh Dorai, Ashwin Balakrishnan, Melissa Smith and others for their helpful suggestions.

I thank my best friend and fiancée Jaya, who has shown untiring patience and support, reminding me of my priorities and keeping things in perspective. Last but not the least, I would like to thank my parents and family who always kept faith in me and offered unconditional support and encouragement. I am forever indebted to the love and caring of my family.

Abstract

A lot of raw computing power is needed in many scientific computing applications and simulations. MATLAB®[†] is one of the popular choices as a language for technical computing. Presented here are approaches for MATLAB based applications acceleration using High Performance Reconfigurable Computing (HPRC) machines. Typically, these are a cluster of Von Neumann architecture based systems with none or more FPGA reconfigurable boards. As a case study, an Image Correlation Algorithm has been ported on this architecture platform. As a second case study, the recursive training process in an Artificial Neural Network (ANN) to realize an optimum network has been accelerated, by porting it to HPC Systems. The approaches taken are analyzed with respect to target scenarios, end users perspective, programming efficiency and performance.

Disclaimer: Some material in this text has been used and reproduced with appropriate references and permissions where required.

[†] MATLAB® is a registered trademark of The Mathworks, Inc. ©1994-2003.

Table of Contents

1	Introduction.....	1
1.1	High Performance Computing (HPC).....	1
1.2	Reconfigurable Computing (RC).....	3
1.3	High Performance Reconfigurable Computing (HPRC)	4
1.4	Problem Addressed	8
1.5	Related Work	9
1.6	Outline Of Thesis.....	14
2	Approaches For Porting Matlab Applications To HPRC	16
2.1	MATLAB [®] - External Interface †	18
2.1.1	Introduction to MATLAB [®] MEX-Files.....	19
2.1.2	C MEX-Files.....	20
2.1.3	Calling MATLAB [®] from C Programs - MATLAB [®] Engine	23
2.2	Parallel Virtual Machine (PVM).....	27
2.2.1	Parallel Programming Paradigms	28
2.2.1.1	Crowd Computation Paradigm	28
2.2.1.2	Tree Computation Paradigm.....	30
2.2.1.3	Hybrid Computation Paradigm.....	31
2.3	Pilchard – A Reconfigurable Computing Platform.....	34
2.4	Approaches to Port MATLAB [®] Applications to HPRC.....	39
2.4.1	Approach I – Library Based Approach.....	39
2.4.2	Approach II – C as a Master Program	40

	vi
3 Case Study I – Implementing Image Correlation On HPRC	50
3.1 Convolution Operation.....	50
3.1.1 FFT Convolution.....	53
3.2 Correlation Function	59
3.3 Implementation on HPRC.....	61
3.3.1 Library Based Approach.....	61
3.3.2 C as a Master.....	66
3.3.3 Hardware Implementation	67
3.3.4 Results.....	74
3.3.5 Limitations	75
4 Case Study II – Artificial Neural Network Training Speedups	76
4.1 Introduction to Artificial Neural Networks (ANN)	76
4.2 Estimation of Solar Particle Event Doses: A Case Study	81
4.3 Results and Discussion	89
5 Discussion And Conclusions	98
5.1 Feasibility and Target Scenarios for both Approaches	100
5.2 Performance Advantage and Run Time Efficiency	103
5.3 End User Friendliness.....	104
5.4 Ease of Programming.....	104
6 Future Work	105
References.....	107
Appendices.....	112
Appendix A – Some figures of Chapter 3.....	113

Appendix B – Steps to port MATLAB functions to HPRC.....	119
Appendix C – Program Codes	122
Vita.....	163

List Of Tables

Table 2.1	C Engine Routines	25
Table 2.2	List of some PVM Routines.....	29
Table 2.3	Xilinx Virtex FPGA Device XCV1000E Product Features.....	35
Table 2.4	Features of Pilchard Platform [38].....	38
Table 3.1	Execution times for serial and parallel executions	74
Table 4.1	Serial and Parallel Execution Times	90
Table 4.2	Parallel Training Result. X- Unsuccessful Training; \checkmark - Successful Training.....	90
Table 5.1	Execution times of various approaches	99

List Of Figures

Figure 1.1	Fixed-Sized Model for Speedup = $1/(s+p/N)$ [1]	2
Figure 1.2	Scaled-Sized Model for Speedup = $s+Np$ [1].....	2
Figure 1.3	Typical HPRC System Architecture [14]	5
Figure 1.4	Hierarchy of parallelism exploited by HPRC[16]	7
Figure 2.1	Screen shot of MEX -setup command on MATLAB prompt	20
Figure 2.2	Flowchart of C MEX cycle [36]	22
Figure 2.3	Parallel Programming Paradigms	32
Figure 2.4	Block Diagram of Pilchard Board [13]	37
Figure 2.5	Dividing MATLAB applications into various tasks.....	41
Figure 2.6	Scenario I - MATLAB as a master program (Library based approach).....	41
Figure 2.7	Scenario II - C as a master Program.....	42
Figure 3.1	Single dimensional convolution in Time domain.....	52
Figure 3.2	Two-dimensional convolution in Time domain	52
Figure 3.3	FFT Convolution of two signals.....	54
Figure 3.4	Convolution outputs using direct calculation and FFT convolution method.....	55
Figure 3.5	Two-dimensional FFT convolution.....	57
Figure 3.6	Execution Times for FFT and Standard Signal Convolutions [39].	57
Figure 3.7	Execution times for Image Convolution [39].....	58

Figure 3.8	Image Correlation Example.....	60
Figure 3.9	Character recognition technique example	63
Figure 3.10	Position of the target in source image as indicated by the white dots	63
Figure 3.11	Flowchart explaining the library based approach applied to image correlation	65
Figure 3.12	Source Text Image.....	66
Figure 3.13	Using approach II - C as a master process	68
Figure 3.14	Details of the MATLAB [®] sessions invoked by slave process	69
Figure 3.15	Communication between Slave process and the FPGA	70
Figure 3.16	Input Square Wave	72
Figure 3.17	FFT calculated using hardware implementation	72
Figure 3.18	FFT calculated using MATLAB toolbox function.....	73
Figure 3.19	Graph of Execution times.....	75
Figure 4.1	A Basic Neuron	77
Figure 4.2	Example of a Neural Network.....	77
Figure 4.3	Flowchart of Neural Network Training Procedure.....	80
Figure 4.4	Sliding Time Delayed Neural Network[43]	82
Figure 4.5	Flowchart of Training process using approach I	85
Figure 4.6	Flowchart of Parallel Training Process using approach II	86
Figure 4.7	Illustration of Input Dataset Selection [43]	87
Figure 4.8	Input Data Set along with the zoomed in version on the..... right showing 2 particular events.....	87

Figure 4.9	Target Output (Dose Infinity) (left) Log Scaled (right)	88
Figure 4.10	Testing Results of the selected optimal network highlighted.....	91
	in Table 4.1	91
Figure 4.11	Snap Shot of the Output Screen.....	92
Figure 4.12	Individual Execution times with single hidden layer	95
Figure 4.13	Individual Execution times with two hidden layers	95
Figure 4.14	Serial and Parallel Execution times	96
Figure 5.1	Graph showing the code profile for case study I.....	99
Figure 5.2	MATLAB interfacing with a library of optimized routines build with 'Mex Files'	102
Figure 5.3	MATLAB interfacing with pre-existing libraries in C or Fortran using 'Mex Files'	102
Figure 5.4	Client-Server topology with computations being performed in MATLAB [®]	102
Figure 5.5	Multi - tier architecture with computations being performed in MATLAB [®]	103
Figure A.1	Source Image	113
Figure A.2	Locations of character 'a'	114
Figure A.3	Locations of character 'e'	114
Figure A.4	Locations of character 'i'	115
Figure A.5	Locations of character 'o'	115
Figure A.6	Locations of character 'u'	116
Figure A.7	Block diagram for pcore.vhd	116

Figure A.8	Block diagram for s_interface.vhd	117
Figure A.9	Block diagram for sms.vhd.....	117
Figure A.10	Layout on Virtex 1000e part.....	118
Figure A 11	Flowchart indicating the steps to port MATLAB to HPRC	121

1 Introduction

1.1 High Performance Computing (HPC)

Growing need for higher computational power and tighter budgets has triggered a lot of research in the field of High Performance Computing (HPC) and Cluster computing. Significant has gone into devising programming methods and tools for efficient use of high performance parallel computers. But parallel programming still remains a challenging task due to many reasons like architectural complexity, higher costs, availability of several custom hardware / software commodities and lack of expertise. All these and many other reasons have led to lesser commercial success and sustainability for HPC platforms. Cheaper alternatives like “Beowolf” clusters of various custom hardware commodities can be implemented but programming and optimal use of the potential of these platforms still is a considerable obstacle and a time consuming practice.

John L. Gustafson of Sandia National Laboratories has shown the performance advantage of parallel processing in his paper “Reevaluating

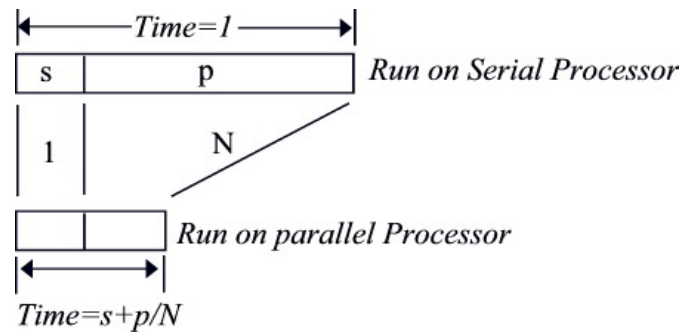


Figure 1.1 Fixed-Sized Model for Speedup = $1/(s+p/N)$ [1]

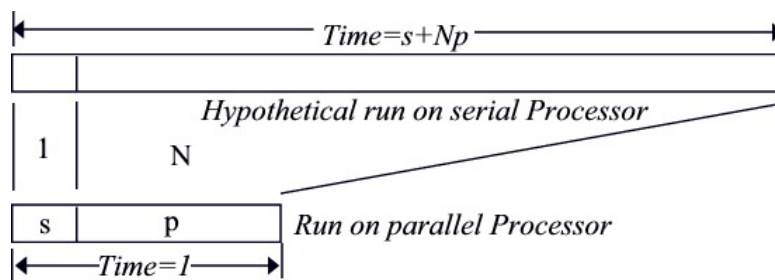


Figure 1.2 Scaled-Sized Model for Speedup = $s+Np$ [1]

Amdahl's Law" [1]. According to him, speedup in an application should be measured by scaling the problem to the number of processors, and not by fixing the problem size. He quotes, "The computing research community needs to overcome the 'mental block' against massive parallelism imposed by misuse of Amdahl's speedup formula [2]". Figures 1.1 and 1.2 [1] show the speedup obtained by the scaled problem size over the fixed problem size.

The efforts of researchers have been directed towards development of parallel libraries and APIs to facilitate distributed computing applications. APIs like PVM [3] and MPI [4] and their various flavors have been extensively used in

development of distributed computing applications and libraries. Libraries of functions such as ScaLAPACK [5] and PLAPACK [6] provide implementations of various functions on parallel hardware using MPI/PVM calls.

1.2 Reconfigurable Computing (RC)

A lot of research effort has also been expended in the field of Reconfigurable Computing (RC) with quite some commercial acceptance. Use of dedicated, reconfigurable hardware for application acceleration has been widely proven to be successful [7-10]. With the capacities in millions of gates of present day FPGA devices, shorter reconfiguration times and availability of ASIC cores on the same die as the configurable logic blocks, gives significantly enhanced performance benefits and flexibility of run time reconfiguration at a very modest cost. A lot of CAD tools are available with support of many Intellectual Property cores to facilitate and reduce the time to market of various applications. But, efficient programming of these reconfigurable elements is still a challenging task.

Various reconfigurable FPGA based boards are available commercially that interface with different bus architectures like VME and PCI. The Wildstar™ boards from Annapolis Microsystems come with interfaces for both PCI and VME buses with capacity of up to 6 million gates. Firebird™ boards also from Annapolis Microsystems come with gate capacities of up to 2 million and with interface to PCI bus. The Ballynuey™ boards from Nallatech are also PCI based boards. The SLAAC™ boards at Information Science Institute at University of

Southern California have interfaces to both PCI and VME busses. The PipeRench reconfigurable chips from Carnegie Mellon is an interconnected network of configurable logic and storage elements, which uses pipeline reconfiguration to reduce overhead which is one of the primary sources of inefficiency in other RC systems [10-12]. The Pilchard [13] FPGA boards developed by The Chinese University of Hong Kong interfaces with the processor through the DIMM slot for closer coupling reducing the I/O time. Even though FPGA market still is a relatively smaller one to that of ASICs, there has been a lot of commercial acceptance and EDA giants like Synopsys, Mentor Graphics etc. have developed EDA tools targeting FPGAs.

1.3 High Performance Reconfigurable Computing (HPRC)

Amalgamation of HPC and RC systems together forms a High Performance Reconfigurable Computing (HPRC) platform. Figure 1.3 shows a block diagram of a typical HPRC system [14, 15]. The goal of HPRC systems is to use the individual performance benefits of HPC and RC systems together to achieve a still higher performance advantages and to provide a computationally intensive hardware platform for many demanding scientific computing applications. As shown in figure 1.3 [14] HPRC platform consists of many computing nodes connected by an interconnection network (the architecture can be a switch, hypercube etc.), with some or all of the computing nodes having one or more reconfigurable processing element(s) associated with them. Additionally, an

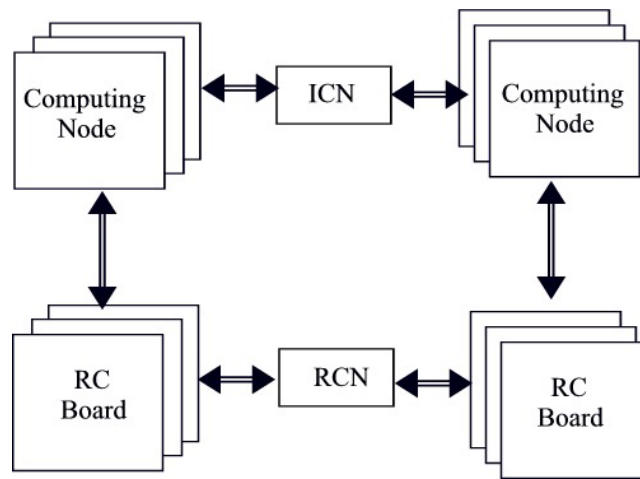


Figure 1.3 Typical HPRC System Architecture [14]

optional configurable network can be constructed to connect the RC elements for synchronization, data exchange etc.

To date, research has been ongoing primarily with focus on a single computing node with one or more reconfigurable processing elements. It is a challenging task to efficiently configure and use even these basic building blocks of the HPRC platform. The FPGA reconfiguration latency, hardware/software co-design issues and sub-optimal design tools make the efficient programming of these systems a formidable task. A layer of abstraction for programmers that can hide the architectural complexities of these complex platforms is critically important so that a programmer can concentrate on the problem domain rather than get overwhelmed with the implementation details. The partitioning of an application into hardware / software chunks and their scheduling plays an

important role in achieving significant performance gain. It is important to efficiently exploit the potential parallelism in the target application at various levels of abstraction. The target applications for such a platform includes but are not limited to signal processing, image processing, simulation, numeric algorithms and other computing intensive applications.

Figure 1.4 [16] shows the hierarchy of parallelism that can be exploited by the HPRC system. A high level software task can be divided into a number of parallel tasks that can execute on multiple shared memory processors on a single computing node or use distributed memory parallel processing by executing on different computing nodes via message passing or distributed shared memory. Further each high-level software task can be divided in to multiple concurrent software and hardware tasks. The hardware tasks can be run on multiple reconfigurable processing elements that could be associated with a computing node or else run as multiple bit wise concurrent tasks on a single FPGA fabric. Thus, multiple levels of parallelism at different levels of granularity can be exploited with the HPRC architecture to achieve significant performance gains. High Performance Reconfigurable Computing promises a cost effective solution for demanding scientific computing applications, with benefits of both HPC and RC systems.

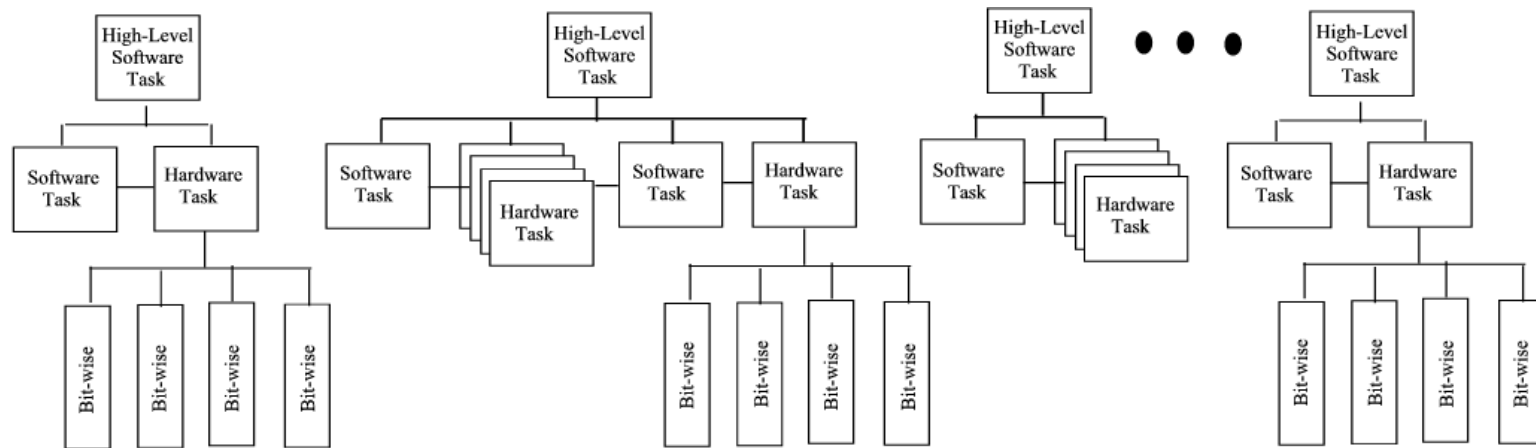


Figure 1.4 Hierarchy of parallelism exploited by HPRC[16]

1.4 Problem Addressed

MATLAB[®]† [17] over the years has evolved and has been widely accepted as a language of technical and scientific computing. With the support of a wide variety of APIs and toolboxes from The Mathworks Inc. and third party vendors, scientific computing using MATLAB[®] has had an advantage. But MATLAB[®] is not a compiler-based language like C, C++, rather, is an interpreted language like Java. Also, MATLAB[®] is not a strongly typed language and all types are represented as an array. This has some very good advantages as far as math and matrix calculations go but affect performance in loops and conditional statements. There is a lot of interest in speeding up execution of MATLAB[®] scripts, which can be very well achieved using platforms like HPRC. One of the most attractive features of MATLAB[®] is its memory model. There are no declarations or allocations – they are handled automatically. This is in contrast to the memory models used in parallel and distributed computers. This poses as one of the hurdles in actually parallelizing MATLAB[®]. According to an article by Cleve Moler, co-founder, The Mathworks Inc. [18], there had been attempts made to actually make a parallel versions of MATLAB[®]. But data distribution between the local memories of processors was an overhead far outweighing performance advantages. Besides, MATLAB[®] spends only a portion of its time in routines that

† MATLAB[®] is a registered trademark of The Mathworks, Inc. ©1994-2003.

can be potentially parallelized and rather spends much more time in places like the parser, the interpreter and the graphic routines where parallelism is difficult to find. There are applications that can exploit parallelism, but to do so requires fundamental changes to MATLAB[®] architecture, which doesn't make a good business sense for The Mathworks Inc. Hence, The Mathworks Inc. doesn't support any parallel MATLAB[®] version.

But applications very much exist which need higher computational speeds and higher speedups in processing. Parallelism can be exploited at an applications level of abstraction if not at a compiler level. The HPRC architecture platform can provide the required higher computational speeds at moderate costs. The aim of this research work is to investigate feasible approaches to accelerate MATLAB[®] based applications using HPRC.

1.5 Related Work

Various research groups and companies have expended a lot of research effort to provide parallel functionality to MATLAB[®]. In general, there are three approaches [19].

- Provide communication routines (MPI/PVM) in MATLAB[®].
- Provide parallel backend to MATLAB[®].
- Compile MATLAB[®] scripts into native parallel code.

MultiMATLAB (MATLAB[®] on multiple processors) from Computer Science Department at Cornell University [20] uses MPICH to run MATLAB[®] on multiple processors. It uses MATLAB[®] style commands like Eval, bcst, Send, Recv etc. to start MATLAB[®] processes on different processors. Currently, the system runs on IBM SP2 and on a network of Unix workstations. MPITB (MPI Toolbox for MATLAB[®]) / PVMTB (PVM Toolbox MATLAB[®]) from University of Granada in Spain [21, 22] are toolboxes written for MATLAB[®] using LAM/MPI and PVM 3.4.2 as backend support to run MATLAB[®] processes on multiple processors. They have successfully tested precompiled versions for RedHat 6.1 and MATLAB[®] 5.3. They provide calls like send, recv etc. in MATLAB[®] for message passing. Distributed and Parallel Application Toolbox for MATLAB[®] from Department of Electrical Engineering at University of Rostock, Germany [23] uses PVM to run MATLAB[®] processes on multiple processors. MatlabMPI from Lincoln Laboratory, MIT [24] uses MPI. It currently implements the basic functions of MPI for point-to-point communication. All of the above fit in the first category of providing communication routines in MATLAB[®] by using message passing environments like MPI and PVM. They all require multiple MATLAB[®] sessions.

There have been numerous compiler-based approaches as well. FALCON (Fast Array Language COmputationN) [25] from Center for Supercomputing Research and Development at the University of Illinois is a programming environment that facilitates the translation of MATLAB[®] code into Fortran 90.

Although FALCON does not directly generate parallel code, the future aim of this project is to annotate the generated Fortran 90 code with directives for parallelization and data distribution. A parallelizing Fortran compiler such as Polaris [26] may then use these directives to generate parallel code. CONLAB (CONcurrent LABoratory) [27] from Department of Computer Science at University of Ume , Sweden is a fully independent system with MATLAB-like notation that extends the MATLAB[®] language with control structures and functions for explicit parallelism. CONLAB programs are compiled into C code with a message passing library, PICL, and the node computations are done using LAPACK [28]. Otter [29] developed by Department of Computer Science at Oregon State University is a compiler that translates ordinary MATLAB[®] scripts into C Programs targeting parallel computers supporting ScaLAPACK [5] and several other supporting numerical libraries. RTExpress[™] from Integrated Sensors Inc., [30] is again a compiler that generates parallel C code directly from MATLAB[®] scripts. It supports many platforms like Linux Clusters, Sun Enterprise Servers, Network of Workstations and Mercury RACE++. They have shown some impressive 16x performance speedup for parallel processing of Hyper Spectral Sensor Data with Adaptive Filtering. ParAL (A Parallel Array Language) from the School of Electrical and Information Engineering, University of Sydney is again a project similar to Otter. It is a system for high-level machine-independent parallel programming for array applications with MATLAB[®] syntax support. All of the above are compiler based approaches to port MATLAB[®] on to parallel processors. Though we can expect better performance as shown by folks

at Integrated Sensors Inc., there is one issue with this approach. Most of these have MATLAB® like implementations and not the MATLAB® system in itself. MATLAB® being a proprietary language of The Mathworks Inc., it is difficult to include and keep up with all its functionality, especially with the rate at which MATLAB® is expanding its horizons. Also, MATLAB® being so widely accepted and used it would be beneficial to actually go with a more general approach of message passing which has been employed here.

Researchers at Electrical and Computer Engineering Department of Northwestern University with funding from DARPA developed MATCH (A MATLAB Compilation Environment for Adaptive Computing) Compiler [31] that generates RTL code directly from MATLAB® code, facilitating running of MATLAB® code on hardware. They formed AccelChip, which now holds legal license to this software and markets it. They have a library of optimized DSP IP cores that the compiler can use for better performance. This is one of the projects that targets the MATLAB® code on the hardware.

A project again sponsored by DARPA had also been undertaken at University of Tennessee, Knoxville under the name “Champion” [7, 32]. This is a library based approach and is a software environment that addressed the issue of porting the high-level design entry, using Cantata Graphical programming environment from KRI to the RC systems.

Other approach is to provide a parallel backend support to MATLAB[®]. NetSolve [33] developed by Innovative Computing Laboratories at Computer Science Department at University of Tennessee, is a client-server system that enables users to solve complex scientific problems remotely. The system allows users to access both hardware and software computational resources distributed across a network. Thus MATLAB[®] functions can be executed on a remote server assigned by scheduling agents. Netsolve has an interface to MATLAB[®] along with other interfaces like Fortran, C or Mathematica^{®†}. The Matlab*P project at MIT [34] is a similar project to NetSolve, providing a parallel backend support with interfaces to Maple^{®††}, Mathematica[®], and MATLAB[®]. PLAPACK: Parallel Linear Algebra Package in development at University of Texas, Austin [6] is mainly a parallel numerical package with an experimental interface to MATLAB[®].

Summarizing, there have been different approaches adopted by different research groups in accordance to their needs and resources. There are pros and cons of all the approaches. A Compiler based approach may prove to be better than libraries using message passing or parallel backend support approach in terms of speedups obtained, but has its own problem with MATLAB[®] being a proprietary software.

[†] Mathematica[®] is a registered trademark of Wolfram Research, Inc. © 2003

^{††} Maple[®] is a registered trademark of Waterloo Maple Inc. © 2003

Also, the target architectures that have been concentrated on are parallel and distributed processors or reconfigurable FPGA hardware, but not both at the same time. There are proven performance speedups in using either of the architectures mentioned above. Using both distributed parallel processors and reconfigurable hardware in conjunction should prove to be even more advantageous. HPRC architecture platform provides such a bed for high end processing. This work adopts a message passing approach to run MATLAB® on parallel machines. Also looked at are ways to run MATLAB® scripts on reconfigurable FPGA boards along with parallel processors. As of now, a library-based approach is used for MATLAB® functions to be run on hardware.

The Air Force Research Laboratories in Rome, NY have an HPRC cluster that they call ‘Heterogeneous HPC (HPTi)’ with 48 dual 2.2 GHz Intel Xeon processor nodes capable of delivering 422.4 GFLOPS, with each node having an FPGA board delivering 34 FIR Tera OPS[35].

1.6 Outline Of Thesis

The next chapter discusses the approaches adopted to run MATLAB® scripts on HPRC platform. As a case study, an Image Correlation Algorithm is ported on HPRC platform. As a second case study, the recursive training process in an Artificial Neural Network (ANN), to realize an optimum network, has been accelerated, by porting it to HPC platform. The reconfigurable card has not been used in the second case study due to dynamic nature of training process requiring

multiple reconfigurations of FPGAs in real time. Both of the above are then analyzed with respect to end users perspective, programming efficiency and performance benefits. This is followed by, some concluding remarks and a look at the future work to be addressed. Also, provided in appendix is a short introduction to MATLAB[®] External API Interface.

2 Approaches For Porting Matlab Applications To HPRC

For various reasons like the structure of the memory model, loosely typed language and business interests, a parallel version of MATLAB[®] is not supported by The Mathworks Inc.[18] But many scientific computing applications need higher computational speeds and more processing power. A lot of research efforts have been concentrated towards developing feasible approaches for exploiting functional parallelism and software concurrency in many scientific computing applications. Many researchers have shown significant performance gains using either HPC or RC systems. An HPRC platform as introduced in the earlier chapter would serve as a cheaper alternative with much higher processing power. A customized ‘Beowolf’ cluster can be set up with one or more reconfigurable hardware units attached to some or all of the computing nodes and can be effectively used for obtaining higher processing speeds. Though this is easier said than done, ‘it is not an insurmountable task to extract very high efficiency from a massively parallel ensemble’ as quoted by researcher John L. Gustafson of Sandia National Laboratories [1]. Many obstacles like the issue of efficiently exploiting a lot of diverse custom hardware(s) / software(s), less optimal design tools, hardware/software co-design issues, higher FPGA reconfiguration times need to be dealt with in order to achieve significant performance gains from the HPRC

platform. Here, an effort is made to research the feasible approaches to speedup the MATLAB[®] applications using the HPRC platform.

Various research groups have chosen different approaches in accordance to their needs and resources, to accelerate MATLAB[®] based applications, as summarized in the earlier chapter. There are pros and cons of all the approaches. A Compiler based approach may prove to be better than message passing or parallel backend support approach in terms of speedups obtained, but has its own problem with MATLAB[®] being proprietary software. Also, the target architectures have either been HPC or RC systems but not both at the same time as in HPRC architecture. The approach chosen here is of message passing over compiler based or backend support approaches adopted by some other research groups. Specific reasons for the choice being –

- Disadvantage of compiler based approach: MATLAB[®] is proprietary software.
- MATLAB[®] version independence.
- May or may not need multiple MATLAB[®] licenses, depending on the choice of the developer which again will be evident in the discussions to follow.
- The approach could be adopted with other languages such as SCILAB, Octave, Khoros etc.
- In fact, multiple languages and resources may be used to address a particular problem.

- Easy interface with reconfigurable hardware resources.

Message passing environment used to exploit the parallelism is ‘Parallel Virtual Machine (PVM)’ [3]. Any other environment like the ‘Message Passing Interface (MPI)’ [4] could also be used with hardly any change in the approach. The reasons for choosing PVM were more of available resources over any technical advantage. MATLAB[®] was interfaced with PVM using C as the middle ground with the help of MATLAB[®] External Interface.

2.1 MATLAB[®] - External Interface †

This section serves as a short introduction to MATLAB[®] External Interface [36]. More details are available on MATLAB[®] website. The External Interface of MATLAB[®] is its window to the outside world. It provides MATLAB[®] an interface capability with other leading languages like the C, Fortran, Java and also integration with technologies like the ActiveX and DDE (Dynamic Data Exchange). Of interest for this work and also discussed here, is mainly the interface to C language.

† MATLAB[®] External Interfaces is owned and maintained by The Mathworks Inc. © Copyright 1984-2001. Some of the figures and text here, as indicated, are reproduced from MATLAB[®] Documentation with appropriate permissions.

2.1.1 Introduction to MATLAB® MEX-Files

MEX-files are MATLAB® callable C and FORTRAN programs. They are dynamically linked subroutines that the MATLAB® interpreter can automatically load and execute. Advantages of MEX-files as listed by Mathworks Inc. are –

- Large pre-existing C and FORTRAN programs can be called from MATLAB without having to be rewritten as M-files.
- Bottleneck computations (usually for-loops) that do not run fast enough in MATLAB can be recoded in C or FORTRAN for efficiency.

These behave just like M-files and other built in functions and have an extension which is platform specific, ‘.mexsol’ for Solaris, as in our case. These can be called by MATLAB® programs just like other functions and in case when MATLAB® finds both, a MEX-file and a M-file, MEX-file takes precedence over the M-file for execution. To compile a C or FORTRAN program and save it as a MEX-file MATLAB® provides a script called ‘MEX’. MATLAB® supports many compilers and provides preconfigured files, called options files, designed specifically for a particular compiler. The default compiler that the MATLAB® uses can be changed by running MEX script with –setup option, as shown in the figure 2.1.

```

>> mex -setup

Using the 'mex -setup' command selects an options file that is
placed in ~/.matlab/R12 and used by default for 'mex'. An options
file in the current working directory or specified on the command line
overrides the default options file in ~/.matlab/R12.

Options files control which compiler to use, the compiler and link command
options, and the runtime libraries to link against.

To override the default options file, use the 'mex -f' command
(see 'mex -help' for more information).

The options files available for mex are:

1: /mnt/sw/matlab6.1/bin/gccopts.sh :
   Template Options file for building gcc MEX-files

2: /mnt/sw/matlab6.1/bin/mexopts.sh :
   Template Options file for building MEX-files via the system ANSI compiler

Enter the number of the options file to use as your default options file:

```

Figure 2.1 Screen shot of MEX -setup command on MATLAB prompt

2.1.2 C MEX-Files

The source code for a C MEX-file consists of two distinct routines

- A *computational routine* that contains the code for performing the computations that you want implemented in the MEX-file. Computations can be numerical computations as well as inputting and outputting data.
- A *gateway routine* that interfaces the computational routine with MATLAB by the entry point `mexFunction` and its parameters `prhs`, `nrhs`, `plhs`, `nlhs`, where `prhs` is an array of right-hand input arguments, `nrhs` is the number of right-hand input arguments, `plhs` is an array of left-hand output arguments, and `nlhs` is the number of

left-hand output arguments. The gateway calls the computational routine as a subroutine.

A flow diagram explaining the C MEX cycle is shown in figure 2.2.[36].

The following pseudo code shows a typical C program used as a MEX-file.

```

/*****
Pseudo C Code  `yourprogram.c' illustrates a typical C
program used as a MEX-file
*****/

#include "mex.h"
/* Other includes that your code in the computational
routine may require */

static void yourfunc(your input arguments)
{
    /* Computational routine containing your C code
    and routines */

    return;
}

void mexFunction( int nlhs, mxArray *plhs[],
                  int nrhs, const mxArray*prhs[] )
{
    /* gateway routine */

    /* Uses functions like 'mxGetM', 'mxGetN',
    'mxGetPr',      'mxCreateDoubleMatrix' etc. */

    /* For further details on these functions please
    refer to MATLAB help files. */

    /* Call to the computational routine */

    yourfunc(Input and Output Data pointers as
    Function parameters);

    return;
}

```

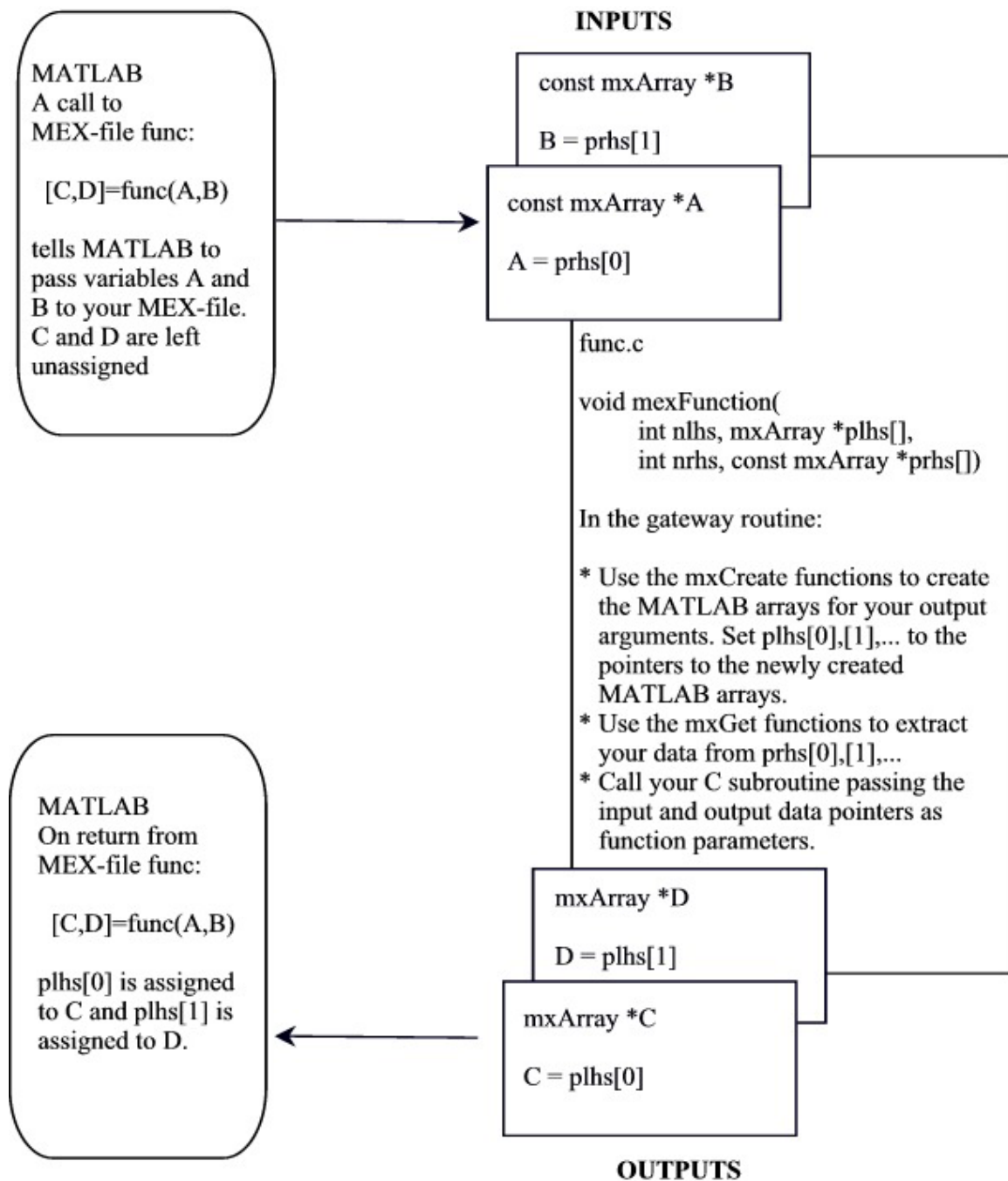


Figure 2.2 Flowchart of C MEX cycle [36]

To generate a MEX-file, at the MATLAB[®] prompt type

```
>> mex yourprogram.c
```

MATLAB[®] generates a MEX-file with the name '*yourprogram*' with appropriate extension for your system, in our case '*yourprogram.mexsol*'.

The above demonstrates how to call a C program from MATLAB[®]. The next section discusses the ways to call MATLAB[®] from C programs.

2.1.3 Calling MATLAB[®] from C Programs - MATLAB[®] Engine

MATLAB[®] commands can be called from C programs using a MATLAB[®] Engine library. These are a set of routines that allow you to call MATLAB[®] from your own program, thereby employing MATLAB[®] as the computation engine. These MATLAB[®] engine programs are C or FORTRAN programs that communicate with MATLAB[®] processes via pipes (in UNIX) or through ActiveX (in Windows). The functions in the library allow you to start or end processes in MATLAB[®], send and receive data from MATLAB[®] and send commands in MATLAB[®] to execute. This is a very useful feature and can be employed to call a specific math routine, for example to invert an array or to compute an FFT from your own program written in C. Or one can build an entire system for a specific task, for example target recognition, radar signature analysis

etc., where the front end GUI can be written in C and all the computations and analysis be done in MATLAB[®], thereby, shortening the development time. The MATLAB[®] engine operated by running as a background process separate from your own program. On UNIX, the MATLAB[®] engine can run on your machine, or any other UNIX machine on your network, including machines of a different architecture. Thus a 2-tier approach of client-server topology can be very well be employed with GUI on the workstation and the computations begin performed on some other much faster machine or may be a cluster of machines. Table 2.1 shows all the available C Engine functions.

The following pseudo code illustrates the sequence of steps to invoke MATLAB[®] engine and run MATLAB[®] functions.

```

/*****
C pseudo code to illustrate the use of MATLAB®
engine functions
*****/

#include <stdio.h>
#include "engine.h"

/* Other Includes your C code may need */

#define BUFSIZE 25000

main()
{
    Engine *ep;
    char buffer[BUFSIZE];
    int d;

```

Table 2.1 C Engine Routines

Function	Purpose
engOpen	Start up MATLAB [®] engine
engClose	Shutdown MATLAB [®] engine
engGetArray	Get a MATLAB [®] array from MATLAB [®] engine
engPutArray	Send a MATLAB [®] array to the MATLAB [®] engine
engEvalString	Execute a MATLAB [®] command
engOutputBuffer	Create a buffer to store MATLAB [®] text output
engOpenSingleUse	Start a MATLAB [®] engine session for a single non-shared use.
engGetVisible	Determine visibility of MATLAB [®] engine session
engSetVisible	Show or hide MATLAB [®] engine session


```

/***** Your other C declarations and code *****/

- - - - -
- - - - -
- - - - -
- - - - -
- - - - -

/***** starting matlab engine *****/

ep=engOpen("\0");

if (!(ep)) {
    fprintf(stderr, "\nCan't start MATLAB
engine\n");
    return EXIT_FAILURE;
} /* end if */

/***** Initialize MATLAB output buffer *****/

engOutputBuffer(ep,buffer,25000);

/***** calling your Matlab function *****/

d=engEvalString(ep,"yourfunction");

/***** Your other C code *****/

- - - - -
- - - - -

/***** closing Matlab engine *****/

engClose(ep);

}

```

To compile and link these programs proper paths should be specified.

These can also be compiled and linked using the MATLAB[®] mex script as –

```
>> mex -f <matlab>/bin/engopts.sh <pathname>/program.c
```

2.2 Parallel Virtual Machine (PVM)

Parallel processing has emerged as a key technology in modern computing. A large problem can be broken down into many parallel processes and executed concurrently on multiple processing units to achieve speed up in execution times. This has been facilitated by two major developments: massively parallel processors (MPPs) and the widespread use of distributed computing. MPPs are the fastest computers in the world today and probably the most expensive ones too. They have enormous processing power in the range of a few teraflops. These are used to solve computationally most intensive problems like the global climate modeling, drug design etc. The second major development in the parallel processing world is distributed computing. In this a set of computers connected via a network are used to solve a single large problem. With the high-speed networks of today, interconnecting many general purpose workstations, the combined processing power may exceed that of a high performance computer. The advantage of distributed computing is the cost. MPPs typically cost in tens of million dollars, which is extravagantly higher than that of a network of workstations. It is uncommon to achieve the processing power of a MPP using distributed computing, but large problems can be solved with much higher execution rates with the help of distributed computing.

In parallel processing data must be exchanged between cooperating tasks. Several paradigms exist, including shared memory, parallelizing compilers and message passing. Parallel Virtual Machine (PVM) [3] system used here is a

message passing model to allow programmers to exploit distributed computing across a wide variety of computer types, including MPPs. A key concept in PVM is that it makes a collection of computers appear as one large virtual machine. PVM is a collaborative effort of Oak Ridge National Laboratories, University of Tennessee, Emory University and Carnegie Mellon University. This is a freeware with the source code available on the PVM homepage [37]. It is very portable and has been compiled on everything, from laptops to CRAYs. It has a set of routines callable from C/C++ and FORTRAN programs, facilitating in sending and receiving data between multiple processes, spawning new tasks, process control, dynamic configuration etc. Table 2.2 lists some of the routines and their functions, for more details and a complete list of routines refer to PVM Users Guide [3].

2.2.1 Parallel Programming Paradigms

There are three common parallel programming paradigms: Crowd computation, Tree computation and hybrid, based on the organization of the computing tasks. The choice of a paradigm is application specific and should be determined with the application in mind.

2.2.1.1 Crowd Computation Paradigm

In this paradigm a set of closely related processes perform computations on different portion of the workload, usually involving periodic exchange of

Table 2.2 List of some PVM Routines

Function	Purpose
pvm_spawn	Spawns off a new task
pvm_addhosts / pvm_delhosts	Adds / deletes hosts to / from the virtual machine
pvm_mytid	Gives the task ID of the current process
pvm_kill	Kills some other PVM task identified by task ID
pvm_exit	Leave the PVM
pvm_initsend	Initialize send buffer
pvm_pk* / pvm_unpk*	These are data packinbg/unpacking routines e.g. pvm_pkint, pvm_pkstr etc.
pvm_send	Send data to another PVM process
pvm_mcast	Send data to a set of processes as specified by the task IDs
pvm_recv	Blocking data receive routine
pvm_nrecv	Nonblocking data receive routine
pvm_joingroup / pvm_lvgroup	Join a dynamic process group
pvm_bcast	Broadcast a message to all processes in a group.
pvm_gettid	Get task ID of of a process with the given group name and instance number
pvm_gsize	Get number of members in a group

intermediate results. This has two scenarios, a master-slave scenario and a node-only scenario as detailed below.

- Master-Slave Scenario

In this scenario a master program controls the behavior of a slave task. The master is responsible for process spawning, initialization, collection and display of results, and also timing functions. The slave program does the actual computational work. The child processes can be allocated their workloads by the master program, statically or dynamically, or they may perform allocations themselves. This paradigm is also called a host-node model

- Node-only Scenario

In node-only scenario multiple instances of the same program are spawned and executed. Each spawned task performs computation on its allocated data. The manually initiated process takes up the non-computational responsibility as well as computational work.

2.2.1.2 Tree Computation Paradigm

In this model the processes are spawned off (usually dynamically) during runtime in a tree like manner. A very good example would be of a split-sort-merge algorithm. Here the manually initiated process reads in the data to be sorted. It spawns of a child task and gives it half the amount of workload. Now there are two processes with half of the workload with each. Each one, splits up its own workload in two halves and spawns off a child task, giving it the half of

their share of the workload. This goes on in a tree like fashion until a manageable workload size is reached and each process sorts the data. After this the merge operation begins wherein we climb up the tree merging the data from various child processes.

2.2.1.3 Hybrid Computation Paradigm

This can be thought of as a combination of the above two paradigms, the crowd computation model and the tree computation model.

Figures 2.3 shows the various computation paradigms. The following pseudo code illustrates how a typical PVM code looks like.

```

/*****
C pseudo code to illustrate the use of PVM functions
*****/

/* Master.c */

#include "pvm3.h"

main() {

    int cc, tid, msgtag;
    char buf[100];

    printf("i'm t%x\n", pvm_mytid());

```

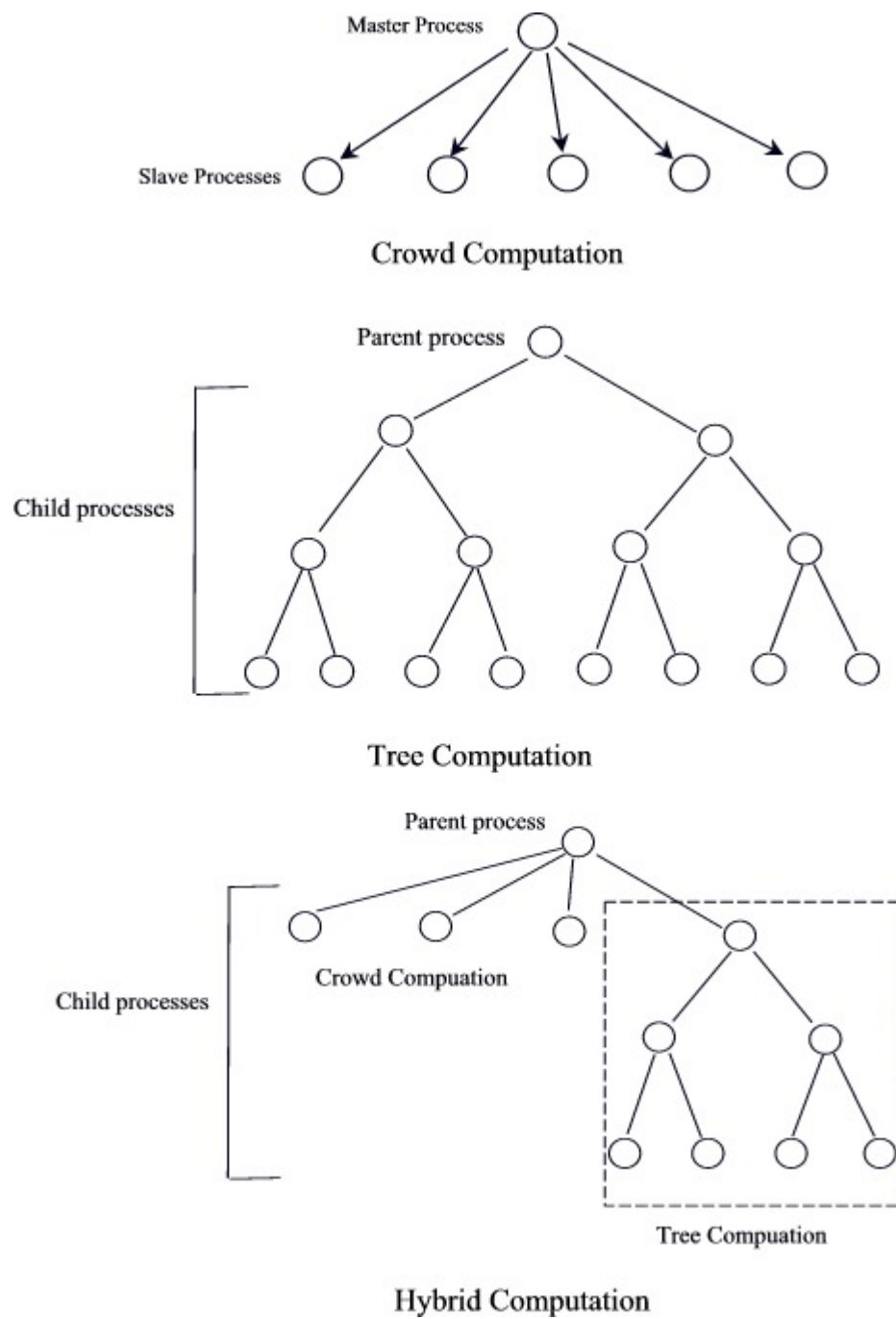


Figure 2.3 Parallel Programming Paradigms

```

cc = pvm_spawn("Slave", (char**)0, 0, "", 1,
&tid);

if (cc == 1) {
    msgtag = 1;
    pvm_recv(tid, msgtag);
    pvm_upkstr(buf);
    printf("from t%x: %s\n", tid, buf);
} else
    printf("can't start Slave Program\n");

/* other code and send and receive statements*/

-----
-----
pvm_exit();
}

/* Slave.c */

#include "pvm3.h"

main(){

    int ptid, msgtag;
    char buf[100];

    ptid = pvm_parent();
    strcpy(buf, "This is Slave Reporting from ");
    gethostname(buf + strlen(buf), 64);

    msgtag = 1;
    pvm_initsend(PvmDataDefault);
    pvm_pkstr(buf);
    pvm_send(ptid, msgtag);

    /* other code and send and receive statements*/
    -----
    -----

    pvm_exit();
}

```


2.3 Pilchard – A Reconfigurable Computing Platform

One of the important features of HPRC platform is the simultaneous use of both distributed as well as reconfigurable computing to achieve much higher speedups in execution times. The reconfigurable boards used for this research are ‘Pilchard’ boards [13, 38]. These were developed at ‘Computer Science and Engineering department, The Chinese University of Hong Kong’. The ones worked with for this research had Xilinx Virtex FPGA device XCV1000E as the reconfigurable element. Table 2.3 details some of the features of the XCV1000E FPGA part, used in the pilchards, as obtained from Xilinx website.

Pilchard boards have a memory slot interface i.e. the DIMM slot with the microprocessor unlike other boards like Firebird or Wildforce from Annapolis Microsystems, which have a PCI bus interface. The advantage of this feature is the higher I/O speed. Although FPGA systems can operate at clock frequencies over 100 MHz and microprocessors above 1 GHz, the bottleneck to higher speedups is I/O. Most personal computers still use the original 32 bit PCI bus, PC132, which has a speed of 33 MHz with maximum transfer rate of 132 MB/s. This limits the I/O speeds and is a bottleneck in achieving higher speedups. The memory bus in a PC or workstation has higher bandwidth and lower latency than the peripheral bus. The standard Dual Inline Memory Modules (DIMMs) have bandwidth of 100-133 MHz with 64 bit data, providing a maximum bandwidth of 1064 MB/s. Pilchard uses this DIMM slot to interface with the microprocessor to

Table 2.3 Xilinx Virtex FPGA Device XCV1000E Product Features

Feature	Specification
Package Used in Pilchard	HQ240 (32mm x 32mm)
CLB Array (Row x Col.)	64x96
Logic Cells	27,648
System Gates	1,569,178
Max. Block RAM Bits	393,216
Max. Distributed RAM Bits	393,216
Delay Locked Loops (DLLs)	8
I/O Standards Supported	20
Speed Grades	6,7,8
Available User I/O	158 pins (for package PQ240) max. 660 (for Device family)

over come the bottleneck. Figure 2.4 shows the block diagram of pilchard board [13]. Table 2.4 shows the features of pilchard platform. [38]

The software source files are –

iflib.h – The header file, provides API function prototypes

iflib.c – The implementation of the API functions

pilchard.c – The device driver for LINUX

The software interface available has four API functions as below –

- void read64(int64, char *) - To read 64 bits from pilchard
- void write64(int64, char *) - To write 64 bits to pilchard
- void read32(int, char *) - To read 32 bits from pilchard
- void write32(int, char *) - To write 32 bits from pilchard

int64 is a special data type that is defined in the header file iflib.h as a 2-element integer array. ‘download.c’ a configuration utility, can be used to configure the FPGA with the bit file generated from synthesis.

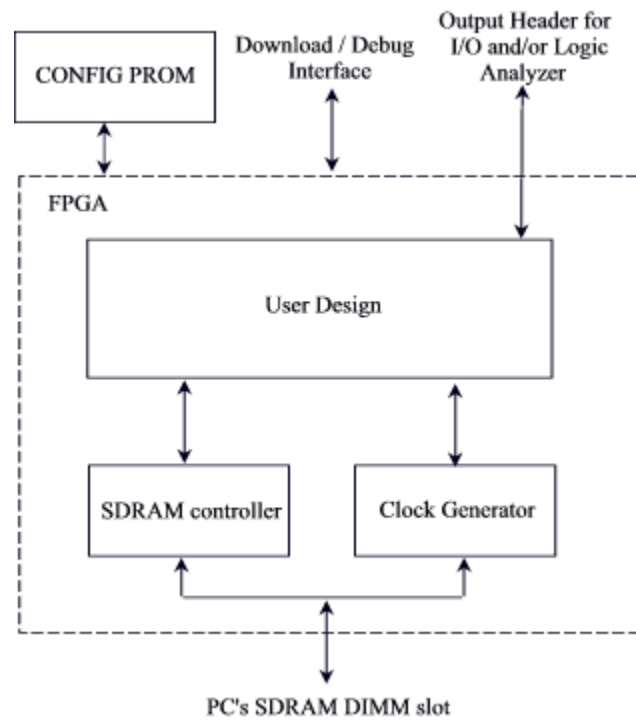


Figure 2.4 Block Diagram of Pilchard Board [13]

Table 2.4 Features of Pilchard Platform [38]

Feature	Specification
Host Interface	<ul style="list-style-type: none"> • DIMM Interface • 64-bit Data I/O • 12-bit Address Bus
External (Debug) Interface	27-Bits I/O
Configuration Interface	X-Checker, MultiLink and JTAG
Maximum System Clock Rate	133 MHz
Maximum External Clock Rate	240 MHz
FPGA Device	XCVE1000E-HQ240-6
Dimension	133mm x 65mm x 1mm
OS Supported	GNU/LINUX
Configuration Time	16s Using Linux download program

2.4 Approaches to Port MATLAB® Applications to HPRC

A high level MATLAB® application can be divided into various concurrent software and hardware tasks which can execute on various different nodes of an HPRC platform. MATLAB® External Interface can be used to interface MATLAB® programs to C code which in turn can be used in conjunction with PVM and pilchard's C interface to port the MATLAB® programs to HPRC platform. Two approaches are envisioned here.

2.4.1 Approach I – Library Based Approach

In this approach a MATLAB® program makes function calls to optimized, parallel and/or hardware routines, which execute on a remote nodes (either a computing node or a reconfigurable element or both) and return the results of computation back to the calling MATLAB® program. Thus a library of optimized routines (e.g. FFT, DES function) can be pre-built and used at will in MATLAB® programs. The MATLAB® program may also actually choose the number of tasks to be spawned and the nodes to be used, whether only the computing nodes or just the reconfigurable elements or both. Hence in this scenario MATLAB® program acts as a master program invoking tasks on various different nodes of the HPRC. A user not interested in the mechanics of the underlying architecture can directly use pre-optimized functions that would execute at various different nodes of an HPRC platform and return the result to the calling program, and thus give higher speedups. An advanced user can have more flexibility. A library of PVM functions and functions to execute code on the reconfigurable elements can be

built as a toolbox in MATLAB[®] that can be used, to directly spawn off multiple processes on different nodes and manage them directly from MATLAB[®]. This scenario is useful when developing the entire system in MATLAB[®].

2.4.2 Approach II – C as a Master Program

In this approach the non-computational task of multiple process management is handled by a master C program that spawns off various tasks that may invoke MATLAB[®] engine routines and/or execute code on hardware and return the results of computations to the master C program. This approach is feasible in systems not completely built on MATLAB[®] and using MATLAB[®] just as the computational engine. Thus a multi tier architecture may be supported with say GUI developed in C interfaced with OpenGL and actual computations being performed as call backs to MATLAB[®] functions. This approach is also feasible when using other native codes.

Figures 2.5, 2.6 and 2.7 detail these approaches. The pseudo code for the approaches is given next.

Scenario I

Master.c

```

/*****
Pseudo C Code  'master.c' illustrates a typical C
program used in scenario I
*****/

```

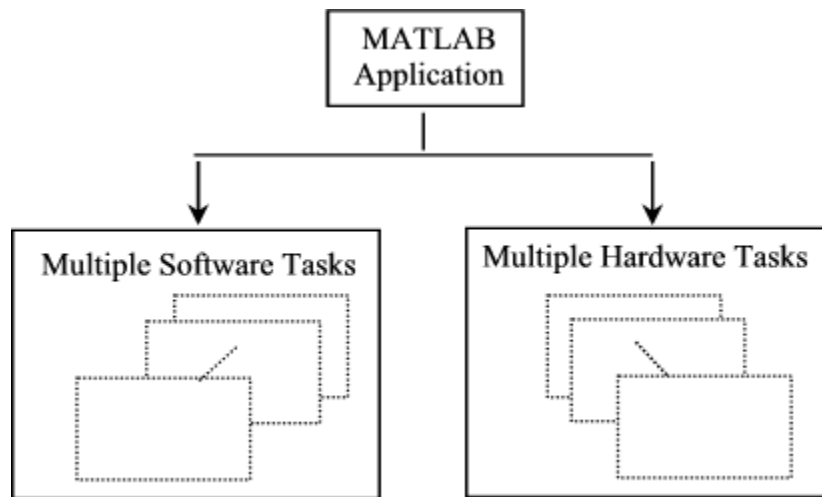


Figure 2.5 Dividing MATLAB applications into various tasks

Scenario I

MATLAB as a Master - Library Based Approach

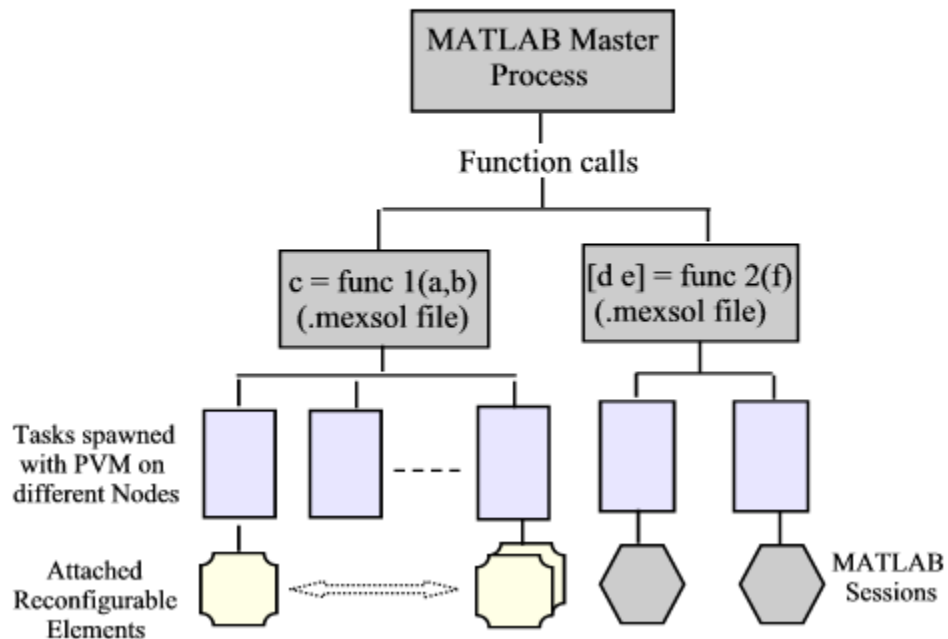


Figure 2.6 Scenario I - MATLAB as a master program (Library based approach)

Scenario II

C program as a Master Process

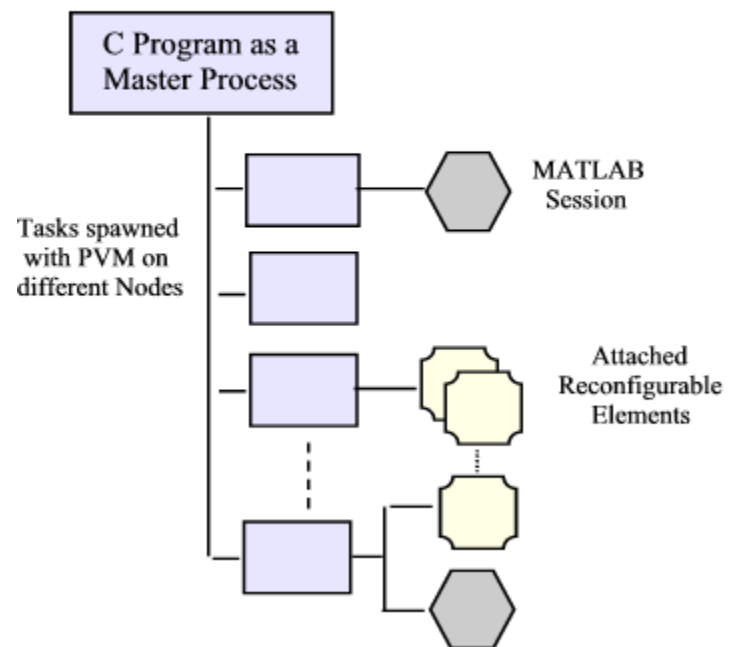


Figure 2.7 Scenario II - C as a master Program

```

#include "mex.h"
#include "pvm3.h"

/* Other includes that your code in the computational
routine may require */

static void yourfunc(your input arguments)
{
    /* Computational routine containing your C code
    and routines */

    /*** Spawning NTASK number of slave processes ***/

        cc = pvm_spawn("slave", (char**)0, 0, "",
        NTASK-1, tid);

    /***** Doing other computations *****/

        -----
        -----

    /** Receiving computed data from Slave processes **/

return;
}

void mexFunction( int nlhs, mxArray *plhs[],
                  int nrhs, const mxArray*prhs[] )
{
    /* gateway routine */

    /* Uses functions like 'mxGetM', 'mxGetN',
    'mxGetPr',          'mxCreateDoubleMatrix' etc. */

    /* For a further details on these functions
    please refer to MATLAB help files. */

    /* Call to the computational routine */

    yourfunc(Input and Output Data pointers as
    Function parameters);

return;
}

```

Slave.c

```

/*****
Pseudo C Code  `slave.c' illustrates a typical C slave
program used in scenario I
*****/

```

```

#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <sys/mman.h>
#include "iflib.h"

```

```

int main (void)
{

```

```

    /*** declarations ***/

```

```

        ptid = pvm_parent();

```

```

        /*** mapping pilchard a to a memory space ***/
        fd = open(DEVICE, O_RDWR);
        memp = (char *)mmap(NULL, MTRRZ, PROT_READ,
        MAP_PRIVATE, fd, 0);
        if (memp == MAP_FAILED) {
            perror(DEVICE);
            exit(1);
        }

```

```

        /*** writing data to pilchard ***/

```

```

        data.w[1] = 0xfefe00aa;
        data.w[0] = 0xffff0000;
        write64(data, memp+(0<<3));

```

```

        for(i=0;i<10;i++) {}

```

```

        /*** reading back from pilchard ***/
        read64(&data, memp+(0<<3)); /* get d0 */
        printf("d0 :%08x, %08x\n", data.w[1], data.w[0]);

```

```

        /******* Doing other computations *****/

```

```

        -----

```

```

-----
-----

/*****sending the output to the parent *****/

    pvm_send(ptid,2);

/**** exit****/

pvm_exit();
munmap(memp, MTRRZ);
close(fd);
return 0;
}

```

Scenario II

Master.c

```

/*****
pseudo code to illustrate the implementation of
Scenario II
*****/

#include <stdio.h>
#include "/sw/matlab6.1/extern/include/engine.h"
#include "/usr/local/pvm3/include/pvm3.h"
#include "/sw/matlab6.1/extern/include/matrix.h"

/**** other includes that may be needed ****/

int main(){

/**** declarations ****/

/***** Spawning NTASK number of slave processes
using

```

```

pvm_spawn routine *****/

cc = pvm_spawn("slave1", (char**)0, 0, "", NTASK-1,
tid);
cc = pvm_spawn("slave2", (char**)0, 0, "", NTASK-1,
tid);

/***** invoking MATLAB engine and doing
computations

in MATLAB *****/

ep=engOpen("\0");
    if (!(ep)) {
        fprintf(stderr, "\nCan't start MATLAB
engine\n");
        return EXIT_FAILURE;
    } /* end if */

d=engEvalString(ep,"yourmatlabfunction.m");

    /***** receiving the computed data from slave
process *****/

    pvm_recv(-1,-1);

    /** exit **/

    engClose(ep);
    pvm_exit();

}/** main end **/

```

Slave1.c

```

/*****
pseudo code to illustrate the implementation of
Scenario II
*****/

#include <stdio.h>
#include "/sw/matlab6.1/extern/include/engine.h"
#include "/usr/local/pvm3/include/pvm3.h"
#include "/sw/matlab6.1/extern/include/matrix.h"

/**** other includes as may be needed ****/

int main(){

/**** declarations ****/

    ptid = pvm_parent();

/***** starting matlab engine and doing computations
*****/

    ep=engOpen("\0");
    if (!(ep)) {
        fprintf(stderr, "\nCan't start MATLAB
engine\n");
    }

    d=engEvalString(ep,"yourMATLABfunction.m");

/**** sending the output to the parent ****/

    pvm_send(ptid,2);

/**** end the matlab session and exit****/

    engClose(ep);
    pvm_exit();

}

```

Slave2.c

```

/*****
pseudo code to illustrate the implementation of
Scenario II
*****/

include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <sys/mman.h>
#include "iflib.h"

int main (void)
{
    /*** declarations ***/

    ptid = pvm_parent();

    /*** mapping pilchard a to a memory space ***/
    fd = open(DEVICE, O_RDWR);
    memp = (char *)mmap(NULL, MTRRZ, PROT_READ,
MAP_PRIVATE, fd, 0);
    if (memp == MAP_FAILED) {
        perror(DEVICE);
        exit(1);
    }
    /*** writing data to pilchard ***/

    data.w[1] = 0xfefe00aa;
    data.w[0] = 0xffff0000;
    write64(data, memp+(0<<3));

    for(i=0;i<10;i++) {}

    /*** reading back from pilchard ***/
    read64(&data, memp+(0<<3)); /* get d0 */
    printf("d0 :%08x, %08x\n", data.w[1],
data.w[0]);

    /***** sending the output to the parent *****/

    pvm_send(ptid,2);

```

```
    /**** exit****/  
  
    pvm_exit();  
    munmap(memp, MTRRZ);  
    close(fd);  
    return 0;  
}
```


3 Case Study I – Implementing Image Correlation On HPRC

3.1 Convolution Operation

Convolution is a formal mathematical operation on two signals producing a third signal. This is a very fundamental operation in subjects like Signals and Systems theory, Image processing and Digital Signal Processing. In system analysis, response of a Linear Time Invariant system to an input signal can be calculated by convolving the input signal with the impulse response of the system. In Image processing, convolution can be used for many operations on images like edge detection, smoothing, linear filtering etc. Mathematically convolution operation can be expressed as, as in the case of an LTI system –

$$y[n] = x[n] * h[n] = \sum_{j=0}^{M-1} h[j]x[n-j]$$

$x[n] \Rightarrow$ N point, discrete time signal

$h[n] \Rightarrow$ M point, Convolution kernel in this case, impulse response of an LTI system

$*$ \Rightarrow Convolution operator

$y[n] \Rightarrow$ N + M - 1 point, Output Signal, in this case response of the LTI system

The operation can be thought of physically as sliding one signal over the time flipped version of the other in discrete time intervals and calculating the sum

of individual responses by adding each of the corresponding impulses. The total sum of all such overlaps gives the final convolved signal output. Figure 3.1 shows an example of convolution of two signals calculated and plotted in MATLAB[®] using the above formula. The convolution kernel, also called the filter kernel acts as a low pass filter smoothing out the output signal, as can be seen in the figure.

The above equation is for a single dimension convolution, to be more precise, for signals. To convolve images, which have two dimensions in spatial domain, we can use an extension of the above equation as given below –

$$y[m,n] = x[m,n] \otimes h[m,n] = \sum_{j=-\infty}^{\infty} \sum_{k=-\infty}^{\infty} h[j,k] x[m-j, n-k]$$

$x[m,n]$ => Rectangular image

$h[n]$ => Convolution kernel

\otimes => Convolution operator

$y[m,n]$ => Output Image

Just as in case of signals, image convolution can be physically thought of as sliding the convolution kernel over the flipped version of the rectangular image in discrete time steps, calculating the sum of the individual images in each step. The final convolved output image is the sum of all the images obtained in each time step. Figure 3.2 shows an image convolved with two different convolution

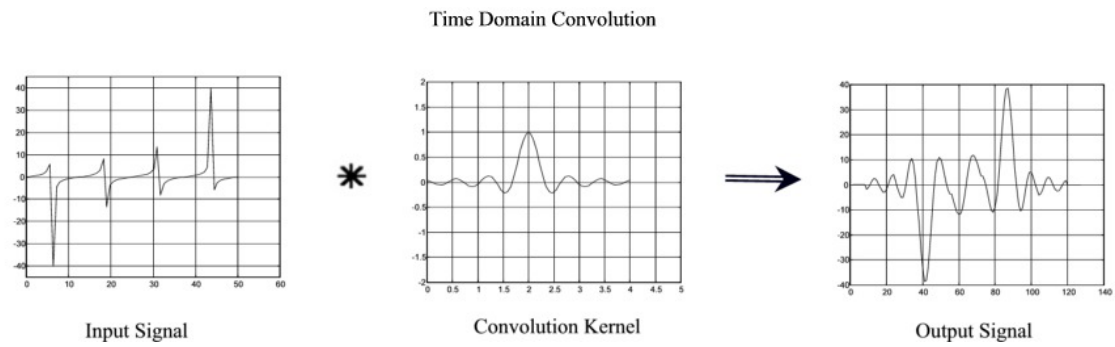


Figure 3.1 Single dimensional convolution in Time domain

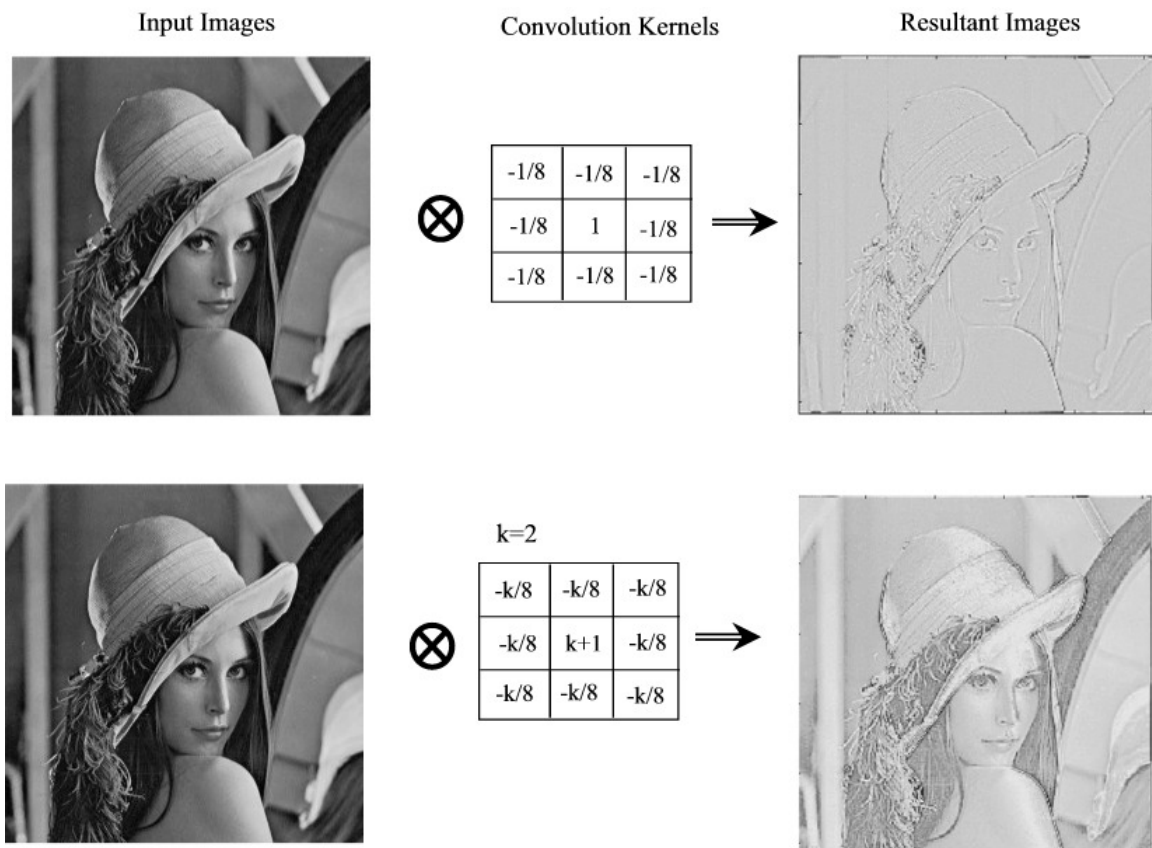


Figure 3.2 Two-dimensional convolution in Time domain

kernels. The convolution kernels are for edge detection and edge enhancement respectively.

3.1.1 FFT Convolution

There are many methods to calculate convolution. Calculating using the above formula directly is time consuming for larger datasets of signals/images. FFT convolution uses the principle that *multiplication* in the frequency domain corresponds to *convolution* in the time domain. The input signal is transformed into the frequency domain using the DFT, multiplied by the frequency response of the filter, and then transformed back into the time domain using the Inverse DFT. By using the FFT algorithm to calculate the DFT, convolution via the frequency domain can be *faster* than directly convolving the time domain signals. For this reason, FFT convolution is also called *high-speed convolution*.

The following equation is a mathematical formula for FFT convolution

$$y[n] = x[n] * h[n] = FFT^{-1}[FFT(x[n]) \times FFT(h[n])]$$

Figure 3.3 shows the FFT convolution steps. First we take FFTs of Signals (a) and (b) to be convolved using FFT convolution technique. (c) and (d) show the magnitude and phase of the FFTs of the signals (a) and (b). These are multiplied together (e) and an inverse FFT operation is performed to get the convolved output. (f) shows the magnitude and phase of the convolved output.

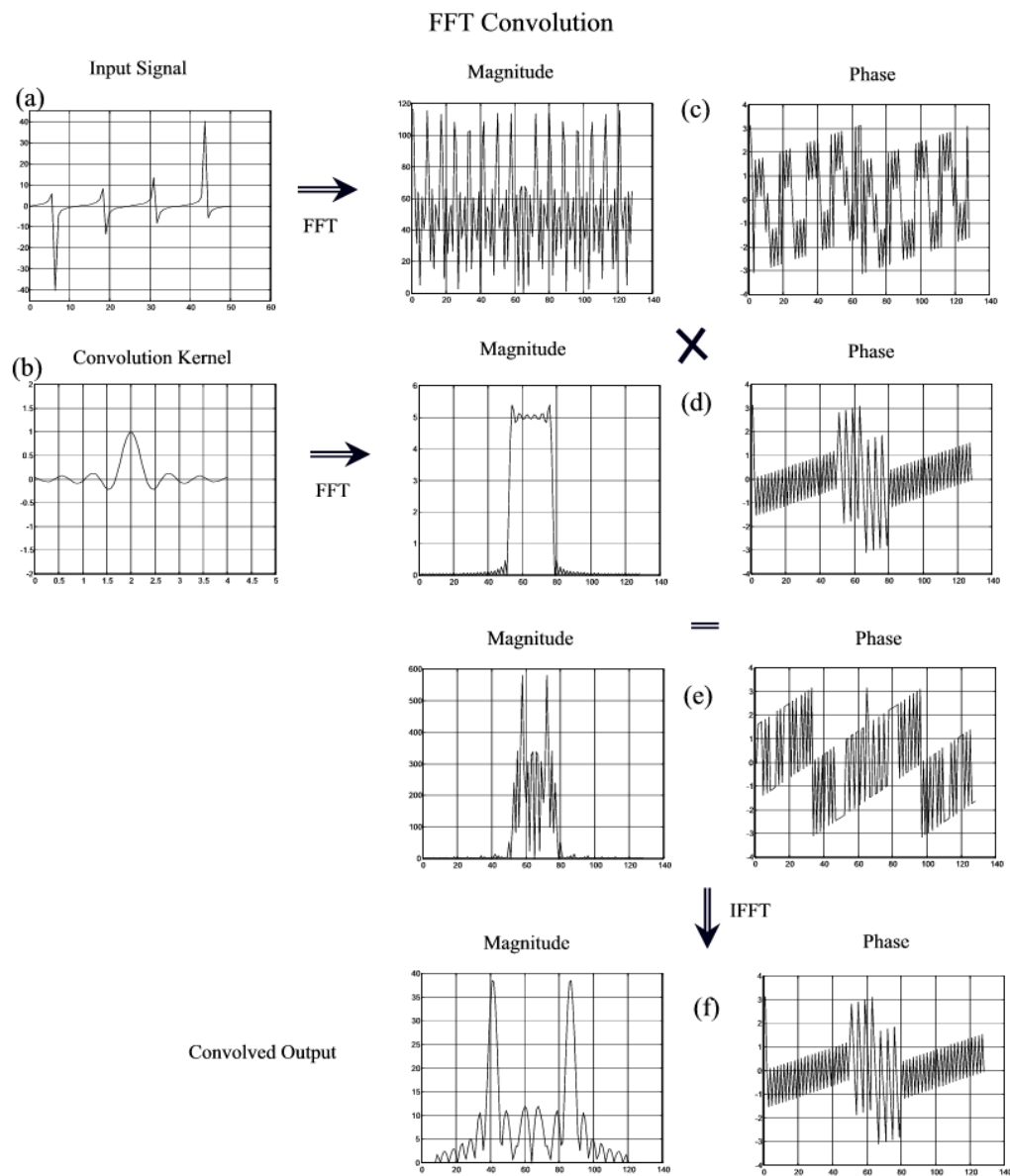


Figure 3.3 FFT Convolution of two signals

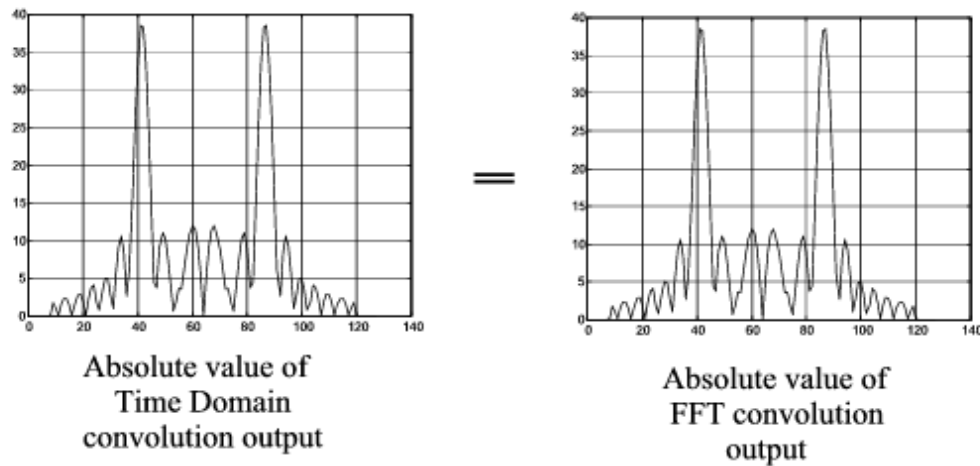


Figure 3.4 Convolution outputs using direct calculation and FFT convolution method

Note that the convolved signals are same as the ones convolved using the formula for convolution as was shown in figure 3.1. Figure 3.4 shows the absolute value of the outputs calculated using both methods to be equal.

The DFTs used must be long enough that *circular convolution* does not take place. This means that the DFT should be the same length as the output signal. So, if input signals are N and M points in length then the output signal will be $N+M-1$ points long. Hence DFTs used should at least be $N+M-1$ points long. For instance, in the example of figure 3.3, the filter kernel and the signal contains 64 points each. Hence the DFT used should be 127 points in length at least. Since we are using FFT algorithm for DFT calculations we use a 128-point FFT. This means that the input signals need to be padded with zeros to bring it to a total length of 128 points.

For two-dimensional convolution using FFT we need to use two-dimensional FFT algorithm to obtain the frequency domain representation of the images. Figure 3.5 shows the results obtained using two-dimensional FFT convolution.

From the data provided in ‘*The Scientist and Engineer's Guide to Digital Signal Processing*’ by Steven W. Smith [39], in one-dimensional convolution, the time taken by the standard convolution is directly proportional to the number of points in the filter kernel. In comparison, the time required for FFT convolution increases very slowly, only as the *logarithm* of the number of points in the filter kernel. This is shown in Figure 3.6 [39]. In case of image convolution the execution time required for FFT convolution does not depend on the size of the kernel, resulting in flat lines in the graph of figure 3.7 [39]. A 128×128 image can be convolved in about 15 seconds using FFT convolution, while a 512×512 image requires more than 4 minutes on a 100 MHz Pentium personal computer. The execution time for FFT convolution is proportional to, $N^2 \log_2(N)$, for an $N \times N$ image. That is, a 512×512 image requires about 20 times as long as a 128×128 image. Conventional convolution has an execution time proportional to $N^2 M^2$ for a $N \times N$ image convolved with a $M \times M$ kernel. In other words, the execution time for conventional convolution depends *very strongly* on the size of the kernel used. As shown in the graph, FFT convolution is faster than conventional convolution

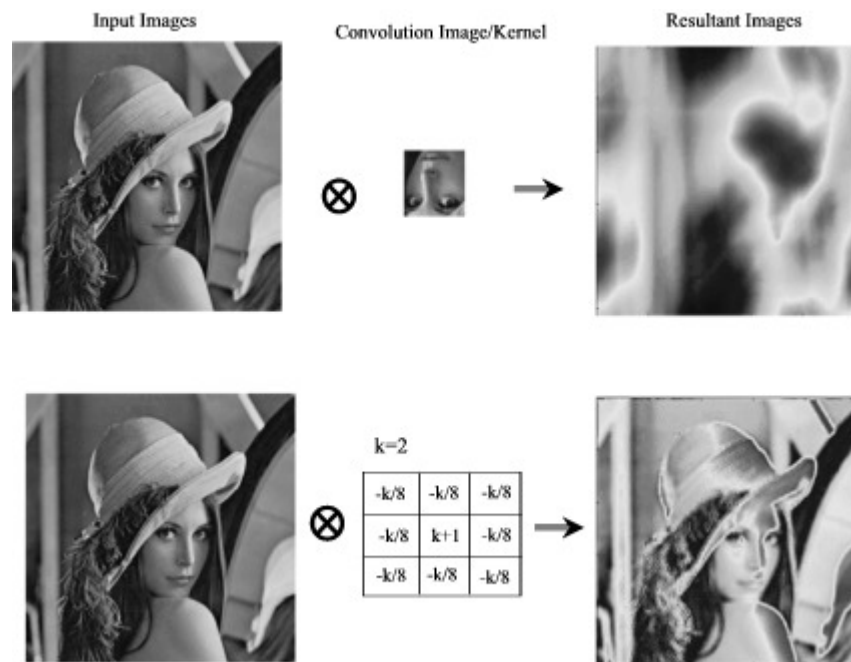


Figure 3.5 Two-dimensional FFT convolution

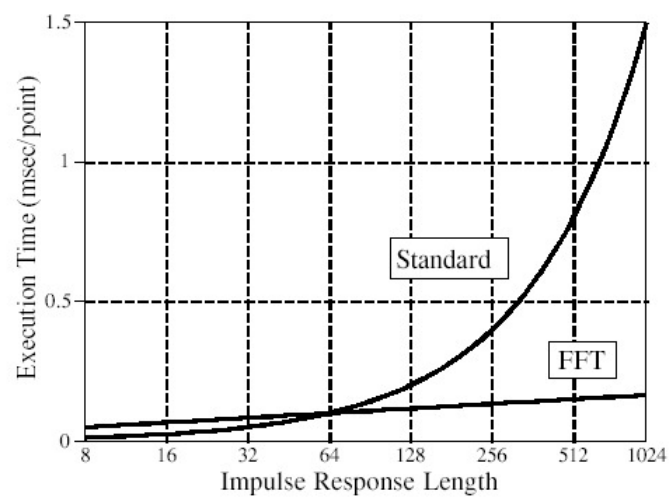


Figure 3.6 Execution Times for FFT and Standard Signal Convolutions [39]

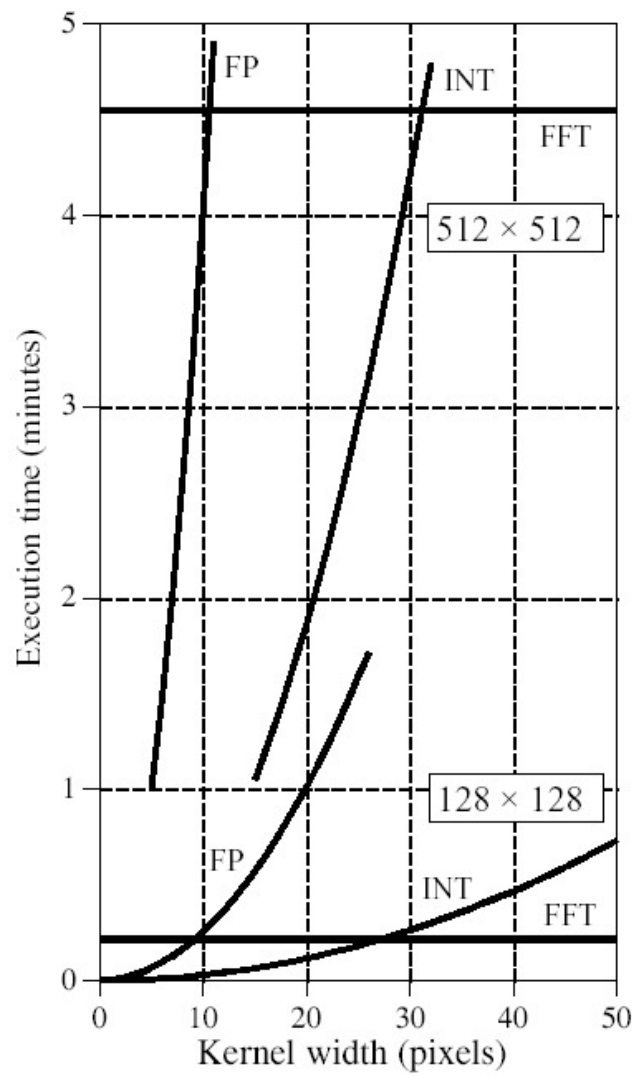


Figure 3.7 Execution times for Image Convolution [39]

using floating point if the kernel is larger than about 10×10 pixels. The concept to remember is that FFT convolution is only useful for *large* filter kernels.

3.2 Correlation Function

Correlation between two signals gives the extent by which the two signals are correlated or are similar to each other in any aspect. Correlation function is a mathematical expression of how correlated two signals are as a function of how much one of them is shifted. The correlation function between two signals can be mathematically stated as –

$$R_{xy}[m] = \sum_{j=0}^{M-1} x[j-m]y[j]$$

$R_{xy}[m] \Rightarrow N + M - 1$ point Correlation Function

$x[m] \Rightarrow N$ point signal

$y[m] \Rightarrow M$ point correlation kernel

A very close similarity exists between the convolution and correlation equations. Only difference between the two is that, in convolution one of the signals is flipped in time (i.e. time inverted). Hence the same algorithms can be used to calculate correlation function as were used to calculate convolution, with only difference being in omitting the flipping step in correlation calculations. Both convolution and correlation may be mathematically very similar, but they shouldn't be confused to be similar in physical significance. They both have very different significances and uses.

As in the case of convolution, we can correlate images. ‘Image Correlation’ is a machine vision technique that compares a template of the desired image (the correlation kernel) with the actual camera image of an object and generates a new image (the correlation image) that indicates where the template matches the camera image. This has many applications and can be used for part location and gauging, feature or flaw detection, character recognition and rectification, target recognition, terrain recognition etc. Figure 3.8 illustrates one example of image correlation.

More information on correlation and convolution can be found in ‘*The Scientist and Engineer's Guide to Digital Signal Processing*’ by Steven W. Smith [39].

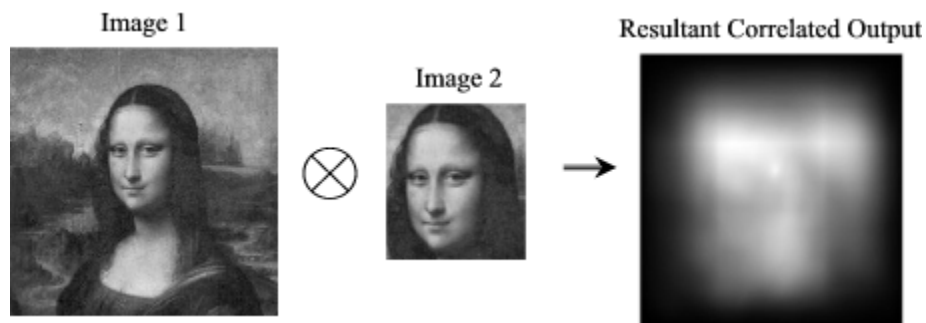


Figure 3.8 Image Correlation Example

3.3 Implementation on HPRC

The convolution and correlation algorithms discussed in the earlier sections have a wide area of application in image processing and signal processing fields. Applications like target recognition, face recognition, character recognition, unknown terrain explorations, medical imaging etc. extensively use these techniques. For example in target recognition the target image obtained by the camera eye needs to be correlated with every single image in a huge database of images and analyzed for amount of correlation. Hyper spectral images of target can be correlated with a database of different hyper spectral images to identify different objects. A lot of computational power is needed to execute these applications in real time. Thus correlation of images forms a good case study for implementation on HPRC using the approaches discussed the earlier chapter.

3.3.1 Library Based Approach

In the library based approach a MATLAB[®] program acts as a master process calling functions that dynamically link with computational C routines. The C routines then may spawn of multiple processes to execute various pieces of the application. These processes can also invoke MATLAB[®] engine on one or more other computing nodes to perform computations. The computational C routine can also directly interface with pilchard boards to perform computations on hardware.

As an example, character recognition has been implemented here [40]. In character recognition technique a source image containing the text is convolved with a target image (correlation kernel). Target image is the sequence of characters to search, in our case, ‘MATLAB®’. Figure 3.9 shows the source, target and the resultant correlated image. The following are the sequence of steps involved. The ‘*showimage()*’ function shown is just a dummy function to display image.

$$\begin{aligned} \text{Im}_{\text{Source}} \otimes \text{Im}_{\text{target}} &= \text{Im}_{\text{Corr}} \\ \text{i.e. } \text{Im}_{\text{Corr}} &= \text{fft}2^{-1} \{ \text{fft}2(\text{Im}_{\text{Source}}) \times \text{fft}2(\text{Im}_{\text{target}}) \} \\ &\text{showimage}(\text{Im}_{\text{Source}}) \\ &\text{showimage}(\text{Im}_{\text{target}}) \\ &\text{showimage}(\text{Im}_{\text{Corr}}) \end{aligned}$$

As can be seen the output image doesn’t identify the exact positions of the target and is blurred wherever the characters are present. To find the exact position of the target image we normalize the correlated output to set the pixel values between 0.0 and 1.0, and isolate the points of maximum correlation. We find a maximum value in the normalized correlated output and set a ‘threshold’ value about 5% lesser than the maximum value. We display the all the points of the output correlated image that are greater than the threshold value to see the exact positions of the target image in the source. This is indicated by the white dots in the final image. This can be seen in figure 3.10. The steps to show the final output image are –



Figure 3.9 Character recognition technique example

Final Image indicating the position of the target

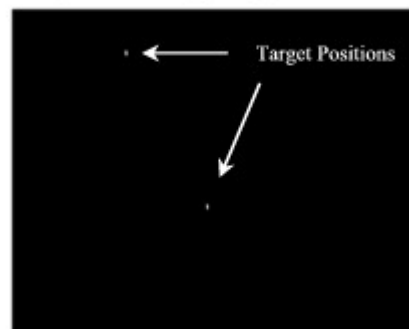


Figure 3.10 Position of the target in source image as indicated by the white dots

$$\begin{aligned}
sq_Im_{source} &= (Im_{source})^2 \\
Flat_Im_{target} &= ones(size(Im_{target})) \\
norm1_Im_{corr} &= sq_Im_{source} \otimes Flat_Im_{target} \\
norm_Im_{corr} &= \frac{Im_{corr}}{norm1_Im_{corr} + 0.1} \\
threshold &= 0.05 \times \max(norm_Im_{corr}) \\
showimage(norm_Im_{corr} > threshold)
\end{aligned}$$

A C Mex file, '*corr.mexsol*', is used to compute the correlated image of two input images, source and target. Function '*corr*' integrates PVM and MATLAB[®], takes as inputs the two images, an output data file name, and the name of the remote computing node on which to perform the correlation computation. It spawns off a slave process that computes correlation on a remote node. Control is returned to MATLAB[®] program, making a function call to '*corr*', immediately after the slave process is spawned. The slave process invokes MATLAB[®] engine and computes correlation. It saves the resultant image in an output data file specified in the function call to '*corr*', in .mat format. Flowchart in figure 3.11 details this process. Function '*corr*' can also interface with FPGA hardware on pilchard for computations.

The above method was applied to search for vowels 'a', 'e', 'i', 'o' and 'u' in the text of image in figure 3.12. The figures for the resultant final output locations are kept in appendix to maintain the flow of the text. Function '*corr*' was called multiple times in a loop supplying it with different sets of input images

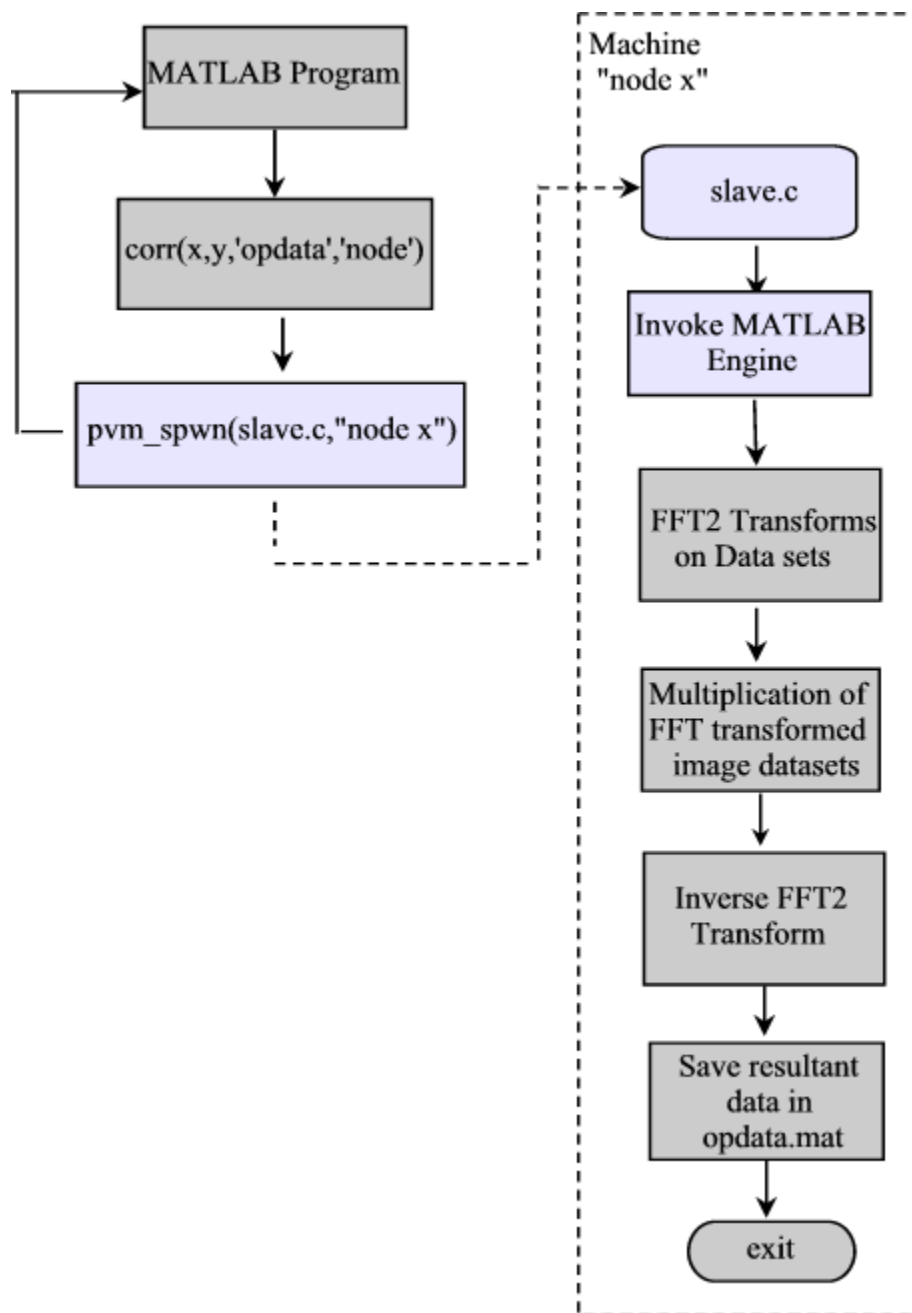
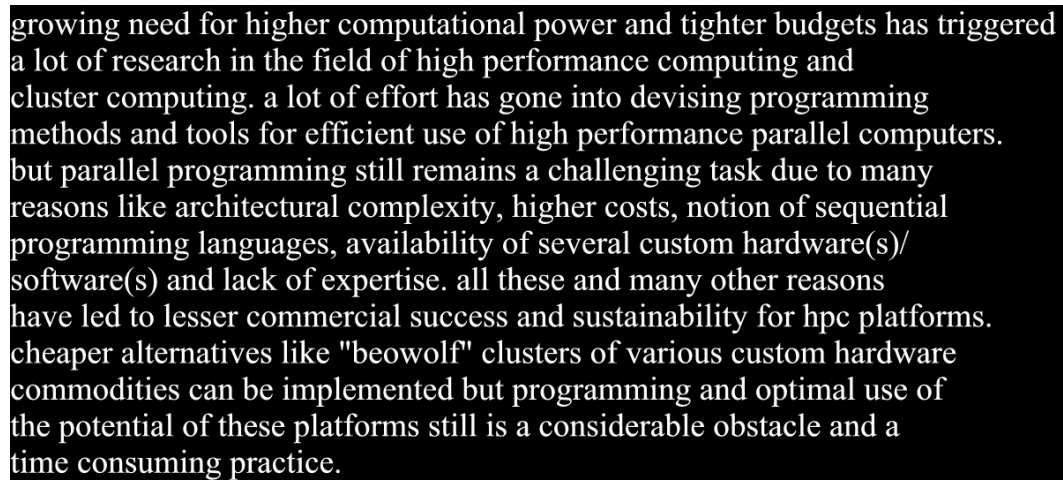


Figure 3.11 Flowchart explaining the library based approach applied to image correlation



growing need for higher computational power and tighter budgets has triggered a lot of research in the field of high performance computing and cluster computing. a lot of effort has gone into devising programming methods and tools for efficient use of high performance parallel computers. but parallel programming still remains a challenging task due to many reasons like architectural complexity, higher costs, notion of sequential programming languages, availability of several custom hardware(s)/software(s) and lack of expertise. all these and many other reasons have led to lesser commercial success and sustainability for hpc platforms. cheaper alternatives like "beowolf" clusters of various custom hardware commodities can be implemented but programming and optimal use of the potential of these platforms still is a considerable obstacle and a time consuming practice.

Figure 3.12 Source Text Image

and different processing nodes. As in the above case, function '*corr*' spawns off a slave process on a specified node and returns the control to MATLAB[®] calling program which loops and again makes a call to function '*corr*', with new sets of parameters. The spawned slave processes compute the correlation, save the data in an output data file and exit.

3.3.2 C as a Master

The above approach is very suitable for an end user who is not interested in details of underlying cluster architecture. Just a function call from MATLAB[®] would spawn a process on some remote machine and compute correlation. Thus, MATLAB[®] acts as a master process responsible for spawning various tasks on remote machines. The second approach is that of keeping the management task with a master C process. In this approach a master C program spawns of various

slave tasks at different nodes in a cluster of machines and manages the processes. Each slave task computes the correlation between a set of two images and sends back the result to the master process. The slave process invokes MATLAB[®] engine on a remote computing node and executes MATLAB[®] functions to compute correlation. The slave can also interface with pilchard boards connected to the computing nodes and perform computations on the reconfigurable FPGA hardware. For this application one-dimensional 1024 point FFT was implemented in hardware, which is required for computation of correlation. Figures 3.13 and 3.14 illustrate the process.

The same application of recognition of vowels ‘a’, ‘e’, ‘i’, ‘o’ and ‘u’ in the text of figure 3.12 was repeated here and the results obtained are exactly same as in the earlier case, the difference being in the speedup.

3.3.3 Hardware Implementation

For this application one-dimensional 1024 point FFT was implemented on FPGA hardware on pilchard boards. For the actual FFT computations inside hardware Xilinx Intellectual Property (IP) core was used along with Virtex Block RAMs. The block diagram of the architecture is attached in the appendix. Figure 3.15 shows the communication process between the slave process and the FPGA to calculate two-dimensional FFT from one-dimensional implementation on hardware. The image to be correlated is read by MATLAB[®] and the data is passed on to the pilchard board using the pilchard API function `write64()`. Due to

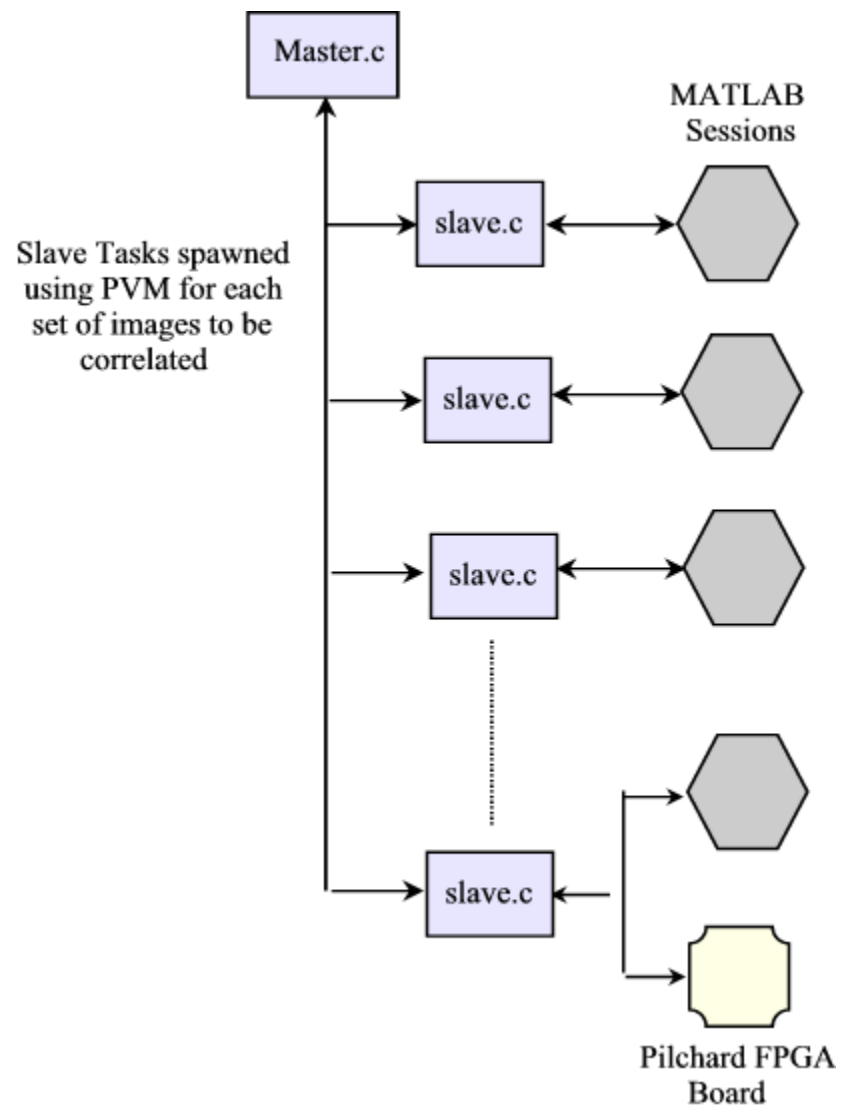


Figure 3.13 Using approach II - C as a master process

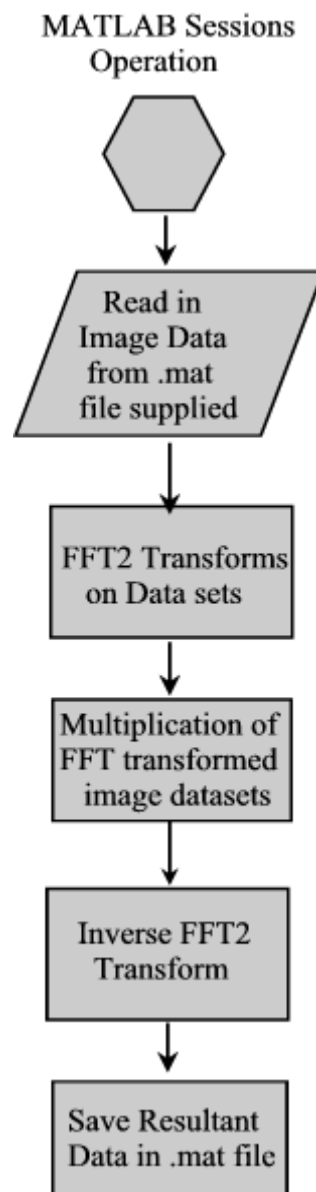


Figure 3.14 Details of the MATLAB[®] sessions invoked by slave process

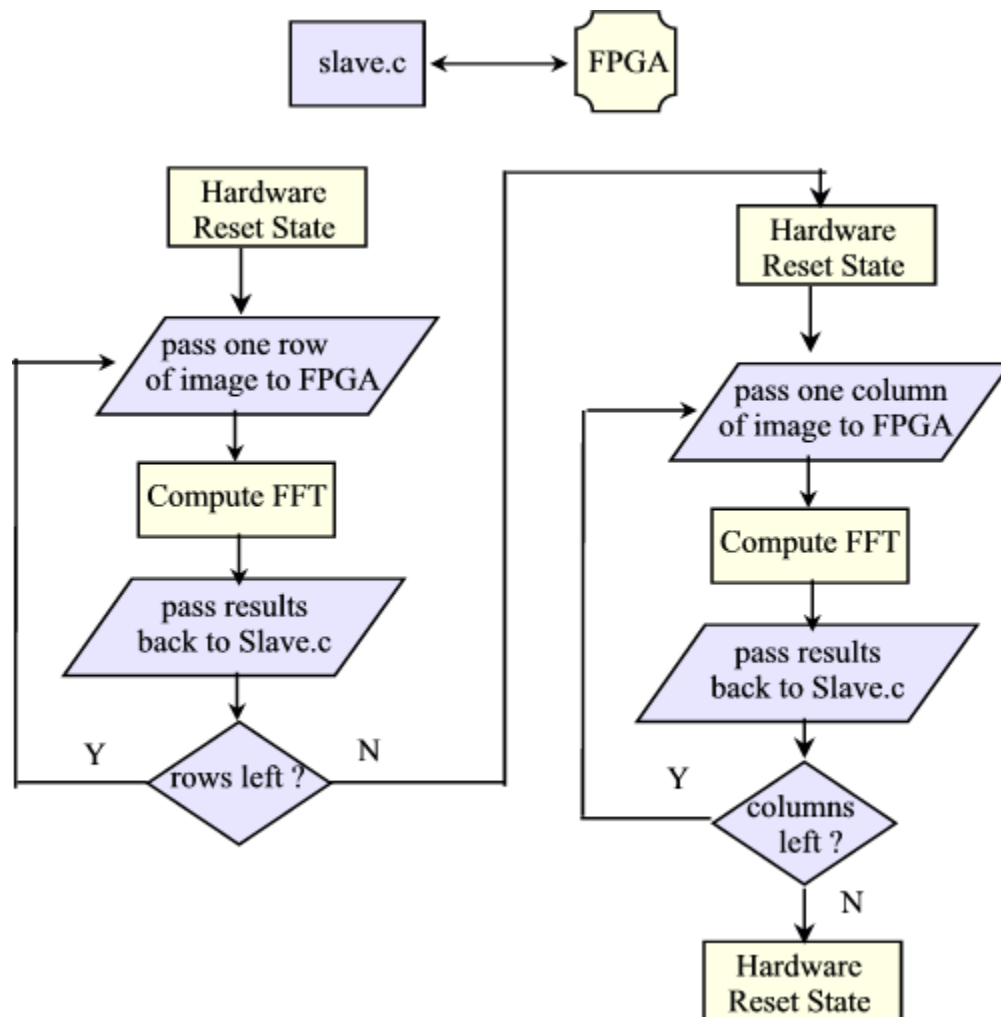


Figure 3.15 Communication between Slave process and the FPGA

limitations of the number of locations that can be addressed using the address bus provided on pilchard boards, a work around solution has been implemented. The data passed in is 16-bits each, i.e. the real part and imaginary part (which is actually set to zero initially). The address is passed on the data bus itself along with the data instead of using the address bus due to the addressing limitations. Since the data bus is 64 bits in length it can accommodate both the 16-bit data and the 10-bit address in a single write operation. A row of image is passed on to the hardware for each computation of FFT. It is padded with zeros in case if needed to make the length to 1024 points. Thus, 1024 write operations are performed after which the computation is started on the hardware. The computation takes 6200 clock cycles to complete and is run at 25 MHz clock. The results stored in the Block RAMs on Virtex FPGA are read back using `read64()` API function and are over written on the input row that had been supplied for FFT computation. FFT for every row is computed and overwritten by the results. Once all the rows are over the columns of data are send in similarly to compute the two-dimensional FFT.

The FFT on the hardware can also be directly run from MATLAB[®] as a function call using the approach discussed in the earlier section. Thus, a call to function '*myfft*' from a MATLAB[®] program would compute the FFT on the hardware and return the results back to the calling MATLAB[®] program. Figures 3.17 and 3.18 show the result of FFT computed for a square wave of figure 3.16 by both FFT on the hardware, and using MATLAB[®] toolbox function '*fft*'. The

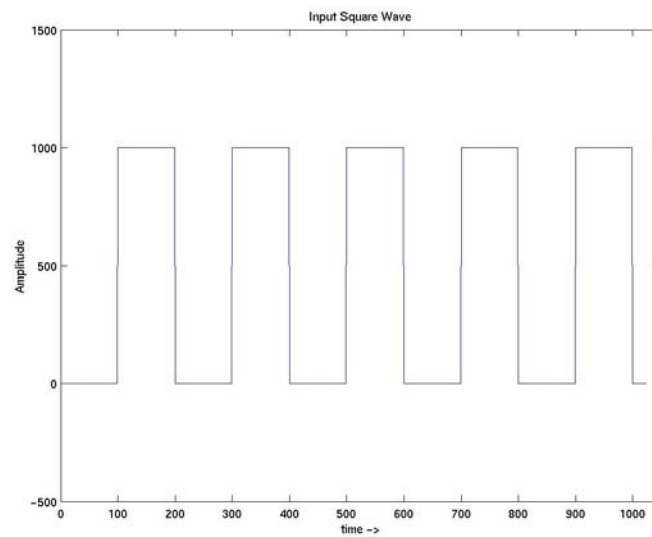


Figure 3.16 Input Square Wave

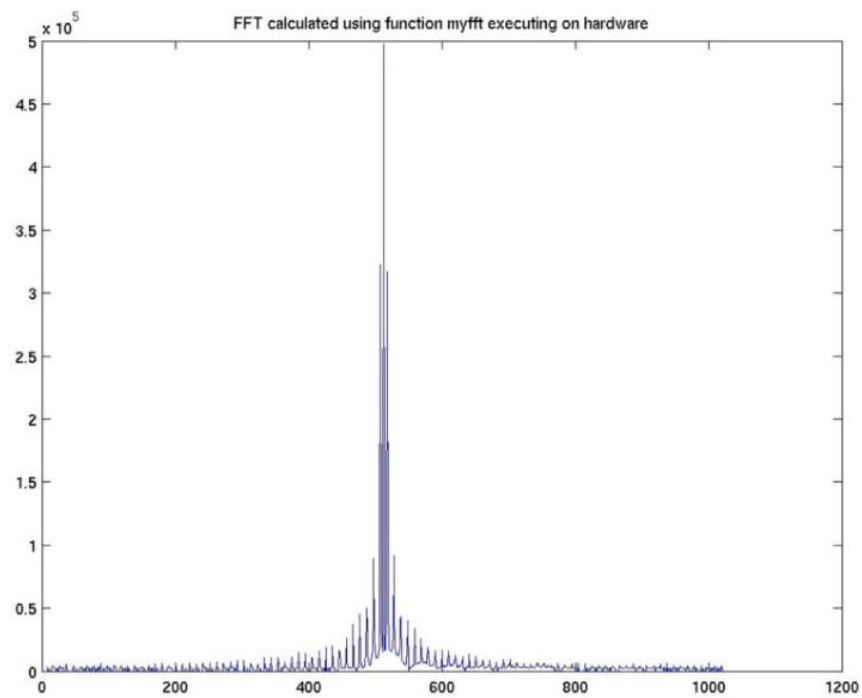


Figure 3.17 FFT calculated using hardware implementation

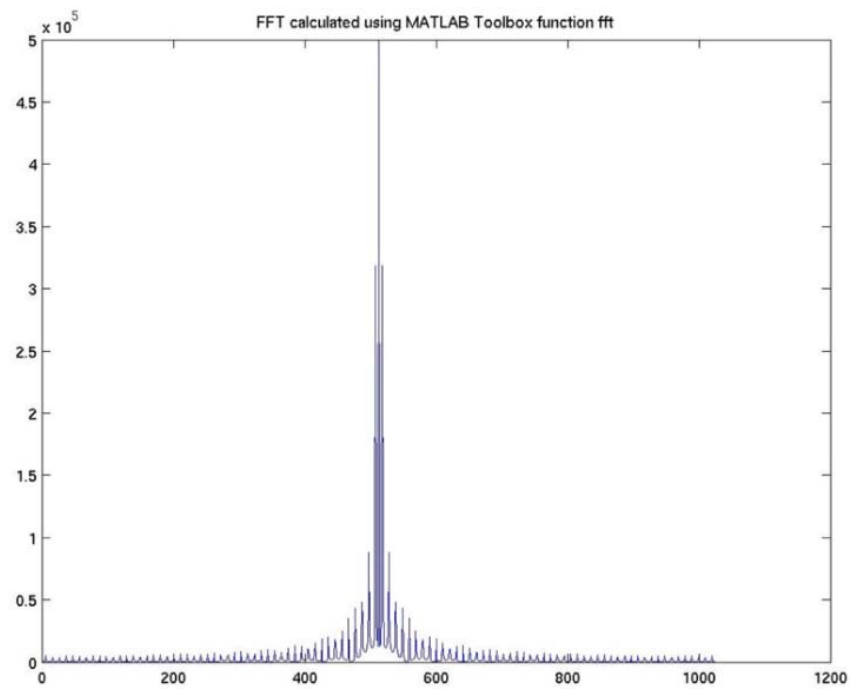


Figure 3.18 FFT calculated using MATLAB toolbox function

results are identical within round off errors. The layout diagram on Virtex1000e part is attached in the appendix.

3.3.4 Results

The execution times recorded with both the approaches are shown in table 3.1. Graph in figure 3.19 shows the execution times of serial as well as both of the parallel approaches. The speedups with both the approaches are as below –

$$speedup_{approachI} = \frac{86.2319}{37.9642} = 2.2714$$

$$speedup_{approachII} = \frac{86.2319}{41.7133} = 2.0673$$

Table 3.1 Execution times for serial and parallel executions

<u><i>Mode</i></u>	<u><i>Execution Time in secs</i></u>
Serial	86.2319
Parallel - Approach I	37.9642
Parallel – Approach II	41.7133

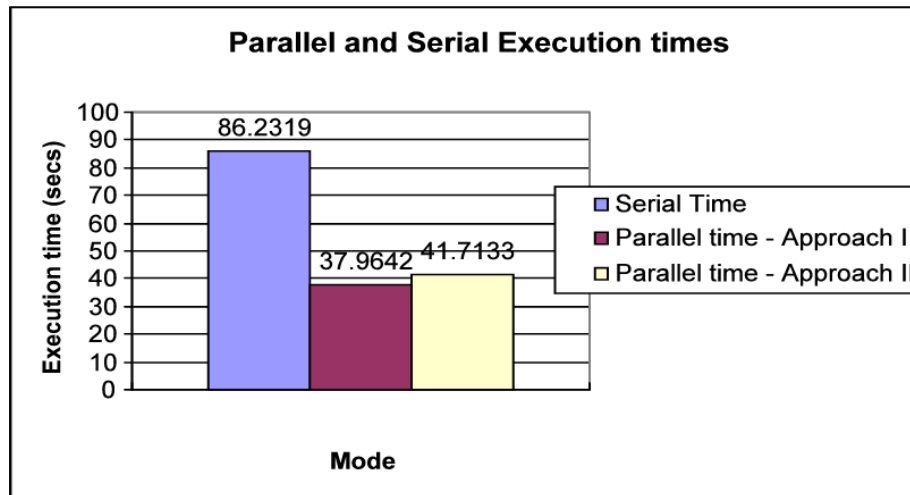


Figure 3.19 Graph of Execution times

Hardware Specifications:

Machine hardware - sun4u

OS version - 5.8

Processor type – Sparcv9 @ 450 MHz, Dual processors

Hardware - SUNW, Ultra-60

Memory – 2048 Mbytes

3.3.5 Limitations

There is an issue yet unresolved with the hardware implementation of FFT. It gives erroneous results sometimes on multiple iterations. Hence the results with reconfigurable card employed for computations are not available as of now.

4 Case Study II – Artificial Neural Network Training Speedups

4.1 Introduction to Artificial Neural Networks (ANN)

An Artificial Neural Network (ANN) is a massively parallel, distributed processor that has a natural propensity for storing exponential knowledge and making it available for later use. The network consists of many interconnected ‘Neurons’- the basic processing unit of a neural network. A basic diagram of a neuron is shown in figure 4.1. It has signal inputs $p(n)$ that are modified by weights $w(n)$ and fed to the processing unit, which gives the output $a = f(w \times p + b)$. The processing unit consists of two blocks as shown. The first one sums up the weighted inputs and feeds it to a monotonically increasing activation function $f()$. Also fed in to the activation function is a bias b . These neurons are grouped in layers and the layers are grouped into a network (figure 4.2). A neural network acquires knowledge through a process called ‘Learning’, also called as ‘Training’. This is a process by which a neural network’s free parameters are changed through a continuous process of stimulation by the environment. A neural network simply maps the inputs to the outputs. The neurons in a network can be trained to perform a desired mapping (‘Supervised Learning’) or they can create their own mappings (‘Unsupervised Learning’).

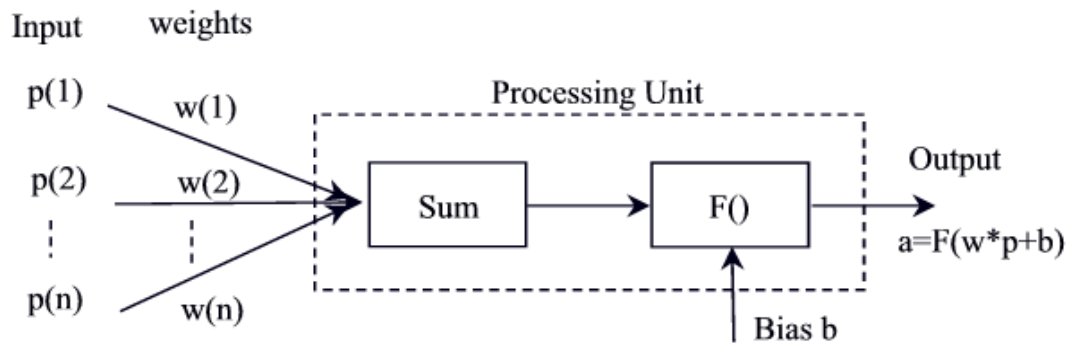


Figure 4.1 A Basic Neuron

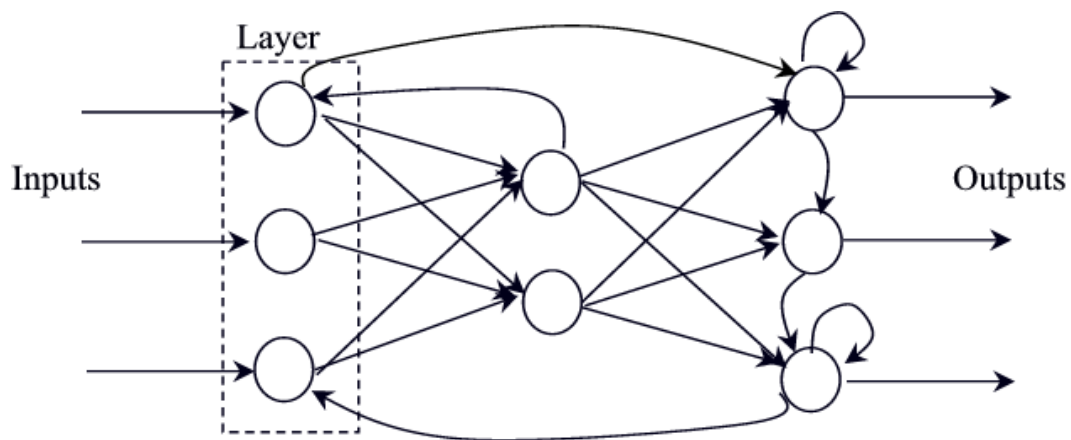


Figure 4.2 Example of a Neural Network

Depending on the activation function $f()$, some neurons are nonlinear. A network of such neurons can be highly nonlinear and can be beneficial in applications requiring nonlinearity. Neural networks have a potential to be fault tolerant since its performance only partially degrades from failure of a single neuron. Also, these can be implemented in hardware and the processing units be implemented in a parallel configuration[41]. Information on ANNs can be found in a book by Simon S. Haykin, “Neural Networks: A Comprehensive Foundation”[42] and many other technical literatures on the subject.

Artificial Neural Networks (ANNs) over the years have gained popularity in many application domains. Pattern classification, financial analysis, electrocardiogram analysis, speech or handwriting identification, credit card application reviews, insurance fraud, functional approximation, control systems, noise cancellation etc. are few of the examples of the vast variety of applications that artificial neural networks can address. Parallelism is one of the underlying principles of ANNs. Also, ANNs are time consuming, especially in the learning phase. A lot of research effort has gone into exploiting the inherent parallelism in ANNs and to speedup the learning phase by using reconfigurable or parallel computing techniques for a variety of architectures. Depending on the nonlinearity in the error surface, the size of the neural network being trained and the size of the data set, the training process can sometimes be very time consuming and often recursive, in order to realize an optimal network; usually the smallest and the most compact. Standard ANN training algorithms like the Back

Propagation, Levenberg-Marquardt algorithms are sequential in nature[42]. But to realize an optimal network design various different architectures of an ANN are trained and the smallest most efficient network is chosen. This is a recursive process and can be done in parallel simultaneously on various different nodes in a cluster of machines. Figure 4.3 shows a flowchart detailing the steps in a Neural Network training procedure.

A dataset that covers the operating region well should be chosen and split into a training and testing dataset. Care should be taken while splitting, such that both the training and the testing datasets cover the entire operating region of the network. Many times an odd-even split procedure is used. Depending on the application, neural network architecture is chosen. Statistically, most of the problems can be solved using a ‘Multi Layer Perceptron’ (MLP)[42] unless the application demands otherwise, like in our case study discussed below. Number of layers and hidden nodes (neurons) per layer are selected judiciously and the weights and biases initialized. The network is then trained to meet a specified error goal using a training algorithm. If the error goal is met then the training is successful; weights and biases are saved. Next a network smaller than the one trained is chosen and the training procedure is repeated until an optimized compact network is realized. In case if the training is not successful, the weights and biases are reinitialized and the training is repeated. There are various factors for which a network may not train successfully.

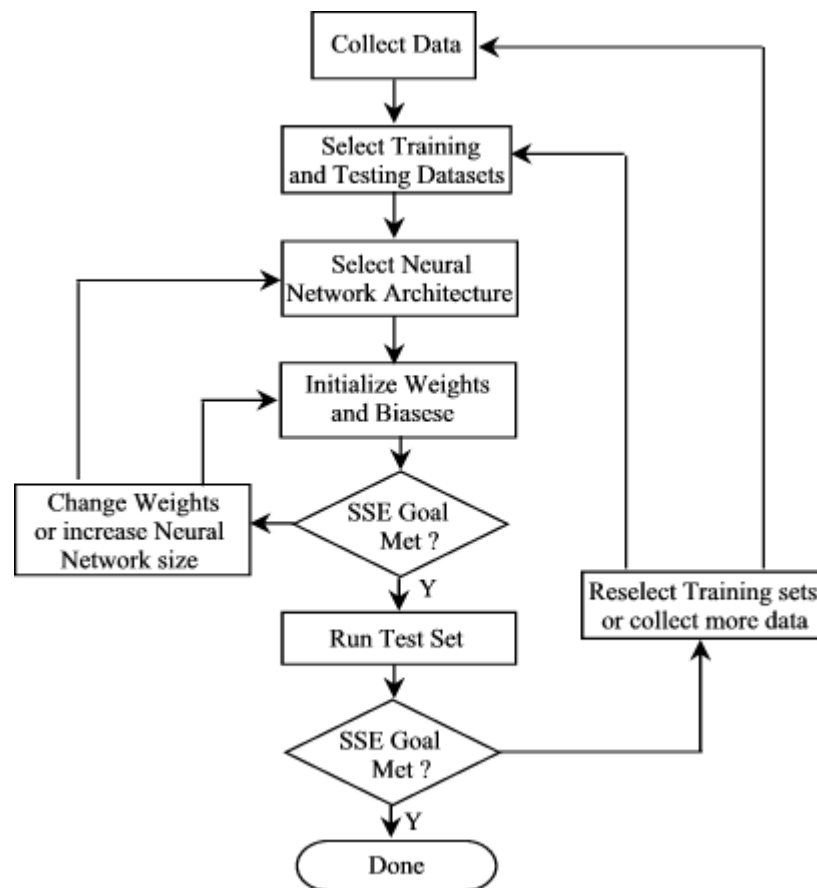


Figure 4.3 Flowchart of Neural Network Training Procedure

- The training gets stuck in the local minima. The best solution to this is to try to train the network again with a larger step size.
- The network does not have enough degrees of freedom to fit the desired input/output model. Hence more neurons need to be added.
- There is not enough information in the training data to perform the desired mapping. More training data may be needed.

As can be seen the training procedure is recursive and multiple network architectures need to be trained in order to realize the smallest network. The HPRC architecture platform can aptly be used to speedup the training process, by training multiple architectures simultaneously at different nodes.

4.2 Estimation of Solar Particle Event Doses: A Case Study

As a case study a data set on Solar Particle Event doses has been selected[43-46]. A Weibull model has been used to fit SPE dose and dose rate-time profiles. The Weibull equation is as follows.

$$D(t) = D_{\infty} (1 - e^{-(\alpha t)^{\gamma}})$$

D_{∞} – > Maximum Dose Value

$D(t)$ – > Dose value at time t

α & γ – > Fitting Parameters

The objective is to estimate the maximum radiation dose D_{∞} that an astronaut is likely to be exposed to in space during a particular Solar Particle

Event. Standard Multi Layer Perceptrons (MLP) can process only static mappings. Since the data has a temporal nature a Sliding Time Delayed Neural Network (STDNN) is used to estimate the maximum dose values. STDNN is a variant of Time Delayed Neural Network (TDNN) using a variable size of time delay τ . Figure 4.4 shows the STDNN[43].

The training procedures are written using functions in MATLAB[®] Neural Network Toolbox, Version 3.0[47]. This case study has only been ported to the computing nodes of the HPRC architecture. Reconfigurable hardware units have not been used in the implementation, as dynamic changes in the size of the network being trained would increase the over all execution cost due to multiple FPGA reconfiguration times. Different architectures of the ANN are simultaneously trained on various different computing nodes of the HPRC architecture using the approaches discussed in the chapter 2.

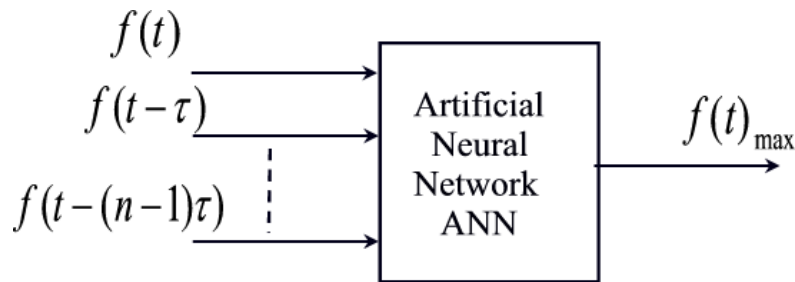


Figure 4.4 Sliding Time Delayed Neural Network[43]

In approach I a C – Mex file spawns of a child task which invokes a MATLAB[®] engine on a remote node that performs the training operation on a specific neural network architecture and returns the result to the calling MATLAB[®] session. Thus multiple calls are made to the C – Mex file to train different network architectures.

In approach II, a C parent program spawns of child tasks on various different nodes in a cluster of machines that in turn invoke MATLAB engine routines to run the ANN training functions written in MATLAB. Each child task trains a different architecture of ANN as specified in the *archspeg.m* MATLAB file. In case of successful training the resulting weights and biases are saved in an output *.mat* file. The child program notifies the results of training to the parent program, which in case of successful training kills all other child tasks that are currently training a larger network than the one successfully trained. The child tasks training a smaller more compact network than the one successfully trained continue training in an effort to realize a more compact network. If the training goal is not met the child program reinitializes the weights and biases and runs through the training again. This way, multiple different ANN architectures are trained simultaneously at various different nodes in a cluster of machines and the result of the training is made available to the parent program. Thus an optimal network is realized in less time and the training phase is shortened considerably.

Figure 4.5 shows the training process invoked on a remote node using approach I. Figure 4.6 shows the parallel training process using approach II.

The original dataset used is available from the National Oceanic and Atmospheric Administration. The time delayed input data for the STDNN was created by G. Forde *et al* [43]. The original data set was sampled at time $T = n\tau$ where n is the number of data points chosen along a particular Weibull Curve, which is actually the number of input neurons in the STDNN. Thus, in our case the number of input neurons is 5. τ is the sliding time delay of the interval. Thus dose values are obtained at arbitrary time intervals $t, t - \tau, \dots, t - (n - 1)\tau$. Figure 4.7[43] shows the sample selection. The input data is as shown in figure 4.8. It shows 106 events with 50 samples per event.

Since STDNN has no feedback paths standard feed forward training algorithms can be used to train the network. In our case Levenberg-Marquardt (LM) training algorithm is used for its faster and reliable convergence properties[42]. Since we are trying to predict the maximum dose value in an event, we only need to look at the portions of each event where the dose is still rising or has just peaked. Hence to decrease the training time we reduce our data set to remove the unwanted data discarding all the data beyond the 99% of the maximum dose value in the fourth time stamp. We thus reduce our dataset by 86.24% leaving in total 729 samples. Also, since our cost function is Sum of Squared Error (SSE) then some of the smaller events will be allowed to have large

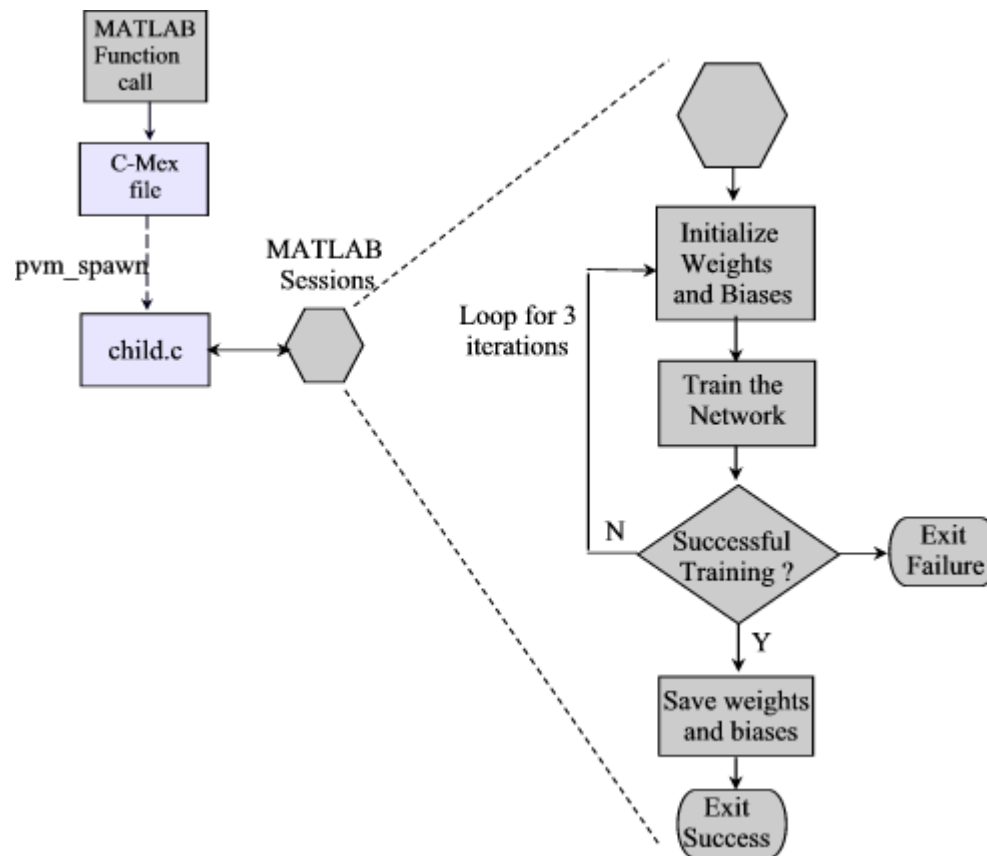


Figure 4.5 Flowchart of Training process using approach I

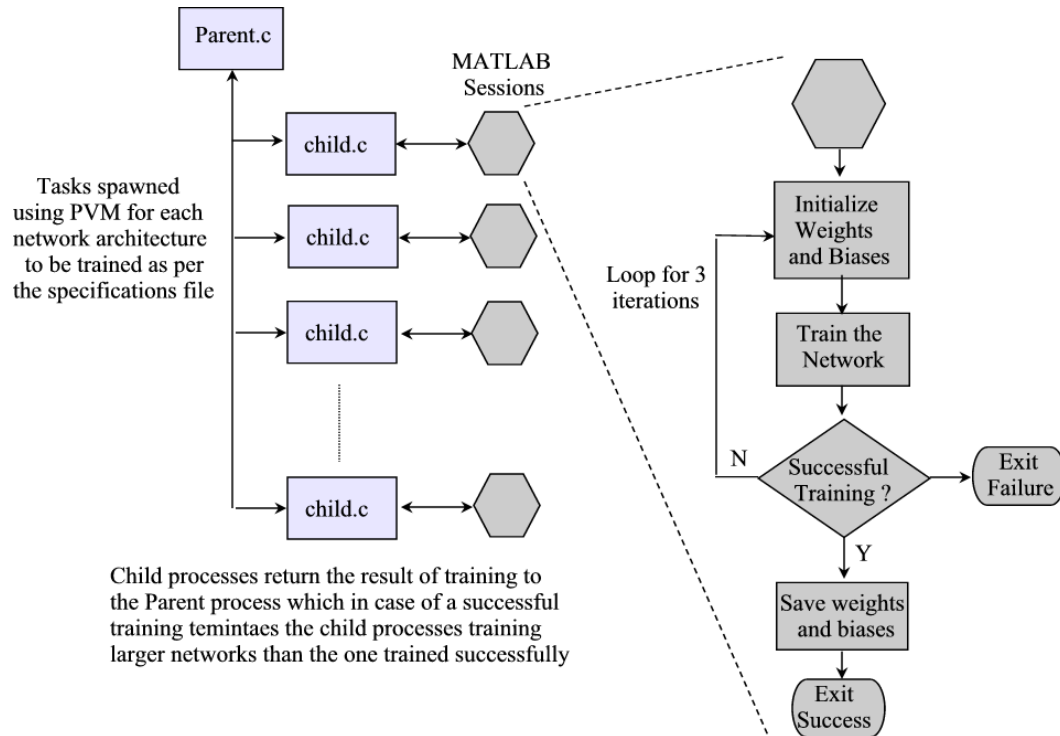


Figure 4.6 Flowchart of Parallel Training Process using approach II

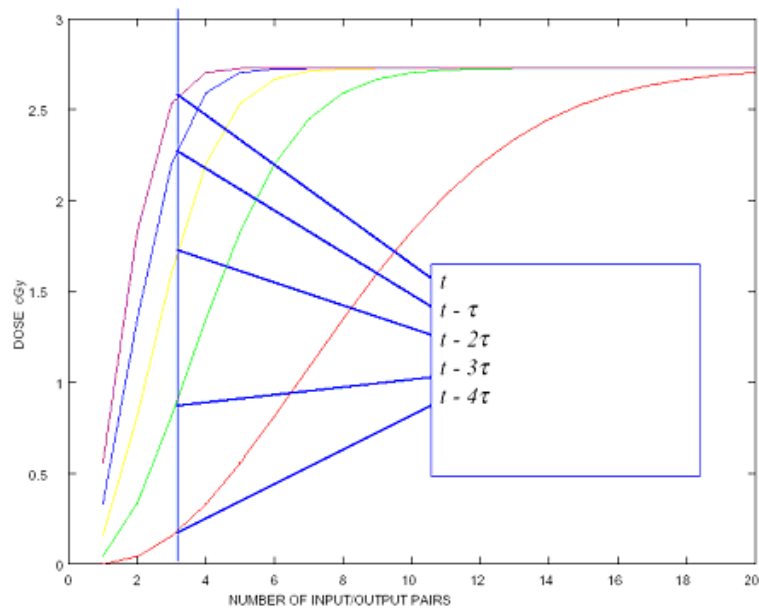


Figure 4.7 Illustration of Input Dataset Selection [43]

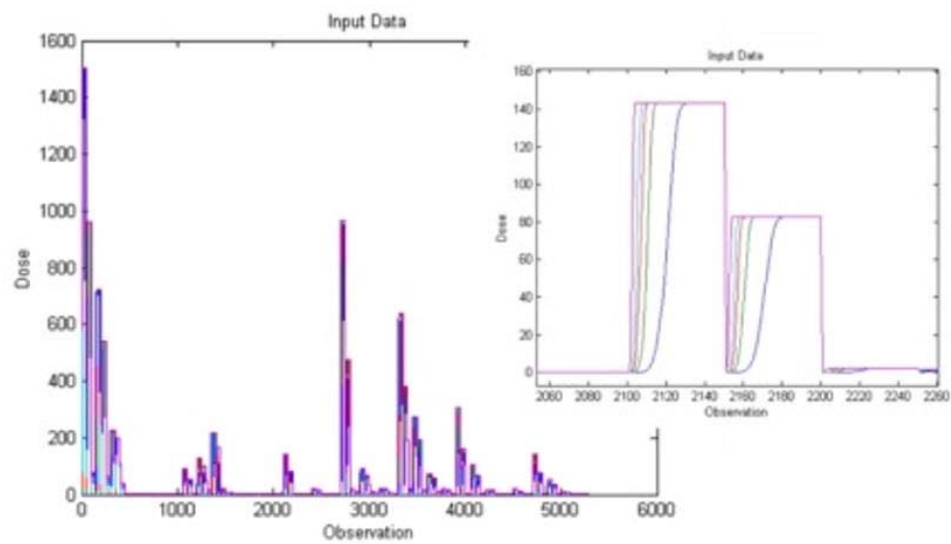


Figure 4.8 Input Data Set along with the zoomed in version on the right showing 2 particular events

percentage errors while the larger events will have small percentage errors even though the magnitudes will be similar. Hence we log scale the output data so that percentage errors are about the same. Figure 4.9 shows the target output and the log scaled target output. The data set obtained is divided into training set and testing set by putting a breakpoint at 620. So the first 620 samples will be used for training the ANN and the remaining for testing purposes. The training and testing dataset is saved in a .mat file. The hidden layer activation function used is Hyperbolic Tangent function. Since we are using ‘tansig’ hidden layer activation function we use z-score scaling (mean center, unit variance) on our input data so that the data is centered around zero. The network is trained for an error goal of 100, with maximum training epochs set at 2000. An error goal of 100 is reasonable as the need is to predict the approximate maximum dose that an astronaut will be exposed to in a particular solar particle event. So on an average about 5-10% difference in the predicted maximum dose is an acceptable

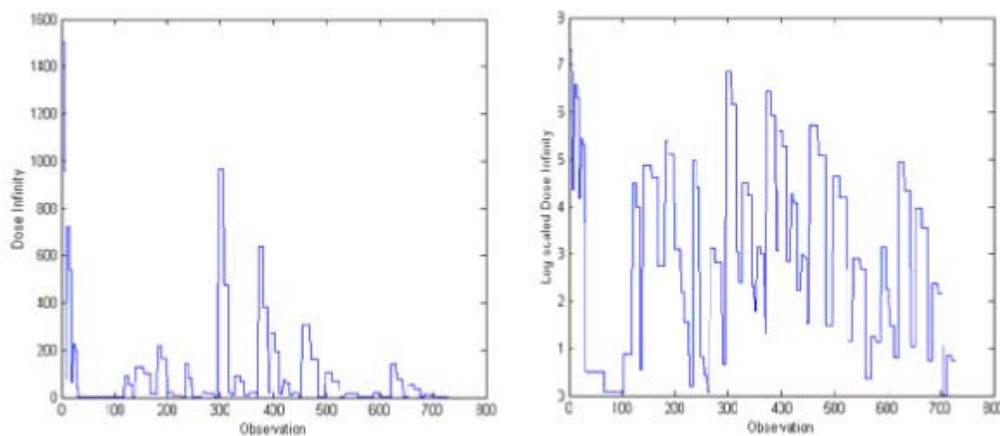


Figure 4.9 Target Output (Dose Infinity) (left) Log Scaled (right)

difference. All the network architectural specifications are saved in *spec.mat* file created using *archspect.m* script in MATLAB. The parent program is invoked and the data and spec file names specified. The parent program spawns multiple child tasks, which in turn invoke MATLAB engine routines and run through the training procedure.

4.3 Results and Discussion

Multiple network architectures were trained in parallel at various different nodes of a cluster of Sun Sparc machines. The hardware specifications are given below. The results obtained were similar using both the approaches and are tabulated in Table 4.1 and 4.2. For a single hidden layer network the smallest network that trained successfully was with 9 hidden neurons. Whereas for a 2 hidden layer network the smallest network trained successfully was with 5 hidden neurons. Thus the optimal network realized is single hidden layer with 9 hidden neurons, highlighted in Table 4.2. This network was tested with the test data saved in the .mat file. The resultant plots are as shown in the figure 4.10. A snapshot of the output screen for approach II is as shown in figure 4.11.

Hardware Specifications:

Machine hardware - sun4u

OS version - 5.8

Processor type – Sparcv9 @ 450 MHz, Dual processors

Table 4.1 Serial and Parallel Execution Times

<u><i>Mode</i></u>	<u><i>Execution Time in secs</i></u>
Serial	2245.6458
Parallel-Approach I	438.359
Parallel-Approach II	443.636

Table 4.2 Parallel Training Result. X- Unsuccessful Training; \sqrt - Successful Training

<u><i>Number of Hidden Layers</i></u>	<u><i>No. of Hidden Neurons per layer</i></u>	<u><i>Training Result</i></u>
1	5	X
	6	X
	7	X
	8	X
	9	\sqrt
	10	X
	11	\sqrt
	12	\sqrt
2	2	X
	3	X
	4	X
	5	\sqrt
	6	\sqrt

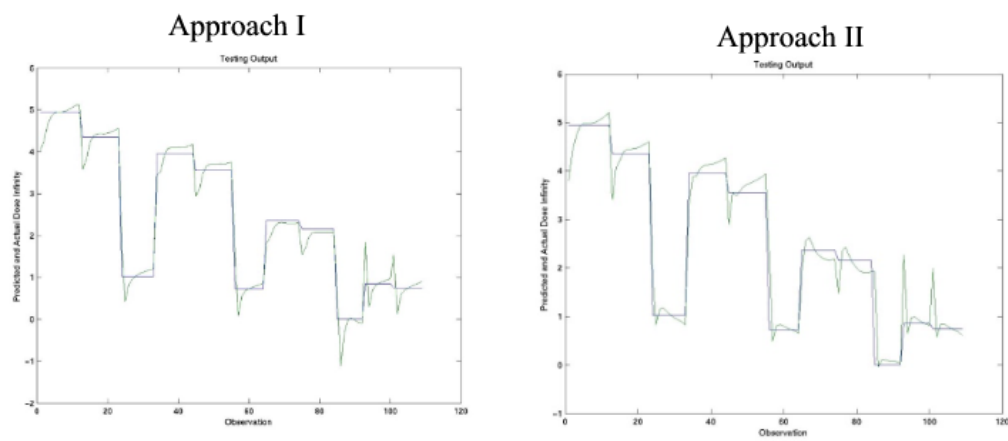


Figure 4.10 Testing Results of the selected optimal network highlighted in Table 4.1

```
faster:/home/smerchan/thesis/application_2 % parent
Enter the datafile name:
trdata
Enter the architecture specifications filename:
Spec
Joined group nnet
Joined group nnet
Joined group nnet
Joined group nnet
Joined group nnet
Joined group nnet
Joined group nnet
Joined group nnet
Training the network. Pls wait ....
Training goal couldn't be met with 5 hidden nodes
Training successful for 9 hidden nodes
Killing all other tasks with larger networks ...
Training successful for 11 hidden nodes
Killing all other tasks with larger networks ...
Training goal couldn't be met with 6 hidden nodes
Training successful for 12 hidden nodes
Killing all other tasks with larger networks ...
Training goal couldn't be met with 5 hidden nodes
Training goal couldn't be met with 7 hidden nodes
Training goal couldn't be met with 8 hidden nodes
Training goal couldn't be met with 6 hidden nodes
Training goal couldn't be met with 8 hidden nodes
Training goal couldn't be met with 7 hidden nodes
faster:/home/smerchan/thesis/application_2 %
```

Figure 4.11 Snap Shot of the Output Screen

Hardware - SUNW, Ultra-60

Memory – 2048 Mbytes

A cluster of nine such machines.

From figure 4.10 we can observe that we have obtained quite good estimates of maximum dose except for the 9th event, which is outside training space. The initial performance is poorer for most events but further in time the performance is quite good. This indicates that more training data is required for still better performance. It can be seen in figure 4.11 that in some cases the results of the larger network have been outputted even after the smaller networks have been already successfully trained and the child tasks with larger networks killed. This happens due to the following reasons:

- The computing speed of different nodes varies according the load on the particular node from other processes running simultaneously at the time of training. Thus a larger network being trained on a computing node, which can offer faster computing speed will train faster than a smaller network that is being trained on a node that can offer lesser computing time.
- The results are printed as soon as the parent process receives the notification from the child process. But there is no guarantee of receipt in order due to possible network delays between nodes. A child process could be running on a node, which is in a different subnet than the one on which the parent process is running.

The execution time shown in the results section is obtained using ‘*gettimeofday*’ Unix function, which doesn’t represent the exact CPU time taken by the training process due to process swapping and may vary with time. The performance of parallel training may vary according to the load on the network and computing nodes at the particular time of training. It is difficult to actually estimate the speedup of parallel training process over the conventional sequential training, as the latter needs a lot of user intervention in the recursive training process to achieve an optimal network design. The user would train one network and analyze the result and accordingly choose a network architecture for the next training process and may not necessarily train the network architecture in sequence followed by the automated process here

Figures 4.12, 4.13 and 4.14 show the serial and parallel execution times. Graphs in figures 4.12 and 4.13 show the individual execution times for training of each of the network topology serially. Graph in figure 4.14 shows the total serial execution time and the parallel execution times recorded using both the approaches. The times recorded may vary with different iterations and depend on the dataset trained.

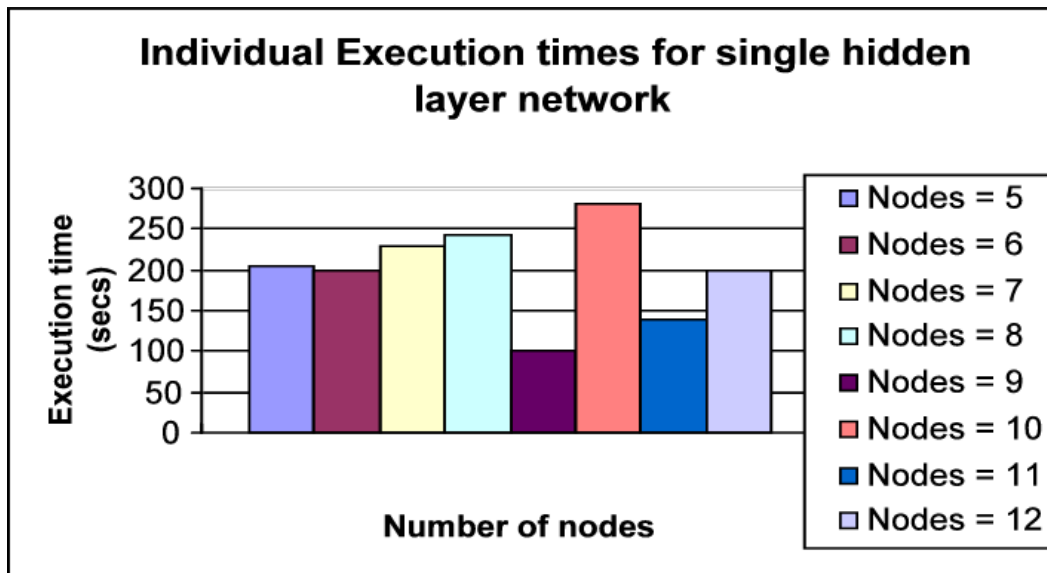


Figure 4.12 Individual Execution times with single hidden layer

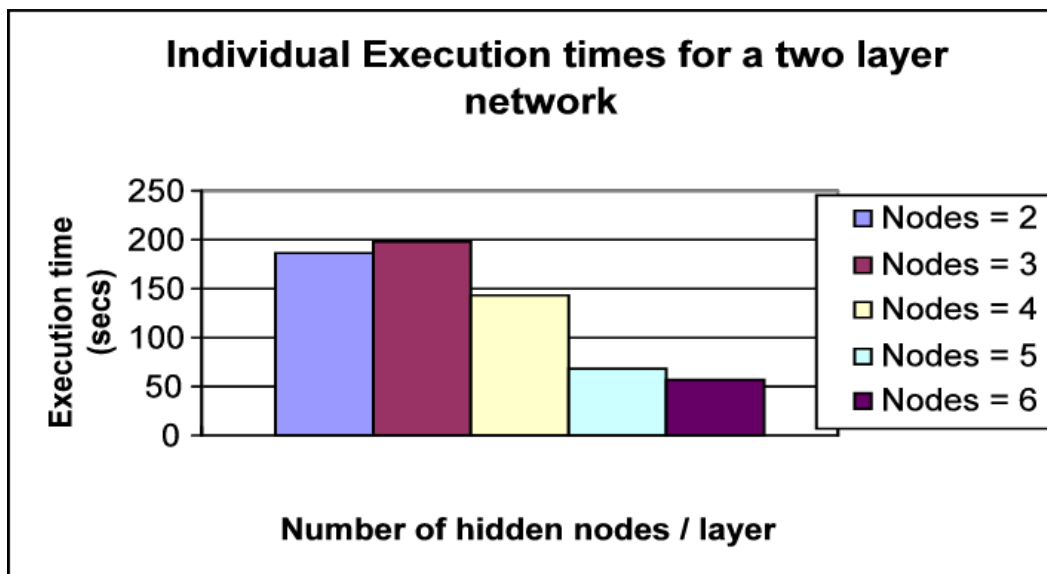


Figure 4.13 Individual Execution times with two hidden layers

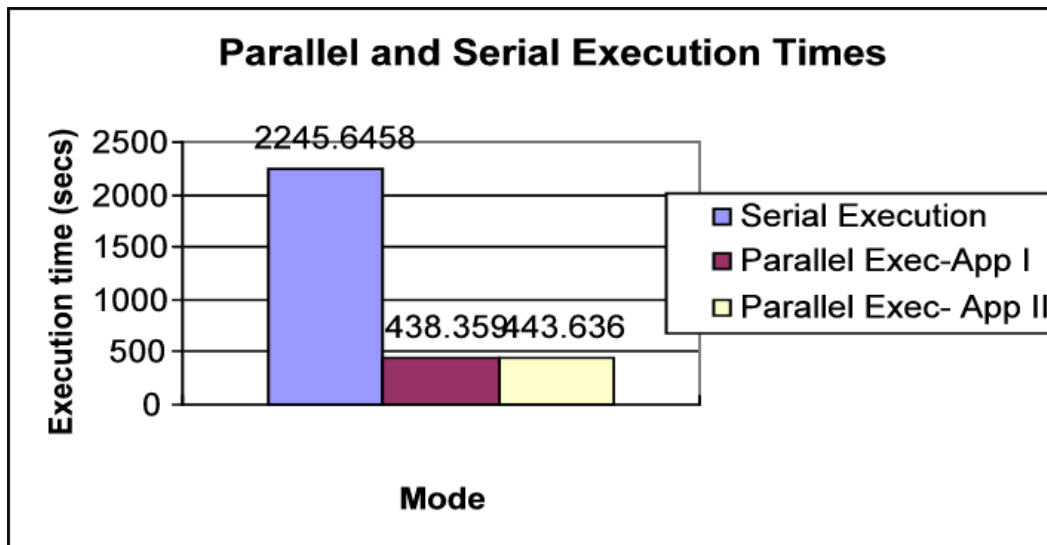


Figure 4.14 Serial and Parallel Execution times

The speedup obtained can be calculated as

$$speedup_{approachI} = \frac{2245.6458}{438.359} = 5.123$$

$$speedup_{approachII} = \frac{2245.6458}{443.636} = 5.062$$

The parallel training process as shown here is very convenient for a researcher who is training a network that takes a long time to train. The process is fully automated once the user specifies the range of network architectures he/she would like to train. This methodology is quite portable and can be adapted to

other neural network applications with little or no modifications on the C programming side at least. The user might need to edit the MATLAB training files to suit his/her particular application and be sure to adhere to the interface expected by the C code.

A lot of work can still be done further by completely automating the training process such that all the user needs to do is supply the training and testing data sets. The program automatically would analyze the data, select the appropriate training procedures and give the results or better, the program could give results with multiple training procedures for the user to compare and analyze.

5 Discussion And Conclusions

Various approaches to port MATLAB[®] applications to HPRC have been discussed in the earlier chapters. We do not have concrete data as yet for performance with the reconfigurable card also employed for computation. But the approach to directly execute MATLAB[®] functions on a remote reconfigurable hardware resource and receive the results back in MATLAB[®] has been clearly established and outputs shown in chapter 3 (Figures 3.617- 3.18). The results with parallel only computations are tabulated in the table 5.1. Using both the approaches discussed in the earlier section we have obtained speedups of about 2 in the first case study and about 5 times the serial execution time in the second case study. We would expect to obtain still higher speedups by increasing the problem size. Graph in figure 5.1 shows the execution times required by various pieces of the program in case study I. The `pvm_spawn` time and the MATLAB[®] invocation time should be about the same in both the case studies. The graph clearly shows the expense of invoking the MATLAB computational engine. We have about linear speedup in second case study. If one takes MATLAB startup times out then we would have about linear speedup even in the first case study. The two approaches that have been discussed to port MATLAB[®] applications to HPRC are, the library based approach, where the MATLAB[®] program is

Table 5.1 Execution times of various approaches

<u>Case Study</u>	<u>Mode</u>	<u>Execution Time in secs</u>
I	Serial	86.2319
	Parallel – Approach I	37.9642
	Parallel – Approach II	41.7133
II	Serial	2245.6458
	Parallel-Approach I	438.359
	Parallel-Approach II	443.636

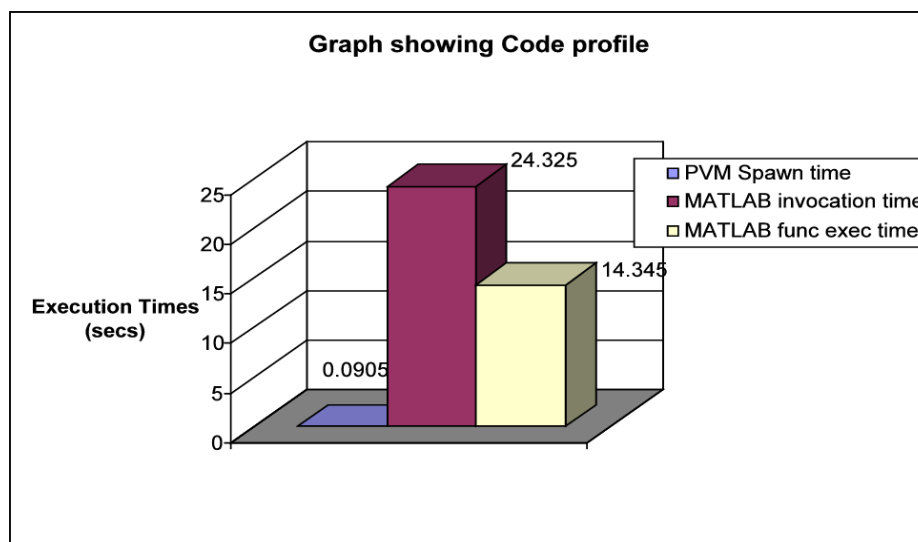


Figure 5.1 Graph showing the code profile for case study I

responsible for process spawning and management and the other being the approach where a C process is responsible for task spawning and management. The questions that arise now are of feasibility of each approach and the scenario in which each is best suited. These need to be compared and contrasted on the basis of metrics like end user friendliness, performance advantage, run time efficiency and ease of programming. This chapter delves into the vast problem space on hand and tries to address the various permutations and combinations possible with these approaches.

5.1 Feasibility and Target Scenarios for both Approaches

Each of the approaches fit in a set of applications and scenarios. Approach (i) of MATLAB[®] program being a master process is very suitable for an end user who is not interested in the details of the underlying hardware architecture and is just interested in higher computational speed. He /she would need to just make a function call just as the other functions in MATLAB[®], and not be worried about the optimized implementation. The function call would dynamically link with the respective function in a library of optimized ‘Mex files’ and execute on some remote node or FPGA hardware unknown to the end user and return the results to the calling program. In fact, routines can be built using this approach that would take a user defined MATLAB[®] function and execute it on a remote node returning the result back to the user. Also, this approach can be used to dynamically link existing libraries in C/C++ or Fortran with MATLAB[®] and the functions be called directly from MATLAB[®] as if they were MATLAB[®] functions. Thus, a layer of

abstraction has been built for a user not interested in the ‘black box’ below. Figures 5.2 and 5.3 illustrate this idea. For advanced users a library of message passing functions like PVM or MPI can be dynamically linked with MATLAB® using ‘Mex files’ and the power of distributed and reconfigurable computing can be made available directly to end users who would just require to make simple function calls in MATLAB®.

The second approach is more suitable for an advanced user. It fits well in scenario of multi-tier architecture. An application, being developed in some other language like C/C++ or Java can invoke MATLAB® engine to perform computations and return the results back to the original application. Figure 5.4 shows a two tier, client server architecture with GUI on the client developed in C/C++ interfaced with OpenGL or some other graphics library and computations being performed as call backs to MATLAB® routines on a remote server. In a multi-tier approach MATLAB® can be used in the middle layer as a computational engine for the logic, with the outer GUI layer on the client, and database on the remote server. This is illustrated in figure 5.5

Both the approaches suit different target scenarios, but still can be combined with various combinations. The MATLAB® computation, being performed as a call back in the second approach, can use functions from the optimized library as in the first approach for computations.

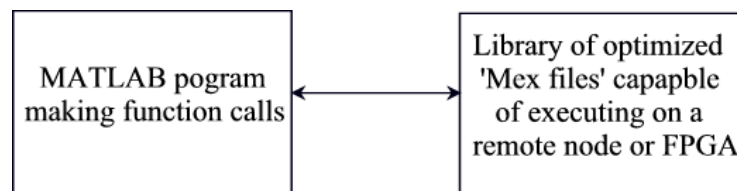


Figure 5.2 MATLAB interfacing with a library of optimized routines build with 'Mex Files'

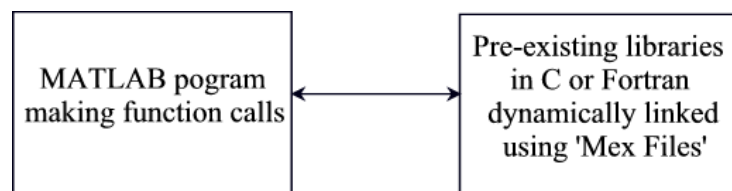


Figure 5.3 MATLAB interfacing with pre-existing libraries in C or Fortran using 'Mex Files'

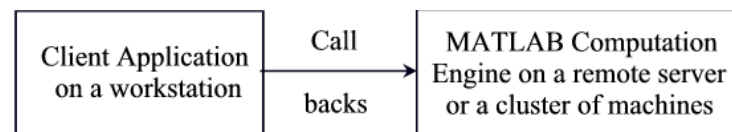


Figure 5.4 Client-Server topology with computations being performed in MATLAB[®]

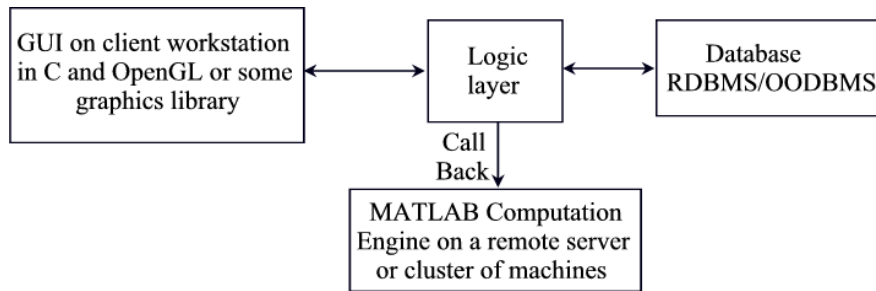


Figure 5.5 Multi - tier architecture with computations being performed in MATLAB[®]

5.2 Performance Advantage and Run Time Efficiency

The speedups obtained by using approach I are slightly higher than that with approach II as evident from table 5.1. But these are not considerably higher the reason being that even while using approach I the child tasks spawned by the Mex files invoke MATLAB[®] engines for computations. If the computations were performed on hardware or in an optimized parallel C routine we would expect much higher speedups. The optimized ‘Mex Files’ have a faster execution time and hence higher run time efficiency. In MATLAB[®], by design itself, if a function exists both as a ‘*.m’ file and a ‘*.mexsol’ (or any other extension depending on the platform) file, the ‘*.mexsol’ file is given preference for execution automatically over the ‘*.m’ file.

5.3 End User Friendliness

As discussed in the earlier section for a user not interested in underlying hardware details, first approach of library-based computations is more suitable. The second approach with C as a master process is more suitable for larger designs and an advanced user.

5.4 Ease of Programming

There is no particular ease in programming of one approach over the other. This would be actually application complexity and scenario dependent. Both the approaches follow some simple steps that need to be followed for successful implementations and the approach chosen is very much application dependent and the choice of the programmer. Many times the chosen approach is a hybrid of both of these approaches. The first approach sets a level of abstraction higher hiding the details of the underlying architectures, thus can be said to be end user friendly. But, then that assumes a library of optimized routines already implemented and ready to use.

6 Future Work

The first and foremost here is to verify results with the RC component employed in execution. This would validate the entire effort. We have analyzed approaches for programming MATLAB[®] on High Performance Reconfigurable Computers. We have opened the Pandora's box. This opens up a huge problem space and there are a host of other problems and issues to deal with. An obvious next step would be building optimized libraries of functions that would dynamically link with 'Mex files' that would perform computations on remote nodes or reconfigurable FPGAs. Toolboxes of message passing libraries like PVM and MPI can be built using the approaches discussed. Functions to spawn and manage processes on remote machines can be built which would enable MATLAB[®] to be easily ported on distributed machines. Many preexisting libraries in C and Fortran can be coupled with 'Mex files' and be made available as direct function calls from MATLAB[®].

MATLAB[®] is growing at a very fast pace. Along with the computer science world, MATLAB[®] also has realized and adopted the advantages of Object Oriented programming. MATLAB[®] introduced objects from version 5.0 onwards. MATLAB[®] also has opened its doors to component object technologies.

MATLAB[®] COM builder can compile MATLAB[®] algorithms into COM objects that are accessible from any COM based application. This opens up a whole new problem space where approaches to use objects in distributed computing need to be researched.

There are many other issues that need to be dealt with, like scheduling, load balancing, optimum resource utilization, and modeling and performance analysis of High Performance Reconfigurable systems. Eventually, moving towards building a development system to efficiently utilize the processing power of such systems is the goal.

References

- [1] J. L. Gustafson, "Reevaluating Amdahl's Law," *Communications of ACM*, pp. 532-533, 1988.
- [2] G. M. Amdahl, "Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities," presented at AFIPS conference proceedings, Atlantic City, N.J., 1967.
- [3] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam, *PVM: Parallel Virtual Machine, A Users' Guide and Tutorial for Networked Parallel Computing*: MIT Press, 1994.
- [4] M. Snir, S. Otto, S. Huss-Lederman, D. Walker, and J. Dongarra, *MPI: The Complete Reference*, 2nd ed: MIT Press, 1998.
- [5] L. S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and W. R. C., *ScaLAPACK Users' Guide*: SIAM, Philadelphia, PA, 1997.
- [6] R. A. van de Geijn, *Using PLAPACK*: MIT Press, 1997.
- [7] S. Natarajan, B. Levine, C. Tan, D. Newport, and D. Bouldin, "Automatic Mapping of Khoros-based Applications to Adaptive Computing Systems," *Proceedings of 1999 Military and Aerospace Applications of Programmable Devices and Technologies International Conference (MAPLD)*, pp. 101-107, 1999.
- [8] Z. Ye, P. Banerjee, S. Hauck, and A. Moshovos, "CHIMAERA: A High-Performance Architecture with a Tightly-Coupled Reconfigurable Unit," presented at International Symposium on Computer Architecture, Toronto, CANADA, 2000.
- [9] J. R. Hauser and J. Wawrzynek, "Garp: A MIPS Processor with a Reconfigurable Coprocessor," presented at IEEE Symposium on FPGAs for Custom Computing Machines, 1997.
- [10] S. C. Goldstein, H. Schmit, M. Moe, M. Budiu, S. Cadambi, R. R. Taylor, and R. Laufer, "PipeRench: a Coprocessor for Streaming Multimedia Acceleration," presented at ISCA, 1999.
- [11] S. C. Goldstein, H. Schmit, M. Budiu, S. Cadambi, M. Moe, and T. R.R., "PipeRench: A Reconfigurable Architecture and Compiler," *IEEE Computer*, vol. 33 No. 4, 2000.
- [12] H. Schmit, D. Whelihan, A. Tsai, M. Moe, B. Levine, and R. R. Taylor, "PipeRench: A Virtualized Programmable Datapath in 0.18 Micron Technology," presented at IEEE Custom Integrated Circuits Conference (CICC), 2002.
- [13] P. H. W. Leong, M. P. Leong, O. Y. H. Cheung, T. Tung, C. M. Kwok, M. Y. Wong, and K. H. Lee, "Pilchard - A Reconfigurable Computing Platform with Memory Slot Interface," *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*, 2001.
- [14] G. D. Peterson and M. C. Smith, "Programming High Performance Reconfigurable Computers," *SSGRR 2001*, 2001.
- [15] M. C. Smith and G. D. Peterson, "Programming High Performance Reconfigurable Computers (HPRC)," *SPIE International Symposium ITCOM*, 2001.

- [16] M. C. Smith and G. D. Peterson, "Analytical Modeling for High Performance Reconfigurable Computers," *Proceedings of the SCS International Symposium on Performance Evaluation of Computer and Telecommunications Systems*, 2002.
- [17] Matlab.Documentation, "MATLAB-The Language of Technical Computing, Using Matlab version 6.0," August 2002 ed: COPYRIGHT 1984 - 2002 by The MathWorks, Inc., 2002.
- [18] C. Moler, "Why there isn't a parallel MATLAB," *Matlab News and Notes*, 1995.
- [19] R. Choy, "Parallel Matlab Survey (<http://supertech.lcs.mit.edu/~cly/survey.html>)," 2003.
- [20] C. C. Chang, G. Czajkowski, X. Liu, V. Menon, C. Myers, A. Trefethen, and L. N. Trefethen, "The Cornell MultiMATLAB Project," presented at Parallel Object-Oriented Methods and Applications Conference, Santa Fe, New Mexico, 1996.
- [21] B. Javier, "MPI ToolBox for MATLAB (MPITB) http://atc.ugr.es/javier-bin/mpitb_eng," vol. 2003, June, 2003.
- [22] B. Javier, "PVM ToolBox for MATLAB http://atc.ugr.es/javier-bin/pvmtb_eng," vol. 2003, June, 2003.
- [23] S. Pawletta, "The DP-Toolbox Home Page => http://www-at.e-technik.uni-rostock.de/rg_ac/dp/," 2001.
- [24] J. Kepner, "Parallel Programming with MatlabMPI," presented at High Performance Embedded Computing Workshop, 2001.
- [25] L. DeRose, K. Gallivan, E. Gallopoulos, B. Marsolf, and D. Padua, "FALCON: A MATLAB Interactive Restructuring Compiler," presented at Languages and Compilers for Parallel Computing, Springer-Verlag, 1995.
- [26] D. Padua, R. Eigenmann, J. Hoeflinger, P. Petersen, P. Tu, S. Weatherford, and K. Faigin, "Polaris: A New-Generation Parallelizing Compiler for MPP's," Univ. of Illinois at Urbana-Champaign, Center for Supercomputing Res. & Dev, Technical Report 1306, June 1993.
- [27] J. Eriksson, P. Jacobson, and E. Lindström, "The CONLAB Environment: A Tool for Developing and Testing of Parallel Algorithms in NUMerical Linear Algebra," Institute of Information Processing, University of Umeå, S-901 87 Umeå, Technical Report UMNAD-47.88, 1988.
- [28] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and S. D., *LAPACK Users' Guide*, 3rd ed: SIAM, Philadelphia, PA, 1999.
- [29] M. Quinn, A. Malishevsky, and N. Seelam, "Otter: Bridging the gap between MATLAB and ScaLAPACK," presented at 7th IEEE International Symposium on High Performance Distributed Computing, 1998.
- [30] "RTExpress: Integrated Sensors Inc. <http://www.rtexpress.com/isi/>,"
- [31] P. Banerjee, N. Shenoy, A. Choudhary, S. Hauck, C. Bachmann, M. Haldar, P. Joisha, A. Kanhare, A. Nayak, S. Periyacheri, M. Walkden, and

- D. Zaretsky, "A MATLAB Compiler For Distributed, Heterogeneous, Reconfigurable Computing Systems," *IEEE Symposium on FPGAs for Custom Computing Machines*, pp. pp. 39-48, 2000.
- [32] S.-W. Ong, N. Kerkiz, B. Srijanto, C. Tan, M. A. Langston, D. Newport, and D. Bouldin, "Automatic Mapping of Multiple Applications to Multiple Adaptive Computing Systems," presented at Proceedings of 2001 IEEE Symposium on Field-programmable Custom Computing Machines (FCCM), Rohnert, CA, 2001.
- [33] H. Casanova and J. Dongarra, "NetSolve: A Network Server for Solving Computational Science Problems," *The International Journal of Supercomputer Applications and High Performance Computing*, vol. 11, pp. 212-223, 1997.
- [34] R. Choy and A. Edelman, "MATLAB*P 2.0: Interactive Supercomputing Made Practical," in *EECS: MIT*, 2002.
- [35] "URL for AFRL/IF Distributed Center - <http://www.if.afrl.af.mil/tech/facilities/HPC/hpcf.html>."
- [36] Matlab.Documentation, "MATLAB-The Language of Technical Computing, External Interfaces. ver. 6.0," July 2002 ed: The Mathworks, Inc. COPYRIGHT 1984 - 2002 by The MathWorks, Inc., 2002.
- [37] "PVM Home Page http://www.csm.ornl.gov/pvm/pvm_home.html."
- [38] K. H. Tsoi, "Pilchard User Reference (v0.1)," Department of Computer Science and Engineering, The Chinese University of Hong Kong, Shatin, NT Hong Kong January 22 2002.
- [39] S. W. Smith, *The Scientist and Engineer's Guide to Digital Signal Processing*: California Technical Publishing, 1997.
- [40] I. Chakrabarti, T. Corde, B. Flatt, A. Gill, and C. Pepys, "Pattern Matching (link - <http://www.owl.net.rice.edu/~elec431/projects96/pictomaniacs/DSP.html>), " Electrical Engineering Department, Rice University 1996.
- [41] M. Misra, "Parallel Environments for implementing Neural Networks," *Neural Computing Surveys*, vol. 1, pp. 48-60, 1997.
- [42] S. Haykin, *Neural Networks: A Comprehensive Foundation*, 2nd ed: Prentice Hall, 1998.
- [43] G. Forde, L. Townsend, and J. W. Hines, "Application of Artificial Neural Networks in Predicting Astronaut Doses From Large Solar Particle Events in Space," *Proceedings of the 1998 ANS Radiation Protection and Shielding Division Topical Conference*, 1998.
- [44] J. Hoff, L. Townsend, and J. W. Hines, "Prediction of Energetic Solar Particle Event Dose-time Profiles Using Artificial Neural Networks," presented at 4th Nuclear and Space Radiation and Effects Conference NSFEC, Monterey, California, 2003.
- [45] L. W. Townsend, J. S. Neal, and J. W. Hines, "Solar Particle Event Doses and Dose Rates for Interplanetary Crews: Predictions Using Artificial Intelligence and Bayesian Inference," presented at COSPAR, Warsaw, Poland, 2000.

- [46] N. H. Tehrani, L. W. Townsend, J. W. Hines, and G. M. Forde, "Predicting Astronaut Radiation Doses from Large Solar Particle Events Using Artificial Intelligence," presented at International Conference on Environmental Systems, Denver, Colorado, 1999.
- [47] H. Demuth and M. Beale, "Neural Network Toolbox, User's Guide (MATLAB Documentation)," July 2002 ed: COPYRIGHT 1984 - 2002 by The MathWorks, Inc., 2002.

Appendices

Appendix A – Some figures of Chapter 3

Figures A.2 to A.6 show the outputs for character recognition application of chapter 3. Figure A.1 shows the source image.

Figures A.7 to A.9 show the block diagram for the FPGA implementation. Figure A.10 shows the layout on Virtex 1000e part

growing need for higher computational power and tighter budgets has triggered a lot of research in the field of high performance computing and cluster computing. a lot of effort has gone into devising programming methods and tools for efficient use of high performance parallel computers. but parallel programming still remains a challenging task due to many reasons like architectural complexity, higher costs, notion of sequential programming languages, availability of several custom hardware(s)/ software(s) and lack of expertise. all these and many other reasons have led to lesser commercial success and sustainability for hpc platforms. cheaper alternatives like "beowolf" clusters of various custom hardware commodities can be implemented but programming and optimal use of the potential of these platforms still is a considerable obstacle and a time consuming practice.

Figure A.1 Source Image

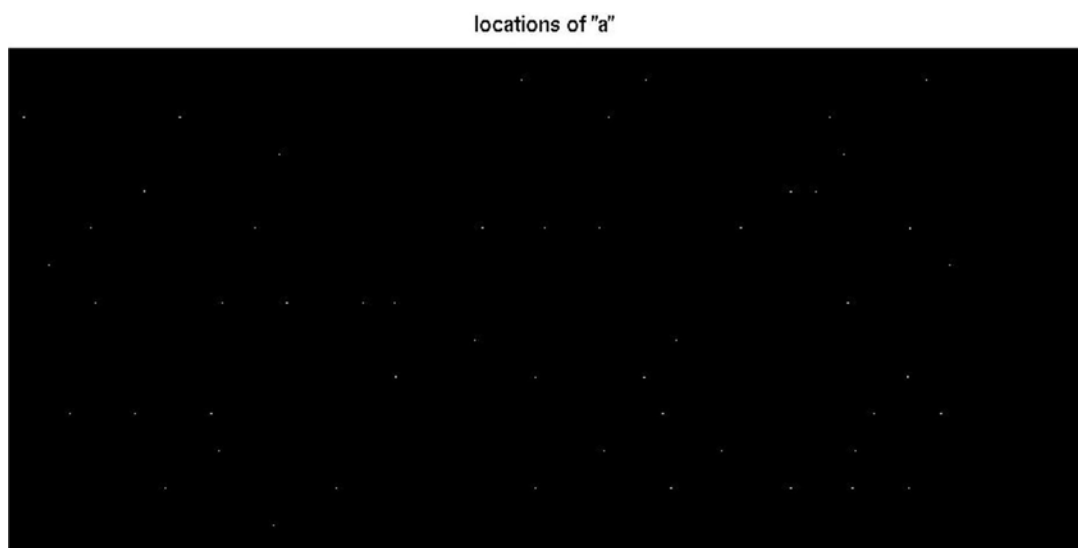


Figure A.2 Locations of character 'a'



Figure A.3 Locations of character 'e'

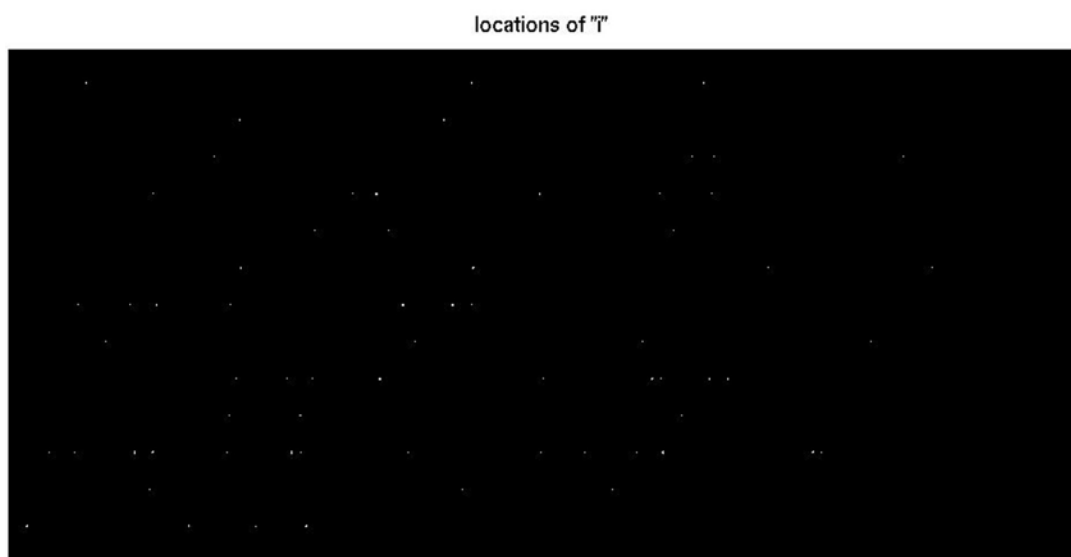


Figure A.4 Locations of character 'i'

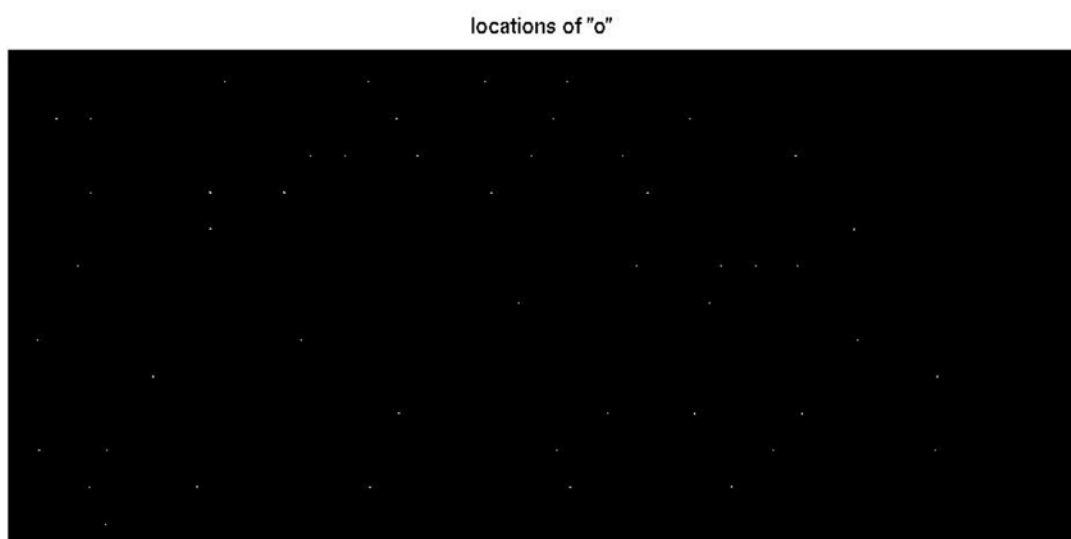


Figure A.5 Locations of character 'o'

locations of "u"



Figure A.6 Locations of character 'u'

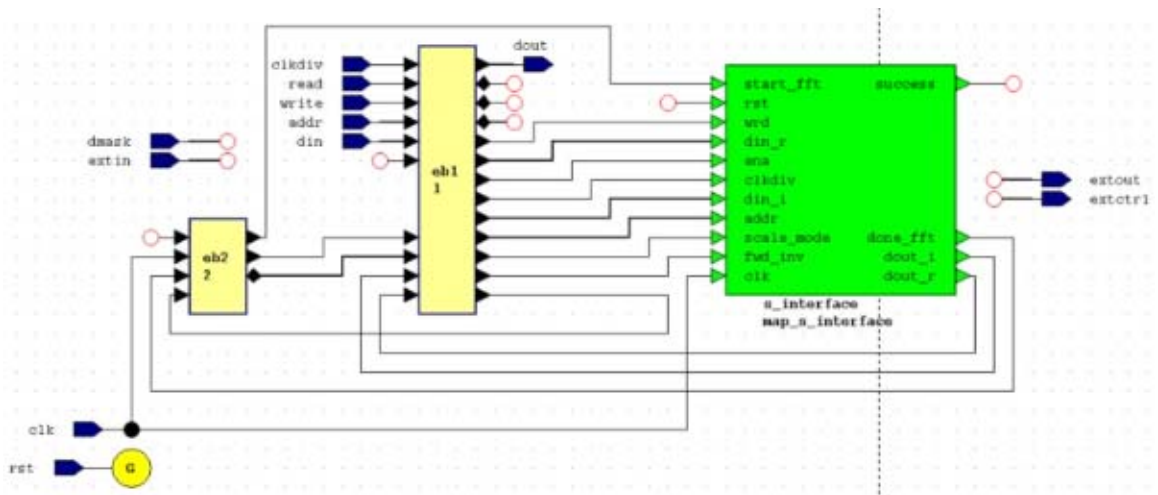


Figure A.7 Block diagram for pcire.vhd

Figure A.9 Block diagram for sms.vhd

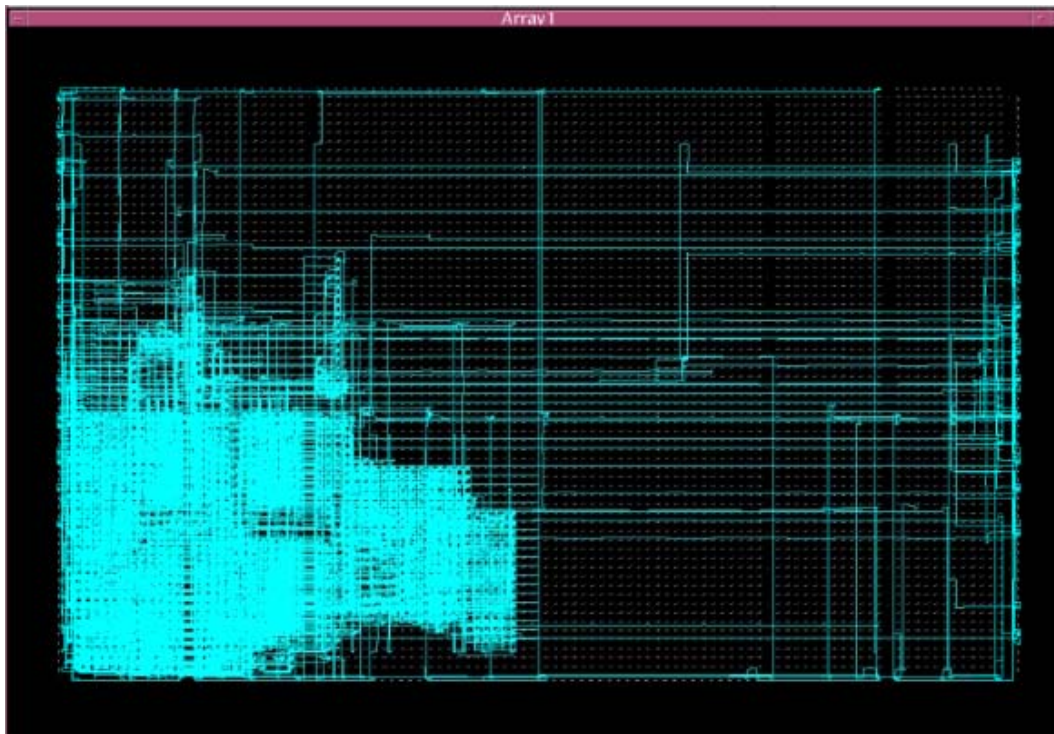


Figure A.10 Layout on Virtex 1000e part

Appendix B – Steps to port MATLAB functions to HPRC

1. First select the approach depending on the application. Reasons to consider –
 - a. The application being coded in MATLAB[®] or C. In case of MATLAB[®] approach I could be more suitable. In case of coding in C, approach II might be more suitable
 - b. What are you coding? In case of a parallel library for use in MATLAB[®] applications, approach I is suitable.

In case of Approach II jump to step 6. In case of approach I jump to the step 2.
2. Select the target for execution. Targets can be FPGA(s), or just a remote node or a cluster of nodes or may be all of the above depending on the complexity the user wishes to address. In case of remote nodes the function can be executed by invoking MATLAB[®] computational engine or in C or invoking functions in other libraries.
3. Implement the function in the target language e.g. C, VHDL, MATLAB[®].
4. Link it as a Mex – file. Refer to chapter 2 and also if needed, MATLAB[®] External Interfaces documentation for the details on Mex files.

5. Compile the Mex file and it is ready for execution by directly calling it as a MATLAB[®] function
6. In case of approach II, the target language is obviously MATLAB[®]. Write C code to invoke MATLAB[®] computational engine. Refer to chapter 2 and also if needed, MATLAB[®] External Interfaces documentation for more details.
7. Set up the variables in MATLAB[®] workspace.
8. Execute the MATLAB[®] function and read back the results in C.

Flowchart in Figure A 11 shows the various steps involved.

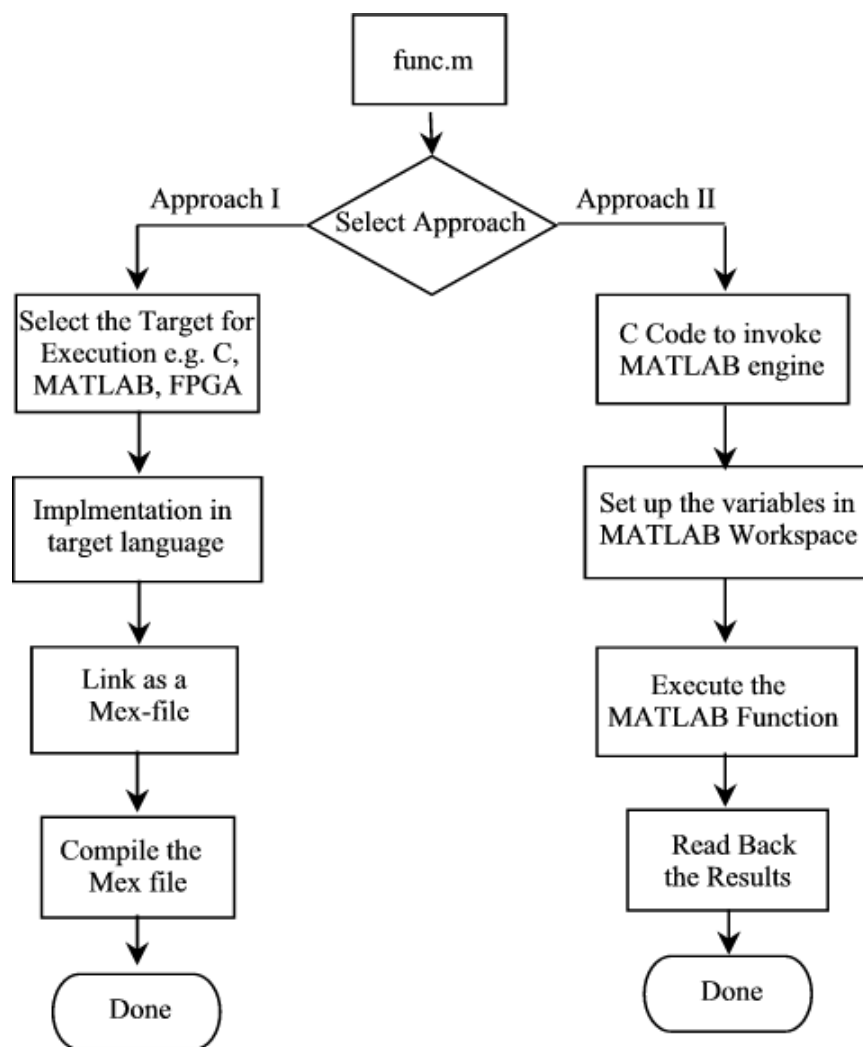


Figure A 11 Flowchart indicating the steps to port MATLAB to HPRC

Appendix C – Program Codes

Character Recognition

Approach I

mycorr.c

```

/*****
/**  Mex file for performing Character Recognition using Approach
I  **/
/**  -----  **/
/**  Program spawns the NTASK-1 child processes child.c **/
/**  Each child process calculates the image correlation of 2
images  /**  and returns the output to master who displays it
**/
/**  -----  **/
/**  Author : Saumil Merchant  **/
/**          University of Tennessee  **/
/**          Electrical & Computer Engineering Department  **/
*****/

#include <stdio.h>
#include "/sw/matlab6.1/extern/include/mex.h"
#include "/sw/matlab6.1/extern/include/matrix.h"
#include "/usr/local/pvm3/include/pvm3.h"
#include <sys/time.h>
#include <sys/resource.h>

/*#define test*/

void mycorr(double *x,int cx, int rx, double *y,int cy,int
ry,char *name,char *node);

void mexFunction( int nlhs, mxArray *plhs[],
                  int nrhs, const mxArray *prhs[] ){

    int cx,rx,cy,ry,op_len,info,nd_len;
    double *x,*y,t;
    char *op_name,*node;
    struct timeval tmout;

```

```

info = gettimeofday(&tmout,NULL);
t=tmout.tv_sec + (tmout.tv_usec)/1000000.0;
mexPrintf("Time: %lf\n",t);
if(nrhs != 4)
    mexErrMsgTxt("please give two real matrices and
output datafilename as inputs");

if(nlhs > 0)
    mexErrMsgTxt("output saved in .mat file, not
returned");

if (mxIsComplex(prhs[0]) || mxIsComplex(prhs[1]))
    mexErrMsgTxt("Input cannot be complex");

if ( !mxIsChar(prhs[2]) || !mxIsChar(prhs[3]))
    mexErrMsgTxt("3rd and 4th input arguments should be
strings");

if ( mxGetM(prhs[2]) != 1 || mxGetM(prhs[3]) != 1)
    mexErrMsgTxt("3rd and 4th input arguments should be a
row vectors");

/* get  the length of the input vector */

cx = mxGetN(prhs[0]);
rx = mxGetM(prhs[0]);
cy = mxGetN(prhs[1]);
ry = mxGetM(prhs[1]);
op_len = ( mxGetN(prhs[2]) * mxGetM(prhs[2]) ) + 1;
nd_len = ( mxGetN(prhs[3]) * mxGetM(prhs[3]) ) + 1;

/* pointer to real data */
x = mxGetPr(prhs[0]);
y = mxGetPr(prhs[1]);

/* Allocate memory for string name */
op_name = mxCalloc(op_len, sizeof(char));
node = mxCalloc(nd_len, sizeof(char));

/* pointer to op_name string*/

info = mxGetString(prhs[2],op_name,op_len);
if (info != 0)
    mexErrMsgTxt("output data filename not read");

info = mxGetString(prhs[3],node,nd_len);
if (info != 0)
    mexErrMsgTxt("node name not read");

/* Call your C subroutine */
mycorr(x,cx,rx,y,cy,ry,op_name,node);

```

```

        return;
    }

void mycorr(double *x,int cx, int rx, double *y,int cy,int
ry,char *name,char *node) {

    int nrows,ncols;
    char teststr[100];
    int info,cc,tid;

    /***** Spawning the child processes *****/

    cc =
    pvm_spawn("/home/smerchan/thesis/application_1/mtoc/slave",
(char**)0, 0, node, 1, &tid);
    if (cc == 0) { pvm_exit(); mexErrMsgTxt("Tasks cannot be
spawned"); }

    #ifdef test
        puts("Test Mode");

        /***** ping ponging test messages
        *****/

        /***** sending test messages *****/

        pvm_initSend(PvmDataDefault);
        pvm_pkstr("Hello from ");
        info = pvm_send(tid,10);

        /***** receiving and printing test
        mssages *****/

        info = pvm_recv(tid,11);
        if (info>0){
            pvm_upkstr(teststr);
            mexPrintf("%s\n",teststr);
        }
    #endif

    #ifdef test
        mexPrintf("\ncheck 1\n");
    #endif

    /***** sending data to the slave *****/

    pvm_initSend(PvmDataDefault);
    pvm_pkint(&rx,1,1);
    pvm_pkint(&cx,1,1);
    pvm_pkdouble(x,rx*cx,1);
    pvm_pkint(&ry,1,1);
    pvm_pkint(&cy,1,1);

```

```

pvm_pkdouble(y, ry*cy, 1);
pvm_pkstr(name);

    pvm_send(tid, 1);

#ifdef test
    mexPrintf("\ncheck 2\n");
#endif

    #ifdef test

/***** data received echo *****/

        pvm_rcv(tid, 12);
        pvm_upkstr(teststr);
        mexPrintf("%s\n", teststr);

/***** Receiving echo confirmation for Matlab
status*****/

        pvm_rcv(tid, 13);
        pvm_upkstr(teststr);
        mexPrintf("%s\n", teststr);

/***** Receiving echo confirmation of done*****/

        pvm_rcv(tid, 14);
        pvm_upkstr(teststr);
        mexPrintf("%s\n", teststr);

    #endif

    pvm_exit();
}

```

Child.c

```

/*****
/** Child program invoked by Mex file mycorr.c for performing
/** Character Recognition
/** ----- **/
/** Each child process calculates the image correlation of 2
images /** and returns the output to master who displays it
**/
/** ----- **/
/** Author : Saumil Merchant **/
/** University of Tennessee **/
/** Electrical & Computer Engineering Department **/

```

```

/*****

#include <stdio.h>
#include "/sw/matlab6.1/extern/include/engine.h"
#include "/usr/local/pvm3/include/pvm3.h"
#include "/sw/matlab6.1/extern/include/matrix.h"
#include <sys/time.h>
#include <sys/resource.h>

#define BUFSIZE 25000
/*#define test*/

double cpusecs() {
    struct rusage ru;
    getrusage(RUSAGE_SELF, &ru);
    return(ru.ru_utime.tv_sec +
    ((double)ru.ru_utime.tv_usec)/1000000.0);
}/* cpusecs end*/

int main(){

    int ptid, r1, c1, d, test1;
    int *r, *c;
    char filename[25], buffer[BUFSIZE], testbuf[100],
err[]="ERROR:";
    mxArray *datafile =NULL, *cor2=NULL;
    Engine *ep;
    double *cor2data, t1, t2, dt;

    /***** start the timer *****/
    t1 = (double)cpusecs();

    ptid = pvm_parent();

    #ifdef test

        /***** receiving test message *****/

        pvm_recv(ptid, 10);
        pvm_upkstr(testbuf);

        /***** sending test message *****/

        pvm_initSend(PvmDataDefault);
        gethostname(testbuf+ strlen(testbuf), 64);
        pvm_pkstr(testbuf);
        pvm_send(ptid, 11);
    #endif

    /***** receiving the datafilename *****/

    pvm_recv(ptid, 1);
    pvm_upkstr(filename);

```

```

#ifdef test

    /***** echoing back *****/

    pvm_initsend(PvmDataDefault);
    pvm_pkstr(filename);
    pvm_send(ptid,12);
#endif

    /***** starting matlab engine *****/

    ep=engOpen("\0");
    if (!ep) {
        fprintf(stderr, "\nCan't start MATLAB engine\n");
#ifdef test
        /***** echoing the error *****/
        pvm_initsend(PvmDataDefault);
        pvm_pkstr("Matlab not started");
        pvm_send(ptid,13);
#endif
        pvm_exit();
        return EXIT_FAILURE;
    } /* end if */

#ifdef test
    /***** echoing back *****/

    pvm_initsend(PvmDataDefault);
    pvm_pkstr("Matlab started");
    pvm_send(ptid,13);
#endif

    d=engEvalString(ep,"addpath('/home/smerchan/thesis/applicat
ion_1');");

    datafile=mxCreateString(filename);
    mxSetName(datafile,"datafile");
    engPutArray(ep,datafile);          /*** putting it
in Matlab workspace ***/
    engOutputBuffer(ep,buffer,BUFSIZE);
    d=engEvalString(ep,"eval(['load ' datafile]);");
    d=engEvalString(ep,"x=imacor(x1,x2);");
    cor2=engGetArray(ep,"x");
    cor2data = mxGetPr(cor2);
    r1 = mxGetM(cor2);
    c1 = mxGetN(cor2);

#ifdef test
    /***** echoing back *****/
    pvm_initsend(PvmDataDefault);
    pvm_pkstr(buffer);
    pvm_send(ptid,14);

```

```

#endif

/***** sending the output to the parent
*****/

    pvm_initsend(PvmDataDefault);
    pvm_pkint(&r1,1,1);
    pvm_pkint(&c1,1,1);
    pvm_pkdouble(cor2data,r1*c1,1);
    pvm_send(ptid,2);

/**** freeing the allocated memory and ending the matlab
session *****/

    mxDestroyArray(datafile);
    mxDestroyArray(cor2);
    engClose(ep);

/***** stop the timer *****/
    t2 = (double)cpusecs();
    dt=t2-t1;
    pvm_initsend(PvmDataDefault);
    pvm_pkdouble(&dt,1,1);
    pvm_send(ptid,3);

    pvm_exit();

} /***** main end *****/

```

Approach II

Parent.c

```

/*****
/** Parallel Character Recognition Approach II **/
/** ----- **/
/** Program for character recognition using Image Correlation**/
/** Uses custom Matlab functions written by the author **/
/** The following functions are used **/
/** imacor.m --> to do correlation **/
/** Program spawns the NTASK-1 child processes child.c **/
/** Each child process calculates the image correlation of 2
images **/
/** and returns the output to master who displays it **/
/** ----- **/
/** Author : Saumil Merchant **/
/** University of Tennessee **/
/** Electrical & Computer Engineering Department **/
*****/

#include <stdio.h>

```

```

#include "/sw/matlab6.1/extern/include/engine.h"
#include "/usr/local/pvm3/include/pvm3.h"
#include "/sw/matlab6.1/extern/include/matrix.h"
#include <sys/time.h>
#include <sys/resource.h>

#define BUFSIZE 25000
/*#define test*/

#define NTASK 6    /** starts NTASK-1 child processes **/

/***** to calculate the execution time *****/
double cpusecs() {
    struct rusage ru;
    getrusage(RUSAGE_SELF,&ru);
    return(ru.ru_utime.tv_sec +
    ((double)ru.ru_utime.tv_usec)/1000000.0);
}/* cpusecs end*/

int main(){

    Engine *ep;
    char buffer[BUFSIZE];
    int d, info, cc, i, tid[NTASK-1];
    char filename[100],teststr[100];
    mxArray *datafile =NULL,*cor2=NULL;
    int bufid,r1,c1;
    double *pcor2,t1,t2,ct;
    struct timeval tmout;

    /***** Start the CPU timer *****/

    info = gettimeofday(&tmout,NULL);
    t1=tmout.tv_sec + (tmout.tv_usec)/1000000.0;

    /***** Spawning the child processes *****/

    cc =
pvm_spawn("/home/smerchan/thesis/application_1/child", (char**)0,
0, "", NTASK-1, tid);

    #ifdef test
        puts("Test Mode");

        /***** ping ponging test messages *****/
        /***** sending test messages *****/

        pvm_initsend(PvmDataDefault);
        pvm_pkstr("Hello from ");
        info = pvm_mcast(&tid[0],NTASK-1,10);

```



```

                                /***** receiving and printing test
messages *****/

        for (i=0; i<NTASK-1; i++){
            info = pvm_recv(-1,11);
            if (info>0){
                pvm_upkstr(teststr);
                puts(teststr);
            } /* if end */
        } /* for end */
    #endif

    /***** sending the imagefilename to the child
*****/

    pvm_initSend(PvmDataDefault);
    pvm_pkstr("im_a");
    pvm_send(tid[0],1);

    pvm_initSend(PvmDataDefault);
    pvm_pkstr("im_e");
    pvm_send(tid[1],1);

    pvm_initSend(PvmDataDefault);
    pvm_pkstr("im_i");
    pvm_send(tid[2],1);

    pvm_initSend(PvmDataDefault);
    pvm_pkstr("im_o");
    pvm_send(tid[3],1);

    pvm_initSend(PvmDataDefault);
    pvm_pkstr("im_u");
    pvm_send(tid[4],1);

    #ifdef test

        /***** Receiving echo confirmation for image
filename*****/
        for (i=0; i<NTASK-1; i++){
            pvm_recv(-1,12);
            pvm_upkstr(teststr);
            puts(teststr);
        } /* for end */
    #endif

    #ifdef test

        /***** Receiving echo confirmation for
Matlab status*****/

        for (i=0; i<NTASK-1; i++) {

```

```

        pvm_recv(-1,13);
        pvm_upkstr(teststr);
        puts(teststr);
    } /* for end */
#endif

    /***** starting matlab engine and computing image
    correlation *****/

    ep=engOpen("\0");
    if (!(ep)) {
        fprintf(stderr, "\nCan't start MATLAB engine\n");
        return EXIT_FAILURE;
    } /* end if */

    #ifdef test

        /***** Receiving echo confirmation
        *****/

        for (i=0; i<NTASK-1; i++){
            pvm_recv(-1,14);
            pvm_upkstr(teststr);
            puts(teststr);
        } /* for end */
    #endif

    /***** receiving the data from child and
    displaying it *****/

    for (i=0; i<NTASK-1; i++){
        pvm_recv(-1,2);
        pvm_upkint(&r1,1,1);
        pvm_upkint(&c1,1,1);
        pcor2=(double*)malloc(r1*c1*8); /**** allocating
memory for data *****/
        pvm_upkdoube(pcor2,r1*c1,1);
        cor2 = mxCreateDoubleMatrix(r1,c1,mxREAL);
        mxSetName(cor2,"cor2");
        memcpy((void *)mxGetPr(cor2), (void *)pcor2,
r1*c1*8);
        d=engPutArray(ep,cor2);
        d=engEvalString(ep,"imaprint(cor2);");
    } /* for end */

    info = gettimeofday(&tmout,NULL);
    t2=tmout.tv_sec + (tmout.tv_usec)/1000000.0;
    printf("\nExecution Time of master process = %lf\n",t2-t1);

    printf("type OK to continue\n\n");
    scanf("%s",&teststr);
    puts("Done");

```

```

    /** free up memory and exit **/

    mxDestroyArray(datafile);
    engClose(ep);
    pvm_exit();

}/*** main end ***/

```

Child.c

```

/*****
** Parallel Character Recognition Approach II **/
** ----- **/
** Program for character recognition using Image Correlation**/
** Uses custom Matlab functions written by the author **/
** The following functions are used **/
** imacor.m --> to do correlation **/
** Program spawns the NTASK-1 child processes child.c **/
** Each child process calculates the image correlation of 2
images **/
** and returns the output to master who displays it **/
** ----- **/
** Author : Saumil Merchant **/
** University of Tennessee **/
** Electrical & Computer Engineering Department **/
*****/

#include <stdio.h>
#include "/sw/matlab6.1/extern/include/engine.h"
#include "/usr/local/pvm3/include/pvm3.h"
#include "/sw/matlab6.1/extern/include/matrix.h"
#include <sys/time.h>
#include <sys/resource.h>

#define BUFSIZE 25000
/*#define test*/

double cpusecs() {
    struct rusage ru;
    getrusage(RUSAGE_SELF, &ru);
    return(ru.ru_utime.tv_sec +
    ((double)ru.ru_utime.tv_usec)/1000000.0);
}/* cpusecs end*/

int main(){

    int ptid, r1, c1, d, test1;
    int *r, *c;

```

```

    char filename[25], buffer[BUFSIZE], testbuf[100],
err[]="ERROR:";
    mxArray *datafile =NULL, *cor2=NULL;
    Engine *ep;
    double *cor2data,t1,t2,dt;

    ptid = pvm_parent();

    #ifdef test

        /***** receiving test message *****/

        pvm_rcv(ptid,10);
        pvm_upkstr(testbuf);

        /***** sending test message *****/

        pvm_initSend(PvmDataDefault);
        gethostname(testbuf+ strlen(testbuf),64);
        pvm_pkstr(testbuf);
        pvm_send(ptid,11);
    #endif

    /***** receiving the iamgefilename *****/
    pvm_rcv(ptid,1);
    pvm_upkstr(filename);

    #ifdef test

        /***** echoing back *****/

        pvm_initSend(PvmDataDefault);
        pvm_pkstr(filename);
        pvm_send(ptid,12);
    #endif

    /***** starting matlab engine *****/

    ep=engOpen("\0");
    if (!(ep)) {
        fprintf(stderr, "\nCan't start MATLAB engine\n");
        #ifdef test
            /***** echoing the error *****/
            pvm_initSend(PvmDataDefault);
            pvm_pkstr("Matlab not started");
            pvm_send(ptid,13);
        #endif
        pvm_exit();
        return EXIT_FAILURE;
    } /* end if */

```

```

#ifdef test
    /***** echoing back *****/

    pvm_initsend(PvmDataDefault);
    pvm_pkstr("Matlab started");
    pvm_send(ptid,13);
#endif

    engOutputBuffer(ep,buffer,BUFSIZE);
    d=engEvalString(ep,"addpath('/home/smerchan/thesis/applicat
ion_1');");

    datafile=mxCreateString(filename);
    mxSetName(datafile,"datafile");
    engPutArray(ep,datafile);          /**** putting it
in Matlab workspace ****/
    d=engEvalString(ep,"x1=double(imread('text','tif'))");
    d=engEvalString(ep,"x2=double(imread(datafile,'tif'))");
    d=engEvalString(ep,"x=imacor(x1,x2)");
    cor2=engGetArray(ep,"x");
    cor2data = mxGetPr(cor2);
    r1 = mxGetM(cor2);
    c1 = mxGetN(cor2);

#ifdef test
    /***** echoing back *****/
    pvm_initsend(PvmDataDefault);
    pvm_pkstr(buffer);
    pvm_send(ptid,14);
#endif

    /***** sending the output to the parent
    *****/

    pvm_initsend(PvmDataDefault);
    pvm_pkint(&r1,1,1);
    pvm_pkint(&c1,1,1);
    pvm_pkdouble(cor2data,r1*c1,1);
    pvm_send(ptid,2);

    /**** freeing the allocated memory and ending the matlab
    session *****/

    mxDestroyArray(datafile);
    mxDestroyArray(cor2);
    engClose(ep);

    pvm_exit();

} /***** main end *****/

```

MATLAB Function imacor.m

```

function x=imacor(x1,x2)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Saumil Merchant
% ECE Department
% University Of Tennessee
%
% Function to perform character recognition using correlation
% operation on 2 data sets x1 and x2. The function uses fft
% algorithm to implement correlation.
%
%       x=ifft(fft(x1).*fft(x2))
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if nargin < 2
    error('Too few arguments in the function call');
end

x2 = flipud(fliplr(x2));

szel=size(x1);
sze2=size(x2);

r= szel(1)+sze2(1)-1;
c= szel(2)+sze2(2)-1;

x1f=fft2(x1,r,c);
x2f=fft2(x2,r,c);

x_raw=ifft2(x1f.*x2f);

x1_sq=x1.^2;
flat_x2=ones(size(x2));
x1f_sq=fft2(x1_sq,r,c);
flat_x2f=fft2(flat_x2,r,c);

x_sqnorm=ifft2(x1f_sq.*flat_x2f);
x_norm1=x_sqnorm.^0.5;

x_norm1=x_norm1+0.1;

x_norm = real(x_raw./x_norm1);

x=x_norm;

```

VHDL Design files**pcore.vhd**

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

LIBRARY DWARE;
use DWARE.DWpackages.all;

entity pcore is
generic ( width : Natural := 16;
          depth : Natural := 1024);

port (
    clk: in std_logic;
    clkdiv: in std_logic;
    rst: in std_logic;
    read: in std_logic;
    write: in std_logic;
    addr: in std_logic_vector(13 downto 0);
    din: in std_logic_vector(63 downto 0);
    dout: out std_logic_vector(63 downto 0);
    dmask: in std_logic_vector(63 downto 0);
    extin: in std_logic_vector(25 downto 0);
    extout: out std_logic_vector(25 downto 0);
    extctrl: out std_logic_vector(25 downto 0));
end pcore;

architecture syn of pcore is

component s_interface
generic ( width : Natural := 16;
          depth : Natural := 1024);
port(
    din_r  : in  std_logic_vector(width-1 downto 0);
    din_i  : in  std_logic_vector(width-1 downto 0);
    wrd    : in  std_logic;
    ena    : in  std_logic;
    addr   : in  std_logic_vector(bit_width(depth)-1 downto 0);
    clk    : in  std_logic;
    clkdiv : in  std_logic;
    start_fft : in std_logic;
    fwd_inv  : in std_logic;
    scale_mode : in std_logic;
    rst     : in std_logic;

    dout_r : out std_logic_vector(width-1 downto 0);
    dout_i : out std_logic_vector(width-1 downto 0);
    done_fft : out std_logic;
    success : out std_logic;
    state_out : out std_logic_vector(3 downto 0));

```

```
end component;
```

```
signal din_r,din_i : std_logic_vector(width-1 downto 0);
signal dout_r,dout_i : std_logic_vector(width-1 downto 0);
signal addra : std_logic_vector(bit_width(depth)-1 downto 0);
signal add_buf : std_logic_vector(bit_width(depth)-1 downto 0);
signal wea,ena,clkd : std_logic;
signal start_fft,fwd_inv,scale_mode : std_logic;
signal done_fft,success: std_logic;
signal state : std_logic_vector(3 downto 0);
signal succ_buf : std_logic;
signal start_compute : std_logic;
signal start : std_logic;
signal start_debug : std_logic;
signal start_cntl : std_logic;
signal state_s_int : std_logic_vector(3 downto 0);
signal count : std_logic_vector(4 downto 0);
```

```
begin
```

```
map_s_interface : s_interface
port map (din_r => din_r,
          din_i => din_i,
          wrd => wea,
          ena => ena,
          addr => addra,
          clk => clk,
          clkdiv => clkd,
          start_fft => start_fft,
          fwd_inv => fwd_inv,
          scale_mode => scale_mode,
          rst => rst,
          dout_r => dout_r,
          dout_i => dout_i,
          done_fft => done_fft,
          success => success,
          state_out => state_s_int);
```

```
clkdiv <= clkdiv;
fwd_inv <= '1';
scale_mode <= '1';
```

```
din_r <= din(15 downto 0);    -- real part
din_i <= din(47 downto 32);  -- imag part
```

```
--dout(15 downto 0) <= dout_r;    -- real part
--dout(47 downto 32) <= dout_i;    -- imag part
--dout(31 downto 16) <= (others=>'0'); --when dout_r(15)='0' else
(others=>'1');
--dout(63 downto 48) <= (others=>'0'); --when dout_i(15)='0' else
(others=>'1');
```

```
dout(15 downto 0) <= dout_r; -- real part
dout(31 downto 16) <= dout_i; -- imag part
```



```

dout(47 downto 32) <= (others =>'1') when succ_buf='1' else
(others =>'0');
dout(51 downto 48) <= (others =>'1') when start_fft='1' else
(others=>'0');
dout(55 downto 52) <= (others =>'1') when start_compute='1' else
(others=>'0');
dout(59 downto 56) <= state;
dout(63 downto 60) <= state_s_int;

succ_buf <= '0' when rst='1' or start_compute='1' else success or
succ_buf;

--dout(31 downto 0) <= "11000100110100000001101011100000" when
success='1' else (others=>'1');
--dout(63 downto 32) <= (others=>'1') when start_compute='1' else
(others=>'0');

addra <= add_buf;

add_buf <= din(25 downto 16) when write='1' else
"0000000000" when rst='1' else add_buf;  -- address bus

ena<='0' when addr(7 downto 0)="11111111" else '1';

wea <= write;

start <= '1' when addr(7 downto 0)="11111111" and read='1' else
'0';
start_debug <= '0' when rst='1' or state="0000" else start or
start_debug;
start_compute <= start_debug and start_cntl;

process(clk,rst)

--variable count: Integer;

begin
if rst='1' then
    start_fft<='0';
    count<=(others=>'0');
    start_cntl<='1';
    state<="0001";

elsif clk'event and clk='1' then

    case state is

        when "0000" =>
            start_fft<='0';
            count<=(others=>'0');
            start_cntl<='1';
            state<="0001";

        when "0001" =>
            if start_compute = '1' then

```

```

        start_fft<='1';
        state<="0010";
    end if;

    when "0010" =>    count<=count+'1';
                    start_cntl<='0';
                    if count="11111" then
                        count<=(others=>'0');
                        start_fft<='0';
                        state<="0011";
                    end if;

    when "0011" =>    start_fft<='0';
                    if done_fft='1' then
                        state<="0100";
                    end if;

    when "0100" =>    if start_compute = '0' then
                        state<="0000";
                    end if;

    when others =>    state<="0000";

    end case;
end if;

end process;

end syn;

```

s_interface.vhd

```

LIBRARY ieee,dware,dw03;
USE ieee.std_logic_1164.ALL;
use IEEE.STD_LOGIC_ARITH.all;
use IEEE.std_logic_unsigned.all;
use DWARE.DWpackages.all;
use DW03.DW03_components.all;

LIBRARY DWARE;
use DWARE.DWpackages.all;

entity s_interface is
generic ( width : Natural := 16;
          depth : Natural := 1024);
port(
    din_r : in  std_logic_vector(width-1 downto 0);
    din_i : in  std_logic_vector(width-1 downto 0);
    wrd   : in  std_logic;
    ena   : in  std_logic;
    addr  : in  std_logic_vector(bit_width(depth)-1 downto 0);

```

```

    clk : in std_logic;
    clkdiv: in std_logic;
    start_fft : in std_logic;
    fwd_inv : in std_logic;
    scale_mode : in std_logic;
    rst : in std_logic;

    dout_r : out std_logic_vector(width-1 downto 0);
    dout_i : out std_logic_vector(width-1 downto 0);
    done_fft : out std_logic;
    success : out std_logic;
    state_out : out std_logic_vector(3 downto 0)); --state_out
added for debugging purposes

end s_interface;

architecture s_arch of s_interface is

component sms
port( clk : in std_logic;
      xn_r : in std_logic_vector(width-1 downto 0);
      xn_i : in std_logic_vector(width-1 downto 0);
      start : in std_logic;
      fwd_inv : in std_logic;
      ce : in std_logic;
      rs : in std_logic;
      mrd : in std_logic;
      mwr : in std_logic;
      scale_mode : in std_logic;

      dob_i,dob_r : out std_logic_vector(width-1 downto 0);
      edone : out std_logic;
      done : out std_logic;
      result : out std_logic;
      ovflo : out std_logic;
      busy : out std_logic;
      ext_addr : out std_logic_vector(bit_width(depth)-1 downto
0);
      ext_addrw : out std_logic_vector(bit_width(depth)-1 downto
0);
      io_n : out std_logic);

end component;

component blockram_1024x16
port (
    addra: IN std_logic_VECTOR(bit_width(depth)-1 downto 0);
    addrb: IN std_logic_VECTOR(bit_width(depth)-1 downto 0);
    clka: IN std_logic;
    clkb: IN std_logic;
    dina: IN std_logic_VECTOR(width-1 downto 0);
    dinb: IN std_logic_VECTOR(width-1 downto 0);
    douta: OUT std_logic_VECTOR(width-1 downto 0);
    doutb: OUT std_logic_VECTOR(width-1 downto 0);
    ena: IN std_logic;

```

```

        enb: IN std_logic;
        wea: IN std_logic;
        web: IN std_logic);
END component;

signal pull_up : std_logic;
signal pull_down : std_logic;
signal state : std_logic_vector(3 downto 0);

signal ext_addrw,ext_addr : std_logic_vector(bit_width(depth)-1
downto 0);
signal dob_r, dob_i : std_logic_vector(width-1 downto 0);
signal data_r,data_i : std_logic_vector(width-1 downto 0);
signal io_n, wrb : std_logic;
signal start, mrd, mwr : std_logic;
signal ce,rs,edone,done : std_logic;
signal result,ovflo,busy : std_logic;

signal self_addr : std_logic_vector(bit_width(depth)-1 downto 0);
--signal cntl_addr: std_logic;
--signal addr_flag: std_logic;
signal clkd : std_logic;

-- lfsr signals
signal data : std_logic_vector(31 downto 0);
signal lfsr_en : std_logic;
signal lfsr_rsn : std_logic;
signal lfsr_sign : std_logic_vector(31 downto 0);
--signal success : std_logic;
signal ref_signature : std_logic_vector(31 downto 0);

begin

dataram_real: blockram_1024x16
port map (  addra => addr,
            addrb => self_addr,
            clka => clk,
            clk_b => clkd,
            dina => din_r,
            dinb => dob_r,
            douta => dout_r,
            doutb => data_r,
            ena => ena,
            enb => pull_up,
            wea => wrd,
            web => wrb);

dataram_imag: blockram_1024x16
port map (  addra => addr,
            addrb => self_addr,
            clka => clk,
            clk_b => clkd,
            dina => din_i,
```

```

        dinb => dob_i,
        douta => dout_i,
        doutb => data_i,
        ena => ena,
        enb => pull_up,
        wea => wrd,
        web => wrb);

sms_fft : sms
port map ( clk => clkd,
          xn_r => data_r,
          xn_i => data_i,
          start => start,
          fwd_inv => fwd_inv,
          ce => ce,
          rs => rs,
          mrd => mrd,
          mwr => mwr,
          scale_mode => scale_mode,
          dob_i => dob_i,
          dob_r => dob_r,
          edone => edone,
          done => done,
          result => result,
          ovflo => ovflo,
          busy => busy,
          ext_addr => ext_addr,
          ext_addrw => ext_addrw,
          io_n => io_n);

-- Instance of DW03_lfsr_load
misr1 : DW03_lfsr_load
generic map (width => 32)

port map ( data => data,
          load => pull_down,
          cen => lfsr_en,
          clk => clkd,
          reset => lfsr_rsn,
          count => lfsr_sign);

lfsr_rsn<= not rs;
ref_signature <= "11000100110100000001101011100000";
data(15 downto 0)<= dob_r;
data(31 downto 16)<= dob_i;

success<='1' when lfsr_sign=ref_signature else '0';

state_out <= state; -- for debugging purposes

pull_down <='0';
pull_up <= '1';

```

```

clkd<=clkdiv;

--add_gen: process (clkd,rst)
--
--begin
    if rst='1' then
        addr_flag<='0';
        self_addr<=(others=>'0');
    elsif clkd'EVENT and clkd='1' then
        if cntl_addr='1' then
            if addr_flag='0' then
                self_addr<=(others=>'0');
                addr_flag<='1';
            else
                self_addr<=self_addr+'1';
                if self_addr="111111111" then
                    addr_flag<='0';
                end if;
            end if;
        else
            addr_flag<='0';
            self_addr <= (others=>'0');
        end if;
    end if;
--end process;

fst: process(clkd,rst)

--variable count: Integer;
begin

if rst='1' then
    state<="0001";
    self_addr <= (others=>'0');
    mrd <= '0';
    mwr <= '0';
    start<='0';
    ce <= '1';
    rs <= '0';
    wrb <= '0';
    done_fft<='0';
    lfsr_en<='0';
    --count:=0;

elsif clkd'event and clkd='1' then

    case state is

        when "0000" =>
            mrd <= '0';
            mwr <= '0';
            self_addr <= (others=>'0');
            start<='0';
            ce <= '1';

```

```

        rs <= '0';
        wrb <= '0';
        done_fft<='0';
        lfsr_en<='0';
        state<= "0001";
        --count:=0;

when "0001" =>
    rs <= '1';
    state <= "0010";

when "0010" =>
    rs <= '0';
    if start_fft='1' then
        state <= "0011";
    end if;

when "0011" =>
    mwr <= '1';
    wrb<='0';
    self_addr <= (others=>'0');
    state <= "0100";

when "0100" =>
    mwr <= '0';
    self_addr<=self_addr+'1';
    if self_addr="1111111111" then
        state<="0101";
    end if;

when "0101" =>
    self_addr <= (others=>'0');

    state <= "0110";

when "0110" =>
    start <= '1';
    state<= "0111";

when "0111" =>
    start<='0';
    state<= "1000";

when "1000" =>
    if busy='0' then
        state <= "1001";
    end if;

when "1001" =>
    mrd <='1';
    state <= "1010";

when "1010" =>
    mrd <= '0';
    state<= "1011";

when "1011" =>
    wrb <= '1';
    lfsr_en<= '1';
    self_addr<=(others=>'0');
    state<="1100";

when "1100" =>
    self_addr<=self_addr+'1';
    if self_addr="1111111111" then
        state<="1101";
        lfsr_en<='0';

```

```

        wrb<='0';
    end if;

    when "1101" =>    self_addr<=(others=>'0');
                    wrb<='0';
                    lfsr_en<='0';
                    done_fft<='1';
                    state <= "1110";

    when "1110" =>    done_fft<='0';
                    state <= "0000";

    when others      =>    state <= "0000";

    end case;
end if;
end process;

end s_arch;

```

sms.vhd

```

-- Single Memory Space Configuration for fft.
-- uses Virtex Blockrams and 1024-point complex fft core.

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
use IEEE.STD_LOGIC_ARITH.all;

LIBRARY DWARE;
use DWARE.DWpackages.all;

ENTITY sms is
generic ( width : Natural := 16;
          depth : Natural := 1024);
port( clk : in std_logic;
      xn_r,xn_i : in std_logic_vector(width-1 downto 0);
      start : in std_logic;
      fwd_inv : in std_logic;
      ce : in std_logic;
      rs : in std_logic;
      mrd : in std_logic;
      mwr : in std_logic;
      scale_mode : in std_logic;

      dob_i,dob_r : out std_logic_vector(width-1 downto 0);
      edone : out std_logic;
      done : out std_logic;
      result : out std_logic;
      ovflo : out std_logic;
      busy : out std_logic;

```



```

        -- added for using the buffer rams for pilchard
functionality
    ext_addr : out std_logic_vector(bit_width(depth)-1 downto
0);
    ext_addrw : out std_logic_vector(bit_width(depth)-1 downto
0);
    io_n : out std_logic);
end sms;

architecture conf of sms is

----- Begin Cut here for COMPONENT Declaration -----
COMP_TAG
component blockram_1024x16
    port (
        addra: IN std_logic_VECTOR(9 downto 0);
        addrb: IN std_logic_VECTOR(9 downto 0);
        clka: IN std_logic;
        clkb: IN std_logic;
        dina: IN std_logic_VECTOR(15 downto 0);
        dinb: IN std_logic_VECTOR(15 downto 0);
        douta: OUT std_logic_VECTOR(15 downto 0);
        doutb: OUT std_logic_VECTOR(15 downto 0);
        ena: IN std_logic;
        enb: IN std_logic;
        wea: IN std_logic;
        web: IN std_logic);
end component;

-- FPGA Express Black Box declaration
--attribute fpga_dont_touch: string;
--attribute fpga_dont_touch of blockram_1024x16: component is
"true";

-- COMP_TAG_END ----- End COMPONENT Declaration -----

----- Begin Cut here for COMPONENT Declaration -----
COMP_TAG
component vfft1024
    port(
        clk          : in std_logic;
        rs           : in std_logic;
        start        : in std_logic;
        ce           : in std_logic;
        scale_mode    : in std_logic;
        di_r         : in std_logic_vector(15 downto 0);
        di_i         : in std_logic_vector(15 downto 0);
        fwd_inv       : in std_logic;
        io_mode0      : in std_logic;
        io_model      : in std_logic;
        mwr           : in std_logic;
        mrd           : in std_logic;
        ovflo         : out std_logic;

```

```

        result          : out std_logic;
        mode_ce         : out std_logic;
        done            : out std_logic;
        edone           : out std_logic;
        io              : out std_logic;
        eio             : out std_logic;
        bank            : out std_logic;
        busy            : out std_logic;
        wea             : out std_logic;
        wea_x           : out std_logic;
        wea_y           : out std_logic;
        web_x           : out std_logic;
        web_y           : out std_logic;
        ena_x           : out std_logic;
        ena_y           : out std_logic;
        index           : out std_logic_vector(9 downto 0);
        addr_r_x        : out std_logic_vector(9 downto 0);
        addr_r_y        : out std_logic_vector(9 downto 0);

        addrw_x         : out std_logic_vector(9 downto 0);

        addrw_y         : out std_logic_vector(9 downto 0);

        xk_r            : out std_logic_vector(15 downto 0);
        xk_i            : out std_logic_vector(15 downto 0);
        yk_r            : out std_logic_vector(15 downto 0);
        yk_i            : out std_logic_vector(15 downto 0));
end component;

-- XST black box declaration
attribute box_type : string;
attribute box_type of vfft1024: component is "black_box";

-- FPGA Express Black Box declaration
attribute fpga_dont_touch: string;
attribute fpga_dont_touch of blockram_1024x16: component is
"true";
attribute fpga_dont_touch of vfft1024: component is "true";

-- Synplicity black box declaration
attribute syn_black_box : boolean;
attribute syn_black_box of vfft1024: component is true;

-- COMP_TAG_END ----- End COMPONENT Declaration -----

signal di,dr : std_logic_vector(width-1 downto 0);
signal xk_r,xk_i : std_logic_vector(width-1 downto 0);
--signal yk_r,yk_i : std_logic_vector(width-1 downto 0);
--signal xn_r,xn_i : std_logic_vector(width-1 downto 0);
signal addr_r,addr_w : std_logic_vector(bit_width(depth)-1 downto
0);
--signal ce : std_logic;
--signal clk : std_logic;
signal wea,io : std_logic;
--signal mrd,mwr,fwd_inv,start,rs : std_logic;

```

```

--signal scale_mode,ovflo,result : std_logic;
--signal done,edone,bank,busy : std_logic;
signal mode_ce,eio : std_logic;
--signal wea_x,wea_y,web_x,web_y,ena_x,ena_y : std_logic;
--signal index : std_logic_vector(bit_width(depth)-1 downto 0);
signal pull_up : std_logic;
signal pull_down : std_logic;

begin

real: blockram_1024x16
port map(   addra => addrw,
           addrb => addrr,
           clka => clk,
           clkb => clk,
           dina => xk_r,
           dinb => xn_r,
--         douta =>
           doutb => dr,
           ena => ce,
           enb => pull_up,
           wea => wea,
           web => io);

imag: blockram_1024x16
port map(   addra => addrw,
           addrb => addrr,
           clka => clk,
           clkb => clk,
           dina => xk_i,
           dinb => xn_i,
--         douta =>
           doutb => di,
           ena => ce,
           enb => pull_up,
           wea => wea,
           web => io);

fft: vfft1024
port map (  clk => clk,
           rs  => rs,
           start => start,
           ce => ce,
           scale_mode => scale_mode,
           di_r => dr,
           di_i => di,
           fwd_inv => fwd_inv,
           io_mode0 => pull_down,
           io_model => pull_up,
           mwr => mwr,
           mrd => mrd,
           ovflo => ovflo,
           result => result,
           mode_ce => mode_ce,
           done => done,

```

```

        edone => edone,
        io => io,
        eio => eio,
--      bank => bank,
        busy => busy,
        wea => wea,
--      wea_x => wea_x,
--      wea_y => wea_y,
--      web_x => web_x,
--      web_y => web_y,
--      ena_x => ena_x,
--      ena_y => ena_y,
--      index => index,
        addr_x => addr,
        --addr_y =>
        addrw_x => addrw,
--      addrw_y =>
        xk_r => xk_r,
        xk_i => xk_i);
--      yk_r => yk_r,
--      yk_i => yk_i);

dob_r <= dr;
dob_i <= di;
ext_addr <= addr;
ext_addrw <= addrw;
io_n <= not io;
pull_down <='0';
pull_up<='1';

end conf;

```

Distributed ANN Training programs**parent.c**

```

#include <stdio.h>
#include "/sw/matlab6.1/extern/include/engine.h"
#include "/usr/local/pvm3/include/pvm3.h"
#include "/sw/matlab6.1/extern/include/matrix.h"
#include <sys/time.h>
#include <sys/resource.h>

#define BUFSIZE 25000
/*#define test*/

#define NTASK 5 /** starts NTASK child processes **/

/***** to calculate the execution time *****/
double cpusecs() {
    struct rusage ru;
    getrusage(RUSAGE_SELF, &ru);
    return(ru.ru_utime.tv_sec +
    ((double)ru.ru_utime.tv_usec)/1000000.0);
}/* cpusecs end*/

int main(){
    Engine *ep;
    double t,t1,t2,numhid,testdouble,goal;
    char
    filename1[25],filename2[25],teststr[100],teststr1[25000];
    int
    tid[NTASK],testids[NTASK],i,j,instnum,info,size,stop=0,cc,child_in
    st,child_tid=1;
    struct timeval tmout1,tmout2;

    /***** Enter the Data filename *****/

    puts("Enter the datafile name:");
    scanf("%s",&filename1);                /*** reading the
datafile name ***/
    puts("Enter the Architecture Specifications Filename:");
    scanf("%s",&filename2);                /*** reading the
archspec filename ***/

    /******* Start the CPU timer *****/
/*      t1=(double)cpusecs();*/
    info=gettimeofday(&tmout1,NULL);
/*      printf("tmout1 = %d\n", info);*/

    /******* Spawning the child processes *****/

    instnum = pvm_joyingroup("nnet");
    cc = pvm_spawn("/home/smerchan/577/project/child",
(char**)0, 1, "vlsil", 1, tid);

```

```

        if (cc == 0) { pvm_exit(); return -1; }
        cc = pvm_spawn("/home/smerchan/577/project/child",
(char**)0, 1, "vlsi2", 1, tid);
        if (cc == 0) { pvm_exit(); return -1; }
        cc = pvm_spawn("/home/smerchan/577/project/child",
(char**)0, 1, "vlsi3", 1, tid);
        if (cc == 0) { pvm_exit(); return -1; }
        cc = pvm_spawn("/home/smerchan/577/project/child",
(char**)0, 1, "vlsi4", 1, tid);
        if (cc == 0) { pvm_exit(); return -1; }
        cc = pvm_spawn("/home/smerchan/577/project/child",
(char**)0, 1, "vlsi5", 1, tid);
        if (cc == 0) { pvm_exit(); return -1; }
        /*cc = pvm_spawn("/home/smerchan/577/project/child",
(char**)0, 1, "vlsi6", 1, tid);
        if (cc == 0) { pvm_exit(); return -1; }
        cc = pvm_spawn("/home/smerchan/577/project/child",
(char**)0, 1, "vlsi7", 1, tid);
        if (cc == 0) { pvm_exit(); return -1; }
        cc = pvm_spawn("/home/smerchan/577/project/child",
(char**)0, 1, "vlsi8", 1, tid);
        if (cc == 0) { pvm_exit(); return -1; }*/

/***** checking for group membership *****/

for (i=0; i<NTASK; i++){

    info = pvm_recv(-1,1);
    if (info>0){
        pvm_upkstr(teststr);
        puts(teststr);
    }/* if end */
} /* for end */

/***** sending the data file name,
architecture spec file name and the TASK IDs to the child
processes *****/

pvm_initsend(PvmDataDefault);
pvm_pkstr(filename1);
pvm_pkstr(filename2);
/*
pvm_pkint(&tid[0],NTASK,1);
pvm_mcast(&tid[0],NTASK,1);*/
pvm_bcast("nnet",2);

#ifdef test
    puts("Test Mode");

/***** Receiving echo confirmation for
datafile and spec file names *****/
for (i=0; i<NTASK; i++){
    pvm_recv(-1,12);
    pvm_upkstr(teststr);

```

```

        puts(teststr);
        pvm_upkstr(teststr);
        puts(teststr);
        /*pvm_upkint(&testids[0],NTASK,1);
        for (j=0; j<NTASK; j++){
            printf("%d\t",testids[j]);
        */
        puts("\n");
    } /* for end */
#endif
#ifdef test

        /***** Receiving "Matlab Started"
Confirmation *****/

        for (i=0; i<NTASK; i++) {
            pvm_recv(-1,13);
            pvm_upkstr(teststr);
            puts(teststr);
        } /* for end */
#endif

#ifdef test

        /***** Receiving "addpath" confirmation
*****/

        for (i=0; i<NTASK; i++) {
            pvm_recv(-1,14);
            pvm_upkstr(teststr);
            pvm_upkint(&cc,1,1);
            puts(teststr);
            printf("value:  %d\n",cc);
        } /* for end */
#endif

#ifdef test

        /***** Receiving "datafile loading"
confirmation *****/

        for (i=0; i<NTASK; i++) {
            pvm_recv(-1,15);
            pvm_upkstr(teststr);
            puts(teststr);
            pvm_upkstr(teststr);
            puts(teststr);
        } /* for end */
#endif

#ifdef test

        /***** Receiving "specfile loading"
confirmation *****/

```

```

        for (i=0; i<NTASK; i++) {
            pvm_recv(-1,16);
            pvm_upkstr(teststr);
            puts(teststr);
            pvm_upkstr(teststr);
            puts(teststr);
        } /* for end */
    #endif

    #ifdef test

        /***** Receiving "Number of Hidden Nodes"
confirmation *****/

        for (i=0; i<NTASK; i++) {
            pvm_recv(-1,17);
            pvm_upkstr(teststr);
            puts(teststr);
            pvm_upkstr(teststr);
            puts(teststr);
            pvm_upkdouble(&testdouble,1,1);
            printf("ptrindex = %lf\n",testdouble);
        } /* for end */
    #endif

    #ifdef test

        /***** Receiving " Training Start"
confirmation *****/

        /*
            for (i=0; i<NTASK; i++) {
                pvm_recv(-1,18);
                pvm_upkstr(teststr);
                puts(teststr);
                pvm_upkstr(teststr);
                puts(teststr);*/
        /*
            }*/ /* for end */
        #endif

        #ifdef test

            /***** Receiving " Training Start"
confirmation *****/

            /*
                for (i=0; i<NTASK; i++) {
                    pvm_recv(-1,19);
                    pvm_upkstr(teststr);
                    puts(teststr);
                    pvm_upkstr(teststr);
                    puts(teststr);*/
            /*
                }*/ /* for end */
            #endif
            puts("Training the network. Pls wait .... ");

            do {

```



```

        pvm_recv(-1,-1);
        pvm_upkdouble(&goal,1,1);
        pvm_upkdouble(&numhid,1,1);
        pvm_upkint(&child_inst,1,1);
        if (goal==1) {
            printf("Training Successful for %d Hidden
Nodes\n", (int)numhid);
            puts("killing all other tasks with larger
networks ...");
            /*      info=gettimeofday(&tmout2,NULL);
                    t1=tmout1.tv_sec + (tmout1.tv_usec)/1000000.0;
                    t2=tmout2.tv_sec + (tmout2.tv_usec)/1000000.0;
                    t=t2-t1;
                    printf("Execution Time: %lf secs\n",t);*/

            while (child_tid > 0){
                child_tid =
pvm_gettid("nnet",child_inst+1);
                if (child_tid > 0)
{pvm_kill(child_tid);};
            }
            } else {
                printf("Training goal couldn't be met with %d
Hidden Nodes\n", (int)numhid);
                /*      info=gettimeofday(&tmout2,NULL);
                        t1=tmout1.tv_sec + (tmout1.tv_usec)/1000000.0;
                        t2=tmout2.tv_sec + (tmout2.tv_usec)/1000000.0;
                        t=t2-t1;
                        printf("Execution Time: %lf secs\n",t);*/

                }
                size=pvm_gsize("nnet");
                size=pvm_gsize("nnet");
                size=pvm_gsize("nnet");
                size=pvm_gsize("nnet");
            } while (size > 1);

        pvm_exit();

        /***** stop the timer *****/
        /*t2 = (double)cpusecs();
        t=t1-t2;*/
        info=gettimeofday(&tmout2,NULL);
        /*      printf("tmout2 = %d\n", info);*/
        t1=tmout1.tv_sec + (tmout1.tv_usec)/1000000.0;
        t2=tmout2.tv_sec + (tmout2.tv_usec)/1000000.0;
        t=t2-t1;
        printf("Execution Time: %lf secs\n",t);
        return 1;
    }
}

```

child.c

```
#include <stdio.h>
```

```

#include "/sw/matlab6.1/extern/include/engine.h"
#include "/usr/local/pvm3/include/pvm3.h"
#include "/sw/matlab6.1/extern/include/matrix.h"
#include <sys/time.h>
#include <sys/resource.h>

#define BUFSIZE 25000
/*#define test*/

double cpusecs() {
    struct rusage ru;
    getrusage(RUSAGE_SELF, &ru);
    return(ru.ru_utime.tv_sec +
    ((double)ru.ru_utime.tv_usec)/1000000.0);
}/* cpusecs end*/

int main(){

    Engine *ep;
    double t1, t2, *res, *numnod, *ptrindex;
    int ptid, mytid, *tid, instnum, count=0, d;
    char
testbuf[100], filename1[25], filename2[25], buffer[BUFSIZE];
    mxArray *datafile=NULL,
    *specfile=NULL, *ptrhid=NULL, *goal=NULL, *numhid=NULL;

    /***** start the timer *****/
    t1 = (double)cpusecs();

    /***** getting the task IDs *****/
    instnum = pvm_joyingroup("nnet");
    ptid = pvm_parent();
    mytid= pvm_mytid();

    /***** sending group joined confirmation
*****/

    pvm_initsend(PvmDataDefault);
    pvm_pkstr("joined group nnet");
    pvm_send(ptid, 1);

    /***** receiving the tids, datafilename and the
spec filename *****/

    pvm_rcv(ptid, 2);
    pvm_upkstr(filename1);
    pvm_upkstr(filename2);
/*    pvm_upkint(tid, 1, 1);*/

    #ifdef test

        /***** echoing back *****/

        pvm_initsend(PvmDataDefault);

```

```

        pvm_pkstr(filename1);
        pvm_pkstr(filename2);
        pvm_send(ptid,12);
    #endif

    /***** starting matlab engine *****/

    ep=engOpen("\0");
    if (!(ep)) {
        fprintf(stderr, "\nCan't start MATLAB engine\n");
        #ifdef test
            /***** echoing the error *****/
            pvm_initsend(PvmDataDefault);
            pvm_pkstr("Matlab not started");
            pvm_send(ptid,13);
        #endif
        pvm_exit();
        return EXIT_FAILURE;
    } /* end if */

    #ifdef test
        /***** echoing back *****/

        pvm_initsend(PvmDataDefault);
        pvm_pkstr("Matlab started");
        pvm_send(ptid,13);
    #endif
    engOutputBuffer(ep,buffer,BUFSIZE);

    d=engEvalString(ep,"addpath('/home/smerchan/577/project');");

    #ifdef test
        /***** echoing back *****/

        pvm_initsend(PvmDataDefault);
        pvm_pkstr("addpath done");
        pvm_pkint(&d,1,1);
        pvm_send(ptid,14);
    #endif

    /***** loading the datafile *****/
    datafile=mxCreateString(filename1);
    mxSetName(datafile,"datafile");
    engPutArray(ep,datafile);          /*** putting it
in Matlab workspace ***/

    #ifdef test
        /***** echoing back *****/

        pvm_initsend(PvmDataDefault);
        pvm_pkstr("datafile loaded");
        pvm_pkstr(buffer);
        pvm_send(ptid,15);
    #endif

```

```

#endif

/***** loading the spec file *****/
specfile=mxCreateString(filename2);
mxSetName(specfile,"specfile");
engPutArray(ep,specfile);          /*** putting it
in Matlab workspace ***/

#ifdef test
    /*** echoing back *****/

    pvm_initsend(PvmDataDefault);
    pvm_pkstr("specfile loaded");
    pvm_pkstr(buffer);
    pvm_send(ptid,16);
#endif

/**** Choosing the Number of Hidden Neurons *****/

ptrhid = mxCreateDoubleMatrix(1, 1, mxREAL);
mxSetName(ptrhid,"ptrhid");
ptrindex=(double *)malloc(1*sizeof(double));
*ptrindex= (double)instnum;
mxSetPr(ptrhid,ptrindex);
d=engPutArray(ep,ptrhid);
d=engEvalString(ep,"ptrhid");

#ifdef test
    /*** echoing back *****/

    pvm_initsend(PvmDataDefault);
    pvm_pkstr("Number of Hidden Nodes: ");
    pvm_pkstr(buffer);
    pvm_pkdoube(ptrindex,1,1);
    pvm_send(ptid,17);
#endif

/***** Start the training *****/

while (count < 2){

    d=engEvalString(ep,"nntwarn off;");
    d=engEvalString(ep,"close all;");
    d=engEvalString(ep,"[goal SSE
numhid]=newtrain(datafile,specfile,ptrhid);");
    #ifdef test
        /*** echoing back *****/

/*
        pvm_initsend(PvmDataDefault);
        pvm_pkstr("Starting Training");
        pvm_pkstr(buffer);
        pvm_send(ptid,18+count);*/
    #endif

```

```

d=engEvalString(ep,"goal");

goal=engGetArray(ep,"goal");
res = mxGetPr(goal);
numhid=engGetArray(ep,"numhid");
numnod = mxGetPr(numhid);

pvm_initsend(PvmDataDefault);
pvm_pkdouble(res,1,1);
pvm_pkdouble(numnod,1,1);
pvm_pkint(&instnum,1,1);
pvm_send(ptid,3);
if (*res==1) break;

count=count+1;
}
pvm_lvgroup("nnet");
mxDestroyArray(datafile);
mxDestroyArray(specfile);
mxDestroyArray(ptrhid);
mxDestroyArray(numhid);
mxDestroyArray(goal);
engClose(ep);
pvm_exit();
}

```

MATLAB files

archspeg.m

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Script for defining the Architecutral Specifications %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% for Activation Functions %
% t => tansig %
% l => logsig %
% p => purelin %
% for Input Scaling %
% m => MCVU %
% l => Linear %
% n => None %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Defining Network Architecture Specification Matrix %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

numlay = 1; % number of hidden layers
numnod = [5:1:16]; % number of hidden neurons/hidden layer
vector

```

```

Fh='tansig';    % Hidden Layer Activation Function
Fo='purelin';   % o/p Layer Activation Function
sc='m';         % Input Scaling
sc_out='m';     % output scaling

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Specifying the Training Parameters                                     %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

tolerance = 100;    % error goal
epoch_disp = 1000;plotflag = 1; % Display rate
maxepochs = 2000;  % max epochs to train until

TP=[epoch_disp maxepochs tolerance .001 .01 10 .1 1e10]; %
Training parameter vector

save spec numlay numnod Fh Fo sc sc_out TP tolerance maxepochs;
% saving specification

```

datasetup.m

```

clear all;

load data; % loading the data set

% data set reduction
ind=find(x(:,4)<(.99*y)&y>1);
xt=x(ind,:);
yt=y(ind);
s=size(xt,1)
reduction=100*(length(x)-s)/length(x)

% output scaling
ytl=log(yt);

% training and testing data sets
breakpoint=620;
xtrn=xt(1:breakpoint,:);
xtest=xt(breakpoint+1:s,:);
ytrn=ytl(1:breakpoint);
ytest=ytl(breakpoint+1:s);

% saving the training and testing data
save trdata xtrn ytrn ytest xtest;

```

[illegible]

```

        y=log(y);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%       Mean center and unit variance scale input.
%
%       Transpose input/output vectors to conform to MATLAB standard.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%

if sc == 'm'
    [xn, xm, xs] = zscore1(x);
elseif sc == 'l'
    [xn, xm, xs] = scale1(x);
else
    xn=x;    % Input vector
end
xn=xn';    % Target vector
yn=y';    % Target vector

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Initializing the weights and biases and training the network %
% using levenberg-Marquardt Algorithm                          %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if numlay==1
    F1=Fh;
    F2=Fo;
    [W1,B1,W2,B2]=initff(xn,numhid,F1,numout,F2);    %Initializing
weights and biases
    figure;
    [W1,B1,W2,B2,epochs,TR]=trainlm(W1,B1,F1,W2,B2,F2,xn,yn,TP);
%Training the network
elseif numlay==2
    F1=Fh;
    F2=Fh;
    F3=Fo;
    [W1,B1,W2,B2,W3,B3]=initff(xn,numhid,F1,numhid,F2,numout,F3);
%Initializing weights and biases
    figure;

    [W1,B1,W2,B2,W3,B3,epochs,TR]=trainlm(W1,B1,F1,W2,B2,F2,W3,B3,F3,
xn,yn,TP); %Training the network
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%       Tolerance criteria not met by LM.                      %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

SSE=min(TR);
if SSE>tolerance
    goal=0;

```


Vita

Mr. Saumil Merchant was born (1976) and brought up in Mumbai, India. He did his schooling from New Era High School, Mumbai and joined Jai Hind College of Science, Mumbai in 1992. From there he went on to pursue a professional career in engineering at University of Mumbai and graduated with a Bachelors in Electronics Engineering in 1999. He joined University of Tennessee, Knoxville in January 2001 to pursue Masters of Science in Electrical Engineering. He has worked as a Windows Systems Administrator at Office of Research and Information Technology, Client and Network Services at University of Tennessee since February 2001 till present. He is a student member of IEEE. He plans to graduate with a Masters degree in Electrical Engineering in August 2003 and wishes to pursue a doctorate in Electrical Engineering at University of Tennessee.